ABSTRACT

Title of dissertation: RAPID RESOURCE TRANSFER FOR MULTILINGUAL NATURAL LANGUAGE PROCESSING

Okan Kolak, Doctor of Philosophy, 2005

Dissertation directed by: Professor Philip Resnik
Assoc. Prof., Dept. of Linguistics and
Institute for Advanced Computer Studies;
Affiliate Assoc. Prof., Dept. of Computer Sci.

Until recently the focus of the Natural Language Processing (NLP) community has been on a handful of mostly European languages. However, the rapid changes taking place in the economic and political climate of the world precipitate a similar change to the relative importance given to various languages. The importance of rapidly acquiring NLP resources and computational capabilities in new languages is widely accepted. Statistical NLP models have a distinct advantage over rule-based methods in achieving this goal since they require far less manual labor. However, statistical methods require two fundamental resources for training: (1) online corpora (2) manual annotations. Creating these two resources can be as difficult as porting rule-based methods.

This thesis demonstrates the feasibility of acquiring both corpora and anno-

tations by exploiting existing resources for well-studied languages. Basic resources for new languages can be acquired in a rapid and cost-effective manner by utilizing existing resources cross-lingually.

Currently, the most viable method of obtaining online corpora is converting existing printed text into electronic form using Optical Character Recognition (OCR). Unfortunately, a language that lacks online corpora most likely lacks OCR as well. We tackle this problem by taking an existing OCR system that was desgined for a specific language and using that OCR system for a language with a similar script. We present a generative OCR model that allows us to post-process output from a non-native OCR system to achieve accuracy close to, or better than, a native one. Furthermore, we show that the performance of a native or trained OCR system can be improved by the same method.

Next, we demonstrate cross-utilization of annotations on treebanks. We present an algorithm that projects dependency trees across parallel corpora. We also show that a reasonable quality treebank can be generated by combining projection with a small amount of language-specific post-processing. The projected treebank allows us to train a parser that performs comparably to a parser trained on manually generated data.

RAPID RESOURCE TRANSFER FOR MULTILINGUAL NATURAL LANGUAGE PROCESSING

by

Okan Kolak

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2005

Advisory Committee:

Professor Philip Resnik, Chair/Advisor
Doctor David S. Doermann
Professor Lise Getoor
Professor Rebecca Green
Professor Douglas W. Oard

Anneme; sonsuz, sınırsız, ve kayıtsız sevgisi için. Babama; sevgisi, desteği, ve anlayışı için. Ve kardeşime; suç ortagım, bir tanem.

To my mother; for her endless, boundless, and unconditional love. To my father; for his love, support, and understanding. And to my brother; my partner in crime, my dearest.

ACKNOWLEDGMENTS

First, I would like to thank my advisor, Dr. Philip Resnik. This dissertation would not be possible without his support and guidance. He was kind enough to let me choose where to go, and was always there to help me find my way. It was a pleasure and privilege to work with him.

I would like to thank to my committee members Dr. David Doermann, Dr. Lise Getoor, Dr. Rebecca Green, Dr. Douglas Oard for taking the time to help me complete my journey and for their useful feedback. Special thanks to Dr. Douglas Oard and Dr. David Doermann for their incredibly useful discussions throughout my research, for going beyond what was required of them, and for helping me grow. Many thanks to Dr. Tapas Kanungo and Dr. William Byrne for useful discussions. I also would like to thank Dr. Amy Weinberg and Dr. Bonnie Dorr for their help and kindness.

My life would be unbearable without my circle of friends, who kept me going with their love. Many thanks to Betül Atalay, Çağdaş Dirik, Nihal Bayraktar, Burcu Karagöl-Ayan, Fazıl Ayan, Füsun Yaman, Evren Şirin, Funda Ertunç, Mirat Satoğlu, Cemal Yılmaz, Esin Yılmaz, Ulaş Kozat, Akın Aktürk, Aydın Balkan, Fuat Keçeli, and Cihan Taş for making my life easier, joyful, and happy. And a big thank you from the bottom of my heart to my dearest friend Mustafa Murat Tıkır. I cannot imagine how my graduate life would be without his genuine friendship and

unconditional support. He was always there to help me through the ups and downs of life, and he never let me walk alone.

Many thanks to my friends at CLIP, who shared not only their office, but a piece of their life with me: Mona Diab, Clara Cabezas, Rebecca Hwa, Nizar Habash, Adam Lopez, Dina Demner-Fushman, Grazia Russo-Lassner, Nitin Madnani, Laura Bright, Kareem Darwish, and David Zajic.

Finaly, and most importantly, I would like to thank my family; this dissertation is dedicated to them. My deepest gratitude goes to my mother Emel and my father Hüseyin. I could not have done anything without their love, trust, and support throughout my life. They are a true blessing from God. I love them with all my heart. And to my brother Ozan, my first and foremost friend. I love him dearly.

This research has been supported in part by Department of Defense contract RD-02-5700, DARPA/ITO Cooperative Agreement N660010028910, ONR MURI Contract FCPO.810548265, DARPA/ITO Contract N66001-97-C-8540, National Science Foundation grant EIA0130422, Mitre agreement 010418-7712, and Department of Defense contract MDA90496C1250.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Problem Description and Proposed Solution

While the state-of-the-art in Natural Language Processing is advancing steadily, the progress is not uniform across languages. As a matter of fact, many of the world's languages still lack even the most basic computational resources such as electronic corpora. Until recently the focus of the NLP community, both academic and commercial, has been on a handful of mostly European languages. However, the rapid changes taking place in the economic and political climate of the world precipitate a similar change to the relative importance given to various languages, sometimes overnight. The importance of rapidly acquiring NLP resources and computational capabilities in new languages is widely accepted [100].

Unfortunately, it takes a considerable amount of time, effort, and money to retarget existing technologies to new languages. Statistical NLP models have a distinct advantage over rule-based approaches to achieve this goal since they require far less manual labor to port to new languages. However, statistical NLP methods require two fundamental resources for training: (1) online corpora (2) manual annotations. Creating these resources, annotations in particular, can make porting statistical methods as difficult as their rule-based counterparts.

One way to address this problem is to exploit existing resources for well-studied languages. By utilizing these resources cross-lingually, basic resources for new languages can be put in place in a rapid and cost-effective manner. The quality of the resulting resources are generally not ideal. However, they are not intended to replace full-fledged language-specific development efforts, but rather to allow early entry into new languages and provide basic functionality until better resources are developed.

The very first resource one needs for a new language is electronic corpora, be it for training purposes or the actual data to be processed. For many languages, however, even this basic resource is not available. Consequently, it is a natural starting point to create resources for a less-studied language. Assuming the language in question is a written language, the most cost effective and rapid method of obtaining electronic text is converting existing printed text into electronic form using Optical Character Recognition. Furthermore, the use of OCR for acquiring resources is not limited to corpora. For example, converting printed language resources such as dictionaries into a machine-readable form is an actively researched approach [28, 80].

However, it is very unlikely to have an OCR system for a language that does not even have electronic corpora. We solve this problem by cross-lingual utilization of an existing, well-developed OCR system designed for a language with a similar script. To achieve this, we developed a generative probabilistic OCR model that allows post-processing of OCR text to reduce its error rate. The model is effective enough to enable cross-language utilization. We show that by using a small amount

of training data, it is possible to achieve OCR performance close to, or even better than, the performance of a native OCR system. We further demonstrate that the same method can be used to improve the accuracy of a trained OCR system for the new language, as well as the performance of a native OCR system on its intended target language.

In addition to evaluating the OCR performance using intrinsic recognition quality metrics, we perform extrinsic evaluations; evaluating the impact of OCR errors and their correction on NLP application performance. While OCR errors reduce application performance considerably, error correction brings the performance much closer to the performance obtained using clean text.

Once the electronic corpora problem is addressed, the next step is to acquire annotated data. Such data usually does not occur naturally and is labor intensive to generate. Luckily, a considerable amount of annotated data is already generated for well-studied languages, and cross-language utilization of the existing data can provide the initial boost to allow basic processing for new languages.

Parsing is a very good platform for evaluating the effectiveness of using projected training data. First, statistical parsers achieve state-of-the-art performance and are much easier to port to new languages provided that a treebank is available to train on. Second, treebanks are not a resource that occur naturally and creating a treebank for a new language is slow and costly, making the cross-lingual utilization idea very attractive. We present a projection algorithm that takes a parallel corpus and dependency trees for one side of the corpus, and infers the dependency trees for the other side. The inferred treebank can then be employed to train a

statistical parser, making it possible to parse new sentences that do not have parsed parallel sentences. We show that a reasonable quality treebank can be generated by combining direct projection with minimal language-specific post-processing. The performance achieved by a parser trained on the inferred treebank is comparable to a parser trained on a manually generated treebank that took a couple of years to put in place.

1.2 Research Contributions

Contributions of the research presented in this dissertation are:

- A novel, end-to-end, probabilistic, generative model to describe the OCR process. It explicitly models conversion of the words to characters, segmentation of the character sequence, and character-level recognition.

- Finite State Machine (FSM)-based implementation of the proposed model, including parameter estimation methods and decoding.[1]

- Application of the proposed method to various languages to demonstrate the feasibility of performing post-processing to improve the quality of OCR output, and positive effects of OCR correction on NLP application performance.

- An edit distance-based method for mapping the words in clean text and its noisy version when word merge/split errors exist. The method combines

[1]Some estimation methods make use of existing software, such as language modeling tools, simply encoding their output as FSM, while others methods are implemented from scratch.

character-based and word-based methods to achieve better accuracy and finer granularity than either method alone.

- A new precision/recall-based metric for evaluating word-level alignment quality for parallel text with word merge/split errors introduced by noisy text sources. Existing metrics are not directly applicable when there is no one-to-one correspondence between the words in the gold-standard and the test data. However, the proposed metric can handle many-to-many mappings.

- A novel algorithm for projecting dependency parse trees over word alignments from one side of a parallel corpus to the other. We demonstrate that with a small amount of linguistic post-processing, the quality of the projected trees are good enough to train a statistical parser. Evaluation of the trees produced by the algorithm and linguistic post-processing work were carried out by our colleagues, as reported in [57, 58, 60].

- Rigorous evaluation of the cross-lingual utilization idea to acquire two basic resources, corpora and annotations, required to port NLP applications to new languages.

1.3 Overview

The rest of the dissertation is organized as follows: In Chapter 2 we will discuss relevant scientific literature. Chapter 3 focuses on OCR post-processing as an effective method of electronic corpora acquisition. Chapter 4 adapts the post-processing

model presented in Chapter 3 for low-density languages that do not have an available lexicon. Chapter 5 extends the cross-lingual utilization idea to annotations, and presents an algorithm for projecting parse trees across parallel text. Finally, Chapter 6 presents our conclusions, and potential future directions.

Chapter 2

Related Work

2.1 Optical Character Recognition

Designing machines that can read is a field of research that has attracted interest and attention for a long time. The first patent for character recognition was obtained in 1929 in Germany [reported in 95]. Numerous researchers have worked on the problem since then, even more so with the arrival of electronic computers that turned OCR into reality. The brief review of OCR presented in this section is mainly based on papers by Mori et al. [95], who provide an excellent and detailed review of OCR in general; Eikvil [31], who presents a high-level, easy to follow review of the field; and Trier et al. [138], who focus on the aspect of feature extraction, providing a rather detailed and technical coverage of the subject. References to other relevant papers are given in the text, as necessary.

2.1.1 Character Recognition Process

While reading is quite natural process for humans, it involves a series of complex operations. A piece of printed text goes through the following steps while it is being processed by a typical modern OCR system:[1]

[1]The steps provided here are not universal. OCR systems may deviate from the process outlined here.

1. Color or gray-scale scanning of the page image to obtain a digital version
2. Preprocessing of the image prior to the character recognition process
    (a) Binarization of the image into a binary valued bitmap
    (b) Zoning to identify various regions such as text blocks, graphics, and page structures such as columns
    (c) Segmentation to isolate individual characters and words
    (d) Possible conversion into a different format, such as skeleton, contour, blurred characters, and so on.
3. Feature extraction
4. Classification of character images into character classes
5. Post-processing to improve recognition performance using contextual information

While our research focuses on Item 5, historically the main focus points of OCR research have been items 3 and 4. Post-processing is covered in detail in Section 2.2.2. We will briefly cover items 3 and 4 in this section.

Feature Extraction

This is an important step in achieving quality OCR performance. However, OCR can be performed without this step. Early OCR systems used a method called template matching, which skips this step altogether. In template matching, the image of the

character to be recognized is compared against a set of character templates, and the class represented by the most similar template is selected as the class of the input character. Various methods for representing the characters and measuring the similarity have been used, such as slit projections, peepholes, moments, and so on. While each method has various advantages and disadvantages, the template matching method overall is inherently restricted by the limited number of segments used for recognition, and cannot handle much variation in the appearance of the characters.

In order to overcome the limitations of the template matching method, feature-based recognition systems have been developed. This approach identifies distinguishing features of the characters, and defines their appearance in a more abstract manner. Once these features are identified, matching can be made at the feature level, allowing more variation than the exact shape match of the template-based methods. One important concept used in the feature extraction approach is the concept of invariants. An invariant is a feature that does not change as the character is transformed in various ways, such as rotation, scaling, skew, stretch, etc. Some auxiliary processing, such as resizing, thinning, gap filling, etc., is used to normalize and improve the shape of characters.

The types and number of features used is an open question. Some features are based on the distribution of points within the bounding box of the character. One example is zoning, where the bounding box is divided into cells, and the intensity in each cell is used as a feature. Another approach is to use moments[2] of various

[2]In the statistical sense of the term

degrees of the black points around a point of origin. The number of times the lines of the character cross each other, or some other lines that are superimposed on the character has also been used as a feature, among numerous others.

Transformations of the characters and various series expansions have also been used to represent the shape of the characters. This approach transforms characters and represents them in a more abstract way, preserving the overall distinguishing features while abstracting away the details that make matching difficult. Some well known examples are Fourier and KL transformations, which are widely used in applications of signal processing. Other examples are contour profiles and spline curve approximations.

Another approach is to perform a structural analysis of the character, and describe its shape using its sub-components, such as arcs, vertical and horizontal lines, gaps, loops, etc. The strength of this approach is its tolerance to style variations among fonts, and in hand-writing, as well as high levels of noise. No matter which font is used, or who wrote it, an 'H' is basically made of two parallel vertical bars joined by a horizontal bar somewhere near their center. The problem with this approach is the difficulty of extracting the structural features. Moreover, while the overall structure does not change, the features themselves are not very tolerant to noise and distortions. For example, the vertical bars of the 'H' are not always vertical, neither are they always straight bars.

Note that the features covered here are just a sample, and in no way complete. There are a vast number of features, that can define the shape of a character in a variety of ways. While the selection of features to be used has a great impact on the

recognition performance, the selection itself is highly dependent on the particular recognition problem.

Character Classification

Regardless of the features that are used to describe the characters, and the way they are extracted, the next step is to compare and match the features of the characters to be recognized with the features of the character classes. This step is where the actual character recognition takes place. In essence, it is a classification problem, where each character that appears on papers is assigned a character class. For instance, no matter in which shape or form it appears on paper, an image of the character '`A`' should be classified as such.

At this stage, any feature-based classifier can be used for assigning the character classes. One common approach is to use a statistical classifier, where for each character, the class with the highest probability is selected. More formally, it is the usual Bayesian classification where the class that maximizes the probability $P(features|class)P(class)$ is selected. This approach requires the ability to compute both the a priori probability of the class, which is usually derived from a language model, and the probability of that class generating the character with certain features. This second component is usually learned from a training corpus where various characters and their correct classification are available.

Another similar approach is based on using some measure of similarity, dissimilarity, or distance to compare the feature vector of the character to be recognized

with the feature vectors representing the character classes. More specifically, assuming the measure being used is a distance metric, say Euclidean distance, the distance between the feature vector of the character and the feature vectors of each class is computed, and the character is assigned the class with the smallest distance.

Neural networks are another classification method that is used for some OCR applications. Neural networks are designed to mimic the operation of the brain. This method can be quite effective in learning the associations and patterns hidden in data. Another advantage of neural networks is their tolerance of degradation and adaptability. They can even be used to process the character images directly, hence performing implicit feature extraction along with classification. One major problem with the neural network approach is the difficulty of understanding and analyzing the decision process that takes place.

Character classification methods based on structural analysis, on the other hand, can employ a formal description, such as a finite state network or a grammar, to describe how a certain character class generates the observed image from its components. Once the features of the character to be recognized are extracted, the character is assigned the class whose grammar can generate it.

2.1.2 HMM-based Recognition

A different approach to character recognition worth mentioning is the use of Hidden Markov Models (HMMs). An HMM recognizer acts as the classification component of the overall OCR process as given in Section 2.1.1. With this approach the feature

extraction step would need to be designed accordingly. Given the success of HMMs in the speech domain, many researchers investigated the possibility of transferring this mature speech technology to the OCR domain. The main problem in this technology transfer is the fact that speech is a one-dimensional signal whereas document images are two dimensional. While some researchers reduced the image recognition problem to a single dimension using techniques such as line extraction followed by the recognition of single lines, some others chose to extend the HMM approach to create two-dimensional analogs [70].

A good example is the system presented by Lu et al. [79]. While most of the OCR systems make use of language and/or character set specific features and constraints, the authors propose a language-independent OCR systems that is based on speech recognition techniques. As a matter of fact, the OCR system is implemented using an existing speech recognition system without modification, except for the preprocessing and feature extraction stages. Major strengths of the approach is its language-independent training and recognition, ability to train on unsegmented input, and recognition without prior segmentation of input. It works on the line level text images, and uses a vertical strip to scan the line horizontally, which mimics the time frames used in speech recognition. Various features of each frame are computed based on image characteristics such as intensity. Experimental evaluations on English, Arabic, and Chinese are presented using various levels of image quality. The evaluations demonstrate the applicability of the proposed approach to different scripts and to various levels of image degradation. The character error rate ranges from 0.8% to 10% under various conditions.

2.1.3 Performance Evaluation

Evaluating the cause of errors is as important as quantifying them in terms of analyzing and improving OCR systems. Typical OCR errors originate at different steps of the recognition process for various reasons. The major common reason being low quality documents.

Image quality problems arise when the contrast between the text and the background is low or varies within a page, a low resolution image is used, or the document is a multi-generation copy. This usually results in poor binarization of the image, which can result in smeared or broken characters in the best case, and completely washed out regions in the worse. The zoning process can also introduce errors due to poor image, complex or unusual page structure, or heavy use of graphics. As a result, some text regions might be left out of recognition, while some non-text regions are treated as text. Another common result is the ordering of text regions in the output in a way different from the intended reading order. Errors made in the segmentation process cause images to be broken into individual characters at incorrect positions which usually lead to merged or split characters and words. Such errors can be caused by poor image or print quality, where characters appear to be touching, broken, or too light. Another common error type is classification errors, where a character is assigned a wrong character class. Reasons for this type of error include: inherent similarities between some characters; poor selection of features, such as features that do not have enough distinguishing power; a mismatch between training and test data; and so on.

The evaluation of the OCR system performance has been traditionally at the character level, since it is primarily a character recognition problem. There are three rates that are used to quantify the recognition performance of a system. The first is the recognition rate, which is the percentage of the characters in the input image that are correctly classified. The second is the error rate, which is the percentage of characters that are assigned a wrong class. OCR systems may reject a character when they are unable to assign a class reliably, instead of assigning a wrong class. The percentage of characters that are not assigned any class is represented by the reject rate.

These metrics are centered around the classification component, where the OCR system is given isolated characters one at a time, and it either returns a class, or rejects them. In the actual recognition task, however, the system might not even consider some of the characters for recognition, or introduce extra text which does not appear on the page, due to zoning errors. For instance, if an OCR system misses all the text on a page, and hence does not process any characters, its recognition, error, and reject rates would all be zero for that page.

While evaluating the final output generated by an OCR system, using the ground-truth text as the reference point, word error rate and character error rate may be better metrics. The word error rate is a widely used metric in speech recognition, where the edit distance between the original input words and the words generated by the recognition system is computed and normalized by the length of the original input. Character error rate is essentially the same metric computed at character-level.

Note, however, that these metrics are not very useful without some knowledge about the requirements of the particular application. For instance, a higher reject rate is preferable to a higher error rate for barcode reading.[3] When a barcode is rejected, the user can either retry, or manually enter the code, whereas a misrecognized barcode might go unnoticed. As another example, the actual values of any of the metrics is not relevant for letter sorting, as long as recognized output is good enough to route the letter to the correct destination.

2.1.4 Use of Optical Character Recognition

A vast amount of document collections are available in print, which require efficient retrieval methods for easy access. The usual approach to the problem is to convert the collection to electronic form using OCR, and then use an Information Retrieval (IR) system for retrieval. However, the OCR'd collection contains a considerable amount of noise, which can affect IR performance.

Taghva et al. [133] provide an extensive evaluation of retrieval effectiveness on OCR data. Experiments on real and simulated OCR collections show that the retrieval performance is not affected significantly by OCR errors, except for very short documents. Although the documents are still being retrieved, however, the ranking and feedback methods are unable to handle the noise and their performance degrades. Another effect of the OCR collections is a considerable increase in the index size, which results from increased vocabulary size.

[3]Barcodes can be viewed as an extreme example of fonts that are designed for automated recognition. Reading barcodes is technically still OCR.

On the other hand, Lopresti [77] shows that moderate levels of OCR error have a significant impact on Boolean query models, and proposes an approximate string matching algorithm and uses fuzzy logic to address the problem. The string matching algorithm is a modified version of the edit distance algorithm. It finds a segment in the document that has the smallest distance to the query term and computes a membership function in the range [0,1] depending on that distance. The membership values of query terms are combined using fuzzy logic. Evaluations on simulated OCR data show that at a 12% error rate, a traditional boolean model has a recall lower than 50%, while the proposed model has a recall of 95%. When noise level reaches 36%, the recall of the traditional model is below 10%, while the recall of the new model is still over 50%. However, the precision of the new model is very low, around 30%.

Collins-Thompson et al. [21] propose a more sophisticated string matching algorithm that uses the confusion probabilities of the recognition system. The system first performs a fast approximate match using a distance threshold. Once a set of matching words is generated, each match is scored using the noisy channel model. Words with a score over the final match threshold are considered a match. An evaluation of the system demonstrates an improvement of up to 38.61% in F-measure over the exact match model. Improvement drops to 23.36% when the error model is not used. It is worth noting that these results are obtained on a test collection with 90% Word Error Rate (WER). Improvements are much lower, even negative, for a collection with 10% WER.

Beitzel et al. [7] present a survey of methods that are used to improve IR

performance on OCR collections. One method is to use error correction methods to reduce the noise level, which is discussed in detail in Section 2.2.2. Another method is using retrieval models that account for OCR errors, such as n-gram-based retrieval models [49] and approximate string matching techniques [46, 97].

Translating OCR output is also an important application. FALCon is a document translation system that was developed at US Army Research Laboratory [141]. It is a portable, embedded system that combines scanning, OCR, Machine Translation (MT), and display functions. The system is designed to be used on the field by army units to asses the significance of the documents that they encounter. The recognition and translation problems are not addressed directly by the system. The system is designed only to allow filtering of relevant documents to be read and translated properly by human translators. The operation mode of the system does not require high quality output, as long as the number of content words that can be translated are enough to allow the user to perform the filtering task. Therefore, FALCon is implemented using of-the-shelf components combined in a pipeline, where there is no interaction between the components. Preliminary evaluations show that each step of the process introduces some noise. For instance, for the Spanish system, the OCR process reduced the number of words that can be recognized by the translation module by more than 60%. Overall, 17% of the words that are generated by translation were semantically related or domain relevant. When clean text is translated, 22% of the words are domain relevant [141]. While the authors view this as a low degradation ratio, it should be noted that the actual number of relevant words produced by translation were 5 and 17 for OCR

and clean text, which can be viewed as 70.58% degradation in output quality.

Miller et al. [93] evaluate the impact of noisy text sources such as OCR and Automatic Speech Recognition (ASR) on named entity extraction performance, which can be defined as identifying and labeling words in text that refer to people, organizations, locations, dates, etc. For OCR evaluations, they trained an HMM-based named entity extractor on clean text, and evaluated its performance on OCR text with varying levels of WER. The performance of the extractor is evaluated using standard F-measure metric. When the input is clean text, the extractor achieves an F-measure of 95.3%, which drops to 82.5% for OCR output with a WER of 19.1%. Interpolation of performance points for several experiments using different WER values shows that a 1% increase in WER translates to 0.6% drop in F-measure. They also evaluated the impact of the training data size on performance. Although more training data results in a logarithmic increase in performance regardless of the WER of the test data, the performance gap between clean and noisy text remains the same.

2.2 Error Correction in Text

There are many aspects of error reduction, from creating better user interfaces to training users. Detection of an error is also an important, non-trivial problem. However, we will mainly focus on error correction methods. Kukich [72] provides a very nice, comprehensive survey of the error correction techniques for text. The paper categorizes text error correction into three classes, in increasing order of difficulty.

The simplest problem is the detection of non-word errors. The second problem is correction of word errors in isolation, which is inherently limited to correction of non-word errors. The most difficult problem is context dependent error correction, which has the potential to correct both non-word and valid-word errors. Existing research in all three categories are reviewed and related research issues are discussed. We will focus on a subset of the correction methods, which is more relevant to our research, and organize our review based on the the source of the text that is being corrected.

2.2.1 Spelling Correction

Spelling errors and typos were around long before the arrival of computers. Therefore initial textual error correction research was centered around correction of spelling errors and typos. Most of the correction methods for other types of errors are also based on spelling correction techniques or their adaptations.

Alberga [3] defines the spelling correction problem as follows: "Given a list of words...and a string of characters which does not occur in that list, can one determine with some fair degree of certainty whether that string is a misspelling of one of the words in the list?" Two basic approaches to the problem are also identified. The first is based on string similarity, where the correction task is viewed as finding the word in the list which is most similar to the one to be corrected. The second is a probabilistic method, where given two words, the task is to compute the probability of one being a misspelled instance of the other. If this can be done, then

the task is simply to find the word in the list that yields the highest probability when paired with the misspelled word. The paper also presents a comparative evaluation of various correction methods available at the time.

While numerous methods have been proposed to compute the similarity of two strings, the most well-known and widely used method is probably the edit distance algorithm [142]. The algorithm is presented in the context of string-to-string correction problem, which is basically spelling correction. It measures the distance between two strings as the minimum possible cost of converting one string into the other using edit operations. There are three operations allowed in the conversion process: Inserting a character, deleting a character, replacing a character with a different character. The edit distance algorithm is based on the dynamic programming concept, and can compute the minimum cost edit sequence in $O(m \times n)$ time for strings of length m and n. An extension of the algorithm that allows transposition errors, where two adjacent characters are swapped, has also been developed. The extended version of the algorithm still has the same time complexity, with certain restrictions on edit costs [78].

Church and Gale [16] proposed a probabilistic method for spelling correction, where the focus was on typos. Their approach is based on the noisy channel model, where the correct version of a typo t is found by finding the correct word c that maximizes the probability $P(t|c)P(c)$. Given a word with a typo, this method first finds the words from a word list that has an edit distance of one to the input word. $P(c)$ for each of the candidates is computed using their occurrence frequency in AP newswire. $P(t|c)$ is also computed from the same corpus using the frequency of

the edit operation that converts c to t. Confusion probabilities are computed using a bootstrapping approach. The first pass computes the correction for each typo using uniform confusion probabilities. When the correction is completed, confusion probabilities are updated using their frequency in the first pass. The process is then iterated using the new probabilities. The proposed method achieved 87% correction rate, where three human judges determined the correct spelling using a majority voting scheme. Accuracy of two judges were 99% while the third one achieved 96%. The authors also experimented with exploiting the context of the words, and were able to boost their accuracy from 87% to 90%. They also show that a poor estimation of the context hurts the performance. The major limitation of their work is that it is limited to the correction of words with a single typo.

Ristad and Yianilos [118] proposed a stochastic edit distance model, where the cost of an edit operation reflects its likelihood. The costs associated with the operations are learned from a training corpus of string pairs, using an expectation maximization (EM) approach [23]. They model the edit distance problem with a stochastic string transduction approach, where one string generates the other one. One implication of this approach is that even the identical string pairs will have non-zero distance, while traditionally the distance is zero. Another difference is that the model is designed to compute a probability function between string pairs, rather than an edit sequence. One way to get the probability of a string pair is to simply use the probability of the edit sequence with the lowest cost. This is simply a stochastic version of the traditional edit distance methods, and the authors call it Viterbi edit distance. Another method, called stochastic edit distance,

is to sum over probabilities of all possible edit sequences, which gives the overall probability of the pair regardless of the actual transduction steps. The probabilistic edit distance approach achieves a much lower error rate for a word pronunciation problem compared to non-probabilistic edit distance.

Oommen and Kashyap [103] presented a new model for edit operations, that is still based on basic character-level operations. However, their model allows for probability distributions of these character level operations that are arbitrary. They present the model as a formal basis for information theoretic syntactic pattern recognition. They model is optimal in the sense that it minimizes the risk of errors. The evaluate the system using synthetic data, and achieve 97.66% accuracy. The performance of the system degrades slowly as noise is added to the underlying probability distributions. We used their model in our initial OCR correction work. The details of this model are provided in Section 3.3.1.

Brill and Moore [12] introduced an improved noisy channel model for spelling correction that allows arbitrary string-to-string edit operations. Their model can map strings of arbitrary length. Their set of edit operations is a superset of the traditional character-based edit operations. Moreover, they condition the probability of edit operations on the position of the character to be edited within the string. Their model of the typing process assumes that the user picks a word to type, segments it into smaller parts, and types each segment, possibly with errors. The major strength of this formulation is its ability to handle cognitive and pho-

netic spelling errors, not just typographical ones.[4] To illustrate, the user may chose to type '`physical`', partition it as '`ph y s i c al`', and then type '`fisikle`' with probability $P(f|ph)P(i|y)P(s|s)P(k|c)P(le|al)$. While the probability distribution for the partitions is part of the model, they drop this item in practice since poorly estimating it hurts the performance. Probability of the original word is computed using a language model. The parameters of the model is estimated by training the model on a corpus of misspelling/correction pairs. For each pair, first the edit sequence with the minimum cost is computed; then all the non-match edit operations, and also their extended versions including their context, are counted for frequency-based probability estimation. Empirical evaluation of the model shows that it performs better than single character-based edit models. Moreover, the positional information about the errors improve performance, especially for richer edit sets. Brill and Moore achieve a correction accuracy of 95%, without using a language model. Although they report results with a language model as well, the figures are not comparable since they are computed differently.

While string similarity-based methods can be quite effective in correction of non-word errors, valid word errors cannot be handled without using context. Golding and Roth [40] propose applying the Winnow [76] algorithm to perform context sensitive spelling correction for valid-word errors. The problem is defined as word disambiguation where a correct word, referred to as a concept, is to be selected from a pre-defined confusion set. The system is based on neural network paradigm,

[4] A cognitive spelling error occurs when the user types a string the way the user intended. However, the string is either a word with a different meaning than what the user intended, or the string is not a word at all.

and uses a network of nodes each representing a word or a collocation feature. A group of these nodes represents each concept in the confusion set. A variation of the Winnow algorithm is used by each node to learn the node's dependence on other nodes. At correction time, a weighted sum of all the nodes representing a concept is computed, and the concept with the highest sum is selected as system output. The system is evaluated on various confusion sets, and achieves an accuracy ranging from 70% to 100%.

Mangu and Brill [83] proposed a rule-based approach to the problem. They use a transformation-based learning approach to automatically learn a set of rules for correction from a training corpus. The rules are designed to capture contextual information to determine the correct word in that context. For example, {`except, accept`} can be a confusion set, and a transformation rule for that set can be "if the previous words is `to`, then `except` → `accept`". Transformation-based learning starts from an initial prediction of the correct solution, and iterates over a set of rules generated from pre-defined templates to improve accuracy of the system. At each pass, the rule in the candidate list that best improves an objective function is selected and added to the list of selected rules. The objective function is usually the classification accuracy on a test set for which a gold standard is available. The output of the learning process is an ordered list of transformation rules similar to the example rule given above. The proposed rule-based system is evaluated on the Brown corpus, using commonly confused words listed in Random House dictionary. The best result achieved was 93%, which was achieved when part-of-speech tags of the words were used as available features in the rules. The accuracy of this approach

is slightly lower than the Winnow-based methods proposed by Golding and Roth [40]. However, the set of rules learned is much smaller. Moreover, the rules are highly readable, hence they can easily be extended or augmented manually.

2.2.2 OCR Correction

Improving the accuracy of OCR systems has been an important area of research, resulting in numerous correction techniques. Some systems integrate the post-processor with the actual character recognizer to allow interaction between the two. Others treat OCR as a black box, and perform error detection and correction on its output, generally employing word and/or character-level n-grams along with character confusion probabilities.

An early example of work in this area was presented by Riseman and Hanson [117]. Their approach to OCR correction used dictionaries and positional n-grams to post-process recognized text. Dictionary-based correction methods detect an error if the word in question does not appear in the dictionary. Correction is performed by searching through the dictionary to find the most similar word. If the search provides a result of only one word that differs by only one character, it is selected as the correct version of the erroneous word. If there are multiple such words, the system rejects the word. In a situation where no words differ by only one character, but a single word exists that differs by two characters, it is selected for correction. The storage and processing requirements of the dictionary-based system increases with the dictionary size, which was a problem at the time the research

was conducted due to memory size limitations. N-gram-based methods overcome this problem by using positional n-grams extracted from a dictionary instead of searching a dictionary directly. In order to reduce storage requirements, only the existence of n-grams are considered and n-gram probabilities are ignored. An error is detected if it contains an n-gram that does not occur in the dictionary. Once the error is detected, and its position within the word is determined, all possible n-grams involving the erroneous position are searched. If a single character can be found that does not violate any n-gram constraints, the error is corrected by using that character. While using n-grams is more efficient in terms of processing time, their detection and correction coverage is worse than that of dictionaries. Dictionary-based system and various variations of the n-gram-based method were evaluated. Correction rates ranged from 60% to 95%.

An extension to the independent post-processor discussed above was also developed by integrating it with the classifier [48]. When the post-processor is used independently, about 99% of the errors can be detected and over 50% of the errors detected can be corrected. However, a high recognition accuracy comes at the expense of a high reject rate of 21%. With this approach, words are rejected by the correction system when multiple admissible substitution alternatives exist for a mis-recognized character. When this occurs, the post-processor does create a list of candidate characters and send this list to the classifier for reclassification. Since the number of alternatives are reduced, the classifier might be able to make a better decision. Using this approach, 4,000 out of the 4,500 test words that were originally rejected were then corrected. The best results achieved were as follows: about a 2%

word error rate, a 1% reject rate. The initial error rate was 45%.

Rosenbaum and Hilliard [122] designed a post-processor to improve the output of an OCR system specifically designed to process mail. This OCR system has two different channels: one for recognizing numeric fields and one for recognizing alphabetic fields. However, only alphabetic fields are corrected. The correction system is lexicon-based. Correction is performed using the standard noisy channel framework to select the most likely entry in the lexicon. The channel model uses character confusion probabilities to estimate the probability of OCR output given a correct word. Character merge/split errors are also handled if the system flags the possibility of such an error based on the presence of error-prone characters. The correction system manages to correct 97.7% of mis-recognized words.

Srihari et al. [129] use three information sources to perform correction. Channel characteristics provide the confusion probabilities for the recognition system. A top-down context is used to refer to the lexicon, and a bottom-up context is provided by a character-level language model. The lexicon is represented as a trie for efficiency. A set of algorithms are presented to efficiently lookup and correct words. The system achieves correction rates of up to 87% on artificially corrupted data.

Omnifont recognition system developed by Kahan et al. [62] include a spelling correction component that is used after the recognition is completed. Each output is examined using the UNIX "spell" program. If a word is rejected by spell, a list of spelling variants for that word are generated, along with their costs. The costs of the variants are computed using the confusion probabilities that are estimated during training. The variant with the lowest cost in the list is tested using spell, and

if it is accepted, it replaces the misspelled word. If spell rejects the variant, another list of variants is generated from it, and added to the original list. The search stops when a correction is found, or the minimum cost gets too high. The system also uses an n-gram-based error detection system to catch some of the errors that are missed by spell.

Goshtasby and Ehrich [42] provide a brief overview of post-processing methods and present a new method based on probabilistic relaxation labeling [123]. The input to their method is a set of candidates and associated probabilities for each character to be recognized. The candidates and their probabilities are provided by the classification component of the OCR system. The post-processor updates these probabilities in an interactive fashion using the context information provided by the neighboring characters on each side. The contextual probability of a character is determined using occurrence frequencies of character pairs. The formulation allows the influence of a neighboring character to be adjusted based on its positional distance from the target character. While the performance of the system is demonstrated on some sample text, no accuracy figure is reported.

Sinha and Prasada [127] present a post-processing system that uses various heuristics, an augmented dictionary, and a modified Viterbi algorithm to perform error correction. A set of most frequent words are used to form an initial dictionary. This dictionary is then extended to include all words that differ by only one character from the words already in the dictionary. If a word and its variations generated using a character confusion table are not present in the dictionary, then a modified Viterbi algorithm is used to perform correction. At correction time, the system creates a

transient dictionary by adding words from the document being corrected that did not match any word in the augmented system dictionary. The confusion matrix is also modified using the character substitutions observed in the document, which is important for adaptation if the collection being processed is not homogeneous in terms of error characteristics. The system also uses a set of heuristics to determine the word boundaries, for dictionary matching, and for correction via the Viterbi search. The resulting OCR system achieves 97% word recognition rate, ignoring punctuation.

Jones et al. [61] describe an OCR post-processing system that combines various information sources that characterize the recognition system and the input documents. The documents are modeled using various character and word-level n-gram statistics. The recognition system is characterized using mainly mappings between character sequences that are empirically learned from training data. A second model uses information about the shapes of the characters to provide a crude estimate of mapping probabilities to be used when empirical estimation is not possible or precise estimates are not necessary. This second model is also used for ranking and selecting the rewrite rules generated by the empirical model during training. The correction component of the system is designed around a stratified algorithm. The first phase performs isolated word correction using both recognizer and source probabilities. The second phase attempt to correct word split errors by identifying possible split pairs and by evaluating their probability using the first phase. The last phase uses word bi-gram probabilities to reorder the correction candidates and also to suggest new candidates. The three phases interact with each other to guide the search.

Jones et al. report error reduction rates of 70% to 90% under various conditions.

Tong and Evans [137] present a post-processor that is based on statistical language modeling. The system uses an indexing system based on the character n-grams of the lexicon. The first step of correction is to pick top M candidates from the lexicon for each word in the line to be corrected. The top M candidates are selected based on the number of character n-grams that a target word and a lexicon word have in common. Next, a probability is assigned to each of the M candidates using the probabilistic edit distance between the M candidates and the target OCR word. Top N candidates are selected for the final phase for each OCR token processed. Once this step is completed, there are N potential corrections for each token of the input sentence that does not appear in the lexicon. If valid words are also being corrected, then all tokens in the input will have N candidate corrections. Next, the Viterbi algorithm is used to find the most probable sequence through the lattice generated for the whole sentence. The confusion probabilities for the probabilistic edit distance computation are estimated using an iterative process. Correction starts with an initial estimate. The system re-estimates the probabilities after each correction step, using the corrected text as ground truth, which removes the need for ground truth text for training the confusion probabilities. Word merge/split errors are not handled, and initial n-gram indexing may fail to fetch the correct lexicon entry, especially for short words. The post-processor achieves an error reduction up to 60.2%, considering only alphabetic tokens.

Perez-Cortes et al. [107] describe a correction system based on a stochastic FSM parser that accepts the smallest k-testable language consistent with a repre-

sentative language sample.[5] Depending on the value of k, correction can be restricted to words from the training sample, or variations may be allowed. The system uses both the probabilities of the grammar rules representing the language and the confusion probabilities of the recognizer. Correction is performed using a stochastic error-correcting parser [5]. Perez-Cortes et al. report reducing the error rate from 33% to below 2% on OCR output of hand-written Spanish names from forms.

Pal et al. [105] describe a method for OCR error correction of an inflectional Indian language, Bangla, using morphological parsing. The inflectional nature of the language is captured using two lexicons, one for roots, and one for suffixes. The first step in the post-processing of OCR output is error detection. The error detection is performed by forward matching the word with the root lexicon, and reverse matching it with the suffix lexicon. If a valid root/suffix combination covers the whole word, it is assumed correct, otherwise it is passed on to the correction routine. The correction phase can only handle words with a single error. It is based on a set of heuristics that determine whether the mis-recognized character is in the root or the suffix. Once the position of the error is determined, all possible variations of the erroneous segment are generated using the confusion sets for each character. These confusion sets are empirically estimated, and the search is performed in decreasing order of probability. The correction system achieves 84.22% correction accuracy, which improves the accuracy of the OCR system from 80.55% to 95.29%. Although it is limited to single errors, the system demonstrates the possibility of correcting

[5]Simply put, a language is k-testable if membership of a string can be determined by checking its substrings of length k or less.

OCR errors in morphologically rich languages.

Taghva and Stofsky [134] take a different approach to post-processing and propose an interactive spelling correction system specifically designed for OCR error correction. It works within an Emacs buffer, using the same interface as the spelling systems for Emacs. The first step is to determine the tokens to be spell checked. This differs considerably from traditional spell checking, since OCR misspellings may contain non alphabetical characters as well as spaces. The second step is to generate a ranked list of correction candidates that are presented to the user for selection. The list is generated using various algorithms and heuristics based on existing research on OCR error correction. The system also dynamically incorporates the mappings extracted from the words that are manually corrected by the user. The performance of the system is lower than other post-processing systems. However, to evaluate this system properly would require spell checking a large homogeneous collection.

Dictionary-based correction techniques provide very good error detection and correction capabilities, usually better than methods that do not use a lexicon. However, they are based on the assumption that all the words in the document being processed are in the dictionary. While hybrid approaches address this problem, they do so by making compromises. Words that are in the dictionary but not in the document are also a problem, since they increase the chance of mis-corrections. Strohmaier et al. [132] address these problems by dynamically extending dictionaries using documents from the Internet that are relevant to the document being processed. Correction performance of various dictionaries, including Web-based ones, are evaluated, where a "perfect dictionary", which includes all the words in the

document and no words outside the document, is used as the upper-bound. The results show that the coverage of combined dictionaries that include both the static and crawled terms is the best. The best correction results are also obtained using crawled or combined dictionaries.

Languages that do not have explicit word boundaries present a challenge for correction systems, which are almost always designed to work on words. Meknavin et al. [88] present an error correction method for Thai OCR that uses a part-of-speech trigram model and the Winnow algorithm [76]. The system first identifies the regions in the text that have a low probability using a word segmentation algorithm based on dictionary lookups. This step is used to narrow the scope of regions that will be considered for correction, unlike traditional methods that assumes any section of the input may contain an error. Once the problem segments are identified, first a Part-of-Speech (POS)-based correction is performed, which is followed by a feature-based correction that combines various features using the Winnow algorithm. The overall post-processing system corrects 90% of non-word errors, 88% of real-word errors, and introduces 1.6% new errors.

Unfortunately, there is no standard evaluation benchmark for OCR correction, and implementations are usually not publicly available, making a direct comparison difficult. Most of the post-processing systems discussed in this section are not suitable for post-processing low-density languages as they rely on lexicons or require extra information, such as probabilities, to be provided by the OCR system. Furthermore, very few systems actually address word merge/split errors. Instead most systems focus on individual tokens separated by whitespace.

2.2.3 Diacritic Restoration

Missing diacritics is an important problem for many applications, because many words are disambiguated solely by diacritics. For example, the word '`isin`' in diacritics free text may be interpreted as four different Turkish words: '`ısın`' (warm up), '`ışın`' (ray), '`işin`' (your job/work), and '`isin`' (your/of soot).

Yarowsky [152] describes and compares three corpus-based methods, and a baseline, for accent restoration in Spanish and French. It is observed that many words in these languages have only a single accent pattern, and therefore do not exhibit any ambiguity. Many of the words that have multiple accent patterns have a skewed distribution which makes one pattern dominant. As a result, when all words in a sample are considered a simple baseline using only the most common accent pattern achieves a 98.7% mean accuracy for Spanish and 97.6% for French. Note, however, the accuracy for ambiguous accented words is much lower. The first restoration technique is based on n-gram tagging. One variation uses a model of suffix sequences, since many accent patterns can be disambiguated given the suffix. Another variation uses a lexicon and a morphological analyzer, and includes traditional POS tags in the analysis. The accuracy of this approach ranges from 52.3% to 97.8% depending on the target ambiguous word, where accuracy refers to how often the correct accented version is selected for a particular unaccented word that has more than one accented version. The second approach uses a Bayesian classifier. This approach is based on the difference in probability distributions of the neighboring words in a window surrounding the word to be disambiguated. Given

two alternative patterns, likelihood ratios for the words in context are computed, and the pattern that yields a higher probability is selected. The accuracy of this approach ranges from 62.9% to 94.4%. The two approaches have different strengths and limitations. Therefore, a third model that combines the two approaches is developed. The combination is performed using decision lists [119]. The features are collocations of words and word combinations at various positions. If additional resources such as morphological analyzers and POS taggers are available, collocations of roots and POS tags are also considered. For each ambiguous word and class, a classifier is trained. The word classifiers are retained if the classifier for its general class cannot reliably disambiguate the word. The combined approach outperforms the other two methods, achieving 78.4% to 98.4% accuarcy.

Zweigenbaum and Grabar [156] present their work on accenting French MeSH terms. They focus on accenting unknown words, since their problem domain contain a lot of special terms that are unlikely to be covered using a lexicon-based approach. The first approach formulates the problem as POS tagging, where individual characters in a word are treated as "words", and the accented form of each character is treated as its tag. The second approach represents each word using a mixed content representation that uses the character to be accented as the pivot and generates a context list by alternately picking characters to its right and left. A transducer is generated using these context representations, where the input is the context and the output is the accented character. A minimal combined transducer that can discriminate all training samples is generated from individual word transducers. At correction time, if an input word causes ambiguous output, relative

frequencies of competing paths are used to determine the output. They report a set of precision/recall figures that indicates good performance on unknown words under various conditions.

Mihalcea and Nastase [92] present a correction approach based on letter level learning that does not require any resources other than some properly accented text for training. They use the unaccented context characters as features without any processing, and train classifiers using instance-based learning methods. The method is evaluated on Czech, Hungarian, Polish, and Romanian, and achieves average precision rates of 97.83%, 97.04%, 99.02%, and 98.17% respectively for disambiguating the character confusion sets in these languages. Interestingly, the accuracy achieved on Hungarian, which has five confusion sets, is lower than the accuracy on Czech, which has thirteen ambiguous sets and a slightly smaller amount of training data.

When an existing OCR system is used to recognize text in a language that the system was not originally designed for, most of the recognition errors are made on characters specific to the new language. For the Latin script, these language specific characters almost always have diacritics. Consequently, the OCR system might recognize these language specific characters as their diacritic free versions, which would reduce the correction problem to that of diacritic restoration. We have not pursued this route because (1) accented characters are not always recognized as their diacritic free versions, (2) diacritic restoration relies on context that might contain other OCR errors, making restoration less reliable.

2.3 Knowledge Elicitation for Low-Density Languages

As the need for rapidly acquiring computational resources for new languages became more evident, research on quickly acquiring resources in a controlled way gained some momentum. While acquiring linguistic knowledge from native speakers is a usual method employed by field linguists, our focus here is on acquiring computational resources. The elicitation efforts that we are aware of are all built around the ultimate goal of creating an MT system between a well-studied language (e.g. English, Spanish) and a low-density one.

Project Boas [87, 99, 126] is an example system that is developed for machine-aided linguistic resource acquisition. The goal is to rapidly create an MT system for translation from a low-density language into English. For this project, the envisioned scenario involves a two person team, one person is a bilingual speaker and the other person is a computer programmer, who will work together to create the necessary resources for MT in six months. The output of Project Boas is morphological and syntactic grammars, language specific features (e.g. dates, numbers, punctuation, orthography, etc), a translation lexicon, and a transfer grammar between the low-density language and English.

Another similar approach is the NICE/AVENUE MT project from Carnegie-Mellon University [110, 111]. The goal of the project is to semi-automatically learn transfer rules between a well studied language and a low-density one using a carefully designed elicitation corpus and a language informant who is a bilingual speaker. The informant is asked to translate phrases and sentences from the elicitation corpus

into the low-density language being studied, and align the source sentence and its translation at word level. The system automatically generates and refines transfer rules based on the translations provided by the informant. The corpus is crafted specifically to cover as many grammatical features and constructions as possible with a minimal amount of manual translations. A minimal pair of phrases, which differ only in aspect, allows the system to infer how that particular aspect is transferred over to the target language. The system dynamically chooses the sentences to be translated based on the evidence collected so far to minimize the number of translations required to infer the transfer rules.

The advantage of the elicitation-based methods is that they require a smaller number of translation pairs to achieve good coverage, since the sentences are carefully selected to test various linguistic phenomena. Unfortunately, knowledge elicitation is a very challenging task that presents numerous challenges, as discussed in [111]. However, for languages where there is no parallel corpora or written resources, elicitation may well be the only possible way to obtain computational resources.

If a sizable parallel corpus exists for a given language pair, the value of elicitation is dubious.[6] Although it is true that uncontrolled parallel corpora may not contain certain linguistic features and elicitation allows particular features to be targeted since the process is tightly controlled by the researcher, the number of features to be covered is huge. For example, many languages that assign gender to nouns do it virtually randomly, making it almost impossible to learn a concise set of rules

[6]We are looking at the problem from an NLP perspective. While statistical methods provide state-of-the-art performance, they provide very little intelligible linguistic knowledge. If the goal is to acquire linguistic knowledge, then controlled acquisition using elicitation is a valuable method.

regarding gender, which means that one would need to have each noun translated in order to infer its gender. Furthermore, an uncontrolled corpus will cover most of the frequent constructions and features. While the number of constructions covered may be small, being frequent constructions, they will probably account for a big percentage of the observed utterances for that language.

2.4 Methods for Reducing Annotation Requirements

An advantage of supervised learning models is that they generally perform better than unsupervised models. They are also easier to develop. A disadvantage of supervised learning models is the associated annotation cost, which can be quite high. The difficulty and cost of obtaining manual annotations has lead researchers to devise numerous methods for reducing the amount of annotated training data required to train supervised models.

2.4.1 Active Learning

One way to reduce the amount of annotations required is to couple training and annotation together to allow the learner direct annotation based on the learner's needs. This approach is termed Active Learning (AL), as the learner is actively involved in the process.

AL assumes the availability of a seed set of labeled data to start the training process, a manual annotator to provide correct annotations when necessary, and a pool of unlabeled data to choose from for manual annotation. The basic idea is that

learners know best where they are having trouble and which samples would help them learn better, and therefore should guide the annotation process. The training is done in an iterative fashion. At each iteration the learner goes over the pool of unlabeled data and selects a set which is expected to improve performance most if labeled and included in the training data. Overall structure of the AL approach is the same across applications and learners. The challenging part of the problem is the selection of training samples to be manually annotated. The specifics of the selection process differ depending on the learning method and the application. Furthermore, the learner's selection performance directly affects the performance of AL.

AL is studied in the greater context of Machine Learning (ML) [18, 36] and has been applied to various NLP problems [6, 56, 86, 104, 135, 136]. In this section, we will cover some of the AL studies on parsing.

Tang et al. [135] use AL to train a shallow semantic parser. They focus on two factors while selecting the samples to be annotated. First, the selection should be representative, reflecting the characteristics of the training corpus. The training data is clustered based on the similarity of their parse structures. Clustering similar samples together simplifies the creation of a representative selection. Second, the selection should come from portions of the training data where the current model is not performing well. It is expected that learners can improve their performance by training more on samples from problematic regions. Various entropy-based measures are used to select samples for which the current parsing model is uncertain. The samples are then manually annotated and added to the training set for the next iteration. The parsing performance is evaluated using an exact match method

where a parse is considered correct only if it is identical to the reference parse. AL uses less than a third of the training samples used by random sampling to achieve the same performance level. Furthermore, AL results obtained after selecting only 2,800 samples is very close to the performance achieved by training on the complete training set of 20,000 samples.

Baldridge and Osborne [6] apply AL to the Head-driven Phrase Structure Grammar (HPSG) framework. The problem setting is slightly different compared to the usual statistical parsing framework, since the goal is to choose among multiple parse trees generated by a non-probabilistic grammar [33]. The researchers experimented with two different learners: one using log-linear models and the other using perceptrons. The two learners each used two feature sets, effectively providing results for four different learners. For sample selection, the researchers compare two different approaches. The first approach uses the tree entropy [55] metric to guide sample selection. The second approach is committee-based. With this approach, multiple parsing models are used. A sample is selected if the parsing models disagree on the parse tree for that particular sample. Both approaches perform better than random sampling, and require fewer training samples, requiring only 1,450 samples to achieve the same performance obtained using 3,000 random samples. In a later study Osborne and Baldridge [104] show that ensemble-based models can be combined with AL to improve the performance even further, and the annotation cost can be reduced as much as 73% using ensemble-based AL.

Hwa [56] provides a more detailed study of the sample selection process using a single learner. Each sample in the training pool is associated with a Training

Utility Value (TUV), which indicates how much the learner can benefit from training on that sample. There are three classes of criteria for computing TUV. *Problem Space* focuses on the properties of the problem and the learner, assigning higher values to samples that are inherently difficult to learn, or occur more frequently. *Performance of the hypothesis* focuses on the samples with which the current model is having difficulty. *Parameters of the hypothesis* selects the samples that would have a bigger impact on the parameters of the model. Evaluation of TUV functions based on each criteria on Prepositional-Phrase (PP) attachment problem shows that they perform better than the baseline, and the best performance is achieved by a combined metric that uses both performance of the hypothesis and parameters of the hypothesis, achieving a final performance level of 83.8% using 47% fewer training samples than used with random sampling. Evaluations on parsing are also positive. Tree entropy-based selection [55] achieves a final parsing performance of 88%, computed using crossing-bracket metric, using 27% fewer constituents for training compared to random sampling.

As we discuss in Section 5.4.3, selection of training samples becomes quite important when the training data is noisy. AL research provides valuable insight into devising clever sample selection methods to get the most out of noisy annotations generated via cross-lingual utilization.

2.4.2 Co-training

Co-training is similar to AL in many ways. Training starts with a seed set of annotated samples, and new samples are annotated iteratively from a pool of unlabeled samples and added to the training set. The difference is the way new samples are annotated and selected. Co-training utilizes two sufficiently different learners[7] to create training samples for each other, rather than having a manual annotator as in AL. At each iteration, both learners label the pool of available samples and assign a confidence value to their annotations. A set of high confidence samples are then added to the training set for the next iteration. Co-training has been successfully applied to various NLP problems [10, 41, 98, 109, 151], including parsing [124, 130].

Sarkar [124] presents a method for applying co-training to parsing. The parsing paradigm used is Lexicalized Tree Adjoining Grammar (LTAG), which is performed in two steps. The first step assigns a lexicalized structure to each word in the sentence and the second step combines these structures to form the parse tree. These steps have their own probabilistic models, which constrain each other, making them suitable for co-training. To address the lexical coverage problem, a tag dictionary is created that covers possible tags for words in the unlabeled training data. The training process starts with a small set of labeled samples, and iteratively extends this set using confidently labeled samples provided by the current parser. The parser trained on 9,695 labeled sentences achieve 72.2% precision and 69.1% recall using labeled bracketing for evaluation. When co-training is used to take advantage of

[7]The learners can be two distinct learning methods, or a single method that works on a problem from two different perspectives.

30,137 unlabeled sentences, performance reaches 80% precision and 79.6% recall.

Steedman et al. [130] employ two different parsers for co-training, as opposed to using two components of a single parser as in [124]. They use a Probabilistic Context Free Grammar (PCFG) and an LTAG parser and train them iteratively, extending the training set of each parser using a set of high-probability parses generated by the other parser. Both parsers are initialized using a seed set of parse trees, and co-trained using a much larger set of unlabeled sentences. Parsing evaluations, using F-measure over labeled constituents, show that co-training improves parsing performance up to 3.2%. Evaluations also show that co-training is useful when the seed training set is small and gets less useful as the seed set size increases. Furthermore, co-training helps when the seed set is from a different domain than the unlabeled or test set.

2.5 Projection of NLP Resources via Parallel Corpora

Exploiting parallel corpora (1) to transfer existing resources from one language to another and (2) to acquire new resources is an active area of research. Resnik [112] provides an excellent argument for the utility of parallel corpora in acquiring monolingual resources. Here, we present some specific applications of the idea.

Yarowsky and Ngai [154] present, to our knowledge, the first attempt to utilize parallel text to create stand-alone monolingual tools/resources for a new language utilizing existing resources in another language. Word level alignment links are utilized as a bridge to project POS tags and Noun Phrase (NP) brackets from English

to French. After the projection, a stand-alone monolingual POS tagger and an NP bracketer are trained on the projected annotations. The researchers observe that projected POS tags achieve only 69% accuracy for automatic word alignments, and 78% for manual alignments. The accuracy of a French POS tagger trained on noisy projection is 82%. To improve the performance, they employ filtering to exclude poor POS projections from the training data, and develop more robust methods with which to train the POS tagger. Noise-robust training achieves accuracy well above 90%, performing at a level close to a tagger trained on clean French data. NP bracket projection achieves a 53% F-measure using manual alignments, while the performance of a robust bracketer trained using Transformation-Based Learning (TBL) [11] on noisy projections is 81% F-measure. Evaluation used exact match of brackets. When the computed brackets are manually labeled acceptable/unacceptable by annotators, performance reaches 91%. Projection of NP brackets are also evaluated for Chinese, where the percentage of acceptable brackets is 51% for automatic word alignments and 86% for manual alignments.

Yarowsky et al. [155] extend the same approach to named entity tagging and morphological analysis. The accuracy of named entity tags projected from English onto French is 64% while the accuracy of a co-training-based tagger trained on the noisy projections is 85%, which is only 1% below the accuracy of the English tagger that generated the original tags. Projecting morphological analysis is a different process, which uses the English words as a bridge to link morphological variations of French words together. That is, given two English words that are morphological variations of each other, the French words that they are aligned to are also likely to

be morphologically related. The precision of a morphological analyzer created by utilizing a parallel corpus is as high as 99.9% for French, 98.6% for Spanish, and 91.3% for Czech.

Riloff et al. [116] utilize the projection idea to create Information Extraction (IE) systems. Rather than relying on the existence of a suitable parallel corpus, they use MT to create a pseudo-parallel corpus by translating available monolingual data. Once the parallel corpus is generated, they use an existing IE system to tag the source side, and project the IE annotations and NP boundaries to the target side using the mechanisms described in [154]. The last step is to train a stand-alone IE tagger for the target language using the projected annotations. They use English and French data for experiments, achieving 45% and 54% F-measures respectively using TBL-based annotators trained monolingually on manually-annotated data. After exploring various projection paths, they achieve the best performance using a combination approach that uses multiple IE annotation systems created via projection; this approach achieves a 48% F-measure for French. As a reference, annotator agreement is 51% to 59% for English and 43% to 54% for French.

Rogati et al. [121] present a method for building an Arabic stemmer that utilizes the following: an English stemmer, an English-Arabic parallel corpus, and a monolingual Arabic corpus. The first steps of the process involve stemming the English side of the parallel corpus and assigning an initial probability distribution to possible Arabic stems. Once the initial stemming is ready, a translation model using a modified version of IBM Model 1 [13] is used to compute translation probabilities between English and Arabic stems. Next, a score for each possible Arabic stem

is computed based on translation probabilities. The process is then iterated to update the translation probabilities using the new stem probabilities and vice versa. To reduce the effect of sparsity, stem probabilities are computed based on affix probabilities, which is a much smaller set. After the cross-lingual step is complete, an optional monolingual step can be used to update the statistical profile of the model and learn new stems on a new corpus. The corpus is stemmed using the current stemmer, and the stem and affix probabilities are updated based on the stemming results. This is also an iterative process, stemming the corpus with the updated stemmer, and updating the stemmer using the new results. The stemmer created in this fashion achieves 87.5% agreement with a proprietary Arabic stemmer that is built using rules, affix lists, and manual annotations. A task-based evaluation of the stemmer is also performed using IR. The projected stemmer improved mean-average-precision 22% to 38% relative to unstemmed text, achieving 93% to 96% percent of the performance obtained using the proprietary stemmer.

Bentivogli et al. [8] evaluate the projection approach within the framework of MultiSemCor project [9], which aims to build an English-Italian parallel corpus with POS, lemma, and word sense annotations. The starting point of the MultiSemCor is SemCor corpus in English [74]. An Italian version of the raw text in SemCor is generated by having the English text professionally translated. The resulting parallel corpus is word-aligned using a dictionary-based aligner, KNOwledge-intensive Word Aligner (KNOWA) [108], while the annotations on the English side are projected onto the Italian side over word alignments. The resulting Italian annotations have 86.0% precision and 69.7% recall. This process left 19% of the Italian words unan-

notated. The paper by Bentivogli et al. provide a detailed analysis of the quality issues and the types of errors.

Diab and Resnik [26] present an unsupervised Word Sense Disambiguation (WSD) method that utilizes parallel corpora in a slightly different way.[8] Their method does not project existing annotations across parallel corpora, but rather treats one side of the parallel corpora as implicit sense annotations for the other side and generates explicit annotations for both sides. The method is based on the observation that different senses of a word often translate into distinct words in other languages. For example, financial and geographical senses of the English word '`bank`' are translated as '`banque`' and '`rive`' into French. For each French word, possible senses of all the English words that align to it are grouped together and assigned a score using mutual-reinforcement. Most of the English words aligned with '`banque`', for example, would have a financial sense, allowing the system to mark '`banque`' with the correct sense, and then project that sense over to '`bank`' on the English side when the two words are aligned to each other. The WSD system generated in this fashion achieved about 60% precision and 52% recall, performing better than many comparable unsupervised WSD systems.

Diab [25] utilizes the same idea to generate the first WSD system for Arabic. For the English words that are assigned the correct sense tag, the projected Arabic sense tags are accurate about 90% of the time. When evaluated accross all words, accuracy is 56.9%. For comparison, accuracy of unsupervised WSD systems on

[8]WSD is the task of selecting the correct sense for a word that has multiple senses, given the context in which the word is used. The correct sense is usually from a sense repository such as WordNet.

English are in the lower 50% range. The Ph.D. thesis of Diab [24] provides a very detailed presentation of the method and its evaluation.

Volk and Samuelsson [140] present ongoing work on building parallel treebanks, including cross-lingual utilization of a German parser to bootstrap a German-Swedish parallel treebank. Swedish sentences are first POS tagged using a Swedish tagger. Next, the POS tags are converted to the corresponding German tags. Finally, a German chunker is used to process the Swedish sentences. As the two languages are quite similar, performance is remarkably good, achieving 89% node label accuracy and 93% edge label accuracy.

All projection methods assume some form of direct correspondence between sentences that are translations of each other. Our projection work is the application of the same idea to parse trees, which have a deeper structure and hence are more challenging to project. Among the research we discussed above, work reported in [140] is the work that comes closest to tree projection. The main difference between their work and projection is the fact that in their work an existing chunker is used to process a very similar language directly; no parse trees are transferred from one language to another.

2.6 Multi-lingual Parsing Models

Multi-lingual parsing has become a very active research area, mainly driven by the desire to enhance existing Statistical Machine Translation (SMT) models that are linguistically impoverished.

Wu [144] introduced the Inversion Transduction Grammar (ITG) formalism that generates parallel sentences synchronously using a Context Free Grammar (CFG). Word ordering differences are addressed by applying the grammar rules from left-to-right or right-to-left, depending on the language. For example, the rule '`NP` $\rightarrow$ `A N`' would be applied right-to-left for English, and in the opposite direction for French, placing the adjective after the noun. The probabilistic version of the grammar is termed Stochastic Inversion Transduction Grammar (SITG) [146]. The model parameters are estimated iteratively using Expectation-Maximization (EM). Parsing within this framework is carried out simultaneously, guided by the following: (1) rules of the grammar (2) lexical translation probabilities for words that correspond to each other on either side of the derivation. Further biases and constraints are also employed, such as punctuation constraints to bound phrases and attachment preferences for words that do not have a counterpart in the other language. After the parsing is completed, the resulting brackets are post-processed to improve accuracy. A coarse grammar for one of the participating languages can be used as the basis of the bilingual grammar. SITG is employed for various applications such as translation guided segmentation, parallel bracketing, alignment at word and phrase level, and bilingual constraint transfer [145, 147]. Bilingual constraint transfer get pretty close to the projection idea discussed in Section 2.5. The basic difference is that an existing monolingual parse tree is used to constrain the bilingual parsing process, but the actual parsing is still performed using the bilingual grammar.

Hermjakob and Mooney [50] present a syntactic SMT system that relies on a deep analysis semantic parser and a mapper that maps the source language parse

trees to target language parse trees. Linearization of the target parse tree finalizes the translation process. While the parsing of the source tree is interesting in its own right, the most relevant part of this work is the tree mapping phase. The source parse tree is recursively mapped to a target tree. The mapping is performed using two resources: (1) the same machinery, roughly, used for parsing the source sentence (2) a bilingual dictionary that contains word and phrase translations. Reported translation results are better than the commercial MT systems used for comparison. No evaluation results for the quality of target parse trees are reported.

Alshawi et al. [4] introduce a dependency transduction model that is formulated using weighted finite state head transducers. These are more powerful than standard Weighted Finite State Transducers (WFSTs) as they allow reordering of the input, such as reversing the input sequence or moving chunks around. Dependency relations on each side of a parallel text, and the word level alignments between the two are constructed together. The dependency structure produced by the transducer is not meant to represent linguistic dependencies. It is simply a computational tool to model structural mappings between the two languages. The dependency transduction-based SMT system is evaluated on English to Spanish and English to Japanese translation, using speech transcripts as the input text. Accuracy of translations is 75% for Spanish and 70% for Japanese, using word edit distance between the system output and a reference translation. For comparison, simple word-for-word translation results in 47% and 43% accuracy, respectively.

Yamada and Knight [150] extend the IBM models [13] by introducing grammatical information into the scenario. They present a tree-to-string model that

takes a source language parse tree as input, and generates a surface string in the target language. The translation is carried out by reordering the input tree, inserting extra words at each node, and translating leaf words. Some subtrees in the source parse trees are flattened before translation to accommodate word ordering differences between languages. The model is evaluated using English-Japanese word alignment. Each proposed alignment is manually assigned a score: 1 for correct, 0.5 for not sure, and 0 for incorrect. Tree-to-string model achieves an average score of 0.582 while IBM Model 5 gets 0.431.

Eisner [32] presents a tree-to-tree alignment model built around Synchronous Tree Substitution Grammar (STSG) formalism that allows local distortions of the tree structure, in kind allows mappings between non-isomorphic trees. Conceptually, parse trees for each side of a parallel corpus are constructed synchronously, creating alignment links between sub-trees along the way. The model parameters are estimated using EM, which considers all possible derivations of each sentence pair, and which takes individual parse trees or forests as input.

Gildea [39] extends the tree-to-string and tree-to-tree alignment models by introducing a clone operation that creates a copy of a sub-tree from the source tree and places it somewhere in the target tree. This operation makes some re-orderings and structure changes that were not allowed by the original model. Introduction of the clone operation improved Alignment Error Rate (AER) for both alignment models: from 42% to 32% for tree-to-string alignment model and from 49% to 36% for tree-to-tree alignment model.

Kuhn [71] utilizes word alignment over multiple parallel texts to aid in monolingual PCFG grammar induction. Crossing links and discontinuities in aligned chunks are treated as evidence for *non*-constituency, as constituents are expected to map over as a block. The information obtained from parallel corpora regarding non-constituency is used to adjust expectation values computed using the inside-outside algorithm on monolingual text. The use of non-constituency information during training improves the unlabeled bracketing F-measure of a PCFG English parser from 51.3% to 57.5%.

Melamed [89] presents MultiText Grammar (MTG) formalization that allows the generating of multiple parallel text synchronously. This method allows different lengths for surface text, crossing alignment links, bilexical dependencies, and discontinuous constituents. This flexibility sets this method apart from most previous synchronous formalisms. Reordering of parse nodes are done explicitly using special production rules that include a permutation matrix. Potential applications of MTG are the translation of parse structure, generation of aligned parallel treebanks, and use of multitrees for translation, all of which are discussed in [90].

Smith and Smith [128] combine the following existing models for statistical dependency parsing, PCFG, and SMT. The unified model computes the best English parse, Korean parse, and word alignment for English-Korean sentence pairs. Unlike most multi-lingual parsing models, individual parse models are assumed to be independent of each other. Individual parsing models and the word translation model all constrain each other to find a parse pair and an alignment combination with the maximum combined probability. The combined bilingual parsing framework pro-

vides slight improvements over traditional PCFG when the size of the training data is very small (less than 100 sentences).

Multi-lingual parsing models in NLP are almost exclusively developed with SMT and/or word alignment in mind as the application. While some of the multi-lingual parsing models discussed in this section can be used for projection, and this possibility is explicitly acknowledged and explored in some of the referenced articles, our model which is discussed in Chapter 5 is the only model that is specifically designed for parse tree projection. It is also the first model that is actually used for tree projection and evaluated in terms of tree projection performance.

Chapter 3

Get the Words: Acquiring Electronic Corpora

3.1 Motivation

As discussed in Chapter 1, the first language resource required to train a statistical NLP model is electronic corpora. While we have a vast amount of electronic text available, especially since the explosive growth of World Wide Web (WWW), the set of languages that have a sizable electronic presence is still only a fraction of the world's languages. Many languages have virtually no electronic text at all. For such languages, the most practical way to generate electronic corpora is to convert existing corpora in other forms into electronic form.

Language occurs in two major forms: spoken and written. While people continuously generate spoken language, the persistent storage of language is mostly the written form, mainly because it has been the only form of persistent storage for a long time. Although large spoken archives have become a reality thanks to advances in recording and storage technology, not many of these archives exist. The large spoken archives that do exist are usually only available for well-studied languages. Therefore, for acquiring electronic corpora, we focus on written language as our main source. For the scope of this thesis, the corpora acquisition problem is viewed as the problem of converting written corpora into electronic form.

The ability to convert written text into electronic text is important for well-studied languages as well. NLP applications almost exclusively work on electronic text. Critical applications of NLP technology, such as rapid, rough translation in the field [52], IR from scanned documents [22], etc. still require conversion of printed information into electronic form for processing; and their performance can depend heavily on the quality of the conversion. While there are applications that skip the conversion process and work directly on the input such as IR on raw document images, comprehensive solutions that do not require conversion to an online form are still elusive for practical systems [27].

Often, the only feasible method for taking resources and text that are available in printed form and converting them to electronic form is OCR, owing to its speed and cost-effectiveness. Unfortunately, the speed and economy come at the cost of accuracy. Whether it is going to be used for resource acquisition, or as input to applications of NLP technology, the output of commercial OCR systems is far from perfect, especially for resource-poor languages [63]. Consequently, resulting resources may suffer from a considerable amount of noise introduced by OCR errors. More importantly, when it comes to acquiring resources for resource-poor languages the OCR system itself, or rather the lack of it, can become a problem. For many languages, there is no OCR system available at all, and the ones that are available have usually not gone through a development process that is as extensive as the OCR systems designed for well-studied languages.

Consequently, in order to use OCR to acquire resources for a new language, we either need to acquire an OCR system first, or devise methods to improve recog-

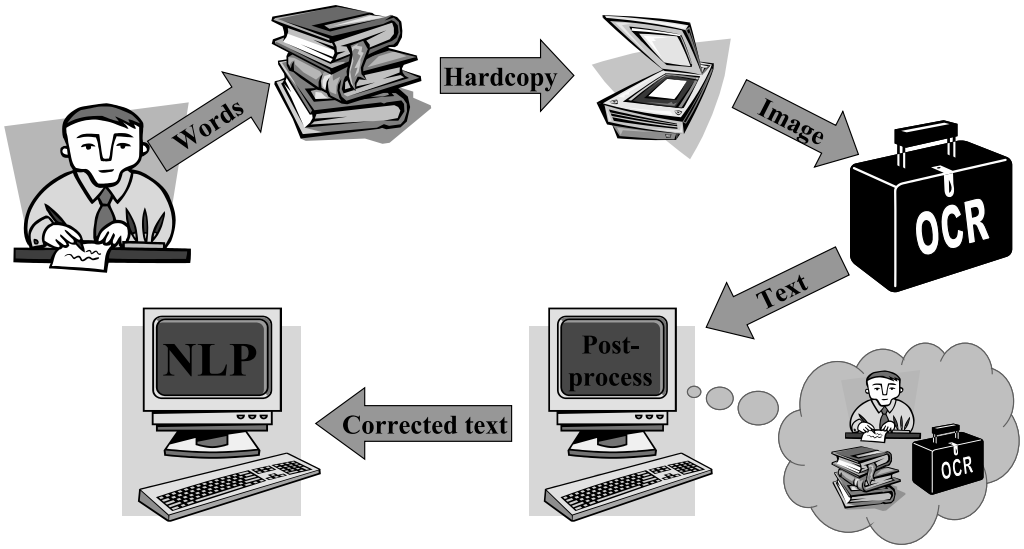

Figure 3.1: Overall OCR pipeline from generation of text to its use.

nition accuracy if an OCR system is available. Working directly on the OCR system itself is not an easy task. Most commercial OCR systems are black-boxes that seldom allow users to modify the systems' internal parameters and/or to re-train the systems on different corpora. Moreover, due to the current state of the OCR technology in general, re-targeting existing OCR systems to new languages, or adapting them to new corpora is difficult. Baird (1999, reported in [35]) comments that the inability to rapidly re-target OCR applications to new languages is "largely due to the monolithic structure of current OCR technology, where language-specific constraints are deeply enmeshed with all the other code." The difficulty of accessing and modifying OCR systems lead us to use a post-processing approach where we treat the OCR system as a complete black-box, and operate only on its output. Figure 3.1 illustrates how we conceptualize the overall process of text generation, OCR, post-processing, and final consumption of electronic text. We use the terms

post-processing and *error correction* interchangeably in this thesis.

We developed a generative probabilistic OCR model that allows post-processing of OCR text, which significantly reduces the OCR error rate. The thought bubble in Figure 3.1 represents the model. The model can be defined as the mental view of the post-processing system regarding the OCR process.[1] The model is effective enough to enable cross-language utilization, allowing the use of an OCR system developed for one language to recognize another language that is written using a similar script. We show that it is possible to achieve OCR performance close to, or even better than, the performance of a native OCR system by utilizing post-processing. We further demonstrate that the same method can be used to improve the accuracy of a native OCR system on its intended target language.

In addition to evaluating the OCR performance using intrinsic recognition quality metrics, we perform extrinsic evaluations, evaluating the impact of OCR errors and their correction on NLP application performance. While OCR errors reduce application performance considerably, error correction brings the performance much closer to the performance obtained using clean text.

3.2 Overview

In this chapter we focus on our OCR post-processing approach and its application to languages for which a moderate amount of resources, such as lexicons, are available. Languages with fewer resources are addressed in Chapter 4. The remainder of

[1]The image step is deliberately omitted, as we ignore image related details in our model.

this chapter is organized as follows: We first present some preliminary methods we investigated for OCR error correction. Next, we present the generative probabilistic OCR model that we have developed, including its FSM-based implementation and evaluation results on real OCR data. Finally, in Section 3.5 we evaluate how NLP applications are affected by OCR errors, and whether post-processing can provide a significant improvement.

3.3 Preliminary Methods Investigated

Before presenting our current correction framework, we briefly discuss some preliminary methods we investigated. The first section below describes the use of a syntactic pattern recognition model for OCR correction. The second section presents a parameter estimation method for character edit operations that is based on SMT models.

3.3.1 OCR Correction as Syntactic Pattern Recognition

Oommen and Kashyap [103] introduce a general framework for syntactic pattern recognition that is well suited for OCR error correction. They provide the following probabilistic generative model, M^*, that characterizes $P(O|C)$ for input and output strings C and O over a finite alphabet A. The authors prove that M^* is stochastically consistent, optimal, and attains the information theoretic upper bound. They also present an efficient dynamic programming algorithm for calculating the probability of transforming one string into another.

We employed their model to perform OCR error correction using a noisy-channel approach on isolated words, as reported in [68]. The correction system used only the probability of transformation, $P(O|C)$, but achieved high correction rates on test data that did not include segmentation errors and did not require case handling. On real OCR data, using a set of mis-recognized words in isolation as the test set, we were able to correct 60 to 70% of the words under various conditions. We conclude that the syntactic pattern recognition-based model is a feasible alternative for implementing the character sequence transformation step in the model presented in Section 3.4.1.

We describe Oommen and Kashyap's model briefly, and illustrate how it transforms a true character sequence, C, to the observed output sequence O.[2] M^* is defined in terms of three probability distributions. The *quantified insertion distribution*, G, controls the number of insertions during the transformation process; in the most general case, it can be conditioned on the input string. The *qualified insertion distribution* defines the probability $Q(a)$, $a \in A$, that symbol '`a`' will be inserted, given that an insertion will be performed. The *substitution and deletion distribution* $S(b|a)$ governs the probability that symbol '`a`' in the input string will be transformed into symbol '`b`' in the output string; deletion is represented as a transformation into a special null symbol, ϵ. Let us define and illustrate the generative process step by step, supposing the input string C is '`hacker`'.

1. Randomly determine z, the number of insertions, using G.

[2]We have modified their notation slightly to make it consistent with ours.

Let us say z is 1 in this case.

2. Insert z instances of the special insertion symbol ξ into randomly selected positions in C, giving C'.

   Let us say we randomly selected position 6, giving us '`hacker`ξ'.

3. Randomly and independently substitute non-ξ symbols in C' using S to get C''.

   Suppose that '`b`' is substituted for '`h`', '`a`' for '`a`', ϵ for '`c`', and '`k`', '`e`', '`r`' for themselves. This gives us '`ba`ϵ`ker`ξ'. Recall that ϵ is the special null symbol that is used to represent deletion.

4. Randomly and independently transform all occurrences of ξ in C'' to symbols in A using Q, giving O'.

   Assume that ξ is transformed into '`y`', giving '`ba`ϵ`kery`' as O'.

5. Remove all occurrences of ϵ from O', giving O.

   We finally get '`bakery`', which is the output string O.

We used Maximum Likelihood Estimation (MLE) to estimate the model parameters using the event counts extracted from the training data. The event counts were extracted using two different methods. The first method used the standard Levenshtein edit distance [75] to compute the minimum cost edit sequence for each training pair. The event counts were then extracted from these edit sequences. The second method treats the problem as translation from truth to OCR, and uses IBM

style SMT models to estimate event counts, as described in Section 3.3.2. We refer to the first estimation method as Edit Distance Estimation (EDE), and the second method as Translation Model Estimation (TME).

3.3.2 Character Sequence Transformation as Translation

It is interesting to note that our parameter estimation problem for the character sequence transformations shares a great deal with the estimation problem for the IBM statistical machine translation models [13, 14].[3] For example, Figure 3.2 shows a word-level alignment representative of the IBM models and a character-level alignment consistent with the model definitions provided in Section 3.4.1. Moreover, efficient parameter estimation algorithms for the IBM models have already been implemented in the GIZA++ software package [101]. This suggests the possibility of using translation models as the basis for parameter estimation in the new setting of OCR error correction.

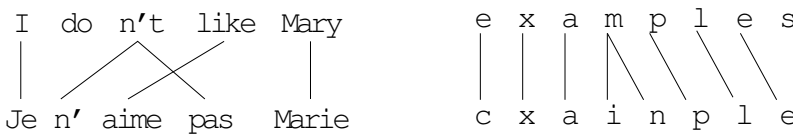

Figure 3.2: Examples of Alignment in Translation and OCR models.

The match between SMT and OCR models is not perfect. For example, symbol re-ordering is crucial for SMT, but not for OCR. The IBM fertility model allows one-to-many symbol alignments (splits) but not many-to-one (merges). Despite their differences, it is possible to use unmodified IBM-style model training on a parallel

[3]For an excellent introduction to the IBM translation models, see [64]

corpus of "sentence" pairs $\langle T, O \rangle$, by treating characters as "words". The trained IBM model parameters can then be used to estimate the event counts to be used for parameter estimation in the pattern recognition approach given in Section 3.3.1.

Estimation of the frequency of insertion counts is performed using the Viterbi alignment between OCR and ground-truth characters. For this approach, the following characters are counted as insertions:

- All the OCR characters that are mapped from NULL
- All but one of multiple characters that are mapped from a single ground-truth character

For instance, if ground-truth symbol '`a`' is mapped to symbols '`p`' and '`q`', we can assume there is one insertion; however, we cannot determine which OCR symbol is the inserted one.[4] Therefore, we use the probability of translating NULL to a particular symbol as an approximation of that symbol's insertion probability. It is only an approximation, since it ignores the insertion cases that do not involve NULL. Specifically, we estimate the insertion count of a symbol '`s`' as follows

$$insertionCount(s) = round\big(translationProb(NULL, s) \times TotalInsertionCount\big)$$

Note that we have three sets of insertion counts:

1. Total insertion count
2. Insertion count for each character

[4] We are ignoring less likely cases such as where '`a`' is deleted and '`p`' and '`q`' are inserted.

3. Insertion count frequencies

These three sets need to be consistent with each other. Informally, the total number of insertions implied by both insertion counts of symbols and frequencies of insertion counts should be consistent with the total insertion count. More formally:

$$\begin{aligned} TotalInsertionCount &= \sum_{i>0} insertionCountFreq(i) \times i \\ &= \sum_{s \in \{symbols\}} insertionCount(s) \end{aligned}$$

where $insertionCountFreq(i)$ is the number of edit sequences with i insertions.

We estimate the total number of insertions and insertion count frequencies together, so that they are consistent. We then round insertion counts for some symbols up or down to make their sum consistent with the total insertion count.

For substitution counts, we use the translation probabilities. However, deletion is considered a special case of substitution, and there is no symbol-to-NULL translation probability; therefore, we use the 0-fertility probability of a symbol to estimate its deletion count. If a symbol '`s`' appears n times in the corpus, then its deletion count is estimated as follows:

$$deletionCount(s) = round\big(symbolCount(s) \times fertilityProb(s, 0)\big)$$

Once having accounted for deleted symbols, we can distribute the remaining counts to substitutions. This means that the substitution count of a symbols '`s`' with a symbol '`t`' is estimated as follows:

$$\begin{aligned} substitutionCount(s,t) = round\big(\big(symbolCount(s) - deletionCount(s)\big) \times \\ translationProb(s,t)\big) \end{aligned}$$

Note that for any symbol, the sum of the deletion and substitution counts should be equal to the symbol count for that symbol as follows:

$$symbolCount(s) = deletionCount(s) \; + \sum_{t \in \{symbols\}} substitutionCount(s,t)$$

This consistency is trivially achieved by setting the symbol count to the number imposed by the sum of the deletion and substitution counts. Also note that we ignore the consistency of our estimated counts and actual counts in the data for simplicity, since it is not crucial for validity of our estimates.

3.3.3 Evaluation

We first performed a "reality check" experiment on artificial data. The goal was to see how well the parameter estimation techniques could recover a set of known parameters by training on a data set generated artificially using those parameters. We created our training data set by using M^* with a fixed set of parameters to generate a noisy version of 30158 common English words with lengths between 7 and 14 characters. We then estimated the values of the parameters using our two training methods, varying the quantity of training data. The Kullback-Leibler (K-L) distance between the parameter estimates and the true parameters that generated the data are given in Figure 3.3 as a function of the amount of training data. There was no significant difference between EDE and TME training methods; both methods perform well when the M^* model is a good characterization of the data.

In our second evaluation scenario, largely because the IBM translation modeling framework made it an easy experiment to perform, we looked at whether or

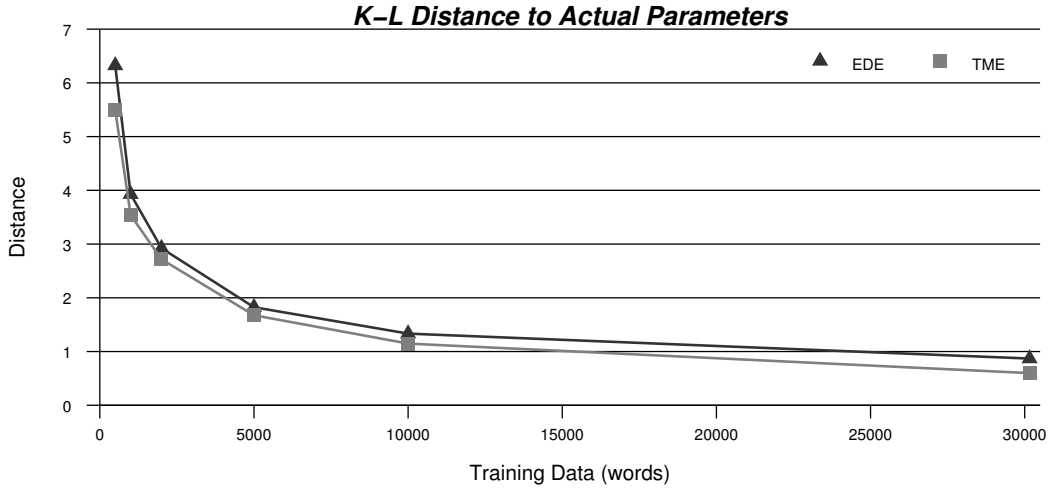

Figure 3.3: K-L Distance between Original and Learned Parameters

not a straightforward statistical MT decoding approach would succeed in recovering an original ground truth string from its OCRed counterpart. Treating characters as "words" in the vocabulary, we used the CMU-Cambridge Toolkit [17] for language modeling, GIZA++ [101] for translation modeling, and the ISI ReWrite Decoder [84] for decoding. This approach performed very poorly, with decoder output on a test set rarely producing the ground truth word, mainly because the translation approach does not capture the fact that a decoded sequence of characters should form a word. Modifying the language model component could solve the problem, but we decided not to pursue this route.

Instead, our third evaluation scenario was to take an incorrectly recognized word o and to select the correct word w from a lexicon L by ranking the choices according to $P_{M^*}(o|w)$. Tokens that are not in the lexicon are identified as incorrect. As is the case with many approaches to error correction, we do not consider valid-

word errors here. We evaluate correction performance using correction rate, which measures the ratio of mis-recognized words o for which the model assigns highest probability to the correct word w.

For the artificial training and test data, where noisy English strings are generated from ground truth English words as described in the first experiment above, correction rate was around 96%. This confirms that both training methods perform quite well in an artificial scenario where the generative process is known to be an instance of the M^* model.

More realistically, we also experimented on real OCR data generated by scanning a hardcopy of the book of Genesis in French using a commercial OCR system.[5] The word error rate for the OCR output was approximately 20%. We used disjoint training and test sets, varying the size of the training data to explore sensitivity to training set size; the test set included 423 mis-recognized words. We experimented with a short lexicon (1,005 words) and a longer lexicon (4,300 words) in order to explore sensitivity to the number of alternatives for w. The number of valid-word errors was 8 for the short lexicon and 17 for the long lexicon.

Table 3.1 presents correction rates for various amounts of training data, two estimation methods, and two lexicon sizes. The baseline is obtained by using the lexicon entry with the lowest Levenshtein edit distance as the correction for each test word. The difference between our methods and the baseline is clear; we nearly double the baseline performance one would obtain in correcting OCR word errors. In addition, TME training consistently performs slightly better than EDE training.

[5]OmniPage 8.0.

Training Size	Long Lexicon		Short Lexicon	
	EDE	TME	EDE	TME
500	55.08	58.39	64.30	65.01
1,000	60.99	61.70	69.27	69.27
2,000	58.39	63.36	69.03	72.10
5,000	59.81	64.54	68.79	71.16
10,000	62.17	63.83	70.21	71.63
21,186	61.70	65.96	69.97	72.34
Baseline	31.91		40.42	

Table 3.1: Preliminary correction accuracy on English text under various conditions.

The difference between TME and EDE is statistically significant when aggregated across all experimental conditions. EDE performs worse most likely because it ignores all but one of the possible edit sequences that could transform an original word to the noisy one, and it might also reflect the fact that the IBM translation models are able to capture character split errors in OCR.

The figures suggest that the size of the candidate lexicon has a significant effect on accuracy, hence one can expect to improve accuracy by using a clever candidate selection mechanism or incorporating evidence from a language model. The results do not improve much with the size of the training data amount after reaching about 2,000 words, which makes practical the creation of manually generated ground truth for training.

3.3.4 Word Mapping with Word Merge/Split Errors

Noisy text sources such as OCR not only garble words, but often change the number of words through omission and insertions, merges and splits. The difference in the number of words in the clean version and the noisy version renders the usual position-based mapping useless. Without the mapping between the words, even determining the number of correct words in the noisy data is a non-trivial task. Obtaining the word mapping between the noisy and clean versions of text is important for generating training pairs for parameter estimation (see Sections 3.3.2, 3.4.2, 4.3.2) and for evaluations involving noisy data (see Section 3.5.1).

Character-based Mapping

One conventional method used for mapping words in clean and noisy text is to compute the character level edit distance operations that transform the clean sequence into the noisy sequence, which provides character level alignment links as a side product. Once the correspondence between characters is determined, pairs of corresponding space characters can be used to break the lines into corresponding chunks. For example, if the true character sequence is '`an important case`' and the OCR sequence is '`unimportant case`', the minimum cost edit sequence that converts truth to OCR is '`scdccccccccccccc`', where s denotes substitution, d deletion, and c copy. Using the beginning, the end, and the space characters that are copied over as anchor points, we can infer the following mapping

```
an important ↦ unimportant
case         ↦ case
```

which is correct. While this method can capture word merge/split errors successfully in general, it is not as successful when it comes to word insertions/deletions. For example, if the true sequence is '`that is wrong`' and the OCR sequence is '`that wrong`', one possible character edit sequence is '`ccccdddcccccc`', which would result in the following mapping

`that is` $\mapsto$ `that`

`wrong` $\mapsto$ `wrong`

which is unlikely to be correct. The desired mapping would be

`that` $\mapsto$ `that`

`is` $\mapsto \epsilon$

`wrong` $\mapsto$ `wrong`

While there are other possible edit sequences as well, they all result in incorrect mappings since the space in the OCR string will map to one of the two spaces in the true sequence; the space will never map to both, which is the only way the deletion error can be captured.

Word-based Mapping

One way to address this problem is to focus on the fact that '`that`' and '`wrong`' both have an exact match on the OCR side. However, this requires looking at the problem at the token level rather than the word level. Therefore, we designed an algorithm to compute the mapping between the noisy and clean text at the word level. The algorithm assumes that many words in the noisy text are actually correct, and uses these words as anchor points for mapping all words to the corresponding words in the clean text.

Correct words are identified by finding a sequence of word edit operations that convert the clean text into the noisy version with the minimum cost; the word operations include the following: copy, insertion, deletion, and substitution. Copied words in this edit sequence are assumed to be correct (and momentarily we will generalize this to non-identical but similar words). Owing to the space requirements of the edit distance algorithm, we assume noisy and clean text files are aligned at line or page level, and work on one line/page pair at a time.

Let us present the algorithm using a sample execution where the original line is '`mapping words is not easy`' and the noisy line is '`mopping words lot easy now`'.

- Find an edit sequence that has the minimum cost. In case of equality, the sequence with more copy operations is favored.

  ```
  Substitution :  mapping ↦ mopping
  Copy         :  words   ↦ words
  Deletion     :  is      ↦ ϵ
  Substitution :  not     ↦ lot
  Copy         :  easy    ↦ easy
  Insertion    :  ϵ       ↦ now
  ```

- For each copy operation, create a bracketed component on both sides that contains only the copied word.

  ```
  mapping [words] is not [easy]
  mopping [words] lot [easy] now
  ```

- Perform the following for both the original and noisy lines:

  - For each existing bracketed component, create a new bracketed component that contains everything to its right up to the next bracket, or up to the end of the line if there is no next bracket.

    ```
    mapping [words] [is not] [easy] []
    mopping [words] [lot] [easy] [now]
    ```

  - Create a bracketed component that contains everything from the beginning of the line to the first existing bracket, or to the end of the line if there is none.

    ```
    [mapping] [words] [is not] [easy] []
    [mopping] [words] [lot] [easy] [now]
    ```

- For each bracketed component in the original line, pair its contents with the corresponding (left-to-right) component in the noisy line, ignoring empty bracket pairs.

  ```
  mapping ↦ mopping
  words   ↦ words
  is not  ↦ lot
  easy    ↦ easy
  ϵ       ↦ now
  ```

Since the mapping is based on the edit sequence generated by the edit distance algorithm, one can tweak it by modifying the edit distance algorithm or its parameters. One such useful modification is to use a similarity threshold instead of exact match to decide if two words match. For example, let '`to illustrate this`' be the original line and '`lo ilustrate ths`' be the noisy line. With exact matching, we would get the following edit sequence

`Substitution : to ↦ lo`

`Substitution : illustrate ↦ ilustrate`

`Substitution : this ↦ ths`

and the mapping '`to illustrate this ↦ lo ilustrate ths`', where the whole string is mapped over as a single chunk. However, if we assume that two words match if their edit distance is less than 20% of the length of the longer one, the edit sequence would become

`Substitution : to ↦ lo`

`Copy` `: illustrate ↦ ilustrate`

`Substitution : this ↦ ths`

giving the more desirable mapping '`to ↦ lo`', '`illustrate ↦ ilustrate`', and '`this ↦ ths`'. Another useful modification is to change the cost of the copy operation. Assume the original line is '`a b c`' and the noisy line is '`c d e`'. With 0 copy costs, we would get the following edit sequence:

`Substitution : a ↦ c`

`Substitution : b ↦ d`

`Substitution : c ↦ e`

and the corresponding mapping would be '`a b c ↦ c d e`'. If we use -1 as the copy cost, the edit sequence would become

`Deletion : a ↦` ϵ

`Deletion : b ↦` ϵ

`Copy : c ↦ c`

`Insertion :` ϵ `↦ d`

`Insertion :` ϵ `↦ e`

and the mapping would be '`a b ↦` ϵ', '`c ↦ c`', and 'ϵ `↦ d e`'. By changing the cost of the operations, one can tune how far an original word and the corresponding word on the noisy side can be from each other.

Combining Word-based and Character-based Mapping

While the word-based mapping algorithm is able to handle word insertions and deletions that are not properly handled by the character-based mapping method, it has its own shortcomings. Since we use correctly recognized words as anchor points, runs of mis-recognized words are bundled together as a single unit on the clean side and mapped to a corresponding set of words on the noisy side. While the mapping is not necessarily incorrect, it is definitely too coarse. Although using a fuzzy match of words rather than exact match as discussed above addresses the problem to some extent, it is not easy to find a similarity threshold that works well across varying levels of accuracy. The character-based mapping process can provide a reasonable solution to this problem. We first handle the mapping of exact matches using the word-based algorithm, and reduce the mapping problem to the runs of non-matching tokens. We then use the character-based mapping method on these non-matching runs to obtain a finer grained mapping for them. This approach is best illustrated in an example. Assume the true sequence is '`Mapping words is not easy`' and the noisy sequence is '`Chopping wood is easy`'. Character-based mapping would return the following mapping

```
Mapping ↦ Chopping
words   ↦ wood
is not  ↦ is
easy    ↦ easy
```

which fails to capture that '`is`' is recognized correctly and '`not`' is deleted. Word-based mapping, on the other hand, would return the following mapping:

`Mapping words` $\mapsto$ `Chopping wood`

`is` $\mapsto$ `is`

`not` $\mapsto \epsilon$

`easy` $\mapsto$ `easy`

The word-based mapping handles the deletion correctly. However, the mapping between '`Mapping words`' and '`Chopping wood`' is a problem. Although it can be argued that it is correct, it definitely is not as fine grained as possible. The combined mapping method solves this problem by using the character-based mapping on all pairs that contain multiple tokens on both sides. For our example, only the first pair fits this description, so we run the character-based mapping between '`Mapping words`' and '`Chopping wood`', which returns the following mapping:

`Mapping` $\mapsto$ `Chopping`

`words` $\mapsto$ `wood`

Then we replace the original sub-mapping in the word-based map with the sub-mapping obtained from the character-based map, resulting in the final, correct mapping:

`Mapping` $\mapsto$ `Chopping`

`words` $\mapsto$ `wood`

`is` $\mapsto$ `is`

`not` $\mapsto \epsilon$

`easy` $\mapsto$ `easy`

While the combined method is able to handle some mapping cases that cannot be handled by the character-based and word-based methods, it still cannot handle all possible mappings. For example, if the true sequence is '`test of the`' and the OCR sequence is '`test .  at the`', the combined method would return the following mapping:

```
test ↦ test
of   ↦ .  at
the  ↦ the
```

This mapping is not correct. A mapping where '`of`' and '`at`' are paired together and '`.`' is marked as an insertion would be preferable. Although the combined mapping is not perfect, we observed only a handful of cases where it failed during our experiments. Many of the failed cases were inherently ambiguous. Therefore we did not pursue the mapping problem any further.

3.4 A Generative Probabilistic OCR Model

Having established that post-processing can significantly improve OCR output quality, we next developed a more sophisticated post-processing system that makes use of a generative probabilistic model specifically designed for OCR. The following sections present the correction framework, its implementation, and evaluation results on real OCR data.

3.4.1 Correction Framework

Our approach to the correction problem uses the noisy-channel framework [125] that has been employed successfully for many similar tasks, including spelling and OCR correction [e.g. 12, 16, 68]. Our work is similar to previous work in the area in terms of its use of the noisy-channel model. However, our work introduces a novel end-to-end model that describes the OCR process. None of the existing OCR models, to the best of our knowledge, model the complete OCR process end-to-end in the way

our model does.

Generative "noisy channel" models relate an observable sequence to an underlying source sequence. In this particular case, the observable sequence is a recognized character string O, and underlying source sequence is the corresponding word sequence W. This relationship is modeled by $P(W, O)$, decomposed by Bayes' Rule into steps modeled by $P(W)$ (the source model) and $P(O|W)$ (comprising sub-steps generating O from W). Each step and sub-step is completely modular, so one can flexibly make use of existing sub-models or devise new ones as necessary. Note that the process of "generating" O from W is a mathematical abstraction, not necessarily related to the operation of any particular OCR system.

Definitions

We begin with preliminary definitions and notation. Some of the concepts that will be defined are illustrated in Figure 3.4.

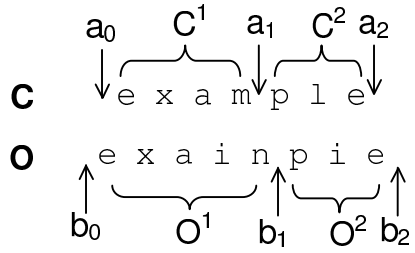

Figure 3.4: Illustration of some of the concepts from the OCR model.

Character and Word Sequences

- True word sequence: $W = \langle W_1, \ldots, W_r \rangle$

- True character sequence: $C = \langle C_1, \ldots, C_n \rangle$
- OCRed character sequence: $O = \langle O_1, \ldots, O_m \rangle$
- "$C_1 \ldots C_n$" is a shorthand notation for the character sequence $\langle C_1, \ldots, C_n \rangle$.

Segmentation

- Segmentation of a true character sequence into p subsequences $a = \langle a_0, \ldots, a_p \rangle$, where $a_i < a_{i+1}$, $a_0 = 0$, and $a_p = n$.
  - a defines character subsequences $C^i = (C_{a_{i-1}}, \ldots, C_{a_i}]$
  - The number of segments p need not equal the number of words r and C^i need not be a word in W.
- Segmentation of OCR character sequence into q subsequences $b = \langle b_0, \ldots, b_q \rangle$, where $b_j < b_{j+1}$, $b_0 = 0$, and $b_q = m$.
  - b defines character subsequences $O^j = (O_{b_{j-1}} \ldots O_{b_j}]$.
- The alignment chunks are the pairs of corresponding truth and OCR subsequences: $\langle O^i, C^i \rangle$, $i = 1, \ldots, p$. If necessary, the model can be modified so that $p \neq q$.

Character-Level Alignment and Confidence Values

- Character-level alignment between segments C^i and O^i is represented by ϕ^i.
  - ϕ^i is a vector of indices where ϕ^i_κ is the index of the true character in C^i that is aligned to the OCR character with index κ in O^i, i.e. $O^i_\kappa \mapsto C^i_{\phi^i_\kappa}$.

– κ ranges from 1 to the size of O^i, $1 \leq \kappa \leq |O^i|$.

The character-level alignments directly follow from the transformation of the original sequence to the OCR sequence. However, given a pair of true and OCR character sequences, the actual transformation is a hidden variable which is not directly available. The character-level alignments can be provided by any of a variety of modeling procedures for sequence alignment, such as HMMs. The specific selection of an alignment procedure is not crucial at this point in the model formulation. We apply the sequence alignment models independently to the hypothesized alignment chunks. The model formulation incorporates these individual character-level alignments into a framework that allows integration of higher level models, such as dictionaries, language models, and translation models.

The character transformation models that we used in our implementations are all variations of string edit distance. Consequently, the character level alignment is produced implicitly during the computation of minimum cost transformation.

- Confidence values provided by the OCR system for the characters in the sequence O^i is a vector of numbers V^i.

  – V^i_κ is the confidence of the OCR system in the OCR character at index κ of the sequence O^i.

  – κ ranges from 1 to the size of O^i, $1 \leq \kappa \leq |O^i|$.

– The actual nature of a confidence value depends on the OCR system used. Since the confidence value is likely to be a real number, quantization will be necessary. While we make necessary provisions to make use of confidence values, they are unlikely to be available for most commercial systems. Therefore, we have not used them in our evaluations.

The Model

We designed a generative model that transforms the ground-truth sentence into an OCR sentence through several steps. Note that this is meant only to be a plausible conceptual description of the OCR process from the point of view of word error correction. The goal is not to provide an exact description of all the steps that are actually performed in OCR, but rather to build a generative model that can be robustly estimated and readily applied to OCR word error correction. Figure 3.5 provides a visual overview of the model. We ignore the imaging step represented by the scanner, other steps are covered in detail in the sections that follow.

Generation of True Word Sequence The generative process begins with production of the true word sequence W with probability $P(W)$. Modeling the underlying sequence at the word-level facilitates integration with NLP models, which is our ultimate goal. For example, the distribution $P(W)$ can be defined using n-grams, parse structure, or any other tool in the language modeling arsenal. We will use a running example to illustrate the model. Let us start with the true word sequence $W = \langle this, is, an, example, . \rangle$

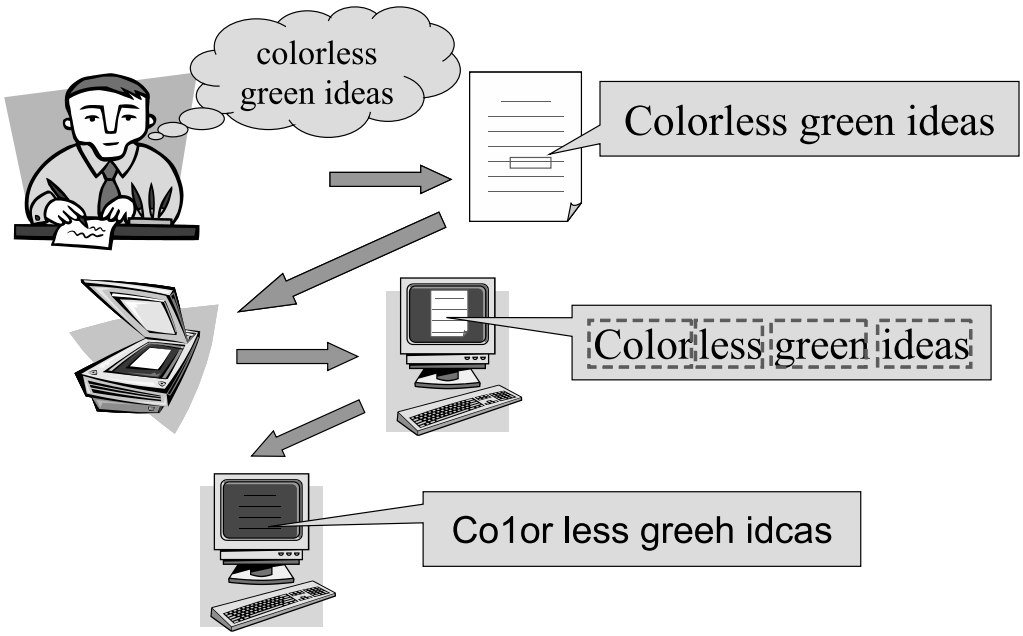

Figure 3.5: Overview of our generative OCR model.

From Words to Characters The first step in transforming W to O is the generation of character sequence C, modeled as $P(C|W)$. This step accommodates the character-based nature of OCR systems, and provides a place to model the mapping of different character sequences to the same word sequence due to case (and perhaps font) variations or vice versa (e.g. ambiguous word segmentation in Chinese, where a character sequence can map to multiple word sequences). If the language in question provides explicit word boundaries (e.g. words are separated by spaces when printed) then we output '#' to represent visible word boundaries.

One possible C we can generate from W for our running example is $C =$ '`This#is#an#example.`'. In this example, there is no # between '`example`' and '`.`' since that boundary is not written out in text. The ability to hypothesize word boundaries is important, since the existence of an adjacent word boundary (e.g. a

space) can affect the recognition of characters by the OCR system. It may also be helpful to introduce a marker to indicate word boundaries that are not visible.

We can simply use this step for case normalization, etc. and ignore the actual probability distribution. Alternatively, we can use more sophisticated models of orthography. For example, a more sophisticated model of orthography could be trained to anticipate an uppercase initial if the word being converted to a character sequence is a proper name.

Segmentation Subsequences C^i are generated from C by choosing a set of boundary positions, a. This step, modeled by $P(a|C, W)$, is motivated by the fact that most OCR systems first perform image segmentation, and then perform recognition on a word by word basis. Note that an OCR system sees the image of the character sequence C, not the actual words W. So it is probably safe to omit the dependency on W for practical purposes if necessary.

For a language with clear word boundaries (or reliable tokenization or segmentation algorithms), one could simply use spaces to segment the character sequence in a non-probabilistic way. However, OCR systems might make segmentation errors that result in word merge/split errors. Therefore, a probabilistic segmentation model that accommodates word merge/split errors is necessary. A merge error occurs when two or more adjacent items are recognized as one, and a split error occurs when an item is recognized as two or more items. While these errors can happen both at word level and character level, our focus in this step is on word level errors; character level errors are handled by the character sequence transformation step.

If a segment boundary coincides with a word boundary, the word boundary marker '#' is considered a part of the segment on both sides. The rationale behind this decision is the fact that the OCR system is expected to segment words at white-spaces, and the space does not really belong to a particular segment on either side.

One possible segmentation for our example is $a = \langle 0, 8, 11, 13, 19 \rangle$, which results in following segments:

C^1 = 'This#is#'

C^2 = '#an#'

C^3 = '#ex'

C^4 = 'ample.'

Note that there is a merge error in segment 1, and a split error involving segments 3 and 4.

Character Sequence Transformation Our characterization of the final step, transformation into an observed character sequence, is motivated by the need to model OCR systems' character-level recognition errors. We model each subsequence C^i as being transformed into an OCR subsequence O^i, so

$$P(O, b|a, C, W) = P(\langle O^1, \ldots, O^q \rangle | a, C, W).$$

and we assume each C^i is transformed independently, allowing

$$P(\langle O^1, \ldots, O^q \rangle | a, C, W) \approx \prod_{i=1}^{p} P(O^i|C^i).$$

Any character-level string error model can be used to define $P(O^i|C^i)$; for example Ristad and Yianilos [118], Brill and Moore [12], or Kolak and Resnik [68].

This is also a logical place to make use of confidence values if provided by the OCR system, as discussed in the following section. We assume that # is always deleted, and can never be inserted because each subsequence generated by the segmentation model is considered to be a single token and cannot include spaces. This allows modeling word merge errors by placing merged words into the same subsequence. Boundary markers at segment boundaries are re-inserted when segments are put together to create O, since they will be part of the OCR output (not as #, but most likely as spaces). For our example C^i, a possible result for this step is:

O^1 = 'Tlmsis'

O^2 = 'an'

O^3 = 'cx'

O^4 = 'amp1e.'

The final generated string would therefore be O = 'Tlmsis#an#cx#am1e.'. Segmentation of OCR character sequence O, $b = \langle 0, 7, 10, 13 \rangle$, directly follows from subsequences O^i. b is required since we need the boundaries of segments for correction, and a will not be available during correction phase.

Assuming independence of the individual steps, the complete model estimates joint probability $P(O, b, a, C, W)$, and $P(O, W)$ can be computed by summing over all possible b, a, C that can transform W to O.

$$P(O, b, a, C, W) = P(O, b|a, C, W)P(a|C, W)P(C|W)P(W)$$

$$P(O, W) = \sum_{b,a,C} P(O, b, a, C, W).$$

OCR Confidence Values Some OCR systems provide a confidence value along with every character they output. This is valuable information that may prove help-

ful if properly used. However, since confidence values are not available in general, we will simply describe various ways they can be incorporated in the model, without going into further details.

It is not clear how this information source should be integrated. One can re-write the sequence conversion formula as follows to have access to links between true and OCR characters

$$P(O^i|C^i) = \sum_{\phi^i} \prod_{\kappa} P(O_{\phi^i{}_\kappa}|C^i_\kappa, \phi^i)P(\phi^i)$$

where ϕ^i is an alignment between true characters and OCR characters. The κth character of the truth sequence is aligned with, hence assumed to produce, the ϕ^i_κth character of the OCR sequence. This formulation assumes the characters are independently transformed, and since the actual alignment is hidden, sums over all possible alignments.[6] Once we have this formulation, we can introduce the confidence values into the mix in several ways.

One possibility is to shift the probability distribution based on the confidence value. We can train different error models for different confidence intervals. We re-write the formula as follows:

$$P(O^i, V^i|C^i) = \sum_{\phi^i} \prod_{\kappa} P(O^i_{\phi^i{}_\kappa}|C^i_\kappa, V^i_{\phi^i_\kappa}, \phi^i)P(V^i_{\phi^i_\kappa}|C^i_\kappa, \phi^i)P(\phi^i)$$

Another option is to have a probability distribution for the confidence value itself. Given enough training data, we can learn the probability of having a certain

[6]Dependence on neighboring characters can easily be introduced.

confidence value when provided with the original and OCR characters as follows:

$$P(O^i, V^i|C^i) = \sum_{\phi^i} \prod_{\kappa} P(V^i_{\phi^i{}_\kappa}|O^i_{\phi^i_\kappa}, C^i_\kappa, \phi^i) P(O^i_{\phi^i_\kappa}|C^i_\kappa, \phi^i) P(\phi^i).$$

It is worth mentioning that the confidence values are usually real numbers, hence they cannot be directly used in our probability formulations. A proper form of quantization, depending on the properties of the particular set of confidence values in use, would be required.

OCR Correction using the Model

The model being presented is a generative model that gives the joint probability $P(O, T)$, and cannot be directly used to correct OCR errors. It is used in the noisy channel framework to predict the most likely input sequence, given the observed output. This intuitively means searching through all possible input sequences, plugging them into the model formula, and picking the sequence that maximizes the probability. See Section 3.4.2 for an efficient way to perform this search.

More formally, given the output of the OCR system and its segmentation, $(\hat{O}, \hat{b})$, the underlying true word sequence would be determined by the following formula:

$$\hat{W} = \underset{W}{\text{argmax}}[\max_{a,C} P(\hat{O}, \hat{b}|a, C, W) P(a|C, W) P(C|W) P(W)]$$

When the segmentation of the OCR output is not available (e.g. it is in a language without printed word boundaries, and there is no reliable segmenter), the

formulation would be changed as follows to account for possible values of b:

$$\hat{W} = \underset{W}{\text{argmax}}[\max_{b,a,C} P(b|\hat{O})P(\hat{O}, b|a, C, W)P(a|C, W)P(C|W)P(W)]$$

3.4.2 Implementation of the Model as a Weighted FSM

We have implemented our generative model using a Weighted Finite State Machine (WFSM) framework, which provides a strong theoretical foundation, ease of integration for different components, and reduced implementation time thanks to available toolkits such as the AT&T FSM Toolkit [94]. It also allows easy integration of our post-processor with numerous NLP applications that are implemented using FSMs [e.g. 2, 65, 73, 120].

Each step is represented and trained as a separate FSM, and the resulting FSMs are then composed together to create a single FSM that encodes the whole model. Sections that follows provide the details of parameter estimation and decoding.

Parameter Estimation

The specific model definition and estimation methods assume that a training corpus is available, containing $\langle O, C, W \rangle$ triples.

Generation of True Word Sequence Various language-modeling tools can be used for estimating the probability distribution for the true word sequence. Some examples are probabilistic parse structures, n-grams on characters, words, or parts-of-speech. We currently use character or word-level n-gram language models as the

source model for the original word sequence. We use the following configuration: an open vocabulary, 3-gram language model with back-off generated using CMU-Cambridge Toolkit v2 [17]. The model is trained on the W from the training data using the type 1 option for vocabulary and the Witten-Bell discounting option for smoothing; all other parameters were left at their default values. Vocabulary type 1 means that out-of-vocabulary words are allowed. During training any words that are not in the vocabulary are mapped to a a single special token for unknowns, which in turn is treated as a regular token to estimate its probability distribution.

From Words to Characters This is a step that has explicit language dependencies, since languages differ significantly in the way they are written.[7] Currently, we assume a Latin alphabet and only consider case variations of a word when it is written out. We generate three different character sequence variants for each word: upper case, lower case, and leading case (e.g. '`this` $\Rightarrow$ {`THIS, this, This`}'). For each word, the distribution over case variations is learned from the $\langle W, C \rangle$ pairs in the training corpus.

For words that do not appear in the corpus, or do not occur frequently enough to allow a reliable estimation, we back off to word-independent case variant probabilities. Mixed case text is not included since it increases the number of alternatives drastically. Mixed-case words in input data are normalized as a preprocessing step.

[7] Other steps also have some dependencies on the particular language being modeled. However, those dependencies can be captured by the parameters of the model, without changing the model itself; whereas this particular step is not able to capture all language specific variations without explicitly handling them. For instance, the model needs to be aware of three different alphabets (Hiragana, Katakana, and Kanji) used for writing Japanese.

Segmentation Our current implementation makes an independent decision for each character pair, in terms of inserting a boundary between them. To reduce the search space associated with the model, we limit the number of boundary insertions to one per word, allowing at most two-way word-level splits. The probability of inserting a segment boundary between two characters is conditioned on the character pair and is estimated from the training corpus, using Witten-Bell discounting [143] to handle unseen character pairs.

We need the segmentation a of C for parameter estimation. However, a is a hidden variable and needs to be extracted. We use the alignment chunks of C and O, as defined in Section 3.4.1, and the segmentation b of O to extract a. Segmentation of the OCR side follows directly from the character sequence, since the model ensures each segment generates a single output token.[8] Alignment chunks, on the other hand, are themselves hidden and need to be extracted. We developed a method to extract alignment chunks, which is described in Section 3.3.4.

Character Sequence Transformation This step is implemented as a probabilistic string edit process, based on [118]. The confusion tables for edit operations are estimated using Viterbi style training on $\langle O, C \rangle$ pairs in training data. Training is done in an iterative fashion using the EM approach [23] until parameter values converge. At each iteration, we use the current Finite State Transducer (FST) that encodes edit operation probabilities to find the best path that generates the OCR output from the underlying ground truth. We then update the probabilities for

[8]For languages without word boundaries, segmentation would be required.

the operations based on their usage frequency. Our current implementation allows for substitution, deletion, and insertion errors, and does not use context characters.[9] Figure 3.6 shows a fragment of a weighted FSM model for $P(O^i|C^i)$: it shows how the observed O^i '`haner`' could be generated by the underlying C^i '`banker`' or '`hacker`'.[10]

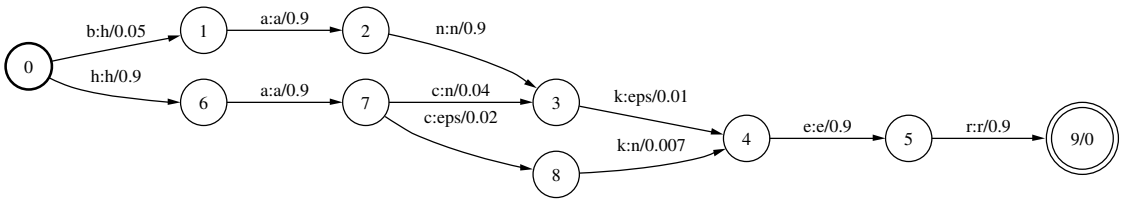

Figure 3.6: Fragment of an FST for $P(O^i|C^i)$.

It is also possible to use other probabilistic string edit models to implement this step. One alternative based on a syntactic pattern recognition framework is presented in Section 3.3.1. Another alternative that we incorporated is a modified version of the spelling correction model presented by Brill and Moore [12]. We also described an alternative parameter estimation method based on statistical translation models in Section 3.3.2.

Final Cleanup At this stage, special symbols that were inserted into the character sequence are removed and the final output sequence is formed. For instance, segment boundary symbols are removed or replaced with spaces depending on the language.

[9]Conditioning on neighbor characters, and using character merge/split errors are trivial conceptually, however practical constraints such as the FSM size make the problem more challenging.

[10]The probabilities are constructed for illustration, but are realistic. Notice how '`n`' is much more likely to be confused for '`c`' than '`k`' is.

Decoding

Decoding is the process of finding the "best" W for an observed $(\hat{O}, \hat{b})$, as discussed in Section 3.4.1. It involves a search through the space of possible correct input sequences to pick the one that maximizes joint probability $P(O, W)$.

Decoding within the FSM framework is straightforward: we first compose all the components of the model in order, and then invert the resulting FST. This produces a single transducer that takes a sequence of OCR characters as input, and returns all possible sequences of truth words as output, along with their weights. One can then simply encode OCR character sequences as FSMs and compose them with the model transducer to perform decoding. The resulting lattice or N-best list is easily integrated with other probabilistic models over words, or the most probable sequence can be used as the output of the post-processing system.

Since the same output sequence can be generated through multiple paths, we need to sum over all paths to find the overall probability of that sequence. This can be achieved by determinizing the output FSM generated by the decoding process. Figure 3.7 shows the determinized version of the error model fragment in Figure 3.6. Note that the most probable path through the lattice in the determinized version gives '`hacker`' while the most probable path in the original version is '`banker`'. Although some of the individual transformation information is lost, it is not a concern since we are interested in input, output, and their probability. However, for practical reasons, we first find the N-best paths in the resulting FSM and then combine the paths that generate the same output; instead of determinizing the FSM.

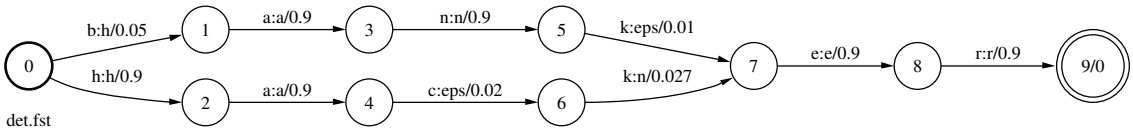

Figure 3.7: Determinized version of the FSM in Figure 3.6

3.4.3 Experimental Evaluation

We have evaluated the performance of our post-processing system on real OCR data in French and Spanish. French OCR data was generated using an English OCR system to simulate a low-density language scenario. A native OCR system was used for Spanish experiments to evaluate the performance of post-processing on native OCR output.

Two evaluation metrics widely used for OCR evaluation are Character Error Rate (CER) and WER, which are defined as follows

$$\text{CER}(C, O) = \frac{CharacterEditDistance(C, O)}{|C|}$$

$$\text{WER}(W_{truth}, W_{OCR}) = \frac{WordEditDistance(W_{truth}, W_{OCR})}{|W_{truth}|}$$

where $CharacterEditDistance$ is the traditional Levenshtein string edit distance [75], and $WordEditDistance$ is a simple variation where the unit of operation is words rather than characters. We only use the WER metric since for almost all NLP applications the unit of information is words. While a word with a single character error is usually preferable to a word with multiple errors for a human reader, they are equally incorrect as far as most NLP applications are concerned.

French Evaluation

Although most researchers are interested in improving the results of OCR on degraded documents, we are primarily interested in improving OCR in new languages for use in NLP. A possible approach to re-targeting OCR for a new language is to employ an existing OCR system from a "nearby" language, and then to apply our error correction framework, as discussed in Section 3.1. We used the output of an English OCR system run on French input to simulate the situation where available resources of one language are used to acquire resources in a similar language.

Training and Test Data We created our French OCR data by scanning a hard-copy Bible using both an English and a French OCR system. (See Kanungo et al. [63] and Resnik et al. [115] for discussion of the Bible as a resource for multilingual OCR and NLP). It was necessary to pre-process the data in order to eliminate the differences, such as footnotes, glossary, cross-references, page numbers, between the on-line version that we used as the ground-truth and the hard-copy. We have not corrected hyphenations, case differences, etc.

We divided the data, which had about 30,000 lines, into 10 equal sized disjoint sets. We used the first 9 as the training data and the first 500 lines of the last set as the test data. Each line contains a complete verse, so they may actually span several lines on a physical page.

Since we are interested in recovering the original word sequence rather than the character sequence, evaluations are performed on lowercased and tokenized data. Note, however, that our system works on the original case OCR data, and generates

a sequence of word IDs, that are converted to a lowercase character sequence for evaluation.

We used a word-level n-gram language model that was trained only on training data. However, we made a closed vocabulary assumption to evaluate the effectiveness of our model when all the correct words were in its lexicon. Therefore, although the language model was trained on only the training data, the words in the test set were included in the language model FSM and treated as unseen vocabulary.

Reduction of OCR Error Rates The WER for the English OCR system on the French test data is 18.31%. The error rate for the output generated by the French OCR system on the same input is 5.98%. When single characters and non-alphabetical tokens are ignored, the WER drops to 17.21% for the English OCR system and 4.96% for the French OCR system.

We evaluate the performance of our model by studying the reduction in WER after correction. The input to the system is original case, tokenized OCR output, and the output of the system is a sequence of word IDs that are converted to lowercase character sequences for evaluation.

The results are summarized in Table 3.2. The conditions side gives various parameters for each experiment. The Language Model (LM) is either 1-gram or 3-gram (word level). Word to Character Conversion (WC) either allows the three case variations mentioned earlier, or simply picks the most probable variant for each word. Segmentation (SG) can be disabled or two-way splits and merges can be allowed. Finally, the character-level Error Model (EM) may be trained on various

subsets of training data.[11] The differences between the original English OCR output and all the corrected results are statistically significant. The difference between the first and last correction rows (WER 7.41% versus 6.75%) are significant, the other pairwise comparisons are not. We use a two-sided z-test and a 95% confidence level to decide significance for all of our OCR experiments.

Conditions				Results	
LM	WC	SG	EM	WER (%)	Red. (%)
English OCR Output				18.31	-
1-gram	3 options	None	Sect. 9	7.41	59.53
1-gram	3 options	None	Sect. 1-9	7.12	61.11
1-gram	3 options	None	Sect. 5-9	7.11	61.17
1-gram	3 options	None	Sect. 5-9	7.06	61.44
3-Gram	**Best case**	**2 way**	**Sect. 5-9**	**6.75**	**63.13**
French OCR Output				5.98	-

Table 3.2: Post-correction WER and reduction rates under various conditions for English OCR output on French text.

Table 3.3 gives the adjusted results when ignoring all single characters and tokens that do not contain any alphabetical characters. The statistical significance of results is identical to those for Table 3.2. the distribution of words in post-correction text is given in Table 3.4. Distribution counts are based on the tokens in the clean text. We compare each token in the clean text to the corresponding token in OCR output and in corrected text and classify it accordingly; *corrected* words are the words that were incorrect in the original OCR output and corrected, *in-corrected* ones were correctly recognized but replaced with an incorrect word, *mis-corrected*

[11] Other sub-models are always trained on all 9 training sections.

words were incorrect and replaced with another incorrect word, and *non-corrected* words are mis-recognized words that are not modified. The remainder of the words were correct both before and after post-processing; their counts are not given in the table. Since we used a closed vocabulary for the experiments, and did not perform valid-word error correction, a correct word is never replaced with an incorrect one. Consequently, the mis-corrected column has no entries.

Conditions				Results	
LM	WC	SG	EM	WER (%)	Red. (%)
English OCR Output				17.21	-
1-gram	3 options	None	Sect. 9	3.97	76.93
1-Gram	3 options	None	Sect. 1-9	3.62	78.97
1-Gram	3 options	None	Sect. 5-9	3.61	79.02
3-Gram	3 options	None	Sect. 5-9	3.52	79.55
3-Gram	**Best case**	**2 way**	**Sect. 5-9**	**3.15**	**81.70**
French OCR Output				4.96	-

Table 3.3: Post-correction WER and reduction rates, ignoring single characters and non-alphabetical tokens, under various conditions for English OCR on French text.

Conditions				Distribution			
LM	WC	SG	EM	Corr.	Incor.	Miscor.	Noncor.
1-Gram	3 options	None	Sect. 9	1446	-	216	185
1-Gram	3 options	None	Sect. 1-9	1483	-	179	185
1-Gram	3 options	None	Sect. 5-9	1485	-	177	185
3-Gram	3 options	None	Sect. 5-9	1493	-	169	185
3-Gram	Best case	2 way	Sect. 5-9	1537	-	127	183

Table 3.4: Distribution of words in post-correction French text, ignoring single characters and non-alphabetical tokens.

As can be seen from the tables, as we increase the training size of the character error model from one section to five sections, the performance increases. Although there is a slight decrease in performance when the training size is increased to 9 sections, it is caused by only two words that are mis-corrected, as can be seen Table 3.4. When we replace the 1-gram language model with a 3-gram model, the results improve as expected. Best results are obtained when word merge/split errors are allowed, as these errors remained uncorrected for other experiments.

Word merge/split errors cause an exponential increase in the search space. If there are n words that need to be corrected together, they can be grouped in 2^{n-1} different ways, ranging from n distinct tokens to a single token. For each of those groups, there are $N^{m \cdot k}$ possible correct word sequences where m is the number of tokens in that group, k is the maximum number of words that can merge together, and N is the vocabulary size. Although it is possible to avoid some computation using dynamic programming, doing so requires some deviation from the FSM framework.

We instead use several restrictions to reduce the search space. First, we allow only two-way merge errors and split errors, restricting the search space to bigrams. We further reduce the search space by searching through only the bigrams that are seen in the training data. We also introduce character error limits, letting us eliminate candidates based on their length. For instance, if we try to correct a sequence of 10 characters and have set an error limit of 20%, we only need to check candidates whose length is between 8 and 12, since all other lengths require at least three deletions/insertions, and thereby exceeding the error limit. The last

restriction we impose is to force selection of the most likely case for each word rather than allowing all three case variations. Despite all these limitations, the ability to handle word merge/split errors improves performance.

It is notable that our model allows global interactions between the distinct components. As an example, if the input is '`ter- re`', the system returns '`mer se`' as the most probable correction. When '`la ter- re`' is given as the input, interaction between the language model, segmentation model, and the character error model chooses the correct sequence '`la terre`'. In this example, the language model overcomes the preference of the segmentation model to insert word boundaries at white-spaces.

Spanish Evaluation

We performed our next set of experiments on Spanish. We use a native Spanish OCR system for this set of experiments to evaluate the correction performance when the main cause of recognition errors is low-quality input images rather than a non-native OCR system. We also experiment with using the many-to-many character edit distance model of [12] for the character sequence transformation step.

Training and Test Data We performed two sets of experiments for Spanish, using training data from United Nations (UN) corpus [43] and Foreign Broadcast Information Service (FBIS) corpus. We used 105 Spanish test sentences that were used for Spanish MT evaluations in Section 3.5.3. Both training and test data were available in electronic form. We need OCR data for training the segmentation

and character error models, and for evaluation. We picked 500 lines from the UN corpus and generated their OCRed versions for training. We also generated the OCRed version of the test data. The OCR generation process was identical for both training and test data. We first typeset the data in 10pt Arial font, and printed it using a 600dpi laser printer. We scanned the printed pages in 100dpi to simulate lower image quality, and generated OCRed text from page images using a commercial system with native Spanish support.[12]

For our first set of Spanish experiments, we used training data from the UN corpus. About 350,000 lines of text containing about 12 million tokens were used for the language model step. Since our test data was all in lowercase, the word-to-character conversion step always generated lowercase, our model does not require training. Other sub-models were trained as explained in Section 3.4.2.

For the second set of Spanish experiments, we replaced the UN corpus with the FBIS corpus for language model training. The FBIS corpus had about 55,000 lines containing about 1.6 million tokens. While it is smaller than the UN corpus, it is in the news domain, which is a better match for the test data. The rest of the training data remained the same. For both sets, we tokenized the training and test data prior to experiments.

We experimented with two different character error models. The first is the same one used for French and described in Section 3.4.2. The second is a slightly modified version of the spelling correction model introduced in [12].[13] This

[12]Abby FineReader Professional Edition Version 7.0.

[13]We ignore the location of the error within the word since it is not as important for OCR as it is for spelling.

spelling model allows many-to-many edit operations, which makes $P(liter|litre) \approx P(l \mapsto l)P(i \mapsto i)P(tre \mapsto ter)$ possible. We refer to the these error models as Single Character Error Model (SC) and Multi-Character Error Model (MC), respectively. We extracted our correction lexicon from only the training data.

Reduction of OCR Error Rates The results for the first set of experiments are presented in Table 3.5. The *EM Type* column represents the Error Model (EM) that we use to implement the character transformation step. The MC model performs better than the SC model under similar conditions, which is not surprising since the MC model has more expressive power. The *EM Data* column shows how many lines of data are used for training the EM. More training data results in better performance, as usual. Note, however, even a small amount of OCR data for EM training is able to provide significant improvement over original OCR output. The differences between the original output and all the correction results are statistically significant. The *Max Err.* column gives the maximum number of character errors per token that is considered by the correction system, if any. Limiting the maximum number of errors considered per token reduces the performance slightly since the system can no longer correct tokens that have more character errors than the limit. This limit is imposed to speed up the correction process by reducing the search space, trading accuracy for speed.

Table 3.6 gives the distribution of words after the correction process. However, 200 correctly recognized words were missing from the lexicon and in-corrected as a result. All non-corrected words were valid word errors.

Conditions			Results	
EM Type	EM Data	Max Err.	WER (%)	Red. (%)
SC	50	-	18.69	9.10
SC	500	-	18.30	10.99
MC	50	3	18.65	9.29
MC	500	3	17.90	12.94
MC	50	-	18.45	10.26
MC	**500**	**-**	**17.77**	**13.57**
Original OCR Output			20.56	-

Table 3.5: Post-correction WER and reduction rates under various conditions for Spanish OCR output on Spanish text (UN Data).

Conditions			Distribution			
EM Type	EM Data	Max Err.	Corr.	Incor.	Miscor.	Noncor.
SC	50	-	313	200	370	304
SC	500	-	333	200	350	304
MC	50	3	315	200	367	304
MC	500	3	353	200	329	304
MC	50	-	319	200	364	304
MC	500	-	354	200	329	304

Table 3.6: Distribution of words in post-correction Spanish text (UN Data).

Conditions			Results	
Vocabulary	Merge/Split	Valid Word	WER (%)	Red. (%)
open	no	no	18.76	8.75
closed	no	no	12.96	36.96
closed	yes	no	12.86	37.45
closed	**no**	**yes**	**11.42**	**44.46**
Original OCR Output			20.56	-

Table 3.7: Post-correction WER and reduction rates under various conditions for Spanish OCR output on Spanish text (FBIS data).

Table 3.7 presents the results for the second set. We used the MC error model trained on all 500 lines of OCR training data for all experiments. We did not use the SC model for this set, as its performance was lower than the performance of the MC model for the first set. Since limiting the number of errors to three per token causes less than 1% relative reduction in correction performance for set 1, we impose the same limit for the second set to reduce execution time. For this set, in addition to an open vocabulary generated from the training data, we also use a closed version of the vocabulary that included all the words in the test set. All improvements are statistically significant.

In Table 3.7, the *Vocabulary* column represents whether an open or closed vocabulary is used for correction. The closed vocabulary contained all the words from the training and test sets while the open vocabulary was generated only from the training set. Since the model always picks the most probable word from its lexicon as the correction, using an open vocabulary means that any novel words that are not in the lexicon are replaced with a word from the lexicon, even if the novel words

are correctly recognized by the OCR system. Consequently, correction performance is much lower for the open vocabulary case. However, it is still significantly better than the original OCR output. Allowing merge/split errors improves results slightly, as suggested by the French experiments.

For this set, we also attempt to correct words that are in the lexicon, which we normally assume to be correct and copy over to output without modification. Valid word correction also helps, but we should note that the valid word errors that are corrected almost exclusively involve non-content words. A valid word correction is driven by the language model, as the character error model prefers to avoid costly edit operations. For the system to make a change, despite edit costs, the language model should exhibit a strong preference for the correction candidate. Since non-content words occur much more frequently compared to content words, they have a stronger bias in the language model. This stronger bias allows corrections to take place.

In Table 3.8, we present the resulting word distributions for the second set of experiments. As the last row shows, attempting to correct valid words allows us to fix 104 valid words errors in exchange for only 27 in-corrections. For the closed vocabulary case, 350 of the non-corrected words are valid word errors, 30 are merge errors, and the rest are word deletions or errors involving digits or punctuation.

Conditions			Distribution			
Vocabulary	Merge/Split	Valid Word	Corr.	Incor.	Miscor.	Noncor.
open	no	no	310	218	256	421
closed	no	no	389	0	186	412
closed	yes	no	391	0	184	412
closed	no	yes	495	27	192	300

Table 3.8: Distribution of words in post-correction Spanish text (FBIS Data).

3.5 Impact of OCR Errors and Error Correction on NLP Applications

While humans are extremely good at processing noisy input sources, current state of the computational technology is far from achieving near human performance. Many important NLP applications work on textual information, and their performance depends on the quality of the input they process, among other things. While performance of all applications are degraded by the noise in their input, the amount of degradation varies widely. Information retrieval, for instance, has been shown to be quite tolerant to considerable amount of noise in various forms [22, 133]. On the other hand, applications such as MT should, intuitively, be more sensitive to the quality of input.

This section evaluates the effects of recognition errors in OCR output on various NLP tasks and applications such as the following: word level alignment, translation lexicon generation, and MT. Since we are interested in the effects of OCR errors on the application performance, rather than overall performance of the application, we use the performance of the applications on clean data as the point

of comparison. However, if a gold-standard is available for a particular evaluation task, comparisons against real gold-standards are made as well.[14]

3.5.1 Word-Level Alignment of Parallel Text

Automated word-level alignment of parallel text is an important task. In addition to being one of the basic steps in training many SMT models [15, 66, 73], it is used to project annotations across parallel text as discussed in Chapter 5 and to acquire translation lexicons as reported in Section 3.5.2, among other things. (vs. a list of two things, since there are other applications, not just the two mentioned here) Therefore, ability to improve the performance of word-level alignment on OCR data is an important step in achieving better NLP performance.

The quality of word-level alignments can be evaluated using a standard precision/recall metric, or specific measures such as AER [101]. However, word merge/split errors introduced by OCR prevent direct application of these methods to evaluate alignment of OCR generated text. We propose a new method to compute precision/recall, which is described in Section 3.5.1. Our proposed method can accommodate merged/split words. We also define a new metric called Coverage-based Alignment Error Rate (CAER) that is a modified version of AER.

We performed our evaluations using English and French versions of the Bible, using the OCR data from Section 3.4.3. We used automatically obtained word-level alignment between clean English and clean French data as the reference set, and

[14]We use gold-standard for referring to the desired output of NLP tasks and applications, as opposed to ground-truth, which refers to an error-free version of OCR text.

computed the precision and recall, using the word coverage method described in Section 3.5.1, for alignments obtained using clean English and OCRed French text. Alignments are computed by training a statistical translation model on approximately 30,000 lines of text using GIZA++ [101]. Using the alignments obtained from clean-English/clean-French as the reference set, alignments obtained using clean-English/OCRed-French have a precision of 92.67%, recall of 91.78%, and a coverage error rate of 7.78%. These numbers clearly show that the OCR errors have a significant effect on the resulting alignments.

We could not use the complete Bible for evaluating the impact of post-processing on word alignment as we corrected only a small portion of it, using the rest to train the post-processing system. Therefore, we used only the test data from Section 3.4.3 for evaluating alignment performance on corrected text. We removed punctuation symbols and digits from the data since lexicon extraction experiments reported in Section 3.5.2 used the resulting word level alignments.

We aligned clean English text with clean French text to generate reference alignments. We then aligned OCRed and corrected French text with clean English text to compare their alignment scores. Table 3.9 shows the alignment evaluation results, where CAER is reduced from 14.33% to 9.08% by using OCR correction. Since the amount of data was much smaller, 500 lines containing about 11,000 tokens, the alignment scores are lower compared to the results presented for the complete Bible.

Data	Precision	Recall	CAER
OCR	86.16	85.20	14.33
Corrected	91.13	90.71	9.08

Table 3.9: Evaluation of word alignments between clean English and OCRed French text before and after post-processing.

Word Coverage Method for Alignment Evaluation

Och and Ney [101] propose AER to evaluate word level alignment between parallel texts. AER is computed using precision and recall metrics, which are slightly modified: The reference alignment links are labeled as *sure* or *possible* links. Missing a link is considered a recall error only if it is a sure link, whereas an extra link is considered a precision error if it is not even a possible link. If the distinction between sure and possible links is ignored, AER is equivalent to 1−F-measure.

However, if reference alignments are available for clean text, which is usually the case, and alignments to be evaluated are for noisy text, computing precision/recall, hence AER, is not straightforward. Merged/split words in the noisy text prevent direct comparison of the word alignments with the alignments on the clean text. For instance, if the clean word '`foobar`' is split in the noisy text as '`foo bar`', one needs a way to compare the alignments for '`foobar`' to the alignments for both '`foo`' and '`bar`'.

We propose a new method, named Coverage-based Alignment Error Rate (CAER), based on word coverage rather than alignment links that can handle merged/split words for precision/recall computation. Since our reference sets do

not have sure/possible links we ignore the distinction between the two.[15] We will assume that the source language is clean text, and that the target language is noisy, without loss of generality. We start with some basic definitions and then provide the details of the proposed evaluation metric.

A sentence S is a sequence of words $s_1, s_2, ..., s_n$. Given two sentences $S = s_1, s_2, ..., s_n$ and $T = t_1, t_2, ..., t_m$, which are translations of each other in two different languages, an alignment A is a set of pairs $\langle i, j \rangle$ that maps the i^{th} word of source sentence S to the j^{th} word of target sentence T, where $1 \leq i \leq n$ and $1 \leq j \leq m$. The following two functions return the set of target tokens aligned to a source token i under alignment A, and vice versa:

$$SrcAln(i, A) = \{j \mid \langle i, j \rangle \in A\}$$

$$TrgAln(j, A) = \{i \mid \langle i, j \rangle \in A\}$$

Given a sentence $S = s_1, s_2, ..., s_n$ and its noisy version $O = o_1, o_2, ..., o_m$, a clean-text/noisy-text mapping M is defined as the partitioning of S and O into an equal number of non-overlapping, consecutive chunks, $m_1, m_2, ..., m_k$ of zero or more words where words in the noisy chunk k are assumed to be the noisy version of words in the clean chunk k. Each chunk is an ordered pair of sequences $\langle (i, i+1, ..., i+p), (j, j+1, ..., j+q) \rangle$ where $1 \leq i \leq n$, $0 \leq p \leq n-i$, $1 \leq j \leq m$, and $0 \leq q \leq m-j$. Note that either sequence in parenthesis can be empty. Refer to Section 3.3.4 for examples and an algorithm to compute M. The following two functions return the

[15] Sure/possible link distinction can be incorporated into the proposed metric easily, if desired.

set of clean words and the set of noisy words for a chunk $m \in M$.

$$Clean(m) = \{j \mid j \text{ is in clean (first) sequence of chunk } m\}$$

$$Noisy(m) = \{j \mid j \text{ is in noisy (second) sequence of chunk } m\}$$

The usual method for word-level alignment evaluation is to compare the proposed alignment A' to reference alignment A, and to use precision/recall or AER. However, since word merge/split errors change the number and position of tokens, the two alignments are no longer directly comparable. First, a mapping between clean reference positions and noisy test positions is needed. However, even when this mapping is present, a traditional method is still inapplicable for merged/split words, as explained previously.

We propose the use of the source tokens that are linked to by each target token as the comparison unit, rather than individual alignment links. When there is a merge/split, we will have more than one word on either (possibly both) sides of the corresponding clean-text/noisy-text mapping, and we will simply use the union of source words covered by all words on the side with multiple words.

Even with all the correct words are covered, we still have some missing information. For instance, a merged noisy word may cover all correct source words, but we do not know which actual clean word was linked to those words. To account for this, we introduce a penalty based on the number of words in each chunk. The penalty is imposed only for recall, since there is missing information, but not for precision, since there is no erroneous information. The penalty is based on a length

factor which is defined as follows

$$LenFact(m) = |Clean(m)| \times |Noisy(m)|$$

Inserted/deleted tokens are trivially handled by counting all their links as extra/missing links, respectively. Note that CAER reduces to a traditional link-based approach when there is no merged/split tokens, which is a desirable property for inter-comparability between the two metrics.

To define the method more formally, given a source sentence S, a target sentence T, a noisy version of the target sentence O, an alignment between S and T A, another alignment A' between T and O, and a clean-text/noisy-text mapping M between T and O, we compute precision and recall as follows:

$$Precision = \frac{CWC}{TWC}$$
$$Recall = \frac{CWC}{RWC}$$

where CWC (correct word coverage), TWC (total word coverage), and RWC (reference word coverage) are defined as

$$CWC = \sum_{m \in M} \frac{\left| \left(\bigcup_{o \in Noisy(m)} TrgAln(o) \right) \bigcap \left(\bigcup_{t \in Clean(m)} TrgAln(t) \right) \right|}{LenFact(m)}$$
$$TWC = \sum_{m \in M} \frac{\left| \bigcup_{o \in Noisy(m)} TrgAln(o) \right|}{LenFact(m)}$$
$$RWC = \sum_{m \in M} \left| \bigcup_{t \in Clean(m)} TrgAln(t) \right|$$

Once precision and recall are computed, CAER can be computed as follows

$$CAER = 1 - \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Although precision and recall ares well-known and intuitive metrics for evaluation, CAER is defined to allow ease of comparison with evaluations performed using AER.

Let us illustrate this on a simple example. Assume we have the following parameters

$$
\begin{aligned}
S &= \text{`}\texttt{the blue house}\text{'} \\
T &= \text{`}\texttt{la maison bleu}\text{'} \\
O &= \text{`}\texttt{la maisonbleu}\text{'} \\
A &= \{\langle 1,1\rangle, \langle 2,3\rangle, \langle 3,2\rangle\} \\
A' &= \{\langle 1,1\rangle, \langle 2,2\rangle, \langle 3,2\rangle\} \\
M &= \{\langle (1),(1)\rangle, \langle (2,3),(2)\rangle\}
\end{aligned}
$$

Computation of CAER would proceed as follows

- $RWC = 1 + 2 = 3$

  The total number of source words aligned with any of the clean words in a chunk, summed over all chunks, is 3. Chunk 1 has clean word 1, which is aligned to one word, chunk 2 has clean words 2 and 3 which are aligned to a total of two words. Note that, if a source word is aligned to words in both chunks, it is counted multiple times, once for each chunk.

- $LenFact(m_1) = 1 \times 1 = 1$

  The clean side of chunk m_1 has 1 word, and the noisy side has 1 word.

- $LenFact(m_2) = 2 \times 1 = 2$

  The clean side has 2 words, and the noisy side has 1 word.

- $TWC = 1/1 + 2/2 = 2$

Noisy words link to 1 source word for chunk 1, and 2 source words fro chunk 2. The total for each chunk is divided by the *LenFact* value, as computed above, for that chunk.

- $CWC = 1/1 + 2/2 = 2$

  CWC is very similar to TWC, except it counts words aligned to noisy words of a chunk if and only if they are aligned to clean words too. For this example, this is the case for all words, so CWC and TWC have the same value, 2.

- $Precision = CWC/TWC = 2/2 = 1.0$

- $Recall = CWC/RWC = 2/3 = 0.67$

- $CAER = 1 - \frac{2 \times 1.0 \times 0.67}{1.0 + 0.67} = 0.198$

We get perfect precision because all the words that we align to a noisy word are aligned to one of the corresponding clean words. However, recall is lower, because some information is lost. For the second chunk, '`maisonbleu`' corresponds to '`maison`' and '`bleu`' and is aligned to '`blue`' and '`house`' which are all the words aligned to '`maison`' and '`bleu`'. All words are there, but individual alignment information is missing.

3.5.2 Automated Translation Lexicon Generation

Almost all multilingual applications in NLP, such as MT and Cross Language Information Retrieval (CLIR), require a machine readable translation lexicon. Such lexicons are not always readily available, particularly for resource-poor languages.

Consequently, various methods to acquire translation lexicons for new languages have been developed. While some approaches exploit lexicons available in printed form [28], some others use statistical methods to extract a lexicon using parallel text [113]. Regardless of the method used, the noise in the input text has an effect on the quality of the resulting lexicons.

For experiments, we assume that the English side of the parallel text is clean and its foreign language translation is noisy, and evaluate the quality of the generated lexicon using the lexicon generated from all-clean data as the reference point.

We use the word level alignments generated for alignment evaluation reported in Section 3.5.1 to extract cross-language word co-occurrence frequencies. Next, we score candidate translation lexicon entries that are extracted from GIZA++ translation table using the log likelihood ratio [30] (cf. [113]), and pick top 1,000 as our translation lexicon. We generate three lexicons by pairing the English text with clean, OCRed, and corrected French text. All text is tokenized and lowercased. Single character tokens and tokens with no letters are removed. This method of generating a translation lexicon works reasonably well, as the top twenty entries from the lexicon generated using clean French text presented in Table 3.10 illustrate.

Figure 3.8 gives the precision-recall curves for the translation lexicon generated from clean English coupled with OCRed and corrected French, using the top 1,000 entries of the lexicon generated from clean-English/clean-French as the target set. Since we are interested in the effect of OCR, independent of the performance of the lexicon generation method, the lexicon auto-generated from clean text provides a reasonable target set.

and	et	for	car
of	de	if	si
god	dieu	ye	vous
we	nous	you	vous
christ	christ	the	le
not	pas	law	loi
but	mais	jesus	jésus
lord	seigneur	as	comme
the	la	that	qui
is	est	in	dans

Table 3.10: Translation lexicon entries extracted using clean English and clean French text.

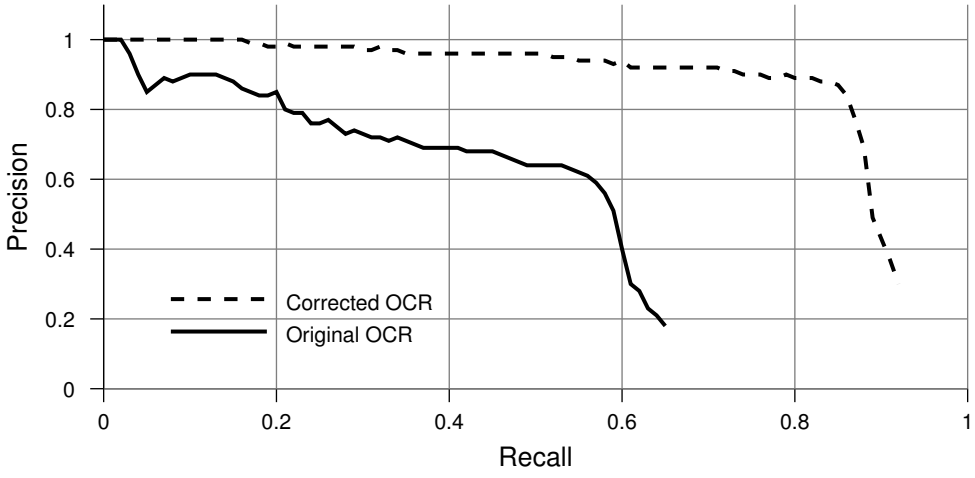

Figure 3.8: Effects of OCR errors and post-processing on translation lexicon acquisition.

The graph clearly illustrates that the precision of the translation lexicon generated using original OCR data degrades quickly as recall increases. Precision is 40% for a recall of 60%, and it drops below 20% before recall reaches 70%. The dashed line shows that error correction significantly improves the performance of automated translation lexicon generation. While the precision of the lexicon generated using original OCR data drops quickly, the corrected version maintains its precision above 90% up to a recall of 80%.

The noise in the text has a two-fold effect on the quality of the lexicon entries. First, it degrades word-level alignment quality as discussed in Section 3.5.1, which makes it harder to acquire the associations between the words in the two languages. Second, some of the tokens that make it into the lexicon are noisy. This second case is illustrated better in an example: One of the entries in the lexicon generated using OCR data is '`it(e) : i1(f)`', which correctly identifies the translational relationship between two tokens in the text, but the French word is a noisy version of the correct word '`il`'. Such entries are counted as wrong in the context of a machine readable lexicon.

3.5.3 Machine Translation

The applications discussed so far are mainly sub-components that are required for other NLP applications, and are not of direct use to the end-user. We use MT to extend our evaluation to end-user applications. MT is becoming more and more important as the amount of foreign language documents available to the general

public increases rapidly thanks to the WWW. MT is an important application to evaluate also because it works with several sub-components to accomplish a complicated goal. Many different approaches to MT have been proposed, but they mainly fall under two categories, statistical or symbolic. We evaluate an example system from each category.

We evaluate the symbolic systems using a commercially available MT system, Systran, which is considered to be state of the art. It is important to see how it reacts to noisy text not only because it is representative of the symbolic MT approach, but also it is a commercial system that has undergone many years of professional research and development.

For evaluating the SMT framework, we use a publicly available implementation of the IBM style SMT system, which consists of components developed at various academic institutions. Translation models are trained using GIZA++ [101], language models are generated using CMU-Cambridge Toolkit [17], and decoding is performed using ISI ReWrite Decoder [37, 38]. We employed the widely used BLEU metric to measure translation quality [106].

The test data for Spanish experiments are generated using the multiple-translation Chinese corpus from Linguistic Data Consortium (LDC) [54]. The original Chinese data contains 105 documents from the news domain, each of which has 10 different manual translations into English. Spanish data is generated by selecting one sentence from each document, and having a native Spanish speaker translate one of the reference English translations into Spanish for each selected sentence. The resulting Spanish translations are then verified by two linguistics graduate students

who are also native Spanish speakers. The result is 105 Spanish sentences with 10 English translations for each.

We translated the clean, OCRed, and post-processed versions of the Spanish test set into English using Systran and ReWrite and evaluated the results using Bi-Lingual Evaluation Understudy (BLEU). Details of the OCR data and post-processing are provided in Section 3.4.3. Language and translation models for ReWrite were trained on the FBIS corpus, using about 55 thousand lines containing about 1.6 million tokens.

MT System	Input Text	BLEU Score
Systran	OCR	0.2000
Systran	Corrected	0.2606
Systran	Clean	0.3188
ReWrite	OCR	0.1792
ReWrite	Corrected	0.2234
ReWrite	Clean	0.2590

Table 3.11: BLEU scores for English translations from OCRed, corrected, and clean Spanish input

Table 3.11 shows the BLEU scores for all combinations of test data and translation systems. As can be seen from the table, OCR errors degrade the translation performance of both MT systems. We are able to bring the performance much closer to the level of performance on clean text by using post-processing.

3.6 Summary

One of the most basic computational resources required for a language is electronic corpora, which is not available for many low-density languages. OCR provides a quick and cost-effective way to acquire electronic corpora provided that printed material is available. However, many low-density languages lack an OCR system altogether; and when available, the quality of OCR systems for less studied languages is far from perfect.

We first presented our preliminary OCR error correction studies in Section 3.3, and established that post-processing can reduce OCR error rate significantly. We were able to correct 65% to 70% of words mis-recognized by an OCR system using only an error model that was coupled with a lexicon containing correct words.

Next, in Section 3.4, we presented a flexible, modular, probabilistic generative OCR model designed specifically for ease of integration with probabilistic models of the sort commonly found in recent NLP work, and for rapid retargeting of OCR and NLP technology to new languages.

In a rigorous evaluation of OCR error correction on real data, illustrating a scenario where a black-box commercial English OCR system is retargeted to work with French data, we obtained a 70% reduction in word error rate over the English-on-French baseline, with a resulting word accuracy of 97%. It is worth noting that post-processing of the English OCR on French text led to better performance than a commercial French OCR system run on the same text. We also evaluated the performance of the correction system on native OCR output, using Spanish data.

WER of the original OCR output was 21%, and post-processing reduced it as much as 44%, achieving a WER of 11% using a closed vocabulary, and 18% when an open vocabulary is employed.

After presenting post-processing methods and evaluating their performance in reducing OCR error rate, we evaluated the impact of OCR errors and post-processing on NLP applications in Section 3.5.

We first experimented with automated word alignment, which is an important sub-task required by many NLP methods and applications. We developed a novel evaluation metric named CAER, presented in Section 3.5.1, for evaluating alignment quality because traditional evaluation metrics are not applicable to OCR data containing word merge/split errors. We show that CAER can be reduced 37% by correcting OCR errors prior to alignment. We should, however, note that the amount of training data for alignment was small; using more data results in lower CAER.

We next evaluated error correction in a resource-acquisition scenario involving translation lexicon acquisition from OCR output. The results show that our post-OCR correction framework significantly improves performance. Although the lexicon obtained using post-processed text contains errors, it still is a useful resource. As demonstrated in [114] demonstrate that even noisy extensions to dictionary-based translation lexicons, acquired from parallel text, can have a positive impact on CLIR.

Finally, we measured the impact of post-processing on MT, quantifying OCR degradation of MT performance and showing that our technique moves the perfor-

mance of MT on OCR data significantly closer to the performance obtained with clean input. The results were valid across translation systems (e.g. symbolic translation using a commercial package and statistical translation using academic software).

Chapter 4

Get the Words: Reading without Knowing the Words

4.1 Correction without Using a Lexicon

The correction method we presented in Chapter 3 requires a lexicon that covers all words in the processed text — a strong assumption, especially for low-density languages. Therefore, we developed a simplified version of our correction system by using a character-based model rather than a word-based one, removing the need for a lexicon, which can be crucial for low-density languages.

Although word-based models generally perform better, moving from words to characters is a necessary compromise because word-based models are useless in the absence of a lexicon, which is the case for many low-density languages. We show that using a small amount of training data, it is possible achieve OCR performance close to, or even better than, the performance of a native OCR system. We further demonstrate that the same method can be used to improve the accuracy of a trained OCR system for the new language, as well as the performance of a native OCR system on its intended target language.

4.2 Modified OCR Model

We replace the generation of W in our original model by the generation of C, which renders $P(C|W)$ irrelevant, and the model becomes the following:

$$P(O, b, a, C) = P(O, b|a, C)P(a|C)P(C)$$

In addition to eliminating the need for a word lexicon, we develop a novel method for handling word merge/split errors. Rather than modeling these errors explicitly using a segmentation model, we simply treat them as character deletion/insertion errors involving the space character, allowing us to handle them within the error model. The segmentation step is absorbed into the character transformation step, so a and b are no longer necessary, hence the final equation becomes the following:

$$P(O, C) = P(O|C)P(C)$$

which is a direct application of the noisy channel model. We can describe the new generative process as follows: First, a sequence of characters C are generated, with probability $P(C)$, and the OCR system converts it into O with probability $P(O|C)$. For example, if the actual input is a_car and it is recognized as $ajar$, $P(ajar, a\_car) = P(ajar|a\_car)P(a\_car)$. Use of the channel model to address word merge/split errors without actually using a word-level model is, to our knowledge, a novel contribution of our approach.

4.3 Implementation

We implemented our new model using WFSM framework that allowed us to use existing components from the original model with little or no modification.

4.3.1 Source Model

The source model assigns probability $P(C)$ to original character sequences, C. We use character-level n-gram language models as the source model, since n-gram models are simple, easy to train, and usually achieve good performance. More complicated models that make use of constraints imposed by a particular language, such as vowel harmony, can be utilized if desired. We use the CMU-Cambridge Language Modeling Toolkit [17] for training. The model is then encoded as a Weighted Finite State Automaton (WFSA). Intuitively, higher order n-grams perform better, as they take more context into account. Unfortunately, as n increases, the size of the language model FSA increases rapidly, limiting the practical n-gram size that can be used. The limit can be extended by using variable-length models [45, 53].

4.3.2 Channel Model

The channel model assigns a probability to O given that it is generated from C. We use SC and MC error models, which are discussed in Section 3.4.3. The only difference between the word-level model and character-level model is how space characters are handled. While the original model deleted all spaces without inserting new spaces, the modified version treats spaces like any other character. For exam-

ple, $P(ajar|a\_car) \approx P(a \mapsto a)P(\_ \mapsto \epsilon)P(c \mapsto j)P(a \mapsto a)P(r \mapsto r)$. (Note that we only consider the most likely edit sequence here, as opposed to summing over all possible ways to convert a_car to $ajar$.)

Training is performed the same as it was performed for the word-level model. We train both error models over a set of corresponding ground truth and OCR sequences, $\langle C, O \rangle$, using Expectation-Maximization. First we find the most likely edit sequence for each training pair to update the edit counts. We then use the updated counts to re-estimate edit probabilities. For MC, after finding the most likely edit sequence, extended versions of each non-copy operation that include neighboring characters are also considered, which allows the system to learn any common multi-character mappings. Following [12], MC training performs only one iteration of Expectation-Maximization.

In order to reduce the time and space requirements of the search at correction time, we impose a limit on the number of errors per token. We use the highest limit possible, given our hardware and time constraints. The error-limit ranged from two to five errors per token. It is possible to correct more errors per token by iterating the correction process. However, iterative correction cannot guarantee that the result is optimal under the model.

4.3.3 Chunking

Since we do not require a lexicon, we work on lines of text rather than words. Unfortunately the search space for correcting a complete line is prohibitively large

and we need a way to break it down to smaller, independent chunks. The chunking step is not part of the model, but rather a pre-processing step:

1. Chunks are identified
2. Each chunk is corrected independently using the model
3. The corrected chunks are put back together to generate the final output.

Spaces provide a natural break point for chunks. However, split errors complicate the process. If parts of a split word are placed in different chunks, the error cannot be corrected. For example, in Figure 4.1, chunking (b) allows the model to produce the desired output, but chunking (a) simply does not allow combining '`sam`' and '`ple`' into '`sample`', as each chunk is corrected independently.

Figure 4.1: Examples of chunking: (a): not desired (b) desired

We address this using the probabilities assigned to spaces by the source model for chunking. We break the line into two chunks using the space with the highest probability and repeat the process recursively until all chunks are reduced to a reasonable size as defined by time and memory limitations. Crucially, spurious spaces that cause split errors are expected to have a low probability, and therefore breaking the line using high probability spaces reduces the likelihood of placing parts of a split word in different chunks.

If a lexicon happens to be available, we can use it to achieve more reliable chunking. The tokens of the input line that are present in the lexicon are assumed to be correct. We identify runs of out-of-lexicon tokens and attempt to correct them together, allowing us to handle split errors. Note that in this case the lexicon is used only to improve chunking, not for correction. Consequently, coverage of the lexicon is far less important.

Chunking Evaluation

We evaluated the performance of our chunking algorithm on the Arabic test data used for OCR experiments in Section 4.4.3. We implemented a greedy chunker to be used as a baseline. The greedy chunker sequentially scans the input, placing a chunk boundary whenever the size of the current chunk reaches a level where including the next token would exceed the size limits. We run the recursive chunker using the 6-gram language model that is also used for correction experiments in Section 4.4.3. We set the maximum chunk size to 3 tokens or 20 characters, using whichever is smaller for each particular case. Chunk boundaries that are placed at word split points, such as the second one in Figure 4.1 (a), are counted as errors, all other positions are considered acceptable.

The greedy chunker placed an erroneous boundary at 31.40% of word split points, while the recursive chunker placed an erroneous boundary at only 11.28% of split points.

Impact of Chunking on Correction

Although the number of erroneous chunk boundaries is a good indication of the overall chunking performance, the real test is how OCR correction performance is effected by chunking decisions. We use the best performing correction setup from Section 4.4.3 and run it using both greedy and recursive chunking. The Arabic correction experiments reported in Section 4.4.3 use lexicon-based chunking, where a boundary is placed on each side of every token that appears in the correction lexicon. Ignoring split errors that result in one or more valid words, lexicon-based chunking is always error-free. While the recursive chunker has an error rate of 11.28%, the correction performance was identical to that of error-free chunking. The greedy chunker, on the other hand, caused the error rate to increase slightly from 17.74% to 17.81%.

Incorrect decisions of the recursive chunker did not hurt because the correction method was not able to fix those particular split errors, regardless. The errors of the chunking and correction models coincided as they both rely on the same language model. Therefore, recursive chunking errors are unlikely to reduce correction performance.

4.3.4 Correction

Correction is performed by estimating the most probable source character sequence $\hat{C}$ for a given observed character sequence O, using the formula:

$$\hat{C} = \underset{C}{\operatorname{argmax}}\big(P(O|C)P(C)\big)$$

We first encode O as an Finite State Automaton (FSA) and compose it with the inverted error model FST. The resulting FST is then composed with the language model FSA. The final result is a lattice that encodes all feasible sequences C, along with their probabilities, that could have generated O. We take the sequence associated with the most likely path through the lattice as $\hat{C}$.

4.4 Evaluation

We evaluate our work on OCR post-processing in a number of scenarios relevant for NLP, including the following:

- Creation of new OCR capabilities for low-density languages
- Improvement of OCR performance for a native commercial system
- Acquisition of knowledge from a foreign-language dictionary
- Creation of a parallel text
- Machine translation from OCR output

The languages studied include Igbo, Cebuano, Arabic, and Spanish. For intrinsic evaluation, we use the conventional WER metric, as described in Section 3.4.3. For extrinsic evaluation of machine translation, we use the BLEU metric [106].

4.4.1 Igbo: Creating an OCR System

Igbo is an African language spoken mainly in Nigeria by an estimated 10 to 18 million people, written in Latin script. Although some Igbo texts use diacritics to

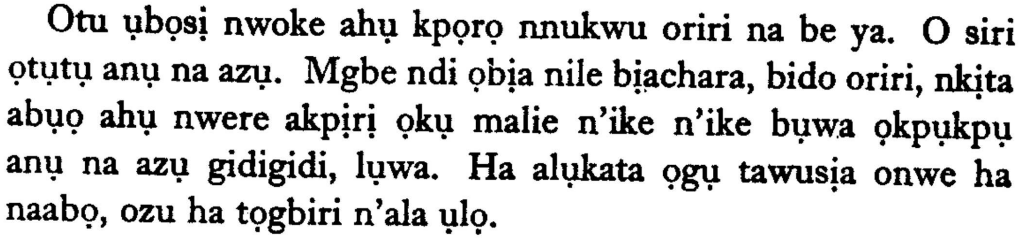

Figure 4.2: A small excerpt from Akụkọ

mark tones, they are not part of the official orthography and they are absent in most printed materials. Other than grammar books, texts for Igbo, even hardcopy, are extremely difficult to obtain. To our knowledge, the work reported here creates the first OCR system for this language.

For the Igbo experiments, we used two sources. The first is a small excerpt containing 6,727 words from the novel "Juo Obinna" [139]. The second is a small collection of short stories named "Akụkọ Ife Nke Ndị Igbo" [44] containing 3,544 words. We refer to the former as "Juo" and the latter as "Akụkọ" hereafter. We generated the OCR data using a commercial English OCR system.[1] Juo image files were generated by scanning a 600dpi laser printer output at 300dpi resolution. Akụkọ image files were generated by scanning photocopies from the bound hardcopy at 300dpi. Figure 4.2 provides a small excerpt from the actual Akụkọ page images used for recognition. For both texts, we used the first two thirds for training and the remaining third for testing.

We trained Error Models (EMs) and Language Models (LMs) using the training sets for Juo and Akụkọ separately, and performed corrections of English OCR

[1] Abby FineReader Professional Edition Version 7.0

Conditions			Results	
LM Data	EM Data	EM Type	WER (%)	Red. (%)
Juo	Juo	MC	8.66	74.18
Juo	Akụkọ	MC	15.23	54.59
Akụkọ	Juo	MC	13.25	60.49
Akụkọ	Akụkọ	MC	19.08	43.11
Juo	**Juo**	**SC**	**7.11**	**78.80**
Juo	Akụkọ	SC	11.49	65.74
Akụkọ	Juo	SC	13.42	59.99
Akụkọ	Akụkọ	SC	18.92	43.59
Original OCR Output			**35.44**	-

Table 4.1: Post-correction WER for English OCR system on Juo

output using different combinations of these models on both test sets. Table 4.1 shows the results for the Juo test set while Table 4.3 presents the results for Akụkọ. The relative error reduction ranges from 30% to almost 80%. The SC error model performs better than the MC error model under all conditions, but the difference is statistically significant only for the correction of Juo using Juo LM Akụkọ EM combination. The performance difference is due to the fact that MC requires more training data than SC. Furthermore, most of the errors in the data did not require many-to-many operations. Results in Tables 4.1 and 4.3 are for a 6-gram language model; corresponding 3-gram error rates were 1% to 2% (absolute) higher. The differences between the original OCR output and the correction results are statistically significant for both tables.

Table 4.2 and Table 4.4 present the distribution of words after correction. When SC is used, the correction system modifies more words compared to MC,

Conditions			Distribution			
LM Data	EM Data	Em Type	Corr.	Incor.	Miscor.	Noncor.
Juo	Juo	MC	531	30	38	94
Juo	Akụkọ	MC	420	42	65	178
Akụkọ	Juo	MC	514	99	65	84
Akụkọ	Akụkọ	MC	387	81	101	175
Juo	Juo	SC	586	56	36	41
Juo	Akụkọ	SC	556	108	62	45
Akụkọ	Juo	SC	544	132	71	48
Akụkọ	Akụkọ	SC	508	199	105	50

Table 4.2: Distribution of words in post-correction English OCR output on Juo.

causing both more corrections and in-corrections, but the net result is positive. Using SC results in more modifications to the original OCR text because it has higher probabilities associated with single character substitution operations compared to MC. While SC assigns the credit for each non-copy edit operation in the training data solely to the character pair involved, MC divides it among various size substrings covering the non-copy operation in order to estimate the probability of many-to-many edit operations.

The best correction performance is achieved when both the EM and Language Model (LM) training data come from the same source as the test data, almost doubling the performance achieved when they are from a different source.[2] Note that the amount of training data is small, four to eight pages, so optimizing performance via the manual entry of document-specific training text is not unrealistic for scenarios

[2]There is no overlap between training and test data under any circumstance.

Conditions			Results	
LM Data	EM Data	EM type	WER (%)	Red. (%)
Juo	Juo	MC	21.42	36.33
Juo	Akụkọ	MC	18.08	46.25
Akụkọ	Juo	MC	21.51	36.06
Akụkọ	Akụkọ	MC	18.16	46.02
Juo	Juo	SC	19.92	40.78
Juo	Akụkọ	SC	16.49	50.98
Akụkọ	Juo	SC	19.92	40.78
Akụkọ	**Akụkọ**	**SC**	**16.40**	**51.25**
Original OCR Output			**33.64**	-

Table 4.3: Post-correction WER for English OCR system on Akụkọ

Conditions			Distribution			
LM Data	EM Data	Em Type	Corr.	Incor.	Miscor.	Noncor.
Juo	Juo	MC	200	54	58	143
Juo	Akụkọ	MC	241	55	46	114
Akụkọ	Juo	MC	210	65	68	123
Akụkọ	Akụkọ	MC	253	68	51	97
Juo	Juo	SC	237	73	72	92
Juo	Akụkọ	SC	303	98	57	41
Akụkọ	Juo	SC	259	95	66	76
Akụkọ	Akụkọ	SC	317	111	44	40

Table 4.4: Distribution of words in post-correction English OCR output on Akụkọ.

involving long documents such as books.

One interesting result in Table 4.3 is that using a language model trained on Juo performs better than one trained on Akụkọ. Although this seems counter-intuitive, it has a very simple explanation: Akụkọ is a collection of short stories. Therefore, the similarity between different parts of Akụkọ is not necessarily higher than the similarity between Akụkọ and Juo.

Using a Trainable OCR System

In an additional experiment with Igbo, we find that post-processing can improve performance substantially even when an OCR system trained on Igbo characters is the starting point. In particular, the commercial OCR system used for Igbo experiments supports user-trained character shape models. Using Juo as the source, we trained the commercial OCR system manually on Igbo characters, resulting in a 7.43% WER on Juo without post-processing. Note that this is slightly higher than the 7.11% WER achieved using an English OCR system together with our

Conditions		Results	
LM Data	EM Data	WER (%)	Red. (%)
Juo	**Juo**	**3.69**	**50.34**
Juo	Akụkọ	5.24	29.48
Akụkọ	Juo	5.08	31.63
Akụkọ	Akụkọ	7.38	0.67
Original OCR Output		**7.43**	-

Table 4.5: Post-correction WER for trained OCR system on Juo

Conditions		Distribution			
LM Data	EM Data	Corr.	Incor.	Miscor.	Noncor.
Juo	Juo	120	50	1	18
Juo	Akụkọ	75	34	7	57
Akụkọ	Juo	118	74	1	20
Akụkọ	Akụkọ	72	71	8	59

Table 4.6: Distribution of words in post-correction trained OCR output on Juo..

post-processing model.[3] Table 4.5 shows that by post-processing the Igbo-trained OCR system, we reduce the word error rate by up to 50%. Reduction rates are statistically significant except for the case where both LM and EM are trained on Akụkọ. Word distributions after correction are presented in Table 4.6. We used SC EM and 6-gram LM for these experiments as they performed better for the experiments on the English OCR data.

Repeating the experiment for Akụkọ, the trained OCR system resulted in 5.86% WER. As expected, models trained on Juo performed much better than the models trained on Akụkọ. We present post-correction WER in Table 4.7 and distribution of words in Table 4.8. As Table 4.7 shows, we were able to provide some improvement when we used the language model trained on Juo, but the error rate actually increased when we used the language model trained on Akụkọ. None of the improvements/degradations are statistically significant. We believe the reason for poor correction performance is that Akụkọ is less consistent, both content-wise and image-wise, compared to Juo. This experiment shows that the amount of training

[3]The system trains by attempting OCR on a document and asking the user for the correct character whenever it is not confident.

Conditions		Results	
LM Data	EM Data	WER (%)	Red. (%)
Juo	Juo	5.36	8.53
Juo	**Akụkọ**	**5.27**	**10.07**
Akụkọ	Juo	7.28	-24.23
Akụkọ	Akụkọ	6.19	-5.63
Original OCR Output		**5.86**	-

Table 4.7: Post-correction WER for trained OCR system on Akụkọ

Conditions		Distribution			
LM Data	EM Data	Corr.	Incor.	Miscor.	Noncor.
Juo	Juo	35	29	5	30
Juo	Akụkọ	38	31	8	24
Akụkọ	Juo	40	57	8	22
Akụkọ	Akụkọ	44	48	8	18

Table 4.8: Distribution of words in post-correction trained OCR output on Akụkọ.

data and having a good match between the training and test data are even more crucial for correcting the output of an OCR system trained on the specific low-density language.

4.4.2 Cebuano: Acquiring a Dictionary

Cebuano is a language spoken by about 15 million people in the Philippines, written in Latin script. The scenario for this experiment is to convert a Cebuano hardcopy dictionary into electronic form, as in DARPA's Surprise Language Dry Run [100]. The dictionary that we used had diacritics, probably to aid in pronunciation. The

starting-point OCR data was generated using a commercial OCR system[4] as part of an effort to acquire an on-line Cebuano dictionary from a hard-copy version [80]. We sampled 566 entries (1,471 tokens, 2,418 tokens including punctuation when tokenized) from the dictionary as the test set and used various portions of the remaining 10,759 entries (27,351 tokens) for training.

The fact that the tokens to be corrected come from a dictionary means (1) there is little context available and (2) word usage frequencies in running text are not reflected. [5] Character-based models may be affected by these considerations, but probably not to the extent that word-based models would be.

Table 4.9 shows WER for Cebuano after post-processing and Table 4.10 presents corresponding word distributions. The *size* column represents the number of dictionary entries used for training, where each entry consists of one or more Cebuano words. As can be seen from the table, our model reduces WER substantially for all cases, ranging from 20% to 50% relative reduction, all of which are statistically significant. As expected, the correction performance increases with the amount of training data. Note, however, that we achieve reasonable correction performance even when we use only 500 dictionary entries for training.

Contrary to the results for Igbo, the Multi-Character Error Model performs

[4]ScanSoft Developer's Kit 2000, which has no built-in support for Cebuano.

[5]We use the term *context* in its n-gram sense. In particular, context means the $n-1$ tokens preceding the token in question. Since the entries are head words from a dictionary, they do not have the context information that is observed in running text. When higher contextual levels, such as page-level and dictionary-level as opposed to word-level, are considered there is context information that could potentially be exploited for this task. For example, the fact that headwords are in alphabetical order can be considered contextual information. We consider only n-gram context here.

Conditions			Results	
Size	LM	EM	WER (%)	Red. (%)
500	3-gram	SC	5.37	33.04
500	3-gram	MC	5.05	37.03
500	6-gram	SC	6.41	20.07
500	6-gram	MC	5.33	33.54
1,000	3-gram	SC	5.33	33.54
1,000	3-gram	MC	4.63	42.27
1,000	6-gram	SC	5.58	30.42
1,000	6-gram	MC	4.67	41.77
27,363	3-gram	SC	4.34	45.89
27,363	3-gram	MC	4.14	48.38
27,363	6-gram	SC	4.55	43.27
27,363	**6-gram**	**MC**	**3.97**	**50.50**
Original OCR Output			**8.02**	-

Table 4.9: Post-correction WER for English OCR output on Cebuano text.

Conditions			Distribution			
Size	LM	EM	Corr.	Incor.	Miscor.	Noncor.
500	3-gram	SC	103	39	35	53
500	3-gram	MC	90	18	36	65
500	6-gram	SC	102	63	40	49
500	6-gram	MC	102	37	37	52
1,000	3-gram	SC	105	40	35	51
1,000	3-gram	MC	91	9	34	66
1,000	6-gram	SC	106	47	34	51
1,000	6-gram	MC	102	21	32	57
27,363	3-gram	SC	113	24	32	46
27,363	3-gram	MC	99	5	31	61
27,363	6-gram	SC	125	41	31	35
27,363	6-gram	MC	120	22	23	48

Table 4.10: Distribution of words in post-correction English OCR output on Cebuano text.

better than the Single Character Error Model. And, interestingly, the 3-gram language model performs better than the 6-gram model, except for the largest training data and Multi-Character Error Model combination. Both differences are most likely caused by the implications of using a dictionary as discussed above.

4.4.3 Arabic: Acquiring Parallel Text

We used Arabic to illustrate conversion from hardcopy to electronic text for a widely available parallel text, the Bible [63, 100, 115]. We divided the Bible into ten equal size segments, using the first segment for training the error model, the first nine segments for the language model, and the first 500 verses from the last segment for testing. Since diacritics are only used in religious text, we removed all diacritics. The OCR data was generated using a commercial Arabic OCR system.[6] Note that this evaluation differs from Igbo and Cebuano, as the experiments were performed using an existing *native* OCR system. It also allowed us to evaluate chunking (See Section 4.3.3), as Arabic data tends to have more word merge/split errors owing to the connected nature of Arabic script.

Table 4.11 shows the correction performance for Arabic under various conditions. Corresponding word distributions are presented in Table 4.12. The differences between the original OCR output and all the correction results are statistically significant. The *Limit* column lists the maximum number of errors per token allowed and the *Merge/Split* column indicates whether correction of word merge/split errors is allowed. We achieve significant reductions in WER for Arabic. The first

[6]Sakhr Automatic Reader Version 6.0

Conditions			Results	
Merge/Split	LM	Limit	WER (%)	Red. (%)
no	3-gram	2	22.14	10.33
no	6-gram	2	17.99	27.14
yes	3-gram	2	18.26	26.04
yes	**3-gram**	**4**	**17.74**	**28.15**
yes	5-gram	2	20.74	16.00
Original OCR Output			**24.69**	-

Table 4.11: Post-correction WER for Arabic OCR output on Arabic text.

Conditions			Distribution			
Merge/Split	LM	Limit	Corr.	Incor.	Miscor.	Noncor.
no	3-gram	2	209	30	241	1140
no	6-gram	2	390	58	396	804
yes	3-gram	2	574	106	318	698
yes	3-gram	4	641	115	341	608
yes	5-gram	2	740	171	340	510

Table 4.12: Distribution of words in post-correction Arabic OCR output on Arabic text.

two rows show that the 6-gram language model performs much better than the 3-gram model. However, higher order n-grams perform worse when we allow word merge/split errors. Note that for handling word merge/split errors we need to learn the character distributions within lines, rather than within words as we normally do. Consequently, more training data is required for reliable parameter estimation. Handling word merge/split errors improve the performance, which is expected assuming the input data contains these errors. Allowing fewer errors per token reduces the performance, since it is not possible to correct words that have more character errors than the limit. Unfortunately, increasing the error limit increases the search space exponentially, making it impossible to use high limits. As mentioned in Section 4.3.2, iterative correction is a way to address this problem.

4.5 Error Analysis

Both for Igbo and Cebuano, almost all the corrections performed by post-processing involve language-specific characters that are not handled properly by the non-native OCR system. The effect of lexicon-free correction method presented here is insignificant or negative for languages such as Spanish, for which the native OCR system's performance is good and there are no unfamiliar characters to confuse the system. We should, however, note that our lexicon-based method is able to provide improvements for these languages as discussed in Section 3.4.3.

Unfamiliar characters are not a problem in case of Arabic, as we used a native OCR system. Post-processing was still effective mainly because native Arabic OCR

performance is relatively low compared to Latin script-based OCR, leaving enough room for improvement. This analysis underlines the fact that post-processing does not replace OCR development, but rather provides a method of improvement until OCR systems reach a certain performance level.

4.6 Language Specific Handling

Both lexicon-based and lexicon free correction models are for the most part language independent and can be used to process any language provided that training data is available. The only step that is explicitly dependent on the language is the word-to-character conversion step of the lexicon-based model. Although the system learns the parameters of this step through training, we assume that the structure of the model is known for each language. For example, we know beforehand languages that use a Latin script insert spaces between words, and we encode this fact into the model.

While other steps of our model are conceptually language-independent, in practice each language will require some special handling. For example, languages such as Chinese and Japanese where the character set is very large makes current implementation of our model virtually useless as the search space becomes prohibitively large. We would need to address this issue before we could process either language.

Even for languages for which there is no such prohibitive factor preventing the applicability of the methods, correction performance can benefit greatly from language specific tweaking. As an example, agglutinative languages would cause the

performance of the lexicon-based model to degrade since it is much more difficult to obtain a lexicon with good coverage. Although the current model would still be able to process them, the results would not be as good as that obtained for languages with simpler morphology. It would be beneficial to adapt the language model component to take the agglutinative nature of the language into account. Another example is languages with complicated orthographies. One good example is Arabic, where the shape of the characters change depending on their position withing the word. As we have demonstrated, correction performance is still good even when we ignore this aspect of the language. However, we would expect the results to improve if the orthography of Arabic was accommodated.

Note that, in both cases, there is nothing inherently insufficient about our model that prevents addressing these issues. For example, the word-to-character conversion step can easily emit different symbols for the same Arabic character depending on their position. Similarly, the character-based model can be modified to do the same at the character generation step. For any language that we attempt to process, it is likely that language specific issues exist that would need special attention. For languages that do not have significantly different characteristics than the ones we have explored so far, we expect to obtain correction performance levels that are similar to that which we have presented. However, we should emphasize that it is difficult to predict the performance of the correction system for a language that is being processed for the first time. Some languages might require a significant amount of tailoring to achieve any accuracy improvement. It is worth noting that this is not a limitation specific to our correction model. Portability across languages

and assumptions about properties of languages are an issue for almost all NLP tasks. As an example, current state-of-the-art Statistical Machine Translation systems are not able to handle languages with rich morphology or free word order as well as they handle languages that have a similar structure to English.

4.7 Extrinsic Evaluation on MT

We evaluated the impact of the new post-processing model on NLP applications using Arabic-English MT. We only evaluated an SMT system, as we did not have access to a symbolic MT system for Arabic. We used the same data that we used for Arabic OCR experiments reported in Section 4.4.3. We generated a parallel version of the text by aligning an English Bible with the Arabic Bible at verse level. The language and translation models were trained on the first nine sections of the Bible data while translation evaluations were performed on the 500 test verses. We used a state-of-the-art phrase-based SMT system, Pharaoh [66]. The parameters of the system are optimized using Minimum Error Rate Training (MERT) [102]. The last 500 verses of the training data are set aside as the reference set for MERT and therefore were not used for training the system itself. The language model was trained using SRI Language Modeling Toolkit (SRILM) [131].

Table 4.13 shows the BLEU scores for OCRed, corrected, and clean text. The absolute BLEU scores are lower than typical results reported in literature mainly because there is only one reference translation per sentence. For comparison, when a reference translation set in 2003 Arabic-English MT Evaluation corpus [81] is

evaluated against a single reference set from the same corpus, the BLEU scores are as low as 0.2777. The same set has a BLEU score of 0.51 when evaluated using three other reference sets.

Input Text	BLEU Score
OCR	0.0666
Corrected	0.0788
Clean	0.0920

Table 4.13: BLEU scores for English translations from OCRed, corrected, and clean Arabic input using Pharaoh

Although the absolute scores are lower, the impact of OCR errors and correction is similar to the impact observed for Spanish. Note that our interest is in the relative performance of clean and noisy text regardless of the absolute performance levels.

4.8 Summary

Reliance on a lexicon is a serious limitation for a low-density language, as it is unlikely that a lexicon with good coverage, if any at all, will be available for such a language. In this chapter we presented a simplified version of our statistical post-processing method for OCR error correction that requires minimal resources, designed particularly for low-density languages.

We rigorously evaluated our approach using real OCR data, and have shown that we can achieve recognition accuracy lower than that achieved by a trainable OCR system for a new language. For Igbo, a very low-density language, adapting

English OCR achieved relative error reductions as high as 78%, resulting in 7% WER. We also showed that the error rate of a trainable OCR system after training can be further reduced up to 50% using post-processing, achieving a WER as low as 4%. Post-processing experiments using Cebuano validate our approach in a dictionary-acquisition scenario, with a 51% relative reduction in error rate from 8% to 4%. Evaluation on Arabic demonstrated that the error rate for a native commercial OCR system can be reduced by nearly 30%.

The correction techniques we presented in this chapter and in Chapter 3 gain leverage from existing OCR systems, which enables the following:

- Minimal-labor adaptation of systems to new low-density languages
- Improvements in native OCR performance

The proper role of our post-processing approach depends on the language. For languages with little commercial potential for OCR, it may well provide the most practical path for language-specific OCR development, given the expensive and time consuming nature of OCR development for new languages and the "black box" nature of virtually all state-of-the-art OCR systems. For languages where native OCR development is more probable, it is a fast, practical method that allows entry into a new language until native OCR development catches up. For these languages and for languages where native systems exist, we show that post-processing can yield improvements in performance.

While resource acquisition is an actively researched area, the focus has been mostly on acquiring annotations, as presented in Chapter 2. The question of elec-

tronic text availability is mostly ignored, as it is not an issue for the set languages that dominate NLP research. However, when it comes to low-density languages even electronic text becomes a scarce resource. We have studied the feasibility of cross-lingual utilization of an OCR system for the first time, and shown that it is possible via post-processing.

OCR post-processing, and text error correction in general, is a widely studied area. Unfortunately, there is no standard evaluation benchmark to allow direct comparison, and correction system implementations are seldom available. Therefore we compare our correction work to existing work qualitatively, rather than quantitatively as we would like. There are very few models that address word merge/split errors, since it is less of an issue for the Latin script. However, as we move to other scripts, especially cursive ones, ability to handle merge/split errors becomes important. Both our word-based and character-based models can correct merge/split errors. Ability to correct these errors improves performance as our evaluations demonstrate. Another important issue is the use of a lexicon. A majority of the correction models require lexicons to perform error detection and correction, which makes them practically useless for low-density languages. Our character-based correction model does not require a lexicon, and unlike other character-based correction work reported in the literature, still allows correction of word merge/split errors using a novel chunking method to keep the problem tractable.

One limitation of our approach is its reliance on an existing OCR system that supports the script of the language of interest. Trainable OCR systems are the only option if there is no OCR system that supports the script of interest; however,

training an OCR system from scratch is usually a tedious and time consuming task. Post-processing can be used to reduce the training time and to improve recognition accuracy by aiding the generation of more training data once a basic recognition capability is in place.

Chapter 5

Get the Knowledge: Acquiring Annotations

5.1 Motivation

The post-processing methods presented in Chapter 3 enable us to utilize existing OCR systems to acquire electronic corpora for low-density languages. The next step for porting statistical NLP systems to a new language is to acquire annotated data to train on. Manual annotations usually do not occur naturally and are labor intensive to generate. Luckily, a considerable amount of annotated data is already generated for well-studied languages. Furthermore cross-language utilization of the existing data can provide the initial boost to allow basic processing for new languages.

Parsing provides an excellent platform to evaluate cross-lingual utilization of training data. Currently, state-of-the-art parsing performance is obtained using corpus-based statistical parsers. Since statistical parsers are simply trained on sample parse trees, porting these parsers to new languages is, for the most part, simply a matter of finding a treebank to train on. Unfortunately, treebanks are not naturally occurring language resources, and constructing a treebank is a long and costly process. For example, Penn Chinese Treebank Version 2, which became available two years after the project started, has only about 4,000 parse trees. Version 4 was released three years after Version 2, and contains about 15,000 parse trees.

The problem of insufficient training data for parsing is an actively researched area. Numerous methods that use active learning and weakly supervised algorithms have been developed to reduce the amount of training data required for parser training [6, 50, 56, 104, 124, 130, 135]. While these approaches require a smaller amount of training data, they do not eliminate the need for training data completely.

Our approach is to use the treebanks and parsers that are already available for well-studied languages (e.g. English) to infer parse trees for less-studied languages for which no treebank is available. We developed a projection algorithm that takes a parallel corpora and dependency trees for one side of the corpora as input to infer the dependency trees for the other side of the corpora. The projection algorithm relies on parallel text and word level alignment to project the syntax trees. The advantage of this approach is that parallel text is a resource that is more readily available compared to parse trees. Many institutions such as newspapers, international companies and organizations, and governments generate parallel text during the course of their normal operations. Word-level alignment between two languages can be computed using well-studied unsupervised methods. Consequently, using projected parse trees is a feasible and much faster method to obtain a treebank for a new language.

The inferred treebank can then be employed to train a statistical parser, making it possible to parse a new sentence that does not have a parallel sentence with a parse tree. We show that a reasonable quality treebank can be generated by combining direct projection with minimal language-specific post-processing. The performance achieved by a parser trained on the inferred treebank is comparable to

a parser trained on a manually generated treebank that took a couple of years to produce.

Figure 5.1 illustrates the end-to-end process starting with a parallel corpus and a treebank, potentially generated using a parser, for one side of that corpus. The end product is a trained statistical parser for the target langauge that can be used to parse new sentences without requiring parallel sentences in the source language.

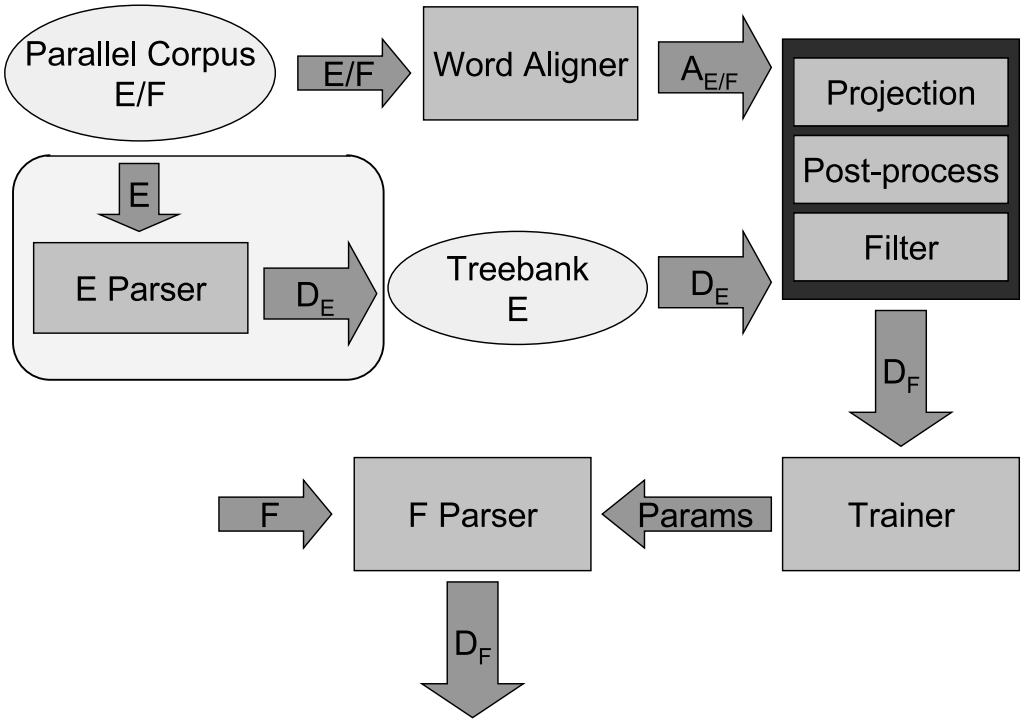

Figure 5.1: Overall parsing pipeline going from parallel corpus and source dependency trees to a trained statistical parser for the target language.

5.2 Background

Projecting annotation for one language across parallel text into another language has been used for tasks such as POS tagging, NP chunking, and verb classification.

Given some parallel text, one side of which is in a resource-rich language and the other side in a resource-poor language, the basic idea is to perform annotations on the resource-rich side and then project them over to the resource poor side using word-level alignments. Once the projection is done, annotated data can be used to train or bootstrap applications in that resource-poor language to annotate new data that is not part of a parallel text collection. Validity of the idea has been shown [60], and it has been successfully employed for word sense disambiguation [24], part-of-speech tagging [154], parsing [57], and so on.

Although not always explicitly stated, all projection algorithms assume syntactic properties and relations are preserved through the translation process. POS projection assumes words are translated into words with the same POS, NP chunk projection assumes chunks are translated together as units, and so on. The same correspondence assumption is also applied to higher level syntax relations such as parse trees. This assumption is the underlying principle behind several cross-language syntactic models as well [4, 32, 39, 91, 128, 145, 150]. As our treebank projection algorithm relies on the same assumption, before describing the actual algorithm, we first present the assumption that forms its basis.

5.2.1 Direct Correspondence Assumption

Informally, Direct Correspondence Assumption (DCA) can be explained as follows: If two sentences are translations of each other, then syntactic relations of these sentences also directly map to each other. Formal definition for the DCA, as given

in [60], is the following[1]

> Given a pair of sentences S and T that are (literal) translations of each other with syntactic structures D_S and D_T, if nodes x_S and y_S of D_S are aligned with nodes x_T and y_T of D_T, respectively, and if the syntactic relationship $R(x_S, y_S)$ holds in D_S, then $R(x_T, y_T)$ holds in D_T.

Note that the DCA does not imply anything regarding the actual syntax formalism used to generate the syntax trees. For a dependency tree, the relation $R(x, y)$ would be a head-modifier relation. Whereas, a constituency tree would require using a different relationship, probably sisterhood.

The DCA assumes literal translations, as more liberal translations may change the sentence structure drastically. However, it does not always hold even for literal translations. The main reason for a failure to occur with DCA is translational divergences [29, 47], which can be described as the use of diverse syntactic mechanisms by different languages to express the same meaning. For example, the English phrase "swim across" would translate to "yüzerek geç" (cross by swimming) in Turkish; "carrot cake" would translate to "gâtau de carotte" (cake of carrot) in French. Even for these simple examples, direct projection would result in incorrect dependencies.

While the syntactic structures of the languages differ considerably, their head-complement ordering is generally stable. For example, in Turkish the head of a noun phrase is almost always to the right of its modifiers. In English, prepositional phrases always appear to the right of the head word they modify. Using these systematic,

[1] We slightly modified the notation to match the notation used in this dissertation.

language-specific preferences can significantly improve the language-blind projection of Dependency Projection Algorithm (DPA).

5.2.2 Dependencies versus Constituencies

The very first choice that needs to be made regarding treebank projection is whether we will project dependency or constituency trees. Both trees encode similar syntactic relations. Furthermore, it is possible to automatically convert one type of tree into the other type [51, 82, 148]. Dependency trees have several advantages over constituency trees from a projection perspective, which led us to design our projection algorithm around dependency trees.

First and foremost, dependency relations are much closer to thematic relations. As thematic relations basically describe "who did what to whom", they are very likely to stay the same when translated into a different language. This implies that dependency relations are also more likely to be preserved.

Second, dependency relations are independent of the surface word order. As different languages have different word orders, keeping syntactic relations and surface word order independent makes projection much easier. For example, if we are projecting from English to French, the relative ordering of adjectives and the nouns that they modify would change. While dependency trees are not affected by this order change, constituency trees are, and therefore would require structural reorderings to accommodate this surface level change, as noted in MT literature [34].

Third, even without surface order issues, constituency trees are usually deeper structures with more nodes, which makes projection more complex compared to dependency trees.

5.2.3 Parallel Corpora

One of the prerequisites for any projection approach is availablility of parallel corpora. For well-studied languages, acquiring parallel corpora is relatively easy. There are numerous sources of corpora that generate parallel text continuosly such as multinational organizations (e.g. UN, European Union), corporations, and newspapers. Low-density languges, however, usually lack readily available parallel corpora. One solution is to use widely translated texts such as the Bible, the Quran, and the Book of Mormon [e.g. 115]. These books are carefully and literally translated, which is a desirable property for projection. However, their sizes are relatively small, and their style is often not representative of the contemporary style of the langauge they are translated into. Another option is to employ an elicitation approach and actively generate parallel corpora. For instance, [153] presents an automated elicitation system for acquiring translation examples using volunteers across the globe.

5.3 Dependency Projection Algorithm

The DPA algorithm is based on the DCA. As a matter of fact, it is a direct application of the DCA for the dependency projection task, and therefore can be used to evaluate the validity of the assumption [60].

While DPA directly follows DCA, the fact that DCA is not always valid makes converting it into an algorithm challenging. For any given source-target language pair, there will be numerous cases where DCA does not hold. Word alignments that are not one-to-one further complicate the projection process. As each language pair requires linguistic decisions specific to that particular pair, we designed the DPA to be language-blind. The goal is to defer any decisions that would require lingustic knowledge so that the projection algorithm can be free of language-specific logic. Furthermore, all available linguistic information is preserved, either in the structure of the tree using special nodes or in the auxiliary information field of each node, so that any linguistic processing step down the line can make full use of them. For example, in case of a one-to-many alignment, DCA cannot be applied directly since multiple words exist on the target side. An optimal way to address this problem would be to find a word among the target set of words that can act as the head of the whole set. Unfortunately, doing so requires lingustic knowledge about the target language. Therefore, we introduce a placeholder node, dubbed $\mathcal{M}$, in lieu of the actual head node and leave the actual decision to a language-specific post-processing step.

The input to the DPA is a source sentence S, a target sentence T, a word level alignment A between the two sentences, and a dependency parse tree D_S for the source sentence. The output of the algorithm is the inferred dependency tree D_T for the target sentence.

5.3.1 Definitions

Before we go into the details of the algorithm we first give the definitions of the concepts that we use.

Sentences and Their Alignment

- Source sentence: $S = \langle s_1, s_2, \ldots, s_m \rangle$

  s_i is a single token, which can be a word, number, punctuation, etc. The exact definition of a token is language and/or application specific. However, the exact definition is not crucial for the DPA algorithm. We simply follow the tokenization conventions of the source parse trees.

- Target sentence: $T = \langle t_1, t_2, \ldots, t_n \rangle$.

  The exact counterpart of source language S for the target language.

- Word level alignment between S and T: $A = \{a_1, a_2, \ldots, a_k\}$

  - Alignment links: $a_i = \langle p, q \rangle$, where $1 \leq p \leq m, \quad 1 \leq q \leq n$

    Indicates that source word s_p and target word t_q are aligned to each other. Both s_p and t_q can take part in other alignment links as well.

  - $\forall_p 1 \leq p \leq m | \big(\forall_{q,q'} q, q' \in SrcAln(p, A) | TrgAln(q, A) \equiv TrgAln(q', A) \big)$

    If a source word p is aligned with multiple target words, none of those target words can have an alignment link with another source word p' unless all other target words aligned with p are also aligned with p'.

This restriction has its roots in the basic intuition behind the DCA. All languages use similar concepts that are universal, but the concepts are represented differently on the surface.[2] The dependency relations are in effect relations between concepts in the sentence, disguised as syntactic relations between the words representing those concepts. Similarly, word level alignments between words are an approximation of the concept level alignments. What we are really aligning are the concepts, but since they are hidden behind the words, we align words instead. Since we assume the concepts are shared, there is a one-to-one correspondence between the concepts in a pair of parallel sentences. If a concept is represented by multiple words on either side, it implies that all the words representing that particular concept should be aligned with all the words representing the corresponding concept in the other sentence. Furthermore, a word is part of only one distinct concept for a given sentence, although it can represent different concepts in different sentences. Consequently, a word cannot take part in two distinct alignment groups that link two distinct pairs of concepts.

Dependency Trees

- Dependency tree: $D = \{N, R, L\}$

D_S represents the source dependency tree and D_T represents the Target de-

[2]Universality of concepts is not completely true. However, we assume universality to be the case as far as DCA is concerned.

pendency tree.

- Dependency nodes: $N = \{n_1, n_2, \ldots, n_u\}$

  N contains a node representing each token in S, and potentially more nodes that do not have a corresponding token in S.

- Dependency relations: $R = \{r_1, r_2, \ldots, r_v\}$, where $r_i = (n_p, n_q, l_k)$

  A dependency relation $r_i = (n_p, n_q, l_k)$ means that node n_p modifies node n_q, and their relation has the label l_k.

- Relation labels: $L = \{l_1, l_2, \ldots, l_z\}$

  Relation labels l_i denote the type of relation between a node and the node it modifies (e.g. subject, object, determiner). The set of labels and their definitions are parser/treebank dependent. The DPA is indifferent to the contents of L.

- Governors and heads: In a dependency relation, the node that is being modified is called *governor* and the node that modifies it is called *modifier*. In a properly formed linguistic phrase, all nodes except one are governed by another node in the same phrase. The only node whose governor is outside the phrase is called the *head* of the phrase.

5.3.2 Projection over Various Alignment Cases

In the following sections, we first go over each alignment case, and describe how a dependency relation is projected for that particular case. We demonstrate each case

with an example of a projection of a dependency relation from English onto French or vice versa.

One-to-One

This is the ideal case, where mapping is straightforward and correct, assuming the alignment is correct and the DCA holds. As can be seen in Figure 5.2, for each dependency relation on the English side, we simply create a corresponding relation of the same type on the French side. For example, since '`the`' is a determiner for '`house`', '`le`' which replaces '`the`' should be a determiner for '`mason`' which replaces '`house`'.

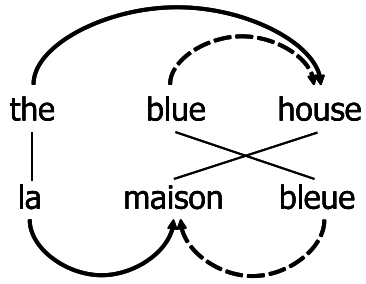

Figure 5.2: Dependency projection for a one-to-one alignment

One-to-Many

If a source word is mapped to multiple target words, direct mapping may no longer be possible. If the source word is the governor of a dependency relation, it would mean that the projected modifier on the target side needs to modify multiple words, which is not possible as a word can only have one governor. Figure 5.3 illustrates the problem, where direct projection causes '`fraîches`' to have three governors.

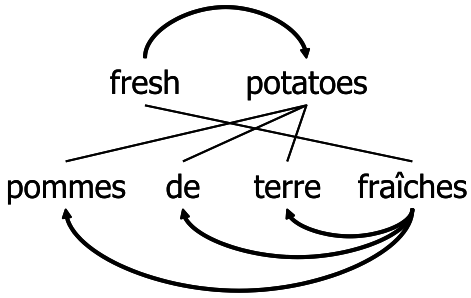

Figure 5.3: Invalid dependency projection for a one-to-many alignment

Alignment of a single word to multiple target words implies that the set of target words, as a unit together, perform the same function as the source word. Therefore, we can treat them as a single unit, which reduces the problem to a one-to-one mapping. However, we need to address a few issues in order to achieve this. First, we need to pick a single node to represent the whole set, and second we need to tie the other words in the unit to the overall structure.

Linguistically, the correct solution is to view the set of words on the target side as a phrase. This would allow us to determine both the head, and the internal relations that would tie the rest of the words together. Unfortunately, this solution requires language-specific processing. We decided to avoid any language specific processing, and, therefore, we introduce a new dummy node instead. This new node is treated as if it is the head word of the phrase. All target words in the set are treated as modifiers of the new node, and are linked to it with a special dependency relation that we term Multi-Aligned Constituent (MAC). Figure 5.4 illustrates the process. '`pommes de terre`' is treated as a single unit that corresponds to '`potatoes`', and is represented by the newly introduced node $\mathcal{M}$.

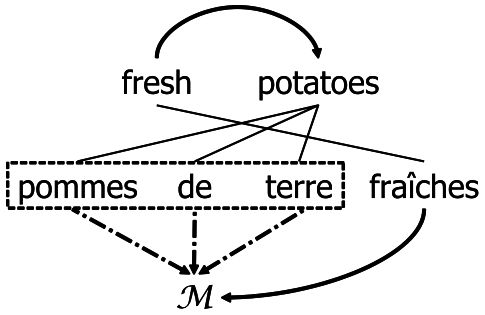

Figure 5.4: Dependency projection over a one-to-many alignment

If the situation is reversed, where the source word is the modifier of a relation, direct projection can be applied without causing an illegal dependency tree. The result would be multiple target words that all modify the same head. However, we opt to treat both cases the same for simplicity.

Many-to-One

Due to the asymmetric nature of the projection process, a many-to-one case is not simply the opposite of a one-to-many case. Just like a one-to-many case, we need to treat the set of words that are aligned together as a single unit. The difference is, since we have the dependency tree for the source sentence, we can actually determine the head of the phrase, and use it as the unique node that represents the whole set. The dependency relations within the set cannot be projected over to the target language, as there is only one node corresponding to the whole set. Instead of discarding that information, we store those relations in the auxiliary data component of the target node, marked as internal links. Figure 5.5, which is the opposite of Figure 5.4 illustrates two points. First, it illustrates how the projection is done.

Second, it illustrates how one-to-many and many-to-one projection cases are not simply mirrored images of each other. The dependency relations on the source side, which is now French rather than English, allows us to determine the head of the phrase '`pommes de terre`'. Hence, we treat the whole dotted box as if it contains only the head word and apply one-to-one projection.

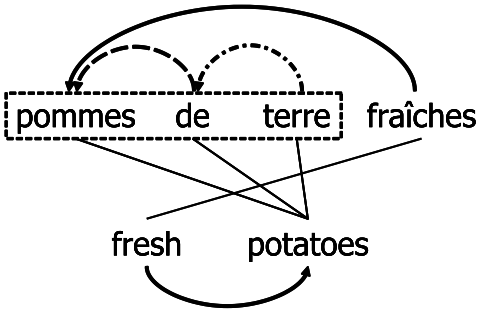

Figure 5.5: Dependency projection for a many-to-one alignment

Although ideally it should not happen, there is another potential problem. If the set of words that are aligned together do not form a proper sub-tree with a single head, then projection is not possible. To address this problem, we pick one of the alternate heads as the head of the set, and the remaining heads are added to the auxiliary data of the node as alternate relations. This allows users of the projected tree to access, if they so desire, both the head nodes and the respective dependency relations that are omitted from the projected tree.

Handling the links that come into the set is simpler. Any dependency relation whose head is one of the words in the set will use the single word on the target side as its head when it is projected to the target side. As multiple words can have the same head, this does not cause any problem.

Many-to-Many

Similar to previous cases, many-to-many links can be converted into one-to-one by viewing sets of words on each side of the alignments as a single unit. We perform the projection in two steps. First, we treat the problem as if it is a one-to-many alignment, generating an $\mathcal{M}$ node to represent the whole set. Once the $\mathcal{M}$ node is in place, we continue processing as if we are dealing with a many-to-one alignment, where the target side has only one word represented by $\mathcal{M}$. For example, in Figure 5.6 '`motion picture`' is mapped as a whole to '`film cinématographique`'. We treat each phrase as a single unit for projection. On the English side, '`picture`' is the head of the phrase, so it represents the whole phrase. On the French side the newly introduced $\mathcal{M}$ node represents the corresponding French phrase. The projection is carried out as if we have a one-to-one alignment between picture and $\mathcal{M}$. Note that many-to-many alignments assume that two sets of words on each side are aligned exclusively and completely to each other, as discussed in Section 5.3.1.

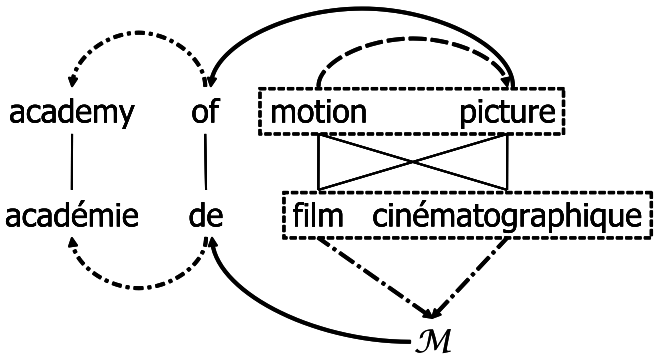

Figure 5.6: Dependency projection for a many-to-many alignment

Unaligned Source

Dependency relations involving unaligned source words cannot be projected to the target side since there is no corresponding word to take part in the projected relation. The obvious solution is to ignore any dependency relations that involve unaligned words. However, rather than discarding this piece of information, we prefer to carry it over to the target side, so that it can be exploited by later linguistic processing. We introduce a new node dubbed $\mathcal{E}$ into the the target dependency tree, representing the missing counterpart of the unaligned source word. Once the $\mathcal{E}$ node is in place, the source word is no longer unaligned, and we can process it as usual, as illustrated in Figure 5.7.

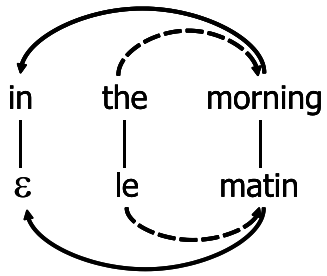

Figure 5.7: Dependency projection for an unaligned source word

Unaligned Target

Unaligned target words pose the biggest challenge since there is no information available for them. We do not have information on the target side other than the surface order. We cannot project any information from the source side as there is no alignment on which to base our inference. The only way to process these nodes

is to use language-specific linguistic knowledge on the target side, which is outside the scope of this algorithm. Consequently, any unaligned word on the target side is represented by a dependency node that only acts as a placeholder, and does not provide any syntactic information other than the surface form of the word and its location. Figure 5.8, provides an example case where the word '`in`' on the English side is left unprocessed as there is no corresponding word for it on the French side.

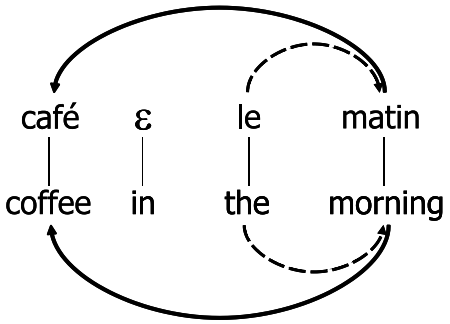

Figure 5.8: Dependency projection for an unaligned target word

5.3.3 Data Structures for DPA

The DPA algorithm does not require any specialized data structures. The data structures presented in this section are all basic concepts and do not refer to any particular instantiation of those concepts. They are presented mainly to clarify the notation used in algorithm definitions.

Sentences and Their Alignment Sentences are represented using an associative array of tokens, where token labels are the key. Consequently, each token in a sentence has a unique label. We use the position of the token in the sentence as

its label for convenience. In addition to the label, each token also holds the actual character string for the token.

Alignments are represented as a list of label pairs, representing the source and target side tokens of each alignment link. We make use of two functions, `SrcAln(i, A)` and `TrgAln(j, A)`, to compute the set of target (source) tokens aligned with a particular source (target) token, respectively. The two functions are the same functions used in Section 3.5.1, and formally defined as follows

$$SrcAln(i, A) = \{j \mid \langle i, j \rangle \in A\}$$

$$TrgAln(j, A) = \{i \mid \langle i, j \rangle \in A\}$$

Clusters We make use of a node cluster to handle alignments involving multiple nodes on the source side. A set of source nodes that are aligned together to one or more target nodes are called a source cluster; target clusters are their counterpart on the target side. Clusters are represented using a simple container with the following fields:

- `members`: The list of the labels of nodes that are part of this cluster.
- `governor`: The label of the node that is selected to be the governor of the cluster. The cluster governor is selected among the governors of the member nodes.
- `governors`: The list of the governors of the member nodes. This field is used to score and sort the member node governors during cluster governor selection.

Dependency Trees Dependency trees are represented using an associative array of dependency nodes, where node labels act as keys, so $D[l]$ returns the dependency node in D whose label is l. Dependency nodes are simple containers with the following fields

- `label`: Each node in a dependency tree has a unique label. Tokens of the words in a sentence and corresponding nodes in the dependency tree for that sentence use the same label. A reference to a dependency node that treats the node as if it is a sentence token should be viewed as a reference to the sentence token with the same label.

- `PoS`: Part-of-Speech for the token represented by the node.

- `governor`: The label of the node that is modified by this node. There should be a node with this label in the tree. This field may be empty since not all nodes modify another node.

- `relation`: The dependency relation between this node and its governor.

- `aux`: An associative array containing any piece of information that is deemed useful. We use this field to store alternative governors and relations, antecedents, the root form of the words represented by the node, etc. Each piece of auxiliary information is represented by a unique key. Multiple values can be stored for each key.

5.3.4 The Algorithm

The DPA is presented in Figure 5.9. We start with an empty target tree, and first call the procedure `ProcessSourceClusters()` to process source nodes that are aligned together to one or more target nodes. We refer to these source and target nodes as clusters. The `ProcessSourceClusters()` procedure identifies the source clusters and selects a governor node for each cluster. Lines 3-15 in Figure 5.9 perform the actual projection of dependency relations. For each node in the source tree, we first find the label of the corresponding target tree node using the function `GetProjectedNodeLabel()`. If the current node is aligned to multiple target nodes, Lines 5-10 create the nodes for the target cluster and links them all to the $\mathcal{M}$ node. The links between the nodes of the target cluster and the $\mathcal{M}$ node are labeled as a MAC relation. The $\mathcal{M}$ node acts as the corresponding target node for the current source node. Once we complete the projection of the node label, we extract the information for the projected governor using the function `GetProjectedGovernorInfo()`. At this point, we have all the information we need to create a node in the target tree, which is a projection of the current source node. If a target node is aligned with multiple source nodes, multiple projected nodes will be generated and added to the target tree for the same target node. Procedure `AddNode()` handles the proper merging of multiple projections onto the same target node. Finally, we complete the projected target tree by inserting a node for all target sentence tokens that are not aligned to any source words.

Procedure `ProcessSourceClusters()`, which is given in Figure 5.10, identifies

```
Input: Target sentence T, Source tree D_S, Alignment A
Output: Target tree D_T
1  D_T ← ∅;
2  C ← ProcessSourceClusters(D_S, A);
3  foreach Node n ∈ D_S do
       // First find the mapping for Node n
4      tLabel ← GetProjectedNodeLabel(n.label);
5      if |SrcAln(n.label, A)| > 1 then
6          foreach Label l ∈ SrcAln(n.label, A) do
7              m ← AllocateNewNode(l, n.pos, "MAC", tLabel);
8              AddNode(D_T, m);
9          end
10     end
       // Then find the mapping for its governor
11     (govLabel, relation, auxLabel, auxValue) ←
       GetProjectedGovernorInfo(C, n.governor);
12     m ← AllocateNewNode(tLabel, n.pos, relation, govLabel);
13     Append(m.aux[auxLabel], auxValue);
14     AddNode(D_T, m);
15 end
   // Finally, add unaligned target words
16 foreach Token w ∈ T do
17     if w.label ∉ D_T then
18         m ← AllocateNewNode(w.label, nil, nil, nil);
19         AddNode(D_T, m);
20     end
21 end
22 return D_T;
```

Figure 5.9: Dependency Projection Algorithm

```
   Data: Source tree D_S, Alignment A
   Result: Source node clusters, C
   // First identify the clusters
 1 C ← ∅;
 2 foreach Node n ∈ D_S do
 3 |   if ∄ c ∈ C | n.label ∈ c then
 4 |   |   if SrcAln(n.label, A) ≡ ∅ then
 5 |   |   |   sLabels ← {n.label};
 6 |   |   else
 7 |   |   |   tLabel ← f, for any f ∈ SrcAln(n.label, A);
 8 |   |   |   sLabels ← TrgAln(tLabel, A);
 9 |   |   end
10 |   |   c ← new Cluster;
11 |   |   c.members ← sLabels;
12 |   |   C ← C ∪ {c};
13 |   end
14 end
   // Then score and sort member governors for each cluster
15 foreach Cluster c ∈ C do
16 |   govScores ← ∅;
17 |   foreach Label l ∈ c.members do
18 |   |   governor ← D_S [l].governor;
19 |   |   while governor ∈ c.members do
20 |   |   |   governor ← D_S [governor].governor;
21 |   |   end
22 |   |   if governor ∈ c.governors then
23 |   |   |   increment govScores[governor];
24 |   |   else
25 |   |   |   c.governors ← c.governors ∪ {governor};
26 |   |   |   govScores[governor] ← 1;
27 |   |   end
28 |   end
29 |   Sort c.governors in descending govScore order;
30 end
   // Select a cluster governor for each cluster
31 if SelectGovernors(C, C) = false then fail;
32 return C;
```

Figure 5.10: Procedure `processSourceClusters(D, A)`: Process source words that are aligned together.

all clusters in $\mathsf{D_S}$ and selects a common governor for the whole cluster among the governors of cluster members. The selection of a common governor is necessary since clusters act as a single unit for projection, and consequently can only have one governor projected. Lines 2-14 find and initialize the clusters. Line 5 creates a single member cluster for unaligned source tokens. Lines 7 and 8 find the labels of all source tokens that have the same alignment target and, therefore, should be in the same cluster. All nodes that are included in the cluster are aligned with the same set of target words as imposed by the definition of A provided in Section 5.3.1. Lines 15-30 fill the `governors` field of each cluster and compute a score for each label in the `governors` field. The score of a governor is simply the number of nodes in the cluster that modifies the governor. The governor of the cluster needs to be a node outside the cluster. Therefore, we follow the governor links until a non-member node is reached. After the `governors` field is computed, its contents are sorted in descending score order, placing the governor with the highest score at the head of the list. After all the clusters are processed, `SelectGovernors()` procedure is called to pick a governor for each cluster without causing any cycles in the resulting dependency graph.

Procedure `SelectGovernors()`, presented in Figure 5.11, recursively processes the clusters, selecting a governor for each cluster until no cluster is left without a governor. Line 1 checks if all clusters are processed. If so, this indicates that the recursion should be terminated. If there are more clusters to process, the system selects one of those clusters for processing. Lines 3-8 go over the candidate governors in the `governors` field in order. Since the `governors` field is already sorted based

```
   Input: Cluster set C, Clusters to process P ⊂ C
   Result: For each cluster in C a governor is selected, without causing cycles
1  if P ≡ ∅ then return true
2  cluster ← c, for any c ∈ P;
3  foreach governor ∈ cluster.governors do
4  |   cluster.governor ← governor;
5  |   if cluster.governor did not create a cycle then
6  |   |   if SelectGovernors(C, P − {cluster}) = true then return true ;
7  |   end
8  end
   // Failed to find a governor that does not cause a cycle
9  return false ;
```

Figure 5.11: Procedure `selectClusterGovernors(C, P)`: Selects a governor for source words that are aligned together

on score, selection starts with the highest scoring governor candidate. The selection procedure is greedy, choosing the best option at each step, so the resulting selection is not guaranteed to have the highest possible total score. If selecting the current candidate as the governor of the cluster does not cause a cycle, a recursive call is made to select the governors for the remaining clusters. The test for cycles is the standard directed graph cycle test, so we do not go into details here. If the remaining clusters are successfully processed, the selection process is completed. If we fail to find a cycle-free selection of cluster governors, we abandon our current selection for the current cluster, and try again with the next candidate governor. If there is no selection of governors that results in a cycle-free graph, then the DPA fails at Line 31 of procedure `ProcessSourceClusters()`.

Function `GetProjectedNodeLabel()`, shown in Figure 5.12, is used to generate projected node labels in a well-defined way, allowing the labels to be used before the actual node with that label is created. Case 0 is for unaligned source nodes.

```
Input: A source node label sLabel
Output: Projected target node label tLabel
1  switch |SrcAln(sLabel, A)| do
2      case 0
3          tLabel ← "E";
4          Append(tLabel, nLabel);
5      end
6      case 1
7          tLabel ← l, for only l ∈ SrcAln(n, A)
8      end
9      otherwise
10         tLabel ← "M";
11         sSet ← TrgAln(f, A), for some f ∈ SrcAln(sLabel, A);
12         foreach Node n ∈ sSet do
13             Append(tLabel, n.label);
14         end
15     end
16 end
17 return tLabel;
```

Figure 5.12: Function `getProjectedNodeLabel(sLabel)`: Compute the projected node label

```
Input: Tree D, Node n
Result: Node n is added in to the Tree D
1  if ∃ m ∈ D | m.label = n.label then
2      Append(m.relation, n.relation);
3      Append(m.pos, n.pos);
4      foreach auxLabel ∈ m.aux.labels do
5          Append(m.aux[auxLabel], n.aux[auxLabel]);
6      end
7  else
8      D ← D ∪ n;
9  end
```

Figure 5.13: Procedure `AddNode(D, n)`: Adds a dependency node to a dependency tree

These nodes are represented using a $\mathcal{E}$ node, as described in Section 5.3.2. The label for an $\mathcal{E}$ node is `E_sLabel`, where `sLabel` is the label of the source node it represents. Case 1 is for one-to-one and many-to-one alignments, where the label of the projected node is simply the label of the token on the target side. The default case handles one-to-many and many-to-many alignments. These alignments result in an $\mathcal{M}$ node on the target side, which is labeled as `M_`t_1`_`t_2`_...._`t_x, where t_i represents the target nodes that are aligned with the current source node.

Procedure `AddNode()`, Figure 5.13, adds dependency nodes to the target tree. The procedure also handles the merging of target nodes if multiple source nodes project onto the same target node. When two nodes need to be merged, we simply append the fields for the new node to the corresponding fields of the existing node in the tree. The only issue is the `governor` field, which should not be modified as it refers to another node. However, since node merges are a result of multiple nodes in a source cluster and we use the cluster governor for all nodes it contains, this issue never arises.

Function `GetProjectedGovernorInfo()`, presented in Figure 5.14, computes the label of the projected governor for a node, as well as some auxiliary information. We first find the cluster that contains the source node n. If the governor of the cluster is the same as the governor of the node, then we simply use the relation of the node. Lines 4-14 deal with the case where the node and the cluster have different labels. If the governor of the node is within the cluster, then the dependency relation is stored as a piece of auxiliary information with the label *internalLink* since this relation would be a self link when projected to the target side. If the

```
Input: Source node clusters C, source node sNode
Output: Projected governor information for node sNode
1  cluster ← c, where sNode.label ∈ c.members and c ∈ C;
2  if cluster.governor = sNode.governor then
3  |  relation ← sNode.relation;
4  else
5  |  relation ← nil ;
6  |  auxValue ← sNode.relation;
7  |  if sNode.governor ∈ cluster.members then
8  |  |  auxLabel ← "internalLink";
9  |  else
10 |  |  auxLabel ← "altRelation";
11 |  |  altGovLabel ← GetProjectedNodeLabel(sNode.governor);
12 |  |  Append(auxValue, altGovLabel);
13 |  end
14 end
15 govLabel ← GetProjectedNodeLabel(cluster.governor);
16 return (govLabel, relation, auxLabel, auxValue);
```

Figure 5.14: Function `getProjectedGovernorInfo(C, sNode)`: Computes projected governor information for node sNode

```
Input: Label l, PoS p, Relation r, Label g
Output: A newly allocated and initialized dependency node n
1  n ← new Node;
2  n.label ← l;
3  n.pos ← p;
4  n.relation ← r;
5  n.governor ← g;
6  return n;
```

Figure 5.15: Function `AllocateNewNode(l, p, r, g)`: Creates and initializes a new dependency node.

node has a governor outside the cluster, then we record this information with the label *altRelation* since this is an alternate governor for the cluster. Finally, we set the projected governor label for the node to the label of the target node that corresponds to the governor of the cluster. The `GetProjectedGovernorInfo()` function is necessary only because we want to save information about governors that are not selected as cluster governors. More specifically information is saved regarding these governors and their relations. Otherwise, Lines 1 and 15 would be enough to compute the projected governor label.

5.4 Evaluation

In this section we provide an evaluation of the DPA using the quality of projected trees as the measure of success. We then describe how projection performance can be improved using linguistic post-processing, and also demonstrate that the quality of the projected trees is good enough to train a statistical parser that achieves state-of-the-art parsing performance.[3]

5.4.1 Evaluating the Direct Projection Algorithm

The first empirical evaluation of the DPA was actually performed as an indirect evaluation of the underlying DCA assumption [60]. As the DPA is a direct application of the DCA, the performance of the former can be viewed as a measure of the

[3]This section reports on collaborative work. The evaluations reported in this section use the DPA and associated tools as their basis. All the evaluation runs reported in this chapter and the development of linguistic postprocessing techniques were conducted primarily by my collaborators [58, 60].

validity of the latter.

As our current goal is to evaluate the projection, without the compounding effect of errors in English parse trees and word level alignment, we opt to experiment with input data that is as clean as possible. The parallel corpus for the experiment was generated by acquiring manual English translations for 124 Chinese sentences from the Chinese Treebank [149]. The sentences had varying degrees of complexity in terms of their syntax. Their average length was 23.7 words. The English part of the corpus was first parsed using a statistical parser [19]; the resulting constituency trees were automatically converted into dependencies using an algorithm similar to the one presented in [148], and then manually corrected. The gold-standard parse trees for the Chinese side were manually generated by two annotators using the Chinese Treebank constituency trees as reference. The agreement between the two annotators was 92.4%, ignoring the actual link labels. Word level alignment links between English and Chinese sentences were also manually generated.

Precision	Recall	F-Measure
0.301	0.391	0.340

Table 5.1: Evaluation results for English to Chinese dependency projection

For evaluation, we used the DPA to infer the dependency trees on the Chinese side using the English dependency trees and the word level alignment between English and Chinese. The resulting Chinese dependency trees were then evaluated against the manually generated ground-truth for Chinese. As Table 5.1 illustrates, the evaluation results for the dependency trees generated by the DPA are pretty

low. However, as following sections demonstrate, the low score is not an indication of the failure of the algorithm, but rather a result of its language-blind nature. A small amount of linguistic post-processing uncovers the syntactic information that is successfully carried over by the algorithm, but remains hidden under the systematic errors caused by the structural differences of languages.

5.4.2 Beyond Direct Projection

The purpose of DPA is to project dependency trees across parallel text, generating a noisy treebank for low-density languages. However, a treebank is not the final product that we seek. The next step after projection is to improve the quality of the projected trees, and finally to train a statistical parser so that new sentences can be parsed directly.

Linguistic Post-Processing

A detailed analysis of the types of errors made by the DPA shows that a significant portion of the errors are caused by one-to-many alignments and unaligned words. This is not a surprising result since these are cases where it is not possible to infer syntactic information on the target side without using some form of language-specific, linguistic processing. Based on our observations, we created a small set of rules that manipulates the projected dependency trees to fix systematical errors.

The very first piece of linguistic information we employ is the consistent "head-edness" of languages. Chinese is a mostly "head first" language except for its noun

phrases. As discussed in Section 5.3.2, when we project a dependency across a many-to-one alignment, we do not have any information regarding the relation between the words on the target side. Correct projection would require determining the head of those words. Headedness provides an excellent heuristic for addressing this problem, as we can simply pick the first word of the set as the head for a head-first language, and the last one for a head-final language. We apply this heuristic to the output of the DPA by promoting the leftmost children of all the $\mathcal{M}$ nodes to replace them, and making the remaining nodes modifiers of the newly promoted head. This is a heuristic that can be applied to any language by simply picking the correct end of the phrase based on the headedness of the language.

Next, we developed a set of more specific rules that apply to Chinese. However, we avoided going down the path of manually generating a parser in the form of correction rules by imposing a restriction that the rules will only refer to closed-class items and POS in the rules. For example, one rule deals with the headedness of noun phrases, which does not follow the tendency of the language:

> If the source word of a many-to-one alignment is a noun, then instead of promoting the leftmost child of the $\mathcal{M}$ node to replace it, promote the rightmost child.

Another example rule deals with the Chinese aspectual markers for verbs, which do not have a counterpart on the English side. These markers either remain unaligned, or become part of a one-to-many alignment. The following rule attaches these markers to the proper verb:

If a Chinese verb v is followed by an aspectual marker a, make a a modifier of v.

Note that for both rules, the POS information for the Chinese side is projected from the English side as part of the dependency projection process, so no POS tagging is necessary on the Chinese side. A very compact rule set that has less than 20 members were generated in just a few days thanks to the following factors:

- Limited scope of the rules
- Working with higher level linguistic abstractions
- Focusing on the systematical violations of the DCA

Our focus here is on the DPA and its applications, therefore the interested reader is referred to [59] for details.

Evaluation of Linguistic Post-processing

Since the error analysis was performed on the evaluation set described in Section 5.4, that particular set was not suitable for evaluating the impact of linguistic post-processing. Therefore, we created a new data set for evaluation. We selected 88 Chinese sentences from the Chinese Treebank. These sentences were already manually translated into English as part of the National Institute of Standards and Technology (NIST) MT evaluation preview. The English dependency trees and the word-level alignments were obtained in the same manner as Section 5.4. For this set, the Chinese gold-standard was generated by automatically converting the manually

generated constituency trees from the Chinese Treebank into dependencies, instead of generating them manually as in the previous set.

Condition	Precision	Recall	F-Measure
DPA only	0.345	0.425	0.381
Head-Initial	0.594	0.594	0.594
Rules	0.680	0.666	0.673

Table 5.2: Evaluation of the projected Chinese dependency trees with post-processing

Table 5.2 shows the results for the new test set with and without post-processing. The first row shows the evaluation results on the output of the DPA without any further processing, and it actually confirms the results obtained using the previous data set, which are presented in Table 5.4. The second row presents the results obtained when we apply the extremely simple headedness heuristic. The F-measure jumps from 0.38 to 0.59 which is almost a 60% relative improvement. Note that this is not truly linguistic post-processing, but rather finalization of a decision that is deferred by the DPA as it is language-specific. The third row presents the results of applying the post-processing rules that are specifically designed for Chinese. They provide an additional boost, raising the F-measure to 0.67.

Impact of Automated Dependencies and Alignments

Producing a sizable treebank requires using automatic word alignments, and potentially a parser if a manual source treebank is not available. Therefore, we evaluated the quality of projected trees as impacted by the alignment and parse errors introduced by automation.

We first performed an experiment on Spanish using 100 sentences with gold-standard parse trees. Details of the test set are given in Section 5.4.3. We experimented using the following:

- Manually generated English dependencies and word alignments, which is the ideal case
- Automatically generated English dependencies and word alignments

The AER of automated word alignments was 24.4% for the test data. Because Spanish is a head-first language, we generated baseline dependency trees by making each word a modifier of the word to its left. Table 5.3 shows the unlabeled F-measure values for all three cases, with and without post-processing. The errors in automatically generated English parse trees and word alignments result in more errors in the projected Spanish trees compared to the manual case, as expected. However, the drop is relatively small. For both automatically and manually generated input, projected trees are better than the baseline. When we perform post-processing using language-specific rules and the headedness of the language, the quality of the trees increases dramatically. Note that the increase in the baseline is much smaller compared to projected trees, which is a good indication that the rules are indeed uncovering the information projected by the DPA rather than manually parsing the Spanish side.

We next generated a third test set for Chinese by automatically converting the constituency trees in Penn Chinese Treebank Version 2 into dependency trees. A portion of the resulting trees are reserved as the development set, and sentences that

	No Post-Processing	Post-Processing
Baseline	0.310	0.391
Projection from Automatic input	0.339	0.657
Projection from Manual input	0.368	0.703

Table 5.3: Evaluation of projected dependency trees for Spanish using automatically and manually generated input data (word alignments and English dependencies).

	No Post-Processing	Post-Processing
Baseline	0.359	0.431
Projection from Automatic input	0.263	0.524
Projection from Manual input	0.381	0.673

Table 5.4: Evaluation of projected dependency trees for Chinese using automatically and manually generated input data (word alignments and English dependencies).

contained more than 40 words were filtered. The remaining 2,800 sentences were used as the test set. Table 5.4 provides the evaluation results. Since English and Chinese are less similar than English and Spanish, the quality of projected trees is lower. For the automatically generated input case, the effect of the dissimilarity was compounded by the fact that we get an AER of 41%, which is much higher compared to Spanish. Post-processing still allows us to produce better Chinese dependency trees than the baseline. The last row of Table 5.4 is borrowed from Table 5.2 for easy comparison. We could not perform manual case evaluation on this test set as we did not have manually generated dependency trees and word alignments.

5.4.3 Parser Training

While we have demonstrated that dependency projection can produce reasonable quality parse trees, it is only useful if we have an English translation of the sentence

to be parsed. The final task that would allow us to parse any sentence is to generate a sizable treebank using projection, and train a statistical parser on that treebank. The steps of this task are given below for the English-Spanish projection. The general steps would be the same for any language pair. However, the software used at each step might be different. The following are the steps:

1. Obtain an English-Spanish parallel corpus.

2. Parse the English side.

   For our experiments, the English sentences were parsed using the statistical parser by Collins [19], trained on the Wall Street Journal portion of the Penn Treebank. The resulting constituency trees were automatically converted to dependency trees.

3. Align the English and Spanish sentences at the word level.

   Word level alignments were obtained using GIZA++ [101], a freely available and improved implementation of the IBM translation models [13].

4. Project the parse trees from the English side to the Spanish side, using the DPA.

5. Filter the projected Spanish parse trees that are deemed low-quality.

6. Train a statistical parser for Spanish using the resulting treebank.

   We used a version of the Collins parser that is adapted for dependency trees [20].

7. Parse new, unseen sentences using the trained Spanish parser.

We first evaluate the feasibility of this approach using Spanish, as it is relatively similar to English. Next we repeat the process on Chinese, which is less similar, hence more challenging.

Spanish

The very first step for training a parser using projected trees is to obtain a parallel corpus. We generated a 100,000 line English-Spanish parallel corpus by combining modern-day Bible translations, a sample from the FBIS corpus, and a sample from the UN corpus. 100 of the sentences in the combined corpus were separated as a development set and another 100 as a test set. As we did not have a Spanish treebank available, the gold-standard parse trees were generated starting with a state-of-the-art commercial Spanish parser, which produces a representation similar to dependency trees. The resulting parse trees were then corrected by two linguists with Spanish syntax training. The agreement between the dependency trees generated by the two linguists was 0.847 evaluated using the average unlabeled F-measure metric.

At the end of the projection step (step 4), we had 98,000 projected Spanish parse trees to train on. In order to reduce the effect of low-quality projected parse trees on the trained Spanish parser (step 5) we discarded sentence pairs if any of the following is true:

- More than 20% of the English words do not have a Spanish counterpart.

- More than 30% of Spanish words do not have an English counterpart.
- An English word is aligned to more than four Spanish words.

These rules were empirically determined using the development set. After the filtering process, the number of Spanish parse trees dropped to 20,000. Table 5.5 compares the performance of the trained Spanish parser to a baseline and a commercial parser using the unlabeled F-measure metric. As can be seen from the table, filtering improves the quality of the trained parser considerably. Both versions of the trained parsers are much better than the baseline. The performance of the parser trained on unfiltered parse trees is pretty close to the commercial parser, while the version trained on filtered trees exceeds its performance. Considering the amount of time and effort spent, the result is even more striking since the commercial parser most likely required much more resources to develop.

Chinese

Training a Chinese parser on parse trees projected from English is much more challenging compared to training a Spanish parser since Chinese is not as similar to

Parser	Performance
Baseline (modify previous)	0.338
Statistical (unfiltered, 98K sentences)	0.673
Statistical (filtered, 20K sentences)	0.721
Commercial	0.692

Table 5.5: Comparative evaluation of a Spanish parser trained on a projected treebank

English. First, the automatic word alignment performs worse for Chinese, which results in incorrect projections. Second, even when the alignments are correct, since Chinese is structurally more different we expect the DCA to be violated more often compared to Spanish.

We used 240,000 sentence pairs from the FBIS corpus for the Chinese experiment. To improve the quality of the word alignment, we appended the English-Chinese translation lexicon from LDC to the corpus during GIZA++ training, which is the usual practice.

The gold-standard parse trees for Chinese were generated by automatically converting the constituency trees from the Penn Chinese Treebank into dependencies. We validated the correctness of the dependency trees by comparing the automatically generated dependency trees in the development set to dependency trees generated by a linguist in the same manner. The agreement between the automatic and manual trees was 97.5%, which shows that the automatic constituency-to-dependency conversion is pretty reliable, and can be substituted for manual dependency generation.

Since Chinese projection produces more errors, we employed a more aggressive filter that considers cross-dependency links, unattached nodes, missing POS tags, in addition to the criteria used for Spanish. Consequently, filtering reduced the number of projected trees from 240,000 to 50,000.

Table 5.6 compares the performance of the Chinese parser trained on the projected trees to two baselines and to a parser that is trained on various numbers of manually created trees from the Penn Chinese Treebank (PCT) v4. The performance

of the projected parser is about 10% absolute better than the baseline and about 10% below the performance of the parser trained on all available manual data.

Training Data	Training Size	Performance
FBIS, projected, filtered	50,000	0.539
Chinese Treebank v4	500	0.509
Chinese Treebank v4	1,000	0.520
Chinese Treebank v4	2,000	0.535
Chinese Treebank v4	4,000	0.556
Chinese Treebank v4	10,000	0.643
Baseline (modify next)		0.351
Baseline and post-processing		0.443

Table 5.6: Comparative evaluation of a Chinese parser trained on a projected treebank

Although the results for Chinese are not as good as Spanish, as expected, it is still worth noting that the performance of the parser trained on the projected treebank is slightly better than the performance of the parser trained on 2,000 manually generated parse trees and very close to the parser trained on 4,000 sentences. It took LDC two years to generate 4,000 parse trees for Chinese, and it took us less than one person-month to write the Chinese-specific post-processing rules. Considering these facts about time spent we conclude that using projected trees to rapidly train a parser for a new language is still a valuable option. The projected parser can provide the initial parsing capability until a reasonable amount of manually generated parse trees become available. Furthermore, the projected parser can speed up the manual annotation process by providing noisy parse trees as a starting point for manual correction [85].

5.5 Summary

Acquiring manual annotations is a crucial step for developing statistical NLP systems in new languages. In this chapter, we focused on parsing as a sample platform for demonstrating cross-lingual utilization of manual annotations. Creating a treebank from scratch is a very challenging task [1]. For example, it took LDC several years to generate 15,000 Chinese parse trees. The difficulty and cost of manual treebank generation makes cross-lingual utilization of an existing treebank very desirable.

In Section 5.3, we presented an algorithm for projecting dependency parse trees from one side of a parallel corpus to the other, across word alignment links. The algorithm leverages existing treebanks or output of an existing parser to infer a treebank in a new language. Input requirements for the algorithm are as follows:

- A parallel text (far easier to come by than a treebank), one side of which has parse trees or can be parsed with an existing parser.
- Word alignments, which can be automatically computed using unsupervised methods.

We first evaluated the performance of the DPA on ideal input, where both the input parse trees and world alignments were manually generated. We used unlabeled precision/recall of dependency nodes as the evaluation metric. We projected dependency trees from English onto Chinese and compared the resulting Chinese trees against a manually generated gold standard. The averaged unlabeled F-measure

was 34%. This number seems pretty low, very close to a simple baseline. However, projection carries over a significant amount of information which is hidden behind systematic, language-specific errors. This fact is revealed by linguistic post-processing experiments.

Section 5.4.2 presented how a small amount of linguistic post-processing can significantly improve the quality of the projected parse trees [58]. The very first piece of information that we used was the headedness of the target language. Whenever we had an $\mathcal{M}$ node representing a set of words, we replaced it with its leftmost or rightmost child depending on the head position preferences of the target language. This single piece of linguistic information about the target language boosted the F-measure significantly. We also added a small set of language specific rules to fix systematic errors, which further improved the results. We were able to achieve F-measures as high as 70% for Spanish, and 67% for Chinese.

Next, in Section 5.4.3, we evaluated the performance of statistical parsers trained on projected treebanks. For this set of experiments, both the source parse trees and the word alignment were generated automatically. The Spanish parser that trained on projected trees achieved an unlabeled F-measure of 67%. Using a few simple rules to filter out low-quality projections, we improved the F-measure to 72%, which was better than a commercial Spanish parser.

Performance of the Chinese parser trained on projected trees was 54%, which is much lower compared to Spanish. This result was expected, though, as Chinese and English are much less similar compared to Spanish and English. The dissimilarity causes more alignment errors, leading to more projection errors. These errors

are in addition to errors caused by violations of DCA. While the performance is lower compared to Spanish, it is still significantly better than the baseline, and very close to the performance of a parser trained on 4,000 manually generated parse trees. Considering the fact that creating that many parse trees takes one or two years, treebank projection is a very attractive method for creating treebanks in new languages.

Similar to OCR post-processing work reported in Chapter 3, dependency projection is not meant to replace other efforts to build treebanks and parsers for new languages. It simply provides a fast and cost-efficient method for acquiring initial parsing capabilities. As a matter of fact, projected parse trees or the output of a parser trained on such trees can be used as the starting point for manual correction efforts [85].

Chapter 6

Conclusions and Future Directions

6.1 Conclusions

In this dissertation we have presented methods for cross-lingual utilization of existing resources to acquire electronic corpora and annotations for new languages. Corpora and annotations are two basic resources required to train statistical NLP methods, and the ability to acquire these two resources rapidly can allow rapid porting of NLP applications to new languages.

We first addressed the corpus acquisition problem by demonstrating the feasibility of using an OCR system to acquire online corpora for new languages. Our approach is based on using existing OCR systems, and improving their output via post-processing. We presented a flexible, modular, probabilistic generative OCR model designed specifically for the following:

- Ease of integration with probabilistic models of the sort commonly found in recent NLP work
- Rapid retargeting of OCR and NLP technology to new languages

We evaluated correction performance on a number of languages, and demonstrated that post-processing achieves significant improvements in recognition accuracy. We also evaluated the impact of OCR errors on NLP resources acquired from OCRed

text, and on NLP applications that take OCRed text as input. OCR errors degrade resource quality and application performance significantly. Post-processing is quite effective in reducing the degradation level.

Next we focused on the annotation problem, using parsing as a sample platform for demonstrating the cross-lingual utilization of manual annotations. We presented an algorithm for projecting dependency parse trees across parallel corpora, utilizing word alignment links as a bridge. The idea is to exploit existing treebanks or parsers using parallel corpora, which is more abundant than treebanks. Evaluation of the projection algorithm showed that a considerable amount of information is transferred via projection and that a small amount of linguistic post-processing can achieve an F-measure as high as 70%. Furthermore, parser training experiments showed that the performance of a statistical parser trained on projected trees can approach state-of-the-art parsing performance.

Having shown that cross-lingual utilization is effective in acquiring both corpora and annotations, we conclude that the ability to exploit existing resources for new languages can be a very attractive method of developing computational capabilities for low-density languages. However, we should emphasize the fact that resources acquired via cross-lingual utilization are far from perfect; they are intended to provide initial computational capabilities quickly until higher quality resources are developed. Furthermore, these noisy resources can be used as the foundation for efforts to create manual resources, simplifying the process and reducing time and cost requirements.

6.2 Contributions

Contributions of the research presented in this dissertation are as follows:

- A novel, end-to-end, probabilistic, generative model to describe the OCR process. It explicitly models conversion of the words to characters, segmentation of the character sequence, and character-level recognition. Most other OCR models focus on isolated words, ignoring word merge/split errors, and none directly addresses the word to character conversion step.

- An FSM-based implementation of the proposed model, including parameter estimation methods and decoding.[1] Use of FSMs provides a strong and well studied theoretical basis, reduces implementation time, and facilitates easier integration with other models implemented using the same framework.

- The application of the proposed method to various languages to demonstrate the feasibility of performing post-processing to improve the quality of OCR output. We have shown that post-processing is effective enough to enable using an OCR system to recognize a language that it does not natively support, assuming the script of the language is supported. We have also shown that native OCR performance can be improved using the same post-processing technique.

- The evaluation of the impact of OCR errors on NLP resources and applications. We have shown that OCR errors degrade the quality of NLP resources

[1]Some estimation methods make use of existing software, such as language modeling tools, and simply encode their output as FSM, while other estimation methods are implemented from scratch.

generated using OCRed text. NLP application performance is also degraded when its input contains OCR errors. Utilizing post-processing to reduce the error rate has a positive impact for both cases, bringing the performance much closer to that of clean text.

- An edit distance-based method for mapping the words in clean text and its noisy version in the presence of word merge/split errors. This method combines character-based and word-based methods to achieve better accuracy and finer granularity than either method alone. The ability to determine word-level mapping between OCR output and ground truth is important both for training OCR models and for performance evaluation on OCR text using gold standards created using ground truth.

- A new precision/recall-based metric for evaluating word-level alignment quality for parallel text in the presence of word merge/split errors introduced by noisy text sources. Existing metrics are not directly applicable when there is no one-to-one correspondence between the words in the gold-standard and the test data. However, the proposed metric can handle many-to-many mappings. The metric differs from the standard precision/recall-based approach only if word/merge errors are actually present.

- A novel algorithm for projecting dependency parse trees from one side of a parallel corpus onto the other across word alignments. The algorithm is designed to be language neutral, avoiding any decisions that would require language-specific knowledge, but retaining as much information as possible to allow

later stages of processing to make more informed decisions. We demonstrate that with a minimal amount of linguistic post-processing, the quality of the projected trees is good enough to train a statistical parser. The evaluation of the trees produced by the algorithm and linguistic post-processing work were carried out by our colleagues, as reported in [57, 58, 60].

- A rigorous evaluation of the cross-lingual utilization idea to acquire two basic resources, corpora and annotations, required to port NLP applications to new languages. We have shown that cross-lingual utilization is a feasible method to bootstrap computational capabilities for new languages.

6.3 Limitations

Some of the limitations of the research and evaluations presented in this dissertation are the following:

- The acquisition of electronic corpora via OCR assumes the availability of printed resources for the language of interest, which is not always the case. For example, we put forth a considerable amount of effort to acquire a small amount of printed Igbo material. For the languages where printed materials are not readily available, acquiring large amounts of corpora remains a challenge.

- The cross-lingual utilization of an OCR system assumes that the language of interest is written using a script that is recognized by the OCR system. If

there are no OCR systems that support a script that resembles the script that needs to be recognized, cross-lingual utilization is not an option. A feasible alternative is to train a script independent OCR system [e.g. 79, 96], and use post-processing to further improve recognition accuracy.

- The cross-lingual utilization of annotations are demonstrated using only parsing. Other annotations require new projection methods to be designed for them, which might require a substantial amount of research. However, there are successful examples of annotation projection for other tasks in literature, as discussed in Section 2.5, which is a good indication that the projection concept is applicable to a wide variety of tasks.

- Although we have evaluated corpus acquisition and annotation acquisition independently, due to time and resource constraints we have not evaluated the complete acquisition process end-to-end starting with printed corpora in a new language and ending with a functioning statistical NLP application. While corpus acquisition and annotation acquisition perform well independently, we do not know much about how they would interact with each other.

6.4 Future Directions

Some possible extensions of the work presented in this dissertation are:

- OCR for languages with unfamiliar scripts: Electronic corpora acquisition where a trainable script-independent OCR system is combined with a post-processing system needs to be explored. While the Latin and Arabic scripts

that we employed for our experiments are used by many languages, there are numerous other scripts used by many languages. Ability to recognize them is just as important.

- The extension of OCR correction to very large character sets: The post-processing models presented in this dissertation are not practical for languages with very large character sets such as Chinese and Japanese. Although there is no theoretical issue in terms of the model, time and memory requirements of a huge character set is prohibitive in practice. Methods for reducing the impact of a large character set, such as pruning and clustering, are required.

- The combination of lexicon-based and lexicon-free correction: Since lexicon-based correction performs better than lexicon-free correction, relying on the lexicon for correction but backing-off to the lexicon-free methods for words that are not in the lexicon would improve correction performance, while still allowing the proper handling of novel words. Deciding which of the correction methods to employ for a given word is a difficult problem that makes combination challenging.

- The automation of linguistic post-processing of projected parse trees: Our linguistic post-processing rules were generated manually by inspecting the types of errors observed in the projected trees. Automating this process, or at least turning it into a machine-assisted task, might be possible by using a small number of manual parse trees in the target language. Given the projected trees and their correct versions, machine learning methods such as TBL [11]

can be used to correct some of the projection errors.

- The projection process as the starting point for manual annotation: Given the fact that manual annotations perform better than the noisy annotations produced using projection, the idea of using projection as a starting point for manual annotation is worth exploring. Presumably, it would be easier for annotators to correct projected annotations than to start from scratch, resulting in faster annotation times and lower costs.

- The end-to-end application of cross-lingual utilization: Starting with printed resources in a low-density language and going through the whole resource acquisition and training process end-to-end, can be useful. It would not only provide valuable information regarding the potential challenges resulting from the interactions of the steps of the process, but it would also provide a reference point for the performance level that can be achieved via cross-utilization for a language with no computational resources.

BIBLIOGRAPHY

[1] Anne Abeillé, editor. *Treebanks: Building and Using Parsed Corpora*. Kluwer Academic Publishers, 2003. ISBN: 1402013345.

[2] Steven Abney. Partial parsing via finite-state cascades. In *Proceedings of the Robust Parsing Workshop at ESSLLI-96*, Prague, Czech Republic, August 1996.

[3] Cyril N. Alberga. String similarity and misspellings. *Communications of the ACM*, 10(5):302–313, May 1967.

[4] Hiyan Alshawi, Srinivas Bangalore, and Shona Douglas. Learning dependency transduction models as collections of finite state head transducers. *Computational Linguistics*, 26(1):45–60, March 2000.

[5] Juan-Carlos Amengual and Enrique Vidal. Efficient error-correcting viterbi parsing. *IEEE Transactions on Pattern Analysis and Machine Intelligence—PAMI*, 20(10):1109–1116, October 1998.

[6] Jason Baldridge and Miles Osborne. Active learning for HPSG parse selection. In *Proceedings of the 7th Conference on Natural Language Learning*, Edmonton, Alberta, Canada, May 2003.

[7] Steven M. Beitzel, Eric C. Jensen, and David A. Grossman. Retrieving OCR text: A survey of current approaches. In *Proceedings of the Workshop on Information Retrieval and OCR: From Convertiong Content to Grasping Meaning, ACM SIGIR 2002 Conference*, Tampere, Finland, August 2002.

[8] Luisa Bentivogli, Pamela Forner, and Emanuele Pianta. Evaluating cross-language annotation transfer in the MultiSemCor corpus. In *Proceedings of the 21st International Conference on Computational Linguistics (COLING-04)*, pages 364–370, Geneva, Switzerland, August 2004.

[9] Luisa Bentivogli and Emanuele Pianta. Opportunistic semantic tagging. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC-02)*, Las Palmas, Canary Islands, Spain, June 2002.

[10] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the 11th Annual Conference on Computational Learning Theory (COLT-98)*, pages 92–100, Madison, Wisconsin, USA, July 1998.

[11] Eric Brill. Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging. *Computational Linguistics*, 24(1):543–565, 1995.

[12] Eric Brill and Robert C. Moore. An improved model for noisy channel spelling correction. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 286–293, Hong Kong, China, October 2000.

[13] Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. A statistical approach to machine translation. *Computational Linguistics*, 16(2): 79–85, 1990.

[14] Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The mathematics of machine translation: Parameter estimation. *Computational Linguistics*, 19(2), 1993.

[15] David Chiang. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL-05)*, pages 263–270, Ann Arbor, Michigan, USA, June 2005.

[16] K. Church and W. Gale. Probability scoring for spelling correction. *Statistics and Computing*, 1:93–103, 1991.

[17] Philip Clarkson and Ronald Rosenfeld. Statistical language modeling using the CMU-Cambridge Toolkit. In *Proceedings of the ESCA Eurospeech*, Rhodes, Greece, 1997.

[18] David A. Cohn, Zoubin Ghahramani, and Michael I. Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4:129–145, 1996.

[19] Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL/EACL-97)*, pages 16–23, Madrid, Spain, July 1997.

[20] Michael Collins, Jan Hajic, Lance Ramshaw, and Christoph Tillman. A statistical parser for Czech. In *Proceeding of the 37th Annual Meeting of the Association for Computational Linguistics (ACL-99)*, pages 505–512, College Park, Maryland, USA, June 1999.

[21] Kevyn Collins-Thompson, Charles Schweizer, and Susan Dumais. Improved string matching under noisy channel conditions. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*, pages 357–364, Atlanta, Georgia, USA, October 2001.

[22] W. B. Croft, S. M. Harding, K. Taghva, and J. Borsack. An evaluation of information retrieval accuracy with simulated OCR output. In *Proceedings of the Symposium of Document Analysis and Information Retrieval, ISRI-UNLV*, 1994.

[23] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[24] Mona Diab. *Word Sense Disambiguation within a Multilingual Framework*. PhD thesis, University of Maryland, College Park, 2003.

[25] Mona Diab. An unsupervised approach for bootstrapping Arabic sense tagging. In *Proceedings of the Arabic Script Based Languages Workshop at COLING-04*, Geneva, Switzerland, August 2004.

[26] Mona Diab and Philip Resnik. An unsupervised method for word sense tagging using parallel corpora. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL-02)*, Philedelphia, Pennsylvania, USA, July 2002.

[27] David Doermann. The indexing and retrieval of document images: A survey. *Computer Vision and Image Understanding: CVIU*, 70(3):287–298, 1998.

[28] David Doermann, Huanfeng Ma, Burcu Karagöl-Ayan, and Douglas W. Oard. Translation lexicon acquisition from bilingual dictionaries. In *Proceedings of the Ninth SPIE Symposium on Document Recognition and Retrieval*, San Jose, California, USA, 2002.

[29] Bonnie Dorr. Machine translation divergences: A formal description and proposed solution. *Computational Linguistics*, 20(4):597–635, December 1994.

[30] Ted Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1):61–74, March 1993.

[31] Line Eikvil. OCR - optical character recognition. Technical Report 876, Statistical Analysis, Image Analysis and Pattern Recognition (SAMBA) Dpt. Norwegian Computing Center, Oslo, Norway, 1993.

[32] Jason Eisner. Learning non-isomorphic tree mappings for machine translation. In *Companion Volume to the Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL-03)*, Sapporo, Japan, July 2003.

[33] Dan Flickinger. On building a more efficient grammar by exploiting types. *Natural Langauge Engineering*, 6(1):15–28, 2000. Special Issue on Efficient Processing with HPSG.

[34] Heidi Fox. Phrasal cohesion and statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 304–311, Philadelphia, June 2002.

[35] Robert Frederking. Summary of the MIDAS session on handling multilingual speech, document images, and video OCR, August 1999. http://www.clis2.umd.edu/conferences/midas/papers/frederking.txt.

[36] Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby. Selective sampling using query by committee algorithm. *Machine Learning*, 28(2-3): 133–168, August 1997.

[37] Ulrich Germann. Greedy decoding for statistical machine translation in almost linear time. In *Proceedings of the Human Language Technology Conference HLT/NAACL-03*, Edmonton, Alberta, Canada, May 2003.

[38] Ulrich Germann, Michael Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. Fast decoding and optimal decoding for machine translation. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL-01)*, pages 228–235, Toulouse, France, July 2001. Association for Computational Linguistics, Morgan Kaufmann Publishers.

[39] Daniel Gildea. Loosely tree-based alignment for machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL-03)*, Sapporo, Japan, July 2003.

[40] Andrew R. Golding and Dan Roth. Applying winnow to context sensitive spelling correction. In *Procedings of the 13th International Conference on Machine Learning*, pages 182–190, San Francisco, California, USA, 1996.

[41] Sally Goldman and Yan Zhou. Enhancing supervised learning with unlabeled data. In *Proceedings of the 17th International Conference on Machine Learning (ICML-00)*, Stanford, California, USA, June 2000.

[42] Ardeshir Goshtasby and Roger W. Ehrich. Contextual word recognition using probabilistic relaxation labeling. *Pattern Recognition*, 21(5):455–462, 1988.

[43] David Graff. UN parallel text. LDC Catalog No.: LDC94T4A, Linguistic Data Consortium, University of Pennsylvania, 1994. ISBN: 1-58563-038-1.

[44] M. M. Green and M. O. Onwuamaegbu, editors. *Akụkọ Ife Nke Ndị Igbo*. Oxford University Press, Ibadan, Nigeria, 1970. ©1962.

[45] Isabelle Guyon and Fernando Pereira. Design of a linguistic postprocessor using variable memory length Markov models. In *Proceedings of the Third International Conference on Document Analysis and Recognition (ICDAR)*, volume 1, pages 454–457, Montreal, Quebec, Canada, August 1995.

[46] Patric A. V. Hall and Geoff R. Dowling. Approximate string matching. *ACM Computing Surveys*, 12(4):381–402, December 1980.

[47] Chung-Hye Han, Benoi Lavoi, Martha Palmer, Owen Rambow, Richard Kittredge, Tanya Korelsky, Nari Kim, and Myunghee Kim. Handling structural divergences and recovering dropped arguments in a Korean/English machine translation system. In *Proceedings of the 4th Conference of the Association for Machine Translation in the Americas (AMTA-2000)*, pages 40–53, Cuernavaca, Mexico, October 2000.

[48] Allen R. Hanson, Edward M. Riseman, and Edward G. Fisher. Context in word recognition. *Pattern Recognition*, 8:33–45, 1976.

[49] Stephen M. Harding, W. Bruce Croft, and C. Weir. Probabilistic retrieval of OCR degraded text using n-grams. In *Proceedings of the European Conference on Digital Libraries—ECDL '97*, pages 345–359, Pisa, Italy, September 1997.

[50] Ulf Hermjakob and Raymond J. Mooney. Learning parse and translation decisions from examples with rich context. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL/EACL-97)*, pages 482–489, Madrid, Spain, July 1997.

[51] Stefan Höfler. *Link2Tree: A Dependency-Constituency Converter*. PhD thesis, University of Zurich, April 2002.

[52] Melissa Holland and Chris Schlesiger. High-modality machine translation for a battlefield environment. In *Proceedings of the NATO/RTO Systems Concepts and Integration Symposium*, Monterey, California, USA, April 1998. 15/1-3. Hull, Canada: CCG, Inc., ISBN 92-837-1006-1.

[53] Jianying Hu, William Turin, and Michael K. Brown. Langauge modeling using stochastic automata with variable length contexts. *Computer Speech and Language*, 11(1):1–16, 1997.

[54] Shudong Huang, David Graff, and George Doddington. Multiple-translation Chinese corpus. LDC Catalog No.: LDC2002T01, Linguistic Data Consortium, University of Pennsylvania, February 2002. ISBN: 1-58563-217-1.

[55] Rebecca Hwa. Sample selection for statistical grammar induction. In *Proceedings of the 2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-00)*, pages 45–52, Hong Kong, China, October 2000.

[56] Rebecca Hwa. Sample selection for statistical parsing. *Computational Linguistics*, 30(3):253–276, September 2004.

[57] Rebecca Hwa, Philip Resnik, and Amy Weinberg. Breaking the resource bottleneck for multilingual parsing. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC-02)*, Las Palmas, Canary Islands, Spain, June 2002. Workshop on Linguistic Knowledge Acquisition and Representation: Bootstrapping Annotated Language Data.

[58] Rebecca Hwa, Philip Resnik, Amy Weinberg, Clara Cabezas, and Okan Kolak. Bootstrapping parsers via syntactic projection across parallel texts. In Rada Mihalcea and Michel Simard, editors, *Special Issue of the Journal of Natural Language Engineering on Parallel Texts*. 2005.

[59] Rebecca Hwa, Philip Resnik, Amy Weinberg, and Okan Kolak. Evaluating translational correspondence using annotation projection. Technical Report CS-TR-4455, Computer Science Department, University of Maryland, College Park, Maryland, USA, 2000.

[60] Rebecca Hwa, Philip Resnik, Amy Weinberg, and Okan Kolak. Evaluating translational correspondence using annotation projection. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL-02)*, Philedelphia, Pennsylvania, USA, July 2002.

[61] Mark A. Jones, Guy A. Story, and Bruce W. Ballard. Integrating multiple knowledge sources in a Bayesian OCR post-processor. In *Proceedings of the IDCAR-91*, pages 925–933, St. Malo, France, 1991.

[62] Simon Kahan, Theo Pavlidis, and Henry S. Baird. On the recognition of printed characters of any font and size. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(2):274–288, March 1987.

[63] Tapas Kanungo, Philip Resnik, Song Mao, Doe wan Kim, and Qigong Zheng. The Bible and multilingual optical character recognition. *Communications of the ACM*, 48(6):124–130, 2005.

[64] Kevin Knight. A statistical MT tutorial workbook, April 1993. http://www.isi.edu/natural-language/mt/wkbk.rtf.

[65] Kevin Knight and Jonathan Graehl. Machine transliteration. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL/EACL-97)*, pages 128–135, Madrid, Spain, July 1997.

[66] Phlipp Koehn. Pharaoh: a beam search decoder for phrase-based statistical machine translation models. In *Proceedings of the 6th Conference of the Association for Machine Translation in the Americas (AMTA-2004)*, Washington DC, USA, September 2004.

[67] Okan Kolak, William Byrne, and Philip Resnik. A generative probabilistic OCR model for NLP applications. In *Proceedings of the Human Language Technology Conference (HLT/NAACL-03)*, Edmonton, Alberta, Canada, May 2003.

[68] Okan Kolak and Philip Resnik. OCR error correction using a noisy channel model. In *Proceedings of the Human Language Technology Conference (HLT-2002)*, San Diego, California, USA, March 2002.

[69] Okan Kolak and Philip Resnik. OCR post-processing for low density languages. In *Proceedings of the Joint Conference on Human Language Technology / Empirical Methods in Natural Language Processing (HLT-EMNLP-05)*, Vancouver, BC, Canada, October 2005.

[70] Gary E. Kopec and Philip A. Chou. Document image decoding using markov source models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6):602–617, June 1994.

[71] Jonas Kuhn. Experiments in parallel-text based grammar induction. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, Barcelona, Spain, July 2004.

[72] Karen Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439, December 1992.

[73] Shankar Kumar and William Byrne. A weighted finite state transducer implementation of the alignment template model for statistical machine translation. In *Proceedings of the Human Language Technology Conference (HLT/NAACL-03)*, Edmonton, Alberta, Canada, May 2003.

[74] Shari Landes, Claudia Leacock, and Randee I. Tengi. Building semantic concordances. In Christiane Fellbaum, editor, *WordNet: An Electronic Lexical Database*, chapter 8. MIT Press, 1998.

[75] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady.*, 10(8):707–710, 1966.

[76] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[77] Daniel P. Lopresti. Robust retrieval of noisy text. In *Proceedings of the Third Forum on Research and Technology Advances in Digital Libraries*, pages 76–85, Washington DC, USA, May 1996.

[78] Roy Lowrance and Robert A. Wagner. An extension of the string-to-string correction problem. *Journal of the ACM*, 22(2):177–183, April 1975.

[79] Zhidong Lu, Issam Bazzi, Andras Kornai, John Makhoul, Premkumar Natarajan, and Richard Schwartz. A robust, language-independent OCR system. In Robert J. Mericsko, editor, *Proceedings of the 27th AIPR Workshop: Advances in Computer-Assisted Recognition*, volume 3584 of *Proceedings of SPIE*, January 1999.

[80] Huanfeng Ma, Burcu Karagöl-Ayan, David Doermann, Jianqiang Wang, and Doug Oard. Parsing and tagging of bilingual dictionaries. *Traitement Automatique Des Languages*, 44,2:125–149, 2003.

[81] Xiaoyi Ma. Multiple-translation Arabic (MTA) part 2. LDC Catalog No.: LDC2005T05, Linguistic Data Consortium, University of Pennsylvania, February 2005. ISBN: 1-58563-328-3.

[82] David Magerman. *Natural Language Parsing as Statistical Pattern Recognition*. PhD thesis, Stanford University, California, USA, February 1994.

[83] Lidia Mangu and Eric Brill. Automatic rule acquisition for spelling correction. In *Proceedings of the 14th International Conference on Machine Learning*, pages 187–194. Morgan Kaufmann, 1997.

[84] Daniel Marcu and Ulrich Germann. The ISI ReWrite Decoder release 0.7.0b. http://www.isi.edu/~germann/software/ReWrite-Decoder/.

[85] Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[86] Andrew McCallum and Kamal Nigam. Employing EM and pool-based active learning for text classification. In *Proceedings of the 15th International Conference on Machine Learning (ICML-98)*, pages 359–367, Madison, Wisconsin, USA, July 1998.

[87] Marjorie J. McShane, Stephen Helmreich, Sergei Nirenburg, and Victor Raskin. Slavic as testing grounds for a linguistic elicitation system. In *Proceedings of the Annual Workshop on Formal Approaches to Slavic Linguistics*, Philedelphia, Pennsylvania, USA, 1999.

[88] Surapant Meknavin, Boonserm Kijsirikul, Ananlada Chotimongkol, and Cholwich Nutee. Combining trigram and winnow in Thai OCR error correction. *NECTEC Technical Journal*, 1(1):19–26, March 1999.

[89] I. Dan Melamed. Multitext grammars and synchronous parsing. In *Proceedings of the Human Language Technology Conference HLT/NAACL-03*, Edmonton, Alberta, Canada, May 2003.

[90] I. Dan Melamed. Statistical machine translation by parsing. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, Barcelona, Spain, July 2004.

[91] I. Dan Melamed, Giorgio Satta, and Ben Wellington. Generalized multitext grammars. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, Barcelona, Spain, July 2004.

[92] Rada Mihalcea and Vivi Nastase. Letter level learning for language independent diacritics restoration. In *Proceedings of the Sixth Workshop on Computational Language Learning (CoNLL-02)*, pages 105–111, Taipei, Taiwan, August 2002.

[93] David Miller, Sean Boisen, Richard Schwartz, Rebecca Stone, and Ralp Weischedel. Named entity extraction from noisy input: Speech and OCR. In *Proceedings of the Sixth Conference on Applied Natural Language Processing*, pages 316–324, Seattle, Washington, USA, April 2000.

[94] Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. A rational design for a weighted finite-state transducer library. *Lecture Notes in Computer Science*, 1436, 1998.

[95] Shunji Mori, Ching Y. Suen, and Kazuhiko Yamamoto. Historical review of OCR research and development. *Proceedings of the IEEE*, 80(7):1029–1058, July 1992.

[96] Premkumar Natarajan, Zhidong Lu, Richard Schwartz, Issam Bazzi, and John Makhoul. Multilingual machine printed OCR. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(1):43–63, 2001.

[97] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, March 2001.

[98] Kamal Nigam and Rayid Ghani. Analyzing the effectiveness and applicability of co-training. In *Processing of the 9th International Conference on Information and Knowledge Management (CIKM-00)*, McLean, Virginia, USA, November 2000.

[99] Sergei Nirenburg. Project Boas: A linguist in the box as a multi-purpose language. In *Proceedings of the First International Conference on Language Resources and Evaluation (LREC-98)*, Granada, Spain, 1998.

[100] Douglas W. Oard. The surprise language exercises. *ACM Transactions on Asian Language Information Processing (TALIP)*, 2(2):79–84, June 2003. ISSN: 1530-0226.

[101] Franz J. Och and Hermann Ney. Improved statistical alignment models. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL-00)*, pages 440–447, Hongkong, China, October 2000.

[102] Franz Josef Och. Minimum error rate training for statistical machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL-03)*, Sapporo, Japan, July 2003.

[103] B.J. Oommen and R.L. Kashyap. A formal theory for optimal and information theoretic syntactic pattern recognition. *Pattern Recognition*, 31:1159–1177, 1998.

[104] Miles Osborne and Jason Baldridge. Ensemble-based active learning for parse selection. In *Proceedings of the Human Language Texhnology Conference (HLT/NAACL-04)*, Boston, Massachusetts, USA, May 2004.

[105] U. Pal, P. K. Kundu, and B. B. Chaudhuri. OCR error correction of an inflectional indian language using morphological parsing. *Journal of Information Science and Engineering*, 16(6):903–922, November 2000.

[106] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL-02)*, pages 311–318, Philedelphia, Pennsylvania, USA, July 2002.

[107] Juan Carlos Perez-Cortes, Juan-Carlos Amengual, Joaquim Arlandis, and Rafael Llobet. Stochastic error-correcting parsing for OCR post-processing. In *Proceedings of the ICPR*, pages 4405–4408, Barcelona, Spain, September 2000.

[108] Emanuele Pianta and Luisa Bentivogli. Knowledge intensive word alignment with KNOWA. In *Proceedings of the 21st International Conference on Computational Linguistics (COLING-04)*, Geneva, Switzerland, August 2004.

[109] David Pierce and Claire Cardie. Limitations of co-training for natural language learning from large datasets. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP-01)*, Pittsburgh, Pennsylvania, USA, June 2001.

[110] Katharina Probst, Jaime Carbonell, Alon Lavie, Lori Levin, and Erik Peterson. Design and implementation of controlled elicitation for machine translation of low-density languages. In *Proceedings of the MT2010 Roadmap Workshop–Towards a Road Map for MT*, Santiago de Compostela, Spain, September 2001.

[111] Katharina Probst and Lori Levin. Challenges in automated elicitation of a controlled bilingual corpus. In *Proceedings of the 9th International Conference on Theoretical and Methodological Issues in Machine Translation*, pages 157–167, Keihanna, Japan, March 2002.

[112] Philip Resnik. Exploiting hidden meanings: Using bilingual text for monolingual annotation. In Alexander Gelbukh, editor, *Lecture Notes in Computer Science: Computational Linguistics and Intelligent Text Processing*, volume 2945, pages 283–299. Springer, 2004. ISBN: 3-540-21006-7.

[113] Philip Resnik and I. Dan Melamed. Semi-automatic acquisition of domain-specific translation lexicons. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, Washington DC, USA, 1997.

[114] Philip Resnik, Douglas Oard, and Gina Levow. Improved cross-language retrieval using backoff translation. In *Proceedings of the Human Language Technology Conference (HLT-2001)*, San Diego, California, USA, March 2001.

[115] Philip Resnik, Mari Broman Olsen, and Mona Diab. The Bible as a parallel corpus: Annotating the 'Book of 2000 Tongues'. *Computers and the Humanities*, 33(1-2):129–153, 1999.

[116] Ellen Riloff, Charles Schafer, and David Yarowsky. Inducing information extraction systems for new languages via cross-language projection. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING-02)*, Taipei, Taiwan, August 2002.

[117] Edward M. Riseman and Allen R. Hanson. A contextual postprocessing system for error correction using binary n-grams. *IEEE Transactions on Computers*, 23(5):480–493, May 1974.

[118] Eric Sven Ristad and Peter N. Yianilos. Learning string edit distance. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, May 1998.

[119] Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

[120] Emmanuel Roche and Yves Schabes, editors. *Finite-State Language Processing*. MIT Press, Cambridge, Massachusetts, USA, June 1997. ISBN: 0-262-18182-7.

[121] Monica Rogati, Scott McCarley, and Yiming Yang. Unsupervised learning of Arabic stemming using a parallel corpus. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL-03)*, pages 391–398, Sapporo, Japan, July 2003.

[122] Walter S. Rosenbaum and John J. Hilliard. Multifont OCR postprocessing system. *IBM Journal of Research and Development*, 19(4):398–421, July 1975.

[123] Azriel Rosenfeld, Robert A. Hummel, and Steven W. Zucker. Scene labelling by relaxation operations. *IEEE Transactions on Systems, Man and Cybernetics*, 6:420–433, 1976.

[124] Anoop Sarkar. Applying co-training methods to statistical parsing. In *Proceedings of the Second Meeting of the North American Chapter of Association for Computational Linguistics (NAACL-01)*, pages 175–182, Pittsburgh, Pennsylvania, USA, June 2001.

[125] Claude Elwood Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3,4):379–423,623–656, July, October 1948.

[126] Svetlana Sheremetyeva and Sergei Nirenburg. Towards a universal tool for NLP resource acquisition. In *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC-00)*, Athens, Greece, May 2000.

[127] R. M. K. Sinha and Biendra Prasada. Visual text recognition through contextual processing. *Pattern Recognition*, 21(5):463–479, 1988.

[128] David A. Smith and Noah A. Smith. Bilingual parsing with factored estimation: Using English to parse Korean. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Barcelona, Spain, July 2004.

[129] Sargur N. Srihari, Jonathan J. Hull, and Ramesh Choudhari. Integrating diverse knowledge sources in text recognition. *ACM Transactions on Office Information Systems*, 1(1):68–87, January 1983.

[130] Mark Steedman, Miles Osborne, Anoop Sarkar, Stephen Clark, Rebecca Hwa, Julia Hockenmaier, Paul Ruhlen, Steven Baker, and Jeremiah Crim. Bootstrapping statistical parsers from small datasets. In *Proceedings of the Tenth Conference of the European Chapter of the Association for Computational Linguistics (EACL-03)*, pages 331–338, Budapest, Hungary, April 2003.

[131] Andreas Stolcke. SRILM—An extensible language modeling toolkit. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP-02)*, Denver, Colorado, USA, September 2002.

[132] Christian M. Strohmaier, Christoph Ringsletter, Klaus U. Schulz, and Stoyan Mihov. Lexical postcorrection of OCR-results: The web as a dynamic secondary dictionary. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR)*, volume 2, pages 1133–1137, Edinburgh, Scotland, August 2003.

[133] Kazem Taghva, Julie Borsack, and Allen Condit. Evaluation of model-based retrieval effectiveness with OCR text. *ACM Transactions on Information Systems (TOIS)*, 14(1):64–93, January 1996.

[134] Kazem Taghva and Eric Stofsky. OCRSpell: an interactive spelling correction system for OCR errors in text. *International Journal on Document Analysis and Recognition—IJDAR*, 3(3):125–137, 2001.

[135] Min Tang, Xiaoqiang Luo, and Salim Roukos. Active learning for statistical natural language parsing. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL-02)*, pages 120–127, Philedelphia, Pennsylvania, USA, July 2002.

[136] Cynthia A. Thompson, Mary E. Califf, and Raymond J. Mooney. Active learning for natural language parsing and information extraction. In *Proceedings of the 16th International Conference on Machine Learning (ICML-99)*, pages 406–414, Bled, Slovenia, June 1999.

[137] Xiang Tong and David A. Evans. A statistical approach to automatic OCR error correction in context. In *Proceedings of the Fourth Workshop on Very Large Corpora*, pages 88–100, Copenhagen, Denmark, August 1996.

[138] Øivind Due Trier, Anil K. Jain, and Torfinn Taxt. Feature extraction methods for character recognition - a survey. *Pattern Recognition*, 29(4):641–662, April 1996.

[139] Tony Uchenna Ubesie. *Juo Obinna*. University Press PLC, Ibadan, Nigeria, 1993. ISBN: 19575395X.

[140] Martin Volk and Yvonne Samuelsson. Bootstrapping parallel treebanks. In *Proceedings of the Workshop on Linguistically Interpreted Corpora (LINC) at COLING-04*, Geneva, Switzerland, August 2004.

[141] Clare R. Voss and Carol Van Ess-Dykema. When is an embedded MT system 'good enough' for filtering? In *Proceedings of the Workshop on Embedded MT Systems, ANLP/NAACL-00*, Seattle, Washington, USA, May 2000.

[142] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, January 1974.

[143] Ian H. Witten and Timothy C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1093, July 1991.

[144] Dekai Wu. An algorithm for simultaneously bracketing parallel texts by aligning words. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL-95)*, pages 189–196, Cambridge, Massachusetts, USA, June 1995.

[145] Dekai Wu. Stochastic inversion transduction grammars, with application to segmentation, bracketing, and alignment of parallel corpora. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 1328–1335, Montreal, Canada, August 1995.

[146] Dekai Wu. Trainable coarse bilingual grammars for parallel text bracketing. In *3rd Annual Workshop on Very Large Corpora (WVLC-3)*, pages 69–82, Cambridge, Massachusetts, USA, June 1995.

[147] Dekai Wu. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–404, September 1997.

[148] Fei Xia and Martha Palmer. Converting dependency structures to phrase structures. In *Proceedings of the Human Language Technology Conference (HLT-2001)*, San Diego, California, USA, March 2001.

[149] Fei Xia, Martha Palmer, Nianwen Xue, Mary Ellen Ocurowski, John Kovarik, Fu-Dong Chiou, Shizhe Huang, Tony Kroch, and Mitch Marcus. Developing guidelines and ensuring consistency for Chinese text annotation. In *Proceedings of the Second Language Resources and Evaluation Conference (LREC-00)*, Athens, Greece, May 2000.

[150] Kenji Yamada and Kevin Knight. A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL-01)*, Toulouse, France, July 2001.

[151] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL-95)*, pages 189–196, Cambridge, Massachusetts, USA, June 1995.

[152] David Yarowsky. A comparison of corpus-based techniques for restoring accents in spanish and french text. In *Natural Language Processing Using Very Large Corpora*, pages 99–120. Kluwer Academic Publishers, 1999.

[153] David Yarowsky. Scalable elicitation of training data for machine translation. Team TIDES, 2003. http://language.cnri.reston.va.us/TeamTIDES.html.

[154] David Yarowsky and Grace Ngai. Inducing multilingual POS taggers and NP bracketers via robust projection across aligned corpora. In *Proceedings of the Second Meeting of the North American Chapter of Association for Computational Linguistics (NAACL-01)*, pages 200–207, Pittsburgh, Philedelphia, USA, June 2001.

[155] David Yarowsky, Grace Ngai, and Richard Wicentowski. Inducing multilingual text analysis tools via robust projection across aligned corpora. In *Proceedings of the Human Language Technology Conference (HLT-01)*, San Diego, California, USA, March 2001.

[156] Pierre Zweigenbaum and Natalia Grabar. Accenting unknown words in a specialized language. In *Proceedings of the Workshop on Natural Language Processing in the Biomedical Domain*, pages 21–28, Philedelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.