UNIVERSITY OF CALIFORNIA, SAN DIEGO

# Preparing PDF Scientific Articles for Biomedical Text Mining

A thesis submitted in partial satisfaction of the
requirements for the degree of Master of Science

in

Computer Science

by

Shitij Bhargava

Committee in charge:

Professor Chun-Nan Hsu, Chair
Professor Vineet Bafna
Professor Julian McAuley

2015

The Thesis of Shitij Bhargava is approved and is acceptable in quality and form for publication on microfilm and electronically:

---

---

---

Chair

University of California, San Diego

2015

# DEDICATION

To my parents and sister, without whom this thesis or a masters degree would not have been possible.

# EPIGRAPH

Let us try to teach generosity and altruism, because we are born selfish. Let us understand what our own selfish genes are up to, because we may then at least have the chance to upset their designs, something that no other species has ever aspired to do.

*Richard Dawkins, The Selfish Gene*

As you think, so you are.

*Buddha*

With the arrival of humans, it has been said, the universe has suddenly become conscious of itself. This, truly, is the greatest mystery of all.

*V.S. Ramachandran, The Tell-Tale Brain*

The hardest thing to explain is the glaringly evident which everybody has decided not to see.

*Ayn Rand, The Fountainhead*

When a man learns to love, he must bear the risk of hatred.

*Uchiha Madara, Naruto*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to acknowledge Professor Chun-Nan Hsu for his support throughout my masters and my thesis work. His guidance at every juncture proved to be invaluable.

I would also like to acknowledge Gordon Lin, Kashyap Tumkur, Suvir Jain and Tsung-Tim Kuo, who have been great team mates to work with and have provided important ideas and support from time to time.

I also want to express my sincere gratitude to the PDFX team, including Steve Pettifer and Alexandru Constantin, who generously shared PDFX binaries for transcribing a large number of PDFs and have been very helpful in all communications. I also thank Lucia Hindorff, program official at NHGRI for sharing PDFs from the catalog of GWAS with us.

Furthermore, I want to thank Prof. Vineet Bafna and Prof. Julian McAuley for their valuable comments and feedback for this thesis.

This work was supported by the National Human Genome Research Institute (NHGRI) of the National Institutes of Health (NIH) under award number U01HG006894.

Chapter 2, 3, 4 and 7 in part is currently being prepared for submission for publication of the material. The thesis author was the primary investigator and author of this material.

VITA

| | |
|---|---|
| 2012 | Bachelor of Technology, G.G.S.I.P. University, Delhi, India |
| 2012-2013 | Assistant Systems Engineer, Tata Consultancy Services Ltd. |
| 2013-2015 | Graduate Student Researcher, Department of Biomedical Informatics, University of California, San Diego |
| 2015 | Master of Science, University of California, San Diego |

ABSTRACT OF THE THESIS

**Preparing PDF Scientific Articles for Biomedical Text Mining**

by

Shitij Bhargava

Master of Science in Computer Science

University of California, San Diego, 2015

Professor Chun-Nan Hsu, Chair

PDF files are not suitable for text mining and must be converted to a plain text format first. For our purpose, we needed text from PDF scientific articles along with section level identification like title, abstract and references. To this end, PDFX is a useful tool which converts PDFs for scientific articles into XMLs, but variability in text quality due to publishing and format of the articles result in incorrect XML that impedes accurate text mining. Additionally, we need to mine PDFs of different types of publications, including manuscripts, research letters, reports and articles which may have radically different formats. Hence we made an ensemble tool to post-process PDFX

XMLs combining multiple sources of inputs from OCR texts, PDFBox and Entrez e-utilities API provided by PubMed to improve quality XMLs of journals. We were able to significantly improve the quality of XML with respect to fidelity of non-alphanumeric characters, segmentation of title, abstract, references and acknowledgment, along with correct word order in text, leading to a data set more suitable for text mining.

# Introduction

PubMed includes about 24 million citations [19] for biomedical literature, but only about 3.4 million [20] of them are available as free text in PubMed Central (PMC) . Although many new publications now provide free text along with the PDF, a substantial number of them do not. Hence right now, researchers wanting to do text mining on a large number of journals in PubMed must work with PDFs instead of plain text directly. There is a variety of tools available which convert PDF to text with some of the popular ones being PDFBox, pdf2text and Adobe Acrobat SDK. These tools can be used to extract the text from a PDF, but do not contain any sectional information directly. To this end, PDFX [5] is a tool made specifically for converting PDF scholarly articles to XMLs which have detailed sectional information like title, abstract, references, tables, acknowledgment and so on, quite similar in schema to the JATS/ NLM DTD format (a.k.a. NXML). Transcribing PDF documents to text accurately is not easy because of a multitude of reasons including, but not limited to publishing and varied sectional formatting (leading to varied reading orders). See [2] for a detailed analysis. Transcribing it to XML is even harder due to the large variety of journals and their sectional organization practices like placement of references (in the end, in footers, with or without heading, etc.), number of columns, reading order and so on. In addition, PDFs of different publication types may have different organizations too. Therefore, while trying to process this variety of PDFs through PDFX for text mining, we encountered issues which directly affected our text mining pipeline. Some of these include quite variable fidelity of non-alphanumeric

characters, for example “=” is often transcribed as “b” in PDFX and “x” (Unicode multiplication symbol) is often transcribed as “â”, along with incorrect, incomplete or missing title, abstract, references and acknowledgment elements. A detailed analysis is given in a later section. These errors were critical to us as we were trying to use PDFX XMLs for tasks such as p-value, number, rsid, gene and disease/ trait tagging as well as ethnicity and sample size extraction after removing references and acknowledgment sections. To address these issues we developed a tool to post-process PDFX XMLs using other text sources like OCR and PDFBox so that these XMLs would be more suitable for text mining. We present details of the methods that we developed in this thesis.

# Chapter 1

# Background and Related Work

PDFs need to be transcribed/ extracted to text based formats like plain text, XMLs and HTMLs for tasks such as text mining and annotation. This problem would not exist in an ideal world, as the PDF itself is generally made from a text source. As the design goals of the PDF standard mainly comprise of presenting documents reliably on their own to the human eye, independent of application software, ease of extracting text is an orthogonal issue not relevant to the PDF standard itself. As explained in the short article titled "Each PDF Page is a painting" in [11], the PDF standard does not attempt to encode any semantic connection between characters in a word or between paragraphs, but characters are "painted" individually at specific coordinates. Because of this, as we will see, PDFs are often published in ways that make it extremely hard or even impossible to extract text with a 100% accuracy. Also, it is important to note the difference between extraction and transcription. "Transcription" means making a written representation of something, while "extraction" just means filtering some information out. For example, people might need to extract all the tables and/ or images from a PDF, but a transcription tool should be able to represent all the visible contents of the PDF in some textual form.

## 1.1 PDF Transcription

This section will survey some of the popular tools and techniques that have been developed to extract text from a PDF or transcribe it to formats like XML and HTML, and the common problems that these tools face.

### 1.1.1 PDF Text Extraction

An ideal text extractor might extract all the textual content from a PDF including images and tables, in an order that makes sense even without the formatting of the PDF. This is hard to do, as the PDF itself barely contains any hints as to what order the text should be extracted in and because extracting text from images is a much harder problem known as Optical Character Recognition (OCR). Hence most text extractors scope themselves down to extracting all the text present in a textual format in a PDF, and putting the responsibility of interpreting the order on the user. Most text extractors use a page segmentation algorithm to first determine all the textual areas of the PDF, page by page. This process largely depends on whitespaces, as there is no inherent way to distinguish boundaries between characters in a word, between words themselves, and similarly between paragraphs. Text extractors often need to distinguish between these boundaries on the basis of the width of whitespaces used, and from these boundaries also determine a reading order. Determining the correct reading order itself is a hard problem due to the large variety of formats PDFs can be in, ranging from single columns to upto 3 or four columns and having intermittent textual areas like headers, footers, tables and captions. It should be noted that some of the problems like segmentation and reading order determination are quite similar to what OCR systems face, and hence text extraction systems have benefited significantly from the well studied field of OCR.

Some of the textual content of a PDF is often represented in a way that it is

impossible for a text extractor to extract it correctly, that is, the same way as it appears in the rendered PDF. This happens because the publishing software might embed a different character than the actual one, but might draw vector graphics over it to make it look the same. Hence the PDF looks correct to the human eye, but the correct information is not present in the "text layer" of the PDF for the extractor to extract. This can often be experienced while copying text from a PDF to a text file, where the copied text looks quite different from what is visible. This is one of the issues that are outside the scope of purely text based extractors, and is largely due to the variability of publishing software. It is technically not a "fault" of the publishing software because the PDF standard doesn't necessarily forbid it, as long as the text appears similar to the human eye.

Some of the popular tools for text extraction are PDFBox, pdf2text and Adobe Acrobat's SDK for text extraction. We decided to use PDFBox as it is open sourced under the Apache License and is being actively developed.

### 1.1.2 PDF to XML Transcription

XML transcription of a PDF in general would involve extraction of text, tables, figures, footers, etc. and then tagging them with appropriate sections/ regions to make an XML. A transcribed XML for a PDF article might then have a tag for the title, the abstract, introduction, tables, etc. The exact format of the XML depends on the specific tool and some tools like PDFX use a schema similar to the JATS/ NLM DTD format, which is a DTD for scientific articles specifically. This specialization of the scope to scientific articles allows the tools to make certain assumptions about the organization of the PDF like the presence of the title and abstract close to each other, presence of bibliographic references in the end, and so on.

PDFX is a free to use rule-based system that transcribes scholarly articles to a JATS like XML format. The key achievement of the tool is that they derive transcription

parameters from the relative font sizes, style and spacing of each article rather than relying on templates or a training phase. Hence, the tool performs quite well on a variety of PDFs in varied scientific domains with reported F-score of 77.45 for top level title identification and 74.03 for individual bibliographic item identification.

## 1.2 Text Post-processing

In this section we will survey some techniques to post-process text to improve the text quality. Most of text post-processing work has been done in the context of OCR systems. As OCR is a well studied domain, there are a variety of techniques available, some of which combine the step of OCR and correction and others which keep them decoupled and separate. We will be focusing on the latter since we want to use similar techniques to post-process text that has already been produced by other text sources, like other text extractors/ XML transcribers.

As most OCR engines classify each character independent of other characters, it is important to post-process the text using some context like statistical language models or syntactic and semantic rules. A system explained in [28] is an example which reduces candidate character search space for characters by using the candidate distance information by an OCR engine in conjunction with an n-gram based language model and a semantic lexicon. Another way to use contextual information to post-process is to model words/ n-grams as sequences of characters and a Hidden Markov Model (HMM) to decide the best sequence, using the given OCR output as the observation. This approach has been used in [27] where transition probabilities of one character to another are calculated from a corpus and emission probabilities are calculated from the output of the OCR engine on a corpus, to statistically record the error patterns of the OCR engine. Using this approach the author was able to boost the accuracy of OCR by about 10%. Another technique as explained in [26] is to try to correct OCR text on a word-level,

by modeling the text as a sequence bigrams (through a word-bigram model) and then using HMM to compute the best word sequence. They are able to reduce the error rate by 60.02% in real OCR text. Also, they are able to learn the character level confusion probabilities for a specific OCR engine and use it to achieve better performance.

There are other issues in text post-processing like correcting reading order, which is correcting the order of words inside or even across sentences, and removing erroneous hyphens from words, which might be inserted due to words wrapping around columns.

## 1.3 Text Similarity Metrics

In this section we will survey some techniques that are used to determine similarity of texts. This is important to measure the quality of post-processed text with some gold standard and determine the improvement upon the raw text.

### 1.3.1 String Similarity Metrics

Approximate string matching (also called fuzzy string matching) can be used to measure how similar two strings are. One natural technique for this is the edit distance, which is counting the minimum number of editing operations to transform one string into the other. This has been used in automatic spelling correction to determine the best candidate for a misspelled word from a dictionary and also for tasks like quantifying similarity of macromolecules like DNA which can be viewed as string of letters of A,C, G and T. A common way to define edit distance is the Levenshtein Distance, where the allowed operations are removal or insertion of a single character, and the substitution of one character for another. The distance can be normalized by string lengths to obtain a similarity score between two strings by the following formula:

$$\frac{LevenshteinDistance(string1, string2)}{MAX(length(string1), length(string2))}$$

where *string1* and *string2* are the strings that we want to measure the similarity of, and the function *length* is the number of the characters in a string.

Another simple way to quantify the similarity of two strings is the Jaccard distance, with the disadvantage being that the order of characters in the two strings is not considered in the metric. The Levenshtein distance on the other hand implicitly considers the order of characters as well.

### 1.3.2 N-gram Based Similarity

Another efficient technique for measuring text similarity is to break the text into n-grams and then compare sequences of n-grams. The sequences can be scored by Jaccard distance or by representing the texts as a vector of n-gram frequencies with a dimension of $(\text{alphabet size})^n$ and then calculating cosine distance between the vectors. This technique has been used in [7] to develop a system that uses similarity measure between documents to sort and cluster them. Similar n-gram based techniques are also popular in plagiarism detection, like in [16], the authors detect copying by measuring trigram based similarity, exploiting the observation that in original text, most trigrams should be unseen. Hence if a text has a significantly higher ratio of seen trigrams, it might indicate plagiarism. Some papers have also used n-gram based author profiles for intrinsic plagiarism detection, when a reference corpus is not present. See [12] and [24].

## 1.4 PDF Metadata Extraction

Many systems have been developed that extract metadata such as authors, year of publication, journal name, and bibliographical information from natural language text or from PDFs directly. The CERMINE system as published in [25] is an example that uses supervised and unsupervised machine learning techniques to extract parsed bibliographic references and metadata directly from PDFs. They use features like presence of a

particular character class, like lower and uppercase letters or special characters like dot, bracket, comma, etc. along with presence of tokens in dictionary of cities or words commonly occurring in journal titles to extract and parse reference strings. Another system, FLUX-CiM as published in [6] uses unsupervised, non-template methods to extract citation components from articles.

# Chapter 2

# Survey of Issues With PDF to XML Transcription

Depending on how PDFs are published for journals, a PDF to text based converter's accuracy varies significantly. For example, many non-alphanumeric characters might be extracted as a completely different character from what appears in the PDF. As PDFX uses a text extraction library underneath (Utopia documents), it is also susceptible to such problems. Similarly, due to the huge variety of ways in which a PDF article may be organized in, it is hard for PDFX to identify sections correctly every time, though it performs well in general. Some of these issues turned out to be important for our text mining pipeline and hence we needed to correct them. For instance, accurate detection of title and abstract sections is critical for tasks such as finding target disease for a GWAS report and determining what the ethnicity and sample size of test subjects is in initial/ replication stages of the experiment, as we found that title and abstract are important features for both. Similarly, accurate reference and acknowledgment detection is important as a significant number of mentions of diseases and genes not necessarily related to the paper being analyzed are present in the references and acknowledgment section. Hence we want to reliably remove them from the XML. This chapter analyzes some of the issues which were found to be important for our text mining pipeline.

## 2.1 Title and Abstract

In PDFs of regular research articles, PDFX performs well in detecting title and abstract, but in manuscript PDFs as the first page might have contents other than title or abstract, PDFX struggles to identify them correctly.

## 2.2 References and Acknowledgment

The format of references varies considerably with the most common being a section in the end with or without the heading "Reference" (see e.g., PubMed ID: 19169254 [18]) or sometimes having references inside footers continuing alongside the content body. Some article PDFs might have additional content after references like diagrams or plots along with captions, or appendix (see PubMed IDs: 19188921 [14], 15761122 [13]). As PDFX is a purely rule-based system it often falters in recognizing references when the common pattern of having references in the end with a heading is not present. The authors of PDFX try to overcome this by providing provision for customizing multiple "cue" files for abstract, bibliography, captions, etc. in which you can give hint words/ phrases to help PDFX identify references, but such a rule based strategy cannot handle the aforementioned variety of reference formats. We noticed similar problems in identifying acknowledgments as well.

## 2.3 Special Character Errors

We define the set of special characters for our purposes as all the characters which are:

1. Not in a-z, A-Z, 0-9;

2. Not English punctuation (as defined by Python's string.punctuation, except <,>,=

and + as they are mathematical operators);

3. Not a whitespace character.

Some examples of special characters include mathematical symbols like "=", "×" (unicode equal and multiply), letters from other languages like Greek: "$\alpha$", and other miscellaneous unicode characters like the horizontal ellipsis: "...". PDFX and all PDF to text converters suffer from this problem depending on the publishing of PDFs. An example would be a string like this encountered in PDFX: "5 â 10-5", which is actually "$5 \times 10 - 5$", hence the unicode multiply and minus symbols were incorrect. Other examples are "hemoglobin-B" which should have been "hemoglobin-$\beta$" and similarly "A-value", which should have been "$\lambda$-value".

## 2.4 Jumbling Errors

Occasionally PDFX jumbles word order in or across sentences. This problem is not unique to PDFX though, as we noticed that using Tesseract (an open source OCR tool) to extract text also gives rise to such errors, meaning the reading order detection/ segmentation in both does not work well sometimes. As explained in Section 1.1.1, this is not an easy problem due to the variety of formats PDFs can be in. Some of which are even unintuitive for a human. We have seen this mostly happening in sentences that are present in the caption of a figure or in tables. An example is given in Figure 2.1, where the exponents of numbers in a table are extracted as a separate column and hence are jumbled with respect to the actual text.

## 2.5 Word-Boundary Errors

Often words that straddle a column boundary in an article have a hyphen inserted in between them. This is not an error on part of PDFX strictly speaking, since the text

A. Discovery group.

| dbSNP | Chromosome | Physical Position | Gene Relationship | Gene Distance | Gene | BP Change | Prob Allele | Prob Genotype |
|---|---|---|---|---|---|---|---|---|
| rs6975107 | 7 | 120168143 | intron | 0 | KCND2 | C-> T | $4.15x10^{-09}$ | $3.74x10^{-09}$ |
| rs318125 | X | 97090551 | downstream | 348,298 | DIAPH2 | C-> T | $4.35x10^{-09}$ | $1.25x10^{-05}$ |
| rs5916727 | X | 104294270 | intron | 0 | IL1RAPL2 | C-> T | $6.66x10^{-09}$ | $1.34x10^{-05}$ |
| rs11863929 | 16 | 86861934 | upstream | 159,445 | ZNF469 | C-> G | $1.77x10^{-08}$ | $5.48x10^{-07}$ |
| rs1578826 | X | 85889112 | intron | 0 | DACH2 | T-> C | $2.20x10^{-08}$ | $5.46x10^{-06}$ |
| rs7616661 | 3 | 5940543 | downstream | 703,894 | EDEM1 | T-> G | $4.82x10^{-08}$ | $3.37x10^{-08}$ |

```
<region class="DoCO:TextChunk" id="586" confidence="possible" page="26"
column="1">A. Discovery group.</region>
<outsider class="DoCO:TextBox" type="sidenote" id="587" page="26"
column="1">-09 -05 -05 -07 -06 -08</outsider>
-<region class="unknown" id="588" page="26" column="1">
   -09 rs6975107 7 120168143 intron 0 KCND2 C-> T 4.15x10 3.74x10 -09 rs318125 X
   97090551 downstream 348,298 DIAPH2 C-> T 4.35x10 1.25x10 -09 rs5916727 X
   104294270 intron 0 IL1RAPL2 C-> T 6.66x10 1.34x10 -08 rs11863929 16 86861934
   upstream 159,445 ZNF469 C-> G 1.77x10 5.48x10 -08 rs1578826 X 85889112 intron 0
   DACH2 T-> C 2.20x10 5.46x10 -08 rs7616661 3 5940543 downstream 703,894 EDEM1
   T-> G 4.82x10 3.37x10
</region>
```

**Figure 2.1.** A kind of jumbling error in a table. The table at the top is what is present in the PDF. The lower portion shows the transcribed XML. The exponents marked in the red box are extracted as a separate column in the XML.

correctly reflects what is present in the PDF, but we want to correct such errors as much as possible.

The content in this chapter in part is currently being prepared for submission for publication of the material. The thesis author was the primary investigator and author of this material.

# Chapter 3

# Goals and Terminology

The goal of this study is to transform a PDFX XML to a better quality XML that is better suited for our text mining tasks, which include tagging mentions of entities such as gene, disease, p-value, and number as well as extraction tasks including target disease extraction, study sample size, ethnicity and stage from articles of GWAS (Genome Wide Association Study). Quality is measured on exactly the issues discussed in Chapter 2, and more details will be given in the next section.

Given a PDF and its PubMed ID, our system, for the purpose of correction, takes additional sources of text as input from

1. PDFBox transcribed text of the given PDF,
2. Tesseract OCR transcribed text of the given PDF,
3. Text of the PubMed ID retrieved by querying the Entrez e-utilities API provided by PubMed.

For measurement of quality, we use XML versions of articles provided in PubMed Central (PMC) called NXMLs as gold standards. Hence for each of the quality metric that we will discuss in the next section (with the exception of references and acknowledgment), we can measure how a corrected/ raw PDFX XML performs by scoring with respect to

the NXML. An overview of the system is given in Figure 3.1. The system has two main components: a correction component and a scoring component.

Here onwards, we will refer to text extracted from NXMLs as the *gold text* and text extracted from PDFX XML as the *test text*. Similarly we will refer to the abstract extracted from NXML as *gold abstract* and abstract extracted from PDFX XML as *test abstract*. We will use similar phrases for title, abstract and acknowledgment. Figure 3.1 summarizes the different text sources that we have, what behaves as the gold standard and what we are trying to correct.

Additionally, we use two terms for n-grams:- overlapping n-grams and non-overlapping n-grams. By non-overlapping n-grams, we just mean disjoint segments of size n from the text. For example, for the text: “For he’s a jolly good fellow”, the overlapping bigrams (n=2) are: “For he’s”, “he’s a”, “a jolly”, “jolly good” and “good fellow”. And the non-overlapping n-grams will just be segments of size 2: “For he’s”, “a jolly” and “good fellow”. Text is segmented into non-overlapping n-grams when we need to reconstruct it easily by just stringing together the segments with space.

The content in this chapter in part is currently being prepared for submission for publication of the material. The thesis author was the primary investigator and author of this material.

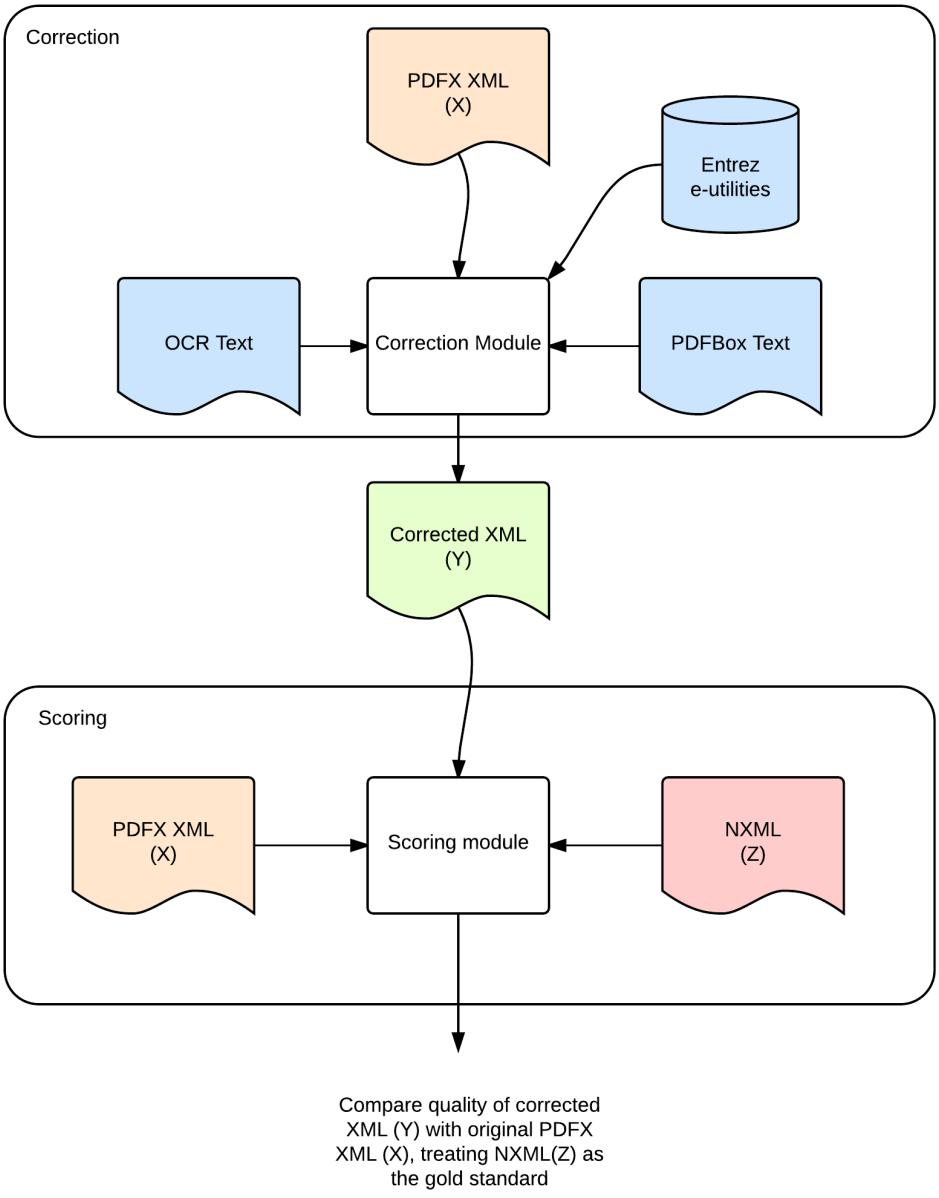


**Figure 3.1.** The overall system consists of a correction module and a scoring module. The figure shows the input and output of each module.

# Chapter 4

# Transcription Quality Metrics

To be able to tell if the quality of XMLs is improved after some kind of correction method, we first need ways to define and measure quality. To do this, we developed different metrics to measure quality of XMLs for different aspects, some of which directly correspond to the text mining tasks we were going to do afterwards. We measure quality against text extracted from NXMLs provided by PubMed Central. In most metrics we use an n-gram based similarity method, which is common in plagiarism detection as explained in Section 1.3.2, though our task of measuring syntactic similarity between two texts is much simpler than plagiarism detection. The idea is to make two sets of n-grams from the gold and test text and calculate true positives, false positives, true negatives and false negatives, where a true positive is counted when an n-gram in the gold n-gram set is present as it is in the test n-gram set. The n-grams are overlapping segments of text and hence such a scoring implicitly includes measuring presence as well as correct order of words. An example of such n-gram based measurement of text quality is given in Figure 4.1.

## 4.1 Title and Abstract

We use the n-gram based similarity metric as shown in Figure 4.1 to obtain an F-score measuring the quality of extracted title/ abstract against those present in the

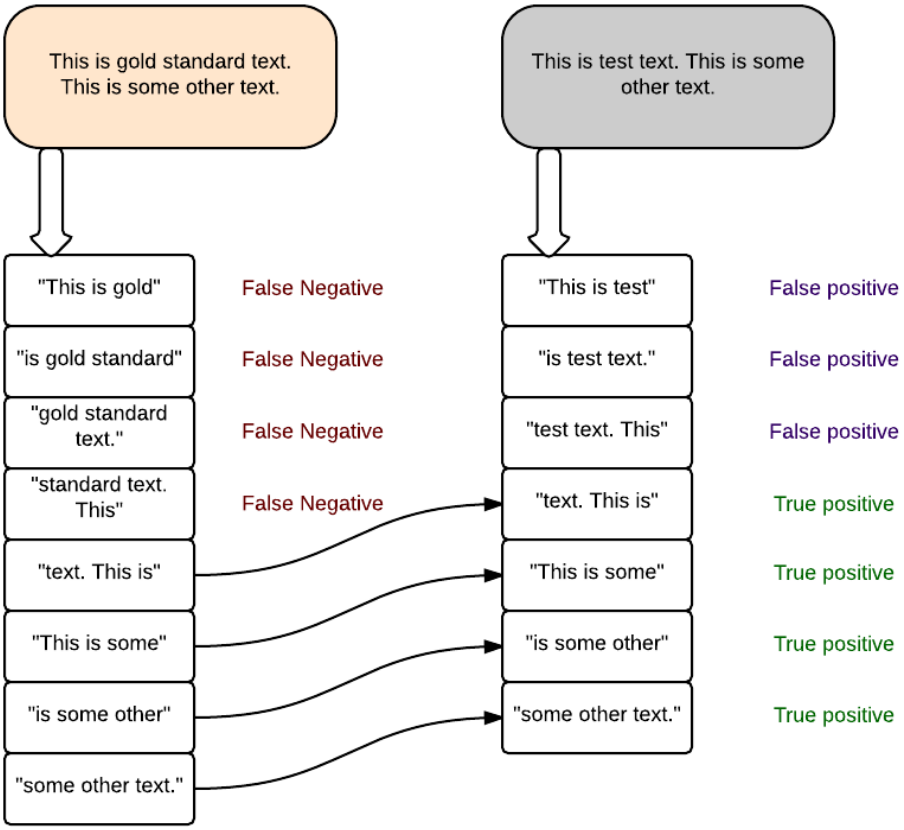


**Figure 4.1.** N-gram based similarity F-score calculation. Precision is $\frac{4}{7} = 0.57$ while recall is $\frac{4}{8} = 0.5$, and hence the F-score is $\frac{2 \times 0.57 \times 0.5}{0.57 + 0.5} = 0.53$

NXMLs for each article in the test set and then take the average F-score over all the articles to obtain a macro F-score.

## 4.2 Reference and Acknowledgment

The way references are present in NXMLs and PDFs are quite different. Often NXML references don't have the complete title or the serial number of a referenced paper in the reference list (reference tags are used instead). Therefore we cannot use NXML reference text as a gold standard. Additionally, acknowledgment is not present as a tag in NXMLs. Therefore, we built our own gold standard for reference and acknowledgment text scoring from PDFs directly. For 70 randomly selected PubMed articles, we manually compiled reference and acknowledgment text from the PDFs for each paper, and we measure the quality of reference/ acknowledgment text extracted from corrected/ raw PDFX XML against this manually compiled gold standard using n-gram based text similarity as shown in Figure 4.1.

## 4.3 Special Characters

We extract special characters, as defined in Section 2.3, from both gold text and test text and calculate the micro F-score, that is, using the counts of True and False Positives and False Negatives over all documents in the test set. This is better than macro F-score, which averages individual F scores for each article, since the number of special characters in an article can vary considerably and we want to have an overall picture of special character transcription quality.

## 4.4 Jumbling

We count the number of jumbling errors as defined in Section 2.4 for each article in the test set. Then we average the counts over all articles to obtain the average number of jumbling errors per article.

## 4.5 Word-Boundary

We extract all the words with a hyphen (or a kind of unicode dash like “en dash” and “em dash”) in them (but no digits) in the gold text and do the same for test text and then calculate the micro F-score.

## 4.6 Overall Text Quality

This score is only meant to analyze how the specific corrections are changing the text overall. It is important to note that the overall text quality, which includes factors such as formatting is not necessarily important to us. This is the reason we devised specialized metrics which would help us gauge how good the text is for our text mining purposes. We use n-gram based similarity for measuring overall text quality, as explained in Figure 4.1.

The content in this chapter in part is currently being prepared for submission for publication of the material. The thesis author was the primary investigator and author of this material.

# Chapter 5

# XML Correction

The corrections are run as a pipeline in the order shown in Figure 5.1. This means that each stage takes the XML output from the previous stage, runs corrections on it and then gives the output to the next stage. Each individual correction stage is described in the sections that follow.

## 5.1 Title and Abstract Correction

As our focus dataset is only the documents found in PubMed (more specifically, only GWAS articles), we used the Entrez e-utilities API to retrieve the correct title and abstract and substituted them for whatever was present in PDFX XML. Although this method results in correct title and abstract, it is possible that if PDFX had completely misclassified a sentence as title and the actual title was somewhere else, then there would be two copies for the title sentence. The same thing can be said for abstracts. However, this possible duplication was not a big concern for us and hence we ignored it. A flow diagram illustrating this is given in Figure 5.2.

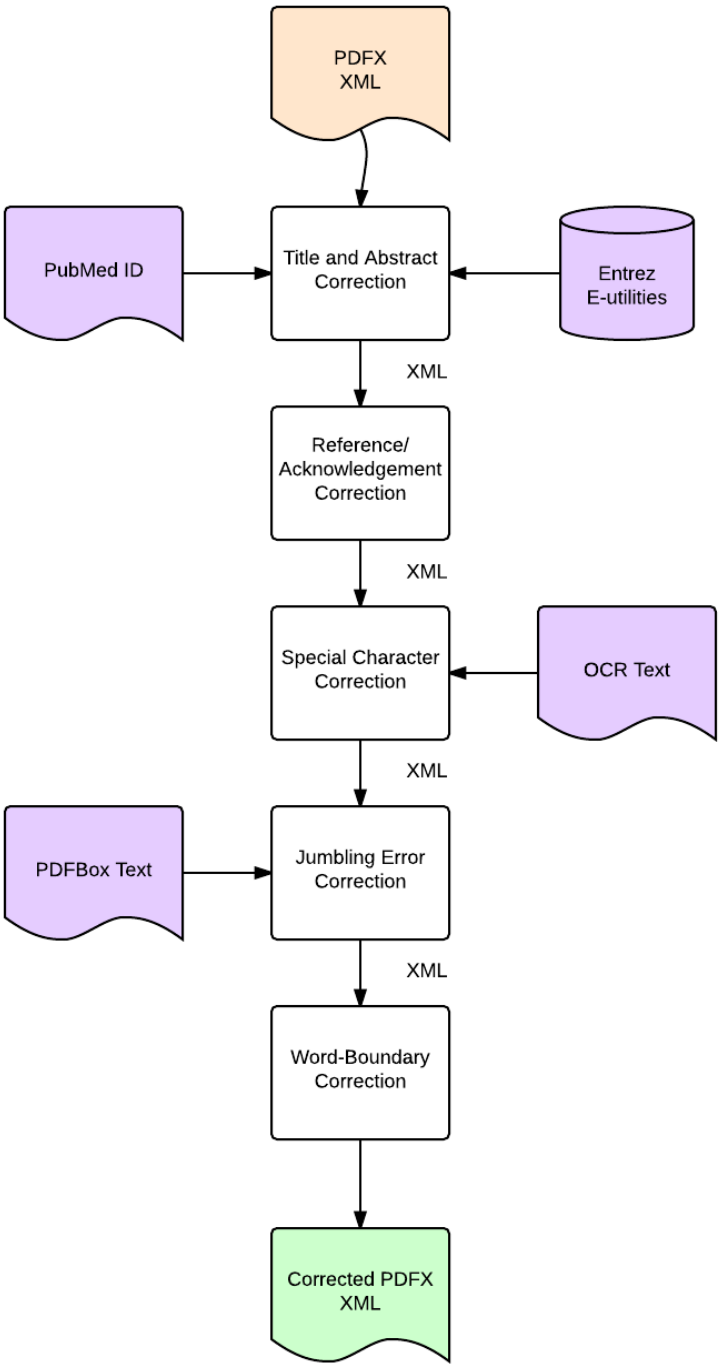


**Figure 5.1.** A flow diagram showing the stages of correction and their inputs. Stages are arranged in a pipelined fashion and take an XML as input and return a corrected XML as output.

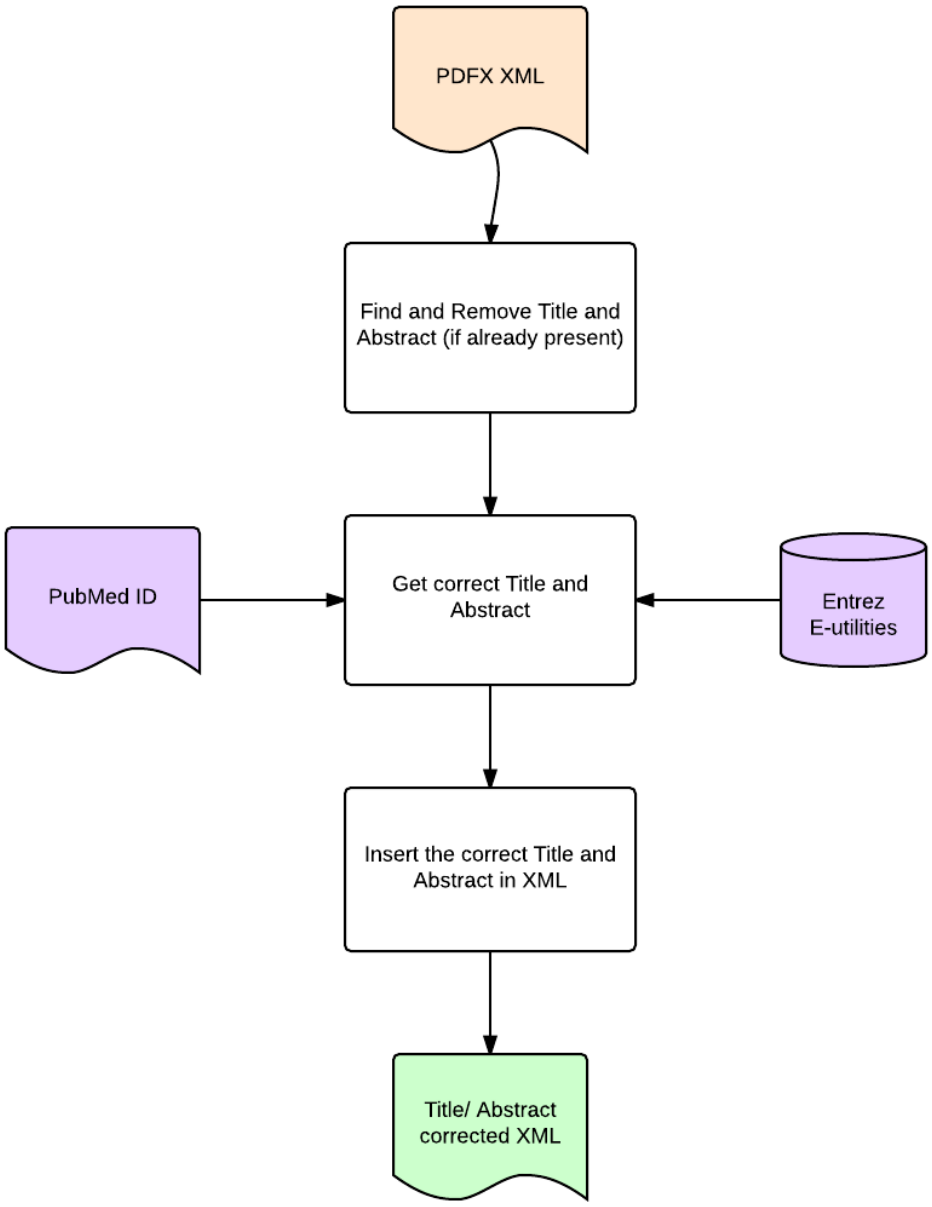


**Figure 5.2.** A flow diagram showing the title and abstract correction.

## 5.2 Reference and Acknowledgment Correction

A key observation was that whenever PDFX identifies references/ acknowledgment in a paper, it is generally correct and we only need to fix the case when it does not identify either. We realized that the text content in references and acknowledgment showed clear patterns in language, and because for our purpose, we did not need to extract individual bibliographic items in the references section like PDFX does, classifying each section as a reference/ acknowledgment was enough. It is important to note that as we only want to detect reference text, which is much more coarse grained and simpler than complete metadata extraction from journals, or metadata extraction for each citation, as explained in Section 1.4. Although we clump references and acknowledgment together (and then remove these from the XML), we made two separate classifiers for them, as the patterns they show are different. Figure 5.3 shows how we trained a classifier for reference classification and Figure 5.4 shows how we actually use the classifier to correct references in an XML.

The same training/ correction flow was used for acknowledgment classifier as well. We used 451 PDFX XMLs which had a PDFX reference related tag to train the reference classifier, by taking reference text as positive examples and rest of the text from the paper as negative examples. We did the same with 418 papers which had an acknowledgment tag in their PDFX XMLs. The following are the list of features in order of their importance for reference classification:

1. Density of numbers followed by dot
2. Density of year-like numbers ($> 1950$ and $< 2016$)
3. Density of journal shorthand name mentions
4. Density of “et al.” strings

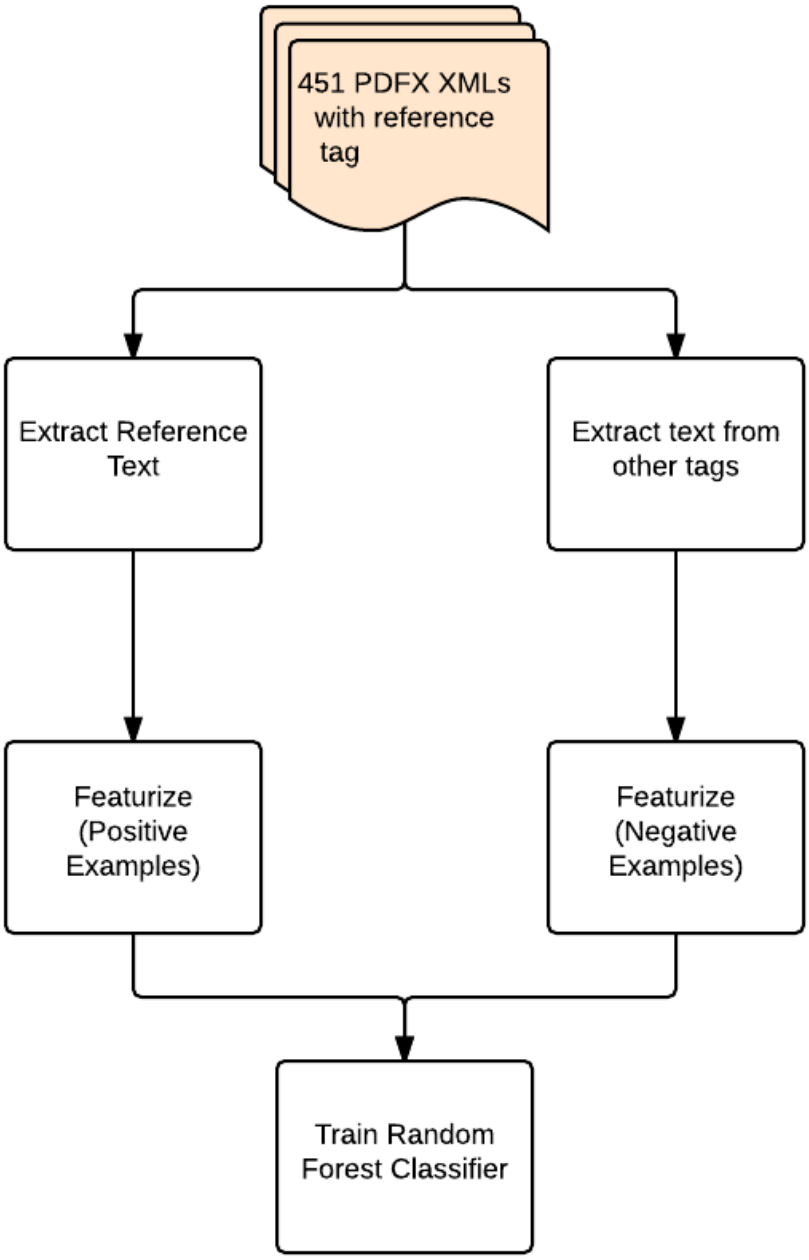


**Figure 5.3.** A flow diagram showing how reference classifier is trained.

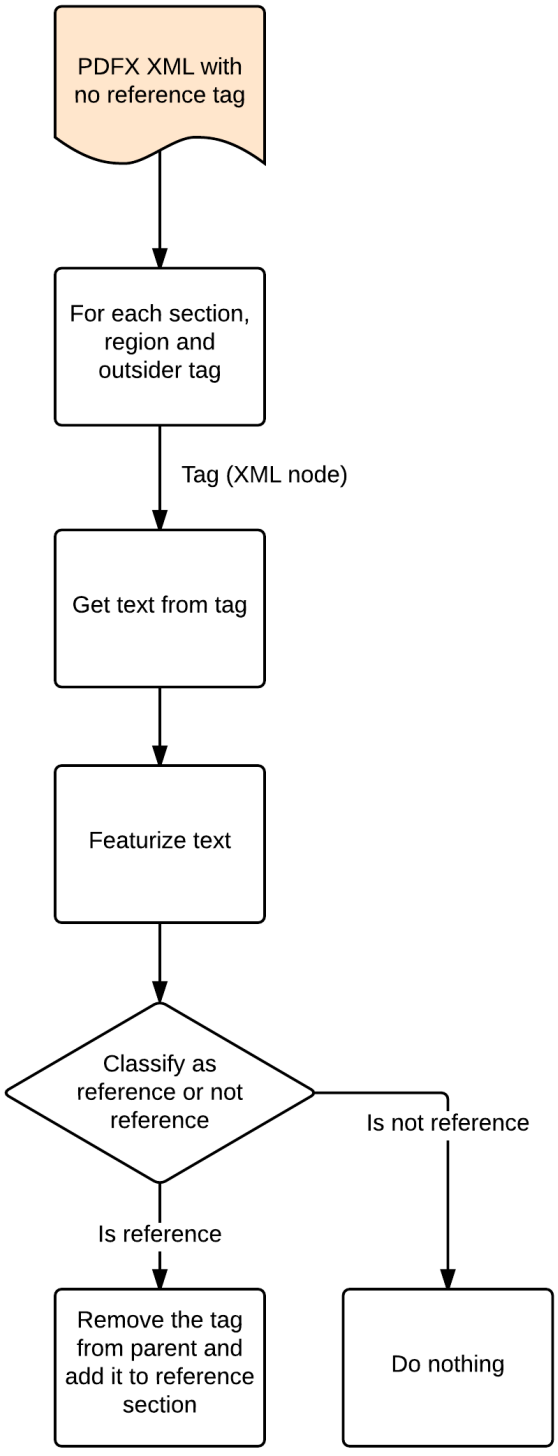


**Figure 5.4.** A flow diagram showing how reference correction is performed using the trained classifier.

5. Density of numbers appearing with a hyphen (i.e. to capture page number - page number pattern)
6. Density of numbers followed by colon
7. Density of year-like numbers in brackets
8. Density of author name mentions
9. Density of doi mentions

Note that each feature listed above is a ratio of the number found vs total number of tokens in the text (and not raw numbers), hence we use the word "density".

The first feature turned out to be important as most journals/ research articles give references as a numbered list (1., 2.,3., etc.). For the third feature, we made a list of journal shorthand names like "J Med Genet" which is a shorthand for "Journal of Medical Genetics" from the free text release archives (http://www.ncbi.nlm.nih.gov/pmc/tools/ftp/) by PubMed Central where each journal folder is named by the shorthand in the archives. Similarly for the eighth feature, we collected authors' first names and last names from the free text release from about 1 million NXMLs which have a separate tag for the first and last name of each author of a paper. Features 5 and 6 were useful because phrases like "J Med Genet. 1987 24 6: 382-383." appeared often. The other features are self-explanatory. With these features, in our training set we reached classification accuracies of over 90% on random 50-50 train-test splits using a Random Forest Classifier in multiple trials.

Acknowledgment detection was an easier problem as phrases like "funded by", "express gratitude" and so on are common. Hence we used a Random Forest classifier with only features being TFIDF followed by Latent Semantic Analysis (Truncated SVD) with 50 components. The LSA step gave us a boost in accuracy of about 4% on a random 50-50 train-test split, with final accuracy over 90%. Recall that the classification

accuracies mentioned here are about binary classifying each PDFX segmented XML section into reference/acknowledgment or not, which is not the n-gram based metrics we use to measure reference/ acknowledgement quality, but are given only for the sake of completeness. The actual reference/ acknowledgment scores, as defined in Section 4.2 are shown in the results section. We used the scikit-learn library in Python [21] for implementing our classifiers.

## 5.3 Special Character Correction

Special character correction proved to be quite tricky. As explained in Section 1.1.1, the problem is that quite often special characters in a PDF are painted instead of coded. The basic idea for special character correction is that given a possibly erroneous n-gram from PDFX in the token/word level, we try to find the corresponding n-gram in OCR and align the two n-grams by their characters. Then for differing characters we make a decision about what character should be present in that position using HMM and language models. This pipeline is summarized in Figure 5.5, and a description of each step is given next.

### 5.3.1 Selecting the most similar n-gram from OCR text

Similarity of two n-grams, a and b, is measured by Levenshtein distance and normalized for length according to the formula shown in Section 1.3.1. If we find an n-gram in OCR text that is at least 0.6 as similar to the PDFX n-ngram, we go to the next step. Otherwise we give up on trying to correct this n-gram, as it means a suitable reference n-gram in OCR is not present.

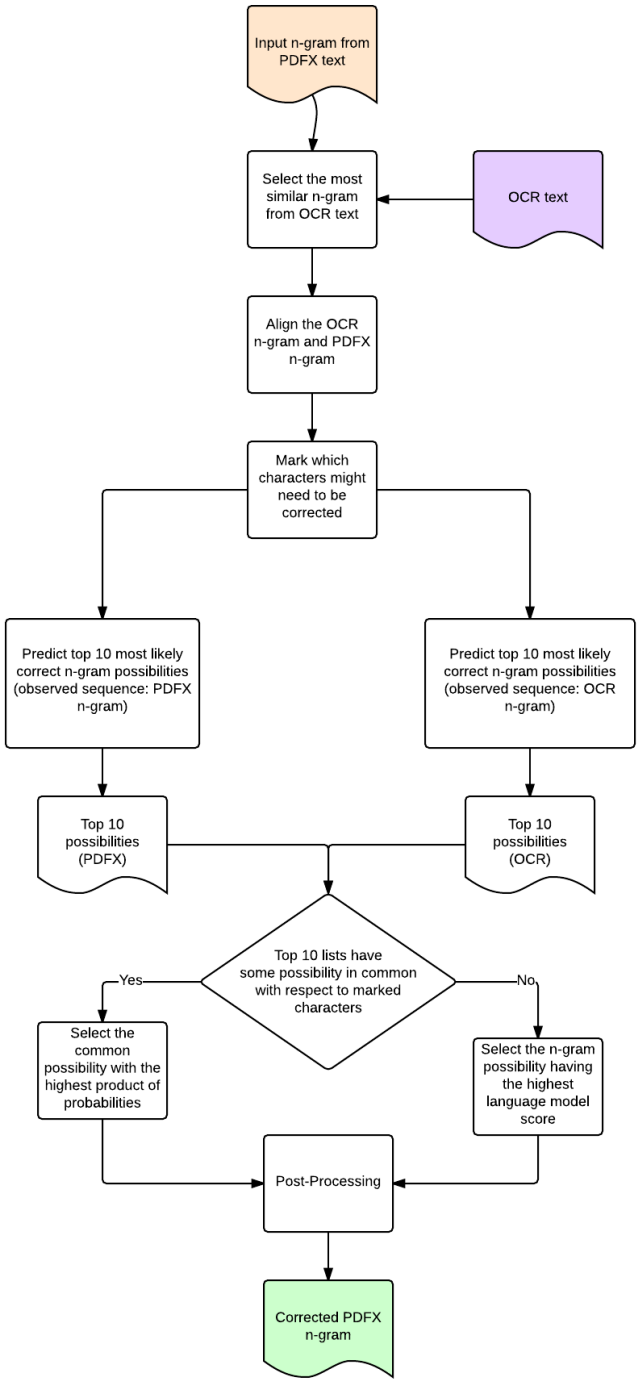


**Figure 5.5.** A flow diagram showing steps taken in correcting special character errors in an n-gram.

### 5.3.2 Aligning the PDFX n-gram with the OCR n-gram

We use the "globalms" method in the Biopython library [4] for global gene sequence alignment of two n-grams at a character level, with the following scoring scheme for alignment:

1. Matching character = +2 points
2. Different character = -1 point
3. Opening a gap = -0.5 point
4. Extending a gap = -0.1 point

The scoring scheme is designed such that the penalty for opening a gap is greater than that of extending it. As a result, this scoring scheme tends to give alignments where gaps are in long contiguous chains, and hence avoids matching a character in one word with a character in a different word. If multiple alignments are found at this step, we take the alignments with the minimum "fragmentation" of words, or equivalently alignments that have the least number of words in them assuming a gap as being equal to space. If there is again a tie, we take the alignment with the most number of leading plus trailing gaps. This scheme gives us well aligned n-grams most of the time, as can be seen in Figure 5.7 and Figure 5.8.

### 5.3.3 Marking characters for correction

The alignment in the last step will tell us which characters differ in the two n-grams, and they are the candidate positions for possible corrections. But not all mismatches are considered opportunities for correction. For example, if a character in one n-gram is aligned with a gap in the other n-gram, it generally means that the n-grams were too different and could not be aligned properly (possibly because of differences in

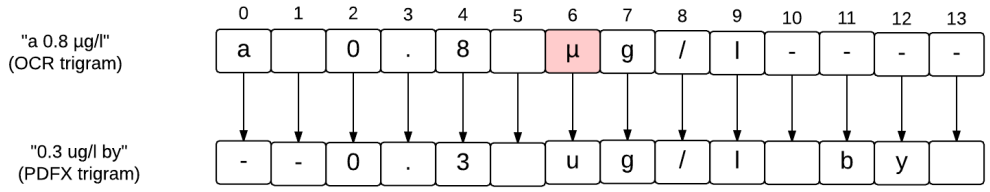


**Figure 5.6.** An illustration showing how characters to be corrected are marked after alignment.

whitespaces, as shown in Figure 5.6). Hence following cases are the exceptions when mismatched characters are not considered for correction:

1. Either of the characters is an alignment gap
2. Either of the characters is a whitespace character
3. Both of the characters are digits

The last exception is because we noticed that we sometimes end up changing ("correcting") an already correct digit to something else. OCR often makes mistakes in digit recognition, but PDFX does not, hence this is a heuristic to avoid such erroneous corrections.

### 5.3.4 Using Hidden Markov Models (HMM) to predict top N possible correct n-grams

At this step we have one n-gram from OCR and one from PDFX, and we have to predict what the corresponding n-gram in NXML would be. The following example takes sample values for these 3 kinds of n-grams, and will be referred to in the explanation that follows.

**Example 5.3** Let us assume that we have:

1. An OCR n-gram, for example "the $\alpha$-value wa3 less than 0.01".
2. A PDFX n-gram, for example "the a-valae was less than 0.01".
3. The true n-gram in the NXML gold standard for our example is "the p-value was less than 0.01".

The problem at hand is to search for the true n-gram in NXML, given a pair of OCR and PDFX n-grams. Our solution is to train an HMM [17] using a training corpus of aligned OCR, PDFX and NXML texts, and infer the most probable NXML n-gram by the Viterbi algorithm.

While formulating this problem in terms of HMM Viterbi inference, one of our assumptions is that characters in the words of meaningful sentences follow the Markov assumption to a good extent, something which is already well proven by handwriting and speech recognition applications, which have used HMMs to achieve good performance [10] [22]. The other assumption is that PDFX and OCR have a statistical pattern or bias in the character level errors that they make, and that these patterns can be learned through a training corpus. The same technique of modeling words as sequences of characters by HMM has been used to boost OCR accuracy in [27].

**Problem definition**

To formally state the problem, we first define a character $x_t$ as the t-th letter in an OCR n-gram. Therefore, each OCR n-gram can be represented as $X = x_1 x_2 ... x_N$, which is a concatenation of N characters. It should be noted that the character whitespace (i.e., " ") is also treated as a character. In Example 5.3, the number of characters in the OCR

n-gram "the $\alpha$-value wa3 less than 0.01" is 30.

Then we define mismatch character as the characters not with consensus between an OCR n-gram and corresponding PDFX n-gram. In Example 5.3, the mismatch characters in OCR n-gram are "$\alpha$", "u", and "3". Next, we compute an ambiguity dictionary, which is a dictionary that records occurrences of observed mismatched characters in the corpus. It can be thought of as a list of tuples of the form: (observed character, actual character, frequency) For example, a tuple ("$\alpha$", "p", 30) means that for 30 times when we saw "$\alpha$" in OCR/ PDFX text corpus, the actual character in NXML was "$p$". Multiple tuples for an observed character, like "$\alpha$" in our example can be aggregated and written as:

"$\alpha$": [("$p$", 150), ("$\alpha$",30), ("$a$", 5)],

which means that for "$\alpha$" in OCR/ PDFX n-gram, the true character should be a "$p$" 150 times, "$\alpha$" 30 times and "$a$", 5 times.

Thus, each mismatch character (from OCR n-gram) $x_t$ corresponds to a candidate-character set $\delta_t$, which can be computed from the ambiguity dictionary. For non-mismatch characters, the candidate character set $\delta_t = \{x_t\}$, contains only one element which is exactly $x_t$. In our OCR n-gram example, the candidate-token set of the first character "t" is "t", while that of the fifth character "$\alpha$" may be {"$\alpha$", "a", "p", "o", "0"}, for example.

Therefore, our problem can be regarded as to choose the "most probable" candidate character for each mismatch character in OCR-n-gram, which can maximize the overall likelihood to recover the NXML-n-gram. In other words, we want to find $Y* = y_1 * y_2 * ...y_N$ = $argmax_Y\ P(Y|X)$, where $y_1 * y_2 * ...y_N$ are the "most probable" candidate characters given $X$, the pair of OCR and PDFX n-grams. In the rest of the paper, we simplify the notation by using P(Y) as the target probability that we want to maximize.

**Solution**

We apply Hidden Markov Model (HMM) to solve our problem. Based on the Markov assumption, the probability we want to maximize can be written as P(Y) = $P(y_1 y_2 ... y_N) = P(y_1)P(y_2|y_1)P(y_3|y_2)...P(y_N|y_N - 1)$. The model of our HMM is defined as follows:

1. Observation

   $\sigma = p_1, p_2, ..., p_M$, which is a set of all possible M values for all characters.

2. State

$$A = \bigcup_{t=1}^{N} \delta_t = \{q_1, q_2, ...q_K\}$$

   which is a set of all possible K values for all candidate characters.

3. Initialization Probabilities

   $\Pi = \{ \pi_i | \pi_i = P(y_t = q_i)\}$, which is a set of initial probability for each state. Therefore, $|\Pi|$ = K. Suppose $L^1(q_i)$ is the conditional probability given by 1-gram language model (trained from the NXML n-grams) for each candidate-character $q_i$ (e.g., "$\alpha$"), we compute $\pi_i$ as follows:

   $\pi_i = L^1(q_i)$

   Note that $\sum_{i=1}^{K} \pi_i = 1$.

4. Transition Probabilities

   A = $\{a_{ij} | a_{ij} = P(y_{t+1} = q_j | y_t = q_i)\}$, which is a set of probability transiting from the t-th candidate character $y_t$ with state $q_i$, to the (t + 1)-th candidate-character $y_{t+1}$ with state $q_j$. Therefore, $|A| = K^2$. Suppose $L^2(q_i, q_j)$ is the conditional probability given by 2-gram language model (trained from the NXML n-grams)

for two consecutive candidate-characters $q_i$ and $q_j$ (e.g., "$\alpha$-"), we compute $a_{ij}$ as follows: $a_{ij} = L^2(q_i, q_j)$.

Note that , $\sum_{j=1}^{K} a_{ij} = 1$ for all states i = 1, 2, ..., K.

5. Emission Probabilities

   B = $\{b_{ik} | b_{ik} = P(x_t = p_k | y_t = q_i)\}$, which is a set of probability for the t-th candidate-character $y_t$ with state $q_i$, to emit the t-th character $x_t$ with observation $p_k$.Therefore, $|B|$ = K*M. Suppose $C(p_k, q_i)$ is the count that a candidate-character $q_i$ (e.g., "p" from NXML n-gram) is recognized as a character $p_k$ (e.g., "$\alpha$" from OCR n-gram), and $\lambda$ is a regularization constant, we compute $b_{ik}$ as follows:

$$b_{ik} = \frac{C(p_k, q_i) + \lambda}{\sum_{k'}^{M} (p_{k'}, q_i) + \lambda}.$$

   Note that, for all states i =1, 2, ..., K. In our example ambiguity dictionary for character "$\alpha$" = [("p", 150), ("$\alpha$", 30), ("a", 5)], the emission probability from "p" to "$\alpha$" (i.e.,the probability that we observe "$\alpha$" in OCR n-gram, but the true character in NXML n-gram is "p") can be estimated as ((30 + $\lambda$) /(150 + 30 + 5 + $\lambda$)).

It should be noted that all language model we use above is in the character-level. That is, we treat each character (including the space) as 1-gram. Finally, we apply the Viterbi algorithm with our HMM model to decode the N most probable Y*.

In this way, we have the top N most probable NXML n-grams given an OCR n-gram as the observation and using PDFX n-gram just to see which characters are not in consensus. We follow the same process for PDFX as well, that is, treating the PDFX n-gram as the observation and using OCR n-gram to see which characters are not in consensus. Similarly, the ambiguity dictionary for PDFX was derived by applying exactly

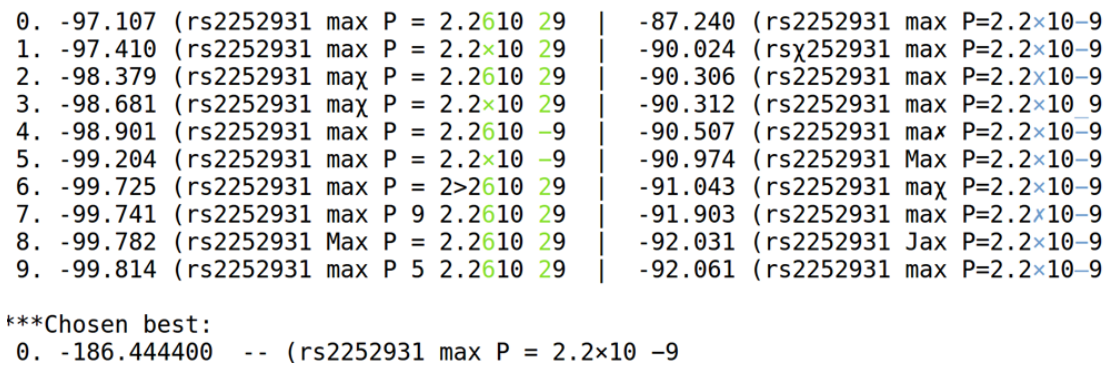


**Figure 5.7.** The screenshot shows two top N sorted list of the most probable NXML n-grams with their log probabilities given by Viterbi inference. Candidate number 5 in the left list (where PDFX n-gram is the observation) matches candidate number 0 in the right list (where OCR n-gram is the observation), which is the "chosen best".

the same method, except that we record ambiguities for PDFX instead of OCR. Hence, following the same process for PDFX observation, we can compute another list of the top N most probable NXML n-grams along with their probabilities.

### 5.3.5 Predict the most probable NXML n-gram from two top 10 lists

Now, we select the most probable NXML n-gram as the n-gram common in both lists (with respect to marked characters only) which has the maximum combined probability. A similar approach was applied to boost the performance by integrating two models previously for gene mention tagging [9]. This is illustrated in Figure 5.7.

The PDFX n-gram in Figure 5.7 is "rs2252931 max P = 2.2610 29" and the OCR n-gram is: "rs2252931 max P=2.2x10_9". The characters highlighted in green/ blue are the characters marked for correction. We won't apply correction to other characters, but instead just keep whatever we observe in the PDFX n-gram because other mis-matches are pairs of digits. The left list of 10 candidates is from treating the PDFX n-gram as the observation, while the right side shows the candidates from treating the OCR n-gram as the observation. The numbers in negative are the log probabilities for each sequence as

```
 0. -76.754 P = 9.0610 29 Beta 21.9  |  -75.916 P= 9.0×10−9 Beta −1.9
 1. -78.122 P = 9.0×10 29 Beta 21.9  |  -76.056 P= 9.0×10−9 Beta −1.9
 2. -78.548 P = 9.0610 −9 Beta 21.9  |  -76.194 P= 9.0×10−9 Beta −1.9
 3. -78.716 P = 9.0610 29 Beta −1.9  |  -77.299 P= 9.0X10−9 Beta −1.9
 4. -79.388 P 9 9.0610 29 Beta 21.9  |  -77.439 P= 9.0X10−9 Beta −1.9
 5. -79.460 P 5 9.0610 29 Beta 21.9  |  -77.577 P= 9.0X10−9 Beta −1.9
 6. -79.583 P 4 9.0610 29 Beta 21.9  |  -78.462 P= 9.0×10−9 Beta +1.9
 7. -79.710 P 3 9.0610 29 Beta 21.9  |  -78.582 P= 9.0−10−9 Beta −1.9
 8. -79.794 P 2 9.0610 29 Beta 21.9  |  -78.659 P= 9.0×10 9 Beta −1.9
 9. -79.916 P = 9.0×10 −9 Beta 21.9  |  -78.723 P= 9.0−10−9 Beta −1.9

***Chosen best:
Nothing in common in the nbest lists...
Lang model score OCR top:  -23.6266403198 P = 9.0×10 −9 Beta −1.9
Lang model score PDFX top:  -25.0832328796 P = 9.0610 29 Beta 21.9
```

**Figure 5.8.** The screenshot shows two top N sorted list of the most probable n-grams with their log probabilities given by Viterbi inference. None of the pair of candidates in the lists match.

given by the Viterbi algorithm.

It can be seen that the candidate numbered 5 in the left list matches the candidate numbered 0 in the right list (for the highlighted characters), and this combination has the highest joint probability (although other combinations also match like 5, 1 and 5, 4). Thus we choose this candidate as the most probable NXML n-gram.

There can be a case where the two lists do not share a common candidate with respect to marked characters. In that case, we use a 6-gram language model trained on our corpus to obtain the probabilities for each of these twenty candidates, and select the most probable n-gram candidate. In case there is again a tie among all twenty, we take the top n-gram candidate from the PDFX candidate list as the most probable NXML n-gram. An example of this scenario is shown in Figure 5.8, where the two lists do not share a candidate in common and the language model chooses the best candidate in the OCR candidate list, as it scores the highest.

We normalize all probabilities to 1 and set the regularization parameter to $10^{-6}$. Also, we use StochHMM [15] library to calculate the N-best sequences by Viterbi

inference and the KenLM Language modeling toolkit [8] to build language models.

Next, we describe ambiguity dictionaries in more detail and explain how they are created from the corpus.

### 5.3.6 Post-processing

We often noticed a specific error of a unicode multiply symbol being transcribed as a "6" in PDFX and a unicode minus symbol as a "2" in PDFX leading to p-value mentions like: "p-value = 561028" which is incorrect, but makes sense on a syntactic and character level, and therefore is hard to correct. The correct corresponding mention would be: "p-value = $5 \times 10 - 2$". Hence in the post-processing step at the end of the pipeline, whenever we encounter the substring 610 and/ or 102, we consider replacing the 6 by a unicode multiply and the 2 by a unicode minus, and see if the language model gives us a higher probability for the modified n-gram.

**Compiling ambiguity dictionaries**

As explained in Section 5.3 on page 33, we create a dictionary like data structure, one for each PDFX and OCR, which we call an ambiguity dictionary from the training corpus and record the true character, the observed character and the frequency with which this mismatch has been observed. In the discussion that follows, we explain with examples how an OCR ambiguity dictionary is constructed, but the process is exactly the same for PDFX ambiguity dictionary. The dictionary consists of tuples of the following form:

FORM 1: (observed character, true character, frequency)

But another way to make the dictionary is by having tuples of the following form:

FORM 2: (true character, observed character, frequency)

It is important to note that these two tuples are not the same, and signify different things.

For example, consider the tuple of FORM 1:(‘|’ , ‘1’, 5) and of FORM 2: (‘1’ , ‘|’ , 38). Second tuple means that OCR confused ‘1’ with a ‘|’ 38 times, but the first tuple means that OCR confused ‘|’ with ‘1’ 5 times. These two ways don’t give symmetrical results. We make ambiguity dictionaries of FORM 1, but ambiguity dictionary of FORM 2 for OCR are interesting to observe as they show how OCR confuses similar looking characters and strings. A snippet of such a dictionary (with FORM 2 tuples) is shown in Figure 5.9. The first character in the line is the actual character, followed by pairs of observed characters and the frequency of observation.

The algorithm to make an ambiguity dictionary is outlined below:

1. Make a non-overlapping n-gram set from NXML text and an overlapping n-gram set from OCR text

2. For each n-gram in NXML text, find the closest (by Levenshtein distance) n-gram from OCR text

3. Globally align the characters in these two n-grams with the same scoring scheme as explained in Section 5.3 on page 31.

4. Iterate over each differing character and if neither of the characters is a whitespace or a gap, record the ambiguity pair and add one to the frequency.

Figure 5.10 shows an example where two n-grams are aligned and we want to record ambiguities from them. The characters highlighted in red are the positions for which ambiguities are recorded. Note that many ambiguities recorded this way are a result of incorrect alignment and not actually ambiguities by the OCR model. This is why we make the dictionary over a sufficiently large set of texts. The assumption is that ambiguities recorded due to incorrect alignments are random, and hence will not appear repeatedly.

↑ T 73, 1 5, ) 2

† † 66, T 64, 1 34, + 14, f 7, * 6

▹ D 3

ν v 31, V 8

∼ ~ 1230, N 41

ë e 8

Ŷ Y 4

∇ V 3

∗ * 45

∧ A 19, /\ 5

ı l 2

° 0 554, O 77, C 23, o 23

β B 2396, [3 584, fi 215, fl 152, β 104, 8 82, F) 81, B 66, 8 37, ?) 31, [5 29, oc 18, |3 17

**Figure 5.9.** A snippet from the OCR ambiguity dictionary with tuples of FORM 2

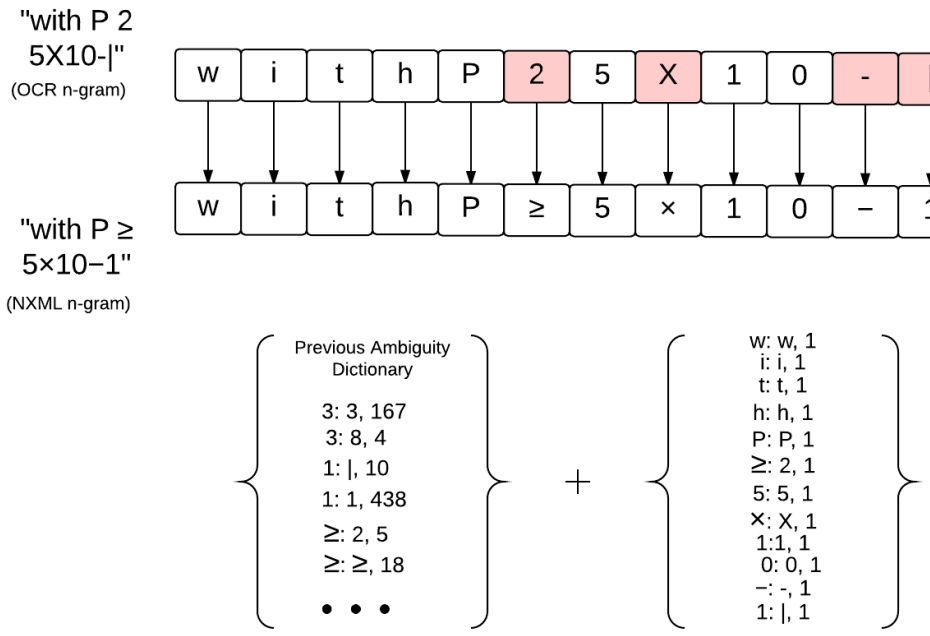


**Figure 5.10.** A schematic diagram showing how the ambiguity dictionary would be expanded given a pair of aligned n-grams.

## 5.4 Jumbling Error Correction

First, we make a non-overlapping n-gram set of the PDFX XML text and an overlapping n-gram set for PDFBox text. Recall that PDFBox is a tool for extracting text from PDF files. The correction is quite similar to how jumbling error is calculated/ detected, with the only difference being that on detection of a jumbled n-gram, we replace it by the non-jumbled version from PDFBox. Then we string together the non-overlapping n-grams with a space to construct the jumbling corrected text.

This method attempts to find and correct jumbling errors in n-grams (we had n=5) as we have defined it, but jumbling in a more general sense manifests itself in much more complicated ways that is hard to define and computationally intensive to find using a naive exhaustive search. This was shown in the example in Figure 2.1. Such jumbling errors are most common in tables and captions for figures. There is no guarantee that jumbling errors occur only over 5 grams, or equivalently within 5 words. In the same paper there might be jumbling errors spanning up to arbitrarily large and varied n. A better approach might be to start with a much larger n and attempt to detect jumbling from there and progressively reduce n one by one, however this approach is computationally intensive. Even operating on 5-grams is computationally expensive and jumbling error correction (and scoring) takes a significant amount of time compared to other correction and scoring methods.

We noticed that a previous version of PDFX (v1.8) gave much more jumbling errors even in the text body than the current version (v1.9). It is possible that they corrected the error or started using a newer version of Utopia Documents API, which resolved the error by itself. Due to this sharp decrease in noticeable errors and the computationally intractable nature of a viable solution for the remaining errors, we believe jumbling error correction is not so important in our pipeline anymore. We have

included it here solely for the sake of completeness.

## 5.5 Word-Boundary Correction

Word boundary correction is done by the following method:

1. For each hyphenated word check if it is present in a English dictionary. We use the enchant library (py-enchant) for this (https://pypi.python.org/pypi/pyenchant/).

2. If the hyphenated word is not present in the English dictionary but the word without the hyphen is present in the dictionary, we remove the hyphen from the word.

If the word with or without hyphen is not present in the dictionary, we keep it unchanged. This would happen for many biomedical terms as the dictionary we use is an English dictionary. We attempted to use a language model (along with the English dictionary) to increase the range of corrections possible for biomedical terms, but it did not give us any performance improvement. We believe this is because word-boundary errors are comparatively infrequent compared to total number of words in an article, and because most words in an article are English words or numbers, most word-boundary errors happen in English words, which are already corrected by the English dictionary. Another possible reason can be that since our language model is trained only on a corpus of 512 papers, it does not have a large enough vocabulary of biomedical terms.

Note that dehyphenation in general is a bigger problem, but in our case we know that the hyphens are inserted from a single source: column boundaries. For a more detailed analysis of some empirical methods for dehyphenation and associated challenges, see [1].

The content in this chapter in part is currently being prepared for submission for publication of the material. The thesis author was the primary investigator and author of this material.

# Chapter 6

# Application Overview

## 6.1 Architecture

We have followed a pipe and filter architecture for the application, where each correction type is a filter and accepts an XML as input, giving a corrected XML as output. This architecture allows much flexibility in what corrections need to be run on a set of documents and in what order. The pipeline is user configurable as a result.

A simplified class diagram is shown in Figure 6.1. The pipeline class is inspired from the scikit-learn Pipeline class, where each stage in the pipeline transforms/ fits the data and there is a predictor in the last stage. In our case, each stage of the pipeline is a corrector which implements the "correct" method which is expected to take an XML tree as input and return another (corrected) XML tree as output. Due to the nature of the pipeline, the order of the correctors can be changed and other correctors can be added simply by extending the base "Corrector" abstract class and implementing the correct method. The pipeline class itself holds objects of the individual correctors in a list and the method runPipeline calls the correct method of each corrector in the list in order, feeding output of one corrector to the input of next.

The class TextCorrector gives a built-in implementation of the correct method by calling the correctText method on each text section of the XML. It iterates over the text

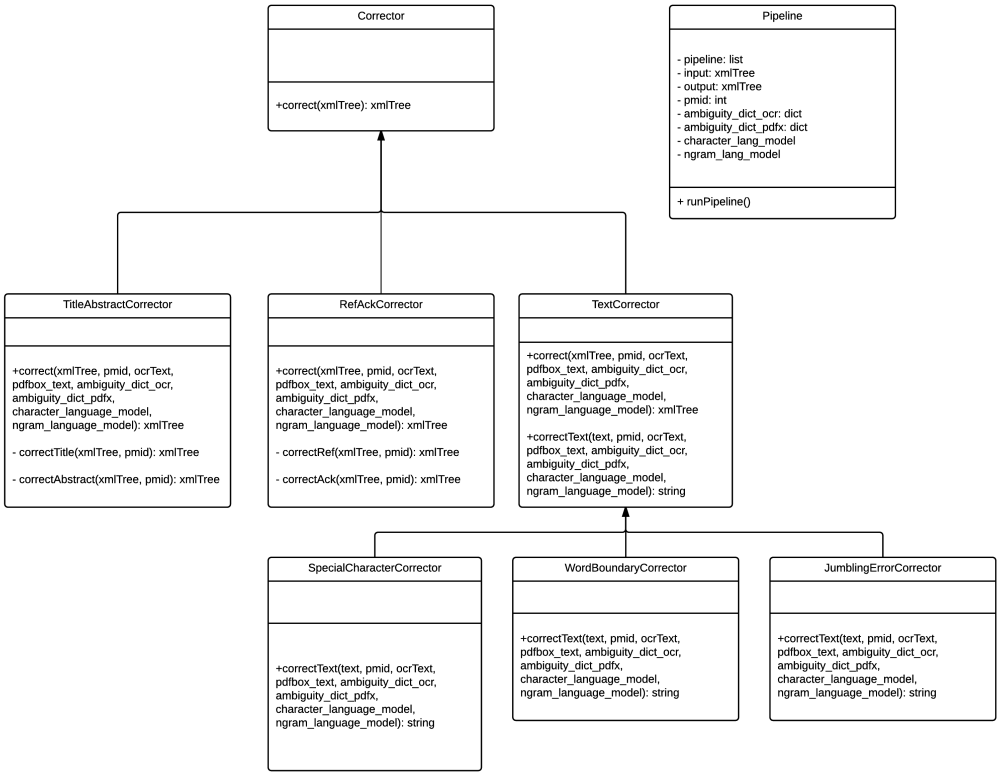


**Figure 6.1.** A simplified UML class diagram illustrating the architecture of the application.

content of each tag, calls the correctText method, gets the correct text for that tag and replaces it in the XML tree. All text based corrections, which do not change the XML structure of the document, inherit this class and implement the correctText method. Other corrections like title and abstract correction and reference/ acknowledgment correction directly inherit from the Corrector abstract class since they change the structure of the XML.

## 6.2 Interface

There are two modes in which the application can run: single file mode and batch mode. Single file mode is for when the application is to be run for a single PDF file. It automatically retrieves the PDFX XML from the web service hosted at http://pdfx.cs.man.ac.uk/, runs OCR and PDFBox and then performs correction. Batch mode is when many files need to be transcribed to XML. It assumes that the filenames are always their PubMed IDs since there is no easy way to determine the PubMed ID given a random PDF file. We opted for this convention instead of requiring the user to also provide a mapping file which maps file name to PubMed ID, since this is more convenient. In addition to this, for the batch mode, we allow the user to point to directories containing OCR texts and PDFX XMLs if already available.

## 6.3 Dependencies

This section lists all the dependencies needed to run the system.

1. Tesseract OCR Engine v3.02
2. Biopython v1.65 (for n-gram alignment)
3. edit_distance v0.2 (implementation of Levenshtein distance)
4. marisa-trie v0.7.2
5. pyenchant v1.6.6 (English Dictionary)
6. scikit-learn v0.16.1
7. KenLM Language Model Toolkit
8. PDFBox v1.8.5

9. Ghostscript v9.10
10. ImageMagick v6.7.7
11. StochHMM v0.34

## 6.4 Release

The software package will be released under the name “bioPDFX”, and is planned to be under an open source license.

# Chapter 7

# Performance Evaluation

We benchmarked our correction methods on a validation set of 100 articles in the Catalog of GWAS for which NXMLs were present, and we obtain the following results according to the metrics discussed in Chapter 1.3. Table 1 shows the scores if each correction is run on its own, independently of other corrections, while Table 2 shows the scores at the end of the pipeline, where the corrections are run in order as a pipeline as shown in Figure 5.1. As noted previously, Title, Abstract, Reference/ Acknowledgment and overall text quality scores are macro F scores, that is averaged F scores of individual papers. On the other hand, special character, p-value, number and word-boundary scores are micro F scores, where True Positives, False Positives and False Negatives are counted across all input articles. This is because there can be a variable number of special characters, p-values, numbers and hyphenations per article. It is important to note that no particular correction is aimed at p-value, number and overall text quality metrics. P-value and number extraction constitute two of the text mining tasks we need to do and the change in performance indicates the effect of our corrections. Similarly, overall text quality just measures how the general quality of text present in the XML has changed.

As can be seen in Figure 7.1, title scores were quite high to begin but were still improved considerably, while abstract scores were improved substantially. This is to

**Table 7.1.** Stage wise quality score comparison of raw and corrected XMLs. Jumbling score is the average number of jumbling errors detected per paper (lower is better).

| | Raw XML Score | Corrected XML Score | Unit |
|---|---|---|---|
| Title | 91.35 | 97.63 | Macro F-score |
| Abstract | 54.28 | 89.20 | Macro F-score |
| Reference/ Acknowledgment | 64.96 | 80.26 | Macro F-score |
| Special Character | 70.71 | 88.60 | micro F-score |
| Jumbling | 2.89 | 1.92 | Average count |
| Word-Boundary | 76.19 | 80.53 | micro F-score |

**Table 7.2.** Final quality score comparison of raw and corrected XMLs at the end of the correction pipeline. Jumbling score is the average number of jumbling errors detected per paper (lower is better).

| | Raw XML Score | Corrected XML Score | Unit |
|---|---|---|---|
| Title | 91.35 | 96.95 | Macro F-score |
| Abstract | 54.28 | 88.77 | Macro F-score |
| Reference/ Acknowledgment | 64.96 | 80.26 | Macro F-score |
| Special Character | 70.71 | 89.32 | micro F-score |
| Jumbling | 2.89 | 1.95 | Average count |
| Word-Boundary | 76.19 | 81.89 | micro F-score |
| P-value | 54.65 | 66.15 | micro F-score |
| Number | 87.33 | 88.53 | micro F-score |
| Overall Text quality | 90.06 | 91.07 | Macro F-score |

be expected as we retrieve titles and abstracts from the Entrez e-utilities API directly, which can be thought of as a gold standard itself. The F-scores are still not 100 due to syntactic differences between title/ abstract text present in NXMLs and the title/ abstract text retrieved from e-utilities API. For example, the API does not seem to use unicode characters like "$\leq$", "$\geq$" and "$\sim$" and would instead have the phrases "less than or equal", "greater than or equal" and "approximately" in their place. Furthermore, as shown in Figure 7.1, for PubMed ID: 23974872 [23], the title from NXMLs is "Genome-wide association analysis identifies 14 new risk loci for schizophrenia", while that retrieved from the e-utilities API is "Genome-wide association analysis identifies 13 new risk loci for schizophrenia.", which matches the PDF title too. Similarly, the NXML for PubMed ID: 23770605 [3] does not contain an abstract even though the paper as well as the e-utilities have it. Due to such and other inconsistencies, for example in punctuation, the score is not close to 100, otherwise we can consider the titles and abstracts to be near perfect in the corrected XMLs (assuming e-utilities API gives correct results). Some errors are introduced due to accumulation of errors from previous stages of corrections in the pipeline, but as can be seen by comparing stage-wise and end of pipeline scores in Tables 1 and 2, they are quite small and can be neglected.

Reference/ Acknowledgment scores also improved considerably and lead to better scores for the remaining stages of the pipeline, by removing a significant number of the False Positives, as can be observed by comparing the stage-wise results in Table 1 and final results in Table 2.

We were able to improve special character F-score by a considerable margin through our technique of combining HMMs, gene sequence alignment, OCR and language models, though we also did some post-processing to improve these scores as explained in Section 5.3 on page 39.

Jumbling errors as we detected them did decrease but we noticed that most of

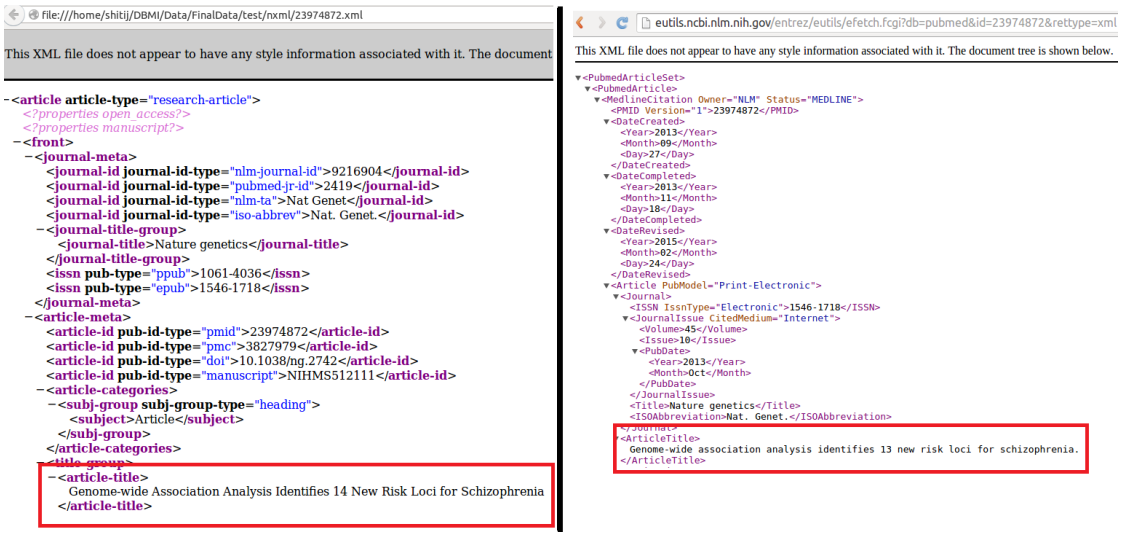


**Figure 7.1.** A screenshot showing inconsistency in the title in NXML and E-utilities API for PubMed ID: 23974872 [23].

these errors were in tables and their column headings, which are not that important to us. Apart from this, jumbling error correction takes significant computation power as we have to check if each overlapping n-gram is a permutation of another n-gram, but the permutation is not present in the text itself. As discussed in Section 5.4 this is still only an approximation in detecting jumbling errors as we set a fixed n=5. Word Boundary scores are also improved modestly, as expected.

P-value scores are improved considerably because of special character correction and title/ abstract correction. Number score improved only slightly because there is a large amount ( average of about 100 per paper) of numbers already present in a paper that are correct to begin with. On the other hand there are about 20 P-values present in an article on average. The overall text quality score, which is just a measure of the overall quality of text, improved slightly as well.

The content in this chapter in part is currently being prepared for submission for publication of the material. The thesis author was the primary investigator and author of this material.

# Chapter 8

# Conclusion

We have developed a tool to post-process PDFX XMLs specifically for biomedical articles (articles in the catalog of GWAS, for our purposes) while expanding its scope to other publication types commonly encountered in text mining like manuscripts. We have shown that the post-processing techniques that we have used, improve the quality of XMLs by the implicit metrics that we have defined and also make them much better suited for two text mining tasks we needed to do: P-value and number extraction, while slightly improving the overall text quality as well.

Although we have trained the tool to be specifically optimized for our target dataset (GWAS), we believe that these techniques can be applied to other kinds of datasets as well. We validated our assumption that for papers in GWAS, authors use a specific vocabulary and exhibit a limited way of using special characters, which can be exploited to correct special character errors using HMMs and language models. We believe that this assumption will hold true in other domains as well, and to a lesser extent across domains. It is important to note that although we use PDFX XMLs as the target, and OCR text as an additional text source, the approach that we present is general and is not dependent on these text sources. Given a text source to correct substitution errors in, we can use other text sources which also make substitution errors different and independent from our target. Also using a gold standard text source for training, we

can apply the same approach of statistically learning error patterns of the different text sources, and then using a committee of HMMs along with a token level language model to correct errors n-gram by n-gram in the target text.

There are some limitations of the approach, and we summarize them below:

1. Title and abstract corrections depend on the e-utilities API
2. As we model n-grams as sequences of characters and due to the Markov assumption, we cannot correct errors in which one character is replaced by multiple, or is deleted.
3. Jumbling error correction is only an approximation since we fix n=5

The content in this chapter in part is currently being prepared for submission for publication of the material. The thesis author was the primary investigator and author of this material.

# Bibliography


[1] Ola sverre Bauge. Dehyphenation: Some empirical methods. Masters Thesis, 2012.

[2] Øyvind Raddum Berg. High precision text extraction from PDF documents. Masters Thesis, 2011.

[3] Sonja I Berndt, Christine F Skibola, Vijai Joseph, Nicola J Camp, Alexandra Nieters, Zhaoming Wang, Wendy Cozen, Alain Monnereau, Sophia S Wang, Rachel S Kelly, et al. Genome-wide association study identifies multiple risk loci for chronic lymphocytic leukemia. *Nature genetics*, 45(8):868–876, 2013.

[4] Peter JA Cock, Tiago Antao, Jeffrey T Chang, Brad A Chapman, Cymon J Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, et al. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423, 2009.

[5] Alexandru Constantin, Steve Pettifer, and Andrei Voronkov. PDFX: fully-automated PDF-to-XML conversion of scientific literature. In *Proceedings of the 2013 ACM symposium on Document engineering*, pages 177–180. ACM, 2013.

[6] Eli Cortez, Altigran S da Silva, Marcos André Gonçalves, Filipe Mesquita, and Edleno S de Moura. FLUX-CIM: flexible unsupervised extraction of citation metadata. In *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, pages 215–224. ACM, 2007.

[7] Marc Damashek et al. Gauging similarity with n-grams: Language-independent categorization of text. *Science*, 267(5199):843–848, 1995.

[8] Kenneth Heafield. KenLM: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197. Association for Computational Linguistics, 2011.

[9] Chun-Nan Hsu, Yu-Ming Chang, Cheng-Ju Kuo, Yu-Shi Lin, Han-Shen Huang, and I-Fang Chung. Integrating high dimensional bi-directional parsing models for gene mention tagging. *Bioinformatics*, 24(13):i286–i294, 2008.

[10] Xuedong D Huang, Yasuo Ariki, and Mervyn A Jack. *Hidden Markov Models for speech recognition*, volume 2004. Edinburgh university press Edinburgh, 1990.

[11] Duff Johnson. Each PDF Page Is a Painting. http://talkingpdf.org/each-pdf-page-is-a-painting/, 2010. [Online; posted 8 November, 2010].

[12] Vlado Kešelj, Fuchun Peng, Nick Cercone, and Calvin Thomas. N-gram-based author profiles for authorship attribution. In *Proceedings of the Conference Pacific Association for Computational Linguistics, PACLING*, volume 3, pages 255–264, 2003.

[13] Robert J Klein, Caroline Zeiss, Emily Y Chew, Jen-Yue Tsai, Richard S Sackler, Chad Haynes, Alice K Henning, John Paul SanGiovanni, Shrikant M Mane, Susan T Mayne, et al. Complement factor h polymorphism in age-related macular degeneration. *Science*, 308(5720):385–389, 2005.

[14] Yao-Zhong Liu, Yu-Fang Pei, Yan-Fang Guo, Liang Wang, Xiao-Gang Liu, Han Yan, Dong-Hai Xiong, Yin-Ping Zhang, S Levy, J Li, et al. Genome-wide association analyses suggested a novel mechanism for smoking behavior regulated by IL15. *Molecular psychiatry*, 14(7):668–680, 2009.

[15] Paul C Lott and Ian Korf. StochHMM: a flexible hidden markov model tool and C++ library. *Bioinformatics*, 30(11):1625–1626, 2014.

[16] Caroline Lyon, Ruth Barrett, and James Malcolm. A theoretical basis to the automated detection of copying between texts, and its practical implementation in the ferret plagiarism and collusion detector. *Plagiarism: Prevention, Practice and Policies*, 2004.

[17] Christopher D Manning and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[18] Rajan P Nair, Kristina Callis Duffin, Cynthia Helms, Jun Ding, Philip E Stuart, David Goldgar, Johann E Gudjonsson, Yun Li, Trilokraj Tejasvi, Bing-Jian Feng, et al. Genome-wide scan reveals association of psoriasis with IL-23 and NF-$\kappa$B pathways. *Nature genetics*, 41(2):199–204, 2009.

[19] NIH. PubMed. http://www.ncbi.nlm.nih.gov/pubmed, 2015.

[20] NIH. PubMed Central. http://www.ncbi.nlm.nih.gov/pmc/, 2015.

[21] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[22] Réjean Plamondon and Sargur N Srihari. Online and off-line handwriting recognition: a comprehensive survey. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(1):63–84, 2000.

[23] Stephan Ripke, Colm O'Dushlaine, Kimberly Chambert, Jennifer L Moran, Anna K Kähler, Susanne Akterin, Sarah E Bergen, Ann L Collins, James J Crowley, Menachem Fromer, et al. Genome-wide association analysis identifies 13 new risk loci for schizophrenia. *Nature genetics*, 45(10):1150–1159, 2013.

[24] Efstathios Stamatatos. Intrinsic plagiarism detection using character n-gram profiles. *Threshold*, 2:1–500, 2009.

[25] Dominika Tkaczyk, Pawel Szostek, Piotr Jan Dendek, Mateusz Fedoryszak, and Lukasz Bolikowski. CERMINE–automatic extraction of metadata and references from scientific literature. In *Document Analysis Systems (DAS), 2014 11th IAPR International Workshop on*, pages 217–221. IEEE, 2014.

[26] 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, 1996.

[27] Prasanna Velagapudi. Using HMMs to boost accuracy in optical character recognition. In *Proceedings of SPIE, 27th AIPR Workshop: Advances in Computer-Assisted Recognition*, volume 3584, pages 96–104. Citeseer, 1999.

[28] Li Zhuang and Xiaoyan Zhu. An OCR post-processing approach based on multi-knowledge. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 346–352. Springer, 2005.