

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/316844094>

Automatic Text Correction of PDF Extracted Text

Technical Report · December 2016

CITATIONS

0

READS

8

1 author:



[Joshua Allan Mathias](#)

Brigham Young University - Provo Main Campus

4 PUBLICATIONS 0 CITATIONS

SEE PROFILE

PDF Text Processing

Joshua Mathias

December, 2016

Table of Contents

| | |
|---------------------------|-----------|
| 1 Introduction | 2 |
| 1.1 Purpose | 2 |
| 1.2 Scope | 3 |
| 2 Problems | 3 |
| 2.1 Introduction | 3 |
| 2.2 Problems | 3 |
| 3 Previous Work | 6 |
| 3.1 Introduction | 6 |
| 3.2 Previous Work | 6 |
| 4 Methods | 8 |
| 4.1 Introduction | 8 |
| 4.2 Methods | 9 |
| 5 Results | 13 |
| 5.1 Introduction | 13 |
| 5.2 Combined Words | 13 |
| 5.3 Empty Lines and Space | 15 |
| 5.4 Unwanted Characters | 16 |
| 5.5 Incorrect Casing | 19 |
| 6 Conclusions | 22 |
| 6.1 Introduction | 22 |
| 6.3 Conclusion | 22 |
| 6.4 Future Work | 22 |
| 7 Appendix | 23 |

1 Introduction

1.1 Purpose

This document describes methods and research for correcting text extracted from PDF files. The purpose of this project was to correct text extracted from PDF files (33285 of which were provided for this project) of The Church of Jesus Christ of Latter-Day Saints, so that they could use the text for machine learning training

purposes (such as for machine translation). Consequently, the project, or final formatting of the files, is intended to comply to the LDS Church's standards and requests.

1.2 Scope

The problems that need to be solved, proposed solutions, concepts behind the implemented solutions, and the results will be covered in this document. There will also be some relatively small examples from files included. The code used for the project is available at <https://github.com/JoshuaMathias/text-correction>.

2 Problems

2.1 Introduction

Here we lay out the problems in the text extracted from PDF files that need to be fixed, as well as problems to solve in order to perform the corrections and evaluate the solutions.

2.2 Problems

Combined words

Words missing a space between them.

Problem:

otherway

Correct:

other way

Empty lines and space

Blank lines or lines with only whitespace. Also any spacing other than a standard space or new line.

Problem:

People who know a lot of otherpeople.

Remember that to achieve quicker, better results, you should:

Correct:

People who know a lot of other people.

Remember that to achieve quicker, better results, you should:

Combined lines

Where one or more words from a different line or sentence are on the same line and consequently combined incorrectly with another sentence.

Problem (test_1.txt):

Dictionary heb An alternate translation from the Hebrew gr An alternate translation from the Greek ie An explanation of idioms and difficult wording

Correct:

Dictionary

heb An alternate translation from the Hebrew

gr An alternate translation from the Greek

ie An explanation of idioms and difficult wording

Split lines

Where one or more words from one line (sentence) are separated and on a different line.

Problem (test_3.txt):

B
e it known unto all nations

Correct:

Be it known unto all nations

Mixed words

Where one or more words are at the incorrect location in the sentence or line.

Problem (test_1.txt):

Isa. Jer. Isaiah Jeremiah 1 Jn. 2 Jn. 1 John 2 John Pearl of Great Price

Correct:

Isa. Isaiah Jer. Jeremiah 1 Jn. 1 John 2 Jn. 2 John Pearl of Great Price

Misspelled words/missing letters

Misspelled words or words missing letters.

Problem (test_5.txt):

Tst

Correct:

Test

Incorrect casing

For each word, one or more letters in the word are either incorrectly lowercase or incorrectly upper case.

Problem (test_4.txt):

The TeSTimOny Of eighT WiTneSSeS

Correct:

The Testimony of Eight Witnesses

Split words

A space in between letters of the same word (but on the same line; otherwise constitutes a split line).

Problem (test_5.txt):

thef ear

Correct:

the fear

Unwanted characters

Unrecognized characters, HTML, URLs, or other technical characters such as OffOff (from PDF check boxes), , • .

Problem:

Change to existing vendor
OffOffOff
fax the completed form to _____.

Correct:

Change to existing vendor
fax the completed form to .

3 Previous Work

3.1 Introduction

Here we describe previous methods or programs to solve the problems listed above, as well as if we used these methods as part of a solution or simply used them for comparison.

3.2 Previous Work

General Text Correction

The following link provides tools for some of the tasks in this project, such as removing extra whitespace and counting characters, but the interface allows for only one file at a time, and this project requires custom solutions to adapt to the Church's data and requirements.

Combined Words

The following StackOverflow conversations provide some answers:

[How can I split multiple joined words?](#)

Contains a possible implementation in Python using the Viterbi algorithm.

We don't use the Viterbi algorithm in this work, but it may be worth trying and comparing, at least if greater efficiency is needed.

[How to split a string into words.](#)

Identifies this problem as a segmentation problem, with a link to the following lecture from Duke, describing a possible solution, "maximum probability segmentation":

<http://www.cs.duke.edu/courses/fall01/cps130/lectures/lect21.pdf>

The lecture provides a possible implementation, and it is intuitively similar (using word probabilities) to what we do in this work, but it may be worth looking into its differences in implementation and calculating probabilities for the most likely segmentation.

Empty Lines

StackOverflow solution:

[Remove all whitespaces from String but keep ONE newline](#)

From this solution I obtained a Regular expression that I used in my implementation: `("^\\s+|\\s+$|\\s*(\\n)\\s*(\\s)\\s*", "$1$2")` (See explanation at the link.)

However, this could only be used after standardizing space characters and new lines, because the escape sequence “\s” doesn’t include all whitespace characters.

Combined lines

StackOverflow solution:

[How to filter word permutations to only find semantically correct ngrams? \(Python 3, NLTK\)](#)

This discusses systematically generating semantically correct phrases using n-grams, and it also lists resources for n-grams.

[Google N-grams](#) and [Google N-Gram Downloader](#) (hard to use in a downloaded form because of it’s size)

[Microsoft Web Language API](#)

www.ngrams.info

www.wordfrequency.info

Because of the limitations of the current language models provided by the Church (it includes many incorrect words), these or other similar language model resources may be worth looking into.

Split lines

Mixed words

Misspelled words/missing letters

SoftCorporation has an open source spell checker in Java:

<http://www.softcorporation.com/products/spellcheck/>

They also provide dictionaries in 9 languages:

<http://www.softcorporation.com/products/spellcheck/dictionaries/>

Spell checking is a common problem that can be solved by using edit distance and Soundex (phoneme representations) and by ranking suggestions by word frequencies.

Incorrect casing

Oracle has a proper case processor that ensure that the first word of each sentence starts uppercase:

http://www.oracle.com/webfolder/technetwork/data-quality/edqhelp/Content/processor_library/transformation/proper_case.htm

Since much of text isn’t properly separated into sentences, and much of the casing is needed for words not at the beginning of sentences or lines, this solution isn’t sufficient.

StackOverflow solution:

[Convert String into Title Case](#)

This discusses how to convert a string of text to title case. However, our main difficulty is determining when words should be capitalized and when not, or which letters in the word (in the case of names) should be capitalized.

Split words

Unwanted characters

StackOverflow solutions:

[Regular expression to remove HTML tags](#)

From this we use the following regular expression to remove HTML and XML code: `<[>]*>`

[Remove Email address from java string](#)

From this we use the following regular expression to remove emails:

`([^\s@]+)(\.[^\s@]+)*@[^\s@]+\.[^\s@]+`

[Removing the url from text using java](#)

From this we use the following regular expression to remove URLs:

`((https?|ftp|gopher|telnet|file|Unsure|http):((/)|(\.\.)))+[\w\.\d:#@%/\$()~_?\\+ -= \\\.\&]*`

There are common solutions for removing HTML, XML, emails, and urls. There are also [character classes in Unicode](#), and we can more safely remove any characters that aren't in specific character classes, such as the `\p{L}` character class for letters in any language.

4 Methods

4.1 Introduction

This section briefly describes the overall architecture of the code and additional details concerning how each of the solutions were implemented.

4.2 Methods

Splitting Words

An A* approach is used, as this allows simple flexibility in the ordering of possible splits while also being efficient as a dynamic programming approach. The implementation goes from the beginning of the string to the end and orders the possibilities by the number of words in the splitting option, and where this is the same, the solutions are ordered descending by likelihoods from the language model. The “currentIndex” addition to the score is added for initial efficiency in ordering solutions. Casing is ignored (the likelihood from the most likely casing in the language model is used). Words are only split if the splitting option includes all letters of the original combined word.

$$\text{score} = \text{currentIndex} - 100 * \text{numWords} + 10 * \text{lmScore};$$

currentIndex: length of the proposed solution

numWords: number of words in the proposed solution

lmScore: average likelihood (from the language model) of each word in the proposed solution

Examples:

Combined word: toshow

| Proposed solutions | tos how | to show |
|--------------------|-------------|-------------|
| currentIndex | 6 | 6 |
| numWords | 2 | 2 |
| lmScore | -9.4758052 | -5.9160817 |
| score | -288.758052 | -253.160817 |

Chosen solution: to show

Combined word: taxwitholding

| Proposed solution | tax withholding | to show |
|-------------------|-----------------|-------------|
| currentIndex | 6 | 6 |
| numWords | 2 | 2 |
| lmScore | -9.4758052 | -5.9160817 |
| score | -288.758052 | -253.160817 |

Chosen solution: tax withholding

A more detailed list of splitting examples, with all possible solutions at the time of returning the solution (ordered by “Calculated score”) and the chosen solution, is found at this link: [Splitting Examples](#)

Empty lines and space

I standardized space characters into a single space from Unicode’s whitespace characters using this list:

<https://www.cs.tut.fi/~jkorpela/chars/spaces.html>

Regular expression for all space characters:

```
[\\t\\f\\x0B\\u0020\\u00A0\\u1680\\u180E\\u2000\\u2001\\u2002\\u2003\\u2004\\u2005\\u2006\\u2007\\u2008\\u2009\\u200A\\u200B\\u202F\\u205F\\u3000\\uFEFF]
```

Example:

This  example 

Correct:

This  example

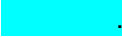

And the following list of characters for new lines:

<http://unicode.org/standard/reports/tr13/tr13-5.html>



Regular expression for all new line characters:

```
[\\u000A\\u000D\\u0085\\u000B\\u000C\\u2028\\u2029]
```

Example:

This


example

Correct:

This


example

(Ignore the periods. It isn't possible to see, but the new line was standardized.)

After standardizing whitespace, all whitespace between words and at the beginning and end of lines and files were replaced with a single space or new line with the following function call: `replaceAll("^\\s+|\\s+$|\\s*(\\n)\\s*|(\\s)\\s*", "$1$2");` ([See this link for an explanation](#))

Example:

This

example

Correct:

This
example

Unwanted characters

I used unicode classes as described at

<http://www.regular-expressions.info/unicode.html>. \{L} includes letters from all supported languages, which are the characters that we most want to keep. I made counts of symbols in all files to determine which characters are most common, which means they may be useful to keep for machine learning training.

The following regular expressions were used to remove unwanted code or formatting:

HTML and XML:

<[>]*\n+[>]*> (any number of lines)

Examples:

<|>

Email addresses:

([^\s@]+)(\.[^\s@]+)*@[^\s@]+\.[^\s@]+

Examples:

cor-intellectualproperty@LDSchurch.org

orderseu@ldschurch.orgMagasin

Note that it includes all characters until a space is found.

URLs:

((https?|ftp|gopher|telnet|file|Unsure|http):((/|)(\\|)))+[\\w\\d:#@%/;\$()~_?\\+.=|\\\\.&]*
)

Examples:

http://www.youtube.com/mormontabchoir.

http://LDS.org/study/

Repeated Off:

Text extracted from PDF files often contain the word Off for each checkbox (note that this only removes the "Off" when there is more than one in succession):

Off(Off)+

Examples:

OffOffOffOffOff

OffOff

Fill in the blanks: _____

_()+

Examples:

Incorrect Casing

The proposed solution to fix incorrect casing is to change words in all uppercase to the most likely casing from the language model (which could be upper case) and this could be done with more confidence for those words that only are found as upper case (like the abbreviation USPS).

One area of concern is handling names that have a capital letter in the middle of the name, as well as languages that have casing in the middle of words (such as Swahili). With this concern in mind, and to avoid changing words that already have correct casing, words must comply with the following condition to be changed:

1. The word is in the language model and has an uppercase letter in the middle of the word. Examples: boOk, MCConkie

A possible second option could be considered for words that don't appear in the language model:

2. The word has an uppercase letter in the middle of the word, and the first letter of the word is not uppercase (even if the word is not in the language model). Example: boOk

However, the second condition may not work for all languages, and to be safe it may be best to not change the casing of unknown words.

Because of the variable nature of the text, changing the first letter of the word to uppercase at the beginning of each sentence may do more harm than good, because it is hard to define where a new sentence starts. However, this is a possibility.

5 Results

5.1 Introduction

This section lists the results for each of the implemented solutions, with commentary.

5.2 Combined Words

The following table shows the results of our implementation of splitting combined words ([The results can also be found here](#)).

Correcting Combined Words

| | Method of splitting | | | | | |
|--|---------------------|----------------------------------|-----------------------------------|------------------------------------|-------------------------------------|--------------------------------------|
| | Manual correction | 1-grams Clean by Likelihood -5.0 | 1-grams Clean by Likelihood - All | 1-grams Ordered by Likelihood -5.0 | 1-grams Ordered by Likelihoods -6.5 | 1-grams Ordered by Likelihoods - All |
| Errors corrected | 121 | 102 | 73 | 103 | 101 | 86 |
| Errors split incorrectly | 0 | 3 | 16 | 0 | 1 | 1 |
| Correct words split | 0 | 51 | 2 | 40 | 12 | 0 |
| Errors remaining | 0 | 19 | 48 | 18 | 20 | 35 |
| Percentage Corrected | 1 | 84.30 | 60.33 | 85.12 | 83.47 | 71.07 |
| Percentage Corrected Adjusted for Errors | 1 | 42.15 | 58.68 | 52.07 | 73.55 | 71.07 |

Table 1. The effectiveness of different methods of correcting combined words,

Errors corrected: The number of combined words correctly separated by spaces.

Example: CowderyDavid -> Cowdery David

Errors split incorrectly: The number of combined words that were split by spaces, but incorrectly. Example: andThummim -> an dT hu mm im

Correct words split: The number of correct words that were split when they shouldn't have been. Example: clippings -> clip pi ngs

Errors remaining: The number of combined words that weren't separated correctly, calculated as the number of errors corrected manually subtracted by the errors corrected for the splitting method. Example: CowderyDavid -> CowderyDavid

Percentage corrected: Errors corrected / the total number of combined words to be corrected (121)

Percentage Corrected Adjusted for Errors: (Errors corrected - correct words split) / the total number of combined words to be corrected (121)

Descriptions in the second row of the table:

Clean: This means using a clean dictionary taken from the 1-grams of the language model provided by the Church, without ordering by language model likelihoods (the first splitting option is chosen with a preference for splitting into fewer words. Example: "and Thummim" (2 words) would be chosen over "an dT hu mm im" (5 words)).

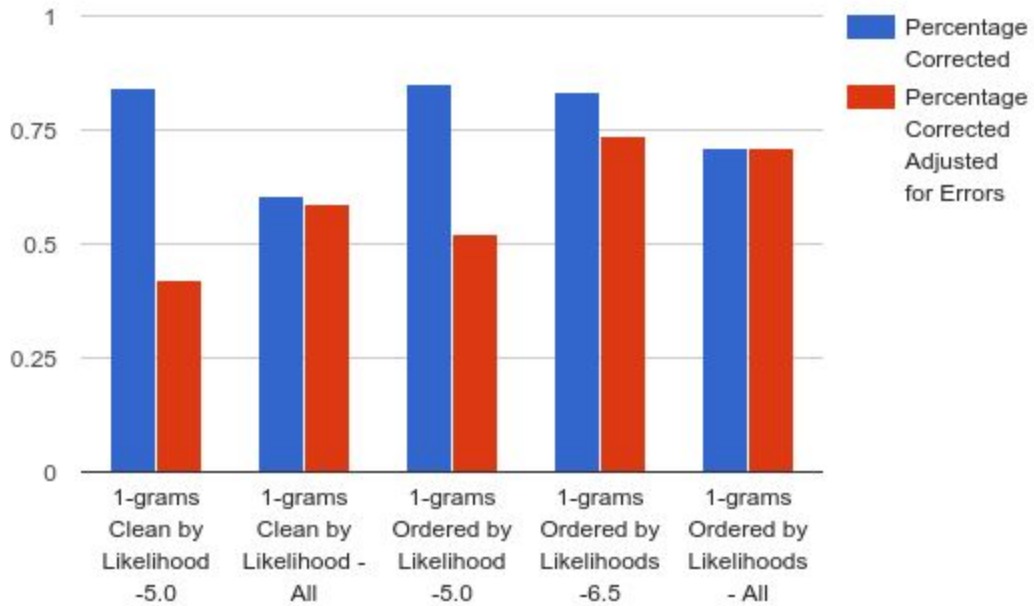
Ordered by likelihood: This means ordering the splitting options by the sum of the language model likelihoods for each word. The language model score is weighted an order of magnitude more than the preference for splitting into less words. If the split with the best language model score and secondly the lowest number of words is considered to be optimal, the algorithm functions as an A* search, with a priority queue ranked as explained, as well as ranking by the number of characters currently included from the combined word, to improve efficiency.

-5.0, -6.5, All: This number represents the cutoff used, where -5.0 means that only words with a likelihood greater than -5.0 in the language model were included in the dictionary for splitting words, and "All" means that all words from the language model were included (See the next paragraph for more information).

See the following file for examples of splitting combined words and their scores:
https://github.com/JoshuaMathias/text-correction/blob/master/splitting_examples.txt

See section 4.2 for an explanation on how the scores are calculated.

Percentage Corrected and Percentage Corrected Adjusted for Errors



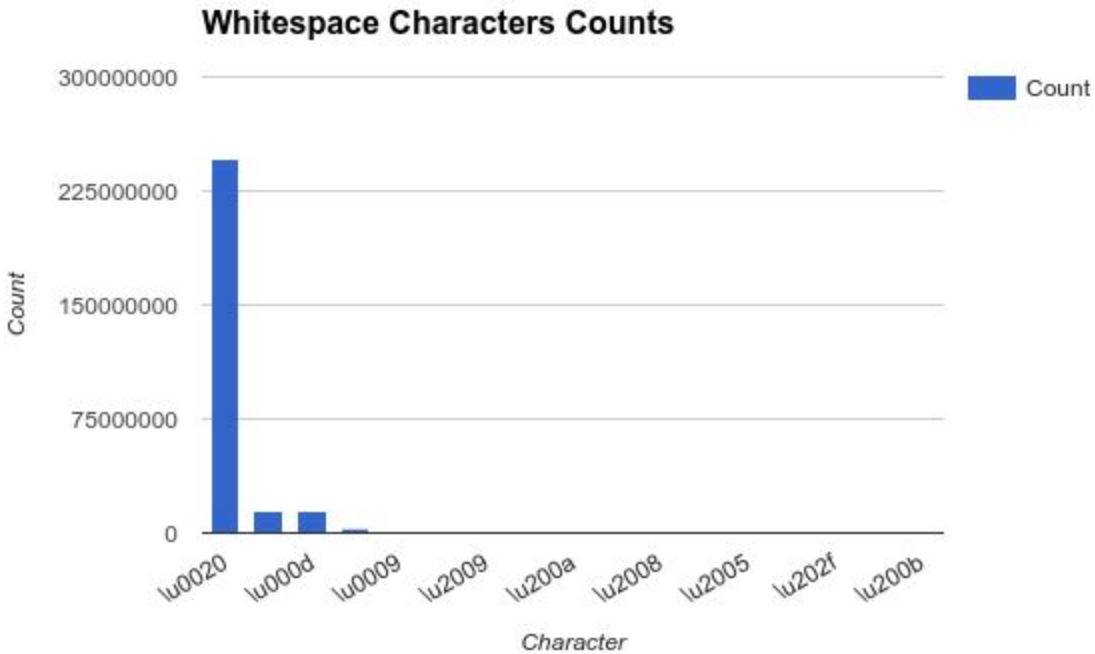
The results reflect some difficulties with the language model, as many combined words are included in the language model. A cutoff was used to not include words with a likelihood below a certain number, with the lowest in this case being -6.5227585, where -6.5227585 indicates that the word was only seen once. The best result in terms of “Percentage Corrected Adjusted for Errors” was “1-grams Ordered by Likelihoods -6.5,” which doesn’t include these one-time-only words. Including less words generally improves the number of errors corrected, because, otherwise, words that should be corrected are considered correct already. However, including more words is only effective by ranking with n-gram likelihoods, as this helps to distinguish good words from improbable or incorrect words.

5.3 Empty Lines and Space

The following regular expression includes the different Unicode new line and whitespace characters:

```
[\u000A\u000D\u0085\u000B\u000C\u2028\u2029\u2019\u201A\u201B\u201C\u201D\u201E\u201F\u2020\u2021\u2022\u2023\u2024\u2025\u2026\u2027\u2028\u2029\u202A\u202B\u202C\u202D\u202E\u202F\u2030\u2031\u2032\u2033\u2034\u2035\u2036\u2037\u2038\u2039\u203A\u203B\u203C\u203D\u203E\u203F\u2040\u2041\u2042\u2043\u2044\u2045\u2046\u2047\u2048\u2049\u204A\u204B\u204C\u204D\u204E\u204F\u2050\u2051\u2052\u2053\u2054\u2055\u2056\u2057\u2058\u2059\u205A\u205B\u205C\u205D\u205E\u205F\u2060\u2061\u2062\u2063\u2064\u2065\u2066\u2067\u2068\u2069\u206A\u206B\u206C\u206D\u206E\u206F\u2070\u2071\u2072\u2073\u2074\u2075\u2076\u2077\u2078\u2079\u207A\u207B\u207C\u207D\u207E\u207F\u2080\u2081\u2082\u2083\u2084\u2085\u2086\u2087\u2088\u2089\u208A\u208B\u208C\u208D\u208E\u208F\u2090\u2091\u2092\u2093\u2094\u2095\u2096\u2097\u2098\u2099\u209A\u209B\u209C\u209D\u209E\u209F\u20A0\u20A1\u20A2\u20A3\u20A4\u20A5\u20A6\u20A7\u20A8\u20A9\u20AA\u20AB\u20AC\u20AD\u20AE\u20AF\u20B0\u20B1\u20B2\u20B3\u20B4\u20B5\u20B6\u20B7\u20B8\u20B9\u20BA\u20BB\u20BC\u20BD\u20BE\u20BF\u20C0\u20C1\u20C2\u20C3\u20C4\u20C5\u20C6\u20C7\u20C8\u20C9\u20CA\u20CB\u20CC\u20CD\u20CE\u20CF\u20D0\u20D1\u20D2\u20D3\u20D4\u20D5\u20D6\u20D7\u20D8\u20D9\u20DA\u20DB\u20DC\u20DD\u20DE\u20DF\u20E0\u20E1\u20E2\u20E3\u20E4\u20E5\u20E6\u20E7\u20E8\u20E9\u20EA\u20EB\u20EC\u20ED\u20EE\u20EF\u20F0\u20F1\u20F2\u20F3\u20F4\u20F5\u20F6\u20F7\u20F8\u20F9\u20FA\u20FB\u20FC\u20FD\u20FE\u20FF]
```

Using this regular expression, in all 33285 documents the following whitespace characters were found (including new lines):



Data (with the actual whitespace character in parentheses):

\u0020 (): 245446697 - (normal space)
 \u000a (): 14523191 - (line feed)
 \u000d (): 14516283 - (carriage return)
 \u00a0 (): 2883183 - (non-breaking space)
 \u0009 (): 349421 - (tab)
 \u2002 (): 84451 - (en space)
 \u2009 (): 33433 - (thin space)
 \u2003 (): 26990 (em space)
 \u200a (): 7977 - (hair space)
 \u2007 (): 2082 - (figure space)
 \u2008 (): 1135 - (punctuation space)
 \u2004 (): 883 - (three-per-em space)
 \u2005 (): 574 - (four-per-em space)
 \u2006 (): 486 - (six-per-em space)
 \u202f (): 315 - (narrow no-break space)
 \ufe00 (): 4 - (zero width no-break space)
 \u200b (): 2 - (zero width space)

There were 3440 files that have only white space. 410 of those files are completely empty. See [blank_files.txt](#) for full results.

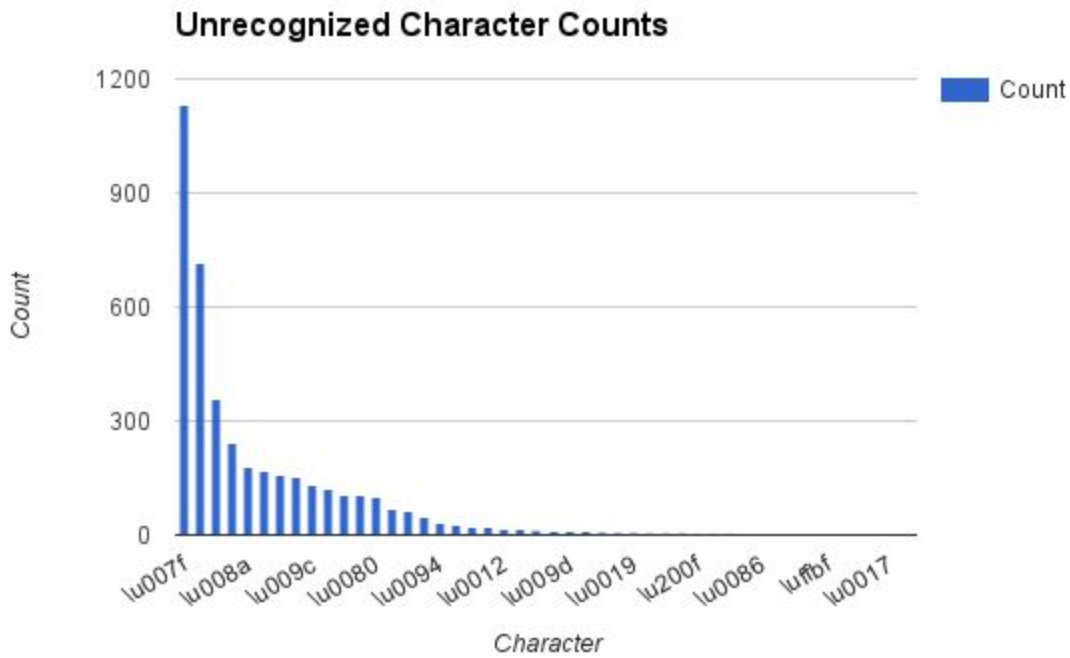
There are also 520 files with 20 or less non-whitespace characters. Full results are at [small_files.txt](#). I noticed that many files have a single double digit number in them (possibly a page number or some other standard numbering), which explains the number of files with only two characters. These files with few or no characters likely don't contain enough text to be helpful for machine learning training purposes.

5.4 Unwanted Characters

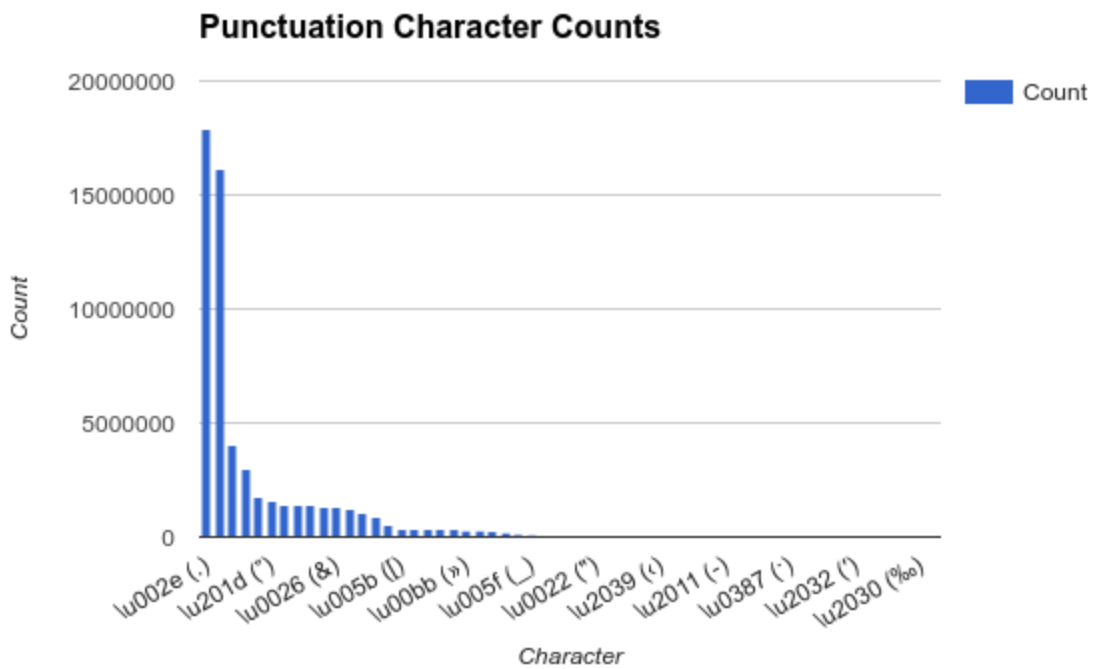
To know which characters to remove, we count the occurrence of different characters or character classes in all files. These counts are found at the following link, with individual files described and linked to below, listing the regular expression used to create the counts.

<https://github.com/JoshuaMathias/text-correction/tree/master/characters>

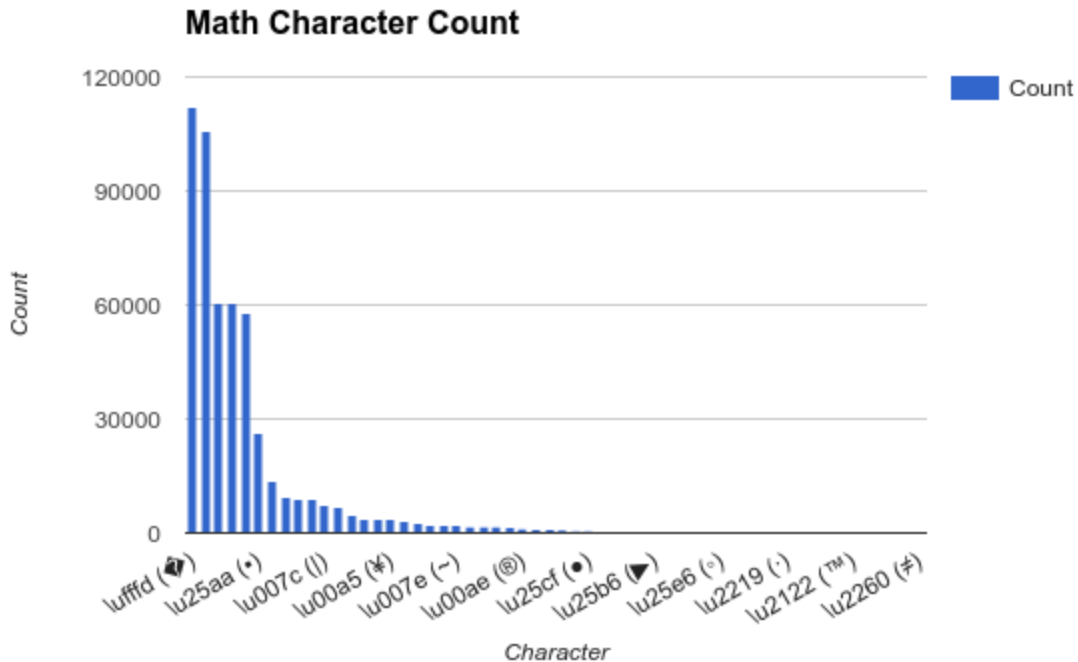
- [Math characters](#): `[\p{S}]`
- [Punctuation characters](#): `[\p{P}]`
- [Special characters](#) (not in the punctuation, letter, mark, or number classes): `[\n\p{P}\p{L}\p{M}\p{N}]`
- [Unrecognized characters](#) (characters that also aren't white space or math symbols; include control characters): `[\n\t\r\f\uFEFF\u0B\p{P}\p{S}\p{Z}\p{L}\p{M}\p{N}]`
- [White space characters](#): `[\t\f\r\n\u0B\u0020\u00A0\u1680\u180E\u2000\u2001\u2002\u2003\u2004\u2005\u2006\u2007\u2008\u2009\u200A\u200B\u202F\u205F\u3000\uFEFF]` and also `[\p{Z}]`
- [Code or formatting characters](#) (HTML, emails, URLs, etc.): `<[^>]*>|([\^.\@\\s]+)(\.[\^.\@\\s]+)*@([\^.\@\\s]+\.)+([\^.\@\\s]+)|((https?|ftp|gopher|telnet|file|Unsure|http):((//)|(\|\\)))+[\w\\d:#@%/$()~_?\\+ -=\\|\\.&]*|Off(Off)+|_(_)+`
`[\p{N}\p{S}\p{P}]+: 50877452 (tokens containing no letters)`
`<[^>*>: 60010 (HTML, XML)`
`([\^.\@\\s]+)(\.[\^.\@\\s]+)*@([\^.\@\\s]+\.)+([\^.\@\\s]+): 12337 (emails)`
`_(_)+: 6188 (fill in the blank underscores)`
`((https?|ftp|gopher|telnet|file|Unsure|http):((//)|(\|\\)))+[\w\\d:#@%/$()~_?\\+ -=\\|\\.&]*): 2307`
`Off(Off)+: 172 (repeated Offs)`



Note that the 8 most common “unrecognized” characters (007f, 008a, 009c, etc.) are control characters. “Unrecognized” here means characters that aren’t a normal part of a language’s writing system (exactly which characters fit that description is what is being studied here).



Some punctuation symbols are very common, while the great majority appear infrequently.



Note that the most common math character is fffd (◆), where unrecognized characters are grouped together. It looks like most math characters and many punctuation characters aren't worth keeping in a language model sense, and some dashes, quotation marks, and other punctuation can be grouped together. I imagine characters like "+", "\$" being there for a reason though, so depending on the use case it could be good to keep less common punctuation symbols.

In summary, I suggest combining spacing, quotation marks, apostrophes, and dashes, using the `\p{L}` letter category, keeping a few math category characters like © and \$, perhaps keeping most (or all) punctuation, and removing unrecognized characters and uncommon symbols.

For removing HTML and XML, I noticed that the regular expression `<[>]*>` will sometimes include a very large portion of a file, until it finds the closing bracket. To evaluate this, I printed (without printing repetitions) and counted the number of occurrences for the following regular expressions (click the link to see examples):

[Brackets with one or more new lines in between:](#) `<[>]*\n+[>]*>` - 50 occurrences

[Brackets with one line in between:](#) `<[>\n]*\n[>\n]*>` - 4 occurrences

Strings found:

<Gesangbuch, Nr.

>

<, 96,06 128:Q2.20

<2./0,0/ <2 D/76?-38,/ 96,06 A/<@/40>

[Brackets with no lines in between](#): <[^\n]*> - 60007 occurrences

Most or all of the occurrences found with more than one line shouldn't be removed (they contain non-formatting text), and the strings found with just one line between brackets aren't important to remove if we also remove any words or tokens that contain only numbers and punctuation or other symbols.

To determine if it's safe to remove tokens with no letters, the following file was generated with a large number of examples of tokens found using the proposed regular expression

[p{N}\p{S}\p{P}]+ :

[Non-words Examples](#)

Since 50877452 of these tokens were found (see counts at the beginning of this section), removing these may make a significant difference, and the examples appear to be correct, unless it is desirable to maintain years or scripture references in the text. It is also possible to maintain only (or almost only) years and scripture references while removing all others that fit the regular expression above.

5.5 Incorrect Casing

In order to know what type of casing should be changed and how, lists were created of examples in the texts of different casings. The following lists were created:

[Word Case Likelihoods](#)

Examples of casing of words and their likelihoods (the number to the left of the word is the language model likelihood and the number to the right is the backoff-weight):

| | | |
|------------|-----|-------------|
| -2.1273189 | the | -0.80314577 |
| -2.716161 | The | -0.38974512 |
| -4.223266 | THE | -0.2248665 |
| -6.5227585 | ThE | -0.11694829 |

| | | |
|------------|----|-------------|
| -2.283247 | of | -0.67790955 |
| -4.08699 | OF | -0.3125919 |
| -4.089751 | Of | -0.35994324 |
| -6.5227585 | oF | -0.11694829 |

| | | |
|------------|---|-------------|
| -2.4493287 | a | -0.72595936 |
| -3.1952667 | A | -0.28206104 |

| | | |
|------------|--------|------------|
| -3.0811715 | Church | -0.5509522 |
|------------|--------|------------|

| | | |
|------------|--------|-------------|
| -3.949039 | church | -0.43682495 |
| -4.9466105 | CHURCH | -0.2573554 |
| -6.5227585 | CHurch | -0.11694829 |
| | | |
| -4.0774627 | house | -0.48284513 |
| -4.5707574 | House | -0.35330078 |
| -5.6514564 | HOUSE | -0.17922539 |
| | | |
| -4.865117 | Apple | -0.24062188 |
| -5.2516074 | apple | -0.14125426 |
| -6.5227585 | APPLE | -0.11694829 |
| | | |
| -4.8906193 | laugh | -0.38095817 |
| -6.1705723 | Laugh | -0.2175827 |
| -6.1705723 | LAUGH | -0.34355363 |
| | | |
| -4.62422 | USA | -0.5954068 |
| -6.315143 | usa | -0.24685998 |
| | | |
| -6.1705723 | USPS | -0.11694829 |
| | | |
| -4.483686 | lds | -0.40179306 |
| -3.5999858 | LDS | -0.4805163 |

The language model seems to do a decent job of showing abbreviations as most likely all cased, and words that aren't names as all lower case (except with Apple, which is also a company).

[Words with the first letter uppercase](#)

Regular expression: `(^\s)\p{Lu}+[\s$]*`

Most of these appear to be names or proper nouns.

[Words with the first and middle letters uppercase](#)

Regular expression: `(^\s)\p{L}+[\s$]*\p{Lu}[\s$]*`

Most of these are all uppercase, names, or combined words.

[Words with the first letter lowercase and at least one uppercase in the middle](#)

Regular expression: `(^\s)\p{L}+[\s$]*\p{Lu}[\s$]*`

Many of these are from languages other than English, but for those words that are English, some are combined, but many are words with incorrect casing.

Consequently, the third regular expression is likely to be more effective at finding words with incorrect casing in English. It may also be effective to find words that have more mixed casing within the word, even if it starts with an uppercase letter.

Handling words that are all upper case:

[Words that are only uppercase](#)

There are some abbreviations like USA, USPS, ISSN, though most of the caps seem to be names of organizations, titles, and information like "ILLUSTRATION BY VAL CHADWICK BAGLEY." Abbreviations we would expect to find in the language model, though a list of abbreviations could be effective for some languages.

[All Language Model Caps](#)

[Language Model Caps -1.0 Cutoff](#)

[Language Model Caps -3.0 Cutoff](#)

[Language Model Caps -6.0 Cutoff](#)

To see how effective the language model will be in correcting casing for words that are all caps, I searched the English language model for capitalized words, and listed the upper case words from the language model (of more than one consecutive letter) and did the same thing with likelihood cutoffs of -6.0, -3.0, and -1.0, to see if that would narrow things down more to abbreviations or commonly uppercase words. It did to some degree, though most are not abbreviations.

6 Conclusions

6.1 Introduction

Here we draw conclusions from the experiments to determine how to continue in the future to correct text from PDF documents.

6.2 Conclusion

Most of the work in correcting the text of PDF documents is evaluating the texts and determining what the most effective methods are for correction while not creating more errors. Also, adaptation of methods is required for different languages (especially with casing). However, with a good (comprehensive and accurate)

language model that can be trusted, many corrections, such as splitting combined words and correcting casing, can be done with high confidence.

6.3 Future Work

Combined words

Bigrams (and to a lesser extent trigrams) can be expected to significantly improve results. Also, a separate dictionary and/or a better language model could be used that doesn't contain combined words, in which case using a cutoff value may not be necessary.

Combined lines

A language model that can be used to It will likely help to perform this before removing extra whitespace, as extra whitespace is often an indicator of a combined line.

Split lines

Unrecognized words at the end of a line can be concatenated to the beginning of the next line, and also the language model can be used to determine if a line has an unlikely sentence ending and a very likely combination with the beginning of an adjacent line (or a line with 1-5 lines away).

Mixed words

Where words are not likely to be together according to the language model, the language model can be used to determine if there are other orderings on the same line that would be more likely in the language.

Misspelled words/missing letters

I started to implement a spell checker that uses the given language model, and this spell checker uses normal edit distance as well as soundex, but there was not enough time to try this and show results. The first step is seeing what kinds of corrections the spell checker would make, in order to identify and count how many words need to be corrected.

Incorrect casing

It may be effective to find words that have mixed casing (multiple case changes) within the word, even if it starts with an uppercase letter. Care must be taken to not change the casing of correct words.

Unwanted characters

It may help to use different sets of punctuation, symbols, and even letters for each language, based on how much each character is used in each language.

More non-word tokens (separated by spaces) containing letters, such as identification numbers, could be removed by a more complicated regular expression, which could include only words that have more than three characters and/or have a disproportionate number of non-letter characters to letter characters. This would have to be tested.

7 Appendix

This section contains files and links for reference.

[Project code on GitHub](#)

Word lists:

<http://dreamsteep.com/downloads/word-games-and-wordsmith-utilities/120-the-english-open-word-list-eowl.html>

Manually corrected files:

Test files and errors to be corrected.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---------------------|-------|---|---|----|----|----|----|----|----|----|
| Combined words | 2 (0) | 0 | 2 | 3 | 11 | 28 | 32 | 25 | 3 | 7 |
| Empty words | 25 | 4 | 4 | 1 | 1 | 0 | 1 | 26 | 28 | 84 |
| Combined lines | 64 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| Split lines | 19 | 0 | 1 | 1 | 1 | 0 | 0 | 7 | 2 | 0 |
| Mixed words | 13 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 3 | 0 |
| Misspelled words | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Incorrect Casing | 27 | 0 | 2 | 10 | 0 | 0 | 0 | 0 | 0 | 0 |
| Unwanted Characters | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 18 | 25 | 0 |

Table 2. Manually found errors in test files.

The top row is the number associated with each file (test_1.txt, etc.).

The first column refers to the name of the error (see section 2. Problems).

Note: Evaluation was done manually, including for “Unwanted Characters.” Due to time and the difficulty of manual evaluation, this data was only used to evaluate correcting combined words (See Table 1 at section 5.2).

The complete list of words that were split (and how they were split) for each test file are found at the following link: https://github.com/JoshuaMathias/text-correction/tree/master/split_output

The test files are found at the following link:

https://github.com/JoshuaMathias/text-correction/tree/master/test_files

The original test files are named test_1.txt, test_2.txt, etc.

The test files that were manually corrected for combined words have “_split” at the end of the name.

The test files that were manually corrected for all errors (“golden standard” files) have the “_corrected” in the file name.

The test files that were corrected and then reviewed by Ryan Lee have “_review” in the file name.