City University of New York (CUNY)
CUNY Academic Works

Dissertations, Theses, and Capstone Projects

CUNY Graduate Center

2-2015

Techniques for Automatic Normalization of Orthographically Variant Yiddish Texts

Yakov Peretz Blum
Graduate Center, City University of New York

How does access to this work benefit you? Let us know!

More information about this work at: https://academicworks.cuny.edu/gc_etds/525
Discover additional works at: https://academicworks.cuny.edu

This work is made publicly available by the City University of New York (CUNY).
Contact: AcademicWorks@cuny.edu

TECHNIQUES FOR AUTOMATIC NORMALIZATION OF ORTHOGRAPHICALLY VARIANT YIDDISH TEXTS

by

YAKOV PERETZ BLUM

A master's thesis submitted to the Graduate Faculty in Linguistics in partial fulfillment of the requirements for the degree of Master of Arts, The City University of New York

2015

2015
© Yakov Blum
Some rights reserved.
This work is licensed under the
Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.
http://creativecommons.org/licenses/by-nc-nd/4.0/

This manuscript has been read and accepted for the
Graduate Faculty in Linguistics in satisfaction of the
requirement for the degree of Master of Arts.

Martin Chodorow

_______________________ ____
Date Thesis Advisor

Gita Martohardjono

_______________________ ____
Date Executive Officer

THE CITY UNIVERSITY OF NEW YORK

Abstract

TECHNIQUES FOR AUTOMATIC NORMALIZATION OF ORTHOGRAPHICALLY VARIANT YIDDISH TEXTS

by

YAKOV PERETZ BLUM

Adviser: Professor Martin Chodorow

Yiddish is characterized by a multitude of orthographic systems. A number of approaches to automatic normalization of variant orthography have been explored for the processing of historic texts of languages whose orthography has since been standardized. However, these approaches have not yet been applied to Yiddish.

Using a manually normalized set of 16 Yiddish documents as a training and test corpus, four techniques for automatic normalization were compared: a hand-crafted set of transformation rules, an off-the-shelf spell checker, edit distance minimization with manually set weights, and edit distance minimization with weights learned through a training set.

Performance was evaluated by calculating the proportion of correctly normalized words in a test set, and by measuring precision and recall in a test of information retrieval.

For the given test corpus, normalization by minimization of edit distance with multi-character edit operations and learned weights was found to perform best in all tests.

Acknowledgments

First and foremost I would like to thank my thesis adviser Martin Chodorow for his unflagging support, availability, and patience over the years as this thesis morphed from one project to another before finally assuming its present form.

Thanks to Assaf Urieli for his help in an earlier incarnation of this project, and for providing both the corpus and the lexicon which made this study possible.

Thanks to the Paris Yiddish Center—Medem Library for permission to use the lexicon based on their dictionary for this project.

Thanks to Tanya Panova for sharing the Yiddish transcription code developed by her and her colleagues, and for reviewing a draft of this study.

Finally, thanks to Baruch Blum, Sosye Fox, Alec Eliezer Burko, and Isaac Bleaman for reading a draft of this thesis and for their valuable comments and corrections. The final content is of course my sole responsibility.

This thesis is dedicated to my friend and mentor Arthur (Avrom) Schwartz, who introduced me to the studies of both linguistics and Yiddish.

Table of Contents

List of Tables

1 Introduction

Yiddish, in contrast to many other literary languages in the twentieth and twenty-first centuries, displays a relatively broad range of orthographic variation up to the present day. In the following introductory remarks, I will explore the origin of this variation and discuss some of the challenges it poses for working with Yiddish-language texts in digital form. I will also review some of the digital resources available today in Yiddish, which are much more limited than those of the major world languages upon which most research is performed, and explain how the challenges presented by orthographic variation are compounded by particularities in the way the Yiddish alphabet is encoded in today's computers. Then I will look at the approaches that have been taken by researchers in dealing with orthographic variation in other languages, before finally presenting the research conducted for the present work.

2 Spelling Variation in Yiddish

Yiddish is written with the Hebrew alphabet, which originally represented consonant sounds only. As the original orthographic system of the Hebrew Bible did not provide all the information necessary to read the text aloud, a system of diacritics was developed by the Masoretes (a group of Jewish scribes) in Tiberia in the second half of the first millennium CE. These diacritics consisted of points and lines placed around or inside the letters. This allowed the Masoretes to preserve their reading tradition of the text as an overlay to the original, without needing to make changes to the letters themselves. The additional information included all vowel sounds, distinctions between spirantized and non-spirantized allophones of certain phonemes, and gemination of consonants. (An additional system of cantillation marks indicates stress placement and some syntactic information, guiding the traditional chanting of the text.) Over time, certain letters came to be used to represent vowels as well. The primary innovation introduced when adapting the Hebrew alphabet for Yiddish writing — a development over centuries — is that this process was brought to completion, so that all vowel sounds are represented by a letter in the Yiddish alphabet, with the exception of words of Hebrew or Aramaic origin, which are written using a traditional spelling for those words (Weinreich 1939).

Yiddish writing does not make use of the full system of diacritics — nor does it need to, as all vowels are represented by letters, there are no geminate consonants, and there are no spirantized/non-spirantized allophones of a single phoneme as in Hebrew. However, a very limited set of diacritic and letter combinations is used in most systems of Yiddish orthography. The diacritics are not used systematically, but merely in order to distinguish what may be thought of as distinct letters. For instance, /a/ and /ɔ/ are represented by אַ and אָ, respectively. The marks at the bottom of these two vowel letters serve to visually distinguish them from each other, but in Yiddish they are not generally used with any other letters, whereas in Hebrew they could be placed under any consonant to represent those respective vowel sounds following that consonant.

There was no standardization in the spelling system of written Yiddish in the premodern era, as was then the case with most other written languages. Over time, most national European languages gradually gravitated towards a uniform standardized spelling. Variation in German spelling, for instance decreased steadily over time from the 15th to the 19th centuries (Pilz 2005). English, similarly, shows a steady decrease in spelling variation from about 1500 to about 1700, by which time the spelling had been largely standardized (Baron 2011). Other languages, such as Spanish and French, with national institutions defining spelling standards, reached orthographic stability even earlier (Gerlach and Fuhr 2006). Yiddish, in contrast, saw a number of radical spelling changes in the 19th and

20th centuries. These were a result of various cultural and political factors. There were two major counteracting trends (Schaechter 1999): The first was to more closely approximate German spelling by inserting the corresponding Hebrew characters. For instance the word for 'cow', /ku/, might previously have been spelled קו, but since in German the cognate word is spelled Kuh, the Yiddish spelling קוה may appear. In general this resulted in numerous "silent" letters in Yiddish words, and other changes. The other counteracting trend was increased phoneticization, namely, to strive towards a one-to-one correspondence between characters and phonemes. The most extreme result of this trend, undertaken in the Soviet Union, included discarding the five word-final forms and spelling phonetically words of Hebrew or Aramaic origin, which in other Yiddish orthographic systems have a spelling derived from the origin language.

Standardization of spelling was seen as important to the legitimization of Yiddish as a language independent from German, and in developing it from a "folk-language" (*folkshprakh*) to a "language of culture" (*kulturshprakh*). At the first international conference in support of the Yiddish language, the 1908 Czernowitz Conference (of Czernowitz, Bukowina), organizers put the regularization of Yiddish orthography at the top of the agenda. In 1937, YIVO (Yidisher Visnshaftlekher Institut, or Yiddish Scientific Institute) published the *Takones fun Yidishn Oysleyg* (Rules of Yiddish Spelling). However, the factors which hastened adoption of standardized spelling in nation-states (such as compulsory public education and other governmental incentives) were never available to Yiddish speakers, except for a short period in the Soviet Union. Therefore the proliferation of spelling styles continued.

The Second World War saw the annihilation of a large proportion of Yiddish speakers, and the almost complete destruction of all Yiddish-speaking society. In 1952, Stalin executed the major Jewish writers in the Soviet Union and essentially put an end to what remained of Yiddish cultural activity there. YIVO found a new home in New York during the war, and its rules of 1936 are accepted as the standardized Yiddish spelling in academic and most Yiddishist circles today. However, the only remaining Yiddish-speaking communities today are composed of Haredi Jews, who never adopted these rules, and instead follow various other spelling traditions. (The Haredim are a group of Orthodox Jews who reject modern secular culture.) Thus, the vast majority of Yiddish publications of both the past and the present do not follow what might otherwise be called the “standard” Yiddish orthography.

“Standard [Yiddish] orthography” in the remainder of this paper refers to the orthography specified in YIVO's *Takones*.

3 Yiddish Digital Resources

A choice few of the world's languages enjoy a wealth of electronic resources available to the interested computational linguist, including raw resources such as large manually-tagged corpora, parallel texts, and digital dictionaries, as well as tools such as part-of-speech taggers, stemmers, sentence parsers, and highly accurate optical character recognition programs. Yiddish is among the vast majority of languages for which many or all of these resources are not (yet) available, and can thus be designated a *resource-poor* language. Fortunately, there has been steady growth in the digital resources available to the Yiddish researcher. Following are listed the main sources of Yiddish-language corpora available today.

The National Yiddish Book Center has scanned over 11,000 works of Yiddish literature and made them freely available online[1], but they are only available as images, and so are not text-searchable. The same is true for a number of websites which have scanned and uploaded historical newspapers and journals in Yiddish, many of which can be found through the Index to Yiddish

[1] https://archive.org/details/nationalyiddishbookcenter

Periodicals[2] of Hebrew University. Yiddish books scanned by Google as part of its Google Books project have been OCR'ed and are text-searchable, but the OCR is optimized for Hebrew text, not Yiddish, and so there are numerous errors. The same is true for the scanned Yiddish books and journals at hebrewbooks.org.

The Corpus of Modern Yiddish[3] (CMY) is a project currently underway at the University of Regensburg. One goal of the project is to gather in one place as many Yiddish textual resources in digital format as possible, and add to these, to eventually reach a representative corpus of several million words covering various genres, styles, and periods. The largest single source of digital Yiddish text in standard orthography is probably the website of the Yiddish Forward[4], a Yiddish newspaper which has made its articles available online since 2006. The majority of the text in the CMY is currently from this source. Another source is a collection of 35 Yiddish short stories and literary excerpts[5] (redone in the standard orthography), as well as the entire Yiddish translation of the Bible by Yehoash[6], typed a number of years ago by volunteers for a project at the site mendele.org. Additionally, the CMY is beginning to incorporate text from Yiddish internet forums, populated primarily by Hasidic Yiddish speakers. This Yiddish tends to be much more vernacular, and not spelled with the standard orthography.

The Penn Yiddish Corpus (Santorini 1997/2008) is a corpus of Yiddish texts, transliterated into Latin characters, of various dialects and eras ranging from 1462 to the 1990s. The corpus contains roughly 200k word tokens and is manually annotated for part-of-speech and syntactic parsing in the Penn Treebank format.

[2] http://yiddish-periodicals.huji.ac.il/
[3] http://web-corpora.net/YNC/search/
[4] http://yiddish.forward.com/
[5] http://yiddish.haifa.ac.il/Stories.html
[6] http://yiddish.haifa.ac.il/texts/yehoyesh/tanList.htm

4 Complexities of Unicode Representation and Processing of Yiddish

There are a number of reasons why processing Yiddish digital text can be challenging. Before the advent of Unicode, there were multiple, mutually inconsistent ways of encoding Hebrew characters. Also, because not all programs were designed to display right-to-left text, sometimes the text was input backwards to make it readable. The widespread implementation of Unicode and its BiDi (bidirectional) algorithm have helped bring order to the chaos. However, there are still a number of subtleties with Unicode which must be kept in mind. For instance, there are certain digraphs in Yiddish which have their own Unicode character. One example is *vov-yud*: ױ. Thus, the word אױ (*oy*) can be encoded in two ways: either

```
א    U+05D0    HEBREW LETTER ALEF
ױ    U+05F0    HEBREW LIGATURE YIDDISH VAV YOD
```

or

```
א    U+05D0    HEBREW LETTER ALEF
ו    U+05D5    HEBREW LETTER VAV
י    U+05D9    HEBREW LETTER YOD
```

The letters with diacritics, such as אַ, אָ, כּ, and שׂ, also have two possible representations, either as a single Unicode point, or as a combination of two points: the bare letter and the diacritic. The letter אָ can thus be encoded in Unicode either as

```
א    U+05D0    HEBREW LETTER ALEF
ָ     U+05B8    HEBREW POINT QAMATS
```

or as

```
אָ    U+FB2F    HEBREW LETTER ALEF WITH QAMATS
```

Since these sequences represent the same Yiddish letters, they should be interchangeable. However, this is unfortunately not always the case. For example, the National Yiddish Book Center now offers search in Yiddish text through the titles and authors of its catalog of uploaded works. Previously the database only included a transcription of the title into Latin characters, which made search difficult because the system of transcription is not intuitive and uses characters such as ḳ and ś which don't exist on the standard keyboard. However, the search function in Yiddish letters is poorly implemented in that it searches only for precise Unicode matching. At the time of this writing, a search for titles containing the word גרויס, inputting the ױ as a single Unicode character results in no matches. However, a search inputting the *vov-yud* as two separate characters results in 16 matches. Not knowing how the titles were originally input (and they might have been input inconsistently), one must search both sequences in order to ensure getting all results. As there are a number of these interchangeable Unicode points, the number of possible sequences can quickly grow. As short a word as אַפּראָפּאָ *apropo* 'apropos' can be input in 32 possible combinations, and for titles with multiple words the number could climb to the hundreds — and that is before accounting for actual spelling variations involving different graphemes. It can thus be a challenge to find a book in the database even if one knows the exact title in Yiddish. And the National Yiddish Book Center is not alone in this type of implementation. (To be fair, the implementation is actually that of the site archive.org, where the books are digitally hosted.)

As there has been a great variety of spelling systems in Yiddish throughout the 20th century and up to the present day, handling spelling variation in Yiddish is even more important than in languages such as English and German, which have enjoyed a mostly standardized spelling for the last century. In order for queries on the various Yiddish corpora to work correctly, and for work to begin on NLP projects such as Yiddish part-of-speech (POS) tagging, it is necessary to solve the problem of orthographic variation.

5 Previous Work on Handling Spelling Variation

Work on regularization of forms not conforming to spelling standards is extensive — at least in English — but most of this work has concentrated on correcting spelling errors in English text. Much less work has been done on dealing with nonstandard orthographic forms which are not errors. Nearly all the work that I have found in this area has dealt with historical (as opposed to contemporary) data, with much of the early work dealing specifically with German and English. Strunk (2003), is exceptional, dealing with contemporary Low Saxon, which lacks a standard orthography.

There has been some difference as to the definition of the problem which these papers attempt to solve. Some focus on the problem of information retrieval — namely in increasing the recall levels of a query which otherwise would miss the forms which do not exactly match the entered string. Others address the problem of normalization — automatically transforming nonstandard forms into corresponding standard ones. The latter problem can, on the one hand, be a prerequisite for other processes — for example, POS tagging — but it can also be used in order to improve information retrieval. For instance, upon indexing a collection of documents, the nonstandard forms can be stored together with one or more normalized forms so that the document is returned in a query. Conversely, any method of retrieving a nonstandard form in a document corpus is an implicit mapping of that form to a standard one (if such exists). So the problems are two sides of the same coin.

Strunk (2003) was one of the first to tackle the problem of information retrieval in languages without a fixed orthography, and as mentioned above, the only one I found who focused on contemporary data. The problem he addressed was finding the orthographic forms in a search index which are variants of the user-input query. His system comprises two steps: first, a coarse search restricting the index to the most likely tokens by using an encoding of the syllable structure or the first letter of the word. Next, on this restricted index he calculated edit distance using both Levenshtein distance and a weighted edit distance (described below), and using various methods of parsing each word into letters or graphemes. Among the results, he found a tradeoff between precision and recall, with the best F-measure achieved by the weighted edit distance, using a parsing which attempted to find the best correspondence between graphemes.

Kempken, et al. (2006) used distance algorithms to identify orthographic variants of a standard form. The algorithms evaluated were:

- Levenshtein distance (edit distance with unit weights);
- similarity based on shared bigrams;
- a phonetic matching algorithm which combines edit distance with grouping of letters that produce similar sounds;
- a stochastic edit distance defined by the probability of string operations, learned from a training set of string pairs;
- "Flexmetric": a form of weighted edit distance, where the weights are determined manually.

The authors evaluated these algorithms using the measures of precision and recall in an information retrieval task. A list of pairs of words — a historical deviant spelling and the standard spelling — was prepared. The corpus to be searched through was a dictionary of modern German words, with the historical word forms from the list added and the corresponding modern terms removed. Each query was a standard word, and a positive result was one of the historical spellings of that word. The best results were achieved with the stochastic and "FlexMetric" algorithms.

Normalization of spelling variants is a related problem to correction of spelling errors — in both cases, one seeks to find a word in the standard modern orthography which matches a given form — but the variation encountered differs for each case. Nevertheless, researchers have sought to take advantage of the advances made in spell-checking technology while adapting the techniques to the specifics of normalizing historical forms. Schneider (2001) attempts normalization of 18th century English by modifying the open-source spell checker Aspell (Atkinson 2004).

Rayson, et al. (2005) consider the task of normalizing historical forms found in English literary works of the 16th – 19th centuries. They compare the performance of a system called VARiant Detector (VARD) to that of two spell checkers: Aspell and the spell checker in Microsoft Word. (The latter was chosen as a baseline because it is the most widely used word processor.) VARD works through a very simple mechanism: a search-and-replace is performed on a text, using a manually compiled list of over 45,000 variant word forms and their standard forms. The authors evaluated these systems on four documents from the same time period as that from which the VARD pair list was compiled. It was found that VARD was more precise in identifying nonstandard variants, and correctly matched the standard form at a higher rate than the two spell checkers.

Bollmann (2012) compares five normalization methods:

- Rule-based: A set of rewrite rules, learned from training data, that replace character sequences with other character sequences within a given context (e.g., j → ih / # _ r, or ‘j’ is replaced by ‘ih’ between a word boundary (‘#’) and ‘r’)
- Levenshtein distance
- Flexmetric (as described above)

- "MultiWLD" (Weighted Levenshtein Distance with multi-character substitutions): similar to Flexmetric, but it is directional (as opposed to symmetric), and weights can be defined for multiple character substitutions. Weights are set manually.

- Wordlist mapper: whole word substitutions are learned from training data

The results obtained suggested that the highest accuracy can be achieved when using the methods in a chain, beginning with the wordlist mapper. A clear best performer could not be determined, although Levenshtein distance performed especially poorly.

Bollmann (2013) continues the line of research from the previous paper, modifying only the MultiWLD method, so that instead of the weights being set manually, they are learned from a training set. The resulting algorithm is thus quite similar to that described by Brill and Moore (2000). An algorithm of this type is among those evaluated in the present study. Bollmann notes that calculating edit distance over an entire lexicon is computationally expensive, adding that heuristics might be used to reduce the number of calculations.

While early work focused mainly on German and English, in recent years a number of studies have appeared applying these techniques to a wider variety of languages.

Oravecz, et al. (2010) applied the noisy channel model presented in Brill and Moore (2000) to a corpus of 24 short Old Hungarian texts of the 12th – 16th centuries. One necessary addition was a morphological analyzer: since Hungarian is an agglutinative language, the number of possible word forms would be too large for any lexical list. Another tool used was a source model (a model of the contemporary language) based on a 10 million word subcorpus of the Hungarian National Corpus. The model was able to return the correct normalization among a list of 20 suggestions 88% of the time, and through decision tree re-ranking was able to rank the correct form highest in the list at a rate of 73%.

Scherrer and Erjavec (2013) test normalization on 18th and 19th century Slovene using off-the-shelf character-based statistical machine translation tools. A set of 45k word pairs out of a manually annotated corpus of 300k words was used both as a bilingual training model and (using the contemporary words) as a model of the target language. They created three models, one for each 50-year period between 1750 and 1900, and tested on a set of 3.5k words from a 3M word partially annotated corpus. Using supervised learning, they were able to achieve a performance improvement of 57% over the baseline, and in an unsupervised test, an improvement of 33.5%.

Sánchez-Martínez, et al. (2013) also utilized statistical machine translation to standardize historical Spanish forms from a large corpus of historical texts dating from the 15th to the 18th centuries. The parallel corpus training set comprised 600k words, and normalization was evaluated on a 5k word test set. Performance was compared with the baseline, with the Ispell spell checker, and with a "naive" approach that selects the most frequent modernized form for each historical form in the training set. It was found that the machine translation normalized at a character error rate of 0.21%, less than half that of the naive approach. The Ispell spell checker did not perform better in this measure than no normalization at all.

Kirjanov, et al. (2014), as part of work on the Corpus of Modern Yiddish, developed a program to automatically transcribe Yiddish text into Latin characters. As a first step in transcribing a given word in this system, a series of transformations is performed on the word to convert it to a standard orthographic form. The authors claim a 94-97% normalization accuracy rate on nonstandard text using this system. However, the methods by which they arrived at this number are not discussed in the paper. A variation of the code used in that study, provided courtesy of one of the authors, is evaluated in the present study.

6 Present Work

The present study compares normalization methods similar to the ones discussed in the above literature, as applied to Yiddish. One difference between this study and most of the others is that the focus is not necessarily on historical data. While there were major efforts at standardization of Yiddish spelling, the standards are very far from achieving universal usage. Therefore, the applications for orthographic normalization include contemporary data as much as historical data. Most of the data in our corpus was published in the 20th century. Additionally, this will be the first such study involving a language that does not employ the Latin alphabet.

The performance of each of the normalization methods was evaluated directly by comparing the proportions of correctly normalized forms from a test corpus. Additionally, the methods were comparatively tested by studying their performance in an information retrieval experiment, to see if they can be used to obtain more relevant results from a query on a collection of Yiddish documents.

The following sections describe the components of the study: the corpus, the lexicon, and the standardization methods.

6.1 The Corpus

The corpus used in this study is a set of 16 documents provided courtesy of Assaf Urieli. Each of these documents consists of one to several pages taken from a Yiddish book from the aforementioned digital collection of the National Yiddish Book Center. The years of publication of these books range from 1892 to 1992, and the places of publication include Warsaw, Moscow, Munich, Vilnius, Jerusalem, New York, and Buenos Aires. Details about each work including the URL where it can be accessed online are in the appendix. The pdfs from the collection were run through an OCR program called Jochre in development by Urieli (Urieli and Vergez-Couvet 2013). Then the text was checked manually by Urieli and his team, and corrected to match the original spelling in the book. The corpus was created to test the Jochre system, but its composition makes it very useful for the present study. The documents were selected to represent a variety of Yiddish orthographic styles, and these styles are reproduced faithfully in the digital text. It is the only available corpus of this kind that I know of.

The documents were saved as plain text files, and then tokenized[7], resulting in 37,092 tokens. The list of tokens was copied into a spreadsheet, one token per row, within which I manually input the standard form of each token. In order to streamline the task, I used the manually-set weighted edit distance program (described below) to propose a list of possible standard candidates, which were included on the spreadsheet row. If the token was already standard, I left the cell blank. If the standard form was among the options proposed, I input the corresponding numeral (1 for the first option, 2 for the second, etc.). Only if the standard form was not among the suggestions did I have to manually type it in, which was the case for about 2,000 tokens. This basic system greatly sped up the manual correction process, so that I was able to review all the tokens in about a dozen hours or so. In the end, a few spreadsheet formulas transformed the input data into a complete pair list of forms found in the documents matched with their standard counterparts.

A number of tokens were discarded: first of all, punctuation, numbers, and single letters (the result of the OCR reading words with too much space between the letters), with the exception of the word אַ, an indefinite article; secondly, a number of other character sequences that I determined during the manual review were not words (often this was due to a word being split up between lines, or other issues in the OCR). Compound nouns were not split. The final number of word tokens used as the

[7] For tokenizing I used the NLTK tokenizer (Bird 2009), but replaced its dependency on the standard library Python module *re* with the alternative model by Barnett (2014) so that it could more effectively handle Yiddish text in Unicode.

corpus in this study was 28,475. This word pair list was saved as a .csv file to be accessed by the Python code.

6.2 Lexicon

Most of the normalization methods considered in this study depend on a lexicon, a reference list of standard forms. The lexicon used here was compiled by Assaf Urieli on the basis of the *Dictionnaire Yiddish-Français* of the Biblothèque Medem (Urieli and Vergez-Couvet 2013). This dictionary is one of the most comprehensive bilingual Yiddish dictionaries currently in existence, and the largest one which is readily available in digital form. Furthermore, it is strict in its compliance with the YIVO orthographic norms. The list of roughly 37,000 lexemes in the dictionary was expanded by Urieli to a total of 396,500 inflected forms. As the lexicon for this study, I used this list with some minor modifications: the addition of some very high-frequency words and a number of contractions, and correction of two systematic errors in the generation of past participles and adjective inflections.[8] (However, the list did not include the proper nouns added by Urieli (2013).)

As the lexicon is loaded from the text file into the program as a Python object, the words are all normalized with regards to Unicode so that words can be compared based solely on the letters that compose them and not the particularities of how they were encoded in Unicode (cf. discussion regarding Unicode above). The normalization proceeds as follows: first, the form is transformed to the Unicode normalization form known as NFKD: Normalization Form Compatibility Decomposition. In general, this normalization is supposed to decompose ligatures into their component parts. However, the Yiddish digraphs וי, יי, and וו are not, according to the Unicode specification, considered to be

[8] The high-frequency words were: מען; דו; סײַ. The contractions were: ס׳איז; מ׳איז; ס׳וועט; ס׳האָט; מ׳האָט; כ׳וועל; ר׳; דר׳רעד; ר׳האָט; ר׳איז; ס׳וואָלט; ס׳אַ; כ׳האָב; ס׳זאָל; ס׳נײַ; כ׳בין. The two errors affected respectively a total of 1,883 and 3,096 lexical entries.

canonically equivalent to their character components[9]. So this decomposition must be done separately afterwards. However, in order to not have the ligature *pasekh-tsvey-yudn* (ײַ) encoded as a *yud* followed by another *yud* with the *pasekh* diacritic (since the diacritic associates with both letters together), I replace sequences of (*yud*, *yud*, *pasekh*) with (*tsvey-yudn*, *pasekh*), or, in Unicode: (U+05D9, U+05D9, U+05B7) → (U+05F2, U+05B7). Finally, the Yiddish character *geresh* (׳), U+05F3 is often substituted with an apostrophe (') U+0027, which has no distinct meaning in Yiddish, so apostrophes are replaced with a *geresh* to complete the normalization.

Of the 28,475 tokens in the corpus, 19,494 (68%) were found in the lexicon, as opposed to 26,950 (95%) of their manually normalized forms. 3,132 of 7,196 types in the corpus were in the lexicon (44% coverage), and 4,727 of 5,661 manually normalized types (83.5% coverage).

6.3 Word Normalization Methods

Four methods are compared in this study:

(1) Rule-based: a set of rules designed by hand to transform a nonstandard word into its standard form

(2) Aspell: An open-source off-the-shelf spell checker

(3) Weighted Edit Distance (manually-set weights)

(4) Weighted Edit Distance (weights set based on a trained error model)

Each of these methods is described in more detail below.

[9] At least with respect to the ligature ײ (U+05F2, two *yud*s), there may be a good rationale for this. If this ligature is normalized under decomposition as two *yud* (U+05D9) characters, then two *yud*s followed by a *pasekh* would be normalized as the sequence *yud-yud-pasekh* which would be the same sequence as a *yud* followed by a single *yud* with a *pasekh* diacritic (also decomposed). These, however, should be distinguished.

6.3.1 Rule-based

This is a set of string transformations developed by Kirjanov, et al. (2014) for use in the Corpus of Modern Yiddish project. These transformations are performed as a series on an encountered word form with the intent of transforming it into a standard form. For instance, it is common in nonstandard Yiddish orthographies to spell the prefix פֿאַר־ as פֿער־. So there is a rule to convert all word-initial sequences פֿער to פֿאַר, with a set of exceptions extracted from a dictionary of standard forms beginning with פֿער.

I made some modifications to the rule set which were necessary since the original set was designed to prepare a word to be transcribed into Latin letters. For instance, in the latter case there is little danger in removing all instances of א, *shtumer alef*, which is silent: און → ון → un. But for the purposes of standardization alone, this should only be done in specific circumstances, as in וואו → וווּ.

Two examples of the algorithm's output are in Table 1. The first was correctly normalized by the algorithm, and the second was not successfully normalized.

Table 1: Sample Output from Rule-based Algorithm

Input	Output	Correct Normalization
טיהר	טיר	טיר
געהן	געהן	גיין

6.3.2 Aspell

GNU Aspell is an open-source spell checker maintained by Kevin Atkinson (2004). It is also already used in a real-world Yiddish application, the online version of the Comprehensive Yiddish-English Dictionary[10], which provides a list of suggestions when a Yiddish query is not found in the

[10] http://verterbukh.org/

dictionary. According to Harry Bochner, editor of the dictionary and the website, these suggestions are produced by Aspell (personal communication).

To implement Aspell in my program, I used the Python module pyenchant[11] which provides Python bindings to Enchant, a program which in turn provides access to various spell checker backends, including Aspell. The Yiddish lexicon described above was used as the dictionary for Aspell, from which it chooses its spelling suggestions.

An example of this algorithm's output is shown in Table 2. In this example, the correct answer, in bold, is ranked second.

Table 2: Sample Aspell Output

Input	Output
לעבען	1. לעבער **2. לעבן** 3. לענען 4. בעבען

6.3.3 Manually Weighted Edit Distance

This method involves determining the item in the lexicon with minimal edit distance from the given nonstandard form, and is used as a baseline for comparison with the next method using a noisy channel model. The edit distance computed is equal to Levenshtein distance (the minimal number of character insertions, deletions and substitutions that are needed to transform one string into another), except for the following heuristic modifications made to take advantage of some knowledge about common differences between nonstandard and standard Yiddish orthography:

[11] https://pypi.python.org/pypi/pyenchant/

1. Diacritics: Usage of diacritics is highly variable in the different orthographic systems, so the cost of removal, addition, or substitution of graphemes differing only in their diacritics was set to 0.2, instead of 1, as it would be under Levenshtein.
2. Final Forms: In Soviet texts, the five final character forms are sometimes replaced with their non-final counterparts. As no word in standard spelling ends with a non-final character, it is highly likely that a nonstandard word ending with a non-final character would correspond to a standard word form ending with the respective final character. Therefore, I assigned this substitution a cost of 0 (instead of Levenshtein 1), with one exception: If a nonstandard form ends with the character פ (this form with no diacritic is not considered to be a letter in the standard alphabet), we have an ambiguous case: the corresponding standard form could end with one of two characters —

    a. פּ — which corresponds to the sound /p/ and differs from פ in that it contains the diacritic *dogesh* (the dot in the center); or

    b. ף — the final form of the letter פֿ corresponding to the sound /f/.

    I considered the second case to be less likely than the first case (where a diacritic is missing), so I assigned the substitution פ for ף a cost of 0.3 (somewhat higher than the diacritic cost 0.2) instead of 0 as for the other final form substitutions.
3. "Silent" letters: A common trend in Germanized orthographies, as mentioned above, is the addition of "silent" letters, especially ה and ע. Another very common feature of nonstandard orthographies is the use of a silent א for certain purposes where it would be absent in the standard orthography. Therefore, I assigned the removal of one of these letters from a nonstandard form a cost of only 0.5, except in word-initial position, where they were unlikely to appear, or, in the case of א, are also mandated in the standard orthography.

Thus, the definition of edit distance used here differs from Levenshtein distance in a few key ways: it is weighted, non-symmetric, and depends on the position of the edit in the word.

The next step is to determine, for a given nonstandard word form, which item in the lexicon is of minimum edit distance. The well-known algorithm for computing edit distance between two strings using dynamic programming, known as the Wagner-Fischer algorithm, involves computing a matrix of (m+1)×(n+1) values, where m and n are the lengths of the two strings, and the values are the edit distances between all prefixes of the strings. Each value is computed by finding the minimum of the value of the adjacent previous cells plus the cost of, respectively, an insertion, deletion, or substitution, and the edit distance between the strings is the final value computed. (The algorithm can be easily adapted for our weighted, non-symmetric, and position-dependent measure.)

Calculating edit distance against each item individually would have been prohibitively slow, especially for the next model (as will be explained below), so I used an optimization method, inspired by the one Brill and Moore (2000) use, which takes advantage of the fact that many of the words in the lexicon share their first string of letters. I converted the lexicon into a prefix trie structure (Black 2011). The lexicon of almost 400K words and about 3.8 million characters is thus compressed to a trie of only about 600K trie nodes. To check a nonstandard word, edit distance is computed one node at a time, from the root down. Each node involves computing one column of the edit distance matrix. This way each column is only computed once, so instead of computing 3.8 million columns (number of characters in the lexicon), we compute at most 600K. Once a leaf node is encountered, corresponding to a full word in the lexicon, the result is stored and the search continues. A running list is kept of the best (smallest) scores. Here is a list of further optimizations employed, which, again, are even more important for the following edit distance method:

- Setting an absolute limit to the edit distance — once the algorithm reaches that limit on a particular branch it will not go further down the trie, but instead moves on to another branch. (This means if the edit distance of the best answer is greater than the limit, it might be missed, but if the limit is set high enough, then the best answer probably was not a very good result anyway.) For this study, the limit was set at 10 plus the length of the word, which is very high for this measure and probably not effective at eliminating many branches, but it made more sense for the next measure.

- Setting a maximum range in edit distance of results: As the algorithm walks down the trie, it keeps a running list of the best answers, but discards any answers with an edit distance higher than the best answer so far plus the maximum, and the trie walk stops searching down branches where edit distance is greater than the worst result on the list. Since edit distance never decreases as one goes down the trie, this saves time without losing any optimal results. For this study, the maximum range was set at 5.

- Setting a maximum number of results: If there are already this many results on the running list of best results, then as each new result is added, the worst one is dropped, and this again saves time because branches worse than the worst result are skipped. This maximum was set at 10 results.

- Due to the previous two optimizations, the search ends sooner if good results are reached earlier. Since the best answers are most likely to start with the same character as the search item (there is no weighting that favors changing the first character of the word), the trie walk begins down the branch corresponding to the first character of the word, instead of down arbitrary branches, as in the rest of the walk. This usually results in a significantly faster search time, and there is no loss in comprehensiveness of the search.

The result of the search is a list of suggestions from the lexicon together with their edit distances from the word being looked up.

An example output for the query פֶערְגֶענּיגֶען is shown in Table 3. The correct answer, in bold, is in this case the highest ranked.

Table 3: Sample Output of Heuristic Method

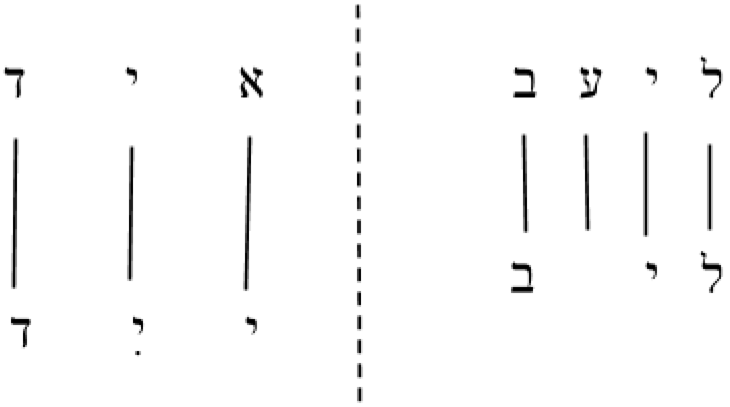
Input	Suggestion	Edit Distance
פֶערְגֶענּיגֶען	**פֿאַרגעניגן**	**2.5**
	פּראַניען	3.6
	געניגן	3.6
	באַגעניגן	3.6
	מערגעניקעס	3.6
	פֿאַרענגען	3.8
	אַפֿערגענומען	3.8
	פֿאָרגענומען	3.8
	פֿאָרגעענגען	3.8
	פֿירגענומען	3.8

I will refer to this method informally as the “heuristic” method, because the edit distance operation weights were set heuristically instead of learned from a training corpus, as in the next model.

6.3.4 Weighted Edit Distance (Learned Weights through Noisy Channel Model)

In this edit distance measure, the costs of additions, subtractions, and substitutions are also weighted, but instead of being set heuristically, they are calculated based on a noisy channel model. The model is intended to reproduce that described by Brill and Moore (2000), and will thus be referred to as the Brill method. Besides the weighting of the costs, this measure is also more powerful than traditional edit distance in that it models not only single-character edit operations, but also ones in which sequences of multiple characters can be added, subtracted, or substituted for each other (or single characters) at once. For example, a common nonstandard variant of the word-initial sequence ייִ is אי. The word ייִד, 'Jew' is commonly spelled איד, and ייִנגל, 'boy', is commonly spelled אינגל. However, in other cases, it is uncommon for an א to replace a י. Under this model, the substitution אי → ייִ can thus be given a relatively low cost compared to the substitution א → י. Furthermore, substitutions are directed, so a substitution in one direction can have a different cost than the reverse substitution. For example, as discussed above, it is common for nonstandard forms to have “silent” letters which do not appear in the standard forms. Thus, the deletion of these letters can carry a lower cost than their addition. Another feature of this model which distinguishes it from general edit distance is that the identity substitution itself carries a cost. The importance of this feature when modeling spelling errors may be doubtful, but when modeling the transformation of one spelling system into another it can be very useful. For example, as mentioned above, the bare grapheme פ is not a letter in the standard alphabet. The graphemes that would most commonly correspond to it in the standard orthography are פּ, פֿ, and ף. This model allows for the substitutions with any of these graphemes to carry a lower cost than the identity substitution פ → פ.

As in Brill and Moore (2000), the model must be trained to calculate the weight given to each substitution operation (an addition or subtraction can be considered to be substitution with an empty character sequence). Using a set of pairs of misspelled words (in our case, word forms in a nonstandard orthography) with their corresponding standard forms, training is accomplished by counting the correspondences in character sequences between these pairs. In order to match character sequences, the words in each pair must be aligned using an already existing edit distance measure. Brill and Moore (2000) used Levenshtein edit distance. In our case, I used the heuristically weighted edit distance described above. Below are two illustrated examples of the alignment using this edit distance.



In the alignment on the left, the middle character י is aligned with a single grapheme יִ composed of two Unicode characters. In the alignment on the right, the third character on the top, ע, is aligned with the empty set, corresponding to a deletion.

A given alignment can be thought of as a sequence of pairs of character sequences (α_i, β_i), i = 1,..., N where N is the size of the partition, and α_i or β_i might be empty. If $\alpha_i = \beta_i$, then (α_i, β_i) is added to the count. Otherwise, in order to include the context of each substitution, each pair of aligned character sequences is expanded to include the surrounding characters. This is called the context window. For instance, a context window of 2 around the middle pair in the alignment on the left above would result in the following expanded set of pairings:

י → יִ

יד → יִד

אי → ייִ

איד → ייִד

And the following pairings would be generated around the third character alignment on the right:

ע → ∅
ליע → לי
יעב → יב
עב → ב
יע → י

Brill and Moore found that their results improved with size of the context window up to a size of two characters[12], after which there was no further improvement. Therefore, for this experiment I set 2 as the maximum size of the context window. Each of the pairings thus found is then given a fractional count. For instance, if there are five substitutions, each is added with a count of 1/5. The probability of the substitution α → β is then count(α → β)/count(α). Brill and Moore (2000) could not calculate count(α) directly because they were only using a list of spelling errors, not a corpus. In the present study, however, a corpus is being used as the source of data, so count(α) can be calculated directly by counting the number of times the substring α appears in the corpus. In the case where α is the empty string, i.e, the substitution is an addition, each word in the corpus is considered to contain a number of empty substrings equal to its length + 1. Brill and Moore (2000) also tested a version of the model using positional information, calculating probabilities such as P(α → β | α, POS), where POS could be word-initial, word-final, or word-internal. However, for the present study, I did not include this as part of the model.

[12] A character here means either a letter or a letter together with its diacritic, or, in the case of ײַ, the ligature together with its diacritic. There's no simple way in Python to split a string in this manner, using either a built-in function or the regex engine in the standard library. Therefore, I used an alternative regex implementation, developed by Matthew Barnett (2014). This implementation supports the Unicode grapheme matcher \x.

An edit distance cost is calculated by taking the absolute value of the natural logarithm of the probability $P(\alpha \rightarrow \beta \mid \alpha)$. A logarithmic function is used because the overall cost is the sum of the individual costs, corresponding to the product of the probabilities, and the absolute value is taken to get nonnegative costs. In a case where $\text{count}(\alpha \rightarrow \beta) = 0$ and thus $P(\alpha \rightarrow \beta \mid \alpha) = 0$ (and so the log is not defined), or where $\text{count}(\alpha) = 0$ so the probability cannot be defined, the cost is set to a very high value, 999999. (I tried other approaches to smoothing, for instance, calculating cost as if $\text{count}(\alpha \rightarrow \beta) = .1$ instead of 0, or setting $\text{count}(\alpha)$ to .5 instead of 0, but I was not able to achieve good results in this manner.)

Given this definition of the edit distance measure, the search through the lexicon trie proceeds much as in the previous method. The main difference is that when computing a cell in the edit distance matrix column, it is not sufficient to check the (at most) three adjacent previous cells. The addition of two characters of context means that insertions, deletions, and substitutions of character sequences up to length 3 (one character plus two characters of context) must be checked. For a context window of size n, the number of edit operations that must be checked for each cell is up to $(n+2)^2 - 1$. Therefore, in our case, up to fifteen nearby cells must be checked, five times as many as in the previous model. The branches of the trie can then not be quit as soon as in the search without a context window, since a substitution of two characters could theoretically be less costly than with one just character. All this means that this search is significantly slower than for the last method. Brill and Moore (2000) were able to achieve speeds of approximately 50ms for a word, although the implementation for the current study is significantly slower.

An example of the output from the Brill model is in Table 4. In this case, the correct answer is the third ranked[13].

Table 4: Sample Output from Brill Model

Input	Suggestion	Edit Distance
וויא	וויאָ	4.32
	וויְאַ	5.27
	ווי	**6.02**
	וויי	8.88
	ווײַך	9.27

6.4 Evaluation

Two types of testing were performed on the algorithms to evaluate their efficacy. The first is a measure of those proportion of words in a test sample the algorithms are able to correctly normalize. Second is a test of information retrieval, as this task is one of the main motivations for developing automatic normalization methods. The two evaluation methods had consistent results, in that the same methods proved themselves to be the most effective in both cases.

The general ranking of methods found was:

Brill Model > Heuristic Method > Aspell > Rule-based Algorithm

[13] This is an example where a language model might have helped rank the correct answer more highly. The two highest ranked answers, while closer in edit distance to the input, are very low-frequency words, while the third-ranked answer is a very high-frequency word.

6.4.1 Accuracy of Normalization

As mentioned above, 68% of the corpus tokens and 44% of the types were already found in the lexicon. Since the task was to normalize nonstandard forms, these items were excluded from this test, leaving 8,981 tokens, or about 4,000 types.

The tokens were run through each of the algorithms, and the output forms saved. For the Brill algorithm, which required training data, the results reported here are based on 5-fold cross validation. The data were partitioned by putting every fifth word into the same fold (after concatenating all the corpus documents). In each round, four folds (80%) of the data were used for training and the remaining fold (20%) for testing. One disadvantage of this method is that data from a document is used to test data from the same document. Due to “burstiness”, the propensity of words, even low-frequency ones, to appear more than once in a document if they appear at all, this could result in overfitting the data. However, since the documents represent a wide range of nonstandard orthographic systems, and there are only 16 documents, this approach was considered preferable to partitioning the data by document. For instance, there was only one document with Soviet-style spelling. This document would have provided poor training data for the other documents, and vice versa.

As none of the algorithms make use of context, each type only had to be computed once — for the Brill algorithm, however, once for each fold. Since each fold has weights learned from different training sets, not each type produces the same results in different folds. Furthermore, while the normalization algorithms are deterministic with regards to type, the gold standard upon which they are judged, human standardization, is not, and so the same answer for a type might sometimes be correct and sometimes incorrect. For both of these reasons, any statistics involving proportion of correct answers cannot be reduced to types, and consequently, all results reported here are in tokens.

For the three algorithms which return a ranked list of normalizations (that is, all except the rule-based algorithm), results were counted where the correct answer was top-ranked, or among the top two- or three-ranked, or among any of the returned forms. (The number of results returned was limited to a maximum of ten.) For those algorithms, I also calculated the mean reciprocal rank (MRR), a measure of how highly ranked the correct answer is. For instance, if the correct answer is the first ranked, reciprocal rank is 1; if it is second, reciprocal rank is 0.5, and so on. If there is no correct answer, the reciprocal rank is set to 0. MRR is the average of these values over all tokens.

Of the 8,981 tokens not found in the lexicon, 7,458 were nonstandard forms of items in the lexicon, and 1,523 were not covered by the lexicon at all (among these words are proper nouns, compound words, and foreign words). Three of the four algorithms derive their answers from a search through the lexicon. Thus, for those nonstandard tokens without corresponding items in the lexicon, these algorithms have no chance of returning the correct normalization. This means that over this data set there is a theoretical maximum normalization accuracy of 83% for this type of algorithm. (The determinism of the algorithms lowers somewhat further the theoretical maximum.) The rule-based algorithm does not in principle have this limitation, but in practice it was only able to correctly normalize 80 (5.25%) of the 1,523 tokens not covered by the lexicon.

Table 5 reproduces the results. The first percentage is out of all tokens, and the second, in parentheses, is out of those tokens represented by standard forms in the lexicon.

Table 5: Proportion correctly normalized

Method	1-Best	2-Best	3-Best	All-Best	MRR
Brill	**77.42% (93.23%)**	**80.54% (96.98%)**	**81.18% (97.76%)**	**81.52% (98.16%)**	**0.793**
Heuristic[14]	63.16% (76.05%)	69.44% (83.61%)	70.69% (85.13%)	73.84% (88.92%)	0.682
Rule-Based	19.27% (22.14%)	N/A	N/A	N/A	N/A
Aspell	52.08% (62.71%)	57.91% (69.74%)	61.22% (73.72%)	68.15% (82.07%)	0.575

The Brill method was in all cases found to be the highest performing. In fact, its performance at the 1-Best level outranked all the others even at the All-Best level. The differences between the methods for the first four columns of Table 5 were tested for statistical significance using McNemar's test between each pair of algorithms. Differences were found to be significant with p < .01%.

The accuracy rate produced here by the rule-based system differs markedly from that found in Kirjanov, et al. (2014). Time did not permit an investigation into the cause of this discrepancy.

In a real-world scenario, the lexicon would not contain the normalized forms of all words contained in a text. Hence, the rates in Table 5 are more indicative of how the methods will perform in the real world. However, to see how well the algorithms perform when they are theoretically able to

[14] Midway through preparing this study, the algorithm used to normalize the words with respect to Unicode was changed to include the decomposition (U+05D9, U+05D9, U+05B7) → (U+05F2, U+05B7) *(yud, yud, pasekh → tsvey-yudn, pasekh*), as described above in the section introducing the lexicon. Prior to this change, the manually-weighted distance method performed better, almost as well as the Brill method (although the difference between them was still statistically significant). The drop in performance is because the method no longer assigned a low cost to an edit transforming ײ to ײַ, although this is a very common variation in nonstandard orthographies. Previously this would be calculated as a change in diacritic only, or a cost of 0.2. With the change, however, the algorithm could not find a distance lower than substituting a full two characters (the two individual י characters with a single character, the ligature ײ, plus diacritic). This resulted in a much higher cost of 3.2. A rule should be added to manually decrease this cost, but since only operations with single characters are allowed in the model, no clear solution presents itself. The Brill method, on the other hand, which already includes arbitrary multi-character substitutions and learns its weights automatically based on the given word encodings, showed very little change in performance after the change. This is a demonstration of the robustness of the Brill method in contrast with one using manually defined weights of single character edits.

reach all the words, the same tests were run after adding to the lexicon the 940 manually standardized forms which were previously missing. The results are listed in Table 6.

Table 6: Proportion correctly normalized when
all manually standardized forms are added to the lexicon

Method	1-Best	2-Best	3-Best	All-Best	MRR
Brill	**92.86%**	**96.63%**	**97.44%**	**97.87%**	**0.951**
Heuristic	76.88%	85.06%	86.87%	89.37%	0.825
Rule-Based[15]	21.74%	N/A	N/A	N/A	N/A
Aspell	61.83%	69.92%	72.44%	81.64%	0.686

Unsurprisingly, the results are comparable to the results in parentheses in Table 5, where only the lexical items which were already in the lexicon were considered. Once again, the Brill method performs higher even at 1-Best than the others at All-Best.

6.4.2 Information Retrieval

The information retrieval task was designed to see how automatic normalization might be used to improve the recall of a search query. Assume that one has a collection of Yiddish documents to search through. A simple search function will return only those results which exactly match the query. If, however, during indexing of the documents, each nonstandard word form is normalized, and the normalized forms are linked to the nonstandard word forms, then a search for a standard form could retrieve not only those words which match the form, but also those words which are linked to that form, that is, that have been normalized to that form.

[15] One might wonder why the performance of the rule-based method would be different here than in Table 5 when the algorithm does not make use of the lexicon. Recall that percentages are taken only over those words not found in the lexicon. When words are added to the lexicon, this set shrinks, and so, while the output of the rule-based algorithm does not change, its performance rate shifts somewhat due to the different test set.

The corpus of 16 Yiddish texts was used as the document collection to be searched. The set of search queries to test was derived from the manually normalized forms of all the words in the corpus. That is, the set of search queries is exactly that set of standard forms that one would expect, ideally, to return at least one result. A result for a given search query is considered *relevant* if its manually normalized form matches the search query. For the three algorithms that return ranked results, scores were calculated judging results as relevant if the search query was, respectively, among the one, two, or three highest-ranked results. A search returning a form only if it exactly matched the query was used as a baseline.

As a second baseline, another search function was added. For a given query, this search function will find all forms which are equal to the query modulo diacritics, namely, if removing all diacritics from the query and the given form results in the same character sequence. This baseline was added because it is a fairly simple way to increase recall and is similar, if not identical, to implementations in use today, such as in Google's search engine. If one searches in Google for פֿון, for example, results with פון will also appear.

For each form, we calculated precision, recall, and F-score. Precision is the ratio of relevant results to all results. Recall is the ratio of relevant results to all relevant items in the corpus. F-score is a combination of these two measures in one value, calculated as the harmonic mean of precision and recall. These scores were then averaged over all types. The average over tokens was also calculated, weighting each type according to its frequency in the corpus.

The results are listed in Table 7.

Table 7: Precision, Recall and F-score (by type)

Method	**Best-1**			**Best-2**			**Best-3**		
	Pres.	**Rec.**	**F**	**Pres.**	**Rec.**	**F**	**Pres.**	**Rec.**	**F**
Baseline	**99.6%**	59.10%	60.80%	**99.6%**	59.1%	60.8%	**99.6%**	59.1%	60.8%
Ignore Diacritics	98.5%	81.6%	81.8%	98.5%	81.6%	81.8%	98.5%	81.6%	81.8%
Brill	98.6%	**90.8%**	**90.3%**	97.2%	**92.1%**	**90.5%**	96.8%	**92.4%**	**90.5%**
Heuristic	97.7%	86.8%	85.8%	94.0%	88.8%	85.2%	90.8%	89.4%	83.6%
Rule-Based	99.4%	67.6%	68.6%	99.4%	67.6%	68.6%	99.4%	67.6%	68.6%
Aspell	97.3%	81.5%	80.8%	95.0%	84.5%	82.2%	93.3%	85.6%	82.1%

The baseline, as expected, has a low recall value, corresponding roughly to the proportion of standard forms in the corpus, and the highest precision at 99.6%. One might wonder why the precision is not 100%. This is because some words appear in the corpus in a nonstandard form which happens to be the standard form of another word. For example, the word נײַן, 'nine', might appear in the nonstandard form נײן, which happens to be the standard form of another word, 'no'. Only a human or a program which took into account context would be able to recognize that such a word was not appearing in standard form. The baseline also had the lowest F-score.

The second baseline, which ignored diacritics in its search, performed better. With only a 1% absolute drop in precision vis-à-vis the first baseline, it achieved a 22.5% (absolute) higher recall rate. In fact, its F-score was higher than that of the rule-based method, and (slightly) higher than that of the Aspell method. This indicates that search results can be greatly improved through a method that is simple to implement, requires little computational power, and needs no other resources such as a lexicon.

The baseline highest value and the rule-based method slightly outperformed the Brill method in precision, but the Brill method outperformed them by far in recall, and so scored highest on the F-score in all cases. It performed slightly better at Best-2 than at Best-1, but the increase in recall at Best-3 was not enough to offset the decrease in precision.

The results when values are calculated by token instead of by type are listed in Table 8.

Table 8: Precision, Recall and F-score (by token)

Method	**Best-1**			**Best-2**			**Best-3**		
	Pres.	**Rec.**	**F**	**Pres.**	**Rec.**	**F**	**Pres.**	**Rec.**	**F**
Baseline	**99.6%**	70.6%	75.8%	**99.6%**	70.6%	75.8%	**99.6%**	70.6%	75.8%
Ignore Diacritics	97.9%	88.6%	89.9%	97.9%	88.6%	89.9%	97.9%	88.6%	89.9%
Brill	99.1%	**95.1%**	**95.4%**	98.0%	**96.0%**	**95.4%**	97.5%	**96.2%**	**95.3%**
Heuristic	98.4%	91.3%	91.5%	95.5%	92.6%	90.8%	93.2%	93.1%	89.7%
Rule-Based	99.5%	76.7%	80.3%	99.5%	76.7%	80.3%	99.5%	76.7%	80.3%
Aspell	97.9%	87.1%	88.4%	96.2%	88.9%	88.8%	94.8%	89.9%	88.7%

The performance hierarchy is once again Brill > Heuristic > Ignore Diacritics > Aspell > Rule-Based > Baseline, with Brill's best performance at Best-1 and Best-2.

It is worth noting that even the baseline has a much higher recall than many real-world applications encountered today. This is because even for the baseline all the queries and corpus words were Unicode-normalized up to graphemic identity, as detailed above. However, I was not able to test such a “real world” model without Unicode normalization to calculate the performance difference, because I had no model on which to base the frequency of different Unicode representations in the query or corpus items.

7 Future Work

There are numerous directions in which to expand on the present research.

Several parameters that Brill and Moore (2000) tested have not been implemented here. For instance, the context window size was set at a fixed value of 2 instead of testing different sizes. One could test to see if a different size is more optimal for the case of Yiddish. Brill and Moore (2000) also tested a version of their model using positional information, calculating probabilities such as $P(\alpha \rightarrow \beta \mid \alpha, POS)$, where POS could be word-initial, word-final, or word-internal. They found that this improved performance in conjunction with a context window. However the present study does not include this as part of the model. Brill and Moore (2000) found that results were also improved by including a language model (as opposed to a null model, which assumes that all words are equally likely). However, prepared corpora suitable for creating a language model do not exist for Yiddish. If a language model could be compiled based on Yiddish linguistic data, one could test to see if results could be further improved.

The types of orthographic variability encountered differ systematically based on the origin of a document. This is because there were different spelling standards in vogue in different times and places. A document from the Soviet Union in the 1930s, one from Poland in the 1890s, New York in the 1920s, or a Hasidic publication of today are likely to differ in systematic ways from each other and from the standard orthography. I considered creating a typology of orthographies and weighting the edit costs separately for different spelling styles, as well as providing a method for the program to decide which type of orthography was being used in a given document. For example, it was not difficult to devise a program that could determine, on the basis of letter frequency, if a document is in the Soviet orthography. (This is one of the simplest cases, as the Soviet orthography introduced specific radical spelling changes.) However, the results of this pilot test were not evaluated for statistical significance. Unfortunately, the size and range of our corpus, with only 16 documents, was insufficient to develop or test such a typology. In the future, with the development of a corpus comprised of more documents from a representative sample of places and times, such a test could be carried out.

Finally, the performance of the Brill model has been proven in the test data, but it remains to apply this model in a real-world application. The most immediate application would be any search function over a collection of Yiddish text which includes forms outside of the standard orthography. In this case, one should also not assume that the input queries are in the standard orthography. A query could be first normalized using a normalization model, and then the search performed as in the model tested in this study. I did not choose to evaluate the performance of such a model, as I did not have access to a model of nonstandard Yiddish that I thought could represent the types of queries that might be input. However data could be collected from a real-world search function to help create such a model. Additionally, research can continue to explore how normalization can help increase the effectiveness of other NLP tools such as OCR and part-of-speech tagging. (Many such NLP tools have yet to be created for Yiddish.)

8 Conclusion

Until quite recently, Yiddish was a major European language. Prior to World War II, it was the fourth most widely spoken Germanic language in the world (after English, German and Dutch), and once had the second largest geographic expanse (after Russian) of any European linguistic/cultural group (Jacobs 2005). Unlike many other resource-poor languages, Yiddish also has a large and varied literature spanning centuries. So there is much to be gained by bringing these cultural riches and linguistic data into the digital domain. However, due to the language's disparate orthographies, some form of normalization is required in order to make such corpora uniformly accessible. Several techniques in orthographic normalization have been explored in this paper. The best performing one, using multi-character weighted edit distance, was more effective on the test data than any other method found to be used in a real world Yiddish application today. But even the simplest of methods, factoring out different Unicode encodings of the same graphemic sequence, could greatly improve on the search functions found in some current Yiddish applications. It is to be hoped that the insights discussed in this paper can be put to practical use by researchers of the Yiddish language as well as in the broader Yiddish linguistic community of today.

9 Appendix

These are the titles from which the corpus documents were excerpted.

1. Amol iz Geven: Zikhroynes fun dem Yidishn Lebn in Lite. 1926. Isidore Kopeloff. New York: M.N. Mayzel. https://archive.org/details/nybc209754

2. An Ski - Gezamlte Shriftn Vol 1. 1920. Sh. An-Ski. Warsaw, New York: Farlag An-Ski. https://archive.org/details/nybc205846

3. Azoy Zaynen Mir Geshtorbn. 1949. Levi Shalit. Munich : Aroysgegebn fun Farband fun Litvishe Yidn in Daytshland. https://archive.org/details/nybc212761

4. Birebidzhaner. 1934. Dovid Bergelson. Moscow: Emes. https://archive.org/details/nybc200561

5. Der inyen numer 5390 (fun di K.G.B. arkhivn). 1992. Boris Sandler. Jerusalem: Yerusholaymer Almanakh. https://archive.org/details/nybc211782

6. Di Gayster. 1914. Henrik Ibsen. New York: Literarisher ferlag. https://archive.org/details/nybc205556

7. Fishke der Krumer. 1928. Mendele Moykher Sforim. Warsaw: Mendele. https://archive.org/details/nybc202698

8. Geshikhte fun der Yidisher Literatur in Amerike. 1943. Elias Shulman. New York: I. V. Biderman. https://archive.org/details/nybc210318

9. Ksovim fun geto. 1961. Emanuel Ringelblum. Warsaw: Yidish-Bukh. https://archive.org/details/nybc210248

10. Khsidishe Mayselekh. 1916. Isaac Leib Peretz. New York: Hebrew Publishing Co. https://archive.org/details/nybc204288

11. Mayn Vaybs Spazmes. 1892. Nokhem Meyer Shaykevitsh. Vilna: Romm Widow and Brothers Printing House. https://archive.org/details/nybc202417

12. Megile Lider. 1936. Itsik Manger. Warsaw: Aleynenyu. https://archive.org/details/nybc201284

13. Motl Peysi dem Khazns. 1917. Sholem Aleichem. New York: Sholem Aleykhem Folksfond. https://archive.org/details/nybc200089

14. Yidishe Folks-Entsiklopedye far Yidishe religye, geshikhte, filozofye... un andere inyonim (Vol 1). 1949. Symcha Pietruszka. New York: Gilad. https://archive.org/details/nybc200232

15. Yitskhok Leybush Perets: Artiklen un Redes. 1951. Chaim Zhitlowsky. New York: Ikuf. https://archive.org/details/nybc202941

16. Zikhroynes fun a Yidisher Revolutsionerin. 1954. Puah Rakovska. Buenos Aires: Tsentral farband fun Poylishe Yidn in Argentine. https://archive.org/details/nybc202581

10 References

Adesam, Yvonne, Malin Ahlberg, and Gerlof Bouma. 2012. bokstaffua, bokstaffwa, bokstafwa, bokstaua, bokstawa... Towards lexical link-up for a corpus of Old Swedish. In Proceedings of the 11th Conference on Natural Language Processing (KONVENS 2012), LThist 2012 workshop, pages 365–369, Vienna, Austria.

Atkinson, Kevin. 2004. GNU Aspell Spell Checker. http://aspell.net/, Accessed on January 19, 2015

Barnett, Matthew. 2014. mrab-regex-hg: New implementation of Python regex. https://code.google.com/p/mrab-regex-hg/. Accessed January 23, 2015.

Baron, Alistair and Paul Rayson. 2008. VARD2: A tool for dealing with spelling variation in historical corpora. *Postgraduate conference in corpus linguistics*.

Baron, Alistair, Paul Rayson, and Dawn Archer. 2011. Quantifying Early Modern English spelling variation: change over time and genre. In Conference on New Methods in Historical Corpora.

Bird, Steven, Edward Loper and Ewan Klein. 2009. Natural Language Processing with Python. O'Reilly Media Inc.

Black, Paul E., "trie", in Dictionary of Algorithms and Data Structures [online], Vreda Pieterse and Paul E. Black, eds. 22 February 2011. (accessed Jan. 19, 2015) Available from: http://www.nist.gov/dads/HTML/trie.html

Bollmann, Marcel. 2012. (Semi-)Automatic Normalization of Historical Texts using Distance Measures and the Norma tool. In: Proceedings of the Second Workshop on Annotation of Corpora for Research in the Humanities (ACRH-2), pp. 3–14. Lisbon, Portugal.

Bollmann, Marcel. 2013. Automatic Normalization for Linguistic Annotation of Historical Language Data. Bochumer Linguistische Arbeitsberichte: 13.

Brill, Eric and Robert C. Moore. 2000. An improved error model for noisy channel spelling correction. *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics,

Ernst-Gerlach, Andrea, and Norbert Fuhr. 2006. Generating search term variants for text collections with historic spellings. *Advances in Information Retrieval*. Springer Berlin Heidelberg, 49-60.

Hauser, Andreas W. and Klaus U. Schulz. 2007. Unsupervised learning of edit distance weights for retrieving historical spelling variations. *Proceedings of the First Workshop on Finite-State Techniques and Approximate Search*.

Herzog, Marvin, Uriel Weinreich, and Vera Baviskar. 1992. The language and culture atlas of Ashkenazic Jewry, Vol. 1. Tübingen: Max Niemeyer Verlag.

Jacobs, Neil G. 2005. Yiddish: A linguistic introduction. Cambridge University Press.

Kempken, Sebastian. 2005. Bewertung historischer und regionaler Schreibvarianten mit Hilfe von Abstandsmaßen.

Kempken, Sebastian, Wolfram Luther, and Thomas Pilz. 2006. Comparison of distance measures for historical spelling variants. *Artificial Intelligence in Theory and Practice*. Springer US. 295-304.

Kirjanov, Denis, Boris V. Orehov, and Tanya A. Panova. 2014. Variativnost' orfografij v idishe i problema ikh avtomaticheskoj transliteracii. http://www.dialog-21.ru/digests/dialog2014/materials/pdf/KirjanovDP.pdf (accessed January 26, 2015)

Oravecz, Csaba, Bálint Sass, and Eszter Simon. 2010. Semi-automatic normalization of Old Hungarian codices. In Proceedings of the ECAI Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities (LaTeCH) (pp. 55-59).

Pettersson, Eva, Beáta Megyesi, and Joakim Nivre. 2012. Parsing the past: identification of verb constructions in historical text. In Proceedings of the 6th Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities (pp. 65-74). Association for Computational Linguistics.

Pilz, Thomas and Wolfram Luther. 2005. Regelbasierte Suche in Textdatenbanken mit nichtstandardisierter Rechtschreibung.

Pilz, Thomas, Andrea Ernst-Gerlach, Sebastian Kempken, Paul Rayson, and Dawn Archer. 2008. The identification of spelling variants in English and German historical texts: Manual or automatic?. *Literary and Linguistic Computing* 23.1: 65-72.

Rayson, Paul, Dawn Archer, and Nicholas Smith. 2005. VARD versus WORD: A comparison of the UCREL variant detector and modern spellcheckers on English historical corpora.

Sánchez-Martínez, Felipe, Isabel Martínez-Sempere, Xavier Ivars-Ribes, and Rafael C. Carrasco. 2013. An open diachronic corpus of historical Spanish: annotation criteria and automatic modernisation of spelling. arXiv:1306.3692v1.

Santorini, Beatrice. 1997/2008. The Penn Yiddish Corpus. University of Pennsylvania.
For details, contact:beatrice@babel.ling.upenn.edu.

Schaechter, Mordkhe. 1999. *Der eynheytlekher yidisher oysleyg*. New York: Yivo and Yiddish Language Resource Center of the League for Yiddish.

Scherrer, Yves and Tomaž Erjavec. 2013. Modernizing historical Slovene words with character-based SMT. In Proceedings of the 4th Biennial Workshop on Balto-Slavic Natural Language Processing, Sofia, Bulgaria.

Schneider, Peter. 2001. Computer assisted spelling normalization of 18th century English. *Language and Computers* 36.1: 199-211.

Strunk, Jan. 2003. Information retrieval for languages that lack a fixed orthography. *Linguistics Department, Stanford University, California*.

Urieli, Assaf and Marianne Vergez-Couret. 2013. Jochre, océrisation par apprentissage automatique: étude comparée sur le yiddish et l'occitan. *Actes de TALARE 2013: Traitement automatique des langues régionales de France et d'Europe*.

van Halteren, Hans and Margit Rem. 2013. Dealing with orthographic variation in a tagger-lemmatizer for fourteenth century dutch charters. Language Resources and Evaluation. doi: 10.1007/s10579-013-9236-1.

Weinreich, Max. 1939. *Di Shvartse Pintelekh.* YIVO.