# Unique Challenges with Hebrew Language NLP Projects

Thesis Submitted in Partial Fulfillment
of the Requirements
of the Jay and Jeanie Schottenstein Honors Program

Yeshiva College Yeshiva University September 2023

Jason Caplan
Mentor: Professor Dave Feltenberger, Computer Science

בס״ד

# Unique Challenges with Hebrew Language NLP Projects

Jason Caplan

Last Edited: Sep 27, 2023 Pages: 14

# NLP Background

## NLP in General

Natural Language Processing, or NLP, is a field of study that utilizes computers to accomplish various tasks that relate to natural, or human-spoken, language. It is a subset of the broader field of Machine Learning, which generally refers to the application of computer algorithms and modeling to solve various complex tasks.

Some of the types of NLP problems that are classically studied are listed below:

### Classes of Problems

#### Classification

Classification is a classic type of machine learning problem, and its prevalence in NLP is no exception. A classification model is a model that takes data as its input, and outputs a classification label. For example, one could have a model that takes an image of an animal as its input data, and outputs a classification prediction label, e.g. “cat” or “dog”.

Similarly in NLP, a classification model will involve an input of text, and an output of a classification prediction. One common example is sentiment analysis, where an NLP model is used to determine the overall sentiment of a piece of text. For example, the input to the model is a restaurant review on Yelp, and the output prediction is whether the review was a positive review of the restaurant or a negative review.

### Translation

One of the most well-known applications of NLP is for translation between different languages. Google Translate is one of the most widely used translation tools, and is built upon NLP technologies and methodologies.[1] Many approaches exist to achieve accurate translation from language to language. Most involve feeding the input language text into a "model" that outputs words of the other language.

### Generative Modeling

One of the hottest, cutting-edge fields within NLP today is that of generative modeling. Broadly speaking, this involves generating new text based on input text.

This can take many forms. One example is masked language modeling. This means that, given a portion of text with a small word or section missing, an NLP model can predict the most likely way to fill in the missing part. For instance, given the sentence "I have <MASK> older brothers, Tom and Joe," the NLP model would have enough of a semantic understanding of the English language to predict that the missing part, i.e. the mask, should be the word "two."

Another example of a generative modeling problem is text-generation based on a prompt. The most well-known example today is ChatGPT, which is OpenAI's model that can generate new text given a promt. This works by predicting the most likely words to follow the "chat" that has taken place thusfar.

[1] See https://en.wikipedia.org/wiki/Google_Translate#Translation_methodology

## Modeling Approaches

On a high-level, these are the most commonly used approaches for building models to solve the classes of problems listed above:

### Naive Models

Some of the more basic approaches to classification involve simple computations like word counts. In the Yelp review sentiment analysis example, one could calculate how positive or negative every word is by scanning "training examples," i.e. reviews that have a known sentiment (for Yelp, this would mean negative for 1-2 stars, and positive for 4-5 stars), and seeing how common each word is in positive reviews vs. negative reviews. This would give a sentiment score for each word, and a naive model could predict how likely a new review is to be positive or negative by adding the scores of all the words in the review and seeing if the aggregate score is higher for positive or negative sentiment.

However, the naivete in this approach will limit how good the model will be, since this approach does not account for different variations of similar words, where words appear in the sentence, and general semantic understanding. It also assumes that every word that is present in the new review will have been present in the exact same spelling in training examples so that the percentages can be calculated, but this is often not true.

### More Advanced Model Pre-requisites

More advanced models require a mapping from natural language to numerical representations, which can be ingested as input by complex mathematical models. There are two main approaches to map words to numbers:

The first is called “Bag-of-words,” which also based on word counts, and works with the following process (oversimplified here): each word in the vocabulary (one can think of this as the English dictionary) is given an index in a list (e.g., “the” corresponds to position 1, “and” to position 2, “language” to position 3, etc), and the values of the list are the number of times each word appears in the text (e.g., if the word “the” appears 10 times, “and” appears 15 times, and “language” appears twice, the list will be [10, 15, 2, ...]). Essentially, this is a word count list that captures how often each word in the vocabulary appears in the text. Through this mapping, one can transform any piece of text into a list of numbers, which can be ingested into a mathematical model.

The second is more complex, and is called “Word-embeddings.” Here, instead of having a list with an element for the word count of each word in the vocabulary, a fixed size list is used – also called a “vector” – to represent each word. For example, in the case where the fixed size is 2, the list [28, 96] may represent “the”, and [82, 71] may represent “and.” The intention is that words that have similar meaning or are used in similar places within a sentence will be “near” each other - where proximity is measured as the distance between the vectors (think of the distance between the points [x=28,y=96] and [x=82,y=71] on a regular X/Y graph). Usually, vectors of much larger length are used, on the order of tens or hundreds of numbers long (where distance between vectors is defined by something like Euclidean Distance). This type of representation is not meant to be understandable just by looking at it, and requires training on a huge amount of unlabeled text.

## Neural Networks

There are many types of neural network models, but most involve a process similar to this: programmatically mapping the input words to a numerical representation (using one of the methods described above), feeding these numbers through a dynamic series of mathematical equations, and finally mapping the final numbers to the desired output format. The dynamism of these mathematical equations is updated iteratively by feeding the model input of "training examples," where the desired output is known, and adjusting the parameters of the equations in light of the known examples.

For example, in language-to-language translation, one would take a text that has already been translated by a human from language A to laguage B, feed the language A version into the model, and compare the output with the language B version, adjusting the parameters of the equations accordingly. Then, the model will be better at predicting unseen examples, like a brand new text to translate.

Often, the larger the network – i.e., the more adjustable parameters that exist – the more complex the underlying model will be in terms of its ability to understand relationships and patterns. This is ultimately supposed to be similar to the human brain, which contains millions of neurons that are interconnected in a complex network and can capture abstract ideas in their vast interconnectedness. However, a common problem in Machine Learning in general is that a network will get so complex that it doesn't really understand the problem space, but rather has simply memorized the training examples, and cannot extrapolate to new unseen data. This is similar to a person who, instead of understanding addition, memorizes thousands of addition problems - this takes a lot of brain power, and they may be able to answer problems they have seen before, but they won't be

able to answer a new unseen addition problem. The Machine Learning term for this phenomenon is "over-fitting."

## Transfer Learning

Today, one of the most common techniques to jump-start a difficult NLP problem is called "transfer learning." This involved utilizing a pre-existing model that has been trained on a problem-space similar to the desired problem, and tweaking it to fit the given use case. An analogy is as follows: a person who is already multi-ligual may more easily learn a new language, because they already know how to break down languages in a certain abstract way (e.g., syntax, grammar, intonation), and all they are missing is the specific application to the new language's vocabulary and idiosyncrasies. Similarly, a neural network that has already learned how to do one task could be tweaked to do another, without having to re-train from scratch.

To take a concrete example: say there is a neural network that has been trained to take a tweet as input text, and output whether the author of the text is more likely to have been a republican or democrat. It is likely that the way the model has learned these patterns is actually in a compartmentalized manner - the first part of the network likely breaks down the text into subject and predicate, further layers parse the parts of speech, subsequent layers extract meaning, and finally the output layer understands political sentiment. Using this architecture, one could theoretically remove the final layer, and retrain a new "head" of the model to determine the gender of the author - since all the groundwork of more abstract language understanding is already handled in the earlier parts of the

network. This is an oversimplificaiton, but should capture the core concept of what happens in transfer learning, and why it is helpful in training models more quickly by reusing previous work.

## Anglo-Centricity of NLP Today

As with most areas in the field of computer science, NLP is highly anglo-centric. Just like almost every modern programming language is written in a form of English, so too almost every NLP resource is made for the English language. For example, the most common toolkits and programming libraries that automate much of the NLP experience are built primarily with English in mind, and range from accommodating other languages with some necessary modifications, to not supporting other languages at all. Furthermore, resources that help jumpstart projects and diagnose issues and bugs in programs are almost always focused on English applications, and therefore getting off the ground or debugging code meant for a non-English language requires extra effort to extrapolate to this other language's use case. Below are details of specific challenges that are latent in Hebrew language NLP projects.

# Unique Challenges with Hebrew

## Differentiating Characteristics of the Hebrew Language

To understand the unique challenges of Hebrew NLP projects, one must first understand the differentiating characteristics of the language in general.

## Vowelization

Hebrew does not represent vowels with letters as does English. Rather, like other semitic languages, it uses a system of symbols like dots and lines above, below, or within letters to signify vowelization. However, in most writing online and otherwise, the vowelization is completely omitted, and the reader must intuit the correct vowelization based on context.

This causes many ambiguities in language, since the same letters can represent completely different words if different vowelization is applied. Take the example of the word with the three letters ע ("ayin"), צ ("tzadi"), ם ("mem"), altogether עצם. If vowelized as עֶצֶם ("etzem"), it means "bone", but if vowelized עָצַם ("otzam") it means "closed."

## Alphabet Size and Root Size

An additional factor that leads to ambiguity in Hebrew is the limited alphabet size and small root size. There are only 22 letters, compared with the 26-letter English alphabet. Furthermore, it is a root-based language, and nearly all word roots are 3 letters. This means that the absolute maximum number of unique root permutations is P(22,3)=9240, and even this is a gross overestimate because of anti-patterns that are avoided (like having the same letter multiple times in a row, and having certain letters adjacent to each other). These factors mean that there is a lot of root-overloading (as in the עצם example above), where the only differentiator is the often omitted vowelization.

## Attached Qualifiers

Another cause for ambiguity in Hebrew is the way it uses qualifying words like "the," "and," and "that." In Hebrew, they are almost always attached to the following word, whereas English would

keep them as separate words. For example, “and” is not a standalone word in Hebrew, rather the letter “ו” (“vav”) is appended to the following word to indicate “and.” Similarly, “the” is replaced with the letter “ה” (“hey”) appended to the following word. However, these letters can also be part of the root word, and not an appended qualifying word.

Take the example of the word spelled with the three letters כ (“kaf”), ל (“lamed”), and ב (“bet”). If the כ (“kaf”) is an attached qualifier, it means “like,” and the remaining letters  (“lamed”), and ב (“bet”) spell the word לב “lev” meaning “heart,” so altogether the word translates to “like a heart.” However, if the כ (“kaf”) is part of the root word, it spells כלב (“kelev”), which means “dog.” Obviously, these convey vastly different meanings, but the spelling is exactly the same.

## Preprocessing

As part of the process of preparing text for being fed into an NLP model, there are often steps taken to filter, clean, and sometimes modify the text, in order to create a more optimal format for the algorithm to learn. These steps are called text preprocessing.

### Stopwords

One common preprocessing step is stopword removal. Stopwords are words that appear often in text but do not have a significant affect on the meaning of the text. In English, this includes words like “of,” “which,” “and,” and many others. Including these words can mislead many algorithms that rely on word occurrence to determine importance. For example, a model might notice that the word “a” is commonly associated with negative sentiment, but this is obviously a false correlation because “a” doesn’t indicate anything negative, it just appears often in every text, including negative texts.

The most common way to remove stopwords is to use a list of stopwords compiled by experts, that has been proven by researchers and NLP programmers to be helpful in improving the performance of NLP models. Stopword lists exist for many common languages, but lists for Hebrew are more rare.[2]

Another issue with using stopword removal for Hebrew NLP projects is inherent to the language. As explained above, there is a lot of word-overloading in Hebrew. Therefore, removing a certain word that would seemingly be an unimportant stopword may really end up removing a significant word that has the same (unvowelized) spelling. For example, the word עד means "until" which would ostensibly be a good candidate for stopword removal. However, since the same word can also mean "a witness," removing it may have negative consequences in certain contexts, e.g. a classification task that attempts to predict whether a document is a legal document or not.

From experience in our YU CS Capstone project, we found that stopword removal cannot be definitely dubbed helpful or harmful to model performance - in some cases it improved performance, and in others it detracted from performance, with no clear pattern therein.

## Stemming

Stemming involves the removal of the final few letters of words in order to arrive at base-words, so that a model will associate words with similar meaning with each other, and so the vocabulary is not as strict. For example, the words "say," "says" and "saying" would all be stemmed to become "say," which would help the model be more flexible in identifying the idea of these words. But this is often

[2] There are a few that exist (see here and here), but are not as tested empirically as their English coutnerparts, and they do not help as much for Hebrew as we explain in the next paragraph

naive, and leads to model mistakes. For example, the word “faster” may be stemmed to be “fast,” and misunderstood to reference the verb meaning refraining from eating, whereas the word “faster” is probably the synonym of “quicker.”

For Hebrew, stemming is theoretically even more important, since more qualifiers are appended as suffixes than in Enlgish. For example, to add a qualifier in English, one simply adds a preceding word, like “we walked” or “I traveled.” But in Hebrew, the qualifier is appended to the word itself, such that the phrases above each truncate into individual words made up of the 3-letter-root composed with the qualifyer (הלכנו and נסעתי respectively).

### Lemmatization

Lemmatization is the process of converting words to their roots, so that a model will treat different forms of the same word similarly, and thus gain a better understanding of the natural language.

This is actually the one area where Hebrew would benefit even more than English from preprocessing, since the language is so heavily based on word roots. However, since the Hebrew NLP community is so small, the robust tooling to automate and streamline the lemmatization process that exists for English does not exist for Hebrew, and therefore the benefits from the sub-par lemmatization available for Hebrew NLP often do not justify their usage.

## Availability of Training Data

Since much of the process of training NLP models requires training data (as described above), the accuracy of models is often associated with the amount of training data available - the more

training data, the more examples the model can see and gain understanding from, and the more the model can extrapolate to new unseen examples. And of course, the main resource for training data is the internet, with somewhere around 100 trillion words. With respect to the English language, the internet is a vast resource, since over 50% of it is in this language. However, when it comes to Hebrew, the internet is far less helpful, with less than 0.5% of the internet written in Hebrew.[3] Additionally, a very good resource for high-quality training data is wikipedia.com (indeed this was used as the plurality of training data for OpenAI's GPT models), but Hebrew wikipedia has less than 350k articles, compared to over 6.6 million English articles.[4]

## Availability of Pre-trained Models

As mentioned above, transfer learning can be tremendously useful in getting started with NLP projects. However, this is only feasible if pre-trained models exist that were trained on a similar problem-space and can extrapolate to the desired problem. Fortunately, the NLP community has forums to share these pre-trained models, like HuggingFace, which houses over 300,000 models. However, as of the publication of this paper, less than 400 have support for Hebrew, and only one (BEREL, mentioned below) seems to be catered toward the problem-space of Torah text.

## Torah Specific Challenges

Additional challenges arise when trying to implement NLP projects in more niche dialects of Hebrew, in particular Torah texts. Torah texts often have ancient versions of Hebrew grammar and

[3] https://www.statista.com/statistics/262946/most-common-languages-on-the-internet/
[4] See older official stats with similar disparity at https://stats.wikimedia.org/EN/TablesWikipediaEN.htm and https://stats.wikimedia.org/EN/TablesWikipediaHE.htm

vocabulary, ranging from Biblical Hebrew to Rabbinic Hebrew. The former has vastly different words in many cases, and the latter has the added complexity of incorporating both Mishnaic Hebrew and Medieval Aramaic words. These niches require training examples within their own niche, which further limits the amount of available training data and other out-of-the-box tooling.

From experimentation during our YU CS Capstone project, we found that pre-trained models were more accurate when trained on niche-specific texts vs. more generalized Hebrew pre-trained models. Specifically, in our topic classification task, the BEREL model which was trained on Torah texts performed better overall than the other general-purpose Hebrew pre-trained models we tested (namely, HeBERT and AlephBERT).[5]

[5] For details on our findings, see this slide from our final presentation, and the data of our results on our github repo here