Spring 2024

A Comparison of Lexical Tokenization Methods

Nathan Culmer
nrc61@uakron.edu

Follow this and additional works at: https://ideaexchange.uakron.edu/honors_research_projects

Part of the Artificial Intelligence and Robotics Commons

Please take a moment to share how this work helps you through this survey. Your feedback will be important as we plan further development of our repository.

Recommended Citation
https://ideaexchange.uakron.edu/honors_research_projects/1831

This Dissertation/Thesis is brought to you for free and open access by The Dr. Gary B. and Pamela S. Williams Honors College at IdeaExchange@UAkron, the institutional repository of The University of Akron in Akron, Ohio, USA. It has been accepted for inclusion in Williams Honors College, Honors Research Projects by an authorized administrator of IdeaExchange@UAkron. For more information, please contact mjon@uakron.edu, uapress@uakron.edu.
A Comparison of Lexical Tokenization Methods

Nathan Culmer
CPSC 498: Senior Honors Project in Computer Science
Dr. Zhong-Hui Duan
April 26, 2024
Abstract

The purpose of this project was to compare tokenization methods, or methods of breaking up a text into meaningful parts for use in natural language processing. The effectiveness of several commonly used tokenization methods were investigated, including morpheme tokenization, which takes into account the linguistic features of the language. In addition, I proposed and implemented a new technique to consider the capitalization pattern of a word in the tokenization process, in order to allow this process to include more natural language features. The effectiveness of these methods was compared by using them in a sentiment analysis model for various datasets, including binary classification and multiclass classification datasets. This report summarizes these methods and the findings from the comparisons.
Table of Contents

1. Introduction
   1.1. Tokenization
   1.2. Whitespace Tokenization
2. WordPiece Tokenization
3. Morpheme Tokenization
   3.1. Morphemes
   3.2. Incorporating Capitalization
4. Methods
   4.1. Sentiment Analysis
   4.2. Environment
   4.3. MorphemeTokenizer
   4.4. Capitalization Tokens
   4.5. Neural Network
   4.6. rpy2
5. Results
   5.1. Sarcasm Dataset
   5.2. Disaster Dataset
   5.3. Happiness Dataset
6. Discussion
   References
A Comparison of Lexical Tokenization Methods

1. Introduction

1.1. Tokenization

In natural language processing, tokenization is the process of dividing text into discrete units of meaning, called tokens, primarily for use in neural networks. These tokens could be words, phrases, or parts of words, called subwords, depending on the method of tokenization used. Each unique token is then assigned a number, referred to as a token id, which is used by the neural network when processing the text. Through this process, a string of characters representing an entire sentence or larger text is converted into a list of integers, a necessary step for inputting text into neural networks, which work at the most basic level by multiplying input numbers by weights. It is the relationships between these tokens that a neural network can learn to understand, and so it is important to consider different methods of tokenizing a text. If entire sentences are made to be single tokens, a neural network will not be able to understand why those sentences mean what they mean and will not be able to make predictions about sentences not included in the training data. At the other extreme, character-level tokenization may not be very effective, as the relationships between individual characters may not indicate much about the meaning of an entire text.

A neural network can only work reliably with tokens that occurred in the dataset on which the network was trained, so a neural network must have a token vocabulary, a list of tokens and their corresponding ids that tokenized texts must match to be well-formed input. Lexical items that do not appear in this vocabulary are replaced with an out-of-vocabulary (OOV) token. The token vocabulary is fit on the training dataset that will be used in the neural network, and is often limited to a subset of the most common tokens in the dataset. This is because, in many cases, it would be impractical to have a vocabulary consisting of the hundreds of thousands or even millions of potential tokens that may appear in a large dataset, and, in any case, there may not be much data on the most uncommon tokens with which to learn. It is
important to note that any text being input into a neural network must use the same fitted vocabulary; that is, the sentence must be tokenized with the same token ids. Inputting text tokenized in some other way will produce either an error or garbage output that reflects a different input sequence to the intended one.

1.2. **Whitespace Tokenization**

The simplest and perhaps most intuitive method of tokenization is *whitespace tokenization*. This is a method of tokenization where the text is split by whitespace and punctuation, and the tokens are thus the strings of characters separated by those whitespace and punctuation characters. In simple terms, the tokens are words. Text is often preprocessed to be made entirely lowercase, because words with different capitalization patterns would otherwise have different token ids and thus be treated as unrelated words by the neural network. Figure 1 shows an example of whitespace tokenization.

| Sentence: | Computer science is enjoyable! |
| Tokens:   | computer science is enjoyable! |
| Token ids:| 1 2 3 4 5 |

**Fig. 1.** An example of whitespace tokenization with punctuation excluded.

However, whitespace tokenization has several drawbacks. Any word that does not appear in the dataset must be represented by the out-of-vocabulary token, even if the word is made up of parts that do appear in the vocabulary. Additionally, while it may make sense to tokenize text this way in English, languages like Mandarin or Japanese do not use whitespace to separate words, and other languages with complex morphology like Finnish or Turkish may contain too much information in space-separated words for this to be a useful method.\(^1\) Therefore, other

---

\(^1\) Consider the Nuxalk word *clhp’xwhtlhlhshkwuts*, meaning “he had [in his possession] a bunchberry plant”. There will naturally be fewer occurrences of any given word in a dataset of texts in a language that contains such large amounts of information in a single word, which makes it more difficult for a neural network to learn what those individual words mean.
methods of tokenization that avoid these problems have been created, which will be investigated below.

2. **WordPiece Tokenization**

WordPiece is a tokenization algorithm developed by Google which splits words into subwords called *wordpieces*. The WordPiece algorithm is similar to *byte-pair encoding*, a method of tokenization which first splits every word in the text into individual characters, forming the initial vocabulary. The pair of tokens that occurs most frequently together is then merged to create a new token, increasing the vocabulary size by one. This process is repeated iteratively until some specified vocabulary size has been reached. WordPiece uses a similar principle, but ensures that tokens are not merged into less common tokens. A score for each potential merge is calculated by dividing the frequency of the potential new token by the product of the frequencies of the two constituent tokens, and the token with the largest score is selected to be added to the vocabulary. The purpose of this method is to create a vocabulary of common words and affixes, while uncommon words are divided into several subword tokens instead of being replaced by an out-of-vocabulary token. An example of WordPiece tokenization is shown in Figure 2.

Sentence: Computer science is enjoyable!

Tokens: com puter science is enjoy able

Token ids: 1 2 3 4 5 6

**Fig. 2.** An example of WordPiece tokenization. This example does not come from any real data and is only intended for demonstration purposes.

Note that WordPiece tokenization does not consider the meaning of potential wordpieces, and so words may be split into tokens that have no real meaning in the language of the text (see Figure 2, where *computer* is split into two meaningless subwords). Nonetheless, it
performs well for many different languages, in terms of both the speed of the tokenization process and the performance of the models in which it is used, and remains a popular method of tokenization.

3. **Morpheme Tokenization**

3.1. **Morphemes**

A *morpheme* is the smallest grammatical unit of speech (one may note that this definition is rather similar to the definition of a token). For example, the word *antidisestablishmentarianism* consists of six morphemes: *anti-*-, *dis-*-, *establish*, -*ment*, -*arian*, and -*ism*; the word *dogs* consists of two morphemes: *dog* and -*s*; and the word *jumped* consists of two morphemes: *jump* and -*ed*. As morphemes are the smallest unit of language that carries meaning, it may be advantageous to use morphemes as tokens, so that all tokens carry some meaning (see Figure 3 below for an example of morpheme tokenization). This would allow neural networks to understand the meaning of uncommon and nonstandard words like *unsatisfyingness* that may otherwise be represented by an out-of-vocabulary token. It seems intuitive that dividing words into smaller meaningful parts would perform better in many language-related tasks than breaking words into smaller meaningless parts, as algorithms like WordPiece do. Additionally, as morphemes are a universal linguistic concept, morpheme tokenization can work with any language, though this method of tokenization may prove more challenging in some languages than others.

<table>
<thead>
<tr>
<th>Sentence:</th>
<th>Computer science is enjoyable!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tokens:</td>
<td>compute ###er science is enjoy ###able !</td>
</tr>
<tr>
<td>Token ids:</td>
<td>1 2 3 4 5 6 7</td>
</tr>
</tbody>
</table>

**Fig. 3.** An example of morpheme tokenization. The double pound signs represent that the token is a morpheme bound to the previous morpheme.

---

2 *Antidisestablishmentarianism* may be considered to be 7 morphemes, if the suffix -*arian* is considered to consist of two morphemes itself, -*ary* and -*an*. 
3.2. Incorporating Capitalization

Tokenization methods often ignore capitalization patterns, such that the words *Computer*, *COMPUTER*, and *computer* would all be represented by the same token. However, capitalization patterns may provide useful information about the meaning of a text, especially in the case of text posted on online social media platforms, where capitalization is often used as a tone indicator. Therefore, it may be useful to include capitalization in the tokenization process, but this introduces several problems. First, assigning different tokens to differently capitalized forms of the same word would cause the relationship between those forms of the word to be lost, such that the tokens *computer* and *Computer* would be as distinct as the tokens *computer* and *tree*. And while the neural network would learn the meanings of specific words when capitalized differently, the neural network would not learn the meaning behind capitalization patterns in general and would thus be unable to extrapolate to new words. Additionally, there may often not be enough data on a word to split it by its capitalization pattern without reducing the network’s understanding of that word; if half of the occurrences of *computer* have an initial uppercase letter, then the neural network’s understanding of the tokens *computer* and *Computer* would only be based on half the data as if they were considered to be the same token. Lastly, requiring every word to have multiple token representations would reduce the number of unique words in the vocabulary, unless the vocabulary was expanded to accommodate this.

4. Methods

4.1. Sentiment Analysis

In order to compare the effectiveness of these tokenization methods, they will be utilized in a neural network built to perform *sentiment analysis*. Sentiment analysis is the process of analyzing a text to determine the sentiment of the author. Determining whether or not a sentence is sarcastic or not, or whether the writer of the text holds a positive or negative opinion of the sentence topic, are examples of sentiment analysis. A simple method of doing this is to
create a *word embedding* for each token, an n-dimensional vector which indicates the sentiment of the token. For instance, words like *literally* or *obviously* may indicate sarcasm, while words like *empirical* and *formal* may indicate a lack of sarcasm, and words like *say* or *do* may not give any strong indication in either direction; the word embeddings for these words would reflect that. To predict the sentiment of a larger text, the embeddings for the tokens in the text can be added together, with the result indicating the overall sentiment of the text.

### 4.2. Programming Environment

The code for this project was written in Python using Google Colab, a “hosted Jupyter Notebook service.” The usage of a Jupyter Notebook allowed the code for the project to be compartmentalized, making it easy to tokenize different datasets with different methods.

### 4.3. MorphemeTokenizer

In order to divide text into morphemes, *morphemepiece*, a package for the R programming language, was used. This package includes a function which breaks a string of text into a sequence of tokens, where each token is a morpheme (see Figure 3 above). While morphemepiece does provide token ids for these morphemes, it does not restrict the vocabulary to the most common tokens, instead providing a unique id for every morpheme in the online dictionary *Wiktionary*. Therefore, a custom tokenizer class, *MorphemeTokenizer*, needed to be created to support the process of fitting a vocabulary, so that other datasets could be tokenized with the same limited, fitted vocabulary. Additionally, the value of token ids cannot be greater than the vocabulary size, so the morphemepiece token ids, whose maximum value is the size of the very large morphemepiece vocabulary, needed to be replaced with new values that range from zero to the size of the fitted vocabulary in order to be used in a neural network. *MorphemeTokenizer* also handles this process. Figure 4 below displays the full functionality of the *MorphemeTokenizer* class.
### 4.4. Capitalization Tokens

In order to include capitalization in the morpheme tokenization process, the discussed problems needed to be solved. This was accomplished by inserting special tokens into the token sequence to indicate the capitalization pattern of the preceding word, and then tokenizing the text such that words with different capitalization patterns would still be represented by the same token (see Figure 5 for an example). Continuing to make all occurrences of the same word be the same token, regardless of their capitalization patterns, means that capitalization can be incorporated into the process without losing the relationships between differently capitalized forms of the same words, and further allows the neural network to learn the meaning of capitalization patterns in general. This solution also solves the problem of the increasing vocabulary size, as only a few tokens are needed to represent all the possible capitalization patterns (see Table 1 for a list of capitalization patterns included in the custom MorphemeTokenizer class).
Sentence: Programming is FUN

Tokens: program ing [Xxxx] is [xxx] fun [XXXX]

Token ids: 1 2 3 4 5 6 7

Fig. 5. An example of morpheme tokenization with capitalization patterns included. Note that the capitalization tokens are inserted at the word level, not at the morpheme level.

<table>
<thead>
<tr>
<th>Token</th>
<th>General Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>tokenizer_code_lower</td>
<td>xxxxxx</td>
</tr>
<tr>
<td>tokenizer_code_upper</td>
<td>XXXXXX</td>
</tr>
<tr>
<td>tokenizer_code_capital</td>
<td>xxxxxx</td>
</tr>
<tr>
<td>tokenizer_code_alter</td>
<td>XxXxx</td>
</tr>
<tr>
<td>tokenizer_code_inv_capital</td>
<td>xXXXX</td>
</tr>
<tr>
<td>tokenizer_code_upper_to_lower</td>
<td>Xxxxxx, XXxxxx, XXXxx...</td>
</tr>
<tr>
<td>tokenizer_code_lower_to_upper</td>
<td>xXXXXX, xxxXXXX, xXxxXXX...</td>
</tr>
<tr>
<td>tokenizer_code_upper_to_upper_to_upper</td>
<td>XxxXXX, XXxxXX, XXXxXX...</td>
</tr>
<tr>
<td>tokenizer_code_lower_to_upper_to_lower</td>
<td>xXXxXx, xxXXxX, xxxXxX...</td>
</tr>
<tr>
<td>tokenizer_code_misc</td>
<td>XxXXxX, XXxXx, xXxXxX...</td>
</tr>
</tbody>
</table>

Table 1. A table containing the capitalization codes implemented in the MorphemeTokenizer class. As these need to be introduced to the sentences before morphemepiece tokenization is performed, and morphemepiece always splits on punctuation characters, the codes could not be contained within symbols like <> or [], as is common practice with similar special tokens.

4.5. **Neural Network**

A neural network to perform sentiment analysis was created with TensorFlow, an open-source library for machine learning developed by Google. Specifically, it was built with Keras, which is described as the “high-level API of the TensorFlow platform.” The keras_nlp package includes a WordPiece tokenizer, which was used to tokenize text with the WordPiece
algorithm. A series of online video tutorials for Keras included a simple model for sentiment analysis, where the input is a list of token ids, and the output is the probability of the sentence matching a label. A modified version of this model was used to investigate the effectiveness of the tokenization methods.

4.6. rpy2

As it would be inconvenient to perform morpheme tokenization in R and the rest of the tokenization and machine learning in Python, the rpy2 Python package was used, which is “an interface to R running embedded in a Python process.” This library allows R functions and data structures to be used in Python, so that the morphemepiece package can be used in Python.

5. Results

5.1. Sarcasm Dataset

The first dataset that will be used to investigate the effectiveness of the discussed tokenization methods is a dataset containing news article headlines, which have been labeled as sarcastic or not sarcastic. The source has preprocessed the headlines to be all lowercase, so the impact of capitalization tokens cannot be investigated with this dataset. The chosen vocabulary size was 10,000 words, at which size morpheme tokenization resulted in far fewer OOV tokens than whitespace tokenization—8,410 for the testing set with whitespace tokenization, and only 3,418 with morpheme tokenization. However, it was the WordPiece method which resulted in the fewest number of OOV tokens, with the testing set only containing 7 OOV tokens. Figure 6 below shows the proportions of OOV tokens in the whitespace and morpheme tokenized datasets.3

---

3 The WordPiece algorithm creates a vocabulary that includes, at a minimum, all of the characters in the training data. Thus, the only way out-of-vocabulary tokens can appear is if the tokenized dataset contains characters not present in the training data, so the OOV data is not particularly useful when evaluating WordPiece.
**Fig. 6.** A graph of the proportion of out-of-vocabulary tokens found in the training and testing sarcasm datasets when tokenized by each method.

The accuracy and loss of the model when run with each tokenization method for 30 epochs can be seen in Figure 7 below.

**Fig. 7.** Graphs displaying the model loss and accuracy of the WordPiece, morpheme, and whitespace tokenization methods when applied to the sarcasm dataset.

The validation accuracy of the models peaks at around epoch four in the case of all three tokenization methods. After this point, the validation accuracy of the models slowly trends downwards as the model begins to overfit for the training data, indicated by the increasing
validation loss. The validation accuracies of WordPiece and morpheme tokenization remained very close throughout the training process, while whitespace tokenization was much more volatile, both in terms of validation loss and accuracy over time. WordPiece also had a consistently lower training accuracy.

5.2. Disaster Dataset

The next dataset is a dataset of tweets that have been labeled as either being about a real disaster or not. The tweets have not been preprocessed in any way, so capitalization patterns are preserved and capitalization can thus be included in the tokenization process. Again, the vocabulary size was 10,000 words, and again morpheme tokenization resulted in fewer OOV tokens than whitespace tokenization—5,641 OOV tokens in the testing set for whitespace tokenization, and 4,021 in the testing set for morpheme tokenization. There were zero OOV tokens in the WordPiece testing set. The OOV proportions in the training and testing datasets can be seen in Figure 8 below.

![Figure 8](image)

**Fig. 8.** A graph of the proportion of out-of-vocabulary tokens found in the training and testing disaster datasets when tokenized by each method.

---

4 Consider a tweet like “That video really blew up!!!”, where *blew up* means that the video gained popularity, not that there was a physical explosion.
The model was trained for 30 epochs. The results of morpheme tokenization both with and without capitalization tokens, along with those of WordPiece and whitespace tokenization, can be seen in Figure 9 below.

![Figure 9](image)

**Fig. 9.** Graphs displaying the model loss and accuracy of the WordPiece, morpheme, and whitespace tokenization methods when applied to the disaster dataset.

Once again, the validation accuracy of the models peaked within the first few epochs before the model began to become overfitted, which can be observed through the increasing validation loss. The validation accuracy began to trend downward after that point, and though it was very volatile, morpheme tokenization seems to consistently have a slightly higher validation accuracy than the WordPiece method. Including capitalization tokens seems to have neither improved nor worsened the accuracy of the model, though the volatility makes it difficult to compare their performances.

5.3. **Happiness Dataset**

The last dataset is a dataset of responses to a survey about happiness. The respondents were asked to recall an event that made them happy, and their responses were categorized by the source of their happiness.\(^5\) Unlike the previous datasets, this dataset has multiple categories, 

\(^5\)The survey responses were each placed into one of the following categories: *achievement, affection, bonding, enjoy the moment, exercise, leisure*, and *nature.*
so the neural network was modified to accommodate this; instead of outputting one probability, one probability is output for each category and the one with the highest probability is chosen. The vocabulary was 10,000 tokens, and the WordPiece testing set contained zero OOV tokens. Morpheme tokenization again resulted in fewer OOV tokens than whitespace tokenization, with a significantly lower OOV proportion, as can be seen in Figure 10 below.

![Graph](image)

**Fig. 10.** A graph of the proportion of out-of-vocabulary tokens found in the training and testing happiness datasets when tokenized by each method.

The model was fit for 30 epochs. While the morpheme tokenization and WordPiece methods were still becoming more accurate by the thirtieth epoch, preliminary results indicated that neither method’s accuracy would surpass that of the whitespace tokenization method after even sixty epochs, so fitting was stopped at thirty epochs, as in the previously investigated datasets. These results can be seen in Figure 11 below.
Fig. 11. Graphs displaying the model loss and accuracy of the WordPiece, morpheme, and whitespace tokenization methods when applied to the happiness dataset.

Perhaps surprisingly, whitespace tokenization significantly outperformed the morpheme tokenization and WordPiece models, quickly achieving a high validation accuracy by the fifth epoch that the morpheme tokenization and WordPiece methods could not surpass even by the thirtieth epoch. Unlike the previous dataset, the inclusion of capitalization tokens seems to have worsened the model, as the model without capitalization tokens consistently has higher validation accuracy, though they do seem to have begun to converge by the thirtieth epoch.

6. Discussion

Morpheme tokenization did not prove to be more effective than WordPiece or whitespace tokenization for predicting the sentiment of a text, with both methods consistently performing better than or equal to the morpheme tokenization methods across several distinct datasets. There are several possible reasons why this could be the case.

First, most words in any given text will contain some semantic content that could be used to determine sentiment. However, many morphemes are *functional morphemes* as opposed to *lexical morphemes*— that is, they are used to perform grammatical functions as opposed to expressing things or concepts— and so tokenizing a text by morphemes may result in many tokens that cannot be used to predict sentiment. For example, the morpheme *-ed* is used to
indicate that a verb is in the past tense, and so it does not provide much information about the sentiment of a text in which it is used. Thus, the word embedding for the word *laughed* may strongly indicate sarcasm, while breaking the word into two morphemes—*laugh*, which strongly indicates sarcasm, and *-ed*, which does not indicate any sarcasm—may result in an average sentiment that is less sarcastic than the embedding of the combined form *laughed*. In other words, by tokenizing a text by morphemes, the proportion of tokens that can be used to predict the sentiment may drop, causing predictions to become less certain and thereby less accurate.

Similarly, morpheme tokenization may cause certain words to become less helpful in making predictions when the full word has connotations that cannot be simply derived from the morphemes that make it up. For example, the occurrence of the word *birthday* in a text may be a good indicator that the sentiment of the text is positive, while the occurrence of the morphemes *bear*, *-th*, and *day* may not be as helpful in determining the sentiment; nothing about those morphemes suggests the positive connotation that the word *birthday* has. This explanation may also be the reason for whitespace tokenization outperforming the other methods in the happiness dataset; that is, there may be certain keywords that allowed the model to have such a high accuracy, and dividing those words into subwords, as morpheme tokenization and WordPiece do, caused those keywords to be lost.

It is also possible that, when combining the most common characters, WordPiece may inadvertently create many tokens that represent morphemes. For example, *-ed* may be a common character sequence, and if it is more common than any other character sequence ending in *-ed*, then *-ed* may remain a token on its own. This could result in WordPiece acting as a sort of morpheme tokenizer, and perhaps one that is more efficient, as it only preserves the most common morphemes, combining less common prefixes and suffixes with the words they are attached to.

---

6 The morphemepiece tokenization function does indeed split the word *birthday* into these three tokens.
Morpheme tokenization did, however, prove to be advantageous in at least one way: the percentage and absolute counts of out-of-vocabulary tokens in texts tokenized by morphemes were significantly lower than the proportions and counts in whitespace tokenization, despite the vocabulary sizes being the same. This suggests that less of the meaning of the sentences was lost in the morpheme tokenization process, which could have benefits in other areas of natural language processing. Further research would involve investigating the effectiveness of these tokenization methods in other natural language processing problems, as the results from sentiment analysis cannot be extrapolated to other areas. Machine translation is one area where morpheme tokenization may be especially effective, as it could allow a translator to predict word forms absent from the training dataset. Certainly, if computers are to one day understand language as humans do, then it will be necessary for them to understand language in terms of morphemes as humans do.
References


