Natural Language Processing Techniques in PySpark using Yelp data

MIN GYU WOO, University of Waterloo, Canada

The computational power required to transform raw data into usable features for machine learning models increases drastically with the size of the data set. What is worse, is that Natural Language Processing techniques on text data is notorious for being slow. For example, removing certain words requires the driver to loop through every token in a string. In this project, I will dive deep into the efficient ways of applying natural language processing techniques on big data Yelp. With said transformed data, a model will be fit to show that the transformations are required for text-based models to perform well.

Additional Key Words and Phrases: big data, natural language processing, machine learning, pyspark, distributed computing

ACM Reference Format:
Min Gyu Woo. 2023. Natural Language Processing Techniques in PySpark using Yelp data. 1, 1 (July 2023), 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Raw unprocessed text tends to be extremely noisy. Unlike numerical features, they cannot be easily used and a data processing pipeline is required to extract useful features out of the text. These feature extraction techniques are inherently slow. Given a column of raw text, to obtain useful insight, typically the entire string must be processed. For example, for a given sentence of strings, each word must be read to count the frequency of which the word appears. Thus, for big data sets, it will take an unreasonable amount of time to process the features if standard loops are used. Therefore, the approach is to use distributed computing where possible to speed up the process. The features can be as simple as computing the frequency of each words in the training set or as complicated as determining the most common words and removing said words from each sentences. Regardless, these are necessary steps in order to train a useful machine learning model. The rest of the paper will discuss the pre-processing techniques that are important for text-based classification and compare the performance with well-known machine learning models.

2 PRE-PROCESSING RAW TEXT

The purpose of pre-processing any data set is to produce a clean data set such that a machine learning model can use the important portion of the feature to learn patterns and apply inferences. Numerical features are typically cost less computational power to transform them as the nature of these features are a "single-element" based. In comparison, text data is only limited to the decision of the corporation. For example, Twitter has a maximum tweet length of 280 characters [3], and Yelp can have up to 1500 characters for their Specialities section explaining what the business of interest does [9]!

Author's address: Min Gyu Woo, University of Waterloo, Waterloo, Canada, mgwoo@uwaterloo.ca.

The big question: are all the characters in the text important? The answer is no! Not all words in a sentence are weighted equally. There are potential key words that can summarize the direction of which the text is approaching. For example, let's suppose we have the sentence:

"The food here was amazing! The staff was kind and efficient, will definitely come again!"

Reading the sentence as is, indicates that the customer had a great time at the restaurant. The most important words for this sentence can be summarized with: "amazing", "kind", "efficient", "come again". The rest of the words do no provide enough value in making the decision on whether this comment refers to a positive or negative review. Thus, showing that not all words are important!

2.1 Basic Text Pre-Processing

Consider the following pair of important words from positive reviews:

"Amazing!" vs "Amazed" vs "amazing"

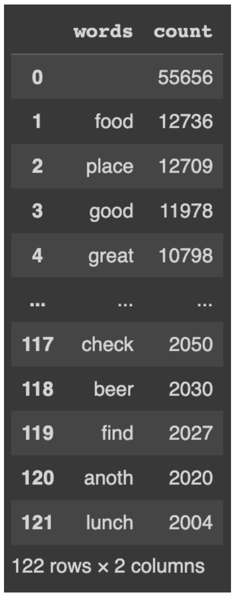
When reading with the human eyes, these words are identical in the context it was taken from. In general, these words all refer to a positive overview of the subject at hand (food, restaurant, staff, etc...). However, to a machine, these words are different strictly because of the characters not being equal. In order for a machine to understand these words are the same, they must be transformed to be made equal.

2.1.1 Lower-casing. When words are at the beginning of a sentence, the first letter is typically capitalized. Also, the fact that strict English rules are not enforced on the internet, there will be cases where a random character within a word is capitalized and the user did not change it. Thus, to make sure each word is the same, transforming all the letter characters to the same case will avoid the case where "AmaziNg" and "amazing" are classified as different words.

2.1.2 Punctuation Removal. For paragraphs, it is crucial to include punctuation in order to create coherent sentences. However, for the purpose of extracting string tokens from sentences, punctuation may cause problems. For example, consider the following sentence "I don't like the food.". The typical approach for tokenization is to split each string token on the white space delimiter. However, that will include the period for the last word. I.e.) "food." will be the token. This means that a machine will consider "food." and "food" different string tokens. Thus, the period should be removed before the tokenization process. In the same sentence, there is a case where punctuation should not be removed. I.e.) "don't". Removing the apostrophe will cause the token to become "dont" which is not the word of interest. Here, it does not matter as much, but if it was another word such as "we'll" vs "well". There is a clear difference between the two words that should be well-defined.

2.1.3 Stop Words Removal. In sentences not all words are weighted equally. Some words are more important than others in that those words determine the overall tone of the sentence, or in this case, the sentiment value. Using the example from before, "The food here was amazing!", the key word in this aspect is "amazing". That in of itself dictates that the sentence refers to a positive sentiment outlook. The other words have little to no value so they should be removed ahead of time and keep only the words that are important. Stop words are essentially the words that have little to no value. These sets of words are typically removed before any text advanced text processing is done. For the purpose of this project, the list of default stop words provided by PySpark StopWordsRemover function will be used. It consists of the most common stop words.

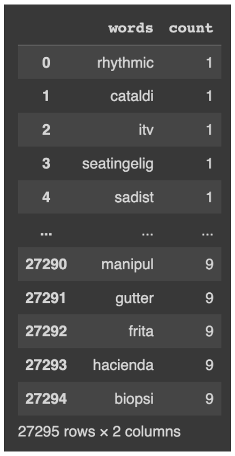
2.1.4 Common/Rare Words Removal. In addition to stop words removal, there is common and rare word removal techniques. For stop words, they refer to the words that appear so commonly that those words/tokens provide little



	words	count
0		55656
1	food	12736
2	place	12709
3	good	11978
4	great	10798
...	...	...
117	check	2050
118	beer	2030
119	find	2027
120	anoth	2020
121	lunch	2004

122 rows × 2 columns

Fig. 1. More than 2000 count example



	words	count
0	rhythmic	1
1	cataldi	1
2	itv	1
3	seatingelig	1
4	sadist	1
...	...	...
27290	manipul	9
27291	gutter	9
27292	frita	9
27293	hacienda	9
27294	biopsi	9

27295 rows × 2 columns

Fig. 2. Less than 10 count example

value in the overall classification task. In the other spectrum, there are words that are so uncommon that they also have little value to the overall problem at hand. These can also help remove the words that were misspelled due to user-error. The words that are spelt incorrectly will tend to be in the minority so this method is good at catching such mistakes. Using a default stop words list is helpful to remove the extremely common words/tokens. However, it is common to use most common and rare words removal because it adapts to the problem at hand. Essentially, after tokenizing the sentences, the frequency of the tokens are calculated. Then, there are options to remove the top N most common or

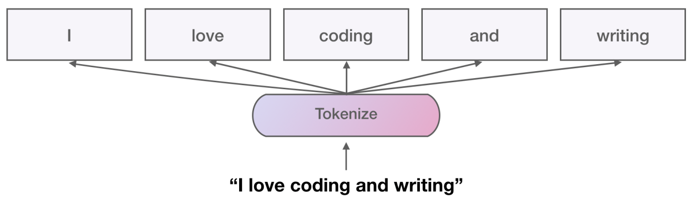


Fig. 3. Tokenization example

most rare words. A threshold can also be used to remove words, for example, that appear more than 2000 times in the training set. Figure 1 and 2 shows the cases with thresholds.

2.2 Tokenization

A computer does not understand the concept of words. Rather, it understands a sequences of string characters that are grouped together that could have meaning, called tokens. In the context of natural language processing, tokens extracted from sentences are typically sequences of strings grouped together split on white spaces. This tends to equate words and tokens together. As shown in the example figure 3 [4], the sentence "I love coding and writing" is split into a tokens. Typically, these are stored in a list where each element is the string token of interest.

2.3 Advanced Text Pre-Processing

So far, the processing techniques refer to simple tasks such as removing string characters, or having consistency such as lower-casing. These advanced techniques require more knowledge of the English language and/or numerically classify the words.

2.3.1 Stemming. Consider the follow words: "Caring", "Cares" and "Care". To an English speaker, these words are essentially the same. Due to grammar, the words are slightly different objectively, but the meaning is preserved. They all come from the word "Care". Thus, these words need to be grouped into one. One approach that can be done is called Stemming [6]. This takes the word of interest and removes the last few characters to produce a shortened form. The upside is, compared to the next technique Lemmatization, it is computationally fast but can produce words that lose the original meaning. In this example, the words will be shortened into "Car" which clearly is not the original meaning of the words. This essentially makes it so that "Caring" and "Cars" will be grouped together as one token "Car".

2.3.2 Lemmatization. Using the same example with "Caring", "Cares" and "Care", lemmatization will correctly classify these words into the token "Care" [6]. Lemmatization considers the context of which the word was used in the sentence. Unfortunately, due to that nature, a look-up table is typically used to correctly classify the words. Hence, the computation speed is much slower than Stemming however the words are typically preserved.

2.3.3 Term Frequency (TF). A set will contain all the possible tokens taken from the training set. One way of numerically quantifying the importance of a word, term frequency can be used. Just as the name suggests, term frequency of a token

$$idf(t, D) = log\left(\frac{N}{count\ (d \in D : t \in d)} \right)$$

Fig. 4. IDF Equation [8]

$$tf\,idf(t, d, D) = tf(t, d)\ .\ idf(t, D)$$

Fig. 5. TF-IDF Equation [8]

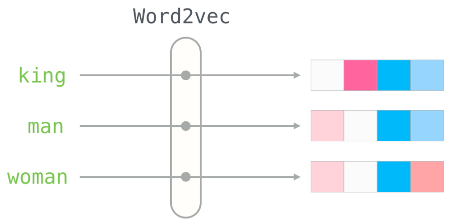


Fig. 6. Word2Vec Visual Example [2]

represents how often a token appears in text. The most common use of term frequency are the absolute frequency (number of occurrences) or relative frequency (number of occurrences / total token count) [5]. For the purpose of this project, the absolute frequency will be used. There are clear weaknesses of using only Term Frequency. For example, words such as "the" are extremely common so the Term Frequency for the token will be quite high. In order to avoid providing too much weight to common words, another method will be considered.

2.3.4 Inverse Document Frequency (IDF). In Figure 4, the Inverse Document Frequency of a token t with a corpus of tokens D, is the log of the total documents (in this case, number of comments N), divided the number of comments that had the token t in. This takes into consideration the example with "the" by weighting it less than say an uncommon word.

2.3.5 TF-IDF. It is most common to use a combination of both Term Frequency and Inverse Document Frequency. The equation used for the combination is in Figure 5. Essentially, the more relevant words will have higher TF-IDF value while the less relevant words will approach 0.

Regardless of if TF, IDF, or TF-IDF is used, there will be a sparse vector representation of all the possible words. That way, for any given text in the training set, there will be a fixed vector representation of said string. For words outside of the training set, they are either ignored or represented in a new way. In this case, the PySpark HashingTF function creates a large enough sparse vector where new words can be considered into the model. For the purpose of the project, a combination of TF-IDF will be used.

2.3.6 Word2Vec. Words need to be represented by numbers in order to train the machine learning models. One approach is word2vec. The general idea behind word2vec is to represent each word in the corpus as a vector of fixed length such that similar words are similar numerical. For example, looking at figure 6, the colours represent the magnitudes word embedding vector for each corresponding word. It can be seen that the word embedding learned the relationship between "king", "man" and "woman" (shown by the 3rd element of the vector). Looking at the vectors of "man" and "woman" since the first three elements are similar compared to "man" and "woman" vs "king". These embedding can be learned by the context and definition of the words within a sentence.

3 MODELLING

The problem statement at hand is to determine the sentiment value of Yelp reviews. Any rating greater than or equal to 3 stars will be considered a "good" sentiment, otherwise it will be considered "bad". The input will be a fixed length sparse vector which will indicate which words are present in a string of text.

3.1 Naive Bayes

One of the most common classification modelling techniques is Naive Bayes [7]. It is a standard model for testing whether an email is spam or not. This model can be adapted to this scenario as well. Essentially, Naive Bayes utilizes Bayes Theorem:

$$Pr(S|W) = Pr(S) * Pr(W|S)/Pr(W)$$

where for this use case, S represents the sentiment good or bad, and W is the occurrence of a specific token.

We "naively" assume independence between the events and calculate the sentiment value based off:

$$Pr(S|W_1 \cap ... \cap W_n) = \frac{Pr(W_1|S) \cdot ... \cdot Pr(W_n|S)}{P(W_1) \cdot ... \cdot P(W_n)}$$

whichever sentiment value gives the max probability will be taken.

3.2 Logistic Regression

Since the TF-IDF vector is a fixed vector, it can be fed into a logistic regression model such that the sentiment value is determined by the weighted sum of the vector of words.

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 W_1 + ... + \beta_n W_n$$

Finally, the output will be a probability between 0 and 1 due to the nature of the sigmoid function. The sentiment value will be determined based off if the probability is closer to 0 or 1:

$$\sigma(z) = \frac{1}{1+\exp(-z)}$$

where $z = \beta_0 + \beta_1 W_1 + ... + \beta_n W_n$.

3.3 Support Vector Machine

A support vector machine classifier will fit a hyperplane that will try to separate the classes. The variation used here is called Linear Support Vector Classifier provided by scikit-learn. For the LinearSVC function, the minimization function is:

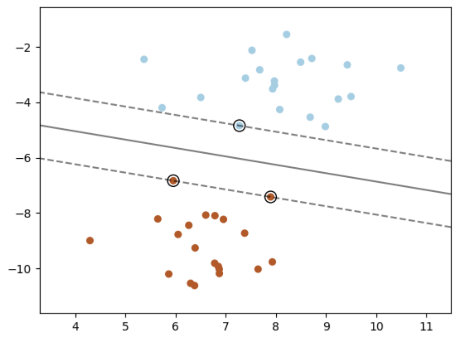
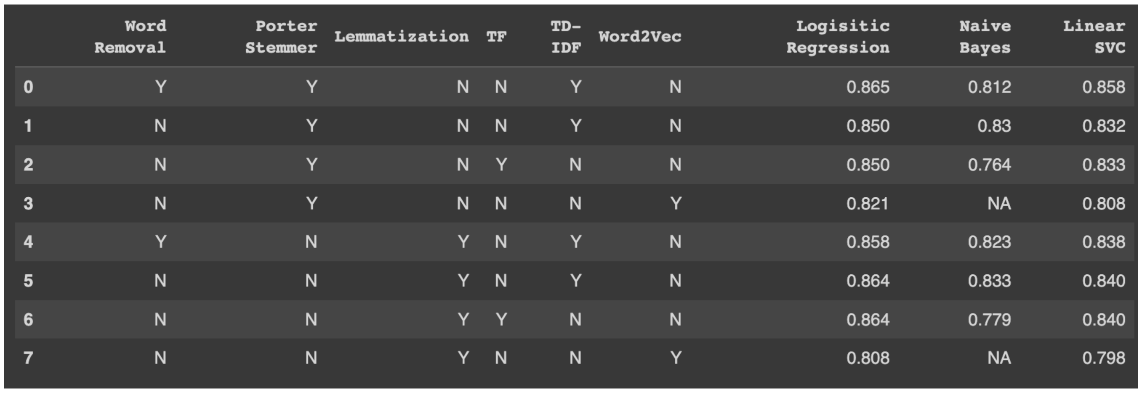


Fig. 7. Support Vector Machine example [1]



	Word Removal	Porter Stemmer	Lemmatization	TF	TD-IDF	Word2Vec	Logisitic Regression	Naive Bayes	Linear SVC
0	Y	Y	N	N	Y	N	0.865	0.812	0.858
1	N	Y	N	N	Y	N	0.850	0.83	0.832
2	N	Y	N	Y	N	N	0.850	0.764	0.833
3	N	Y	N	N	N	Y	0.821	NA	0.808
4	Y	N	Y	N	Y	N	0.858	0.823	0.838
5	N	N	Y	N	Y	N	0.864	0.833	0.840
6	N	N	Y	Y	N	N	0.864	0.779	0.840
7	N	N	Y	N	N	Y	0.808	NA	0.798

Fig. 8. Final Results

$$min_{w,b}\frac{1}{2}w^tw + C\sum_{i=1}^{n}\max(0, 1 - y_i(w^t\phi(x_i) + b))$$

which utilizes the hinge loss and a linear kernel.

4 RESULTS

Due to lack of computational power, only a fraction of the data set was used, in other words 0.1% of the data set. In order to compare the importance of the necessary text features, the models were fit with and without said features. More specifically, comparison was done for Porter Stemmer vs Lemmatization, with and without Removal of Stop Words (including most common and rare word removal), and finally the affect of TF vs TF-IDF vs Word2Vec.

Looking at the results in figure 8, Naive Bayes performing the worst is expected as the assumption of independence between words is quite a strong one. Word2vec seems to have have the worst performance in comparison with using TF and TF-IDF. This may be due to the fact that the word embeddings were fit on a small data set. With enough information of all words in the corpus, the word embeddings should have a better representation of the words and perform better.

The affects of Porter Stemmer vs Lemmatization is unique. Porter Stemmer performed better in the case where word removals were present. Since the stemming technique is applied first before removing most common / rare / stop words, it could be that more low weighted words were removed from the text compared to lemmatization thus increasing performance.

Overall, the best model seems to be performing Logistic Regression on text data that removed most common / rare / stop words, porter stemmer and td-idf.

REFERENCES

[1] *1.4. Support Vector Machines.* URL: %7Bhttps://scikit-learn.org/stable/modules/svm.html#svm-mathematical-formulation%7D.

[2] Alammar, Jay. *The illustrated word2vec.* URL: %7Bhttps://jalammar.github.io/illustrated-word2vec/%7D.

[3] *Counting characters | docs | twitter developer platform.* URL: %7Bhttps://developer.twitter.com/en/docs/counting-characters#:~:text=In%5C%20most%5C%20cases%5C%2C%5C%20the%5C%20text,more%5C%20characters%5C%20as%5C%20its%5C%20weight.%7D.

[4] Dair-Ai. $NLP_FUNDAMENTALS/1_nlp_basics_tokenization_segmentation.ipynbatmasterDair-ai/NLP_FUNDAMENTALS$. URL: %7Bhttps://github.com/dair-ai/nlp_fundamentals/blob/master/1_nlp_basics_tokenization_segmentation.ipynb%7D.

[5] Person. *Understanding TF-IDF for Machine Learning.* Oct 2021. URL: %7Bhttps://www.capitalone.com/tech/machine-learning/understanding-tf-idf/%7D.

[6] Saumyab271. *Stemming vs lemmatization in NLP: Must-know differences.* Jul 2022. URL: %7Bhttps://www.analyticsvidhya.com/blog/2022/06/stemming-vs-lemmatization-in-nlp-must-know-differences/#:~:text=Stemming%5C%20is%5C%20a%5C%20process%5C%20that,’%5C%20would%5C%20return%5C%20’Car’.%7D.

[7] Singh, Sanskriti. *How to use naive bayes for text classification in python?* Jul 2022. URL: %7Bhttps://www.turing.com/kb/document-classification-using-naive-bayes%7D.

[8] *Understanding TF-ID: A simple introduction.* May 2019. URL: %7Bhttps://monkeylearn.com/blog/what-is-tf-idf/%7D.

[9] *Yelp Data Quality Guidelines.* URL: %7Bhttps://docs.developer.yelp.com/docs/yelp-data-quality-guidelines#:~:text=Could%5C%20cover%5C%20things%5C%20like%5C%20where,Character%5C%20limit%5C%20(1000)%7D.