Exploring the Efficacy of Simple Models and Feature Engineering Techniques in Text Classification: A Comparative Analysis

Daniel Pereira da Costa
University of Southern California, Los Angeles, United States
danielp3@usc.edu

Abstract

Text classification is one of the core tasks of Natural Language Processing. Despite being such a complicated task, reasonable results can be obtained by using simple models such as Logistic Regression and Naive Bayes. Beyond the choice of model, the method of feature engineering employed also holds an important role in predicting accurate labels. In this paper, we implement these models from scratch using their default configurations and engage in a comparative analysis with alternative feature engineering techniques such as Bag-of-Words, TF-IDF (Term Frequency - Inverse Document Frequency), and Word Embedding.

1 Introduction

The primary goal of this paper is to provide a comprehensive exploration of the methodologies and strategies involved in constructing Logistic Regression and Naive Bayes models from scratch. When building these models, it is important to focus not only on correctly implementing them but also on the discerning selection of feature engineering techniques that yield optimal results across all four datasets. To achieve this goal, not only are the models effectively developed, but also analyze feature engineering methods.

2 Text preprocessing

The first step of any NLP problem is to preprocess the take before feeding it to the model. In this project, there were five techniques used for preprocessing.

The first technique used was expanding contractions. Contractions are linguistic forms where words or phrases are condensed by omitting certain letters and substituting them with an apostrophe. To accomplish this, a dictionary was constructed, featuring prevalent contractions as keys and their corresponding expanded forms as values. For instance, "won't" was associated with "will not" in this mapping process.

The second step consisted of removing digits and words that did not have any relevant information about the class. To achieve this, the following regex formula \b\w*\d\w*\b, which matches words that contain at least one digit in them, was applied.

Next, punctuations were removed since they were not relevant the corresponding text. To achieve this, all punctuations in the method "string.punctuations" were replaced with an empty space.

In order to enable the machine learning models to prioritize essential information, English stop words were eliminated from the dataset. Although this specifically impacted datasets containing English words ("products.train.txt" and "4dim.train.txt"), it was a worthwhile preprocessing step to enhance the quality of the text data.

Finally, all characters were converted to lowercase to reduce the data dimensionality. Then, the text tokens were created by splitting the text based on white spaces.

3 Naive Bayes

Naive Bayes is a generative algorithm that estimates the joint probability $p(x, y)$, where 'x' represents a bag of words, and 'y' is the corresponding label. As elaborated in Chapter 2 of Eisenstein (2019), predicting a label from a bag-of-words involves constructing a weight vector θ and a feature vector $f(x, y)$. The development of this algorithm can be divided into three key components: Vocabulary, Feature Vector, and Feature Weights.

3.1 Vocabulary

The initial step in constructing the feature vector involved creating a vocabulary. This vocabulary serves as a dictionary containing all the words used during the training phase of the algorithm. However, instead of including every word from the training corpus, words that occurred less frequently than a specified *threshold* were filtered out. This was done in order to reduce noise caused by these infrequent words and also to improve the generalization of the algorithm.

As a result of this filtration, not all words from the training corpus were included in the vocabulary. To address this situation, any words not present in the vocabulary were uniformly replaced with the Out-Of-Vocabulary token, denoted as *OOV*. The same *OOV* token substitution was applied during the inference phase: if a word was not found in the vocabulary, it was mapped to the *OOV* token.

3.2 Feature Vector

For the core part of the assignment, the Bag-Of-Words method was applied to build the feature vector of each sentence. As described in Eisenstein (2019), bag-of-word is a column vector used to count words in the vocabulary, where each item is the number of times that particular word appears in the sentence. To construct this vector, the *Counter* class is applied to each sentence. It returns a dictionary containing the word count for every word within that specific sentence.

3.3 Feature Weights

To build the feature weights vector, it is necessary to create 3 different dictionaries: *count_word_label*, which tracks the counts of (word, label) pairs; *count_words_per_label*, responsible for tallying the number of words associated with each label; and *count_label*, which records the occurrences of each individual label. Once these dictionaries are populated, it is possible to estimate the *p(x,y)* by multiplying the prior probability with the expected frequencies of each possible event. It is important to note that the math.log was applied throughout the assignment to mitigate the risk of numerical underflow.

Since the vocabulary is limited, there are likely to be pairs of labels and words that never appear in the training set. To handle this edge cases, laplace smoothing was used, where each element of the *count_word_label* dictionary received was incremented by one.

3.4 Prediction

After training the model, to predict a new sentence, it is just required to multiply the $feature\_vectors * feature\_weights$ for each of the N labels and choose the label y that has the highest probability.

4 Logistic Regression

According to Eisenstein (2019), Logistic Regression is a model that estimates the conditional probability $p_{Y|X}$, unlike naive Bayes which starts from the joint probability. The development of this algorithm can be divided into three key components: Vocabulary, Feature Vector, and Feature Weights.

4.1 Vocabulary

The process used for building the vocabulary was pretty similar to the on described in section 3.2, with the primary distinction being the incorporation of a *max_features* parameter designed to limit the vocabulary's size. This variable was introduced to reduce the dimensionality of the data and improve performance, as the number of feature become massive for some datasets such as the "products.train.txt" which had a vocabulary of more than 26,000 words.

4.2 Feature Vector

For encoding the features, the TF-IDF (Term Frequency - Inverse Document Frequency) technique was applied. TF-IDF is used to convert a document into a structured format. It is a numerical representation of how important a word is to a document in a collection or corpus (Bafna et al., 2016).

The TF-IDF matrix is created with a shape of (n_documents, vocabulary), where the vocabulary size is limited by the max_featueres variable described in the previous section. To avoid edge cases such as where the Term Frequency part is zero, +1 was added to both the numerator and denominator. It is also important to note that during inference, the Inverse Document Frequency (IDF) component of TF-IDF is derived from the training corpus and should be stored within the model, while the Term Frequency (TF) is calculated afresh for each new sentence during inference.

4.3 Feature Weights

The logistic regression weights are updated on every iteration of the algorithm based on the gradient of the logistic loss, which is equal to the difference between the expected counts under the current model and the observed feature counts. Once the desired number of epochs is reached, the outputs are two matrices: W(num_features, num_labels) and b(num_labels). Since the datasets are multi-label, a numerically stable softmax activation function was employed. Due to the possibility of TF-IDF values becoming quite small, there's a potential for numerical issues to arise. Therefore, a numerically stable softmax was utilized to mitigate this.

Another technique employed to achieve better results was the mini-batch training (Brownlee, 2023). In this technique, the weights are updated using a portion of the dataset at a time (batch_size), reducing computation time and memory usage.

4.4 Prediction

Similar to section 3.4, the prediction is obtained by the formula: $y_{pred} = \sigma(Wx + b)$

5 Extra Mile

5.1 Logistic Regreesion - Word2Vec

As mentioned in the previous section, TF-IDF was the chosen feature engineering technique for implementing logistic regression. However, this method comes with the issue of high dimensionality since the number of features is the same as the vocabulary size. To analyze the effectiveness of TF-Idf, pre-trained embeddings were also used to serve as a comparison. More specifically, Word2Vec (Mikolov et al., 2013) with 300 dimensions were used.

To incorporate the word embeddings, creating a matrix representation of each sentence was necessary. In Logistic Regression, where the number of features is constant, all sentences must be encoded to have the same size. To accomplish this, each sentence was transformed into a 300-dimension array by taking the average of all words within that sentence. Sentences, where any of its terms have an embedding, were replaced by an array of zeros.

Following the encoding of each sentence, the same logistic regression structure was employed (same epochs, batch_size, and learning_rate) to ensure a fair comparison. The results and their comparison are detailed in the Experimental Results section.

5.2 Naive Bayes - TF-IDF

A second model was developed using TF-IDF for preprocessing to evaluate the effectiveness of Naive Bayes with Bag-of-Words. For a fair comparison, the model's text preprocessing (tokenization, removal of stop-words, punctuations, contractions expansion) and vocabulary were kept the same.

To compare time performance, the model was constructed using the scikit-learn built-in package, enabling a time complexity comparison between the pre-built Naive Bayes classifier and the custom implementation.

6 Experimental Results

Each of the models was evaluated across the four different datasets: questions.train.txt, producs.train.txt, odiya.train.txt, and 4dim.train.txt. A manual hyperparameter tuning process was performed for NaiveBayes and Logistic Regression models to enhance performance. This involved fine-tuning several key hyperparameters, including the threshold (minimum required word occurrences), max_features, number of epochs, batch_size, and learning_rate. The final values of each of these parameters were hard-coded into the train.py file as they differ by dataset.

As validation sets were not provided, the training files were partitioned into 80% portion for training and 20% for validation. The results displayed in Table 1 correspond to the accuracy achieved on these validation sets.

6.1 Naive Bayes

In the case of Naive Bayes, the only hyperparameter is the threshold. The best accuracies were obtained from the following values for different datasets: 9 for the "questions" dataset, 2 for the "odiya" dataset, 2 for the "products" dataset, and 0 for the "4dim" dataset. From Table 1, we can observe that Naive Bayes achieved reasonable results across all four datasets, even on question and odiya, which are multi-class datasets.

While Naive Bayes may yield reasonable results, it's crucial to emphasize that its training time is significantly influenced by the size of the vocabulary. This dependency ends up becoming a limiting factor for training this model with large vocabularies. For instance, when training the model on 2.8 million tokens using an M1 Pro with 10 cores, it took 46.1s. In inference, it took 10.1 seconds to classify 700,000 tokens.

When comparing the custom Naive Bayes (BoW) with Naive Bayes (TF-IDF), we can observe that the former achieved a higher accuracy in 3 out of the 4 datasets. However, the latter achieved a better result in the "questions" dataset. Given that both models employed identical vocabularies

Model	**questions**	**odiya**	**products**	**4dim**
NaiveBayes (BoW)	68.70%	91.97%	77.88%	81.85%
Naive Bayes (TF-IDF)	73.35%	82.27%	78.09%	66.35%
Logistic Regression (TF-IDF)	71.27%	84.38%	70.83%	80.20%
Logistic Regression (Word2Vec-300d)	20.90%	45.39%	54.76%	20.83%

Table 1: Accuracy scores of each model evaluated on 20% of each dataset

and text preprocessing techniques, it is reasonable to conclude that the Naive Bayes model is better suited for these datasets. However, in terms of time complexity, the Naive Bayes (TF-IDF) from scikit-learn runs 10x faster, 5s, on the same hardware.

6.2 Logistic Regression

For logistic regression, the hyperparameters were all 5 of them. The best accuracies were obtained from the following values (learning_rate, epochs, threshold, max_features, batch_size) for different datasets: (0.15, 500, 0, 150, 32) for "questions," (0.01, 1000, 10, 1000, 256) for "odiya," (0.95, 100, 2, 1000, 256) "products," and (0.35, 500, 5, 2000, 64) for "4dim".

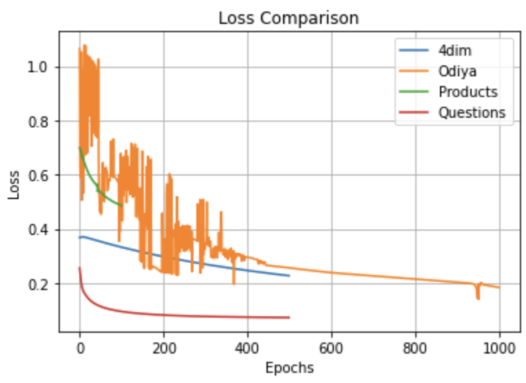

Figure 1: Logistic Regression - Cross-Entropy Loss

The manual hyperparameter tunning was conducted to avoid overfitting, always aiming to minimize the loss function displayed in Fig. 1. From this graph, it is possible to observe that the loss for the Odiya data varied significantly more than the others. This discrepancy can be attributed to the dataset's inherent imbalance, where the 'sports' class accounts for 46% of the total samples.

When comparing the results of the Logistic Regression with TF-IDF with the Word2Vec one, it is easy to see that the former outperformed the latter for all datasets. This is definitely expected for the datasets that did not have English word tokens ("questions" and "odiya") since there is no Word2Vec embedding for them. However, it is surprising that the same trend persisted for the remaining datasets. It is possible to conclude that the approach of averaging all of the word embeddings for each sentence was not effective for these datasets. Additionally, the fact that all of the hyperparameters were kept the same across both models indicates that they were not well-suited for the Word2Vec logistic regression approach.

7 Conclusion

The model that have the best average accuracy across the 4 datasets is the Naive Bayes (BoW) with 80.08%, followed Logistic Regression (TF-IDF) with 76.67%, Naive Bayes (TF-IDF) with 75.02% and Logistic Regression (Word2Vec) with 35.27%.

Each dataset possesses unique characteristics that affect the performance of each model. As discussed previously, the datasets "odiya" and "questions" affected the Word2Vec model since they did not contain English tokens. Since Tf-IDF and BoW are built from the training corpus, they remain unaffected. The hardest dataset to predict its label was the "questions" dataset. The main reason for this is because of the high quantity of labels, 6, and also the fact that the dataset is imbalanced, with the label "6" accounting for only 1.6% of the entire dataset.

Although the products dataset comprises only two labels, it consists of over 32,000 rows. This results in making the training of the algorithm way longer, causing the Naive Bayes (BoW) to take almost 1 minute to train on an M1 Pro (10 cores).

While every model has its own set of advantages and disadvantages, when considering an overall perspective, the Naive Bayes (BoW) model outperformed the others across all four datasets.

External Sources

Grammarly and ChatGPT were used as spell-checker tools. Here is the link to the chat session used on ChatGPT.

References

Prafulla Bafna, Dhanya Pramod, and Anagha Vaidya. 2016. Document clustering: Tf-idf approach. In *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 61–66.

Jason Brownlee. 2023. Mini-batch gradient descent and dataloader in pytorch. September 15, 2023.

Jacob Eisenstein. 2019. *Introduction to Natural Language Processing*. Adaptive Computation and Machine Learning. MIT Press.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space.