Implementation of the Naive Bayes Classifier as a Tweet Sentiment Classifier

An Empirical Study of the Naive Bayes Classifier

Henry Sue
UC Riverside
hsue002@ucr.edu

ABSTRACT

Text classification is the process in which text documents are sorted or "classified" into preset groups automatically with the use of machine learning algorithms. The Naïve Bayes Classifier is a popular text classification algorithm due to its simplicity and performance. It is typically used to establish baseline performance for state-of-the-art models to compare with. Machine learning packages for popular programming languages such as SciKit-Learn for Python or Text-Analysis.jl for Julia, have a pre-built implementation of the naïve bayes algorithm. This paper rebuilds and re-implements the naïve bayes classifier from scratch and benchmarks the implementation against the prebuilt Naive Bayes function in SciKit-Learn.

KEYWORDS

Text Classification, Naïve Bayes Classifier

1 INTRODUCTION

Text classification has been a topic in the machine learning space for many years. Text classification is the automated process of labeling documents with pre-determined class labels. A simple solution that has provided baseline performance is the Naïve Bayes Classifier, which is based on Bayes Theorem. The "Naïve " portion of the Naïve Bayes algorithm comes from the assumption that all features are independent of each other with respect to the class label. When dealing with text-classification problems, there are two types of Naïve Bayes models that are typically used: the "multinomial" Naïve Bayes, and "Bernoulli" Naïve Bayes. The two types assume different distributions for document features or "event models" [7]. The Naïve Bayes classifier then predicts the likelihood that a document fits into each class. Sentiment analysis is a subproblem of text classification that aims to predict whether the underlying meaning of a document carries a positive or negative connotation or meaning. Sentiment classification has the challenge of parsing latent meaning from text, and often struggles with edge cases that can be identified by contextual clues. An example of this phenomenon is sarcasm, where a sentence may comprise positive words but is used in a negative indication.

In the context of text classification, the Naïve Bayes Classifier relies on the bag-of-words model, a sparse representation of word frequency for each document. The classifier then takes each word count as independent features that serve as input for the model. Despite the simplicity of the Naïve Bayes classifier, the classifier performs quite well, and is very efficient, requiring no preset parameters. Further, the Naïve Bayes Classifier still performs quite well even in situations where the independence assumption should not apply. Natural Language is an example of this, as many words are related and conditionally dependent. For example, the term 'Christmas' is almost always accompanied by the words 'Holidays' or 'Merry'. The advantage of Naïve Bayes model is that the model aims to predict the classification by comparing probabilities that a document is of a certain class, rather than attempting to estimate the exact class probability. This leads to a robust model, as documents are usually skewed toward one classification.

2 RELATED WORK

The Naive Bayes Classifier is a classification model that is popularly used as a baseline to compare new or state of the art models with. As such, there is much documentation on similar methods. A main source of information for this paper is Speech and Language Processing (Jurafsky & Martin 2019) [5]. This textbook covers many topics about natural language processing, including a chapter on Naive Bayes and Sentiment Classification. Within this chapter, although there is a focus on the Naive Bayes classifier, the chapter also goes into detail about several tasks for text processing, including sentiment classification, spam detection, and authorship attribution. This chapter also includes psuedocode for the Naive Bayes Algorithm, which my implementation is based upon.

Additionally, a well-cited article "Semantic text classification: A survey of past and recent advances" (Altinel & Ganiz 2018) [1] summarizes advances in text classification models. This article weighs the differences between semantic text classification and traditional text classification. A main topic that they focus on is the reliance of traditional models on the Bag of Words (BoW) or Vector Space Model (VSM). This is relevant to our paper as the Naïve Bayes Classifier also relies on the sparse matrices of term frequencies. Later, our paper will discuss the drawbacks of the Bag of Words model, and how state of the art models use different techniques to save memory space.

In his 2018 article "Bayesian Naïve Bayes classifiers to text classification" [9], Xu asserts that Naïve Bayes classifiers are not fully Bayesian. He investigates the use of a Bayesian conversion of the Naïve Bayes model to mitigate the influence of the independence assumption. In addition to investigating the novel Fully Bayesian Naïve Bayes, Xu also documents the traditional methods for all flavors of the Naïve Bayes, including the Multinomial, Bernoulli and Gaussian event models. This paper's contents are out of the scope of traditional Naive Bayes classifiers.

In order to push state of the art performance, recent advances in Natural Language Processing involve training large neural networks using novel language representation models such as BERT (Devlin et al. 2018) [2]. However, as presented by Friedman, Geiger, and Goldsmith (1997) [3], the Naive Bayes Classifier's performance

is very close to state of the art, and with several revisions to the assumption structure can achieve performance to state of the art with a much simpler model (Bayesian Network Classifier).

3 PROPOSED METHOD

3.1 The Naïve Bayes Classifier

The proposed model that this paper follows is analogous to the model described in "Speech and Language Processing" by Martin and Jurafsky [5]. The Naïve Bayes Classifier relies on the "Naïve " assumption that all features (or, in other words: the number of times a word appears in a document) are independent events. In practice, this generalization is usually false; many words in natural language are highly dependent on other words, as there are special types of words that function by modifying another in context. Despite this, the Naive Bayes classifier is remarkably robust.

The bag-of-words model is a representation of a text document by the frequency of occurrences of each word. For example, a bag-of-words representation of the sentence "dog dog cat dog bird" can be represented by the bag-of-words matrix:

$$\begin{vmatrix} dog & cat & bird \\ 3 & 1 & 1 \end{vmatrix}$$

Thus, each text passage or "document" can be summarized by a vector of each word frequency. Given: Document 1 = "dog dog cat dog bird", and Document 2 = "cat dog bird cat bird". Therefore, we can represent each document by the following vectors:

$$Vocabulary = \left[dog, cat, bird\right]$$

$$Document1 = \left[3, 1, 1\right]$$

$$Document2 = \left[1, 2, 2\right]$$

This is known as *"count vectorization"*. In order to speed up lookup and processing, each unique word can be instead represented by an identifier. This process is called *tokenization*. In our implementation, each document in the dataset is first tokenized by its vector index, then represented as the count vector.

The pseudocode for the process is:

```
1: for document ← dataset do
2:     for all word ∈ document do
3:         if word ∉ vocabulary then
4:             vocabulary+ = word
5:             vector[word] = count(word)
6:         else
7:             vector[word] = count(word)
8:     output(vector)
```

Where *vocabulary* is the set of all words that have occurred at least once, and *vector* is the vector representing the document.

The Naïve Bayes Classifier is based on Bayes Theorem, where the probability of an event A occurring given that B has occurred can be broken down into 3 different probabilities:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{1}$$

Thus, we can apply this to document classification, where the probability the the document is class c given document d is:

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)} \tag{2}$$

Or, in other words, the probability that we observe class c given the document d can be represented as the *prior probability* of class c multiplied by the *likelihood* of document *given* c, divided by the *prior probability* of document d. We can then represent our document as the combined probability of its constituent n features:

$$P(d|c) = P(f_1, f_2, ..., f_n|c) \tag{3}$$

The classification step that the Naïve Bayes classifier takes is by then comparing the posterior probabilities of each class given the document, and outputs the higher one (the *argmax* of P(c|d).)

There are a few simplifications that we are able to do: First, we are able to neglect the prior probability P(d), as this stays constant for all classes, as document d does not change. Because we are comparing the probabilities of each class with respect to the same document d, we are only required to compare the numerator of Bayes' Theorem. Additionally, with the Naïve assumption that each feature is independent, we can decompose the likelihood into the multiplication of the probabilities of each feature:

$$P(f_1, f_2, ..., f_n|c) = P(f_1|c) \cdot P(f_2|c) \cdot ...P(f_n|c) \tag{4}$$

Finally, we can write the predicted class equation as a function of the prior probability $P(c)$, and the decomposition of the likelihood of the document $P(d|c)$:

$$C_{pred} = \text{argmax}\, P(c) \cdot \prod_{f} P(f_n|c)$$

Another typical optimization is taking the log prior and log likelihood instead of the prior or likelihood by itself. As the feature and document space gets larger, the multiplication of the prior and feature-wise likelihood can result in miniscule fractions that the computer cannot represent. Therefore, the use of log space allows for the processing of larger datasets with less risk of underflow and with more efficiency. Using log space approximation is also safe for this calculation as we are comparing the estimated probabilities of each class against each other, rather than trying to predict each probability exactly.

3.2 Training the Naïve Bayes Classifier

In order to train our Naïve Bayes Classifier, again, following the algorithm described in Speech and Language Processing (Jurafsky & Martin 2019) [5]. In order to make our maximum likelihood prediction, we must find the prior probability $P(c)$ and the likelihood $P(f_n|c)$. In context of our dataset of text documents, we can estimate the prior probability

$$P(c) \approx \frac{\text{Number of documents of class 'c'}}{\text{Total number of documents in Dataset}} \tag{5}$$

We then need to estimate the likelihood $P(f_n|c)$. We are able do that by letting each feature be the presence of each word, therefore

the our likelihood of each word occurring given class c can be estimated as:

$$P(w_n|c) \approx \frac{\text{Number of times w appears in class 'c'}}{\text{Sum of all counts of words in class 'c'}} \tag{6}$$

In practice, however, we run into an issue if a word does not appear in the class corpus in the training data! If we run into a word that is absent from the training vocabulary of a class, then the resulting probability $P(w_n|c)$ will always equal 0, which will cause the resulting prediction of that class to always fail. We therefore introduce Laplace smoothing, which is the concept of adding '1' to the potential count of each word in order to guarantee the existence of a count, regardless of if the word is absent in the training documents of class c.

$$P(w_n|c) = \frac{count(w_n, c) + 1}{\sum_{w \in V}(count(w, c) + 1)} = \frac{count(w_n, c) + 1}{(\sum_{w \in V} count(w, c)) + |V|} \tag{7}$$

In addition to Laplace Smoothing, if we encounter a word that does not exist in any document in *any* class, we ignore the effect of that word as we cannot compute a likelihood estimate of that word. In essence, this allows us to process the document without unknown words.

The pseudocode for the training algorithm is then:

1: **Function** Train NB(D,C) **returns** $\log(P(C))$ and $\log(P(w|c))$
2: **for** $classc \in C$ **do**
3: N_{doc} = number of documents in D
4: N_c = number of documents from D in class c
5: logprior = log $\frac{N_c}{N_{doc}}$
6: $V \leftarrow$ vocabulary of D
7: **for all** $word \in V$ **do**
8: count(w,c) ⟵ total occurrences of w in docs in class c
9: loglikelihood(w,c) ⟵ $\frac{count(w,c)+1}{\sum_{w \in V}(count(w,c)+1)}$
10: **return**(logprior, loglikelihood, V)

And the pseudocode for the test algorithm is:

1: **Function** Test NB(testdoc,logprior,loglikelihood, C, V) **returns** best 'c'
2: **for all** $classc \in C$ **do**
3: $sum[c] \leftarrow$ logprior[c]
4: **for all** word in testdoc **do**
5: **if** word ∈ V **then**
6: sum[c] += loglikelihood[word,c]
7: **return**(argmax$_c$ sum[c])

Psudeocode for both training and testing algorithm is adapted from Martin & Jurafsky (2014) [5].

4 EXPERIMENTAL EVALUATION

In order to benchmark the performance of our implementation of our Naïve Bayes Classifier, we apply the Naïve Bayes to Sentiment Classification. We use a novel dataset called "Sentiment140", which comprises short text documents scraped from Twitter that were automatically labeled by the contents of happy or sad emoji (Go, Bahayani, Huang 2009) [4]. The dataset has two comma-delimited files, a training set of 16 million automatically labeled tweets and a holdout test set of 498 hand-labeled tweets. Due to memory limitations of the numpy array datatype, which my implementation of the Naïve Bayes Classifier relies on, we use a randomly generated stratified subset of the training data, using 2% of the training dataset or approximately 320,000 tweets. In order to split the data, we call the train_test_split function from scikit learn with the stratify setting to true.

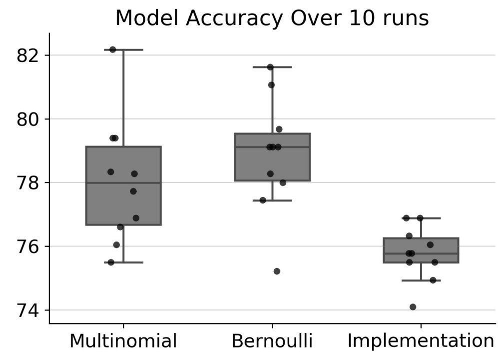

Figure 1: Model Accuracy over 10 runs with random seed

Model Performance		
Classifier	Mean Acc	Std Dev
Dummy Classifier	49.47%	+/- 1.87
sk-learn Multinomial Naive Bayes	78.03%	+/- 1.87
sk-learn Bernoulli Naive Bayes	78.86%	+/- 1.72
Self Implemented Naive Bayes	75.77%	+/- 0.81

Figure 2: Mean Accuracy and Standard Deviation Over 10 Runs

Our model is trained and tested along with the out-of-the-box implementations of the Naïve Bayes classifier on 10 different random seeds to get an aggregate approximation of the performance of my implementation. **Figure 1** shows the distribution of model accuracy over the 10 runs of the sentiment classification. On inspection, the out of the box classifiers were on average slightly more accurate than my implementation. **Figure 2** shows the mean accuracy and standard deviation of all classifiers run over the 10 runs. The "Dummy" classifier is a classifier that will guess a prediction based on the distribution of class labels in the training data. As expected, the dummy classifier guesses correctly approximately 50% of the time.

The self implemented model is almost up to par with the out-of-the-box models in sk-learn. We can see that the Bernoulli Naïve Bayes performs slightly better than the Multinomial model, and both are approximately 3% more accurate on average than my implementation. This is because each event model, Bernoulli and Multinomial (and consequently Gaussian) perform better in specific classification problems. In sentiment classification, the Bernoulli model performs marginally better, as it considers whether a word appears rather than the Multinomial Naïve Bayes, which considers the frequency of each word in a document.

Table 1 Constructs a Confusion Matrix (or also known as a contingency table) that measures the predictions that the classifier

	True Positive	True Negative
Predicted Positive	136	37
Predicted Negative	46	140

Table 1: Confusion Matrix for Implemented Naive Bayes (with random seed 42)

Metric	Result
Accuracy	76.88%
Precision	74.72%
Recall	74.73%
Specificity	79.10%
F_1 Score	76.62%

Table 2: Table of Metrics for Implemented Naive Bayes (with random seed 42)

made versus the actual (human made) predictions. From this confusion matrix, we are able to measure a few key metrics. **Table 2** is a table of metrics for the Naïve Bayes implementation. We can see that overall, our Naïve Bayes is a well rounded classifier, having approximately the same True Positive Rate (Precision), Specificity and Recall. No matter which metric the classifier is measured, it is approximately similarly high-performing. Often F_1 score is used as a performance metric for machine learning models as it normalizes and generalizes the overall performance of the classifier with the tradeoff of losing error detail.

5 DISCUSSIONS AND CONCLUSIONS

5.1 Results and Conclusion

Through our implementation of the Naïve Bayes Classifier, we can empirically conclude that it is a simple, but robust machine learning algorithm. Our implementation performed on par with the implementations in the production scikit-learn library. We can say that the Naïve Bayes Classifier is robust, as it significantly more accurate than guessing based solely on the prior probability of classes (Dummy Classifier), and because it generalizes well to many different classification problems.

Throughout the 10 runs performed to establish a benchmark between the Scikit-Learn implementation and our implementation, we can conclude that our implementation is close, but inferior. There are a few possibilities that may contribute to this: our implementation uses log prior and log likelihood, which are quick and efficient.

5.2 Considerations and Future Work

One limiting factor in my implementation is the reliance on the NumPy array data structure. In the Scikit-Learn implementation, the CountVectorizer module uses SciPy Sparse Matrices to represent the vocabulary. Arrays will keep all information of every cell within the array. With a bag of words model, the matrix will typically grow exponentially in size, because each unique word will add an additional n cells, where n is the number of documents in the dataset. Additionally, Natural Languages (English in this case) have a exceedingly large vocabulary, leading to an array representation of the bag of words to be extremely sparse. Thus, a NumPy array is not memory-efficient, and cannot represent a sparse matrix of vocabulary for large datasets. A future improvement to our method of implementation is to adopt the memory-efficient sparse matrix representation that the Scikit-Learn implementation utilizes.

Another limiting factor in our implementation is the time complexity. The functions written for our implementation use suboptimal indexing, data structures and functional programming methods. The training and testing phases of our implementation take much longer than the implementation in Scikit-Learn. In fact, the difference grows from seconds in small datasets (100 - 1000 rows) to a few minutes in medium-large datasets (100,000+ rows). With enormous data, we would expect this performance to be prohibitive to deploying this model at scale. With refactoring of code to reduce redundant processes, this difference in performance can be reduced, but the largest contributor is the difference in data structures, as a whole array is much slower to process than the sparse representation in the Scikit-Learn Library.

There are also several improvements we can make to the general Naïve Bayes Classifier. With a fully bayesian Naïve Bayes Classifier [9], we may be able to generalize our model to problems without the Naïve assumption of feature independence. This would allow the classifier to more closely consider the relationship of language syntax and context. Another improvement to the Naïve Bayes Classifier is the addition of negation handling to convey negating contextual clues.[5] Negation handling is the representation of the opposite meaning of words, that can be assigned when a negative modification word is present in a sentence. For example, in the sentence "I am not happy", we would add the word "NOT_happy" to the vocabulary instead of "happy". This may allow for more accurate representations of difficult contextual situations such as sarcasm or the presence of the word "not" or "but". The drawback to this approach is the potential additional space necessary to store these additional words in a matrix, and may lead to even greater sparsity within the matrix.

REFERENCES

[1] Berna Altınel and Murat Can Ganiz. 2018. Semantic text classification: A survey of past and recent advances. *Information Processing Management* 54, 6 (2018), 1129 – 1153. https://doi.org/10.1016/j.ipm.2018.08.001

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018). arXiv:1810.04805 http://arxiv.org/abs/1810.04805

[3] Nir Friedman, Dan Geiger, and Moises Goldszmidt. 1997. Bayesian Network Classifiers. *Machine Learning* 29, 2/3 (1997), 131–163. https://doi.org/10.1023/a:1007465528199

[4] Alec Go, Richa Bhayani, and Lei Huang. 2009. Twitter Sentiment Classification using Distant Supervision. *CS@Stanford* (2009). http://help.sentiment140.com/

[5] Daniel Jurafsky and James H. Martin. 2019. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition* (3 ed.). Prentice Hall.

[6] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2018. *Introduction to Information Retrieval.* Cambridge University Press, Chapter 13: Text Classification & Naive Bayes, 253–287.

[7] Andrew McCallum and Kamal Nigam. [n.d.]. A Comparison of Event Models for Naive Bayes Text Classification. ([n. d.]). AAA-I 98 workshop on Learning for Text Categorization.

[8] G. Singh, B. Kumar, L. Gaur, and A. Tyagi. 2019. Comparison between Multinomial and Bernoulli Naïve Bayes for Text Classification. In *2019 International Conference on Automation, Computational and Technology Management (ICACTM).* 593–596. https://doi.org/10.1109/ICACTM.2019.8776800

[9] Shuo Xu. 2018. Bayesian Naïve Bayes classifiers to text classification. *Journal of Information Science* 44, 1 (2018), 48–59. https://doi.org/10.1177/0165551516677946