Building a sentiment analysis system with Naïve Bayes

Shivangi Patel

April 2021

1 Introduction

Sentiment prediction of online reviews and tweets through text analysis is a popular application of Natural Language Processing (NLP). In this study, we we perform sentiment analysis on IMDB reviews dataset using Naive Bayes algorithm. The goal of the study is to perform and analyse the effect of various text pre-processing method, build a Naive Bayes classifier and evaluate its performance using F1-measure; both without using any third party libraries.

2 Methods

2.1 Dataset

IMDB reviews dataset is used in this study. The dataset contains train and test sets with 75000 and 25000 observations respectively. Each observation contains a *review* and corresponding *label* feature which can be either a *positive* or *negative* sentiment. The train set contains 50000 observations with unknown labels; and therefore dropped prior to further analysis. Final number of observations in train set are 25000, with both classes balanced equally as shown in Figure 1.

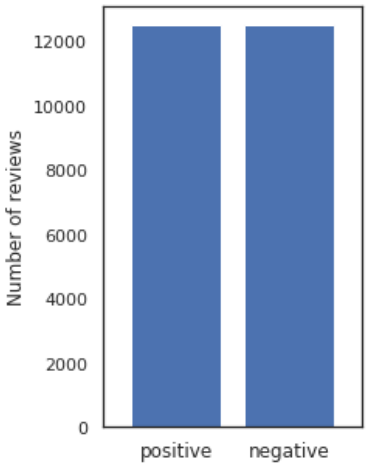

Figure 1: Class distribution in train set

2.2 Text Pre-processing

Text pre-processing [1] [2] is vital in most NLP applications as text data in raw form is not very useful. It consists of a series of steps that convert the raw input text into a format that can be used by machine learning algorithms. These steps are usually not transferable from one task to another, as they are domain specific. Following are the text pre-processing steps in their respective order, that have been used in this study to process text in train and test sets.

2.2.1 Lower casing

Lower casing allows same words with different cases to map to the same lower case word. This step helps to reduce sparsity in vocabulary. For example, as shown in Figure 2, all the raw words in column 1 are mapped to the corresponding lowercase word in column 2.

Raw	Lowercase
Director director DIRECTOR	director

Figure 2: Text Lower casing

2.2.2 HTML tags

Text data scraped from web usually has HTML tags, which do not add any value to the data as they only enable proper browser rendering. HTML tags in the *review* column are removed using regular expression operations from Python library *re*.

2.2.3 Numbers

Numbers in the form of dates, currency etc. also do not add value to the task of sentiment analysis and therefore are removed using Python library *re*

2.2.4 Punctuations

Python library *string* is used to remove punctuation characters in the *review* column.

2.2.5 Tokenize

Word level tokenization is a process of splitting a sentence into words. The tokens can then be used as input for text normalization and cleaning process. Tokenizaton of the text is performed using *word_tokenize* module of *nltk* library.

2.2.6 Stop words

Set of most common words such as 'as', 'in', 'then', 'there', 'because' etc. do not contain any information and therefore do not add any value for a task like sentiment analysis. Such commonly used words known as stop

words can be removed enabling the model to only consider the key features. Stop words are removed using the *stopwords* module of *nltk* library.

2.2.7 Text Normalization

Stemming and Lemmatization are common normalization techniques used to reduce text pattern variance and improve generalization over unseen words. Stemming is the process of reducing inflection in words to their root forms such as mapping a group of words to the same stem even if the stem itself is not a valid word in the Language. Lemmatization, unlike Stemming, reduces the inflected words properly ensuring that the root word belongs to the language. In Lemmatization root word is called Lemma. A major difference between the two is that the stem may not be an actual word, whereas lemma is an actual language word. Stemming follows an algorithm with steps to perform on the words, which makes it faster. Whereas, in Lemmatization, a corpus is used to supply lemma which makes it slower than stemming. Also, one needs to define a parts-of-speech to get the proper lemma.

In this study, we compare and analyse the differences in classifier performance by either of the text normalization techniques - Stemming and Lemmatization. Python *nltk* library is used for both the techniques.

2.3 Naive Bayes Classifier

2.3.1 Introduction - Multinomial Naive Bayes and Binary Naive Bayes

Naive Bayes [3] classifiers are linear classifiers based on Bayes' theorem. The term *naive* comes from the assumption that the features in a dataset are mutually independent, which is often violated in most practical cases. However, the classifier still tends to perform well under this unrealistic assumption.

Bayes' theorem is at the core of Naive Bayes classification. It can be written down as shown in equation 1,

$$\text{posterior probability} = \frac{\text{conditional probability} \times \text{prior probability}}{\text{evidence}} \tag{1}$$

In the context of a classification problem, posterior probability can be interpreted as "What is the probability that a particular sample belongs to class i given its observed feature values?" A general notation of posterior probability can thus be written as,

$$P(c_j \mid x_i) = \frac{P(x_i \mid c_j) \times P(c_j)}{P(x_i)} \tag{2}$$

where,

- x_i is the feature vector of sample i, $i \, \epsilon \, \{1, 2, ..n\}$
- c_j is the notation of class j, $j \, \epsilon \, \{1, 2, ..m\}$
- and $P(x_i \mid c_j)$ is the probability of observing sample x_i given that it belongs to class c_j

The objective function of Naive Bayes is to maximize the posterior probability given the training data in order to formulate the decision rule. An assumption of Naive Bayes classifiers is the conditional independence of features. Under this naive assumption, the class-conditional probabilities or (likelihoods) of the samples can be directly estimated from the training data instead of evaluating all possibilities of x. Thus, given a d-dimensional feature vector x, the class conditional probability can be calculated as follows:

$$P(x \mid c_j) = P(x_1 \mid c_j) \times P(x_2 \mid c_j) \times P(x_3 \mid w_j)...P(x_d \mid c_j) \tag{3}$$

Thus,

$$P(x_i \mid c_j) = \frac{Nx_i, c_j}{N_{c_j}}, \ (i = (1, ..d)) \tag{4}$$

where,

- N_{x_i,c_j} : Number of times feature x_i appears in samples from class c_j
- N_{c_j} : Total count of all features in class c_j

The prior probabilities are class priors, which describe the general probability of encountering a particular class. Thus,

$$P(c_j) = \frac{N_{c_j}}{N_s} \tag{5}$$

where,

- N_{c_j}: Count of all samples in class c_j
- N_s: Count of all samples in training set

Although the evidence term is also required to accurately calculate the posterior probabilities, it can be removed from the decision rule, since it is merely a scaling factor.
The algorithm explained above is the classic **Multinomial Naive Bayes**. A small variation where each feature is considered to be binary valued, is know as **Binary Naive Bayes**. It differs to Multinomial Naive Bayes with the context that the presence or absence of a feature matters more than its frequency.

2.3.2 Implementation of Naive Bayes classifier

To implement Multinomial Naive Bayes classifier, a bag of words representation of text can be used. For a task like sentiment analysis, a bag of words model ignores the order of words, but records the occurrence frequency of each word in the training set. This generates a vocabulary which is nothing but a $\mid V \mid$ dimensional vector, with each dimension indicating the number of occurrence of a particular word in the train set. In this study, *CountVectorizer* module of *sklearn* library is used to generate the vocabulary from the train set.
Also, in practise it is common to use logarithm of probability as shown in below equation,

$$log \prod_i P(x_i \mid c) \times (P(c)) = \sum_i log(P(x_i \mid c)) + log(P(c)) \tag{6}$$

The terms with the same words can be merged together such that,

$$\sum_i log(P(x_i \mid c)) + log(P(c)) = \sum_{k \epsilon |V|} n_k log(P(x_k \mid c)) + log(P(c)) \tag{7}$$

where, n_k is the number of occurrence of the k-th word in the vocabulary. It can be 0, or non-zero. The parameters of Naive Bayes classifier estimated from the training set are thus,

$$P(x_k \mid c) = \frac{count(x_k \text{ in class c})}{\sum_x count(x \text{ in class c})} \text{ and } P(c) = \frac{N_c}{N_s} \tag{8}$$

where,

- x_k is the k-th word in $\mid V \mid$
- N_c is the number of words in class c
- N_s is the number of words in training set

Another practical consideration is the problem of encountering zero-probability for a word $P(x_k \mid c)$, resulting in an overall zero posterior probability. A solution to this problem is performing a smoothing operation. In this study, we perform **add-k smoothing** with $k = 1$, changing the parameter as follows,

$$P(x_k \mid c) = \frac{count(x_k \text{ in class c}) + 1}{\sum_x count(x \text{ in class c}) + \mid V \mid} \tag{9}$$

We use $k = 1$ for smoothing operation with Multinomial Naive Bayes. For implementation of Binary Naive Bayes, we binarize the output of the *Countvectorizer* using the *binarize* module of sklearn library. With Binary naive Bayes, to avoid over-smoothing we use $k = 0.2$.

2.4 Performance evaluation using F1-measure

Figure 3 shows a confusion matrix for a typical binary classification problem. A commonly used metric for

		Predicted	
Actual		Positive	Negative
	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Figure 3: Confusion matrix

classification problems is Accuracy [4] which is a proportion of correct classifications among all classifications, calculated as follows,

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FN} + \text{FP} + \text{TN}} \tag{10}$$

Accuracy is misleading when the the class proportions are heavily imbalanced, which is often true in most real world problems. Two other metrics called Precision and Recall can be used in such cases.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{11}$$

Precision [4] is the proportion of actual positives over all the predicted positives. It is a good measure to determine when the cost of False Positives is high. On the other hand, Recall [4] calculates the proportion of actual positives captured by the model. It is a model metric when there is a high cost associated with False Negatives.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{12}$$

Another metric called F-measure [4] can be used when one wants to seek a balance between Precision and Recall; calculated as follows,

$$F_\beta = \frac{(\beta^2 + 1) \times \text{Precision} \times \text{Recall}}{\beta^2 \times \text{Precision} + \text{Recall}} \tag{13}$$

The β parameter differently weighs the importance of Recall and Precision, based on the needs of an application. Values of $\beta > 1$ favor Recall, while values of $\beta < 1$ favor Precision. When $\beta = 1$, Precision and Recall are equally balanced; this is the most frequently used metric, and is called F1-measure. In this study, we use F1-measure to evaluate the performance of the classifier.

3 Results and Discussion

Figures 4 and 5 show word clouds [5] of positive and negative classes obtained after text pre-processing and normalization with Lemmatization method. Only top 200 words with highest frequency from both classes have been used for word cloud generation. Amongst these 200 words, top 5-6 words with frequency > 11000 are eliminated as they are common in both classes.

Figure 4: Positive class word cloud

Figure 5: Negative class word cloud

Although there are a few common highly frequent words - *get, see, good* in both classes; it can be observed that words *great, like* and *make* are much more frequent in positive class. Similarly, words *bad,even,dont* are

more frequent in the negative class.

Table 1 shows the results obtained with sklearn implementation of Multinomial Naive Bayes classifier compared with implementation of Naive Bayes classifier and calculation of F1-score from scratch. The F1-score values match perfectly for both implementations.

Naive Bayes Classifier	Train F1-measure (%)	Test F1-measure (%)
sklearn implementation	91.0	80.7
Implementation from scratch	91.0	80.7

Table 1: Naive Bayes classifier implementations

Table 2 shows a comparison of the two text normalization methods - Stemming and Lemmatization. Although the vocabulary size is slightly larger with Lemmatization, there is no significant difference between the classifier performance with either of the normalization methods.

Text normalization method	Vocabulary size	Train F1-measure (%)	Test F1-measure (%)
Stemming	90587	91.0	80.7
Lemmatization	104050	91.5	81.0

Table 2: Text normalization methods

Table 3 shows a comparison of F1-measure obtained on train and test sets with Multinomial Naive Bayes and Binary/Clipped Naive Bayes. The performance of Binary NB improved on the training set, however there is no improvement with the test set.

Classifier	Train F1-measure (%)	Test F1-measure (%)
Multinomial Naive Bayes	91.0	80.7
Binary Naive Bayes	94.9	80.6

Table 3: Naive Bayes classifiers

4 Conclusion

In this study, we perform sentiment analysis of IMDB reviews dataset. After using a series of text pre-processing steps and training a Naive Bayes model, we classify the reviews in test set with F1-score of 81 %. Our implementation for Multinomial Naive Bayes and F1-score calculation matches perfectly with that sklearn library implementations. We do not observe any significant differences with the use of text normalization methods - Stemming and Lemmatization. Also, the classifiers Multinomial Naive Bayes and Binary Naive Bayes models show similar prediction performances on test set.

References

[1] K. Ganesan. All you need to know about text preprocessing for nlp and machine learning. [Online]. Available: https://www.kdnuggets.com/2019/04/text-preprocessing-nlp-machine-learning.html

[2] S. Polamuri. 20+ popular nlp text preprocessing techniques implementation in python. [Online]. Available: https://dataaspirant.com/nlp-text-preprocessing-techniques-implementation-python/

[3] S. Raschka. Naive bayes and text classification. [Online]. Available: https://sebastianraschka.com/Articles/2014_naive_bayes_1.html

[4] K. ping Shung. Accuracy, precision, recall or f1? [Online]. Available: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9

[5] D. Wu. Word cloud generation. [Online]. Available: https://www.datacamp.com/community/tutorials/wordcloud-python