SENTIMENT ANALYSIS WITH NAÏVE BAYES CALSSIFIER FROM SCRATCH

Melinda Dong | a1870910@adelaide.edu.au

1. OVERVIEW

This project implements a naïve bayes classifier from scratch on MDB movie review datasets to predict weather a review is positive or negative.

Since this project is mainly focus on naïve bayes classifier implementation, unlabeled samples are simply discarded. After removing unlabeled samples, there are 50,000 samples in total, half training data, and half testing data. Within the training data, half is positive and half negative. Test data is the same. The samples are very balanced.

The baseline model is a binary naïve bayes classifier built from scratch, with k = 1. The final accuracy of the baseline model is 0.825, and the F1 score is 0.830.

There're 2 experiments, the first is to find out how different k value affect the result when doing k-smoothing. The result shows the improvement from not using k-smoothing to using k-smoothing is significantly, the F1 score improved from xxx to xxx. Increasing the k value after k = 1 can still improve the result a little bit, but not too obvious, in our case, the best F1 score is 0.838 when k = 20(the biggest k value in our experiment).

The second experiment is to find out how Byte Pair Encoding affect the result. After BPE, the vocabulary size shrinks from 30948 to 9918. The final accuracy of the BPE-NB model is 0.623, and the F1 score is 0.708, which is not as good as the baseline model.

2. TEXT PREPROCESSING

Text preprocessing is the process of cleaning and transforming raw text data into a format that is suitable for analysis and machine learning models. It is necessary because raw text data often contains noise, irregularities, and inconsistencies. By preprocessing the text, we can reduce the variety of the words and improve the effectiveness of the model.

Here is how the text preprocessing implemented in this project:

1. Remove the punctuations and numbers. I used a regular expression *r'[^A-Za-z] +'* to do so, it means removing any non-alphabetic characters from the reviews, because the numbers and all kinds of punctuations doesn't provide much information and makes the computing complicated, it's better to remove them.

2. Convert the review to lower case. Because the computer will see ‘Happy’ and ‘happy’ as 2 different words, but they have the same meaning, so convert every text in lower case can avoid this happen.

3. Remove the stop words. Stop words are words that are commonly used in a language but do not convey much meaning and can be safely removed from the text without affecting its overall semantic meaning. Examples of stop words in English include "a", "an", "the", "and", "or", "in", "of", "to", etc.

Removing stop words from text data during text preprocessing reduce the size of the data and make it more efficient to process. And improve the accuracy and effectiveness of sentiment analysis by reducing the noise. In this case, I call the ‘stopwords’ method from NLTK library to remove the English stop words.

4. Tokenize the words. Tokenization is the process of breaking down a text into individual words or phrases, known as tokens. It is a critical step in text preprocessing as it helps to convert unstructured text into a structured format that can be easily analyzed by models. It involves splitting a sentence into individual tokens, usually by identifying spaces or other delimiters. In this case, I call ‘word_tokenize’ method from NLTK library to do the tokenization.

5. Stem the words. Stemming is the process of reducing a word to its root or base form by removing its suffixes. For example, the stem of the words "running", "runner", and "runs" is "run". It can help to group together variations of the same word and reduce the number of unique words in a corpus, making it easier for model to process. In this case, I call ‘PorterStemmer’ method from NLTK library to do the Stemming.

Note that I didn’t lemmatize the text, Lemmatization involves reducing words to their base or dictionary form, it has some similar function with stemming. To reduce the computing, I choose the stemming instead of lemma.

After preprocessing, the whole train data has around 20,000,000 tokens. All the following training and estimation are all based on those preprocessed tokens.

3. NAÏVE BAYES CLASSIFIER (BASELINE)

Naive Bayes is a probabilistic machine learning algorithm that is used for classification tasks. It is based on Bayes' theorem, which describes the probability of an event based on prior knowledge of conditions that might be related to the event.

Here is the basic Bayesian equation:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

Where :

P(y) is the prior distribution; it shows how likely to observe each class.

P(x|y) is the likelihood conditioned on each class.

In our case, Bayesian assume features are conditional independent, which means:

$$P(A, B|C) = P(A|C)P(B|C)$$

This assumption makes the algorithm simple, fast, and easy to implement, but it can also lead to suboptimal performance when the features are correlated.

In our case, our goal is to know given the tokens in the review, what's the probability that review is negative/positive, and sign that review into the higher probability class, therefore predict whether a review is negative or positive.

$$\hat{c} = \text{argmax}_c \text{P(c|d)} = \text{argmax}_c \text{ P(d|c)P(c)} = argmax_c \prod_{i \in positions} P(w_i|c)P(c)$$

Where:

c means classes (positive or negative)

d means document (each review)

w means each token in the review.

To reduce computing, we usually use logarithm of probability in practise.

$$log \prod_{i \in positions} P(w_i|c)P(c) = \sum_{i \in positions} logP(w_i|c) + \log\ P(c)$$

The specific implementation are as follows:

1. Build a bag of words. Create a dictionary named as 'vocab', loop through every review in training dataset, and loop through every token in each review, if the token is in vocabulary, count its' frequency, otherwise, add the token to the vocabulary. At the end, we have our vocabulary, with the key as the unique tokens appear in training data and the value is the corresponding frequency.

To reduce the size of the vocabulary and easier for computing, the tokens with only 1 appearance have been removed. The final vocabulary has 30948 tokens.

2. In train dataset, calculate the prior probability and the likelihood for each token in each class. Build a naïve bayes model with default k value as 1(k-smoothing will be explained further in a subsequent paragraph).

Calculate the prior probability of each class, which is just count how many reviews are positive/negative, then divided by the total amount of reviews in training set. In this case, the data is very balanced, we have positive and negative reviews as half and half, so the prior probability for both positive and negative is 0.5.

Calculate the likelihood for each token in each class. Loop through the vocabulary, for each token, count how many times it appears in positive training reviews (add k value), then divided by the total amount of tokens in positive reviews (add the k * vocabulary size). Do the same operation with negative class as well. And store the result in 'cond_prob' dictionary.

For each token in vocabulary, we know given the review is positive/negative, what's the probability that token appear.

3. Apply the model on test dataset and get the prediction. We do the same text pre-processing on our test data. There are 2 assumptions, one assumes the review is positive, another assumes it's negative. According to the logarithm formula mentioned above, loop through every review in testing dataset, and loop through every token in the review. Sum up the likelihood of that token given positive review into the 'pos_prob' and sum up the likelihood of that token given negative review into the 'neg_prob'. At the end, compare which class has higher probability, then sign the review to the test review into that class.

Here is the confusion matrix of our baseline model:

Predicted	negative	positive
Actual		
negative	9929(TN)	2571(FP)
positive	1813(FN)	10687(TP)

4. EVALUATION (F1 SCORE)

We use F1 score as the main estimation in this project. Because F1 score provides a more comprehensive evaluation of model performance. It can take into account the cost associated with different types of errors. F1 score can measure the balance between precision and recall of a model. To put it simply, the precision rate measures how many of the samples predicted by the classifier as positive classes are true positive classes, while the recall rate measures how many true positive class samples the classifier can find.

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

F1 score is a weighted average of precision and recall, where the F1 score reaches its best value at 1 and worst at 0. The formula for calculating F1 score is:

$$\frac{1}{F_\beta} = \frac{1}{1 + \beta^2}\left(\frac{1}{P} + \frac{\beta^2}{R}\right)$$

Where:

β is the weight, if β > 1, recall is more important than precision.

P means precision.

R means recall.

In this, we don't have specific preference to precision or recall, so we set β as 1 and the formula looks like: F1 score = 2 * (precision * recall) / (precision + recall).

The baseline model gives a F1 as 0.830.

5. EXPERIMENTS

There are 2 experiments, the first one is to test how different k value would affect the result, the second one is to test how BPE method would affect the result.

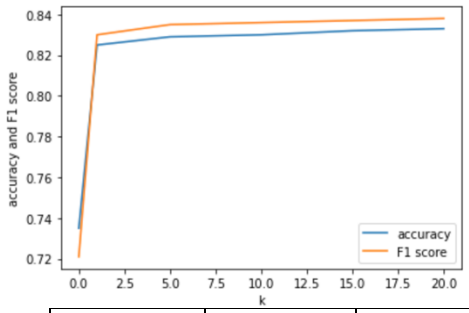
1. Different K values

k-smoothing is a technique used to handle zero probabilities in the dataset. When calculating the probabilities of a particular token given a class label, there may be cases where the count of that token in the training data is zero for a particular class label. This can lead to a probability of zero, which can cause problems when performing Naive Bayes classification.

So, by adding a small constant value k, to the count of each token for each class label. Since for each token we add k, so in total we add (k * vocabulary size) number of counts, that's why we add (k * vocabulary size) to the denominator.

$$P(token|class) = \frac{count\ of\ tokens\ in\ class + k}{count\ of\ all\ tokens\ in\ class + k * vocabulary\ size}$$

Here is the plot of the result with different k values:



K values	0	1	5	10	15	20
accuracy	0.735	0.825	0.829	0.830	0.832	0.833
F1 score	0.721	0.830	0.835	0.836	0.837	0.838

The plot shows apply k-smoothing is much better than without k-smoothing. Increasing the k value from 1 to bigger ones can slightly improve the result as well but the effect is not significant.

2. Byte Pair Encoding

Byte Pair Encoding (BPE) is originally a data compression algorithm. It has been adapted for use in natural language processing a lot because it can effectively handle out-of-vocabulary (OOV) words that are not present in the original vocabulary by

breaking down words into subword units. Therefore, the model can learn to better generalize to new words and improve its performance on downstream tasks.

BPE works by first splitting a given corpus of text into individual characters. It then iteratively merges the most frequent pair of adjacent characters/tokens, replacing them with a new token that represents the concatenation of the two original tokens. This process is repeated until a pre-specified vocabulary size is reached or until no more merges can be made.

The specific operation process is referred to this web:

https://huggingface.co/course/chapter6/5?fw=pt#tokenization-algorithm

1. normalise and pre-tokenize the training review, use numbers and puncatuation remove, lower case, use 'word_tokenize' to tokenise it then remove stopwords. Since the original tutorial is replicating a BPE tokenizer (like GPT-2) and using gpt2 tokenizer for the pre-tokenization. In order to match the format, I overlay the pre-processing by using gpt2 tokenizer as well.

2.build a 'word_freqs' count the unique words from training data with the value as their frequency.

3. Get the unique alphabet, in our case, from A-Za-z, initialise our vocabulary with unique alphabet. Also add the special tokens used by the model at the beginning of that vocabulary. In the case of GPT-2, the only special token is "<|endoftext|>"

4. split the words into individual characters, loop through the 'word_freqs', one pair means the two adjacent letters, count the unique pairs, and record their frequency, find the pair with highest frequency, and merge it into one token, add it into our vocabulary, also add the pairs and its merged token into 'merges' (This is our training BPE codes). Repeating the process till the size of vocabulary is desired. In our case, 10000 vocabularies.

5. using the BPE codes we learned from training data, apply it to our train and test review by looping through the split words, if there's pair that match the 'merges', merge the pair. This step requires a lot of computing, that's why I chose the vocabulary size as 10000 not bigger.

6. apply them with the baseline Naïve bayes model.

Note that after BPE, the bag of words for training data only has 9918 tokens (removed some infrequent tokens), which is only around 32% of the baseline vocabulary. For example, the first sample of test review has 504 tokens before BPE and has 97 tokens after BPE.

Here is the confusion matrix of the BPE - NB model:

Predicted	negative	positive
Actual		
negative	4130(TN)	8370(FP)
positive	1055(FN)	11445(TP)

The accuracy is 0.623 and the F1 score is 0.708.

The results of the BPE experiment are not as good as the baseline, there are some possible reasons:

1. BPE may not be the best text pre-processing technique for sentiment analysis: BPE is typically helpful when dealing with out-of-vocabulary words. However, for sentiment analysis, it is possible that BPE does not add much value, as the sentiment of a sentence is often determined by the overall meaning of the words in the sentence.

2. Due to the limitation of computing, the hyperparameters of the BPE model (10000 BPE codes) may not be optimal and the The BPE model may not have been trained properly so the subword units it generates may not be very helpful for sentiment analysis.

3. The BPE subword units may not be compatible with the Naive Bayes model: Naive Bayes assumes that the features (i.e., words or subwords) are independent of each other, which may not be true for the subword units generated by BPE. If the subword units generated by BPE are not compatible with the Naive Bayes model, the performance of the model may be negatively affected.

6. CONCLUSION

This study focused on implementing a Naïve Bayes classifier from scratch to predict the sentiment of movie reviews in the MDB dataset. The study consisted of two experiments: (1) exploring the effect of k-smoothing on the classifier's performance, and (2) examining the impact of Byte Pair Encoding (BPE) on the classifier's performance.

The results show that k-smoothing can significantly improve the classifier's performance but keep increasing k value can only yield a slightly better performance.

However, BPE did not lead to improved performance in this study, it reduced the vocabulary size but negatively impacted the classifier's accuracy, with a final F1 score of 0.708, lower than the baseline model's F1 score of 0.830.