Don Castillo
001224541

CPSC 4660
May 1, 2022

Movie Review Classifier Using Naïve Bayes Classification Strategy

Naïve Bayes in a Nutshell

Naïve Bayes Classifier is a machine learning model used in categorizing text or datasets based on their certain features. It analyzes a set of words in a text and predicting the category of that text based on the probability of these words from appearing in a particular text pre-assigned in categories. Hence, Naïve Bayes is a supervised classification. Before it can predict, the classifier needs to train pre-classified texts.

This concept of Naïve Bayes has been applied in many text categorization applications. Spam detection, for example, detects whether an email is spam or not. By training pre-classified emails, we then determine that the phrases “Dear Winner or “online pharmaceutical” are more likely to appear in spam emails hence could categorize a new email bearing these phrases as spam. In search engines, we could apply the Naïve Bayes to view documents associated with the search terms entered. In journalism, the classifier can group news articles together based on their subjects - sports, politics, business, or crime.

For opinion-based analysis, we often determine the sentiment of the writer based on the ratings given. In product reviews, a 1 out of 5 star means the writer feels bad about the product while a perfect 5 means the writer is satisfied with it. Though this method of eliciting sentiment measures the degree of the writer’s attitude, it would still require that the writer explicitly enter a rating that s/he believes represent his/her sentiment. Does the rating of 4 mean not very good? Does 3, the middle rating, mean being neutral?

Naïve Bayes can implicitly determine the sentiment by analyzing a text response from the writer, not just ratings. Based on the training of pre-classified responses, we can deduce that the existence of words “awful” and “disgusting” may signal negative sentiment and words “beautiful” and “excellent” may signal positive sentiment.

Movie Review Classifier App

For this project, I created a movie review classifier app that uses the concept of Naïve Bayes classification to predict the sentiment of the user about a movie. The user is presented with a text field where s/he will input the movie title and his/her review. The app then calculates all the classes (positive or negative) scores for that review. Whichever class gets the highest score will become the predicted class of that review.

The app is written primarily in JavaScript. Created in Node environment, I used Express as the back-end framework. It would specify what page to render or what API to run based on the URL or endpoint. I used Embedded JavaScript Templating (EJS) as the front-end framework. Since this is a mini app and the data involved are unstructured and not relational, I used MongoDB as the database. I used Mongoose to model the data and create basic schemas.

A. Importing and Data Processing

First, we needed to import the files to be trained by the classifier. For this app, I used the datasets published by Maas, et al. (2011). It was a collection of 50000 pre-classified movie reviews (25000 for training and 25000 for testing) from IMDb. The training files were composed of 12500 positive and 12500 negative reviews. For efficiency, I only considered 2500 positive and 2500 negative reviews.

For the purpose of explaining the implementation of this app, consider the following pre-classified dataset example (this is only for demonstration and not the actual testing files used in the project):

CLASS	RAW TEXT
POS	“Excellent. Actors are so good. Actors did a great job!”
POS	“Movie is powerful. I recommend you to watch it.”
POS	“Most fun film I have seen.”
NEG	“Very predictable. Predictable plot. Predictable ending.”
NEG	“Fun film but very predictable. Do not watch it.”

Table 1. Sample training dataset

While being imported, each text file was read and processed. The texts were first tokenized into an array of words. The words were then transformed to lowercase. Then punctuation marks and numerical figures were removed. Then, stop words such as “at”, “in”, “on”, etc. were also

removed since they do not give significant meaning. Lastly, the words were stemmed so that two words with the same meaning but different representations are recognized by the app as just one word. For example, “beautiful” and “beauty” are stemmed to “beauti”. This would also recognize words in different tenses as same word like “watch”, “watching”, and “watched”. In removing and stemming words, I used JS libraries *stopword* and *porter-stemmer*. After all these steps, we would now have the processed version of the raw text. The processed text is converted back to a string of text.

We needed to get the vocabulary of the processed text. Vocabulary is a set of unique words. Then we counted the number of times each word in the vocabulary appeared in the processed text.

CLASS	
POS	**RAW**: “Excellent. Actors are so good. Actors did a great job!” **PROCESSED:** “excel actor good actor great job” **VOCABULARY:** [excel, actor, good, great, job] **WORD_FREQ**: {excel: 1, actor: 2, good: 1, great: 1, job: 1}
POS	**RAW:** “Movie is powerful. I recommend you to watch it.” **PROCESSED:** “movi power recommend watch” **VOCABULARY:** [movi, power, recommend, watch] **WORD_FREQ**: {movi: 1, power: 1, recommend: 1, watch: 1}
POS	**RAW:** “Most fun film I have seen.” **PROCESSED:** “fun film seen” **VOCABULARY:** [fun, film, seen] **WORD_FREQ**: {fun: 1, film: 1, seen: 1}
NEG	**RAW:** “Very predictable. Predictable plot. Predictable ending.” **PROCESSED:** “predict predict plot predict end” **VOCABULARY:** [predict, plot, end] **WORD_FREQ**: {predict: 3, plot: 1, end: 1}
NEG	**RAW:** “Fun film but very predictable. Do not watch it.” **PROCESSED:** “fun film predict watch” **VOCABULARY:** [fun, film, predict, watch] **WORD_FREQ**: {fun: 1, film: 1, predict: 1, watch: 1}

Table 2. Processed dataset

After the files were imported and processed, we then needed to get the frequency of words per class. This was like getting the word frequency of each file like we did previously but this time it would be the whole class. This was accomplished by iterating through the processed files for each class, and appending the word frequency of file one after another. For example, if the word frequencies of two files belonging in class “positive” are `{cat: 2, dog: 1}` and `{cat: 1, frog:1}` then the word frequency of the class “positive” would be `{cat: 3, dog: 1, frog: 1}`. We would also need to count the number of files per class.

CLASS	
POS	**WORD_FREQ:** {excel: 1, actor: 2, good: 1, great: 1, job: 1, movi: 1, power: 1, recommend: 1, watch: 1, fun: 1, film: 1, seen: 1} **DOC_COUNT:** 3
NEG	**WORD_FREQ:** {predict: 4, plot: 1, end: 1, fun: 1, film: 1, watch: 1} **DOC_COUNT:** 2

Table 3. Evaluated class dataset

After getting the word frequencies per class, we then needed to get the word frequencies of the whole corpus. Corpus refers to all files regardless of their class. The word frequencies of corpus are just the combination of the word frequencies of "positive" and "negative" classes. We also needed to get the number of all files which is the sum of DOC_COUNT values of the two classes.

CORPUS	**WORD_FREQ:** {excel: 1, actor: 2, good: 1, great: 1, job: 1, movi: 1, power: 1, recommend: 1, watch: 2, fun: 2, film: 2, seen: 1, predict: 4, plot: 1, end: 1} **DOC_COUNT:** 5

Table 4. Evaluated corpus dataset

All the processing outputs were stored to the database to be used for training and testing.

B. Training

After the files were imported and processed, the processed data were used for training. The Naïve Bayes training algorithm implemented in this project was based on the algorithm suggested by Jurafsky and Martin (2020, p. 60):

function TRAIN NAÏVE BAYES(D, C) returns log P(c) and P(w|c)
    for each class c ∈ C
        N_{doc} ← number of documents in D
        N_c ← number of documents from D in class c
        $\text{logprior}[c] = \log \frac{N_c}{N_{doc}}$
        V ← vocabulary of D
        bigdoc[c] ← append(d) for d ∈ D with class c
        for each word w in V
            count(w, c) ← # of occurences of w in bigdoc[c]
            $\text{loglikelihood}[w, c] \leftarrow \log \frac{\text{count}(w,c)+1}{\sum_{w \text{ in } V}(\text{count }(w,c)+1)}$
    return logprior, loglikelihood, V

Algorithm 1. Training algorithm of Naïve Bayes (Jurafsky & Martin, 2020, p.60)

Following the above algorithm (Algorithm 1), N_{doc} refers to the total number of files in the corpus. In this example, we have 5. N_c refers to the number of files in each class (positive = 3, negative = 2). V refers to the vocabulary in the corpus which is just word frequency with only the keys included: example `{cat: 3, dog: 1, frog: 1} => [cat, dog, frog]`. For each word w in V, we count the number of occurrences of w in a class by looking at each class's word frequencies.

CLASS	

POS	**WORD_FREQ:**
	{
	excel: 1, actor: 2, good: 1, great: 1,
	job: 1, movi: 1, power: 1, recommend: 1,
	watch: 1, fun: 1, film: 1, seen: 1
	}
	DOC_COUNT: 3
	LOG_PRIOR: log(N_c / N_{doc}) = log(3/5) = -0.22184874961635637
	LOG_LIKELIHOOD:
	{
	excel: -1.0969100130080565, actor: -0.9208187539523752,
	good: -1.0969100130080565, great: -1.0969100130080565,
	job: -1.0969100130080565, movi: -1.0969100130080565,
	power: -1.0969100130080565, recommend: -1.0969100130080565,
	watch: -1.0969100130080565, fun: -1.0969100130080565,
	film: -1.0969100130080565, seen: -1.0969100130080565,
	predict: -1.3979400086720375, plot: -1.3979400086720375,
	end: -1.3979400086720375
	}

NEG	**WORD_FREQ:**
	{
	predict: 4, plot: 1, end: 1, fun: 1,
	film: 1, watch: 1
	}
	DOC_COUNT: 2
	LOG_PRIOR: log(N_c / N_{doc}) = log(2/5) = -0.3979400086720376
	LOG_LIKELIHOOD:
	{
	excel: -1.3802112417116061, actor: -1.3802112417116061,
	good: -1.3802112417116061, great: -1.3802112417116061,
	job: -1.3802112417116061, movi: -1.3802112417116061,
	power: -1.3802112417116061, recommend: -1.3802112417116061,
	watch: -1.0791812460476249, fun: -1.0791812460476249,
	film: -1.0791812460476249, seen: -1.3802112417116061,
	predict: -0.6812412373755872, plot: -1.0791812460476249,
	end: -1.0791812460476249
	}

Table 5. Trained dataset

C. Testing

In testing, this is when the app would ask the user to enter his/her review. The text inputted by the user in the text field would undergo tokenization and processing, same as what was done when importing test files - transforming words to lowercase, removing stop words, and stemming words. Consider the following hypothetical user-generated review:

"Wrong Turn is a good film but very predictable."

After text processing, this should be the output:

CLASS	
?	**RAW**: "Wrong Turn is a good film but very predictable." **PROCESSED:** "wrong turn good film predict"

Table 6. Processed user-generated text

Then we test the processed dataset using the Naïve Bayes algorithm for testing as suggested by Jurafsky and Martin (2020, p. 60):

```
function TEST NAÏVE BAYES(testdoc, logprior, loglikelihood, C, V) returns best c
    for each class c ∈ C
        sum[c] ← logprior[c]
        for each position i in testdoc
            word ← testdoc[i]
            if word ∈ V
                sum[c] ← sum[c] + loglikelihood[word, c]
    return argmax_c sum[c]
```

Algorithm 2. Testing algorithm of Naïve Bayes (Jurafsky & Martin, 2020, p.60)

In the algorithm above, testdoc refers to the processed text inputted by the user. For each class, we get the summation of the class's logprior and the values of loglikelihood of each word in the testdoc. We ignore the words in testdoc that do not exist in the corpus vocabulary. We then retrieve the summations or scores of classes, and whichever class has the highest score would be the predicted class of the user-generated text or review.

```
CLASS
NEG     RAW: “Wrong Turn is a good film but very predictable.”
        PROCESSED: “wrong turn good film predict”
        POSITIVE_SCORE: logprior[pos] + loglikelihood[good, pos] +
                        loglikelihood[film, pos] + loglikelihood[predict, pos] =
                        -0.22184874961635637 +
                        -1.0969100130080565 +
                        -1.0969100130080565 +
                        -1.3979400086720375 ≈ -3.81
        NEGATIVE_SCORE: logprior[neg] + loglikelihood[good, neg] +
                        loglikelihood[film, neg] + loglikelihood[predict, neg] =
                        -0.3979400086720376 +
                        -1.3802112417116061 +
                        -1.0791812460476249 +
                        -0.6812412373755872 ≈ -3.54
```

In this example, the predicted class is negative since it has higher score than positive’s. At first, the review might be perceived as both negative and positive because of the existence of the words “fun” and “predictable”, but it is predicted as negative overall because the word “predictable” or “predict” (stem) happens to exist multiple times in the training sets which means higher probability.

Conclusion

I have always been interested in data science since I took Natural Language Processing course. It is fascinating to learn different machine learning algorithms to create an intelligent app that would provide useful prediction to the user - one of these is Naïve Bayes Classification. My understanding of Naïve Bayes grew when I did this project.

As we can see, Naïve Bayes classification can really predict sentiment of a text response from a user, but there are some loopholes. Ideally, it is better to train as many texts or documents as possible to increase the accuracy of the prediction. This way, it would learn how to map different and rarely used words to a class. How do we know if the word “cliché” is negative or positive if it does not exist in the training datasets in the first place?

There was an attempt to import and train all of 25000 training files provided by Maas, et al. (2011) for this project. However, there was a limited number of files the system, where the application was ran, could handle. It would also make the performance of the app slower.

Also, since Naïve Bayes is based on probability, there is also a chance that the prediction is not what is first thought. For example, we all know that the phrase “movie is good” is positive, but if the word “good” appears in the negative training documents more than in the positive (it is possible that the word “good” would appear in a negative document if it is preceded by the word “not” such as “not good”), then the classifier may predict the phrase as negative. It would be better if there is a mechanism to determine that not-phrases like “not good” and “not beautiful” are just the same as “bad” and “ugly” respectively. This minimizes inaccuracy in sentiment prediction.

There was a realization made when testing the move review app. The classifier would work best if the context of the text being tested is the same as the context of the files trained. If the files trained are pre-classified movie reviews, then you should test it with a movie review. You cannot test it with book reviews, customer reviews, or product testimonials. Context matters.

References

Jain, R., Jain, D. K., Sharma, D., & Sharma, N. (2021, December). Fake News Classification: A Quantitative Research Description. *ACM Trans. Asian Low-Resour. Lang. Inf. Process., 21*.

Jurafsky, D., & Martin, J. H. (2020). Naive Bayes and Sentiment Classification. In D. Jurafsky, & J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing,* (3rd ed., pp. 55-73).

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011, June). Learning Word Vectors for Sentiment Analysis. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 142-150. Retrieved from https://aclanthology.org/P11-1015/