MA-697 Seminar

SENTIMENT ANALYSIS WITH NAIVE BAYES

KRITIKA GULATI (202123019)

April 22, 2022

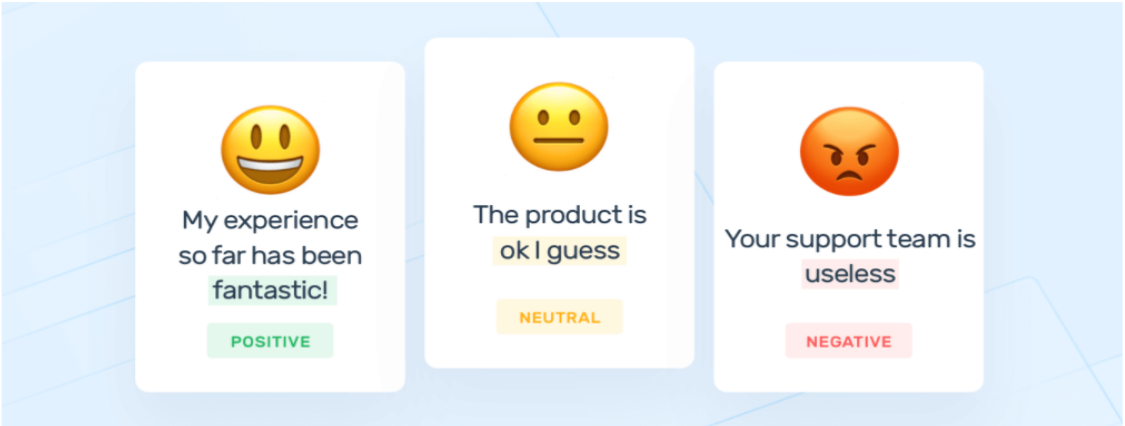
Content

- Introduction to Sentiment Analysis
- Bayes Theorem
- Assumptions
- Derivation of Analysis
- Training the Model
- Optimising the Model
- Implementation of the Model

Sentiment Analysis

Sentiment analysis is the use of Natural Language Processing (NLP), machine learning, and other data analysis techniques to analyse and derive objective quantitative results from raw text.

It uses NLP to determine whether the sentiment/emotion of the text is positive, negative or neutral. It is often performed on textual data to help businesses monitor brand and product sentiment in customer feedback, market research, etc.



Sentiment Analysis

We are going to perform Sentiment Analysis using **Supervised Machine Learning**. We will be provided with training inputs and each of these inputs will be associated with its correct output. Our model will make sense of this data, observe and analyze the relationship between the given inputs and outputs and finally predict with reasonable accuracy the output given a new unseen input.

Bayes Theorem

Bayes Theorem

Given two events A and B,

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

Here,

- $P(A \mid B)$ = Probability of event A happening given that event B happens
- $P(B \mid A)$ = Probability of event B happening given that event A happens
- $P(A)$ = Probability of event A happening
- $P(B)$ = Probability of event B happening

Assumptions

The various assumptions of this model are :

- **bag-of-words** : It implies that all the algorithm really cares about is the *word and its frequency*, that is, how many times the word has appeared in our data. The position of the word in our sentence(document) does not matter at all. We only have to keep track of the number of times a particular word appeared in our document.
- Probability of each feature (words of a sentence in this case) is **independent** of other features given a class (positive/negative).
- The events are independent of each other. That is, we do not have to take into account how two features are related to another or the probability of one feature occurring given another feature. This saves us a lot of computing power.

Derivation of Analysis

Approach to the problem

Our aim is to find the class (positive / negative sentiment) given a particular sentence (document). Suppose we have a set of possible classes C ($\{$ *positive*, *negative*$\}$ in this case). We shall approach the problem as follows :

- We will find the probability of a document being in a particular class, that is, essentially, the conditional probability of a class given a document.
- We shall iterate over all classes and find the class with maximum conditional probability that will provide us the answer.

Derivation of Analysis

$$\hat{c} = argmax_{c \in C}\{P(c \mid d)\} \tag{2}$$

where $\hat{c}$ denotes the class with maximum probability and
c = a particular class (positive/negative)
C = set of all classes ({ positive,negative } in this case)
d = a given sentence/document

Applying Bayes Theorem,

$$\hat{c} = argmax_{c \in C}\{P(c \mid d)\} = argmax_{c \in C}\{P(d \mid c)\frac{P(c)}{P(d)}\} \tag{3}$$

We simplify this equation more. While iterating over the classes our document ofcourse does not change, only the class changes; so we can safely remove the $P(d)$ from our denominator without causing any major errors.

Derivation of Analysis

Hence, the equation becomes

$$\hat{c} = argmax_{c \in C}\{P(c \mid d)\} = argmax_{c \in C}\{P(d \mid c)P(c)\} \tag{4}$$

The term $P(d \mid c)$ is called **likelihood probability**. The second term $P(c)$ is called **prior probability**. We can simplify it even further by dividing each document into a collection of features $f_1, f_2, f_3, \ldots f_n$

$$\hat{c} = argmax_{c \in C}\{P(f_1, f_2, \ldots f_n \mid c)P(c)\} \tag{5}$$

At this point of our derivation, we make use of assumption 2, that is, each feature f_i is **independent** of other features given a class. This is a very crucial step and it reduces the time complexity of our problem by a huge margin.

Derivation of Analysis

Independent Events

If two events X and Y are **independent** of each other then the probability of the events occurring together ($P(X \text{ and } Y)$) becomes :

$$P(X \cap Y) = P(X).P(Y)$$

which means

$$P(f_1 \mid c \cap f_2 \mid c) = P(f_1 \mid c).P(f_2 \mid c)$$

Thus, our final equation becomes :

$$P(f_1, f_2, \ldots, f_n \mid c) = P(f_1 \mid c).P(f_2 \mid c) \ldots P(f_n \mid c)$$

or

$$\hat{c} = argmax_{c \in C}\{P(f_1 \mid c).P(f_2 \mid c) \ldots P(f_n \mid c).P(c)\} \quad (6)$$

Derivation of Analysis

Replacing the features in our equation with w_i for the word at the i^{th} position we can re-frame our equation as follows :

$$\hat{c} = argmax_{c \in C}\{P(c). \prod_{i \in positions} P(w_i \mid c)\} \tag{7}$$

Training The Model

Calculating the Prior Probability

We will first find the number of documents belonging to each class. Finding the percentage of the documents in each class will give us the required prior probability. Let's assume the number of documents in class c is N_c. Total number of documents is assumed to be N_{total}. So,

$$P(c) = \frac{N_c}{N_{total}}$$

Training The Model

Calculating the Likelihood Probability

Our main goal is to find the fraction of times the word w_i appears among all words in all documents of class c. We first concatenate all documents with category c to use the frequency of w_i in this concatenated document to give the likelihood probability.

$$P(w_i \mid c) = \frac{count(w_i, c)}{\sum_{w_i \in V} count(wi, c)}$$

Here V is for the Vocabulary which is a collection of all words in all documents irrespective of class they belong to.

Training The Model

We however face a very unique problem at this point. Suppose the document we have as input is,
$d =$ *"I loved that movie."*
The word *"loved"* is only present in the positive class and no examples of *"loved"* is present in the negative class input. Now from our equation, we have to find the probability by multiplying the likelihood probability for each class. If we calculate out likelihood probability for the word *"loved"* for the class "negative" we get :

$$P(\text{"loved"} \mid \text{"negative"}) = 0$$

Now if we plug in this value in our eqn., the entire probability of our class *"negative"* becomes zero; no matter what the other values are.

Training The Model

To combat this problem we will introduce a constant term, **Laplace Smoothing Coefficient** , to both the numerator and the denominator . Our equation will be modified as follows :

$$P(w_i \mid c) = \frac{count(w_i, c) + a}{\sum_{w_i \in V}(count(w_i, c) + a)}$$

or

$$P(w_i \mid c) = \frac{count(w_i, c) + a}{\sum_{w_i \in V} count(w_i, c) + \mid aV \mid}$$

Here a is the **Laplace smoothing coefficient**. We usually consider its value to be 1.
Now that we have calculated our prior and likelihood probability we can simply plug it in to obtain the result.

Optimising The Model

Log Likelihood

If we apply log on both sides of our equation we can convert the equation to a linear function of the features, which would increase efficiency quite a lot. The original equation we had was :

$$\hat{c} = argmax_{c \in C}\{P(c) \prod_{i \in positions} P(w_i \mid c)\}$$

Now if we apply Logarithm on both sides we get a linear function:

$$\hat{c} = argmax_{c \in C}\{\log P(c) + \sum_{i \in positions} \log P(w_i \mid c)\} \quad (8)$$

Optimising The Model

Stop Words

Words like *the*, *a*, *an*, *was*, *when etc.* do not usually contribute to the sentiment of the statement. We can remove them entirely to streamline our model training.

Unknown Words

Every time we come across a word which is present in the test dataset but absent in the vocabulary created from the training data, it is advisable to drop the words entirely and not consider them in the probability calculations.

Optimising The Model

Binary Mutinomial Naive - Baiyes

This is a slightly modified version of the multinomial Naive-Bayes. Here we are going to **place more importance on whether a word is present or not than its frequency.** As we have already seen a single word can bring about a massive change in the sentiment of the sentence and thus it would be a logical way to disregard how many times that particular word appeared in a sentence and concentrate whether that particular word is present or not in the document.

Now, we shall use this model on a given sample dataset. Later on, we shall also see it's implementation in Python. The data for training set has been collected from *NLTK's Twitter corpus 'twitter _ samples'* in Python.

Implementation of the Model

Training Dataset

DOCUMENT	CLASS
"Boring and Predictable"	-
"Excellent Movie"	+
"Extremely Mediocre"	-
"A pathetic attempt at a romcom"	-
"Good movie with great actors"	+
"Fantastic Job!"	+

Test Dataset

- "Great Movie!"
- "This is boring and pathetic."

Implementation of the Model

Vocabulary

Word	class: positive	class: negative
Boring	0	1
And	0	1
Predictable	1	0
Excellent	1	0
Movie	2	0
Extremely	0	1
Mediocre	0	1
A	0	2
Pathetic	0	1
Attempt	0	1
At	0	1
Romcom	0	1
Good	1	0
With	1	0
Great	1	0
Actors	1	0
Fantastic	1	0
Job	1	0
	10	**10**

Implementation of the Model: First Case

Prior Probability

- $P(c = \text{'positive'}) = \frac{3}{6} = \frac{1}{2} = 0.5$
- $P(c = \text{'negative'}) = \frac{3}{6} = \frac{1}{2} = 0.5$

Likelihood Probability

- $P\ (\text{'Great'} \mid c = \text{'positive'}) = \frac{1+1}{10+20} = 0.0666$
- $P\ (\text{'movie'} \mid c = \text{'positive'}) = \frac{2+1}{10+20} = 0.1$
- $P\ (\text{'Great'} \mid c = \text{'negative'}) = \frac{1+0}{10+20} = 0.0333$
- $P\ (\text{'movie'} \mid c = \text{'negative'}) = \frac{1+0}{10+20} = 0.0333$

Implementation of the Model: First Case

Log likelihood

The log likelihood is given as :

- log (P('Great' | c = 'positive'))= log(0.0666) = −1.176
- log (P('movie' | c = 'positive')) = log(0.1) = −1
- log (P('Great' | c = 'negative')) = log(0.0333) = −1.4777
- log (P('movie' | c = 'negative')) = log(0.0333) = −1.4777
- log (P(c = 'positive')) = log(0.5) = −0.301
- log (P(c = 'negative')) = log(0.5) = −0.301

Hence, summing the respective probabilities,
$log(P(c =' positive' \mid d)) = -2.4777$ and
$log(P(c =' negative' \mid d)) = -3.2564$
$\log \hat{c} = argmax_{c \in C}\{-2.4777, -3.2564\} \implies \hat{c} =$ **POSITIVE**

Implementation of the Model: Second Case

For the second case, d = "This is boring and pathetic."
We shall be finding the sentiment of this sentence by implementing the model in *Python*.

Bibliography

- *Natural Language Processing with Classification and Vector Spaces, Coursera*
- *A Hitchhiker's Guide to Sentiment Analysis using Naive-Bayes Classifier, Towards Data Science https://towardsdatascience.com/a-hitchhikers-guide-to-sentiment-analysis-using-naive-bayes-classifier-b921c0fb694*

THE END