Melvin Mai

CSC/BCS 229

Dr. Robert Jacobs

8 December 2022

Naïve Bayes and Sentiment Classification

Naïve Bayes is a popular machine learning algorithm that is used in natural language processing tasks, including sentiment classification. Sentiment classification is the process of extracting sentiment from a piece of text and classifying whether the writer is expressing a positive or negative opinion towards some object. This object could include reviews on movies, books, products, and even political opinions and actions.

The classification process involves making a single observation, extracting some valuable features, then putting that observation into one of a set of discrete classes. In the case of sentiment classification, the goal is to take a piece of text, often called the document, and put it into one of the discrete classes. For simplification and clarity, we can have two discrete classes, positive and negative. Naïve Bayes uses Bayes' theorem to make predictions, estimating the most likely class for a document.

For sentiment classification, a multinomial Naïve Bayes classifier is used. The algorithm uses a multinomial distribution for each feature, where a feature is a word in the document. The 'naive' part of the algorithm's name comes from the fact this it makes a simplifying assumption about the data. Expressly, it assumes that all the document features are independent of each other, meaning that the presence or absence of one feature does not affect the presence or absence of any other feature.

Some intuition of the classifier is that text documents are represented as bag-of-words, ignoring the position of the words and only accounting for their frequency. Naïve Bayes is probabilistic, meaning that for a document d, out of classes c∈C, the classifier returns the class $\hat{c}$ with the maximum posterior probability given the document. This is the estimate of the correct class:

$$\hat{c} = \max_{c \in C} P(c|d)$$

We use Bayes' rule to break down the conditional probability P(c|d) into other probabilities:

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

However, this is simplified by dropping the denominator P(d) because P(d) does not change for each class. We are always asking about the most likely class for the same document d, which will always have the same probability P(d). The most probable class $\hat{c}$ given some document d is computed by choosing the class with the highest product of these two probabilities: the prior probability of the class P(c) and the likelihood of the document P(d|c):

$$\hat{c} = \max_{c \in C} \overbrace{P(d|c)}^{\text{likelihood}} \overbrace{P(c)}^{\text{prior}}$$

We represent the document as a set of features $f_1, f_2, \ldots, f_n$, and the equation becomes:

$$\hat{c} = \max_{c \in C} \overbrace{P(f_1, f_2, \ldots, f_n|c)}^{\text{likelihood}} \overbrace{P(c)}^{\text{prior}}$$

We must make two simplifying assumptions; otherwise, estimating the probabilities of every possible feature combination would require too much computation. We return to the "naïve" part of the algorithm: this is the conditional independence of the probabilities $P(f_i|c)$ are independent given the class c. This allows us to 'naively' multiply the features:

$$P(f_1, f_2, \ldots, f_n|c) = P(f_1|c) \cdot P(f_2|c) \cdot \ldots \cdot P(f_n|c)$$

The final equation is of the classifier:

$$\hat{c} = \max_{c \in C} \prod_{f \in F} \overbrace{\hat{P}(f|c)}^{\text{likelihood}} \overbrace{\hat{P}(c)}^{\text{prior}}$$

The second assumption is the bag-of-word assumption. We assume that position does not matter and that a word has the same effect on classification no matter where the word is in the document.

To begin classifying text, we must first train the algorithm on a training set of documents that have been labeled as having positive or negative sentiments. This training process involves learning the probabilities of P(c) and $P(f_i|c)$. First, we must consider the maximum likelihood estimate. To find the prior P(c), we find the percentage of documents in our training set in each class. This is done by dividing the number of documents with class c N_c by the total number of documents N_{doc}, giving us our maximum likelihood estimate of the prior $\hat{P}(c)$:

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

To learn the probabilities of $P(f_i|c)$, where we assume a feature is the existence of a word w in the document, so we want to learn $P(w_i|c)$. This is computed as the fraction of times the word w_i appears among all words of vocabulary V in all documents of class c:

$$\hat{P}(w_i|c) = \frac{\text{count}(w_i, c)}{\sum_{w \in V} \text{count}(w, c)}$$

A problem that must be resolved is accounting for words that do not show up in the training set, leading to issues estimating the likelihood of words that were not trained. This is a problem because when trying to estimate the likelihood of a word that does not exist in the training set, the probability will be 0. Since the algorithm ‘naively’ multiplies all likelihoods, the

posterior probability will always be 0 regardless of other evidence given. This problem is fixed with the add-one (Laplace) smoothing solution. The solution is applied here:

$$\widehat{P}(w_i|c) = \frac{\text{count}(w_i, c) + 1}{(\sum_{w \in V} \text{count}(w, c)) + |V|}$$

Once the algorithm is trained, we can give it a new test document, and the algorithm will use the probabilities learned from the training set to classify the new document as positive or negative. The algorithm will get the prior probabilities of the two classes, positive and negative, and the likelihood of words in the test document. The algorithm will then give us the maximum posterior probability between the two classes, giving us our sentiment estimation.

There are some optimization techniques commonly applied to improve performance. One way to optimize is to clip the word counts in each document to one, as a word showing up is more important than its frequency. Another optimization is to deal with negation. For example, “I really like this pizza” versus “I really didn't like this pizza”, the phrase didn’t completely alters the inference drawn. A simple method to solve this is during document processing, where we append NOT in front of every word after a token of negation (n’t, not, no, never) until the next punctuation. These newly formed words, NOT.like, will occur more often in negative documents, acting as cues for negative sentiment. NOT.bored and NOT.bad would give positive cues.

The Naïve Bayes algorithm is an effective and simple solution to sentiment classification. Though the assumptions used, especially the reliance on the naïve Bayes assumption, can sometimes lead to less accurate predictions compared to other methods, the algorithm is easy to implement and performs well.