Machine Learning

Bayesian Classification

Nicusor-Daniel Vlasin
R00124330

Table of Contents

Introduction

In this report I will talk about Bayesian classification and how its use to classified unseen movie review base on some training data. I will use Multinomial Naive Bayes algorithm to predict if an unseen movie review its positive and negative.

Naive Bayes algorithm it is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

Bayes theorem provides a way of calculating posterior probability:

$$P(w|c) = \frac{count(w,c)+1}{count(c)+|V|}$$

Count(w,c) is the number of occurrences of the word W in all documents of class C

Count(c) The total number of words in all documents of class C (including dublicates)

|V| The number of words in the vocabulary.

Getting started.

As I said above I'll use the multinomial Naïve Bayesian algorithm to predict if a new movie review is negative and positive.

My code starts by loading two types of reviews positive and negative. Those reviews are called training data, I will use those to feed the algorithm and then I will test on unseen review.

Next, I use **set()** to create my **unique_words_set**. This will contain all the unique words from both negative and positive reviews.

Next, I'm counting the frequency of each positive and negative word and store it in a variable called **negative_frequency** and **positive_frequency,** those two variables will help calculate the probability later.

Next step is to calculate the probabilities. For this step I create a method that will populate the both dictionary's with the probability of each world give the class, for this task I use the formula provided in the assignment spec.

Cleaning the data set.

Basic methods of cleaning the data set.

In this assignment Multinomial Bayes Naïve algorithm will classify movies reviews as positive or negative base on unclean training data. There are some basic methods to clean the data before feeding in to the algorithm. Methods such as remove punctuation or lowercase each word in the data training.

Converting to lower-case

In this example the result was **79.1%** accuracy for positive test review and **66.1%** for the negative test review. Those are pretty high result because I'm using unclean data to train Bayesian algorithm. By simply converting each word in training data to lower-case (.lower())the accuracy changes a bit. The new results are **84.6%** for positive and **70.6%** for negative.

Non-words (Punctuation)

Another method is removal of non-words: Numbers and punctuation have both been removed. All white spaces (tabs, newlines, spaces) have all been trimmed to a single space character. For this I create a method called **def remove_punctuation(unique_words_set)** and I passing in the unique set of words from both positive and negative review. This method will go through all unique words set al remove punctuation.

This way I should have a clean training data. Removing non-words from my training set did not affect my result.

Stop-words

Next method that I implement to clean my training data is removing the stop. Stop words are words that do not have any value in determining the type of a document. Examples of stop words include "the", "it", "of", and "a". For this I create a method that reads a text file whit all the stop words in English language and check that list against the set of all unique words.

Removing stop words in the pre-processing step is a common practice in artificial intelligence. Doing so minimizes the overall processing load and memory requirements, and results in a narrow set of review relevant terms. My new results after the stop-words methods are: **86.7%** positive and **72.6%** negative.

Advance methods of cleaning the data set.

Negation

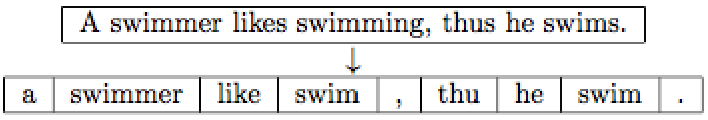
One of the advance methods is where negation terms were removed. Negation terms have a large effect on the meaning of sentences. For example, a positive review may contain the phrase “such a good action”, and a negative review may contain the phrase “not such a good action”. A naive Bayes classifier cannot differentiate between such distinctions, making the difference between these positive and negative reviews ambiguous.

Stemming and Lemmatization

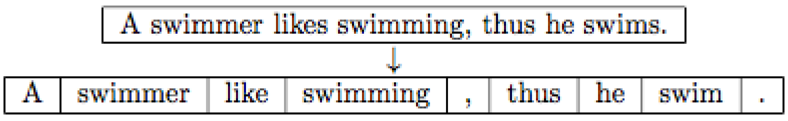
Another advance method is Stemming and Lemmatization. Stemming describes the process of transforming a word into its root form (swimming = swim). For this part of mu assignment I had to “**from nltk.stem.snowball import SnowballStemmer”.** This import gave me the ability to bring all non-root words from my set to their root.

In contrast to stemming, lemmatization aims to obtain the canonical (grammatically correct) forms of the words, the so-called lemmas. Lemmatization is computationally more difficult and expensive than stemming, and in practice, both stemming, and lemmatization have little impact on the performance of text classification.

Example of stemming:



Example of lemmatization:



N-Grams

In the *n*-gram model, a token can be defined as a sequence of *n* items. The simplest case is the so-called unigram (1-gram) where each word consists of exactly one word, letter, or symbol. Choosing the optimal number n depends on the language as well as the particular application.

- unigram (1-gram):

a	swimmer	likes	swimming	thus	he	swims

- bigram (2-gram):

a swimmer	swimmer likes	likes swimming	swimming thus	...

- trigram (3-gram):

a swimmer likes	swimmer likes swimming	likes swimming thus	...

Conclusion

The final result after I clean the data set is **88.6%** positive and **72.6%** negative, but whit some extra time I will be able to bring those percentage up to 90%.

This assignment was very interesting specially for me because of two reasons because it was the first time when I encounter python and I had the chance to work whit it and the assignment itself was interesting. Having the change to create a sentiment analysis algorithm and be able to make it to predict whit a high accuracy was for sure something new for me.

Text mining was also enjoyable, seeing how many ways to clean a text and multiple ways to implement them.

Testing gave me a bit of trouble because of my machine was unbale to process that huge amount of files in a reasonable amount of time so I lost precious time.