

CSE 5334 Data Mining :

Naïve Bayesian learning for classifying news text articles

Sindoor Ravikumar Murthy

1001862126

University of Texas at Arlington

Abstract

Given the goal of training a Naïve bayes classifier, we must use the appropriate methods to train and test the model. This paper will explain my thought process in how I selected what kind of model I decided on, how I implemented it in Python and the experiments.

Naïve Bayesian learning for classifying news text articles

Data:

Naïve Bayes classifiers are one among the most successful known algorithms for learning to classify text documents. The dataset contains 20,000 newsgroup messages drawn from the 20 newsgroups. The dataset contains 1000 documents from each of the 20 newsgroups.

The 20 news groups:

alt.atheism	rec.sport.hockey
comp.graphics	sci.crypt
comp.os.ms-windows.misc	sci.electronics
comp.sys.ibm.pc.hardware	sci.med
comp.sys.mac.hardware	sci.space
comp.windows.x	soc.religion.christian
misc.forsale	talk.politics.guns
rec.autos	talk.politics.mideast
rec.motorcycles	talk.politics.misc
rec.sport.baseball	talk.religion.misc

Training Model - My Contribution:

1. Training a model using Naïve Bayes's theorem to classify news text articles.

The dataset '20newsgroup' is made up of 20 sub-folders of different fields of news articles. The files are of an unknown extension and various encoding types were tried before 'Latin-1' was successful in opening all the files successfully.

Each newsgroup or sub-folder is made up of approximately 1000 files.

The NB classifier – Posterior probability is given by:

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

2. Using the trained model to do the classification

Coming down to the question, how exactly are we using Bayes to predict the newsgroup of a given file.

We can illustrate that, with the help of a small example.

“The microprocessor was made by AT&T, the wireless company that also has a big football stadium in Arlington, Texas”

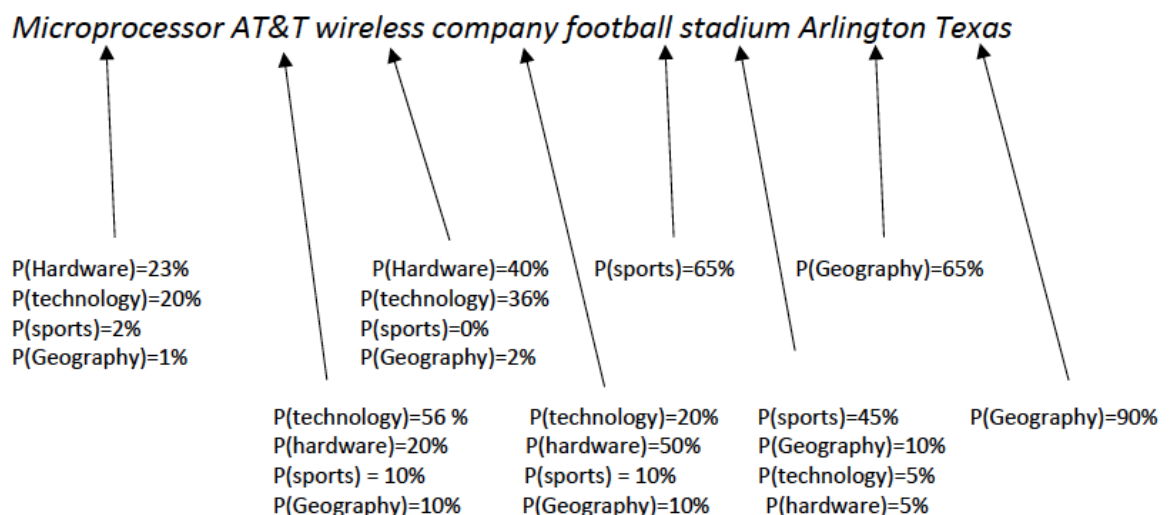
Now, this sentence is the raw form of the text file given. We need to remove certain common words (they are called stop words) that are too generic and just unnecessarily increase the processing time and skew the results.

These stop words are removed from each file and the model is trained. So, the above sentence after data cleaning looks like this.

“Microprocessor AT&T wireless company football stadium Arlington Texas”

Now, let's assume the model has been trained and we have a frequency count of the trained words.

The above example sentence will be classified something like this.



So, essentially the words of a sentence are given a probability score, and the sentence is classified as a particular group according to the max probability score.

Now, this is done over larger sets(files) rather than single sentences.

Coming down to, how the probability of each word is calculated, we iterate through each word in each file. Check, if that particular word was encountered before in the present newsgroup or class. If yes, then we increase the frequency count and if no, then we start counting.

For example, if we encountered the word "onomatopoeia" 75 times in the atheism newsgroup (which has 1000 words), the probability that a word that we encounter is "onomatopoeia" and belongs to the "atheism" newsgroup is 75/1000,

Or in short $P(\text{onomatopoeia} | \text{atheism}) = 0.075$

And to find if the probability of the class, if the word is “onomatopoeia” is given by:

$$P(\text{atheism} | \text{onomatopoeia}) = P(\text{onomatopoeia} | \text{atheism}) \times P(\text{class}) / P(\text{onomatopoeia})$$

Where $P(\text{class})$ is the probability of the class and $P(\text{onomatopoeia})$ is the probability of all instances of the word.

Now, to avoid completely jagged results (because of the absence of certain words), Laplacian smoothing is done when we encounter a new word with zero frequency.

3. Experiments and Results

Stop Word	Dataset %	Laplacian	Time taken in secs	Accuracy in percent
Eliminated	50	0.000001	77	86.8941294530858
Eliminated	50	0.00001	78	87.21525338685399
Eliminated	50	0.0001	74	87.46613146011039
Eliminated	50	0.001	80	87.61665830406422
Eliminated	50	0.01	78	87.60662318113397
Eliminated	50	0.1	78	86.97441043652785
Not Eliminated	50	0.001	65	87.18514801806322
Eliminated	100	0.001	131	93.0636127225445

Prediction accuracy = 87.61665830406422 %
Time elapsed: 80 secs

At optimum efficiency(50% dataset, 0.001 Laplacian smoothing and stop words being eliminated) my model has an accuracy of **87.61665830406422%**

98 % tested
Prediction accuracy = 86.97441043652785 %
Time elapsed: 78 secs

At 50% test, 0.1 Laplacian Smoothing and stop words being eliminated my model has an accuracy of **86.97441043652785%**

Prediction accuracy = 87.60662318113397 %
Time elapsed: 78 secs

At 50% test, 0.01 Laplacian Smoothing and stop words being eliminated my model has an accuracy of **87.60662318113397%**

Prediction accuracy = 87.46613146011039 %
Time elapsed: 74 secs

At 50% test, 0.0001 Laplacian Smoothing and stop words being eliminated my model has an accuracy of **87.46613146011039%**

Prediction accuracy = 87.21525338685399 %
Time elapsed: 78 secs

At 50% test, 0.00001 Laplacian Smoothing and stop words being eliminated my model has an accuracy of **87.21525338685399%**

Prediction accuracy = 86.8941294530858 %
Time elapsed: 77 secs

At 50% test, 0.000001 Laplacian Smoothing and stop words being eliminated my model has an accuracy of **86.8941294530858%**

```
98 % tested
Prediction accuracy = 87.18514801806322 %
Time elapsed: 65 secs
```

At 50% test, 0.001 Laplacian Smoothing and stop words **not** being eliminated my model has an accuracy of **87.18514801806322%**

```
Prediction accuracy = 93.0636127225445 %
Time elapsed: 131 secs
```

At 100% test, 0.001 Laplacian Smoothing and stop words being eliminated my model has an accuracy of **87.18514801806322%**

Methods:

There are 8 methods in the python file.

1. **def stop_words_calc():** The stop_words_calc function is used to read the stopwords from a given .txt file and put them in a list. This list is then stored in a stop_words variable to be used later.
parameters returned: text (a list of single-spaced stop words)
2. **def load_test_files(sub_folders):** The load_test_files function is used to select a sequentially news group and then sequentially select a file from the newsgroup. The selected file is then removed from the newsgroup's list and the file name is returned.
parameters accepted: subfolders(the list of all newsgroups in the dataset)
parameters returned: text(file name of the randomly selected file)
3. **def preprocessing (text,stop_words):** This function is used to clean the data of all the stop words, spaces, next line and punctuation marks and symbols.
parameters accepted: text(the data in a particular file) , stop_words(the list of stop words)
parameters returned: text(the cleaned data of that particular file)
4. **def calculate(_file_,dictionary,sun_folder_list):** This function is used to calculate the posterior probability of a particular file. We are doing Laplacian smoothing for new words with 0 count frequency. The smoothing peaks at 0.001.
parameters accepted: _file_(the data of a file) , dictionary(the word and frequency count of a particular group) , folder_list(list of all newsgroups in the dataset)
parameters returned: probability(the posterior probabilities)
5. **def open_file(file_location):** This function opens the unknown extension files of the dataset. This was required because, the file couldn't be opened in UTF-8 encoding and needed Latin-1 encoding.

parameters accepted: file_location(path of the file)

parameters returned: text(opened file)

6. **def train_model(sub_folder_list, stop_words):** This function is used to train the model. It trains the model on 50% + 1 file of the dataset and then removes those files from the main list, for the testing model. This iterates over the first 50% of the dataset, keeps a count of the words encountered and the newsgroups they were encountered in.

parameters accepted: sub_folder_list(list of all the newsgroups in the dataset) , stop_words(list of all the stopwords)

7. **def test_model(sub_folder_list, stop_words):** This function is used to test the model on last 50% of the dataset . It creates a confusion matrix for the tested sentence and selects the highest probability for the sentence. If the predicted posterior probability is equal to the actual classification, it is considered a success.

parameters accepted: sub_folder_list(list of all the newsgroups in the dataset) , stop_words(list of all the stopwords)

parameters returned: prediction(number of successes) , loop-1(total number of tries)

8. **def pathcheck(path, supporting_file_name):** This function is used to check if the supporting files are present in the current directory. If not, it exits the program

parameters accepted: path(path of the supporting files) , supporting_file_name(name of the file)