Applying Multinomial Naive Bayes to NLP Problems

Last Updated: 14-01-2019

Naive Bayes Classifier Algorithm is a family of probabilistic algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of a feature.

Bayes theorem calculates probability P(c|x) where c is the class of the possible outcomes and x is the given instance which has to be classified, representing some certain features.

```
P(c|x) = P(x|c) * P(c) / P(x)
```

Naive Bayes are mostly used in natural language processing (NLP) problems. Naive Bayes predict the tag of a text. They calculate the probability of each tag for a given text and then output the tag with the highest one.

How Naive Bayes Algorithm Works ?

Let's consider an example, classify the review whether it is positive or negative.

Training Dataset:

TEXT	REVIEWS
“I liked the movie”	positive
“It’s a good movie. Nice story”	positive
“Nice songs. But sadly boring ending. ”	negative
“Hero’s acting is bad but heroine looks good. Overall nice movie”	positive
“Sad, boring movie”	negative

We classify whether the text “overall liked the movie” has a positive review or a negative review. We have to calculate,
P(positive | overall liked the movie) – the probability that the tag of a sentence is positive given that the sentence is “overall liked the movie”.
P(negative | overall liked the movie) – the probability that the tag of a sentence is negative given that the sentence is “overall liked the movie”.

Before that, first, we apply Removing Stopwords and Stemming in the text.

Removing Stopwords: These are common words that don’t really add anything to the classification, such as an able, either, else, ever and so on.

“ilikedthemovi”	positive
“itsagoodmovienicestori”	positive
“nicesongsbutsadlyboringend”	negative
“herosactingisbadbutheroinelooksgoodoverallnicemovi”	positive
“sadboringmovi”	negative

Feature Engineering:

The important part is to find the features from the data to make machine learning algorithms works. In this case, we have text. We need to convert this text into numbers that we can do calculations on. We use word frequencies. That is treating every document as a set of the words it contains. Our features will be the counts of each of these words.

In our case, we have *P(positive | overall liked the movie)*, by using this theorem:

> ***P(positive | overall liked the movie)*** *= P(overall liked the movie | positive) * P(positive) / P(overall liked the movie)*

Since for our classifier we have to find out which tag has a bigger probability, we can discard the divisor which is the same for both tags,

P(overall liked the movie | positive)* P(positive) with P(overall liked the movie | negative) * P(negative)

There's a problem though: "overall liked the movie" doesn't appear in our training dataset, so the probability is zero. Here, we assume the **'naive'** condition that every word in a sentence is independent of the other ones. This means that now we look at individual words.

We can write this as:

> *P(overall liked the movie) = P(overall) * P(liked) * P(the) * P(movie)*

And now, these individual words actually show up several times in our training data, and we can calculate them!

Calculating probabilities:

First, we calculate the a priori probability of each tag: for a given sentence in our training data, the probability that it is positive P(positive) is 3/5. Then, P(negative) is 2/5.

Then, calculating P(overall | positive) means counting how many times the word "overall" appears in positive texts (1) divided by the total number of words in positive (11). Therefore, P(overall | positive) = 1/17, P(liked/positive) = 1/17, P(the/positive) = 2/17, P(movie/positive) = 3/17.

If probability comes out to be zero then By using Laplace smoothing: we add 1 to every count so it's never zero. To balance this, we

overall	1 + 1/17 + 21	0 + 1/7 + 21
liked	1 + 1/17 + 21	0 + 1/7 + 21
the	2 + 1/17 + 21	0 + 1/7 + 21
movie	3 + 1/17 + 21	1 + 1/7 + 21

Now we just multiply all the probabilities, and see who is bigger:

> *P(overall | positive) * P(liked | positive) * P(the | positive) * P(movie | positive) * P(postive) = 1.38 * 10^{-5} = 0.0000138*
>
> *P(overall | negative) * P(liked | negative) * P(the | negative) * P(movie | negative) * P(negative) = 0.13 * 10^{-5} = 0.0000013*

Our classifier gives "overall liked the movie" the positive tag.

Below is the implementation :

```
# cleaning texts
import pandas as pd
import re
import nltk
from nltk.corpus import stopwords
```

```
            ["Nice songs. But sadly boring ending.", "negative"],
            ["sad movie, boring movie", "negative"]]

dataset = pd.DataFrame(dataset)
dataset.columns = ["Text", "Reviews"]

nltk.download('stopwords')

corpus = []

for i in range(0, 5):
    text = re.sub('[^a-zA-Z]', '', dataset['Text'][i])
    text = text.lower()
    text = text.split()
    ps = PorterStemmer()
    text = ''.join(text)
    corpus.append(text)

# creating bag of words model
cv = CountVectorizer(max_features = 1500)

X = cv.fit_transform(corpus).toarray()
y = dataset.iloc[:, 1].values



# splitting the data set into training set and test set
from sklearn.cross_validation import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
          X, y, test_size = 0.25, random_state = 0)
```

```
# predicting test set results
y_pred = classifier.predict(X_test)

# making the confusion matrix
cm = confusion_matrix(y_test, y_pred)
cm
```


Recommended Posts:

Naive Bayes Classifiers

sympy.stats.Multinomial() function in Python

Applying Convolutional Neural Network on mnist dataset

MoviePy – Applying Resize effect on Video Clip

MoviePy – Applying Color effect on Video Clip

MoviePy – Applying Speed effect on Video Clip

Applying Lambda functions to Pandas Dataframe

NLP | Classifier-based Chunking | Set 2

Processing text using NLP | Basics

Readability Index in Python(NLP)

Feature Extraction Techniques - NLP

Python | NLP analysis of Restaurant reviews

NLP | Chunking and chinking with RegEx

NLP | Training Unigram Tagger

NLP | Synsets for a word in WordNet

NLP | Part of Speech - Default Tagging

NLP | Word Collocations

Darshika Sanghi
Check out this Author's contributed articles.

Article Tags : Advanced Computer Subject Machine Learning Python

Practice Tags : Machine Learning

Be the First to upvote.

0

No votes yet.

To-do Done

Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments