**Project**: Dealing with Special Text: Text Data (Spam Email Detector)

**Text classification using Naïve Bayes**

Naïve Bayes classifiers are a collection of classification algorithms based on Bayes’ Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other. Naïve Bayes classifiers have been heavily used for text classification and text analysis machine learning problems.

Text Analysis: is major application field for machine learning algorithms. However, the raw data, a sequence of symbols (i.e. strings) cannot be fed directly to the algorithms themselves as most of them expect numerical features with a fixed size rather than the raw documents with variable length.

Things to understand

1. How Naïve Bayes works
2. How we can use text data and fit them into a model after transforming them into a more appropriate form.
3. Implement a multi-class text classification problem in python. (Spam Email Detection)

**Probability**

* Picking a random card, what is the probability of getting a “queen”?



**4 Queens, 52 total cards**

**P(Queen) = 4/52 = 1/13**

* Pick a random card, you know it is a “diamond”. Now what is the probability of that being a “queen”?



**Total Diamonds = 13**

**Queen = 1**

**P(Queen/Diamond) = 1/13**

This is Conditional Probability.

P(A/B) = Probability of event A knowing that event B has already occurred.

**Naïve Bayes**

Thomas Bayes gave this equation,

Naïve Bayes, the reason it is called “naïve” is that we make an assumption that all the features from our dataset are independent of each other (even if some of them are not) which is naïve of us to think. This makes the algorithm simple and effective.

Naïve Bayes is used in email spam detection, handwritten character reorganization, weather prediction, face detection and news article categorization.

To understand I have performed and made a model based on this algorithm for titanic survival prediction.



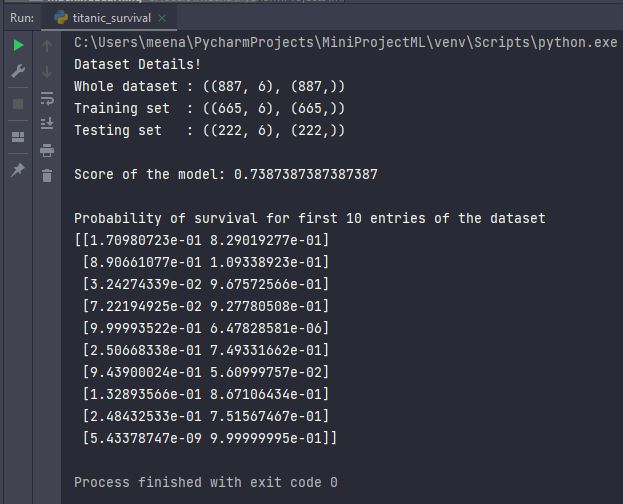
The above piece of code shows how Naïve Bayes Algorithm can be applied on real life problems to predict a specific result. I have used titanic survival dataset here and have used Naïve Bayes classifier to find out the survival probability of titanic travelers. I first investigated the titanic dataset by creating a train and test data set using scikit-learn module. The train set contains all the features (Pclass, Sex {male=1, female=0}, Age, Siblings / Spouse, Parents / Children, Fare) and the target (the variable which outcome I want to predict i.e., Survived). The test dataset is used for submission, so that my model could work and predict the result.

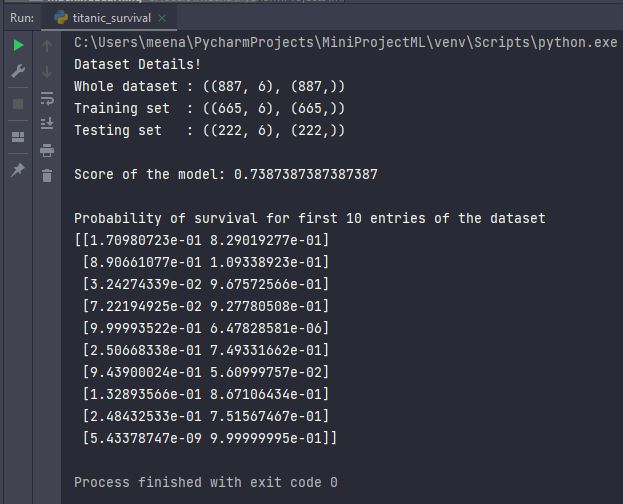
The model that I used is Naïve Bayes’ GaussianNB model.

The above code’s output will show us some detailed information about the dataset and then it prints the score of the model, then the predicted values for the test set on the basis of the training set.

The score of the model is calculated on the basis that how many of the target values the model predicted correctly.

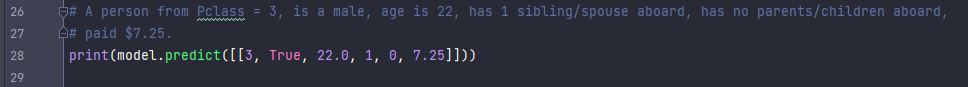
This model has a score of 73.87%

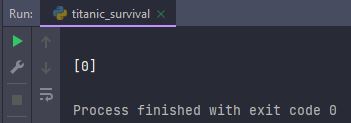




Now, if I fed my model with some information for a passenger with the following data

Pclass = 3, is a male, age is 22, has 1 sibling/spouse aboard, has no parents/children aboard, paid $7.25. Let’s see what the model predicts about the survival of the passenger.



The model gives us an array of a value (which will either be 1 (in case the passenger survived), 0 (in case the passenger did not survive). The output clearly shows us that the passenger did not survive.

**IMPLEMENTATION**

**Dealing with Text Data**

How are we going to use the raw text data to train the model? The raw data is collection of strings!

However, the raw data, a sequence of symbols (i.e. strings) cannot be fed directly to algorithms themselves as most of them expect numerical feature vectors with a fixed size rather than the raw text documents with variable length.

Steps to convert documented data into numerical feature vectors

Step 1: Get the data

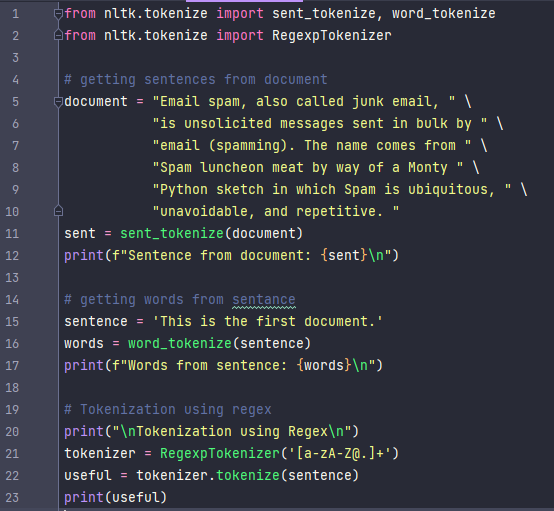
Step 2: Tokenization

* Breaking down documents into sentences and then sentences into words
* Removing unnecessary words i.e., stopwords

Step 3: Stemming, converting different forms of same word into single base word

Step 4: Building a new vocabulary (list of distinct words from training set)

Step 5: Assign number to each word

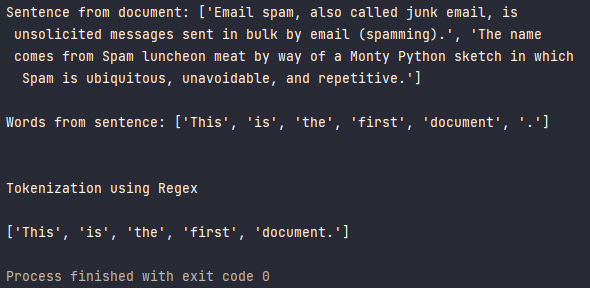


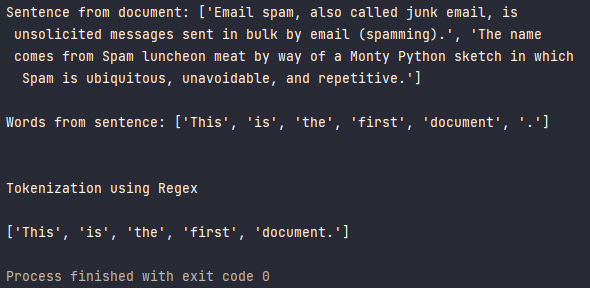
“Tokenization”

Tokenization can be done in two ways: -

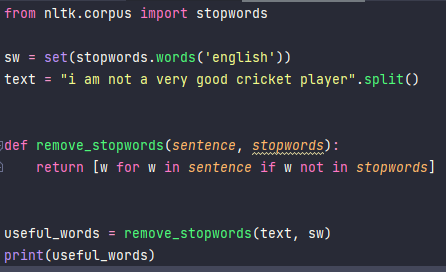
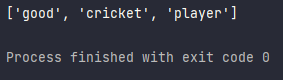
1. sent\_tokenize and word\_tokenise
2. Regular Expression or Regex

The difference in both can be determined by looking at their respective outputs

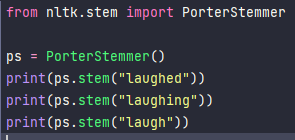
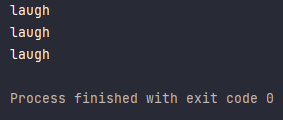


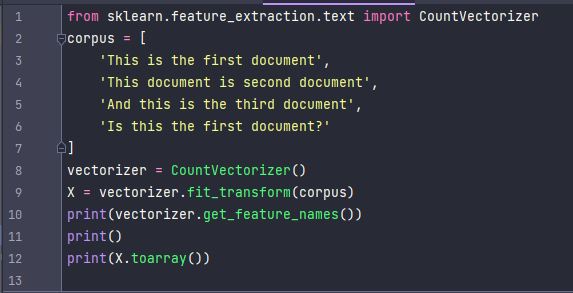


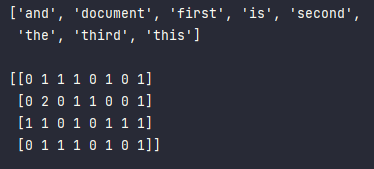
“Stopwords” Example



“Stemming” Example

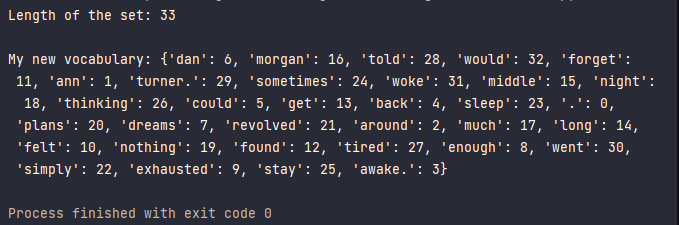
 

“Counting” Example



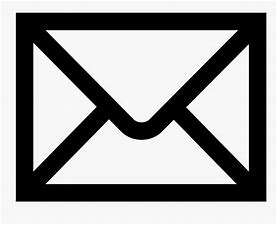
Now, combining all the above concepts…

# Combining Tokenization, Stopwords removal, Stemming and Count Vectorization  
from nltk.tokenize import RegexpTokenizer  
from nltk.corpus import stopwords  
from sklearn.feature\_extraction.text import CountVectorizer  
  
corpus = [  
 'Dan Morgan told himself he would forget Ann Turner.',  
 'Sometimes he woke up in the middle of the night thinking of Ann , and then could not get back to sleep .',  
 'His plans and dreams had revolved around her so much and for so long that now he felt as if he had nothing .',  
 'He found that if he was tired enough at night , he went to sleep simply because he was too exhausted to stay awake.'  
]  
  
  
def remove\_stopwords(*sentence*, *stopwords*):  
 return [w for w in *sentence* if w not in *stopwords*]  
  
  
def myTokenizer(*document*):  
 tokenizer = RegexpTokenizer('[a-zA-Z@.]+')  
 sw = set(stopwords.words('english'))  
 words = tokenizer.tokenize(*document*.lower())  
 # remove the stopwords  
 words = remove\_stopwords(words, sw)  
 return words  
  
  
# print(myTokenizer('this is a random text'))  
vectorizer = CountVectorizer(tokenizer=myTokenizer)  
X = vectorizer.fit\_transform(corpus).toarray()  
print(f"Length of the set: {len(X[0])}")  
print(f"\nMy new vocabulary: {vectorizer.vocabulary\_}")



Now once everything is covered let’s combine them all in one for our Spam Email Detector

**Spam Email Detector**



**Spam (1) Ham (0)**

**A = 0: Received mail is a ham mail**

**A = 1: Received mail is a spam mail**

**B = Received Mail**

What is the probability of a mail being as spam if a document is given?

**Final Prediction =**

In layman’s language, the above statement can be explained as the maximum of the probability of a word A, being present in a ‘ham’ email and the probability of a word A, being present in a ‘spam’ email.

**OUTPUT**

