

## **MACHINE LEARNING - CSE 6363-005 - PROJECT 2**

### **Naive Bayes Classifiers:**

Naive Bayes classifiers are a family of probabilistic algorithms that use Bayes' theorem to make predictions or classify items into one of several categories. Naive Bayes classifiers are often used in machine learning and natural language processing applications because they are simple, fast, and require relatively little training data.

The "naive" in Naive Bayes refers to the assumption that the features used in the classification process are independent of each other, which is often not the case in practice. However, despite this simplification, Naive Bayes classifiers can still achieve high accuracy and are widely used in various applications.

The Naive Bayes classifier works by first learning the probabilities of each feature given each class in the training data. These probabilities are then used to calculate the probability of each class given a new instance, using Bayes' theorem. The class with the highest probability is then assigned as the predicted class for the instance.

### **Bayes Theorem**

Bayes' theorem is a mathematical formula that describes the relationship between the conditional probabilities of two events, A and B. The theorem states that the probability of A given B is equal to the probability of B given A times the prior probability of A, divided by the prior probability of B.

In a classification problem, we use Bayes' theorem to calculate the probability of a class label given a set of input features. Specifically, we want to calculate the posterior probability of the class label given the input features. This probability can be calculated using Bayes' theorem as:

$$P(y | X) = P(X | y) * P(y) / P(X)$$

where  $y$  is the class label,  $X$  is the vector of input features,  $P(X | y)$  is the likelihood of observing the input features given the class label,  $P(y)$  is the prior probability of the class label, and  $P(X)$  is the probability of observing the input features. The denominator  $P(X)$  can be computed using the law of total probability, which involves summing over all possible values of  $y$ .

## About the dataset:

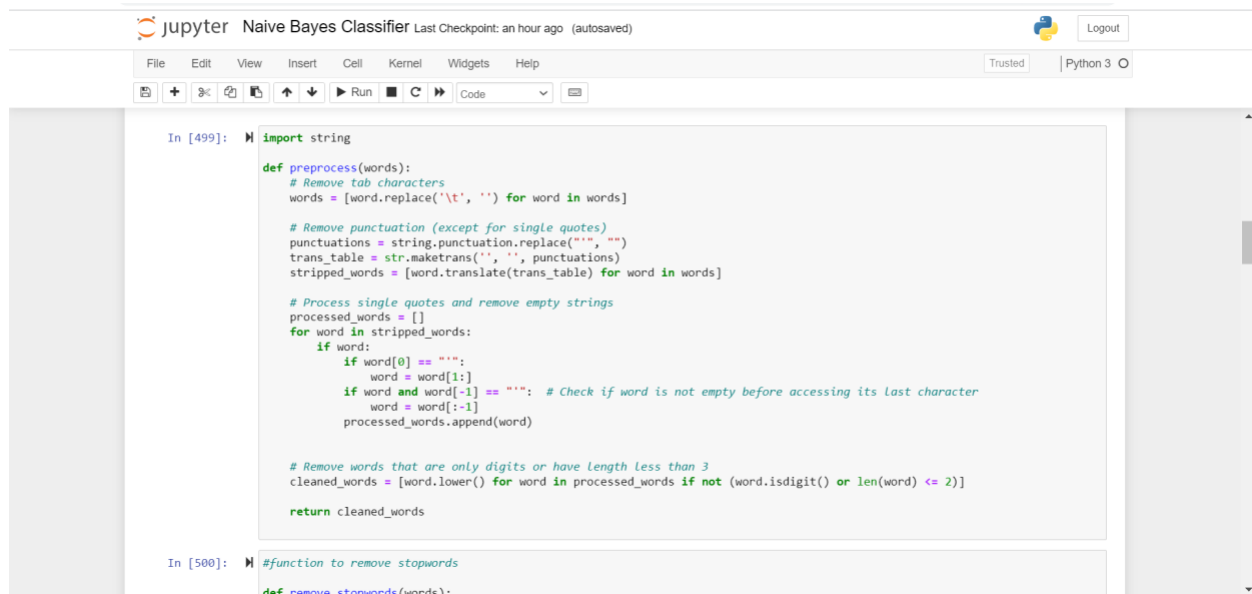
The dataset being used is the 20 Newsgroup Dataset, which is available for download for free at <http://www.cs.cmu.edu/afs/cs/project/theo-11/www/naive-bayes.html>. This dataset is one of the best-known datasets used for document classification and text clustering.

This dataset consists of 20,000 messages, collected from 20 different news groups, partitioned (nearly) evenly across 20 different newsgroups. 1000 messages from each of the twenty newsgroups were chosen at random and partitioned by newsgroup name.

## Implementation:

This model has been implemented using Python 3, which categorizes the data into one of the twenty available newsgroups. The dataset has been subjected to pre-processing where I divided the news group data into two equal parts for training and testing each, after which the model has been trained for classification.

## Pre-processing:



```
In [499]: import string

def preprocess(words):
    # Remove tab characters
    words = [word.replace('\t', '') for word in words]

    # Remove punctuation (except for single quotes)
    punctuations = string.punctuation.replace("'", "")
    trans_table = str.maketrans('', '', punctuations)
    stripped_words = [word.translate(trans_table) for word in words]

    # Process single quotes and remove empty strings
    processed_words = []
    for word in stripped_words:
        if word:
            if word[0] == "'":
                word = word[1:]
            if word and word[-1] == "'": # Check if word is not empty before accessing its last character
                word = word[:-1]
            processed_words.append(word)

    # Remove words that are only digits or have length less than 3
    cleaned_words = [word.lower() for word in processed_words if not (word.isdigit() or len(word) <= 2)]

    return cleaned_words

In [500]: #function to remove stopwords

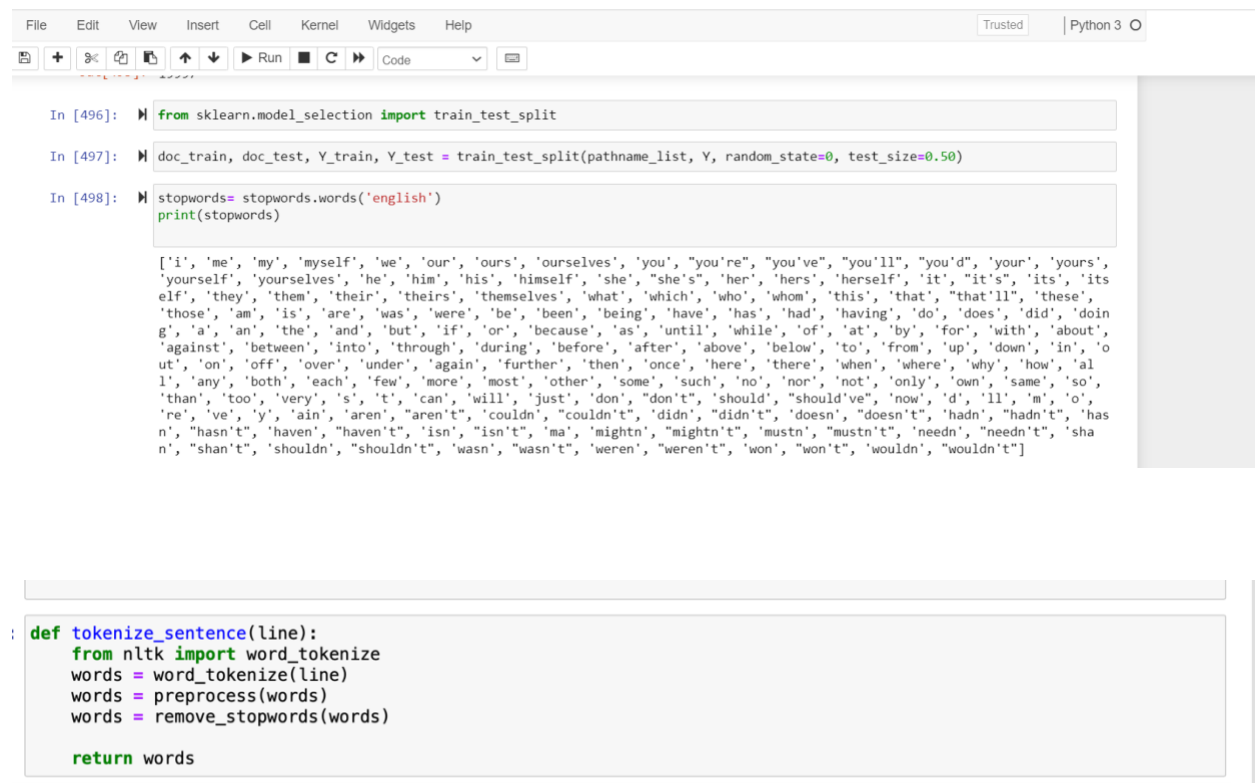
def remove_stopwords(words):
```

The preprocess() function takes in a list of words words and performs several text preprocessing steps to clean up the words before further processing. The function returns a cleaned list of words.

The function first removes any tab characters from the input words using a list comprehension. Next, it removes all punctuation from the input words, except for single quotes, using the string.punctuation constant and the str.maketrans() and translate() methods. This step helps to remove any unnecessary noise from the input words.

The function then processes single quotes in the words by removing leading or trailing single quotes, if present, and appends the processed words to a new list called processed\_words. This step helps to handle single quotes correctly, since they can be part of a word or used to indicate possession.

## Using stop words:



```
In [496]: from sklearn.model_selection import train_test_split

In [497]: doc_train, doc_test, Y_train, Y_test = train_test_split(pathname_list, Y, random_state=0, test_size=0.50)

In [498]: stopwords = stopwords.words('english')
           print(stopwords)

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours',
'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'its
elf', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these',
'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doin
g', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about',
'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'o
ut', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'al
l', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so',
'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o',
're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'has
n', "hasn't", 'haven't', 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'sha
n', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]

def tokenize_sentence(line):
    from nltk import word_tokenize
    words = word_tokenize(line)
    words = preprocess(words)
    words = remove_stopwords(words)

    return words
```

The tokenize\_sentence() function takes in a string line and tokenizes it into a list of words using the word\_tokenize() function from the Natural Language Toolkit (NLTK) library. It then preprocesses the words using a preprocess() function and removes stop words using a remove\_stopwords() function. Finally, it returns the list of processed words.

```
In [128]: def tokenize(path):
          f = open(path, 'r', encoding='iso-8859-1')
          # rest of the function
          text_lines = f.readlines()
          text_lines = remove_metadata(text_lines)
          doc_words = []
          for line in text_lines:
              doc_words.append(tokenize_sentence(line))

          return doc_words
```

The **remove\_metadata()** function takes in a list of strings **lines** representing a text document and removes any metadata information that might be present at the beginning of the document.

## Training:

```
In [513]: X_train = []
          for k in dictionary.keys():
              row = []
              for f in features:
                  if(f in dictionary[k].keys()):
                      row.append(dictionary[k][f])
                  else:
                      row.append(0)
              X_train.append(row)
```

```
In [514]: X_train = np.asarray(X_train)
          Y_train = np.asarray(Y_train)
```

## Testing:

```
In [517]: X_test = []
          for k in dictionary_test.keys():
              row = []
              for f in features:
                  if(f in dictionary_test[k].keys()):
                      row.append(dictionary_test[k][f])
                  else:
                      row.append(0)
              X_test.append(row)
```

```
In [518]: X_test = np.asarray(X_test)
          Y_test = np.asarray(Y_test)
```

```
In [106]: def fit(X_train, Y_train):
          result = {}
          classes, counts = np.unique(Y_train, return_counts=True)

          for i in range(len(classes)):
              curr_class = classes[i]

              result["TOTAL_DATA"] = len(Y_train)
              result[curr_class] = {}

              X_tr_curr = X_train[Y_train == curr_class]

              num_features = len(X_tr_curr)

              for j in range(num_features):
                  result[curr_class][features[j]] = X_tr_curr[:,j].sum()

              result[curr_class]["TOTAL_COUNT"] = counts[i]

          return result
```

The fit() function takes in a training set X\_train and corresponding labels Y\_train and trains a Naive Bayes classifier by computing and storing probabilities based on the frequency of features in each class. It returns a dictionary result containing the calculated probabilities.

The function first initializes an empty dictionary result. It then calculates the total number of instances in the training set and stores it in the result dictionary under the key "TOTAL\_DATA". Next, it calculates the unique classes and their respective counts using np.unique(Y\_train, return\_counts=True)

## Result:

The model has been trained and tested. The model accurately classified the test data, and the following results were obtained.

```
In [93]: accuracy_score(Y_test, my_predictions)
Out[93]: 0.6645664566456646

In [94]: print(classification_report(Y_test, my_predictions))
```

misc.forsale	0.88	0.40	0.55	501
rec.autos	0.89	0.49	0.64	528
rec.motorcycles	0.99	0.60	0.75	501
rec.sport.baseball	0.99	0.71	0.82	501
rec.sport.hockey	0.94	0.91	0.92	501
sci.crypt	0.63	0.88	0.74	488
sci.electronics	0.85	0.47	0.60	519
sci.med	0.92	0.80	0.85	504
sci.space	0.77	0.83	0.80	482
soc.religion.christian	0.81	0.87	0.84	516
talk.politics.guns	0.85	0.55	0.67	504
talk.politics.mideast	0.76	0.91	0.83	520
talk.politics.misc	0.21	0.88	0.34	481
talk.religion.misc	0.61	0.41	0.49	502
accuracy			0.66	9999
macro avg	0.74	0.63	0.65	9999
weighted avg	0.78	0.66	0.68	9999

## References:

<http://www.cs.cmu.edu/afs/cs/project/theo-11/www/naive-bayes.html>