In [ ]:
```
import numpy
import pandas as pd
import csv
import time
import random
import string
import pickle
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk import NaiveBayesClassifier
from nltk import classify
```

In [ ]:
```
#nltk.download('stopwords')
#nltk.download('punkt')
```

In [ ]:
```
# file path to the txt files
data_path = './data/'

# load csv files into pandas dataframes
Custom_Header = ["Sentence", "Sentiment"]
data_amazon = pd.read_csv(data_path+'amazon.txt', sep='\t', quoting=csv.QUO
                          header=None, names=Custom_Header)
data_imdb = pd.read_csv(data_path+'imdb.txt', sep='\t', quoting=csv.QUOTE_NO
                        header=None, names=Custom_Header)
data_yelp = pd.read_csv(data_path+'yelp.txt', sep='\t', quoting=csv.QUOTE_NO
                        header=None, names=Custom_Header)

# print(data_amazon.head(), data_imdb.head(), data_yelp.head())
# print(data_amazon.shape, data_imdb.shape, data_yelp.shape)
# print(data_amazon.info(), data_imdb.info(), data_yelp.info())
```

Stop words are words that do not add much meaning to a sentence and can be removed

In [ ]:
```
stop_words = set(stopwords.words('english'))
stop_words_array = stopwords.words('english')
```

Tokenize every dataset reviews and remove stop words and punctuations

1. Create a new set of words from the words in the dataset.
2. For each sentence in the dataset tokenize the words and remove stop words and punctuations.
3. Create a new set of words from the words in the dataset.
4. It can be written as:

```
tokens_ = set()
for words in data["Sentence"]:
    for word in word_tokenize(words):
        if word.lower() not in stop_words:
            if word.lower() not in string.punctuation:
                tokens.add(word.lower())
```

In [ ]:
```
amazon_tokens = set(word.lower() for words in data_amazon['Sentence'] for wo
    words) if word.lower() not in stop_words_array and word.lower() not in s

imdb_tokens = set(word.lower() for words in data_imdb['Sentence'] for word i
```

```
    words) if word.lower() not in stop_words_array and word.lower() not in s

yelp_tokens = set(word.lower() for words in data_yelp['Sentence'] for word :
    words) if word.lower() not in stop_words_array and word.lower() not in s
```

We create a list of tuples. Each tuple contains a dictionary and a string.

The dictionary contains the words in the token and the string is the sentiment of the review

1. Creates a list of dictionaries. Each dictionary contains a word as a key and a boolean value.
2. The boolean value is true if the word is in the sentence and false if not.
3. Creates a list of tuples. Each tuple contains a dictionary and the sentiment
4. It can be written as :

```
train_alt = []
for i in range(0, len(data)):
    dict_ = {}
    for word in tokens_:
        if word in word_tokenize(data['Sentence'][i].lower()) and
word.lower() not in stop_words_array:
            dict_[word] = True
        else:
            dict_[word] = False
    # create a tuple with the dictionary and the sentiment
    tuple_ = (dict_, data['Sentiment'][i])
    # add the tuple to the train_ array
    train_alt.append(tuple_)
```

In [ ]:
```
amazon_train = [({word: (word in word_tokenize(data_amazon['Sentence'][i].lo
                  for word in amazon_tokens}, data_amazon['Sentiment'][i]) 
imdb_train = [({word: (word in word_tokenize(data_imdb['Sentence'][i].lower
                  for word in imdb_tokens}, data_imdb['Sentiment'][i]) for i :
yelp_train = [({word: (word in word_tokenize(data_yelp['Sentence'][i].lower
                  for word in yelp_tokens}, data_yelp['Sentiment'][i]) for i :
```

For bigger data set we merge the data sets together

In [ ]:
```
data_train = amazon_train + imdb_train + yelp_train
```

FUNCTION to save the data_train to a pickle file

In [ ]:
```
def save_data_train(data, file_name):
    with open(file_name, 'wb') as handle:
        pickle.dump(data, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

Load the data_train from the pickle file

In [ ]:
```
save_data_train(data_train, 'data_train.pickle')
```

FUNCTION to load the data_train from the pickle file

In [ ]:
```
def load_data_train(file_name):
    with open(file_name, 'rb') as handle:
        data_train = pickle.load(handle)
    return data_train
```

Load the data_train from the pickle file

In [ ]:
```
data_train = load_data_train('data_train.pickle')
```

Shuffle the data_train to randomize the data

In [ ]:
```
random.shuffle(data_train)
```

Split the data_train into training and testing data

In [ ]:
```
data_train_x = data_train[:int(len(data_train)*0.8)]
data_test_x = data_train[int(len(data_train)*0.8):]
```

Create the model using NaiveBayesClassifier

In [ ]:
```
model = NaiveBayesClassifier.train(data_train_x)
model.show_most_informative_features()
```

```
Most Informative Features
                   great = True                1 : 0      =     28.3 : 1.0
               excellent = True                1 : 0      =     26.5 : 1.0
                   worst = True                0 : 1      =     21.6 : 1.0
                     bad = True                0 : 1      =     18.7 : 1.0
                   waste = True                0 : 1      =     17.1 : 1.0
                terrible = True                0 : 1      =     15.2 : 1.0
                    nice = True                1 : 0      =     14.3 : 1.0
                   happy = True                1 : 0      =     12.8 : 1.0
                 amazing = True                1 : 0      =     12.6 : 1.0
                    fine = True                1 : 0      =     11.4 : 1.0
```

compare the accuracy of the model to the test data

In [ ]:
```
acc = classify.accuracy(model, data_test_x)
print("Accuracy:", acc)
```

```
Accuracy: 0.47833333333333333
```

FUNCTION to save the model

In [ ]:
```
def save_model(model_, filename):
    with open(filename, 'wb') as f:
        pickle.dump(model_, f)
```

Save the model

In [ ]:
```
save_model(model, 'model.pickle')
```

FUNICTION to load the model from disk

In [ ]:
```
def load_model(filename):
    with open(filename, 'rb') as f:
        return pickle.load(f)
```

Load the model from pickle file

In [ ]:
```
newmodel = load_model('model.pickle')
```

FUNCTION to predict the sentiment of a sentence

In [ ]:
```
def predict(sentence):
    return newmodel.classify(dict([token, True] for token in word_tokenize(s
```

Test the model

In [ ]:
```
test_sentence = "I love this movie"
test_sentence2 = "I hate this movie"

print(predict(test_sentence))
print(predict(test_sentence2))
```

```
1
0
```