```
# Import libraries
import numpy as np
import pandas as pd
import argparse
import re
import nltk
import json
from nltk.tokenize import word_tokenize  # Make sure to install NLTK: pip install nltk
# Note: Required to download punkt package run this script once with  `nltk.download('punkt')`
 uncommented
nltk.download('punkt')


class NaiveBayesFilter:
    def __init__(self, test_set_path):
        self.vocabulary = None
        self.training_set: pd.DataFrame = None
        self.test_set: pd.DataFrame = None
        self.p_spam = None
        self.p_ham = None
        self.test_set_path = test_set_path
        self.p_unseen = None
        pd.set_option('display.max_colwidth', 160)


    def read_csv(self):
        self.training_set = pd.read_csv('train.csv', sep=',', header=0, names=['v1', 'v2'], en
coding = 'utf-8')
        self.test_set = pd.read_csv(self.test_set_path, sep=',', header=0, names=['v1', 'v2'],
 encoding = 'utf-8')


    def replaceURLs(self, msg):
        pattern = re.compile(r'http:\S+') # any words with http: get replaced with DOMAIN
        return pattern.sub('[URL]', msg)

    def replaceMonies(self, msg):
        pattern = re.compile(r'\$\S+') # any words with http: get replaced with DOMAIN
        return pattern.sub('[MONEY]', msg)

    def replacePhoneNumbers(self, msg):
        # Phone number regex generated by chatGPT
        pattern = re.compile(r'\b(?:\+\d{1,2}\s?)?(\d{1,4}[-.\s]?)?\(?\d{1,4}\)?[-.\s]?\d{1,9}
[-.\s]?\d{1,9}\b')
        return pattern.sub('[PHONE_NUM]', msg)

    def symbolReplacement(self):
        self.training_set['v2'] = self.training_set['v2'].apply(self.replaceURLs)
        self.training_set['v2'] = self.training_set['v2'].apply(self.replacePhoneNumbers)
        self.training_set['v2'] = self.training_set['v2'].apply(self.replaceMonies)

    def makeVocabulary(self):
        wordSet = set()
        for _, row in self.training_set.iterrows():
                for word in row['v3']:
                    wordSet.update([word])
        self.vocabulary=list(wordSet)

        with open("vocab.txt", 'w') as file:
            json.dump(sorted(self.vocabulary), file)

    def vectorization(self):
        vectors = list()
        for ind, row in self.training_set.iterrows():
            vectors.append(np.zeros(len(self.vocabulary)))
            for word in row[’v3’]:
                vectors[ind][self.vocabulary.index(word)] += 1
        return vectors


    def makeWordFreq(self, type):
        # make the word frequency dictionary
        wordFreqDict = {}
        for word in self.vocabulary:
            wordFreqDict[word] = 0
        for _, row in self.training_set.iterrows(): # runs 3000 times
            if type==row[’v1’]:
                ind=0
                for count in row[’v4’]: # Runs 8000 times
                    wordFreqDict[self.vocabulary[ind]] += count
                    ind+=1
        return wordFreqDict


    def data_cleaning(self):


        # Normalization
        # Replace addresses (hhtp, email), numbers (plain, phone), money symbols
        # Remove the stop-words
        self.symbolReplacement()

            # Lemmatization - Graduate Students
            # Stemming - Gradutate Students

        # Tokenization
        self.training_set[’v3’] = self.training_set[’v2’].apply(word_tokenize)

        # Create the vocabulary structure (Also handles removing duplicate words from vocab.)
        self.makeVocabulary()

        # Vectorization
        self.training_set[’v4’] = self.vectorization()

        # Create the frequency dictionaries
        spamWordFrequency = self.makeWordFreq("spam")
        hamWordFrequency = self.makeWordFreq("ham")
        self.p_spam = pd.DataFrame(list(spamWordFrequency.items()), columns=[’Word’, ’Frequenc
y’])
        self.p_ham = pd.DataFrame(list(hamWordFrequency.items()), columns=[’Word’, ’Frequency’
])

        pass


    def calcProbSpamAndHam(self):
        numSpam=0
        numHam=0
        numTotal=0
        for ind in self.training_set.index:
            if self.training_set[’v1’][ind] == "spam":
                numSpam+=1
            else:
                numHam+=1
            numTotal+=1
        return numSpam/numTotal, numHam/numTotal
    def getSpamHamAllFreq(self):
        spamWords = 0
        hamWords = 0
        for _,row in self.p_spam.iterrows():
            spamWords += row['Frequency']
        for _,row in self.p_ham.iterrows():
            hamWords += row['Frequency']
        return spamWords, hamWords, spamWords + hamWords


    def calcProbWordsSpam(self, totalSpamWords, totalWords, alpha):
        out = list()
        for _,row in self.p_spam.iterrows():
            out.append((row["Frequency"] + alpha)  / (totalSpamWords + alpha * totalWords)) # 
Implimentation of Laplace smoothing algorithm
            # Source: https://www.analyticsvidhya.com/blog/2021/04/improve-naive-bayes-text-cl
assifier-using-laplace-smoothing/
        return out

    def calcProbWordsHam(self, totalHamWords, totalWords, alpha):
        out = list()
        for _,row in self.p_ham.iterrows():
            out.append((row["Frequency"] + alpha)  / (totalHamWords + alpha * totalWords)) # I
mplimentation of Laplace smoothing algorithm
            # Source: https://www.analyticsvidhya.com/blog/2021/04/improve-naive-bayes-text-cl
assifier-using-laplace-smoothing/
        return out

    def fit_bayes(self):
        # Calculate P(Spam) and P(Ham)
        pSpam, pHam = self.calcProbSpamAndHam()

        # Calculate Nspam, Nham and Nvocabulary
        Nspam, Nham, Nvocabulary = self.getSpamHamAllFreq()

        # Laplace smoothing parameter
        alpha = 1

        self.p_unseen = alpha/(Nspam + alpha * Nvocabulary)

        # Calculate P(wi|Spam) and P(wi|Ham)
        self.p_spam["P(wi|Spam)"] = self.calcProbWordsSpam(Nspam, Nvocabulary, alpha)
        self.p_ham["P(wi|Ham)"] = self.calcProbWordsHam(Nham, Nvocabulary, alpha)


    def train(self):
        self.read_csv()
        self.data_cleaning()
        self.fit_bayes()


    def get_probability_spam(self, word):
        row = self.p_spam[self.p_spam['Word'] == word]
        if not row.empty:
            return row["P(wi|Spam)"].values[0]
        else:
            return self.p_unseen

    def get_probability_ham(self, word):
        row = self.p_ham[self.p_ham['Word'] == word]
        if not row.empty:
            return row["P(wi|Ham)"].values[0]
        else:
            return self.p_unseen

    def probMessageSpam(self, message):
        prob=1
        for word in message:
            wordProbability = self.get_probability_spam(word)
            prob *= wordProbability
        return prob

    def probMessageHam(self, message):
        prob=1
        for word in message:
            wordProbability = self.get_probability_ham(word)
            prob *= wordProbability
        return prob

    def sms_classify(self, message):
        ’’’
        classifies a single message as spam or ham
        Takes in as input a new sms (w1, w2, ..., wn),
        performs the same data cleaning steps as in the training set,
        calculates P(Spam|w1, w2, ..., wn) and P(Ham|w1, w2, ..., wn),
        compares them and outcomes whether the message is spam or not.
        ’’’

        msg = word_tokenize(self.replacePhoneNumbers(self.replaceMonies(self.replaceURLs(messa
ge))))


        p_spam_given_message = self.probMessageSpam(msg)
        p_ham_given_message = self.probMessageHam(msg)

        if p_ham_given_message > p_spam_given_message:
            return ’ham’
        elif p_spam_given_message > p_ham_given_message:
            return ’spam’
        else:
            return ’needs human classification’
        pass

    def classify_test(self):
        ’’’
        Calculate the accuracy of the algorithm on the test set and returns
        the accuracy as a percentage.
        ’’’
        self.train()

        missed=0
        correct=0
        for _,row in self.test_set.iterrows():
            claim=self.sms_classify(row[’v2’])
            if claim == row[’v1’]:
                correct += 1
            else:
                missed += 1

        accuracy = correct / (missed + correct) * 100
        return accuracy


if __name__ == ’__main__’:
    parser = argparse.ArgumentParser(description=’Naive Bayes Classifier’)
    parser.add_argument('--test_dataset', type=str, default = "test.csv", help='path to test dataset')
    args = parser.parse_args()
    classifier = NaiveBayesFilter(args.test_dataset)
    classifier.train()
    acc = classifier.classify_test()
```