# NAIVE BAYES IMPLEMENTATION

import math

from itertools import count

import re

import matplotlib.pyplot as plt

class NaiveBayes:

# to segregate each line into positive or negative class based on 0 or 1

def classification(self, sampleData, classPosParam, classNegParam):

for data in sampleData:

if data[-2] == '1':

classPosParam.append(data)

else:

classNegParam.append(data)

return classNegParam, classPosParam

# Data Cleansing (removing special characters from the data)

def dataCleansing(self, classData):

for i in range(len(classData)):

classData[i] = re.sub('[@\_!#$%^&\*()<>?/|{}~:.,"]', '', classData[i])

return classData

# Adding each word in a list

def splitString(self, classData):

vocabulary = []

for i in range(len(classData)):

vocabulary.extend(

classData[i].casefold().removesuffix('\t1\n').removesuffix('\t0\n').split(

' ')) # To remove trailing 1 or 0 from each document

return vocabulary

def createUniqueVocabOfClass(self, vocab):

uniqueVocab = []

for i in vocab:

if i not in uniqueVocab:

uniqueVocab.append(i)

return uniqueVocab

def createVocabOfTotalWords(self, uniqueVocabPos, uniqueVocabNeg):

vocabBag = []

for i in uniqueVocabPos:

if i not in vocabBag:

vocabBag.append(uniqueVocabPos)

for j in uniqueVocabNeg:

if j not in vocabBag:

vocabBag.append(uniqueVocabNeg)

return vocabBag

# Counting occurrence of each token in each class [count(w|c)]

def countOccurrenceInClass(self, uniqueVocab, vocab):

frequencyOfWord = []

for i in range(len(uniqueVocab)):

frequencyOfWord.append(

vocab.count(uniqueVocab[i])) # counting frequency of each word in class

return frequencyOfWord

# Now calculate probability of each word given class [ count(w|c)+1 ] / [ count(c) + |V| ] } \* p(c)

# prob of each word in pos class: (count of that word in pos class)/(total words in pos class) \* p(c pos)

def calculateProbabiltyOfEachToken(self, m, uniqueVocab, frequencyOfWord, vocabLength, probabilityOfClass):

probOfWords = []

for i in range(len(uniqueVocab)):

probOfWords.append(

(frequencyOfWord[i] + m) / (len(uniqueVocab) + m \* vocabLength) \* probabilityOfClass)

print('Value of "m" used: ' + str(m))

return probOfWords

# creating a dictionary of words along with their probabilities

def createDictionaryofProbabilities(self, uniqueVocab, probOfWords):

probabilityDict = dict(zip(uniqueVocab, probOfWords))

return probabilityDict

# now time to predict test data

# to each word from test data document, we will assign the probability calculated

# from both positive and negative classes above

def classifyTestData(self, data, probabilityDictPos, probabiltyDictNeg):

dataCopy = []

dataArrayPos = []

dataArrayNeg = []

dataProbabilityPos = []

dataProbabilityNeg = []

probSumPos = 0.0

probSumNeg = 0.0

sumArrayPos = []

sumArrayNeg = []

predictedLabelsOfData = []

for w in range(len(data)):

dataCopy.append(data[w])

for i in range(len(data)):

dataCopy[i] = re.sub('[@\_!#$%^&\*()<>?/|{}~:.,"]', '', data[i])

for i in range(len(data)):

dataCopy[i] = data[i].casefold().removesuffix('\t1\n').removesuffix('\t0\n').removesuffix(' ')

# for pos class

for l in dataCopy:

dataArrayPos.append(l.split(' '))

for j in dataArrayPos:

for b in j:

for k, v in probabilityDictPos.items():

if b == k:

dataProbabilityPos.append(v)

probSumPos += float(sum(dataProbabilityPos)) # sum of prob of each word in a given document

dataProbabilityPos.clear()

dataArrayPos.clear()

sumArrayPos.append(probSumPos) # array of sums of probs of all documents

probSumPos = 0.0

# for neg class

for l in dataCopy:

dataArrayNeg.append(l.split(' '))

for j in dataArrayNeg:

for b in j:

for k, v in probabiltyDictNeg.items():

if b == k:

dataProbabilityNeg.append(v)

probSumNeg += float(sum(dataProbabilityNeg)) # sum of prob of each word in a given document

dataProbabilityNeg.clear()

dataArrayNeg.clear()

sumArrayNeg.append(probSumNeg) # array of sums of probs of all documents

probSumNeg = 0.0

for a in range(len(dataCopy)):

if float(sumArrayPos[a]) > float(sumArrayNeg[a]):

predictedLabelsOfData.append('1')

else:

predictedLabelsOfData.append('0')

return predictedLabelsOfData

def calculateAccuracy(self, dataForAccuracy, predictedLabels):

dataLabels = []

correctPredictionCounter = 1

for i in dataForAccuracy:

dataLabels.append(i[-2])

for j in range(len(predictedLabels)):

if predictedLabels[j] == dataLabels[j]:

correctPredictionCounter += 1

accuracy = (correctPredictionCounter / len(dataForAccuracy)) \* 100

print('Actual Labels : ' + str(dataLabels))

print('Accuracy is: ' + str(accuracy) + '%')

return accuracy

yelpFilePath = "C:/Users/Abhishek Sharma/Downloads/sentiment labelled sentences/sentiment labelled sentences/yelp\_labelled.txt"

amazonFilePath = "C:/Users/Abhishek Sharma/Downloads/sentiment labelled sentences/sentiment labelled sentences/amazon\_cells\_labelled.txt"

imdbFilePath = "C:/Users/Abhishek Sharma/Downloads/sentiment labelled sentences/sentiment labelled sentences/imdb\_labelled.txt"

f = open(

amazonFilePath,

'r')

files = f.readlines()

trainData = []

testData = []

# divide data into Train and Test Data in ratio 0.8 : 0.2

for i in range(int(len(files) \* 0.8)):

trainData.append(files[i])

# print(trainData)

for i in range(int(len(files) \* 0.8), int(len(files))):

testData.append(files[i])

nb = NaiveBayes()

classPos = []

classNeg = []

# to segregate each line into positive or negative class based on 0 or 1

nb.classification(trainData, classPos, classNeg)

# Data Cleansing (removing special characters from the data)

classPos = nb.dataCleansing(classPos)

classNeg = nb.dataCleansing(classNeg)

# calculate the probability of each class (p(c))

probabilityOfPositiveClass = float(len(classPos) / (len(classPos) + len(classNeg)))

probabilityOfNegativeClass = float(1 - probabilityOfPositiveClass)

# Adding each word in a list

vocabPos = nb.splitString(classPos)

vocabNeg = nb.splitString(classNeg)

# Creating unique list of words in each class

uniqueVocabPos = nb.createUniqueVocabOfClass(vocabPos)

uniqueVocabNeg = nb.createUniqueVocabOfClass(vocabNeg)

# Creating unique list of words from the whole dataset

vocab = nb.createVocabOfTotalWords(uniqueVocabPos, uniqueVocabNeg)

# Total no of unique words in Vocab

VocabLength = len(vocab)

# Counting occurrence of each token in each class [count(w|c)]

frequencyOfWordPos = nb.countOccurrenceInClass(uniqueVocabPos, vocabPos)

frequencyOfWordNeg = nb.countOccurrenceInClass(uniqueVocabNeg, vocabNeg)

# Now calculate probability of each word given class [ count(w|c)+m ] / [ count(c) + m \* |V| ] } \* p(c)

# prob of each word in pos class: (count of that word in pos class)/(total words in pos class) \* p(c pos)

# To calculate with different values of 'm', change first parameter value here

probOfWordPos = nb.calculateProbabiltyOfEachToken(8, uniqueVocabPos, frequencyOfWordPos, VocabLength,

probabilityOfPositiveClass)

probOfWordNeg = nb.calculateProbabiltyOfEachToken(8, uniqueVocabNeg, frequencyOfWordNeg, VocabLength,

probabilityOfNegativeClass)

# creating a dictionary of words along with their probabilities

probabilityDictionaryPos = nb.createDictionaryofProbabilities(uniqueVocabPos, probOfWordPos)

probabilityDictionaryNeg = nb.createDictionaryofProbabilities(uniqueVocabNeg, probOfWordNeg)

# now time to predict test data

# to each word from test data document, we will assign the probability calculated

# from both positive and negative classes above

predictedLabelsOfTestData = nb.classifyTestData(testData, probabilityDictionaryPos, probabilityDictionaryNeg)

print('Predicted Labels : ' + str(predictedLabelsOfTestData))

# calculate accuracy of prediction

nb.calculateAccuracy(testData, predictedLabelsOfTestData)

amazonAccuracyAvg = [57.49, 57.9, 60, 62.5, 63.5, 64, 64.5, 65, 66]

amazonStdDeviation = [-4.745, -4.70, -3.6, 1.5, 2.5, 2.6, 3.6,3.8, 4.25]

plt.plot(amazonAccuracyAvg, amazonStdDeviation)

plt.show()