#This is a python application that performs the operation feature extraction on a given SMS message using One-Dimensional Ternary Patterns Transformation

#Define a function to generate the UTF-8 value of the given SMS

def genTextUTF8(text):

txtUTF8 = []

for ch in text:

txtUTF8.append(ord(ch))

return txtUTF8

#Define a function to perform this operation

def genTextPatterns(text, i, c):

textList = []

#Convert the text to a list

textList += text

#Remove the ith element/character of the list

a = textList.pop(i)

#Add the removed into the middle/center of the list

textList.insert(c, a)

return textList

#Creating a function to convert binary to decimal

def binaryToDecimal(binaryList):

decimal, n = 0, len(binaryList)

for b in binaryList:

decimal += b\*2\*\*(n-1)

n -= 1

return decimal

#Define a function to extract the unique Histogram Value

def extractUniqueHistogramValues(valueList, histLength):

uniqueHistoValueList = []

for i in range(0, 2\*\*histLength):

if(i in valueList):

uniqueHistoValueList.append(i)

return uniqueHistoValueList

#Define a function to extract the frequency of occurrence of each unique Histogram Value in the value List

def extractHistogramFreq(valueList, uniqHistList):

uniqueHistoFreqList = []

for i in uniqHistList:

uniqueHistoFreqList.append(valueList.count(i))

return uniqueHistoFreqList

import numpy as np

import matplotlib.pyplot as plt

#Read the given SMS

#originalSMS = "A Novel Feature Extraction Approach in SMS Spam Filtering for Mobile Communication: One-Dimensional Ternary Patterns"

originalSMS = "07732584351 - Rodger Burns - MSG = We tried to call you re your reply to our sms for a free nokia mobile + free camcorder. Please call now 08000930705 for delivery tomorrow"

print("========================================== Text Result for Spam Message at P = 10 and B = 2 ======================================================\n")

print("Dataset: 'SMS Spam Corpus v.0.1.txt'\n")

print("The original sms (SPAM): '", originalSMS, "' with length: ", len(originalSMS))

#Remove all flux from the SMS

originalSMS = originalSMS.replace(" ","").replace("'","")

SMS\_length = len(originalSMS)

#Set the initial value of P and the threshold value and perform the 1D-TP transformation

P, B = 10, 2

#Get the number of possible pattern P that can be formed on the SMS

noOfPatterns = SMS\_length//(P+1)

upFeaturesList, lowFeaturesList = [],[]

n, step = 1, P+1

for k in range(0, (noOfPatterns\*step), step):

text = originalSMS[k:n\*step]

print("Pattern ",n," Text is: ", text)

for x in range(0, len(text)):

#call the function to generate the pattern for the given text

textPattern = genTextPatterns(text, x, len(text)//2)

#Partition the text into 3 list(Pl, Pr, Pc)

Lp, Cp, Rp = textPattern[:P//2], textPattern[P//2:P//2+1], textPattern[P//2+1:]

print("The left list is: ", Lp)

print("The centre list is: ", Cp)

print("The right list is: ", Rp)

#Get the UTF-8 of the characters in the given SMS text pattern

Pl, Pr, Pc = [],[], ord(Cp[0])

for i in range(0,len(Lp)):

Pl.insert(i, ord(Lp[i]))

for i in range(0,len(Rp)):

Pr.insert(i, ord(Rp[i]))

print("The left list is: ", Pl)

print("The centre list is: ", Pc)

print("The right list is: ", Pr)

#Comparison of Pc with neighbors (Pi),

Tpl, Tpr = [],[]

for i in range(0,len(Lp)):

if(Pc > (Pl[i] + B)):

Tpl.insert(i, 1)

elif(Pc <= (Pl[i] + B) and Pc >= (Pl[i] - B)):

Tpl.insert(i, 0)

elif(Pc < (Pl[i] - B)):

Tpl.insert(i, -1)

for i in range(0,len(Rp)):

if(Pc > (Pr[i] + B)):

Tpr.insert(i, 1)

elif(Pc <= (Pr[i] + B) and Pc >= (Pr[i] - B)):

Tpr.insert(i, 0)

elif(Pc < (Pr[i] - B)):

Tpr.insert(i, -1)

print("The left list is: ", Tpl)

print("The right list is: ", Tpr)

#Separation positive and negative values,

upF, lowF = [],[]

for i in range(0,len(Lp)):

if(Tpl[i] == -1):

upF.insert(i, 0)

lowF.insert(i, 1)

else:

upF.insert(i, Tpl[i])

lowF.insert(i, 0)

for i in range(0,len(Rp)):

if(Tpr[i] == -1):

upF.insert(len(Lp)+i, 0)

lowF.insert(len(Lp)+i, 1)

else:

upF.insert(len(Lp)+i, Tpr[i])

lowF.insert(len(Lp)+i, 0)

print("The up list is: ", upF)

print("The low list is: ", lowF)

#Conversion of binary values to decimal

upFeatures = binaryToDecimal(upF)

lowFeatures = binaryToDecimal(lowF)

print("The upFeatures is: ", upFeatures)

print("The lowFeatures is: ", lowFeatures)

#Populate the both upFeatures List and lowFeaturesList

upFeaturesList.append(upFeatures)

lowFeaturesList.append(lowFeatures)

#increase the value of n for the pattern to be selected for computation

n += 1

#Define two separate lists for each of the up and low 1D-TP signal histogram values

upHistValues = extractUniqueHistogramValues(upFeaturesList, P)

lowHistValues = extractUniqueHistogramValues(lowFeaturesList, P)

upHistFreq = extractHistogramFreq(upFeaturesList, upHistValues)

lowHistFreq = extractHistogramFreq(lowFeaturesList, lowHistValues)

print("The preprocessed sms: '", originalSMS, "'\n")

print("The number of pattern is: ", noOfPatterns, " and value of P is: ", P, " with threshold value (B) is: ", B, " \n")

print("The UTF-8 values for the sms is: ", genTextUTF8(originalSMS), " \n")

print("The total number of Features for the sms is: ", 2\*\*P, " \n")

print("The up Features 1D-TP signals for the sms is: ", upFeaturesList, " \n")

print("The total number of up Features for the sms is: ", len(upFeaturesList), " \n")

print("The low Features 1D-TP signals for the sms is: ", lowFeaturesList, " \n")

print("The total number of low Features for the sms is: ", len(lowFeaturesList), " \n")

print("The Up Features 1D-TP signals unique Histogram Values for the sms is: ", upHistValues, " \nwith frequency values: ", upHistFreq, " \n")

print("The Low Features 1D-TP signals unique Histogram Values for the sms is: ", lowHistValues, " \nwith frequency values: ", lowHistFreq)

#Plotting a graph to show the output of the UTF-8 of the SMS

fig1 = plt.figure(1)

x = np.arange(0, len(originalSMS))

y = genTextUTF8(originalSMS)

plt.plot(x, y)

#plt.title("Graph of Unicodes of a sample SMS Message at P = 8 and B = 3")

plt.title("Spam Message Unicodes Signal")

plt.ylabel("UTF-8 values of characters")

#Plotting a graph to show the output of the UPFeatures 1D-TP signals for the SMS

fig2 = plt.figure(2)

x = np.arange(0, len(upFeaturesList))

y = upFeaturesList

plt.plot(x, y)

plt.title("1D-TP Upper Features signal for SPAM Message at P = " + str(P) + " and B = " + str(B))

plt.ylabel("UP Features values")

#Plotting a graph to show the output of the lowFeatures 1D-TP signals for the SMS

fig2 = plt.figure(3)

x = np.arange(0, len(lowFeaturesList))

y = lowFeaturesList

plt.plot(x, y)

plt.title("1D-TP Lower Features signal for SPAM Message at P = " + str(P) + " and B = " + str(B))

plt.ylabel("Low Features values")

#Plotting a graph to show the output of the Histogram of the UPFeatures 1D-TP signals for the SMS

fig2 = plt.figure(4)

x = upHistValues

y = upHistFreq

plt.plot(x, y)

plt.title("1D-TP Histogram(Upper) for SPAM Message at P = " + str(P) + " and B = " + str(B))

plt.ylabel("Frequency")

plt.xlabel("Unique Value for Up Features")

#Plotting a graph to show the output of the Histogram of the LOWFeatures 1D-TP signals for the SMS

fig2 = plt.figure(5)

x = lowHistValues

y = lowHistFreq

plt.plot(x, y)

plt.title("1D-TP Histogram(Lower) for SPAM Message at P = " + str(P) + " and B = " + str(B))

plt.ylabel("Frequency")

plt.xlabel("Unique Value for Low Features")

#Show all the graphs

plt.show()

#Press any key to continue

input("Press any key to continue")

import tkinter

from tkinter import \*

from tkinter import messagebox

from tkinter.filedialog import askopenfilename

mainDisplay = tkinter.Tk()

mainDisplay.geometry("850x600")

mainDisplay.title('A novel feature extraction approach in SMS spam filtering for mobile communication: One-Dimensional ternary patterns')

main1 = PanedWindow()

main1.pack(fill=BOTH, expand=1)

main2 = PanedWindow(main1, orient=VERTICAL)

main2.configure(background="green")

main3 = PanedWindow()

text = Text(main2, height=1, width=35)

text.insert(INSERT, "Select File...")

text.place(x=10, y=10)

var = StringVar()

selectLabel = Label(main2, textvariable=var, relief=FLAT)

var.set("<-- Click on Browse File Button to select SMS file.\n it must be in .txt format")

selectLabel.place(x=380, y=8)

var = StringVar()

selectLabel = Label(main2, textvariable=var, relief=FLAT)

var.set("--> ρ")

selectLabel.place(x=650, y=10)

textP = Text(main2, height=1, width=5)

textP.insert(INSERT, "8")

textP.place(x=688, y=10)

var = StringVar()

selectLabel = Label(main2, textvariable=var, relief=FLAT)

var.set("-->β")

selectLabel.place(x=750, y=10)

textB = Text(main2, height=1, width=5)

textB.insert(INSERT, "3")

textB.place(x=788, y=10)

textSMS = Text(main2, height=5, width=103)

textSMS.insert(INSERT, "")

textSMS.place(x=10, y=40)

main1.add(main2)

#Function to Select SMS file (txt) and display it content(s)

global originalSMS

def displayBrowser():

global originalSMS

# Tk().withdraw() # we don't want a full GUI, so keep the root window from appearing

text.delete('1.0', END)

text.place(x=10, y=10)

textSMS.delete('1.0', END)

textSMS.place(x=10, y=40)

filename = askopenfilename()

import os, fnmatch

pattern = "\*.txt"

if fnmatch.fnmatch(filename, pattern):

text.insert(INSERT, filename)

text.place(x=10, y=10)

main1.add(main2)

raw = open(filename, "r") # Get the name of file from here to generate txt file

mess = raw.read()

textSMS.insert(INSERT, mess)

originalSMS = mess

else:

text.insert(INSERT, "Select File...")

text.place(x=10, y=10)

main1.add(main2)

messagebox.showwarning('Error Message', 'File must be a text file')

global P

P = int(textP.get('1.0', END))

upFeaturesList, lowFeaturesList = [], []

#Function were unwanted characters were removed

def removeUnwanterSpace():

global originalSMS

# Remove all flux from the SMS

originalSMS = originalSMS.replace(" ", "")

showSMS = "The given sms: ", originalSMS, " with length: ", len(originalSMS)

# Set the initial value of P

textUnwanterSpace = Text(main2, height=5, width=103)

textUnwanterSpace.insert(INSERT, showSMS)

textUnwanterSpace.place(x=10, y=170)

SMS\_length = len(originalSMS)

# Set the initial value of P

global P

P = int(textP.get('1.0', END))

# Get the number of possible pattern P that can be formed on the SMS

noOfPatterns = SMS\_length // (P + 1)

n, step = 1, P + 1

AA = "The number of pattern is: ", noOfPatterns, " and value of ρ is: ", P

var = StringVar()

extractLabel = Label(main2, textvariable=var, relief=FLAT)

var.set(str(AA).replace("{", "").replace("}", "").replace("'", "").replace("(", "").replace(")", "").replace(",", ""))

extractLabel.place(x=10, y=260)

addAllGetParttern = []

for k in range(0, (noOfPatterns \* step), step):

text = originalSMS[k:n \* step]

aa = "Pattern ", n, " Text is: ", text

addAllGetParttern.append(aa)

print("Pattern ", n, " Text is: ", text)

for x in range(0, len(text)):

# call the function to generate the pattern for the given text

import genTextPatterns

textPattern = genTextPatterns.genTextPatterns(text, x, len(text) // 2)

# Partition the text into 3 list(Pl, Pr, Pc)

Lp, Cp, Rp = textPattern[:P // 2], textPattern[P // 2:P // 2 + 1], textPattern[P // 2 + 1:]

print("The left list is: ", Lp)

bb = "The left list is: ", Lp

addAllGetParttern.append(bb)

print("The centre list is: ", Cp)

cc = "The centre list is: ", Cp

addAllGetParttern.append(cc)

print("The right list is: ", Rp)

aa = "The right list is: ", Rp

addAllGetParttern.append(aa)

# Get the UTF-8 of the characters in the given SMS text pattern

Pl, Pr, Pc = [], [], ord(Cp[0])

for i in range(0, len(Lp)):

Pl.insert(i, ord(Lp[i]))

for i in range(0, len(Rp)):

Pr.insert(i, ord(Rp[i]))

print("The left list is: ", Pl)

aa = "The left list is: ", Pl

addAllGetParttern.append(aa)

print("The centre list is: ", Pc)

aa = "The centre list is: ", Pc

addAllGetParttern.append(aa)

print("The right list is: ", Pr)

aa = "The right list is: ", Pr

addAllGetParttern.append(aa)

# Set the threshold value and perform the 1D-TP transformation

global B

B = int(textB.get('1.0', END))

# Comparison of Pc with neighbors (Pi),

Tpl, Tpr = [], []

for i in range(0, len(Lp)):

if (Pc > (Pl[i] + B)):

Tpl.insert(i, 1)

elif (Pc <= (Pl[i] + B) and Pc >= (Pl[i] - B)):

Tpl.insert(i, 0)

elif (Pc < (Pl[i] - B)):

Tpl.insert(i, -1)

for i in range(0, len(Rp)):

if (Pc > (Pr[i] + B)):

Tpr.insert(i, 1)

elif (Pc <= (Pr[i] + B) and Pc >= (Pr[i] - B)):

Tpr.insert(i, 0)

elif (Pc < (Pr[i] - B)):

Tpr.insert(i, -1)

print("The left list is: ", Tpl)

aa = "The left list is: ", Tpl

addAllGetParttern.append(aa)

print("The right list is: ", Tpr)

aa = "The right list is: ", Tpr

addAllGetParttern.append(aa)

# Separation positive and negative values,

upF, lowF = [], []

for i in range(0, len(Lp)):

if (Tpl[i] == -1):

upF.insert(i, 0)

lowF.insert(i, 1)

else:

upF.insert(i, Tpl[i])

lowF.insert(i, 0)

for i in range(0, len(Rp)):

if (Tpr[i] == -1):

upF.insert(len(Lp) + i, 0)

lowF.insert(len(Lp) + i, 1)

else:

upF.insert(len(Lp) + i, Tpr[i])

lowF.insert(len(Lp) + i, 0)

print("The up list is: ", upF)

aa = "The up list is: ", upF

addAllGetParttern.append(aa)

print("The low list is: ", lowF)

aa = "The low list is: ", lowF

addAllGetParttern.append(aa)

# Conversion of binary values to decimal

# from binaryToDecimal import binaryToDecimal

import binaryToDecimal

upFeatures = binaryToDecimal.binaryToDecimal(upF)

lowFeatures = binaryToDecimal.binaryToDecimal(lowF)

print("The upFeatures is: ", upFeatures)

aa = "The upFeatures is: ", upFeatures

addAllGetParttern.append(aa)

print("The lowFeatures is: ", lowFeatures)

aa = "The lowFeatures is: ", lowFeatures

addAllGetParttern.append(aa)

# Populate the both upFeatures List and lowFeaturesList

upFeaturesList.append(upFeatures)

lowFeaturesList.append(lowFeatures)

# increase the value of n for the pattern to be selected for computation

n += 1

fileList = Listbox(main2, height=18, width=40)

i = 0

for m in addAllGetParttern:

i += 1

fileList.insert(i, str(m).replace("'", "").replace(",", "").replace("(", "").replace(")", ""))

fileList.place(x=10, y=290)

##################################################################

import extractUniqueHistogramValues, extractHistogramFreq

# Define two separate lists for each of the up and low 1D-TP signal histogram values

upHistValues = extractUniqueHistogramValues.extractUniqueHistogramValues(upFeaturesList, P)

lowHistValues = extractUniqueHistogramValues.extractUniqueHistogramValues(lowFeaturesList, P)

upHistFreq = extractHistogramFreq.extractHistogramFreq(upFeaturesList, upHistValues)

lowHistFreq = extractHistogramFreq.extractHistogramFreq(lowFeaturesList, lowHistValues)

print(upHistValues, "Here")

global upFeaturesList1, lowFeaturesList1, upHistValues1, upHistFreq1, lowHistValues1, lowHistFreq1

upFeaturesList1 = upFeaturesList

lowFeaturesList1 = lowFeaturesList

upHistValues1 = upHistValues

upHistFreq1 = upHistFreq

lowHistValues1 = lowHistValues

lowHistFreq1 = lowHistFreq

var = StringVar()

controlLabel = Label(main2, textvariable=var, relief=FLAT)

var.set("<-- Control Buttons. Click on any of the control buttons")

controlLabel.place(x=385, y=260)

genTextUTF8Button = Button(mainDisplay, text="UTF Code & Histograph", height=1, command=controlGenTextUTF8)

genTextUTF8Button.place(x=285, y=290)

upFeaturesListButton = Button(mainDisplay, text="1D-TP signal(Upper)", height=1, command=upFeaturesOf1DT)

upFeaturesListButton.place(x=285, y=340)

lowFeaturesButton = Button(mainDisplay, text="1D-TP signal(Lower)", height=1, command=lowFeaturesOf1DT)

lowFeaturesButton.place(x=285, y=390)

UpFeaturesButton = Button(mainDisplay, text="1D-TP Histogram(Upper)", height=1, command=upHistValues\_upHistFreqOf1DT)

UpFeaturesButton.place(x=285, y=450)

LowFeaturesButton = Button(mainDisplay, text="1D-TP Histogram(Lower)", height=1, command=lowHistValues\_lowHistFreqOf1DT)

LowFeaturesButton.place(x=285, y=510)

#Button to to select SMS file

browseButton = Button(mainDisplay, text="Browse File", height=1, command=displayBrowser)

browseButton.place(x=300, y=10)

#Button to extract Unwanted characters from SMS file

var = StringVar()

extractLabel = Label(main2, textvariable=var, relief=FLAT)

var.set("<-- Click on Remove Unwanted Character(s) Button to remove unwanted character(s) from SMS \nand perform the 1D-TP transformation.")

extractLabel.place(x=298, y=130)

extractButton = Button(mainDisplay, text="Rem. Unwanted Character(s) and Perform the 1D-TP", height=1, command=removeUnwanterSpace)

extractButton.place(x=10, y=130)

# SMS Control Buttons and Displays

import numpy as np

import matplotlib.pyplot as plt

def controlGenTextUTF8():

import genTextUTF8

print("The UTF-8 values for the sms is: ", genTextUTF8.genTextUTF8(originalSMS), " \n")

aa = "The UTF-8 values for the sms is: ", genTextUTF8.genTextUTF8(originalSMS)

text = Text(main2, height=2, width=50)

text.insert(INSERT, str(aa).replace("'", "").replace("{", "").replace("}", "").replace("(", "").replace(")", ""))

text.place(x=423, y=290)

# Plotting a graph to show the output of the UTF-8 of the SMS

fig1 = plt.figure(1)

x = np.arange(0, len(originalSMS))

y = genTextUTF8.genTextUTF8(originalSMS)

plt.plot(x, y, color='green', linestyle='dashed', marker='o', markerfacecolor='blue', markersize=12)

titleIt = "Graph of Unicodes of a sample SMS Message at ρ = "+ str(int(textP.get('1.0', END))) + " and β = "+ str(

int(textB.get('1.0', END)))

plt.title(titleIt)

plt.ylabel("UTF-8 values of characters")

plt.show()

# = 0

def upFeaturesOf1DT(upFeaturesList=None):

global upFeaturesList1, lowFeaturesList1

# upFeaturesList = upFeaturesList

print("The upFeatures 1D-TP signals for the sms is: ", upFeaturesList1, " \n")

aa = "The upFeatures 1D-TP signals for the sms is: ", upFeaturesList1, " \n"

text = Text(main2, height=2, width=50)

text.insert(INSERT, str(aa).replace("'", "").replace("{", "").replace("}", "").replace("(", "").replace(")", ""))

text.place(x=423, y=340)

import plotting\_graph\_output\_UPFeatures\_1D\_TP\_signals\_SMS

titleIt = "Graph of the 1D-TP signal(Upper) for SMS Message at ρ ".replace("'", "") + str(int(textP.get('1.0', END))) + " and β = " + str(int(textB.get('1.0', END)))

plotting\_graph\_output\_UPFeatures\_1D\_TP\_signals\_SMS.plotting\_graph\_output\_UPFeatures\_1D\_TP\_signals\_SMS(upFeaturesList1, titleIt )

def lowFeaturesOf1DT(upFeaturesList=None):

global upFeaturesList1, lowFeaturesList1

# upFeaturesList = upFeaturesList

print("The lowFeatures 1D-TP signals for the sms is: ", lowFeaturesList1, " \n")

aa = "The lowFeatures 1D-TP signals for the sms is: ", lowFeaturesList1, " \n"

text = Text(main2, height=2, width=50)

text.insert(INSERT, str(aa).replace("'", "").replace("{", "").replace("}", "").replace("(", "").replace(")", ""))

text.place(x=423, y=390)

titleIt = "Graph of the 1D-TP signal(Lower) for SMS Message at ρ = " + str(int(textP.get('1.0', END))) + " and β = " + str(int(textB.get('1.0', END)))

import plotting\_graph\_show\_output\_lowFeatures\_1D\_TP\_signals\_SMS

plotting\_graph\_show\_output\_lowFeatures\_1D\_TP\_signals\_SMS.plotting\_graph\_show\_output\_lowFeatures\_1D\_TP\_signals\_SMS(lowFeaturesList1, titleIt )

def upHistValues\_upHistFreqOf1DT():

global upHistValues1, upHistFreq1

# upFeaturesList = upFeaturesList

print("The Up Features 1D-TP signals unique Histogram Values for the sms is: ", upHistValues1,

" \nwith frequency values: ", upHistFreq1, " \n")

aa = "The Up Features 1D-TP signals unique Histogram Values for the sms is: ", upHistValues1, " \nwith frequency values: ", upHistFreq1, " \n"

text = Text(main2, height=3, width=50)

text.insert(INSERT, str(aa).replace("'", "").replace("{", "").replace("}", "").replace("(", "").replace(")", "").replace(",", ""))

text.place(x=428, y=450)

titleIt = "Graph of the 1D-TP Histogram(Upper) for SMS Message at ρ = " + str(int(textP.get('1.0', END))) + " and β = " + str(int(textB.get('1.0', END)))

import plotting\_graph\_show\_output\_Histogram\_UPFeatures\_1D\_TP\_signals\_SMS

plotting\_graph\_show\_output\_Histogram\_UPFeatures\_1D\_TP\_signals\_SMS.plotting\_graph\_show\_output\_Histogram\_UPFeatures\_1D\_TP\_signals\_SMS(upHistValues1, upHistFreq1, titleIt)

def lowHistValues\_lowHistFreqOf1DT():

global lowHistValues1, lowHistFreq1

# upFeaturesList = upFeaturesList

print("The Up Features 1D-TP signals unique Histogram Values for the sms is: ", lowHistValues1,

" \nwith frequency values: ", lowHistFreq1, " \n")

aa = "The Up Features 1D-TP signals unique Histogram Values for the sms is: ", lowHistValues1, " \nwith frequency values: ", lowHistFreq1, " \n"

text = Text(main2, height=3, width=50)

text.insert(INSERT, str(aa).replace("'", "").replace("{", "").replace("}", "").replace("(", "").replace(")", "").replace(",", ""))

text.place(x=428, y=510)

titleIt = "Graph of the 1D-TP Histogram(Lower) for SMS Message at ρ = " + str(int(textP.get('1.0', END))) + " and β = " + str(int(textB.get('1.0', END)))

import plotting\_graph\_show\_output\_Histogram\_LOWFeatures\_1D\_TP\_signals\_SMS

plotting\_graph\_show\_output\_Histogram\_LOWFeatures\_1D\_TP\_signals\_SMS.plotting\_graph\_show\_output\_Histogram\_LOWFeatures\_1D\_TP\_signals\_SMS(lowHistValues1, lowHistFreq1, titleIt)

main1.configure(background='black')

mainDisplay.mainloop()

####### Optimization Source Code #####

## #This is a python application that performs the operation of evaluating the optimal value of B and P for feature extraction on a given sms using One-Dimensional Ternary Patterns Transformation

# Import some other libraries that we'll need

# matplotlib and numpy packages must also be installed

import matplotlib

import numpy as np

import matplotlib.pyplot as plt

import random

import math

#+++++++++++++++++++++++++++ THIS IS FOR THE NORMAL 1D-TP OPERATION ON SMS WHICH IS INVOKED IN THE OPTIMIZATION PART ++++++++++++++++++++++++++++++++++++++++++++++++

#Creating a function to convert binary to decimal

def binaryToDecimal(binaryList):

decimal, n = 0, len(binaryList)

for b in binaryList:

decimal += b\*2\*\*(n-1)

n -= 1

return decimal

#Define a function to perform the 1D-TP feature extraction on SMS

def perform1DTPOnSMS(x1, x2):

#Read the given SMS

#originalSMS = "URGENT!! Your 4\* Costa Del Sol Holiday or £5000 await collection. Call 09050090044 Now toClaim. SAE, TC s, POBox334, Stockport, SK38xh, Cost£1.50/pm, Max10mins"

originalSMS = "Havent planning to buy later. I check already lido only got 530 show in e afternoon. U finish work already?"

#Remove all flux from the SMS

originalSMS = originalSMS.replace(" ","")

print("The given sms: ", originalSMS, " with length: ", len(originalSMS))

SMS\_length = len(originalSMS)

#Set the initial value of P

P = x2

#Get the number of possible pattern P that can be formed on the SMS

noOfPatterns = SMS\_length//(P+1)

print("The number of pattern is: ", noOfPatterns, " and value of P is: ", P)

upFeaturesList, lowFeaturesList = [],[]

n, step = 1, P+1

for k in range(0, (noOfPatterns\*step), step):

text = originalSMS[k:n\*step]

print("Pattern ",n," Text is: ", text)

#Partition the text into 3 list(Pl, Pr, Pc)

Lp, Cp, Rp = text[:P//2], text[P//2:P//2+1], text[P//2+1:]

print("The left list is: ", Lp)

print("The centre list is: ", Cp)

print("The right list is: ", Rp)

#Get the UTF-8 of the characters in the given SMS text pattern

Pl, Pr, Pc = [],[], ord(Cp)

for i in range(0,len(Lp)):

Pl.insert(i, ord(Lp[i]))

for i in range(0,len(Rp)):

Pr.insert(i, ord(Rp[i]))

print("The left list is: ", Pl)

print("The centre list is: ", Pc)

print("The right list is: ", Pr)

#Set the threshold value and perform the 1D-TP transformation

B = x1

print("The value of B is: ", B)

#Comparison of Pc with neighbors (Pi),

Tpl, Tpr = [],[]

for i in range(0,len(Lp)):

if(Pc > (Pl[i] + B)):

Tpl.insert(i, 1)

elif(Pc <= (Pl[i] + B) and Pc >= (Pl[i] - B)):

Tpl.insert(i, 0)

elif(Pc < (Pl[i] - B)):

Tpl.insert(i, -1)

for i in range(0,len(Rp)):

if(Pc > (Pr[i] + B)):

Tpr.insert(i, 1)

elif(Pc <= (Pr[i] + B) and Pc >= (Pr[i] - B)):

Tpr.insert(i, 0)

elif(Pc < (Pr[i] - B)):

Tpr.insert(i, -1)

print("The left list is: ", Tpl)

print("The right list is: ", Tpr)

#Separation positive and negative values,

upF, lowF = [],[]

for i in range(0,len(Lp)):

if(Tpl[i] == -1):

upF.insert(i, 0)

lowF.insert(i, 1)

else:

upF.insert(i, Tpl[i])

lowF.insert(i, 0)

for i in range(0,len(Rp)):

if(Tpr[i] == -1):

upF.insert(len(Lp)+i, 0)

lowF.insert(len(Lp)+i, 1)

else:

upF.insert(len(Lp)+i, Tpr[i])

lowF.insert(len(Lp)+i, 0)

print("The up list is: ", upF)

print("The low list is: ", lowF)

#Conversion of binary values to decimal

upFeatures = binaryToDecimal(upF)

lowFeatures = binaryToDecimal(lowF)

print("The upFeatures is: ", upFeatures)

print("The lowFeatures is: ", lowFeatures)

#Populate the both upFeatures List and lowFeaturesList

upFeaturesList.insert(n-1, upFeatures)

lowFeaturesList.insert(n-1, lowFeatures)

#increase the value of n for the pattern to be selected for computation

n += 1

print("The upFeatures List is: ", upFeaturesList)

print("The lowFeatures List is: ", lowFeaturesList)

avg = 0.0

if(len(upFeaturesList) != 0):

avg = sum(upFeaturesList)/len(upFeaturesList)

return avg

#============================== OPTIMIZATION PART USING SIMULATED ANNEALING ALGORITHM ========================================================

# define the objective function

def f(x):

x1 = x[0]

x2 = x[1]

if(x2 < 6):

x2 = abs(x2) + 6

obj = perform1DTPOnSMS(int(abs(x1)), int(abs(x2)))

return obj

# Start initial value for B and P

x\_start = [2, 10]

# Design variables at mesh points

i1 = np.arange(-10.0,10.0, 0.01)

i2 = np.arange(-10.0, 10.0, 0.01)

x1m, x2m = np.meshgrid(i1, i2)

fm = np.zeros(x1m.shape)

for i in range(x1m.shape[0]):

for j in range(x1m.shape[1]):

fm[i][j] = 0.2 + x1m[i][j]\*\*2 + x2m[i][j]\*\*2 \

- 0.1\*math.cos(6.0\*3.1415\*x1m[i][j]) \

- 0.1\*math.cos(6.0\*3.1415\*x2m[i][j])

# Create a contour plot

plt.figure()

# Specify contour lines

#lines = range(2,52,2)

# Plot contours

CS = plt.contour(x1m, x2m, fm)#,lines)

# Label contours

plt.clabel(CS, inline=1, fontsize=10)

# Add some text to the plot

plt.title('Non-Convex Function')

plt.xlabel('B')

plt.ylabel('P')

##################################################

# Simulated Annealing

##################################################

# Number of cycles

n = 50

# Number of trials per cycle

m = 50

# Number of accepted solutions

na = 0.0

# Probability of accepting worse solution at the start

p1 = 0.7

# Probability of accepting worse solution at the end

p50 = 0.001

# Initial temperature

t1 = -1.0/math.log(p1)

# Final temperature

t50 = -1.0/math.log(p50)

# Fractional reduction every cycle

frac = (t50/t1)\*\*(1.0/(n-1.0))

# Initialize x

x = np.zeros((n+1,2))

x[0] = x\_start

xi = np.zeros(2)

xi = x\_start

na = na + 1.0

# Current best results so far

xc = np.zeros(2)

xc = x[0]

fc = f(xi)

fs = np.zeros(n+1)

fs[0] = fc

# Current temperature

t = t1

# DeltaE Average

DeltaE\_avg = 0.0

for i in range(n):

print('Cycle: ' + str(i) + ' with Temperature: ' + str(t))

for j in range(m):

# Generate new trial points

xi[0] = xc[0] + (random.random() - 0.5) \* 8

xi[1] = xc[1] + (random.random() - 0.5) \* 20

# Clip to upper and lower bounds

xi[0] = max(min(xi[0],8.0),-8.0)

xi[1] = max(min(xi[1],20.0),-20.0)

DeltaE = abs(f(xi)-fc)

if (f(xi)>fc):

# Initialize DeltaE\_avg if a worse solution was found

# on the first iteration

if (i==0 and j==0): DeltaE\_avg = DeltaE

# objective function is worse

# generate probability of acceptance

p = math.exp(-DeltaE/(DeltaE\_avg \* t))

# determine whether to accept worse point

if (random.random()<p):

# accept the worse solution

accept = True

else:

# don't accept the worse solution

accept = False

else:

# objective function is lower, automatically accept

accept = True

if (accept==True):

# update currently accepted solution

xc[0] = xi[0]

xc[1] = xi[1]

fc = f(xc)

# increment number of accepted solutions

na = na + 1.0

# update DeltaE\_avg

DeltaE\_avg = (DeltaE\_avg \* (na-1.0) + DeltaE) / na

# Record the best x values at the end of every cycle

x[i+1][0] = xc[0]

x[i+1][1] = xc[1]

fs[i+1] = fc

# Lower the temperature for next cycle

t = frac \* t

#Normalize the value of P

if(xc[1] < 6):

xc[1] = abs(xc[1]) + 6

# print solution

print('============================== Optimized values ==============================================')

print('Best solution with optimal value B is: ' + str(int(abs(xc[0]))) + ' and P is: ' + str(int(abs(xc[1]))))

print('Best objective(1D-TP): ' + str(int(fc)))

plt.plot(x[:,0],x[:,1],'y-o')

plt.savefig('contour.png')

fig = plt.figure()

plt.title("Graph of the Optimal solution for both P and B")

ax1 = fig.add\_subplot(211)

ax1.plot(fs,'r.-')

ax1.legend(['Objective (1D-TP)'])

ax2 = fig.add\_subplot(212)

ax2.plot(x[:,0],'b.-')

ax2.plot(x[:,1],'g--')

ax2.legend(['B','P'])

# Save the figure as a PNG

plt.savefig('optimization.png')

plt.show()

import os

import tkinter

from tkinter import \*

from tkinter import messagebox

from tkinter.filedialog import askopenfilename

import numpy as np

import matplotlib.pyplot as plt

import random

import math

try:

if not os.path.exists("generated/upper/optimization"):

os.makedirs("generated/upper/optimization")

except OSError:

print('Error: Creating Folder1')

try:

if not os.path.exists("generated/lower/optimization"):

os.makedirs("generated/lower/optimization")

except OSError:

print('Error: Creating Folder1')

try:

if not os.path.exists("generated/bestObjective/optimization"):

os.makedirs("generated/bestObjective/optimization")

except OSError:

print('Error: Creating Folder1')

try:

if not os.path.exists("generated/left/optimization"):

os.makedirs("generated/left/optimization")

except OSError:

print('Error: Creating Folder1')

try:

if not os.path.exists("generated/center/optimization"):

os.makedirs("generated/center/optimization")

except OSError:

print('Error: Creating Folder1')

try:

if not os.path.exists("generated/right/optimization"):

os.makedirs("generated/right/optimization")

except OSError:

print('Error: Creating Folder1')

try:

if not os.path.exists("generated/upperFeature/optimization"):

os.makedirs("generated/upperFeature/optimization")

except OSError:

print('Error: Creating Folder1')

try:

if not os.path.exists("generated/lowerFeature/optimization"):

os.makedirs("generated/lowerFeature/optimization")

except OSError:

print('Error: Creating Folder1')

try:

if not os.path.exists("generated/sms/optimization"):

os.makedirs("generated/sms/optimization")

except OSError:

print('Error: Creating Folder1')

mainDisplay = tkinter.Tk()

mainDisplay.geometry("850x600")

mainDisplay.title('A novel feature extraction approach in SMS spam filtering for mobile communication: One-Dimensional ternary patterns')

main1 = PanedWindow()

main1.pack(fill=BOTH, expand=1)

main2 = PanedWindow(main1, orient=VERTICAL)

main2.configure(background="green")

main3 = PanedWindow()

text = Text(main2, height=1, width=35)

text.insert(INSERT, "Select File...")

text.place(x=10, y=10)

var = StringVar()

selectLabel = Label(main2, textvariable=var, relief=FLAT)

var.set("<-- Click on Browse File Button to select Message \nfile. it must be in .txt format")

selectLabel.place(x=380, y=8)

var = StringVar()

selectLabel = Label(main2, textvariable=var, relief=FLAT)

var.set("--> ρ")

selectLabel.place(x=650, y=10)

textP = Text(main2, height=1, width=5)

textP.insert(INSERT, "8")

textP.place(x=688, y=10)

var = StringVar()

selectLabel = Label(main2, textvariable=var, relief=FLAT)

var.set("-->β")

selectLabel.place(x=750, y=10)

textB = Text(main2, height=1, width=5)

textB.insert(INSERT, "6")

textB.place(x=788, y=10)

textSMS = Text(main2, height=5, width=103)

textSMS.insert(INSERT, "")

textSMS.place(x=10, y=40)

main1.add(main2)

#Function to Select SMS file (txt) and display it content(s)

global originalSMS

def displayBrowser():

global originalSMS

# Tk().withdraw() # we don't want a full GUI, so keep the root window from appearing

text.delete('1.0', END)

text.place(x=10, y=10)

textSMS.delete('1.0', END)

textSMS.place(x=10, y=40)

filename = askopenfilename()

import os, fnmatch

pattern = "\*.txt"

if fnmatch.fnmatch(filename, pattern):

text.insert(INSERT, filename)

text.place(x=10, y=10)

main1.add(main2)

raw = open(filename, "r") # Get the name of file from here to generate txt file

mess = raw.read()

textSMS.insert(INSERT, mess)

originalSMS = mess

else:

text.insert(INSERT, "Select File...")

text.place(x=10, y=10)

main1.add(main2)

messagebox.showwarning('Error Message', 'File must be a text file')

# global P

# P = int(textP.get('1.0', END))

upFeaturesList, lowFeaturesList = [], []

#Function were unwanted characters were removed

def removeUnwanterSpace():

# Start initial value for B and P

x\_start = [int(textP.get('1.0', END)), int(textB.get('1.0', END))]

global originalSMS

# Remove all flux from the SMS

originalSMS = originalSMS.replace(" ", "")

showSMS = "The given Message: ", originalSMS, " with length: ", len(originalSMS)

# Set the initial value of P

textUnwanterSpace = Text(main2, height=5, width=103)

textUnwanterSpace.insert(INSERT, showSMS)

textUnwanterSpace.place(x=10, y=170)

# Design variables at mesh points

i1 = np.arange(-10.0, 10.0, 0.01)

i2 = np.arange(-10.0, 10.0, 0.01)

x1m, x2m = np.meshgrid(i1, i2)

fm = np.zeros(x1m.shape)

for i in range(x1m.shape[0]):

for j in range(x1m.shape[1]):

fm[i][j] = 0.2 + x1m[i][j] \*\* 2 + x2m[i][j] \*\* 2 \

- 0.1 \* math.cos(6.0 \* 3.1415 \* x1m[i][j]) \

- 0.1 \* math.cos(6.0 \* 3.1415 \* x2m[i][j])

# Create a contour plot

plt.figure()

# Specify contour lines

# lines = range(2,52,2)

# Plot contours

CS = plt.contour(x1m, x2m, fm) # ,lines)

# Label contours

plt.clabel(CS, inline=1, fontsize=10)

# Add some text to the plot

plt.title('Non-Convex Function')

plt.xlabel('β')

plt.ylabel('ρ')

##################################################

# Simulated Annealing

##################################################

# Number of cycles 50

n = 2

# Number of trials per cycle

m = 2

# Number of accepted solutions

na = 0.0

# Probability of accepting worse solution at the start

p1 = 0.7

# Probability of accepting worse solution at the end

p50 = 0.001

# Initial temperature

t1 = -1.0 / math.log(p1)

# Final temperature

t50 = -1.0 / math.log(p50)

# Fractional reduction every cycle

frac = (t50 / t1) \*\* (1.0 / (n - 1.0))

# Initialize x

x = np.zeros((n + 1, 2))

x[0] = x\_start

xi = np.zeros(2)

xi = x\_start

na = na + 1.0

# Current best results so far

xc = np.zeros(2)

xc = x[0]

fc = f(xi, 0, n, 0)

fs = np.zeros(n + 1)

fs[0] = fc

# Current temperature

t = t1

# DeltaE Average

DeltaE\_avg = 0.0

fwriter = open("generated/bestObjective/optimization/best\_objective.txt", "a+")

for i in range(n):

print('Cycle: ' + str(i) + ' with Temperature: ' + str(t))

fwriter.write('Cycle: ' + str(i) + ' with Temperature: ' + str(t)+ '\n')

for j in range(m):

# Generate new trial points

xi[0] = xc[0] + (random.random() - 0.5) \* 8

xi[1] = xc[1] + (random.random() - 0.5) \* 20

# Clip to upper and lower bounds

xi[0] = max(min(xi[0], 8.0), -8.0)

xi[1] = max(min(xi[1], 20.0), -20.0)

DeltaE = abs(f(xi, i, n, 1) - fc)

if (f(xi, i, n, 1) > fc):

# Initialize DeltaE\_avg if a worse solution was found

# on the first iteration

if (i == 0 and j == 0): DeltaE\_avg = DeltaE

# objective function is worse

# generate probability of acceptance

Del\_avg\_T = (DeltaE\_avg \* t)

if (DeltaE\_avg \* t) <= 0:

Del\_avg\_T = 1

p = math.exp(-DeltaE / Del\_avg\_T)

# determine whether to accept worse point

if (random.random() < p):

# accept the worse solution

accept = True

else:

# don't accept the worse solution

accept = False

else:

# objective function is lower, automatically accept

accept = True

if (accept == True):

# update currently accepted solution

xc[0] = xi[0]

xc[1] = xi[1]

fc = f(xc, i, n, 2)

# increment number of accepted solutions

na = na + 1.0

# update DeltaE\_avg

DeltaE\_avg = (DeltaE\_avg \* (na - 1.0) + DeltaE) / na

# Record the best x values at the end of every cycle

x[i + 1][0] = xc[0]

x[i + 1][1] = xc[1]

fs[i + 1] = fc

# Lower the temperature for next cycle

t = frac \* t

# Normalize the value of P

if (xc[1] < 6):

xc[1] = abs(xc[1]) + 6

# print solution

print('Best solution with optimal value β is: ' + str(int(textP.get('1.0', END))) + ' and ρ is: ' + str(int(textB.get('1.0', END))))

aa = 'Best solution with optimal value β is: ' + str(int(textP.get('1.0', END))) + ' and ρ is: ' + str(int(textB.get('1.0', END)))

fwriter.write(str(''.join(aa).encode('utf-8'))+'\n')

text = Text(main2, height=2, width=50)

text.insert(INSERT, str(aa).replace("'", "").replace("{", "").replace("}", "").replace("(", "").replace(")", ""))

text.place(x=423, y=290)

print('Best objective(1D-TP): ' + str(int(fc)))

aa = 'Best objective(1D-TP): ' + str(int(fc))

print(lowFeaturesListAll)

ij = 0

for upL in upFeatureListAll:

ij = ij + 1

if ij == len(upFeatureListAll):

print(upL,' SIZE ',len(upFeatureListAll))

fwriter.write(aa+'\n')

fwriter.close()

text = Text(main2, height=2, width=50)

text.insert(INSERT, str(aa).replace("'", "").replace("{", "").replace("}", "").replace("(", "").replace(")", ""))

text.place(x=423, y=340)

plt.plot(x[:, 0], x[:, 1], 'y-o')

plt.savefig('contour.png')

fig = plt.figure()

plt.title("Graph of the Optimal solution for both ρ and β")

ax1 = fig.add\_subplot(211)

ax1.plot(fs, 'r.-')

ax1.legend(['Objective (1D-TP)'])

ax2 = fig.add\_subplot(212)

ax2.plot(x[:, 0], 'b.-')

ax2.plot(x[:, 1], 'g--')

ax2.legend(['β', 'ρ'])

# Save the figure as a PNG

plt.savefig('generated/optimization.png')

plt.show()

# SMS\_length = len(originalSMS)

# # Set the initial value of P

# global P

#

# P = int(textP.get('1.0', END))

# # Get the number of possible pattern P that can be formed on the SMS

# noOfPatterns = SMS\_length // (P + 1)

# n, step = 1, P + 1

# AA = "The number of pattern is: ", noOfPatterns, " and value of P is: ", P

#

# var = StringVar()

# extractLabel = Label(main2, textvariable=var, relief=FLAT)

# var.set(str(AA).replace("{", "").replace("}", "").replace("'", "").replace("(", "").replace(")", "").replace(",", ""))

# extractLabel.place(x=10, y=260)

#

# addAllGetParttern = []

##################################################################

# import extractUniqueHistogramValues, extractHistogramFreq

# # Define two separate lists for each of the up and low 1D-TP signal histogram values

# upHistValues = extractUniqueHistogramValues.extractUniqueHistogramValues(upFeaturesList, P)

# lowHistValues = extractUniqueHistogramValues.extractUniqueHistogramValues(lowFeaturesList, P)

# upHistFreq = extractHistogramFreq.extractHistogramFreq(upFeaturesList, upHistValues)

# lowHistFreq = extractHistogramFreq.extractHistogramFreq(lowFeaturesList, lowHistValues)

# print(upHistValues, "Here")

# global upFeaturesList1, lowFeaturesList1, upHistValues1, upHistFreq1, lowHistValues1, lowHistFreq1

#

# upFeaturesList1 = upFeaturesList

# lowFeaturesList1 = lowFeaturesList

# upHistValues1 = upHistValues

# upHistFreq1 = upHistFreq

# lowHistValues1 = lowHistValues

# lowHistFreq1 = lowHistFreq

# var = StringVar()

# controlLabel = Label(main2, textvariable=var, relief=FLAT)

# var.set("<-- Control Buttons. Click on any of the control buttons")

# controlLabel.place(x=385, y=260)

#

# genTextUTF8Button = Button(mainDisplay, text="UTF Code & Histograph", height=1, command=controlGenTextUTF8)

# genTextUTF8Button.place(x=285, y=290)

#

# upFeaturesListButton = Button(mainDisplay, text="1D-TP signal(Upper)", height=1, command=upFeaturesOf1DT)

# upFeaturesListButton.place(x=285, y=340)

#

# lowFeaturesButton = Button(mainDisplay, text="1D-TP signal(Lower)", height=1, command=lowFeaturesOf1DT)

# lowFeaturesButton.place(x=285, y=390)

#

# UpFeaturesButton = Button(mainDisplay, text="1D-TP Histogram(Upper)", height=1, command=upHistValues\_upHistFreqOf1DT)

# UpFeaturesButton.place(x=285, y=450)

#

# LowFeaturesButton = Button(mainDisplay, text="1D-TP Histogram(Lower)", height=1, command=lowHistValues\_lowHistFreqOf1DT)

# LowFeaturesButton.place(x=285, y=510)

#Define a function to perform the 1D-TP feature extraction on SMS

addAllGetParttern = []

upFeatureListAll = []

lowFeaturesListAll = []

def perform1DTPOnSMS(x1, x2, ii, nn, mm):

import binaryToDecimal

global originalSMS

print("The given sms: ", originalSMS, " with length: ", len(originalSMS))

fwritersms = open("generated/sms/optimization/sms.txt", "a+")

fwritersms.write("The given sms: "+ originalSMS+ " with length: "+ str(len(originalSMS))+'\n')

SMS\_length = len(originalSMS)

# Set the initial value of P

P = x2

# Get the number of possible pattern P that can be formed on the SMS

noOfPatterns = SMS\_length // (P + 1)

print("The number of pattern is: ", noOfPatterns, " and value of ρ is: ", P)

aa = 'The number of pattern is: '+ str(noOfPatterns), ' and value of ρ is: '+ str(P)

fwritersms.write(str(''.join(aa).encode('utf-8')))

fwritersms.close()

addAllGetParttern.append(aa)

upFeaturesList, lowFeaturesList = [], []

n, step = 1, P + 1

fwriterupper = open("generated/upper/optimization/upper.txt", "a+")

fwriterupperFeature = open("generated/upperFeature/optimization/upperFeature.txt", "a+")

fwriterlower = open("generated/lower/optimization/lower.txt", "a+")

fwriterlowerFeature = open("generated/lowerFeature/optimization/lowerFeature.txt", "a+")

######################################

fwriterleft = open("generated/left/optimization/left.txt", "a+")

fwritercenter = open("generated/center/optimization/center.txt", "a+")

fwriterright = open("generated/right/optimization/right.txt", "a+")

# fwriterlowerFeature = open("generated/lowerFeature/lowerFeature.txt", "a+")

for k in range(0, (noOfPatterns \* step), step):

text = originalSMS[k:n \* step]

aa = "Pattern "+ str(n)+" Text is: "+ str(text)

fwriterupper.write(aa+'\n')

fwriterupperFeature.write(aa+'\n')

fwriterlower.write(aa+'\n')

fwriterlowerFeature.write(aa+'\n')

fwriterleft.write(aa+'\n')

fwritercenter.write(aa+'\n')

fwriterright.write(aa+'\n')

addAllGetParttern.append(aa)

print("Pattern ", n, " Text is: ", text)

# Partition the text into 3 list(Pl, Pr, Pc)

Lp, Cp, Rp = text[:P // 2], text[P // 2:P // 2 + 1], text[P // 2 + 1:]

print("The left list is: ", Lp)

aa = "The left list is: "+ str(Lp)

fwriterleft.write(aa+'\n')

addAllGetParttern.append(aa)

print("The centre list is: ", Cp)

aa = "The centre list is: "+str(Cp)

fwritercenter.write(aa+'\n')

addAllGetParttern.append(aa)

print("The right list is: ", Rp)

aa = "The right list is: "+str(Rp)

fwriterright.write(aa+'\n')

addAllGetParttern.append(aa)

# Get the UTF-8 of the characters in the given SMS text pattern

Pl, Pr, Pc = [], [], ord(Cp)

for i in range(0, len(Lp)):

Pl.insert(i, ord(Lp[i]))

for i in range(0, len(Rp)):

Pr.insert(i, ord(Rp[i]))

print("The left list is: ", Pl)

aa = "The left list is: "+ str(Pl)

fwriterleft.write(aa+'\n')

addAllGetParttern.append(aa)

print("The centre list is: ", Pc)

aa = "The centre list is: "+str(Pc)

fwritercenter.write(aa+'\n')

addAllGetParttern.append(aa)

print("The right list is: ", Pr)

aa = "The right list is: "+str(Pr)

fwriterright.write(aa+'\n')

addAllGetParttern.append(aa)

# Set the threshold value and perform the 1D-TP transformation

B = x1

print("The value of β is: ", B)

aa = "The value of β is: "+str(B)

fwriterupper.write(str(''.join(aa).encode('utf-8'))+'\n')

fwriterupperFeature.write(str(''.join(aa).encode('utf-8'))+'\n')

fwriterlower.write(str(''.join(aa).encode('utf-8'))+'\n')

fwriterlowerFeature.write(str(''.join(aa).encode('utf-8'))+'\n')

fwriterleft.write(str(''.join(aa).encode('utf-8'))+'\n')

fwritercenter.write(str(''.join(aa).encode('utf-8'))+'\n')

fwriterright.write(str(''.join(aa).encode('utf-8'))+'\n')

addAllGetParttern.append(str(''.join(aa).encode('utf-8'))+'\n')

# Comparison of Pc with neighbors (Pi),

Tpl, Tpr = [], []

for i in range(0, len(Lp)):

if (Pc > (Pl[i] + B)):

Tpl.insert(i, 1)

elif (Pc <= (Pl[i] + B) and Pc >= (Pl[i] - B)):

Tpl.insert(i, 0)

elif (Pc < (Pl[i] - B)):

Tpl.insert(i, -1)

for i in range(0, len(Rp)):

if (Pc > (Pr[i] + B)):

Tpr.insert(i, 1)

elif (Pc <= (Pr[i] + B) and Pc >= (Pr[i] - B)):

Tpr.insert(i, 0)

elif (Pc < (Pr[i] - B)):

Tpr.insert(i, -1)

print("The left list is: ", Tpl)

aa = "The left list is: "+str(Tpl)

fwriterleft.write(aa+'\n')

addAllGetParttern.append(aa)

print("The right list is: ", Tpr)

aa = "The right list is: "+str(Tpr)

fwriterright.write(aa+'\n')

addAllGetParttern.append(aa)

# Separation positive and negative values,

upF, lowF = [], []

for i in range(0, len(Lp)):

if (Tpl[i] == -1):

upF.insert(i, 0)

lowF.insert(i, 1)

else:

upF.insert(i, Tpl[i])

lowF.insert(i, 0)

for i in range(0, len(Rp)):

if (Tpr[i] == -1):

upF.insert(len(Lp) + i, 0)

lowF.insert(len(Lp) + i, 1)

else:

upF.insert(len(Lp) + i, Tpr[i])

lowF.insert(len(Lp) + i, 0)

print("The up list is: ", upF)

aa = "The up list is: "+str(upF)

fwriterupper.write(aa+'\n')

addAllGetParttern.append(aa)

print("The low list is: ", lowF)

aa = "The low list is: "+str(lowF)

fwriterlower.write(aa+'\n')

addAllGetParttern.append(aa)

# Conversion of binary values to decimal

upFeatures = binaryToDecimal.binaryToDecimal(upF)

lowFeatures = binaryToDecimal.binaryToDecimal(lowF)

print("The upFeatures is: ", upFeatures)

aa = "The upFeatures is: "+str(upFeatures)

fwriterupperFeature.write(aa+'\n')

addAllGetParttern.append(aa)

print("The lowFeatures is: ", lowFeatures)

aa = "The lowFeatures is: "+str(lowFeatures)

fwriterlowerFeature.write(aa+'\n')

addAllGetParttern.append(aa)

# Populate the both upFeatures List and lowFeaturesList

upFeaturesList.insert(n - 1, upFeatures)

lowFeaturesList.insert(n - 1, lowFeatures)

# increase the value of n for the pattern to be selected for computation

n += 1

print("The upFeatures List is: ", upFeaturesList)

aa = "The upFeatures List is: "+str(upFeaturesList)

if mm == 2 and ii == 1:

upFeatureListAll.append(upFeaturesList)

lowFeaturesListAll.append(lowFeaturesList)

fwriterupperFeature.write(aa+'\n')

addAllGetParttern.append(aa)

print("The lowFeatures List is: ", lowFeaturesList)

aa = "The lowFeatures List is: "+str(lowFeaturesList)

fwriterlowerFeature.write(aa+'\n')

addAllGetParttern.append(aa)

avg = 0.0

# fileList = Listbox(main2, height=18, width=40)

# i = 0

# for m in addAllGetParttern:

# i += 1

# fileList.insert(i, str(m).replace("'", "").replace(",", "").replace("(", "").replace(")", ""))

#

fwriterupper.close()

fwriterupperFeature.close()

fwriterlower.close()

fwriterlowerFeature.close()

fwriterleft.close()

fwritercenter.close()

fwriterright.close()

# fileList.place(x=10, y=290)

if (len(upFeaturesList) != 0):

avg = sum(upFeaturesList) / len(upFeaturesList)

return avg

#============================== OPTIMIZATION PART USING SIMULATED ANNEALING ALGORITHM ========================================================

# define the objective function

def f(x, i, n, m):

x1 = x[0]

x2 = x[1]

if(x2 < 6):

x2 = abs(x2) + 6

obj = perform1DTPOnSMS(int(abs(x1)), int(abs(x2)), i, n, m)

return obj

# print('ALEXXXX ', upFeatureListAll)

#Button to to select SMS file

browseButton = Button(mainDisplay, text="Browse File", height=1, command=displayBrowser)

browseButton.place(x=300, y=10)

#Button to extract Unwanted characters from SMS file

var = StringVar()

extractLabel = Label(main2, textvariable=var, relief=FLAT)

var.set("<-- Click on Remove Unwanted Character(s) Button to remove unwanted character(s) from Message and\n perform the Optimization.")

extractLabel.place(x=298, y=130)

extractButton = Button(mainDisplay, text="Rem. Unwanted Character(s) and Perform the 1D-TP", height=1, command=removeUnwanterSpace)

extractButton.place(x=10, y=130)

# SMS Control Buttons and Displays

main1.configure(background='black')

mainDisplay.mainloop()

######## Classifiers Source Code #################

from sklearn.naive\_bayes import GaussianNB

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import confusion\_matrix

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns; sns.set()

# Import data

training = pd.read\_csv('iris\_train.csv')

test = pd.read\_csv('iris\_test.csv')

# Create the X, Y, Training and Test

xtrain = training.drop('Species', axis=1)

ytrain = training.loc[:, 'Species']

xtest = test.drop('Species', axis=1)

ytest = test.loc[:, 'Species']

# Init the Gaussian Classifier

model = GaussianNB()

# Train the model

model.fit(xtrain, ytrain)

# Predict Output

pred = model.predict(xtest)

# Plot Confusion Matrix

mat = confusion\_matrix(pred, ytest)

names = np.unique(pred)

sns.heatmap(mat, square=True, annot=True, fmt='d', cbar=False,

xticklabels=names, yticklabels=names)

plt.xlabel('Truth')

plt.ylabel('Predicted')

from builtins import print

import pandas as pd

import numpy as np

from nltk.tokenize import word\_tokenize

from nltk import pos\_tag

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

from sklearn.linear\_model import LogisticRegression

from sklearn.preprocessing import LabelEncoder

from collections import defaultdict

from nltk.corpus import wordnet as wn

from nltk.tokenize import word\_tokenize, sent\_tokenize

from nltk.corpus import stopwords

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.ensemble import RandomForestClassifier

from sklearn import model\_selection, naive\_bayes, svm

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

#Set Random seed

# np.random.seed(500)

# Add the Data using pandas

Corpus = pd.read\_csv(r"..\generated\data\lowFeature.csv", encoding='latin-1')

# print(Corpus)

# Step - 1a : Remove blank rows if any.

Corpus['lowFeatureList'].dropna(inplace=True)

# print(Corpus)

# Step - 1b : Change all the text to lower case. This is required as python interprets 'dog' and 'DOG' differently

# Corpus['v2'] = [entry.lower() for entry in Corpus['v2']]

#

# # Step - 1c : Tokenization : In this each entry in the corpus will be broken into set of words

# Corpus['v2']= [word\_tokenize(entry) for entry in Corpus['v2']]

# Step - 1d : Remove Stop words, Non-Numeric and perfom Word Stemming/Lemmenting.

# WordNetLemmatizer requires Pos tags to understand if the word is noun or verb or adjective etc. By default it is set to Noun

# tag\_map = defaultdict(lambda : wn.NOUN)

# tag\_map['J'] = wn.ADJ

# tag\_map['V'] = wn.VERB

# tag\_map['R'] = wn.ADV

# for index,entry in enumerate(Corpus['v2']):

# # Declaring Empty List to store the words that follow the rules for this step

# Final\_words = []

# # Initializing WordNetLemmatizer()

# word\_Lemmatized = WordNetLemmatizer()

# # pos\_tag function below will provide the 'tag' i.e if the word is Noun(N) or Verb(V) or something else.

# for word, tag in pos\_tag(entry):

# # Below condition is to check for Stop words and consider only alphabets

# if word not in stopwords.words('english') and word.isalpha():

# word\_Final = word\_Lemmatized.lemmatize(word,tag\_map[tag[0]])

# Final\_words.append(word)

# # The final processed set of words for each iteration will be stored in 'text\_final'

# Corpus.loc[index,'text\_final'] = str(Final\_words)

# Step - 2: Split the model into Train and Test Data set

Train\_X, Test\_X, Train\_Y, Test\_Y = model\_selection.train\_test\_split(Corpus['lowFeatureList'],Corpus['Spam\_Ham'],test\_size=0.4, random\_state=0)

#

# Step - 3: Label encode the target variable - This is done to transform Categorical data of string type in the data set into numerical values

Encoder = LabelEncoder()

Train\_Y = Encoder.fit\_transform(Train\_Y)

Test\_Y = Encoder.fit\_transform(Test\_Y)

#

#

#

#

# # Step - 4: Vectorize the words by using TF-IDF Vectorizer - This is done to find how important a word in document is in comaprison to the corpus

Tfidf\_vect = TfidfVectorizer(max\_features=1000)

Tfidf\_vect.fit(Corpus['lowFeatureList'])

Train\_X\_Tfidf = Tfidf\_vect.transform(Train\_X)

Test\_X\_Tfidf = Tfidf\_vect.transform(Test\_X)

# print('HERRRR')

# print(Train\_X\_Tfidf)

# print(' kkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkk ')

# print(Train\_Y)

# Step - 5: Now we can run different algorithms to classify out data check for accuracy

from sklearn.neural\_network import MLPClassifier

# mlp = MLPClassifier(hidden\_layer\_sizes=(10, 10, 10), max\_iter=1000)

# mlp.fit(Train\_X\_Tfidf,Train\_Y)

#

# predictions = mlp.predict(Test\_X\_Tfidf)

from sklearn.metrics import classification\_report, confusion\_matrix

# print(confusion\_matrix(Test\_Y,predictions))

# print(classification\_report(Test\_Y,predictions))

# Classifier - Algorithm - Naive Bayes

# fit the training dataset on the classifier

# Naive = naive\_bayes.MultinomialNB()

# Naive.fit(Train\_X\_Tfidf,Train\_Y)

#

# # predict the labels on validation dataset

# predictions\_NB = Naive.predict(Test\_X\_Tfidf)

#

# # Use accuracy\_score function to get the accuracy

# print("Naive Bayes Accuracy Score -> ",accuracy\_score(predictions\_NB, Test\_Y)\*100)

# print(' ',classification\_report(Test\_Y, predictions\_NB))

#

# print(confusion\_matrix(Test\_Y,predictions\_NB))

# # print(classification\_report(Test\_Y, y\_pred))

#

#

# # # Classifier - Algorithm - SVM

# # # fit the training dataset on the classifier

SVM = svm.SVC(C=1.0, kernel='linear', degree=2, gamma='auto')

SVM.fit(Train\_X\_Tfidf,Train\_Y)

# predict the labels on validation dataset

predictions\_SVM = SVM.predict(Test\_X\_Tfidf)

# Use accuracy\_score function to get the accuracy

print("SVM Accuracy Score -> ",accuracy\_score(predictions\_SVM, Test\_Y)\*100)

print(' SVM ',classification\_report(Test\_Y, predictions\_SVM))

lr = LogisticRegression()

lr.fit(Train\_X\_Tfidf, Train\_Y)

predictions\_LR = lr.predict(Test\_X\_Tfidf)

# Cross validation

# accuracy = cross\_validate(Random\_Forest\_model,Train\_X\_Tfidf,Train\_Y,cv=10)['test\_score']

print('Random accuracy is: ',accuracy\_score(predictions\_LR, Test\_Y) \* 100,'%')

print(' SVM ',classification\_report(Test\_Y, predictions\_LR))

# from sklearn.ensemble import RandomForestClassifier

# from sklearn.preprocessing import LabelEncoder

# from sklearn.model\_selection import cross\_validate

# Random\_Forest\_model = RandomForestClassifier(n\_estimators=100,criterion="entropy")

# Random\_Forest\_model.fit(Train\_X\_Tfidf, Train\_Y, sample\_weight=None)

#

# predictions\_RF = Random\_Forest\_model.predict(Test\_X\_Tfidf)

# #Cross validation

# # accuracy = cross\_validate(Random\_Forest\_model,Train\_X\_Tfidf,Train\_Y,cv=10)['test\_score']

# #

# # print('Random accuracy is: ',sum(accuracy)/len(accuracy)\*100,'%')

# print("RF Accuracy Score -> ",accuracy\_score(predictions\_RF, Test\_Y)\*100)

#

#

#

# from sklearn import tree

# # #

# clf = tree.DecisionTreeClassifier(criterion='entropy')

# clf = clf.fit(Train\_X\_Tfidf,Train\_Y)

#

# # SVM.fit(Train\_X\_Tfidf,Train\_Y)

#

# # predict the labels on validation dataset

# predictions\_DT = clf.predict(Test\_X\_Tfidf)

#

#

#

# # Use accuracy\_score function to get the accuracy

# print("DT Accuracy Score -> ",accuracy\_score(predictions\_DT, Test\_Y)\*100)

# from sklearn.metrics import classification\_report,confusion\_matrix

# print(classification\_report(Test\_Y,predictions\_DT))

# print(confusion\_matrix(Test\_Y,predictions\_DT))

#

#

# from sklearn.neural\_network import MLPClassifier

# clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden\_layer\_sizes=(5, 2), random\_state=1)

#

# clf.fit(Train\_X\_Tfidf,Train\_Y)

# MLPClassifier(alpha=1e-05, hidden\_layer\_sizes=(5, 2), random\_state=1,

# solver='lbfgs')

#

# predictions\_FFNN = clf.predict(Test\_X\_Tfidf)

# print("FFNN Accuracy Score -> ",accuracy\_score(predictions\_FFNN, Test\_Y)\*100)

# # print(Test\_X\_Tfidf)

# # from sklearn.kernel\_approximation import RBFSampler

# # from sklearn.linear\_model import SGDClassifier

# #

# # rbf\_feature = RBFSampler(gamma=1, random\_state=1)

# # X\_features = rbf\_feature.fit\_transform(Test\_X\_Tfidf)

# # clf = SGDClassifier(max\_iter=5)

# # clf.fit(Train\_X\_Tfidf, Test\_Y)

# # SGDClassifier(max\_iter=100)

# # # clf.score(X\_features, Test\_Y)

# #

# # predictions\_RB = clf.predict(Test\_X\_Tfidf)

# # print("RB Accuracy Score -> ",accuracy\_score(predictions\_RB, Test\_Y)\*100)

#

#

# # from sklearn.preprocessing import StandardScaler

# # scaler = StandardScaler()

# # scaler.fit(Train\_X\_Tfidf)

# # #

# # X\_train = scaler.transform(Train\_X)

# # X\_test = scaler.transform(Test\_X)

#

# from sklearn.neighbors import KNeighborsClassifier

# classifier = KNeighborsClassifier(n\_neighbors=5, weights='distance', algorithm='brute', leaf\_size=30, p=2,

# metric='cosine', metric\_params=None, n\_jobs=1)

# classifier.fit(Train\_X\_Tfidf,Train\_Y)

#

#

# y\_pred = classifier.predict(Test\_X\_Tfidf)

# #

# #

# from sklearn.metrics import classification\_report, confusion\_matrix

# print(confusion\_matrix(Test\_Y,y\_pred))

# print(classification\_report(Test\_Y, y\_pred))

#

# classifier.fit(Train\_X\_Tfidf,Train\_Y)

# predicted = classifier.predict(Test\_X\_Tfidf)

# acc = accuracy\_score(predicted, Test\_Y)

# print('KNN with TFIDF accuracy = ' + str(acc \* 100) + '%')

# #

# # # scores = cross\_val\_score(knn, X\_train, y\_train, cv=3)

# # # print("Cross Validation Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() \* 2))

# # # print(scores)

#

#

# # from sklearn.kernel\_approximation import RBFSampler

# from sklearn.linear\_model import SGDClassifier

# X = [[0, 0], [1, 1], [1, 0], [0, 1]]

# y = [0, 0, 1, 1]

# # rbf\_feature = RBFSampler(gamma=1, random\_state=1)

# # X\_features = rbf\_feature.fit\_transform(X)

# clf = SGDClassifier(max\_iter=5)

# clf.fit(Train\_X\_Tfidf,Train\_Y)

# predicted = clf.predict(Test\_X\_Tfidf)

# SGDClassifier(max\_iter=5)

# acc = accuracy\_score(predicted, Test\_Y)

# print('RB with TFIDF accuracy = ' + str(acc \* 100) + '%')

#

# print(confusion\_matrix(Test\_Y,predicted))

# print(classification\_report(Test\_Y, predicted))

#

# into = 0

# Corpus1 = list(Corpus)

# Corpus2 = pd.read\_csv(r"data\spam.csv",encoding='latin-1')

# print(Corpus2['v2'][5])

# # print(Corpus1)

# i = 0

# # model\_results = {'Message': [],

# # 'Original\_Spam\_Ham': [],

# # 'Predicted': [],

# # 'Spam\_Ham': []

# # }

# # for i in range(len(Test\_X)):

# # if Corpus2['v1'][Test\_X.index[i]] == 'ham':

# # model\_results['Message'].append(Corpus2['v2'][Test\_X.index[i]])

# # model\_results['Original\_Spam\_Ham'].append('0')

# # model\_results['Predicted'].append(predicted[i])

# # model\_results['Spam\_Ham'].append(Corpus2['v1'][Test\_X.index[i]])

# #

# # # print(Corpus2['v2'][Test\_X.index[i]],' 0 ',predicted[i],' ',Corpus2['v1'][Test\_X.index[i]])

# # elif Corpus2['v1'][Test\_X.index[i]] == 'spam':

# # model\_results['Message'].append(Corpus2['v2'][Test\_X.index[i]])

# # model\_results['Original\_Spam\_Ham'].append('1')

# # model\_results['Predicted'].append(predicted[i])

# # model\_results['Spam\_Ham'].append(Corpus2['v1'][Test\_X.index[i]])

# # # print(Corpus2['v2'][Test\_X.index[i]], ' 1 ', predicted[i], ' ', Corpus2['v1'][Test\_X.index[i]])

# # # i = i + 1

# #

# #

# # pd.DataFrame(model\_results).to\_csv('sms\_results.csv', index=False)

import os

import re

import string

import math

DATA\_DIR = 'enron'

target\_names = ['ham', 'spam']

def get\_data(DATA\_DIR):

subfolders = ['enron%d' % i for i in range(1, 7)]

data = []

target = []

for subfolder in subfolders:

# spam

spam\_files = os.listdir(os.path.join(DATA\_DIR, subfolder, 'spam'))

for spam\_file in spam\_files:

with open(os.path.join(DATA\_DIR, subfolder, 'spam', spam\_file), encoding="latin-1") as f:

data.append(f.read())

target.append(1)

# ham

ham\_files = os.listdir(os.path.join(DATA\_DIR, subfolder, 'ham'))

for ham\_file in ham\_files:

with open(os.path.join(DATA\_DIR, subfolder, 'ham', ham\_file), encoding="latin-1") as f:

data.append(f.read())

target.append(0)

return data, target

class SpamDetector(object):

"""Implementation of Naive Bayes for binary classification"""

def clean(self, s):

translator = str.maketrans("", "", string.punctuation)

return s.translate(translator)

def tokenize(self, text):

text = self.clean(text).lower()

return re.split("\W+", text)

def get\_word\_counts(self, words):

word\_counts = {}

for word in words:

word\_counts[word] = word\_counts.get(word, 0.0) + 1.0

return word\_counts

def fit(self, X, Y):

self.num\_messages = {}

self.log\_class\_priors = {}

self.word\_counts = {}

self.vocab = set()

n = len(X)

self.num\_messages['spam'] = sum(1 for label in Y if label == 1)

self.num\_messages['ham'] = sum(1 for label in Y if label == 0)

self.log\_class\_priors['spam'] = math.log(self.num\_messages['spam'] / n)

self.log\_class\_priors['ham'] = math.log(self.num\_messages['ham'] / n)

self.word\_counts['spam'] = {}

self.word\_counts['ham'] = {}

for x, y in zip(X, Y):

c = 'spam' if y == 1 else 'ham'

counts = self.get\_word\_counts(self.tokenize(x))

for word, count in counts.items():

if word not in self.vocab:

self.vocab.add(word)

if word not in self.word\_counts[c]:

self.word\_counts[c][word] = 0.0

self.word\_counts[c][word] += count

def predict(self, X):

result = []

for x in X:

counts = self.get\_word\_counts(self.tokenize(x))

spam\_score = 0

ham\_score = 0

for word, \_ in counts.items():

if word not in self.vocab: continue

# add Laplace smoothing

log\_w\_given\_spam = math.log(

(self.word\_counts['spam'].get(word, 0.0) + 1) / (self.num\_messages['spam'] + len(self.vocab)))

log\_w\_given\_ham = math.log(

(self.word\_counts['ham'].get(word, 0.0) + 1) / (self.num\_messages['ham'] + len(self.vocab)))

spam\_score += log\_w\_given\_spam

ham\_score += log\_w\_given\_ham

spam\_score += self.log\_class\_priors['spam']

ham\_score += self.log\_class\_priors['ham']

if spam\_score > ham\_score:

result.append(1)

else:

result.append(0)

return result

if \_\_name\_\_ == '\_\_main\_\_':

X, y = get\_data(DATA\_DIR)

MNB = SpamDetector()

MNB.fit(X[100:], y[100:])

pred = MNB.predict(X[:100])

true = y[:100]

accuracy = sum(1 for i in range(len(pred)) if pred[i] == true[i]) / float(len(pred))

print("{0:.4f}".format(accuracy))

import csv

import analysis.vectorize

def txt\_to\_csv():

'''Method for creating .csv file from .txt file given the filename'''

file = open('../imdb\_labelled.txt', "r", encoding='utf-8')

li = file.readlines()

file.close()

header = li[0]

li.remove(li[0])

sentiment = []

sentence = []

sentencetrimmed = []

for item in li:

sentiment.append(item[-2])

new = item.replace(item[-2], "")

sentence.append(new)

for item in sentence:

sentencetrimmed.append(item.strip())

for sent in sentencetrimmed:

print(sent)

analysis.vectorize.cleanText(sent, True, False)

with open ('../imdb\_labelled.csv', 'w', newline='', encoding='utf-8') as csvfile:

csvWriter = csv.writer(csvfile, delimiter = '\t', quoting=csv.QUOTE\_MINIMAL)

csvWriter.writerow(['sentence', 'sentiment'])

for sentence, sentm in zip(sentencetrimmed, sentiment):

csvWriter.writerow([sentence, sentm])

from sklearn import metrics, neighbors

from sklearn.model\_selection import train\_test\_split, cross\_val\_score, cross\_val\_predict

import vectorize

import util

def bow\_knn():

"""Method for determining nearest neighbors using bag-of-words model and K-Nearest Neighbor algorithm"""

training\_data = util.get\_parser\_data("knn/amazon.csv")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(training\_data["sentence"], training\_data["sentiment"], test\_size=0.2, random\_state=5)

X\_train, X\_test = vectorize.createBagOfWords(X\_train, X\_test, remove\_stopwords=True, lemmatize=True, stemmer=False)

knn = neighbors.KNeighborsClassifier(n\_neighbors=5, weights='uniform', algorithm='auto', leaf\_size=30, p=2, metric='jaccard', metric\_params=None, n\_jobs=1)

knn.fit(X\_train, y\_train)

predicted = knn.predict(X\_test)

acc = metrics.accuracy\_score(y\_test, predicted)

print('KNN with BOW accuracy = ' + str(acc \* 100) + '%')

scores = cross\_val\_score(knn, X\_train, y\_train, cv=3)

print("Cross Validation Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() \* 2))

print(scores)

print('\n')

def tfidf\_knn():

"""Method for determining nearest neighbors using bag-of-words model and K-Nearest Neighbor algorithm"""

training\_data = util.get\_parser\_data("amazon.csv")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(training\_data["sentence"], training\_data["sentiment"],

test\_size=0.2, random\_state=5)

X\_train, X\_test = vectorize.createTFIDF(X\_train, X\_test, remove\_stopwords=True, lemmatize=True, stemmer=False)

knn = neighbors.KNeighborsClassifier(n\_neighbors=5, weights='distance', algorithm='brute', leaf\_size=30, p=2,

metric='cosine', metric\_params=None, n\_jobs=1)

knn.fit(X\_train, y\_train)

predicted = knn.predict(X\_test)

acc = metrics.accuracy\_score(y\_test, predicted)

print('KNN with TFIDF accuracy = ' + str(acc \* 100) + '%')

scores = cross\_val\_score(knn, X\_train, y\_train, cv=3)

print("Cross Validation Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() \* 2))

print(scores)