import pandas as pd

import numpy as np

import re

from sklearn import tree

from sklearn.tree import DecisionTreeClassifier

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

from sklearn import metrics

import math

from sklearn.metrics import confusion\_matrix

import graphviz

from matplotlib.ticker import MaxNLocator

import nltk

nltk.download('stopwords')

import matplotlib.pyplot as plt

from wordcloud import WordCloud , STOPWORDS , ImageColorGenerator

from sklearn.metrics import accuracy\_score, log\_loss

from sklearn.naive\_bayes import GaussianNB

from plotly import express as px

df=pd.read\_csv("/content/drive/MyDrive/chat.csv",encoding='cp1252')

print(df)

df1=pd.read\_csv("/content/drive/MyDrive/chat.csv",encoding='cp1252')

print("Persons in WhatsApp Group are:")

S=df.PERSON.unique()

list(S)

print(S)

print("\n")

print("Number ofPersons in WhatsApp Group are:")

s1=df['PERSON'].nunique()

print(s1)

df1['Letters'] = df['messages'].apply(lambda s : len(s))

df1['Words'] = df['messages'].apply(lambda s : len(s.split(' ')))

MEDIAPATTERN = "Media omitted"

df1['Media\_Count'] = df1.messages.apply(lambda x : re.findall(MEDIAPATTERN, x)).str.len()

media\_messages = np.sum(df1.Media\_Count)

URLPATTERN = r"(?i)\b((?:https?://|www\d{0,3}[.]|[a-z0-9.\-]+[.][a-z]{2,4}/)(?:[^\s()<>]+|\(([^\s()<>]+|(\([^\s()<>]+\)))\*\))+(?:\(([^\s()<>]+|(\([^\s()<>]+\)))\*\)|[^\s`!()\[\]{};:'\".,<>?«»“”‘’]))"

df1['Url\_Count'] = df1.messages.apply(lambda x: re.findall(URLPATTERN, x)).str.len()

links = np.sum(df1.Url\_Count)

total\_messages = df1.shape[0]

links = np.sum(df1.Url\_Count)

print('Group Chat Status \n')

print('Total Number of Messages : {}'.format(total\_messages))

print('Total Number of Media Messages : {}'.format(media\_messages))

print('Total Number of Links : {}'.format(links))

l = df1.PERSON.unique()

for i in range(len(l)):

  req\_df = df1[df1["PERSON"] == l[i]]

  print(f'MESSAGES BY {l[i]} \n ')

  print('Total Message Sent : ', req\_df.shape[0])

  wordspermessage = (np.sum(req\_df['Words']))/req\_df.shape[0]

  media = sum(req\_df["Media\_Count"])

  print('Total Media Message Sent : ', media)

  links = sum(req\_df["Url\_Count"])

  print('Total Links Sent : ', links)

  print("\n")

listML=df['ML related'].tolist()

listClass=df['classlabel'].tolist()

lendf=len(df)

yesClass=listClass.count('yes')

noClass=listClass.count('no')

y=yesClass/lendf

n=noClass/lendf

ey=-y\*math.log2(y)

en=-n\*math.log2(n)

entro=en+ey

hy,ly=0,0

hn,ln=0,0

for i in range(0,130):

  if listClass[i]=='yes' and listML[i]=='yes':

    hy=hy+1

  if listClass[i]=='yes' and listML[i]=='no':

    ly=ly+1

  if listClass[i]=='no' and listML[i]=='yes':

    hn=hn+1

  if listClass[i]=='no' and listML[i]=='no':

    ln=ln+1

print(hy,ly)

print(hn,ln)

high=hy+hn

low=ly+ln

print(high,low)

c\_h\_en=-(hy/high)\*math.log2(hy/high) -(hn/high)\*math.log2(hn/high)

print("entropy yes: %2f" % c\_h\_en)

if ly!=0:

  c\_l\_en=-(ly/low)\*math.log2(ly/low) -(ln/low)\*math.log2(ln/low)

  print("entropy no: .%2f" % c\_l\_en)

en\_h1=hy/lendf\*c\_h\_en

entropy=en\_h1

info=entro-entropy    #information gain for ML related

LL=df['Letters'].tolist()

#letter count

hy1,ly1=0,0

hn1,ln1=0,0

for i in range(0,130):

  if listClass[i]=='yes' and LL[i]>=10:

    hy1=hy1+1

  if listClass[i]=='yes' and LL[i]<=10:

    ly1=ly1+1

  if listClass[i]=='no' and LL[i]>=10:

    hn1=hn1+1

  if listClass[i]=='no' and LL[i]<=10:

    ln1=ln1+1

print(hy1,ly1)

print(hn1,ln1)

high1=hy1+hn1

low1=ly1+ln1

print(high1,low1)

c\_h\_en1=-(hy1/high1)\*math.log2(hy1/high1) -(hn1/high1)\*math.log2(hn1/high1)

print("entropy yes: .%2f" % c\_h\_en1)

if ly1!=0:

  c\_l\_en1=-(ly1/low1)\*math.log2(ly1/low1) -(ln1/low1)\*math.log2(ln1/low1)

  print("entropy no: .%2f" % c\_l\_en1)

en\_h=hy1/lendf\*c\_h\_en1

entropy1=en\_h

info1=entro-entropy1  #info for letter count

WL=df['Words'].tolist()

hy2,ly2=0,0

hn2,ln2=0,0

for i in range(0,130):

  if listClass[i]=='yes' and WL[i]>=1:

    hy2=hy2+1

  if listClass[i]=='yes' and WL[i]<=1:

    ly2=ly2+1

  if listClass[i]=='no' and WL[i]>=1:

    hn2=hn2+1

  if listClass[i]=='no' and WL[i]<=1:

    ln2=ln2+1

print(hy2,ly2)

print(hn2,ln2)

high2=hy2+hn2

low2=ly2+ln2

print(high2,low2)

c\_h\_en2=-(hy2/high2)\*math.log2(hy2/high2) -(hn2/high2)\*math.log2(hn2/high2)

print("entropy high: %2f" % c\_h\_en2)

if ly2!=0:

  c\_l\_en2=-(ly2/low2)\*math.log2(ly2/low2) -(ln2/low2)\*math.log2(ln2/low2)

print("entropy low: %2f" % c\_l\_en2)

en\_h=hy2/lendf\*c\_h\_en2

en\_l=hn2/lendf\*c\_l\_en2

entropy1=en\_h+en\_l

info2=entro-entropy1        #info gain for word count

print("ENTROPY FOR ML RELATED CONTENT")

print(info)

print("\n")

print("ENTROPY FOR WORD COUNT")

print(info1)

print("\n")

print("ENTROPY FOR LETTER COUNT")

print(info2)

print("\n")

feature\_cols = ['URL', 'media', 'ML related','Letters','Words','Media\_count','Url\_count']

X = df[feature\_cols]

y = df.classlabel

X=X.replace({"ML related":  {"yes":1,"no":2}})

X=X.replace({"media":  {"yes":1,"no":2}})

X=X.replace({"URL":  {"yes":1,"no":2}})

X=X.replace({"Words ":  {"yes":1,"no":2}})

X=X.replace({"Media\_count":  {"yes":1,"no":2}})

X=X.replace({"URL\_count":  {"yes":1,"no":2}})

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=1)

clf = DecisionTreeClassifier()

clf = clf.fit(X\_train,y\_train)

y\_pred = clf.predict(X\_test)

feature\_cols = ['URL', 'media', 'ML related','Letters','Words','Media\_count','Url\_count']

X = df[feature\_cols]

Y = df.classlabel

dot\_data = tree.export\_graphviz(clf, out\_file=None,

                feature\_names=feature\_cols,

                class\_names=df.classlabel,

                filled=True, rounded=True,

                special\_characters=True)

graph = graphviz.Source(dot\_data)

graph

cm=confusion\_matrix(y\_test,y\_pred)

from mlxtend.plotting import plot\_confusion\_matrix

fig, ax = plot\_confusion\_matrix(conf\_mat=cm, figsize=(6, 6), cmap=plt.cm.Greens)

plt.xlabel('Predictions', fontsize=18)

plt.ylabel('Actuals', fontsize=18)

plt.title('Confusion Matrix', fontsize=18)

plt.show()

tp=cm[0][0]

tn=cm[1][1]

fp=cm[0][1]

fn=cm[1][0]

precision=tp/(tp+fp)

print("Precision =",precision\*100)

recall=tp/(tp+fn)

print("Recall =",recall\*100)

fm=2\*precision\*recall/(precision+recall)

print("F measure=",fm\*100)

gnb = GaussianNB()

gnb.fit(X\_train, y\_train)

y\_pred1 = gnb.predict(X\_test)

cm1=confusion\_matrix(y\_test,y\_pred1)

from mlxtend.plotting import plot\_confusion\_matrix

fig, ax = plot\_confusion\_matrix(conf\_mat=cm1, figsize=(6, 6), cmap=plt.cm.Greens)

plt.xlabel('Predictions', fontsize=18)

plt.ylabel('Actuals', fontsize=18)

plt.title('Confusion Matrix', fontsize=18)

plt.show()

tp1=cm1[0][0]

tn1=cm1[1][1]

fp1=cm1[0][1]

fn1=cm1[1][0]

precision1=tp1/(tp1+fp1)

print("Precision =",precision1\*100)

recall1=tp1/(tp1+fn1)

print("Recall =",recall1\*100)

fm1=2\*precision1\*recall1/(precision1+recall1)

print("F measure=",fm1\*100)

classifiers = [

    DecisionTreeClassifier(),

    GaussianNB()]

log\_cols=["Classifier", "Accuracy", "Log Loss"]

log = pd.DataFrame(columns=log\_cols)

for clf in classifiers:

    clf.fit(X\_train, y\_train)

    name = clf.\_\_class\_\_.\_\_name\_\_

    print(name)

    print('Results')

    train\_predictions = clf.predict(X\_test)

    acc = accuracy\_score(y\_test, train\_predictions)

    print("Accuracy: {:.4%}".format(acc))

    train\_predictions = clf.predict\_proba(X\_test)

    ll = log\_loss(y\_test, train\_predictions)

    print("Log Loss: {}".format(ll))

    log\_entry = pd.DataFrame([[name, acc\*100, ll]], columns=log\_cols)

    log = log.append(log\_entry)

data = {'DECISION TREE': 98.48,'NAIVE BAYESIAN':95.45}

Algorithm = list(data.keys())

values = list(data.values())

fig = plt.figure(figsize = (5, 5))

plt.bar(Algorithm, values, color ='green',

        width = 0.6)

plt.ylim(70,100)

plt.xlabel("ACCURACY")

plt.ylabel("PERCENTAGE")

plt.title("PERFORMANCE OF ALGORITHMS")

plt.show()

X = ['DECISION TREE','NAIVE BAYES']

PRECISION= [97.22,91.66]

RECALL = [100,100]

X\_axis = np.arange(len(X))

plt.bar(X\_axis - 0.2, PRECISION, 0.4, label = 'PRECISION')

plt.bar(X\_axis + 0.2,RECALL , 0.4, label = 'RECALL')

plt.xticks(X\_axis, X)

plt.xlabel("ALGORITHMS")

plt.ylabel("PERCENTAGE")

plt.title("PRECISION,RECALL FOR DIFFERENT ALGORITHMS")

plt.ylim(70,100)

plt.legend()

plt.show()

text = " ".join(review for review in df.messages)

wordcloud = WordCloud(stopwords=STOPWORDS, background\_color="white").generate(text)

plt.figure( figsize=(10,5))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis("off")

plt.show()

plt.figure(figsize=(9,6))

mostly\_active = df['PERSON'].value\_counts()

### Top 10 peoples that are mostly active in our Group is :

m\_a = mostly\_active.head(10)

bars = ['Nanthitha','Karthika','Shobika','Deepthi','karthika','Aarthi','Srivarshini','Subashini','Agalya','asvika''navaneetha']

x\_pos = np.arange(len(bars))

m\_a.plot.bar()

plt.xlabel('Authors',fontdict={'fontsize': 14,'fontweight': 10})

plt.ylabel('No. of messages',fontdict={'fontsize': 14,'fontweight': 10})

plt.title('Mostly active member of Group',fontdict={'fontsize': 20,'fontweight': 8})

plt.xticks(x\_pos, bars)

plt.show()

plt.figure(figsize=(8,5))

t = df['TIME'].value\_counts().head(20)

tx = t.plot.bar()

tx.yaxis.set\_major\_locator(MaxNLocator(integer=True))  #Converting y axis data to integer

plt.xlabel('TIME',fontdict={'fontsize': 12,'fontweight': 10})

plt.ylabel('No. of messages',fontdict={'fontsize': 12,'fontweight': 10})

plt.title('Analysis of time when Group was highly active.',fontdict={'fontsize': 18,'fontweight': 8})

plt.show()

plt.figure(figsize=(8,5))

df['DATE'].value\_counts().head(15).plot.bar()

plt.xlabel('DATE',fontdict={'fontsize': 14,'fontweight': 10})

plt.ylabel('No. of messages',fontdict={'fontsize': 14,'fontweight': 10})

plt.title('Analysis of Date on which Group was highly active',fontdict={'fontsize': 18,'fontweight': 8})

plt.show()

from numpy import array

import numpy as np

import pandas as pd

df = pd.read\_csv('/content/drive/MyDrive/chat.csv',encoding='cp1252')

df = df.dropna()

df = df.reset\_index(drop=True)

df1 = df[['PERSON', 'DATE']]

df1

import networkx as nx

#G = nx.Graph()

from matplotlib.pyplot import figure

G = nx.from\_pandas\_edgelist(df1, 'PERSON', 'DATE')

figure(figsize=(10, 8))

nx.draw\_shell(G, with\_labels=True)

leaderboard = {}

for x in G.nodes:

 leaderboard[x] = len(G[x])

s = pd.Series(leaderboard, name='connections')

df2 = s.to\_frame().sort\_values('connections', ascending=False)

print("TOP 4 ACTIVE PERSONS")

print("\n")

print(df2[:4])

betweeness=nx.betweenness\_centrality(G)

print(betweeness)

import operator

b1 = dict( sorted(betweeness.items(), key=operator.itemgetter(1),reverse=True))

b1

b2 = list(b1.items())

print(b2)

print("\033[1mPerson with highest betweeness centrality is : \033[0m",b2[0])

closeness=nx.closeness\_centrality(G)

print(closeness)

import operator

c1 = dict( sorted(closeness.items(), key=operator.itemgetter(1),reverse=True))

c2 = list(c1.items())

print(c2)

print("\033[1mPerson with highest closeness centrality is : \033[0m",c2[0])

print("\033[1mPerson with highest Betweeness  centrality is : \033[0m",b2[0])