**FINDING INCONSISTENCY OF SECURITY INFORMATION FROM UNSTRUCTURED TEXT**

|  |  |  |
| --- | --- | --- |
| NAME OF THE STUDENTS | : | SANGGAMI R  PUNITHA G  SARANYA R |
| REGISTER NUMBER(S) | : | 211417104238  211417104205  211417104244 |
| NAME OF THE GUIDE | : | Mrs. SANGEETHA K |

**CYBER BULLYING**

#!/usr/bin/env python

# coding: utf-8

# In[104]:

import numpy as np

import pandas as pd

# In[105]:

import os

print(os.listdir("dataset"))

# In[106]:

df = pd.read\_json('dataset/dataset.json', lines= True)

# In[107]:

df.head(25)

# In[108]:

df.tail(25)

# In[109]:

df.shape

# In[110]:

df["label"] = df.annotation.apply(lambda x: x.get('label'))

df["label"] = df.label.apply(lambda x: x[0])

df.head()

# In[111]:

df.extras.unique()

# In[112]:

df["notes"] = df.annotation.apply(lambda x: x.get('notes'))

df.notes.unique()

# In[113]:

df.groupby('label').describe()

# In[114]:

import nltk

nltk.download(['punkt', 'wordnet'])

#Import

import re

import numpy as np

import pandas as pd

from nltk.tokenize import word\_tokenize

from nltk.stem import WordNetLemmatizer

from sklearn.metrics import confusion\_matrix

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.naive\_bayes import MultinomialNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.feature\_extraction.text import CountVectorizer, TfidfTransformer

# In[115]:

def load\_data(path):

df = pd.read\_json(path, lines= True)

df["label"] = df.annotation.apply(lambda x: x.get('label'))

df["label"] = df.label.apply(lambda x: x[0])

X = df.content.values

y = df.label.values

return X, y

# In[116]:

def tokenize(text):

tokens = word\_tokenize(text)

lemmatizer = WordNetLemmatizer()

clean\_tokens = []

for tok in tokens:

clean\_tok = lemmatizer.lemmatize(tok).lower().strip()

clean\_tokens.append(clean\_tok)

return clean\_tokens

# In[117]:

def display\_results(y\_test, y\_pred):

labels = np.unique(y\_pred)

confusion\_mat = confusion\_matrix(y\_test, y\_pred, labels=labels)

accuracy = (y\_pred == y\_test).mean()

print("Labels:", labels)

print("Confusion Matrix:\n", confusion\_mat)

print("Accuracy:", accuracy)

# In[118]:

def main():

url = 'dataset/dataset.json'

X, y = load\_data(url)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y)

print ("X\_train: ", len(X\_train))

print("X\_test: ", len(X\_test))

print("y\_train: ", len(y\_train))

print("y\_test: ", len(y\_test))

vect = CountVectorizer(tokenizer=tokenize)

tfidf = TfidfTransformer()

#classfier model

clf = RandomForestClassifier()

# train classifier

X\_train\_counts = vect.fit\_transform(X\_train)

X\_train\_tfidf = tfidf.fit\_transform(X\_train\_counts)

clf.fit(X\_train\_tfidf, y\_train)

# predict on test data

X\_test\_counts = vect.transform(X\_test)

X\_test\_tfidf = tfidf.transform(X\_test\_counts)

y\_pred = clf.predict(X\_test\_tfidf)

# predict on test data

X\_test\_counts = vect.transform(["whoa stop you stupid sjw"])

X\_test\_tfidf = tfidf.transform(X\_test\_counts)

print("\nGiven text: 'whoa stop you stupid sjw' ")

print("Prediction: {}\n".format(clf.predict(X\_test\_tfidf)))

print("Random Forest")

# display results

display\_results(y\_test, y\_pred)

mnb = MultinomialNB()

# train classifier for mnb

X\_train\_counts = vect.fit\_transform(X\_train)

X\_train\_tfidf = tfidf.fit\_transform(X\_train\_counts)

mnb.fit(X\_train\_tfidf, y\_train)

# predict on test data

X\_test\_counts = vect.transform(X\_test)

X\_test\_tfidf = tfidf.transform(X\_test\_counts)

y\_pred = mnb.predict(X\_test\_tfidf)

# predict on test data

X\_test\_counts = vect.transform(["whoa stop you stupid sjw"])

X\_test\_tfidf = tfidf.transform(X\_test\_counts)

print("\nGiven text: 'whoa stop you stupid sjw' ")

print("Prediction: {}\n".format(mnb.predict(X\_test\_tfidf)))

print("Naive Bayes")

# display results

display\_results(y\_test, y\_pred)

dtc = DecisionTreeClassifier()

# train classifier

X\_train\_counts = vect.fit\_transform(X\_train)

X\_train\_tfidf = tfidf.fit\_transform(X\_train\_counts)

dtc.fit(X\_train\_tfidf, y\_train)

# predict on test data

X\_test\_counts = vect.transform(X\_test)

X\_test\_tfidf = tfidf.transform(X\_test\_counts)

y\_pred = dtc.predict(X\_test\_tfidf)

# predict on test data

X\_test\_counts = vect.transform(["whoa stop you stupid sjw"])

X\_test\_tfidf = tfidf.transform(X\_test\_counts)

print("\nGiven text: 'whoa stop you stupid sjw' ")

print("Prediction: {}\n".format(dtc.predict(X\_test\_tfidf)))

print("Decision Tree")

# display results

display\_results(y\_test, y\_pred)

svm = SVC()

# train classifier

X\_train\_counts = vect.fit\_transform(X\_train)

X\_train\_tfidf = tfidf.fit\_transform(X\_train\_counts)

svm.fit(X\_train\_tfidf, y\_train)

# predict on test data

X\_test\_counts = vect.transform(X\_test)

X\_test\_tfidf = tfidf.transform(X\_test\_counts)

y\_pred = svm.predict(X\_test\_tfidf)

# predict on test data

X\_test\_counts = vect.transform(["whoa stop you stupid sjw"])

X\_test\_tfidf = tfidf.transform(X\_test\_counts)

print("\nGiven text: 'whoa stop you stupid sjw' ")

print("Prediction: {}\n".format(svm.predict(X\_test\_tfidf)))

print("SVM")

# display results

display\_results(y\_test, y\_pred)

main()

**SPAMMING**

#!/usr/bin/env python

# coding: utf-8

# In[1]:

import numpy as np

import pandas as pd

import os

# In[2]:

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

from collections import Counter

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

import re

import nltk

from nltk.corpus import stopwords

from nltk.stem.porter import PorterStemmer

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.svm import SVC

from sklearn.naive\_bayes import MultinomialNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

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

# In[3]:

##Datasets

# In[4]:

data = pd.read\_csv("spam.csv", encoding = 'latin1')

data.head()

# In[5]:

# drop unavailable attributes

data = data.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], axis = 1)

# In[6]:

# rename columns

data = data.rename(columns ={"v1":"target", "v2":"text"})

data.head()

# In[7]:

#for counting ham and spam

data.target.value\_counts()

# In[8]:

data.groupby('target').describe()

# In[9]:

#plotting graph for distribution

sns.countplot(x = "target", data = data)

data.loc[:, 'target'].value\_counts()

plt.title('Distribution of Spam and Ham')

# In[10]:

# plotting graph by length.

ham =data[data['target'] == 'ham']['text'].str.len()

sns.distplot(ham, label='Ham')

spam = data[data['target'] == 'spam']['text'].str.len()

sns.distplot(spam, label='Spam')

plt.title('Distribution by Length')

plt.legend()

# In[11]:

#plotting graph by digits.

ham1 = data[data['target'] == 'ham']['text'].str.replace(r'\D+', '').str.len()

sns.distplot(ham1, label='Ham')

spam1 = data[data['target'] == 'spam']['text'].str.replace(r'\D+', '').str.len()

sns.distplot(spam1, label='Spam')

plt.title('Distribution by Digits')

plt.legend()

# In[12]:

#plotting graph for non-digits.

ham2 = data[data['target'] == 'ham']['text'].str.replace(r'\w+', '').str.len()

sns.distplot(ham2, label='Ham')

spam2 = data[data['target'] == 'spam']['text'].str.replace(r'\w+', '').str.len()

sns.distplot(spam2, label='Spam')

plt.title('Distribution of Non-Digits')

plt.legend()

# In[13]:

#for counting frequently occurence of spam and ham.

count1 = Counter(" ".join(data[data['target']=='ham']["text"]).split()).most\_common(30)

data1 = pd.DataFrame.from\_dict(count1)

data1 = data1.rename(columns={0: "words of ham", 1 : "count"})

count2 = Counter(" ".join(data[data['target']=='spam']["text"]).split()).most\_common(30)

data2 = pd.DataFrame.from\_dict(count2)

data2 = data2.rename(columns={0: "words of spam", 1 : "count\_"})

# In[14]:

data1.plot.bar(legend = False, color = 'purple',figsize = (20,15))

y\_pos = np.arange(len(data1["words of ham"]))

plt.xticks(y\_pos, data1["words of ham"])

plt.title('Top 30 words of ham')

plt.xlabel('words')

plt.ylabel('number')

plt.show()

# In[15]:

data2.plot.bar(legend = False, color = 'green', figsize = (20,17))

y\_pos = np.arange(len(data2["words of spam"]))

plt.xticks(y\_pos, data2["words of spam"])

plt.title('Top 30 words of spam')

plt.xlabel('words')

plt.ylabel('number')

plt.show()

# In[16]:

# Function to plot the confusion matrix (code from https://scikit-learn.org/stable/auto\_examples/model\_selection/plot\_confusion\_matrix.html)

from sklearn import metrics

import itertools

def plot\_confusion\_matrix(cm, classes,

normalize=False,

title='Confusion matrix',

cmap=plt.cm.Blues):

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

print("Normalized confusion matrix")

else:

print('Confusion matrix, without normalization')

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, cm[i, j],

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.ylabel('Actual label')

plt.xlabel('Predicted label')

# In[17]:

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data['text'], data['target'], test\_size = 0.3, random\_state = 37)

print ("X\_train: ", len(X\_train))

print("X\_test: ", len(X\_test))

print("y\_train: ", len(y\_train))

print("y\_test: ", len(y\_test))

# In[18]:

corpus = []

for i in range(0, 5572):

review = re.sub('[^a-zA-Z]', ' ', data['text'][i])

review = review.lower()

review = review.split()

ps = PorterStemmer()

review = [ps.stem(word) for word in review if not word in set(stopwords.words('english'))]

review = ' '.join(review)

corpus.append(review)

# In[19]:

cv = CountVectorizer(max\_features = 1500)

cv.fit(X\_train)

# In[20]:

X\_train\_cv = cv.transform(X\_train)

X\_train\_cv

# In[21]:

X\_test\_cv = cv.transform(X\_test)

X\_test\_cv

# In[22]:

mnb = MultinomialNB(alpha = 0.5)

mnb.fit(X\_train\_cv,y\_train)

y\_mnb = mnb.predict(X\_test\_cv)

# In[23]:

print('Naive Bayes Accuracy: ', accuracy\_score( y\_mnb , y\_test))

print('Naive Bayes confusion\_matrix: ', confusion\_matrix(y\_mnb, y\_test))

cm = metrics.confusion\_matrix(y\_test, y\_mnb)

plot\_confusion\_matrix(cm, classes=['spam', 'ham'])

# In[24]:

svc = SVC(kernel='sigmoid', gamma=1.0)

svc.fit(X\_train\_cv,y\_train)

y\_svc = svc.predict(X\_test\_cv)

# In[25]:

print('SVM Accuracy: ', accuracy\_score( y\_svc , y\_test))

print('SVM confusion\_matrix: ', confusion\_matrix(y\_svc, y\_test))

cm = metrics.confusion\_matrix(y\_test, y\_svc)

plot\_confusion\_matrix(cm, classes=['spam', 'ham'])

# In[26]:

knc = KNeighborsClassifier(n\_neighbors=100)

knc.fit(X\_train\_cv,y\_train)

y\_knc = knc.predict(X\_test\_cv)

# In[27]:

print('KNeighborsAccuracy\_score: ',accuracy\_score(y\_test,y\_knc))

print('KNeighborsconfusion\_matrix: ', confusion\_matrix(y\_test, y\_knc))

cm = metrics.confusion\_matrix(y\_test, y\_knc)

plot\_confusion\_matrix(cm, classes=['spam', 'ham'])

# In[28]:

dtc = DecisionTreeClassifier(min\_samples\_split=7, random\_state=252)

dtc.fit(X\_train\_cv,y\_train)

y\_dtc = dtc.predict(X\_test\_cv)

# In[29]:

print('Decision Tree Accuracy: ',accuracy\_score(y\_test,y\_dtc))

print('Decision Tree confusion\_matrix: ', confusion\_matrix(y\_dtc, y\_test))

cm = metrics.confusion\_matrix(y\_test, y\_dtc)

plot\_confusion\_matrix(cm, classes=['spam', 'ham'])

# In[30]:

rfc = RandomForestClassifier(n\_estimators=37, random\_state=252)

rfc.fit(X\_train\_cv,y\_train)

y\_rfc = rfc.predict(X\_test\_cv)

print('Random Forest Accuracy\_score: ',accuracy\_score(y\_test,y\_rfc))

print('Random Forest confusion\_matrix: ', confusion\_matrix(y\_rfc, y\_test))

cm = metrics.confusion\_matrix(y\_test, y\_rfc)

plot\_confusion\_matrix(cm, classes=['spam', 'ham']

**FAKE NEWS**

#!/usr/bin/env python

# coding: utf-8

# # Fake news detection

# In[1]:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfTransformer

from sklearn import feature\_extraction, linear\_model, model\_selection, preprocessing

from sklearn.metrics import accuracy\_score

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

from sklearn.pipeline import Pipeline

# ## Read datasets

# In[2]:

fake = pd.read\_csv("data/Fake.csv")

true = pd.read\_csv("data/True.csv")

# In[3]:

fake.shape

# In[4]:

true.shape

# ## Data cleaning and preparation

# In[5]:

# Add flag to track fake and real

fake['target'] = 'fake'

true['target'] = 'true'

# In[6]:

# Concatenate dataframes

data = pd.concat([fake, true]).reset\_index(drop = True)

data.shape

# In[7]:

# Shuffle the data

from sklearn.utils import shuffle

data = shuffle(data)

data = data.reset\_index(drop=True)

# In[8]:

# Check the data

data.head()

# In[9]:

# Removing the date (we won't use it for the analysis)

data.drop(["date"],axis=1,inplace=True)

data.head()

# In[10]:

# Removing the title (we will only use the text)

data.drop(["title"],axis=1,inplace=True)

data.head()

# In[11]:

# Convert to lowercase

data['text'] = data['text'].apply(lambda x: x.lower())

data.head()

# In[12]:

# Remove punctuation

import string

def punctuation\_removal(text):

all\_list = [char for char in text if char not in string.punctuation]

clean\_str = ''.join(all\_list)

return clean\_str

data['text'] = data['text'].apply(punctuation\_removal)

# In[13]:

# Check

data.head()

# In[14]:

# Removing stopwords

import nltk

nltk.download('stopwords')

from nltk.corpus import stopwords

stop = stopwords.words('english')

data['text'] = data['text'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))

# In[15]:

data.head()

# ## Basic data exploration

# In[16]:

# How many articles per subject?

print(data.groupby(['subject'])['text'].count())

data.groupby(['subject'])['text'].count().plot(kind="bar")

plt.show()

# In[17]:

# How many fake and real articles?

print(data.groupby(['target'])['text'].count())

data.groupby(['target'])['text'].count().plot(kind="bar")

plt.show()

# In[18]:

# Word cloud for fake news

from wordcloud import WordCloud

fake\_data = data[data["target"] == "fake"]

all\_words = ' '.join([text for text in fake\_data.text])

wordcloud = WordCloud(width= 800, height= 500,

max\_font\_size = 110,

collocations = False).generate(all\_words)

plt.figure(figsize=(10,7))

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

plt.axis("off")

plt.show()

# In[19]:

# Word cloud for real news

from wordcloud import WordCloud

real\_data = data[data["target"] == "true"]

all\_words = ' '.join([text for text in fake\_data.text])

wordcloud = WordCloud(width= 800, height= 500,

max\_font\_size = 110,

collocations = False).generate(all\_words)

plt.figure(figsize=(10,7))

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

plt.axis("off")

plt.show()

# In[20]:

from nltk import tokenize

token\_space = tokenize.WhitespaceTokenizer()

def counter(text, column\_text, quantity):

all\_words = ' '.join([text for text in text[column\_text]])

token\_phrase = token\_space.tokenize(all\_words)

frequency = nltk.FreqDist(token\_phrase)

df\_frequency = pd.DataFrame({"Word": list(frequency.keys()),

"Frequency": list(frequency.values())})

df\_frequency = df\_frequency.nlargest(columns = "Frequency", n = quantity)

plt.figure(figsize=(12,8))

ax = sns.barplot(data = df\_frequency, x = "Word", y = "Frequency", color = 'blue')

ax.set(ylabel = "Count")

plt.xticks(rotation='vertical')

plt.show()

# In[21]:

# Most frequent words in fake news

counter(data[data["target"] == "fake"], "text", 20)

# In[22]:

# Most frequent words in real news

counter(data[data["target"] == "true"], "text", 20)

# ## Modeling

# In[23]:

# Function to plot the confusion matrix (code from https://scikit-learn.org/stable/auto\_examples/model\_selection/plot\_confusion\_matrix.html)

from sklearn import metrics

import itertools

def plot\_confusion\_matrix(cm, classes,

normalize=False,

title='Confusion matrix',

cmap=plt.cm.Blues):

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45)

plt.yticks(tick\_marks, classes)

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

print("Normalized confusion matrix")

else:

print('Confusion matrix, without normalization')

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, cm[i, j],

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.ylabel('Actual label')

plt.xlabel('Predicted label')

# ### Peparing the data

# In[24]:

# Split the data

X\_train,X\_test,y\_train,y\_test = train\_test\_split(data['text'], data.target, test\_size=0.2, random\_state=42)

print ("X\_train: ", len(X\_train))

print("X\_test: ", len(X\_test))

print("y\_train: ", len(y\_train))

print("y\_test: ", len(y\_test))

# ### Logistic regression

# In[25]:

# Vectorizing and applying TF-IDF

from sklearn.linear\_model import LogisticRegression

pipe = Pipeline([('vect', CountVectorizer()),

('tfidf', TfidfTransformer()),

('model', LogisticRegression())])

# Fitting the model

model = pipe.fit(X\_train, y\_train)

# Accuracy

prediction = model.predict(X\_test)

print("accuracy: {}%".format(round(accuracy\_score(y\_test, prediction)\*100,2)))

# In[26]:

cm = metrics.confusion\_matrix(y\_test, prediction)

plot\_confusion\_matrix(cm, classes=['Real', 'Fake'])

# ### Decision Tree Classifier

# In[27]:

from sklearn.tree import DecisionTreeClassifier

# Vectorizing and applying TF-IDF

pipe = Pipeline([('vect', CountVectorizer()),

('tfidf', TfidfTransformer()),

('model', DecisionTreeClassifier(criterion= 'entropy',

max\_depth = 20,

splitter='best',

random\_state=42))])

# Fitting the model

model = pipe.fit(X\_train, y\_train)

# Accuracy

prediction = model.predict(X\_test)

print("accuracy: {}%".format(round(accuracy\_score(y\_test, prediction)\*100,2)))

# In[28]:

cm = metrics.confusion\_matrix(y\_test, prediction)

plot\_confusion\_matrix(cm, classes=['Fake', 'Real'])

# ### Random Forest Classifier

# In[29]:

from sklearn.ensemble import RandomForestClassifier

pipe = Pipeline([('vect', CountVectorizer()),

('tfidf', TfidfTransformer()),

('model', RandomForestClassifier(n\_estimators=50, criterion="entropy"))])

model = pipe.fit(X\_train, y\_train)

prediction = model.predict(X\_test)

print("accuracy: {}%".format(round(accuracy\_score(y\_test, prediction)\*100,2)))

# In[30]:

cm = metrics.confusion\_matrix(y\_test, prediction)

plot\_confusion\_matrix(cm, classes=['Fake', 'Real'])