21st Nov

```
import json
with open("dict.json", "w") as write_file:
    json.dump(dic, write_file, indent=4)
```

```
from google.colab import files
files.download('dict.json')
```

import dataset

```
import pandas as pd
```

```
import nltk
import re
import string
string.punctuation
from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords

def tokenization(text):
    tokens = text.split(' ')
    return tokens

def remove_punctuation(text):
    punctuationfree=[]
    for t in text:
        punctuationfree.append(''.join(i for i in t if i not in string.punctuation))
    return punctuationfree


nltk.download('stopwords')
stopwords=stopwords.words('english')

def remove_stopwords(text):
    output= [i for i in text if i not in stopwords]
    return output


porter_stemmer = PorterStemmer()

def stemming(text):
    stem_text = [porter_stemmer.stem(word) for word in text]
    return stem_text
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
"""import json

with open('dict.json') as json_file:
    data = json.load(json_file)"""
```

```
'import json\n\nwith open('dict.json') as json_file:\n    data = json.load(json_f
ile)'
```

```
import warnings
warnings.filterwarnings('ignore')
```

```
import pandas as pd

df = pd.read_excel(r'Data.xlsx')
```

```
df['Processed Text']=df['Summary Text'].str.replace('\d+', '')
df['Processed Text']= df['Processed Text'].apply(lambda x: x.lower())
```

```
df['Processed Text']=df['Processed Text'].apply(lambda x: tokenization(x))
df['Processed Text']=df['Processed Text'].apply(lambda x: remove_punctuation(x))
df['Processed Text']=df['Processed Text'].apply(lambda x: remove_stopwords(x))

df['Processed Text']=df['Processed Text'].apply(lambda x: ' '.join(x))
```

```
df.head(5)
```

	CVE ID Number	Vulnerability Type	Summary Text	Processed Text
0	CVE-2022-43766	Denial of Service	Apache IoTDB version 0.12.2 to 0.12.6, 0.13.0 ...	apache iotdb version vulnerable denial ser...
1	CVE-2022-43365	Denial of Service	IP-COM EW9 V15.11.0.14(9732) was discovered to...	ipcom ew v discovered contain buffer overflow ...
	CVE-			

```
df.shape
```

```
(175850, 4)
```

```
df.sample(10)
```

	CVE ID Number	Vulnerability Type	Summary Text	Processed Text
159305	CVE-2009-2332	Gain Information	CMS Chainuk 1.2 and earlier allows remote atta...	cms chainuk earlier allows remote attackers o...
36660	CVE-2020-7154	Code Execute	A ifviewselectpage expression language injecti...	ifviewselectpage expression language injection...
47975	CVE-2016-1027	Code Execute	Adobe Flash Player before 18.0.0.343 and 19.x ...	adobe flash player x x windows os x linux a...
26560	CVE-2003-0926	Denial of Service	Ethereal 0.9.15 and earlier, and Tethereal, al...	ethereal earlier tethereal allows remote atta...
146399	CVE-2009-	Bypass	Mozilla Firefox executes DOM calls in response	mozilla firefox executes dom calls response ja

TF-IDF

```
TF-IDF->->  Term Frequency*Inverse Document Frequency
for finding out the importance of a token in a document
```

```
import heapq
from collections import defaultdict
freq=defaultdict(int)
```

```
#data is a dictionary with all vulnerability types as keys and it contains information on the occurence of each token
for vul in data.keys():
    items=heapq.nlargest(3,data[vul][0],key=data[vul][0].get)
    d=defaultdict(int)
    for item in items:
      d[item]=data[vul][0][item]
    freq[vul]=d
```

```
freq['Overflow']
```

```
for item in freq.keys():
  display(item)
  display(freq[item])
```

```
import os
```

```
import matplotlib.pyplot as plt
import nltk
from tkinter import *
import seaborn as sns
import matplotlib.pyplot as plt
sns.set()
import scipy

import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_datasets as tfds
from tensorflow.python import keras

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, LSTM

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

```
bag_of_words=defaultdict()
```

```
# bag of words contains top 10 major occuring tokens in a particular vulnerability type. this information is very useful while deciding t
for vuln in data.keys():
    items=heapq.nlargest(10,data[vuln][0],key=data[vuln][0].get)
    d=defaultdict(int)
    for item in items:
      d[item]=data[vuln][0][item]
    bag_of_words[vuln]=d
```

```
bag_of_words['Denial of Service']
```

```
#train_data=tf.keras.preprocessing.sequence.pad_sequences(df['Processed Text'],value=)
```

Supervised Classification models using bag of words

```
from sklearn.feature_extraction.text import CountVectorizer
matrix = CountVectorizer(max_features=1000)
X = matrix.fit_transform(df['Processed Text']).toarray()
```

```
y=df.iloc[:, 1]
```

```
X[0],df['Processed Text'][0]
```

```
(array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 2, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
        0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
 'apache iotdb version     vulnerable denial service attack accepting untrusted patterns regexp queries java  users upgrade addresses issue use later version java avoid')
```

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X, y)
y_gnb = gnb.predict(X)
```

```
from sklearn.linear_model import LogisticRegression
reg = LogisticRegression()
#reg.fit(X, y)
#y_reg = reg.predict(X)
```

```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(X, y)
y_dtc = dtc.predict(X)
```

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(X, y)
y_rfc = rfc.predict(X)
```

```
from sklearn.svm import SVC
svc = SVC()
#svc.fit(X, y)
#y_svc = svc.predict(X)
```

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X, y)
#y_knn = knn.predict(X)
```

```
KNeighborsClassifier()
```

```
from sklearn.metrics import accuracy_score
accuracy_gnb = accuracy_score(y, y_gnb)
accuracy_reg = accuracy_score(y, y_reg)
accuracy_dtc = accuracy_score(y, y_dtc)
accuracy_rfc = accuracy_score(y, y_rfc)
#accuracy_svc = accuracy_score(y, y_svc)
#accuracy_knn = accuracy_score(y, y_knn)
```

```
print("gaussian Naive Bayes(NB):  ",accuracy_gnb)
print("Logistic Regression:  ",accuracy_reg)
print("Decision Tree:  ",accuracy_dtc)
print("Random Forest:  ",accuracy_rfc)
#print(":  %d",accuracy_svc)
#print("K-nearest Neighbors:  ",accuracy_knn)
```

```
df['Estimated']=y_dtc
```

```
df.head(5)
```

```
df.sample(n=20)
```

With low sample

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
samp=10       #10 hard limit
```

```
#p=int((samp*n)//100)
```

```
new=df.sample(10000)
```

```
len(new)
```

```
10000
```

```
new.head(5)
```

	CVE ID Number	Vulnerability Type	Summary Text	Processed Text
54561	CVE-2011-5102	Code Execute	The Investigative Reports web interface in the...	investigative reports web interface triton man...
164341	CVE-2010-3387	gain priviledge	** DISPUTED ** vdrleaktest in Video Disk Reco...	disputed vdrleaktest video disk recorder vd...
115098	CVE-2021-	Cross Site Scripting	Cross-site scripting vulnerability in Message	crosssite scripting vulnerability message

```
newvect=TfidfVectorizer()
Xnew=newvect.fit_transform(new['Processed Text'])
ynew=new['Vulnerability Type']
```

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(Xnew,ynew)

from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(Xnew.toarray(),ynew)

from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(Xnew,ynew)

from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(Xnew,ynew)

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(Xnew,ynew)
print(' ')
```

```
from sklearn.metrics import accuracy_score
from collections import defaultdict
dicnew=defaultdict(int)
```

```
lrnew=lr.predict(Xnew)
dicnew['Logistic Regression Sample TF-IDF']=accuracy_score(ynew, lrnew)
```

```
gnbnew=gnb.predict(Xnew.toarray())
dicnew['Gaussian NB Sample TF-IDF']=accuracy_score(ynew, gnbnew)
```

```
dtcnew=lr.predict(Xnew)
dicnew['Decision Tree Sample TF-IDF']=accuracy_score(ynew, dtcnew)
```

```
rfcnew=rfc.predict(Xnew)
dicnew['Random Forest Sample TF-IDF']=accuracy_score(ynew, rfcnew)
```

```
knnnew=lr.predict(Xnew)
dicnew['K Nearest Neighbour Sample TF-IDF']=accuracy_score(ynew, knnnew)
```

```
dicnew              #TF-IDF vectorizer is used as it assigns value to a token based on its importance in the corpus(dataset).This is supp
```

```
defaultdict(int,
            {'Logistic Regression Sample TF-IDF': 0.8395,
             'Gaussian NB Sample TF-IDF': 0.8666,
```

```
'Decision Tree Sample TF-IDF': 0.8395,
'Random Forest Sample TF-IDF': 0.9633,
'K Nearest Neighbour Sample TF-IDF': 0.8395})
```

```
new['TF-IDF Predicted']=rfcnew
```

```
new.sample(10)
```

	CVE ID Number	Vulnerability Type	Summary Text	Processed Text	T Prec
80083	CVE-2017-8748	Overflow	Internet Explorer in Microsoft Windows 7 SP1, ...	internet explorer microsoft windows sp window...	Ov
9343	CVE-2017-2531	Denial of Service	An issue was discovered in certain Apple produ...	issue discovered certain apple products ios a...	De S
39271	CVE-2019-7797	Code Execute	Adobe Acrobat and Reader versions 2019.010.201...	adobe acrobat reader versions earlier earlie...	E
71082	CVE-1999-0935	Code Execute	classifieds.cgi allows remote attackers to exe...	classifiedscgi allows remote attackers execute...	E
	CVE-		Microsoft Publisher		

Best-Models TF-IDF

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X,y1)
```

```
LogisticRegression()
```

```
ylr=lr.predict(X)
```

```
from sklearn.metrics import accuracy_score
accuracy_score(ylr, y1)
```

```
0.7195280068239978
```

```
df['TF-IDF LR predicted']=ylr
```

```
df.sample(10)
```

	CVE ID Number	Vulnerability Type	Summary Text	Processed Text	TF-IDF LR predicted
94425	CVE-2021-0399	Memory Corruption	In qtaguid_untag of xt_qtaguid.c, there is a p...	qtaguiduntag xtqtaguidc possible memory corrup...	Memory Corruption
4638	CVE-2019-11479	Denial of Service	Jonathan Looney discovered that the Linux kern...	jonathan looney discovered linux kernel defaul...	Denial of Service
61997	CVE-2008-0300	Code Execute	mapFiler.php in Mapbender 2.4 to 2.4.4 allows ...	mapfilerphp mapbender allows remote attacker...	Code Execute
76198	CVE-2019-10877	Overflow	In Teeworlds 0.7.2, there is an integer overfl...	teeworlds integer overflow cmapload enginesha...	Overflow
110348	CVE-2005-	SQL Injection	SQL injection vulnerability in	sql injection vulnerability	Code

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
#rfc.fit(X, y1)
```

```
#yrfc=rfc.predict(X)
```

```
from sklearn.metrics import accuracy_score
#accuracy_score(yrfc, y1)
```

```
#df['TF-IDF RFC predicted']=yrfc
```

```
# df.head(10)
```

Classifier

```
from sklearn.feature_extraction.text import TfidfVectorizer
vec = TfidfVectorizer()
```

TF-IDF

```
n=len(df['Processed Text'])
print(n)
```

```
175850
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
nvectorizer = TfidfVectorizer()
X = nvectorizer.fit_transform(df['Processed Text'])
y1=df['Vulnerability Type']
```

```
X.shape,y1.shape
```

```
((175850, 138817), (175850,))
```

```
df.head(10)
```

	CVE ID Number	Vulnerability Type	Summary Text	Processed Text
0	CVE-2022-43766	Denial of Service	Apache IoTDB version 0.12.2 to 0.12.6, 0.13.0 ...	apache iotdb version vulnerable denial ser...
1	CVE-2022-43365	Denial of Service	IP-COM EW9 V15.11.0.14(9732) was discovered to...	ipcom ew v discovered contain buffer overflow ...
2	CVE-2022-43035	Denial of Service	An issue was discovered in Bento4 v1.6.0-639. ...	issue discovered bento v heapbufferoverflow ap...
3	CVE-2022-43033	Denial of Service	An issue was discovered in Bento4 1.6.0-639. T...	issue discovered bento bad free component aph...
4	CVE-2022-	Denial of Service	The py library through 1.11.0 for Python allow	py library python allows remote attackers con

```
train=df[df['Vulnerability Type']=='Denial of Service']
```

```
train.shape
```

```
(28100, 4)
```

```
train.head(5)
```

	CVE ID Number	Vulnerability Type	Summary Text	Processed Text
0	CVE-2022-43766	Denial of Service	Apache IoTDB version 0.12.2 to 0.12.6, 0.13.0 ...	apache iotdb version vulnerable denial ser...
1	CVE-2022-43365	Denial of Service	IP-COM EW9 V15.11.0.14(9732) was discovered to...	ipcom ew v discovered contain buffer overflow ...
	CVE-			

```
df['DoS']=df['Denial of Service'].where()
```