PERSONIFY

- Data Science and Analytics Project
- MBTI Personality Detection from Persons' Text messages or social posts

In [ ]:

```
# Import All Necessary modules
import pandas as pd
import numpy as np
import sklearn as sk
import matplotlib.pyplot as plt
import math

import re
import nltk
wn = nltk.stem.WordNetLemmatizer()
pstemmer = nltk.stem.PorterStemmer()
from nltk.corpus import stopwords
from nltk.tokenize import sent_tokenize, word_tokenize
# nltk.download() # to choose any pkj to download
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
# from concurrent.futures import ThreadPoolExecutor
import multiprocessing
import time
from tqdm import tqdm
```

```
[nltk_data] Downloading package stopwords to /root/nltk_dat
a...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Data

```
    -Two columns-
        -Personality type
        -Posts text - that portrays the personality
    -Contains at least 50 diferrent post texts separated with |||
```

In [ ]:

```
# Read databasea
odata = pd.read_csv("/content/drive/My Drive/Data_Science/data_sets/mbti.csv
odata.head(10)
```

Out[4]:

	type	posts
0	INFJ	'http://www.youtube.com/watch?v=qsXHcwe3krw|||...
1	ENTP	'I'm finding the lack of me in these posts ver...
2	INTP	'Good one _____ https://www.youtube.com/wat...
3	INTJ	'Dear INTP, I enjoyed our conversation the o...
4	ENTJ	'You're fired.|||That's another silly misconce...
5	INTJ	'18/37 @.@|||Science is not perfect. No scien...
6	INFJ	'No, I can't draw on my own nails (haha). Thos...
7	INTJ	'I tend to build up a collection of things on ...
8	INFJ	I'm not sure, that's a good question. The dist...
9	INTP	'https://www.youtube.com/watch?v=w8-egj0y8Qs||...

In [ ]:

```
#Input text to detect personality type
X = odata["posts"]
# # Output Class values 16 personality types or classes
Y = odata["type"]
Y
```

Out[4]:

```
0       INFJ
1       ENTP
2       INTP
3       INTJ
4       ENTJ
        ...
8670    ISFP
8671    ENFP
8672    INTP
8673    INFP
8674    INFP
Name: type, Length: 8675, dtype: object
```

Data Preprocessing

In [ ]:

```
# Check for missing values
odata.isnull().sum()
# if missing values found either remove row or coloumn or replace with mean,
```

In [ ]:

```
odata.shape
```

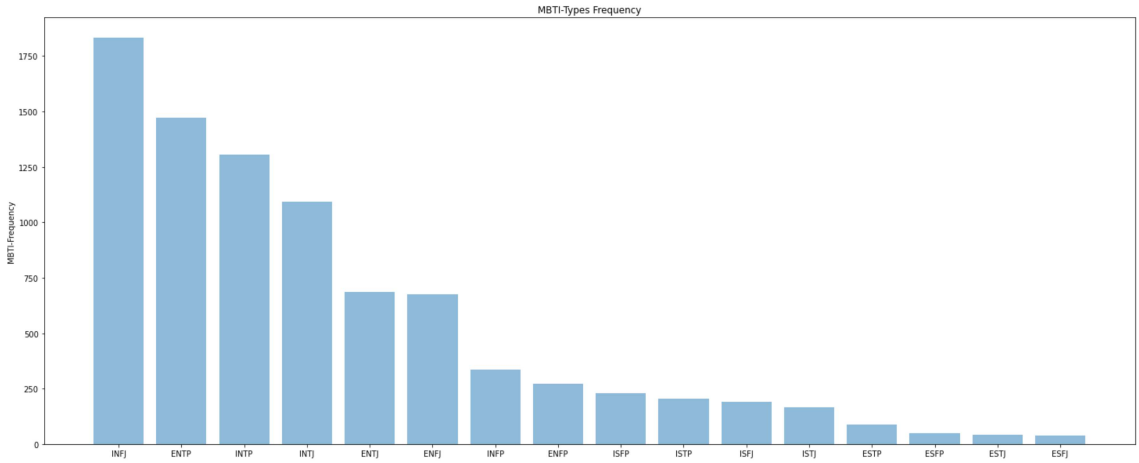
In [ ]:

```
# Showing a bar chart of MBTI Personality types and their occurences in the
mbti_types= Y.drop_duplicates()
mbti_occurances = Y.value_counts()
y_pos = np.arange(len(mbti_types))

plt.figure(figsize=(25,10))
plt.bar(y_pos, mbti_occurances, align='center', alpha=0.5)
plt.xticks(y_pos, mbti_types)
plt.ylabel('MBTI-Frequency')
plt.title('MBTI-Types Frequency')
```

Out[7]:

```
Text(0.5, 1.0, 'MBTI-Types Frequency')
```



In [ ]:

```
#Check if GPU is available
import tensorflow as tf
tf.test.gpu_device_name()
```

Out[4]:

```
''
```

Selective Word Removal

-(removing links, unecessary stoping words like a, the or, and..., or MBTI types if may be given in text and stemize)

-Remove Links
-Tokenize Sentences and words
-Remove Stop words
-Lemmatize (Stemize) words
-Padding (words to ints of same length) - (Only Needed for Deep Learning Models)

In [ ]:

```
!pip install numba
# !pip install cudatoolkit
# !pip install numba cudatoolkit pyculib
# from numba import jit, cuda

import sys
sys.path.append('/content/drive/My Drive/')
!pip install contractions
from tqdm import tqdm
from Data_Science.preprocess import CleanText
ct = CleanText()

# !pip install dask
!pip install 'fsspec>=0.3.3'
import dask.dataframe as dd
from dask.multiprocessing import get
import timeit
# function optimized to run on gpu

# @jit
# (target ="cuda")
def dask_this(df):
  with tqdm(total=len(df)) as pbar:
    res = df['posts'].apply(ct.preprocess_text)
    pbar.update(100)
  return res

%timeit
start_time = time.time()

ddf = dd.from_pandas(data, npartitions=12)
res = ddf.map_partitions(dask_this, meta=pd.Series([], dtype=str, name='post

print("--- %s seconds ---" % (time.time() - start_time))
res
```

```
Requirement already satisfied: numba in /usr/local/lib/pyt
hon3.6/dist-packages (0.48.0)
Requirement already satisfied: setuptools in /usr/local/li
b/python3.6/dist-packages (from numba) (50.3.2)
Requirement already satisfied: numpy>=1.15 in /usr/local/l
ib/python3.6/dist-packages (from numba) (1.18.5)
Requirement already satisfied: llvmlite<0.32.0,>=0.31.0dev
0 in /usr/local/lib/python3.6/dist-packages (from numba)
(0.31.0)
Collecting contractions
  Downloading https://files.pythonhosted.org/packages/00/9
```

```
2/a05b76a692ac08d470ae5c23873cf1c9a041532f1ee065e74b374f21
8306/contractions-0.0.25-py2.py3-none-any.whl (https://fil
es.pythonhosted.org/packages/00/92/a05b76a692ac08d470ae5c2
3873cf1c9a041532f1ee065e74b374f218306/contractions-0.0.25-
py2.py3-none-any.whl)
Collecting textsearch
  Downloading https://files.pythonhosted.org/packages/42/a
8/03407021f9555043de5492a2bd7a35c56cc03c2510092b5ec018cae1
```

In [ ]:

```
# Adding Y (type) values back with their relative posts for a df
processed_data = pd.concat([Y, res], axis=1) # axis 1 = columnj

#Exporting Preprocessed data to a new csv file
processed_data.to_csv("/content/drive/My Drive/Data_Science/data_sets/mbti_p
processed_data
```

Spliting dataset

Train Test Split

Train test split approach
Training set 70%
Test set 30%

In [ ]:

```
# Read partial processed data for comparing the evaluation metrics
import pandas as pd

data = pd.read_csv("/content/drive/My Drive/Data_Science/data_sets/mbti_proc
Y = data["type"]
data.head(10)
```

Out[1]:

	type	posts
0	INFJ	and moment sportscent not top ten play prank w...
1	ENTP	i am find the lack of me in these post veri al...
2	INTP	good one _____ cours to which i say i know tha...
3	INTJ	dear i enjoy our convers the other day esoter ...
4	ENTJ	you are fire that is anoth silli misconcept th...
5	INTJ	scienc is not perfect no scientist claim that ...
6	INFJ	no i can not draw on my own nail haha those we...
7	INTJ	i tend to build up a collect of thing on my de...
8	INFJ	i am not sure that is a good question the dist...
9	INTP	in thi posit where i have to actual let go of ...

In [ ]:

```
#Spliting data into Training and Test sets 80/20% or 70%/30%
from sklearn.model_selection import train_test_split

#Use completely processed data
# X = processed_data["posts"]
# Y = processed_data["type"]

# Using partial processed data
X = data["posts"]
Y = data["type"]

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, ra
X_train
y_train
```

Out[2]:

```
7263    INTJ
8037    ENFP
7864    INFJ
6596    INFP
5008    INTP
        ...
350     INFP
79      INFP
8039    INTP
6936    ISTP
5640    INFJ
Name: type, Length: 6072, dtype: object
```

In [ ]:

Preparing, Vectorizing data

Converting text data into vectors (numbers)
Computer understandable data for ML Model Training

Vectorization

Extracting features(text-to-Nums) from text files

In order to perform machine learning on text documents, we first need to turn the text content into numerical feature vectors.

In [ ]:

```
#Vectorization of text posts into nums(vector counts)
#CountVectorizer -builds a dictionary of features (key(word)->value{num|inde
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(X_train)
X_train_counts.shape

y_train_counts = count_vect.transform(y_train)
y_train_counts.shape



# converting test data set too will be needed.
X_test_counts = count_vect.fit_transform(X_test)
X_test_counts.shape

y_test_counts = count_vect.transform(y_test)
y_test_counts.shape
```

Out[3]:

```
(2603, 47179)
```

In [ ]:

```
# The index value of a word in the vocabulary is linked to its frequency in
count_vect.vocabulary_.get(u'algorithm')
```

Term Frequency (tf)

Dividing the number of occurrences of each word in a document by th
e total number of words in the document
#Term Frequency times
#Inverse Document Frequency" (tf-idf)
Downscaling weights for words that occur in many documents in the c
orpus and are therefore less informative than those that occur only
in a smaller portion of the corpus

In [ ]:

```
#TF-IDF Transformer
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
X_train_tfidf.shape

y_train_tfidf = tfidf_transformer.transform(y_train_counts)
y_train_tfidf.shape

# test data
X_test_tfidf = tfidf_transformer.fit_transform(X_test_counts)
X_test_tfidf

y_test_tfidf = tfidf_transformer.transform(y_test_counts)
y_test_tfidf
```

Out[4]:

```
<2603x47179 sparse matrix of type '<class 'numpy.float64'>'
        with 2603 stored elements in Compressed Sparse Row fo
rmat>
```

In [ ]:

```
# #Pipeline -Combining (vectorizer => transformer => classifier) to work wit
# from sklearn.pipeline import Pipeline
# text_clf = Pipeline([('vect', CountVectorizer()),('tfidf', TfidfTransforme
```

Models Training

Naive Bayes Multinomial Classification

Training Model-1

In [ ]:

```
#Training on 80% Train data
from sklearn.naive_bayes import MultinomialNB
start_time = time.time()
nb_clf = MultinomialNB()
#text_clf needs training data into floats (nums)
nb_clf.fit(X_train_tfidf, y_train)
print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 0.18603134155273438 seconds ---
```

In [ ]:

```
# Predection of Trainded Model-1 on 20% Test Data
# Test data must be featured through the same chain of methods used for trai
X_test_count = count_vect.transform(X_test)
#Here only use tdidf-Transform() method instead of fitTransfer() - since the
X_test_tfidf = tfidf_transformer.transform(X_test_count)
y_pred = nb_clf.predict(X_test_tfidf)
```

Model 1 - Naive Bayes Evaluation matrix and score

- Confusion Matrix & Accuracy
- Precision & Recall
- F1-Score & Support

In [ ]:

```
#Confusion Matrix and Accuracy Score of Naive Bayes
from sklearn.metrics import confusion_matrix
from sklearn import metrics
print(metrics.classification_report(y_test, y_pred, zero_division=1))

print("Confusion Metrix: \n", confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

        ENFJ       1.00      0.00      0.00        53
        ENFP       1.00      0.00      0.00       201
        ENTJ       1.00      0.00      0.00        60
        ENTP       1.00      0.00      0.00       206
        ESFJ       1.00      0.00      0.00        17
        ESFP       1.00      0.00      0.00        14
        ESTJ       1.00      0.00      0.00        10
        ESTP       1.00      0.00      0.00        25
        INFJ       1.00      0.00      0.00       450
        INFP       0.21      1.00      0.34       542
        INTJ       1.00      0.00      0.00       355
        INTP       1.00      0.00      0.00       397
        ISFJ       1.00      0.00      0.00        53
        ISFP       1.00      0.00      0.00        78
        ISTJ       1.00      0.00      0.00        56
        ISTP       1.00      0.00      0.00        86

    accuracy                           0.21      2603
   macro avg       0.95      0.06      0.02      2603
weighted avg       0.84      0.21      0.07      2603

Confusion Metrix:
 [[  0   0   0   0   0   0   0   0   0  53   0   0   0   0   0
0   0]
 [  0   0   0   0   0   0   0   0   0 201   0   0   0   0   0
0]
 [  0   0   0   0   0   0   0   0   0  60   0   0   0   0   0
0]
 [  0   0   0   0   0   0   0   0   0 206   0   0   0   0   0
0]
 [  0   0   0   0   0   0   0   0   0  17   0   0   0   0   0
0]
 [  0   0   0   0   0   0   0   0   0  14   0   0   0   0   0
0]
 [  0   0   0   0   0   0   0   0   0  10   0   0   0   0   0
0]
 [  0   0   0   0   0   0   0   0   0  25   0   0   0   0   0
0]
 [  0   0   0   0   0   0   0   0   0 450   0   0   0   0   0
```

```
0]
 [  0   0   0   0   0   0   0   0   0 542   0   0   0   0   0
0]
 [  0   0   0   0   0   0   0   0   0 355   0   0   0   0   0
0]
 [  0   0   0   0   0   0   0   0   0 397   0   0   0   0   0
0]
 [  0   0   0   0   0   0   0   0   0  53   0   0   0   0   0
0]
 [  0   0   0   0   0   0   0   0   0  78   0   0   0   0   0
0]
 [  0   0   0   0   0   0   0   0   0  56   0   0   0   0   0
0]
 [  0   0   0   0   0   0   0   0   0  86   0   0   0   0   0
0]]
```

In [ ]:

```
#Saving the Naive Bayes machine learning model to a file
!pip install joblib
import joblib
joblib.dump(nb_clf, "/content/drive/My Drive/Data_Science/personify_nb_model
```

```
Requirement already satisfied: joblib in /usr/local/lib/pytho
n3.6/dist-packages (0.17.0)
```

Out[39]:

```
['/content/drive/My Drive/Data_Science/personify_nb_model4.pk
l']
```

Support Vector Machine (SVM) Classification Model

Training Model-2

In [ ]:

```
#SVM -Feature Selection and Training
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn import svm
from sklearn.pipeline import Pipeline

svm_clf = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer(

svm_clf.fit(X_train.values.astype("U"), y_train)
```

Out[40]:

```
Pipeline(memory=None,
         steps=[('vect',
                 CountVectorizer(analyzer='word', binary=Fals
e,
                                 decode_error='strict',
                                 dtype=<class 'numpy.int64'>,
encoding='utf-8',
                                 input='content', lowercase=T
rue, max_df=1.0,
                                 max_features=None, min_df=1,
                                 ngram_range=(1, 1), preproce
ssor=None,
                                 stop_words=None, strip_accen
ts=None,
                                 token_pattern='(?u)\\b\\w\\w
+\\b',
                                 tokenizer=None, vocabulary=N
one)),
                ('tfidf',
                 TfidfTransformer(norm='l2', smooth_idf=True,
                                  sublinear_tf=False, use_idf
=True)),
                ('clf',
                 SVC(C=1.0, break_ties=False, cache_size=200,
class_weight=None,
                     coef0=0.0, decision_function_shape='ov
o', degree=3,
                     gamma='scale', kernel='rbf', max_iter=-
1,
                     probability=False, random_state=None, sh
rinking=True,
                     tol=0.001, verbose=False))],
         verbose=False)
```

In [ ]:

```
#Predicting with SVM Model
y_pred_svm = svm_clf.predict(X_test)
```

Model 2 - Support Vector Machine Evaluation matrix and score

Confusion Matrix & Accuracy

Precision & Recall

F1-Score & Support

In [ ]:

```
#Confusion Matrix and Accuracy Score of Naive Bayes
from sklearn.metrics import confusion_matrix
from sklearn import metrics
print(metrics.classification_report(y_test, y_pred_svm, zero_division=1))

print("Confusion Metrix: \n", confusion_matrix(y_test, y_pred_svm))
```

```
              precision    recall  f1-score   support

        ENFJ       1.00      0.00      0.00        43
        ENFP       0.53      0.06      0.10       139
        ENTJ       1.00      0.00      0.00        35
        ENTP       0.36      0.06      0.10       138
        ESFJ       1.00      0.00      0.00        13
        ESFP       1.00      0.00      0.00         8
        ESTJ       1.00      0.00      0.00         8
        ESTP       1.00      0.00      0.00        16
        INFJ       0.34      0.37      0.35       295
        INFP       0.35      0.76      0.48       356
        INTJ       0.39      0.25      0.30       239
        INTP       0.39      0.64      0.49       277
        ISFJ       1.00      0.00      0.00        35
        ISFP       1.00      0.00      0.00        47
        ISTJ       1.00      0.00      0.00        24
        ISTP       1.00      0.02      0.03        62

    accuracy                           0.36      1735
   macro avg       0.77      0.13      0.12      1735
weighted avg       0.48      0.36      0.29      1735

Confusion Metrix:
 [[  0   0   0   0   0   0   0   0  17  21   3   2   0   0
0   0]
 [  0   8   0   0   0   0   0   0  33  85   4   9   0   0   0
0]
 [  0   0   0   0   0   0   0   0   9   9   8   9   0   0   0
0]
 [  0   1   0   8   0   0   0   0  24  42  11  52   0   0   0
0]
 [  0   0   0   0   0   0   0   0   8   5   0   0   0   0   0
0]
 [  0   0   0   0   0   0   0   0   1   5   0   2   0   0   0
0]
 [  0   1   0   0   0   0   0   0   0   3   1   3   0   0   0
0]
 [  0   1   0   0   0   0   0   0   2   7   1   5   0   0   0
0]
 [  0   1   0   2   0   0   0   0 110 143  17  22   0   0   0
```

```
0]
 [  0   1   0   0   0   0   0   0  38 269   8  40   0   0   0
0]
 [  0   0   0   5   0   0   0   0  37  57  59  81   0   0   0
0]
 [  0   0   0   4   0   0   0   0  20  61  16 176   0   0   0
0]
 [  0   1   0   0   0   0   0   0   9  20   1   4   0   0   0
0]
 [  0   1   0   1   0   0   0   0  10  28   2   5   0   0   0
0]
 [  0   0   0   0   0   0   0   0   1   6  10   7   0   0   0
0]
 [  0   0   0   2   0   0   0   0   9  10  10  30   0   0   0
1]]
```

In [ ]:

```
#Saving the SVM machine learning model to a file
!pip install joblib
import joblib
joblib.dump(svm_clf, "/content/drive/My Drive/Data_Science/personify_svm_mod
```

```
Requirement already satisfied: joblib in /usr/local/lib/pytho
n3.6/dist-packages (0.17.0)
```

Out[27]:

```
['/content/drive/My Drive/Data_Science/personify_svm_model.pk
l']
```

Passive Aggressive Classifier

Training Model-3

In [ ]:

```
#SVM -Feature Selection and Training
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.linear_model import PassiveAggressiveClassifier
from sklearn.pipeline import Pipeline

pa_clf = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()
                   ('clf', PassiveAggressiveClassifier(max_iter=1000, rando

pa_clf.fit(X_train, y_train)
```

Out[24]:

```
Pipeline(memory=None,
         steps=[('vect',
                 CountVectorizer(analyzer='word', binary=Fals
e,
                                 decode_error='strict',
                                 dtype=<class 'numpy.int64'>,
encoding='utf-8',
                                 input='content', lowercase=T
rue, max_df=1.0,
                                 max_features=None, min_df=1,
                                 ngram_range=(1, 1), preproce
ssor=None,
                                 stop_words=None, strip_accen
ts=None,
                                 token_pattern='(?u)\\b\\w\\w
+\\b',
                                 tokenizer=None, vocabulary=N
on...
                 TfidfTransformer(norm='l2', smooth_idf=True,
                                  sublinear_tf=False, use_idf
=True)),
                ('clf',
                 PassiveAggressiveClassifier(C=1.0, average=F
alse,
                                             class_weight=Non
e,
                                             early_stopping=F
alse,
                                             fit_intercept=Tr
ue, loss='hinge',
                                             max_iter=1000, n
_iter_no_change=5,
                                             n_jobs=None, ran
dom_state=0,
                                             shuffle=True, to
```

```
l=0.001,
                                                validation_fract
ion=0.1, verbose=0,
                                                warm_start=Fals
e))],
          verbose=False)
```

In [ ]:

```
#Prediction using Passive Aggressive Classifier
y_pred_pac = pa_clf.predict(X_test)
```

In [ ]:

```
#Confusion Matrix and Accuracy Score of Naive Bayes
from sklearn.metrics import confusion_matrix
from sklearn import metrics
print(metrics.classification_report(y_test, y_pred_pac, zero_division=1))

print("Confusion Metrix: \n", confusion_matrix(y_test, y_pred_pac))
```

```
              precision    recall  f1-score   support

        ENFJ       0.11      0.05      0.07        43
        ENFP       0.28      0.30      0.29       139
        ENTJ       0.20      0.11      0.15        35
        ENTP       0.32      0.23      0.27       138
        ESFJ       0.00      0.00      0.00        13
        ESFP       0.00      0.00      0.00         8
        ESTJ       0.00      0.00      0.00         8
        ESTP       0.00      0.00      0.00        16
        INFJ       0.39      0.44      0.42       295
        INFP       0.44      0.55      0.49       356
        INTJ       0.37      0.36      0.36       239
        INTP       0.43      0.49      0.46       277
        ISFJ       0.44      0.20      0.27        35
        ISFP       0.24      0.09      0.12        47
        ISTJ       0.32      0.29      0.30        24
        ISTP       0.36      0.27      0.31        62

    accuracy                           0.38      1735
   macro avg       0.24      0.21      0.22      1735
weighted avg       0.36      0.38      0.37      1735

Confusion Metrix:
 [[  2   3   0   3   0   0   0   0  16  11   3   1   1   1
1   1]
 [  5  42   1   7   0   0   0   1  24  39   6  13   0   0   0
1]
 [  1   3   4   4   0   0   0   0   3   3  10   4   1   0   0
2]
 [  1  11   3  32   0   0   0   0  11  16  14  39   0   2   2
7]
 [  0   2   0   0   0   0   0   1   6   3   0   0   0   0   0
1]
 [  0   1   0   2   0   0   1   0   2   0   0   2   0   0   0
0]
 [  0   1   0   1   0   0   0   0   0   3   2   0   0   1   0
0]
 [  0   2   0   1   0   0   0   0   1   4   3   2   0   0   0
3]
 [  4  18   3   6   0   0   0   2 131  81  27  14   0   3   3
```

```
3]
 [  1  25   3   5   0   0   1   0  57 197  24  34   4   3   0
2]
 [  1  13   3  17   0   0   0   1  33  31  87  45   0   1   1
6]
 [  1  17   1  14   1   0   0   0  25  39  40 135   0   0   2
2]
 [  1   5   1   0   1   0   0   1   4   5   1   3   7   1   4
1]
 [  1   4   1   1   0   0   0   0  14  10   5   5   1   4   1
0]
 [  0   1   0   2   0   0   0   0   1   2   9   1   0   0   7
1]
 [  0   2   0   6   0   1   0   0   4   8   7  13   2   1   1
17]]
```

In [ ]:

```
#Saving the SVM machine learning model to a file
!pip install joblib
import joblib
joblib.dump(pa_clf, "/content/drive/My Drive/Data_Science/personify_pac_mode
```

```
Requirement already satisfied: joblib in /usr/local/lib/pytho
n3.6/dist-packages (0.17.0)
```

Out[31]:

```
['/content/drive/My Drive/Data_Science/personify_pac_model.pk
l']
```

Trying out Neural Nets for better performance on data set

Deep Learning Models

Training Model 4

1. Simple Neural Network

In [1]:

```
# # Tokenize the data into a format that can be used by the word embeddings
# from keras.preprocessing.text import Tokenizer

# tokenizer = Tokenizer(num_words=5000, lower=False)
# tokenizer.fit_on_texts(X_train)

# print(X_train[2])
# X_train = tokenizer.texts_to_sequences(X_train)
# X_test = tokenizer.texts_to_sequences(X_test)

# # vocab_size = len(tokenizer.word_index) + 1  # Adding 1 because of reserv

# # tokenizer.fit_on_texts(y_train)
# # y_train = tokenizer.texts_to_sequences(y_train)
# # y_test = tokenizer.texts_to_sequences(y_test)

# print(X_train[2])
```

Word Vector Padding

CountVectorizer produces word vectors with diferent lengths. pad_sequence(), simply pads the sequence of words with zeros

In [2]:

```
# from keras.preprocessing.sequence import pad_sequences
# maxlen = 100

# X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
# X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)

# # y_train = pad_sequences(y_train, padding='post', maxlen=maxlen)
# # y_test = pad_sequences(y_test, padding='post', maxlen=maxlen)


# print(X_train[0, :])
# # print(X_train_tfidf)
# # print(X_test_tfidf)
```

In [3]:

```
# # import tensorflow as tf
# from keras.models import Sequential
# from keras.layers import Dense
# from keras.layers import Flatten
# from keras.layers.convolutional import Conv1D
# from keras.layers import GlobalMaxPool1D
# from keras.layers.embeddings import Embedding
# from keras.preprocessing import sequence


# input_dim = X_train.shape[0]  # Number of features
# embedding_dim = 50

# model = Sequential()
# model.add(Embedding(input_dim=input_dim,
#                             output_dim=embedding_dim,
#                             input_length=maxlen,))
# model.add(GlobalMaxPool1D())
# # model.add(Flatten())
# #Adding two Dense layers in the sequential nn model
# model.add(Dense(10, activation='relu'))
# model.add(Dense(1, activation='sigmoid'))
# #Uses tensorflow in the backend.
# X_train.shape[0]
```

In [4]:

```
#Giving model a loss function, optimizer and a metrics for evaluation
# model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accu
# model.summary()
```

Epochs and Training NN

Training in neural networks is an iterative process, the training w on’t just stop after it is done. You have to specify the number of iterations you want the model to be training. Those completed iter ations are commonly called epochs.

In [5]:

```
# import numpy as np
# X_train = np.asarray(X_train)
# y_train = np.asarray(y_train)
# X_test = np.asarray(X_test)
# y_test = np.asarray(y_test)

# history = model.fit(X_train, y_train, epochs=5, verbose=False, validation_

# loss, accuracy = model.evaluate(X_train, y_train, verbose=False)
# print("Training Accuracy: {:.4f}".format(accuracy))
# loss, accuracy = model.evaluate(X_test, y_test, verbose=False)
# print("Testing Accuracy:  {:.4f}".format(accuracy))
```

Parameter tuning using grid search

Hyper Parameter Tuning