CS8111
MACHINE LEARNING COURSEWORK

By

Student ID: 220528414

PROJECT 1:

PREDICTING SEVERITY OF ACCIDENT BASED ON FARS DATASET

PROJECT 2:

SENTIMENT ANALYSIS ON AIRLINE INDUSTRY CUSTOMER REVIEWS

PROJECT 1 - PROBLEM STATEMENT:

We have a FARS dataset which is the collection of statistics of US road traffic accidents. The class label is the severity of the accident based on the given 20 features. We need to classify the severity of the accidents.

DATA LOADING & PREPROCESSING:

We are fetching the data after installing the pbml library using (pip install pbml). After loading, the data description is as follows:

```
[3]  data = fetch_data('fars')
     for i in data.describe():
         print(i)
```

```
CASE_STATE
AGE
SEX
PERSON_TYPE
SEATING_POSITION
RESTRAINT_SYSTEM-USE
AIR_BAG_AVAILABILITY/DEPLOYMENT
EJECTION
EJECTION_PATH
EXTRICATION
NON_MOTORIST_LOCATION
POLICE_REPORTED_ALCOHOL_INVOLVEMENT
METHOD_ALCOHOL_DETERMINATION
ALCOHOL_TEST_TYPE
ALCOHOL_TEST_RESULT
POLICE-REPORTED_DRUG_INVOLVEMENT
METHOD_OF_DRUG_DETERMINATION
DRUG_TEST_TYPE
DRUG_TEST_RESULTS_(1_of_3)
DRUG_TEST_TYPE_(2_of_3)
DRUG_TEST_RESULTS_(2_of_3)
DRUG_TEST_TYPE_(3_of_3)
DRUG_TEST_RESULTS_(3_of_3)
HISPANIC_ORIGIN
TAKEN_TO_HOSPITAL
RELATED_FACTOR_(1)-PERSON_LEVEL
RELATED_FACTOR_(2)-PERSON_LEVEL
RELATED_FACTOR_(3)-PERSON_LEVEL
RACE
target
```

X variable has all the features and y is the class label (i.e, severity of the accident).

```
X,y = fetch_data('fars',return_X_y=True)
X,y
```

```
(array([[ 0, 34,  1, ..., 29, 19, 17],
        [ 0, 20,  1, ..., 29, 19, 17],
        [ 0, 43,  1, ..., 29, 19,  4],
        ...,
        [50,  7,  0, ..., 29, 19, 11],
        [50,  4,  0, ..., 29, 19, 11],
        [50, 61,  1, ..., 29, 19, 17]]), array([1, 1, 1, ..., 6, 6, 1]))
```

```
[ ]  X = pd.DataFrame(X)
     #y = pd.DataFrame(y)
     y = LabelEncoder().fit_transform(y)
```

We have used sklearn.preprocessing.LabelEncoder to encode target labels (y) with value between 0 and n_classes-1.

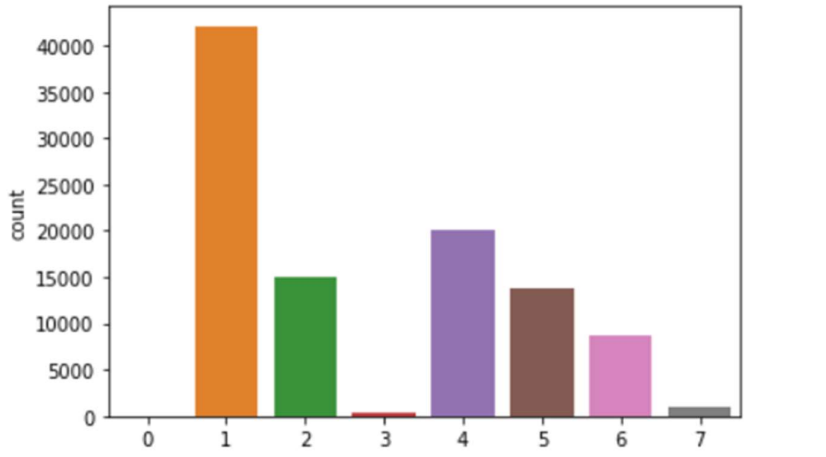
y is passed to the fit_transform() method to fit label encoder with required parameters and return the encoded labels.

DATA PLOTTING:

On plotting the data, it can be seen that the class labels are is highly imbalanced.

```
[ ]  sns.countplot(x=y)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5e27feae50>
```

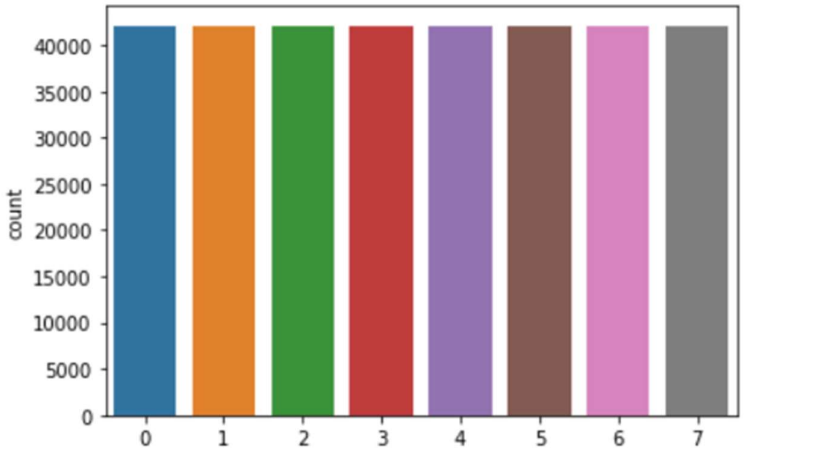


In order to remove the class label imbalance, oversampling is implemented. The smote() function is used which is an algorithm to oversample the minority class through various methods like duplication, etc. This is a type of data augmentation for the minority class and is referred to as the **Synthetic Minority Oversampling Technique.** After oversampling, the class imbalance is removed as can be visualized below.

```
#oversample = RandomOverSampler(sampling_strategy='minority')
oversample = SMOTE()
X_over, y_over = oversample.fit_resample(X, y)
```

```
[ ] sns.countplot(x=y_over)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5e27a2e610>
```



DATA SPLITTING:

The train_test_split from sklearn.model_selection is used to randomly split the data. The data is split such 25% of the entire data is used as test subset and remaining 75% is used as training subset. The accuracy score of each of the machine learning models, which are implemented later is based on this splitting of data.

```
[17] from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X_over, y_over, test_size = 0.25, random_state = 0)
```

DATA NORMALIZATION:

The data is normalized with the use of StandardScaler. This standardizes the features by removal of the mean and scaling to unit variance.This helps in more efficient calculations when there is a zero centred distribution. Also, after standardization, ML models convergence occurs comparatively faster.

The standard score of a sample x is calculated as:

z = (x - u) / s

where u is the mean of the training samples or zero if with_mean = False, and s is the standard deviation of the training samples or one if with_std = False.

IMPLEMENTATION OF MACHINE LEARNING ALGORITHMS:

1. **LOGISTIC REGRESSION:**

It is a ML algorithm which is generally used for binary classification where the sigmoid function is used. In case of multi classification, cross entropy or softmax()is used. In this code, Logistic Regression uses the cross-entropy loss when the 'multi_class' option is set to 'multinomial'.

Both 'saga' and 'newton-cg' as used as optimizers and the best solution is taken. All the parameters used are shown below:

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(solver='newton-cg', multi_class='multinomial', random_state = 10)
classifier.fit(X_train, y_train)
```

```
LogisticRegression(multi_class='multinomial', random_state=10,
                   solver='newton-cg')
```

```
[ ]  from sklearn.metrics import confusion_matrix, accuracy_score
     y_pred = classifier.predict(X_test)


     pred_y = np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1)

     df_pred =pd.DataFrame(pred_y,index=None)
     column_names = ['y_pred','y_test']
     df_pred.columns = column_names
```

```
[ ]  cm = confusion_matrix(y_test, y_pred)
     print(cm)
```

```
[[10164     0     0   205   266     0    29     0]
 [  171  9082   223   210   378   161   167   140]
 [   71    65  4024  1597     5  4742   255     8]
 [   19    90   890  6559   187  2084   274   308]
 [  510   113     3     8  8892    15    98   770]
 [   73   108  1831  1571   340  5938   589    44]
 [  211   170   730   949  2885  3767  1625   121]
 [  121   167    11    47   889    37   306  8919]]
```

```
[ ]  accuracy_score(y_test, y_pred)
```

```
0.6553685060309621
```

The accuracy score after implementing this model is 65.5%.

2. **RANDOM FOREST CLASSIFIER:**

It is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time.

The parameters used in this code are n_estimators which is set as 10, criterion is set to ‘entropy’ & ‘gini’ and random_state is set as 0.

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

```
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=0)
```

```
[ ]  from sklearn.metrics import confusion_matrix, accuracy_score
     y_pred = classifier.predict(X_test)
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     accuracy_score(y_test, y_pred)
```

```
[[10662     0     0     0     1     1     0     0]
 [    0 10512     1     0     4     0     0    15]
 [    0     0  8584    46     7  1529   595     6]
 [    0     0     6 10387     0    10     8     0]
 [    0     0     7     5  9526    72   650   149]
 [    0     0  1826    50   212  7391   991    24]
 [    0     0   774    43  1085   996  7538    22]
 [    0     1     4     3    41     2     9 10437]]
0.8908372115110647
```

The Accuracy score after implementing this model is 89.08%.

3. **Kneighbors Classifier:**
   This Classifier is based on implementing the k-nearest neighbors vote.

   The parameters used here are n_neighbors (i.e no. of neighbors) set as 5, metric set as ‘minkowski’ and p (power parameter for ‘minkowski’ metric) set as 2.

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

```
KNeighborsClassifier()
```

```
[ ]  from sklearn.metrics import confusion_matrix, accuracy_score
     y_pred = classifier.predict(X_test)
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     accuracy_score(y_test, y_pred)
```

```
[[10663     0     0     0     0     1     0     0]
 [    0 10472    18     2     6    13     2    19]
 [    0    11  7562   148     8  2246   782    10]
 [    0     0     9 10374     0    19     9     0]
 [   11     2    23    24  9113    88   895   253]
 [    7     4  2404   149   284  6397  1218    31]
 [    1     0  1106   114  1304  1347  6511    75]
 [    0     1     5     4    68     9    24 10386]]
0.8485848608604806[[10663     0     0     0     0     1     0     0]
 [    0 10472    18     2     6    13     2    19]
 [    0    11  7562   148     8  2246   782    10]
 [    0     0     9 10374     0    19     9     0]
 [   11     2    23    24  9113    88   895   253]
 [    7     4  2404   149   284  6397  1218    31]
 [    1     0  1106   114  1304  1347  6511    75]
 [    0     1     5     4    68     9    24 10386]]
0.8485848608604806
```

The accuracy sore received is 84.85%.

4. **Support Vector Machine:**

   Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). SVM can also be used as a multi class classification. SVM maps training examples to points in space so as to maximise the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

   In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. (Taken from Wikipedia)

   We have set the parameters as kernel = 'linear', random_state = 0.

```
[9]  from sklearn.svm import SVC
     classifier = SVC(kernel = 'linear', random_state = 0)
     classifier.fit(X_train, y_train)
```

```
SVC(kernel='linear', random_state=0)
```

```
[10] from sklearn.metrics import confusion_matrix, accuracy_score
     y_pred = classifier.predict(X_test)
     cm = confusion_matrix(y_test, y_pred)
     print(cm)
     accuracy_score(y_test, y_pred)
```

```
[[10556     0     0     1   107     0     0     0]
 [  129  9241    26   178   352   330   143   133]
 [   11    51  3504  1256     5  5838    92    10]
 [   13    89   693  6967   213  2102   217   117]
 [  242    70     2     5  9421    17    31   621]
 [   24    70  1248  1168   392  7124   417    51]
 [   72   129   407   756  2989  4568  1453    84]
 [   99   115     3    34   878    35   352  8981]]
0.679634818121379
```

We are getting the accuracy as 67.69%.

We have tried using GridSearchCV for hyperparameter tuning which implements a “fit” and a “score” method. The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid. (Modified after Taken from Wikipedia & scikit-learn.org)

```
[ ]  from sklearn.model_selection import GridSearchCV

     params = [{'C':[0.25,0.5,0.75,1],"kernel":['linear']},
                 {'C':[0.25,0.5,0.75,1],"kernel":['rbf'],'gamma':[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]}]

     GS = GridSearchCV(estimator = classifier,
                       param_grid=params,
                       scoring = 'accuracy',
                       cv = 10,
                       n_jobs = -1)

     GS.fit(X_train,y_train)

     best_accuracy = grid_search.best_score_

     optimized_para = grid_search.best_params_


     print("Best Accuracy: {:.2f} %".format(best_accuracy.mean()*100))

     print ("optimized parameters:".format(optimized_para))

     y_pred = GS.predict(X_test)
     accuracy_score(y_test, y_pred)
```

5. **Decision Trees (DTs)** are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation. (Taken from skikit-learn.org)

   A tree is built by splitting the source set, constituting the root node of the tree, into subsets—which constitute the successor children. The splitting is based on a set of splitting rules based on classification features. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node has all the same values of the target variable, or when splitting no longer adds value to the predictions. (Taken from Wikipedia)

```
[ ]
    params = {'max_depth' : [5,8,15,None],
              'max_features': ['sqrt','log2',None],
              }
    dt = DecisionTreeClassifier()
    scorer = make_scorer(f1_score, average = 'weighted')
    rdt = RandomizedSearchCV(dt, param_distributions=params, scoring= scorer, verbose=1, cv=2)
    rdt.fit(X_train,y_train)
```

```
Fitting 2 folds for each of 10 candidates, totalling 20 fits
RandomizedSearchCV(cv=2, estimator=DecisionTreeClassifier(),
                   param_distributions={'max_depth': [5, 8, 15, None],
                                        'max_features': ['sqrt', 'log2', None]},
                   scoring=make_scorer(f1_score, average=weighted), verbose=1)
```

RandomizedSearchCV is used for hyperparameter tuning.

```
[ ] rdt.best_params_
```

```
{'max_features': None, 'max_depth': None}
```

```
[ ] modelD = DecisionTreeClassifier(max_features= None,max_depth=None,criterion='entropy')
    modelD.fit(X_train,y_train)
```

```
DecisionTreeClassifier(criterion='entropy')
```

```
[ ] y_pred_on_test = modelD.predict(X_test)
```

```
[ ] from sklearn.metrics import confusion_matrix, accuracy_score
    accuracy_score(y_test, y_pred_on_test)
```

```
0.8824674707949473
```

The accuracy score is 88.24%. After this, we have used K-fold cross validation to improve our Decision tree model.

CROSS VALIDATION EVALUATION:

K-fold CV: Problems faced with normal cross validation is that the training data gets reduced. In ML, we know that the more the data we have, the stronger our learning gets. K-fold cross validation uses a strategy in which we smartly use all of our data to train and all of it to validate. This way we get a large data to train our model.

```
[ ]  from sklearn.model_selection import cross_val_score
```

```
[ ]  accuracies = cross_val_score(estimator=dt, X =X_train, y=y_train,cv=100)

     print("Accuracy: {:.2f} %".format(accuracies.mean()*100))

     print ("Standard deviation: {:.2f}%".format(accuracies.std()*100))
```

```
Accuracy: 87.81 %
Standard deviation: 0.59%
```

SL No.	MODEL NAME	ACCURACY	REMARKS
1	LOGISTIC REGRESSION	65.5%	
2	RANDOM FOREST CLASSIFIER	89.08%	This model performed the best.
3	Kneighbors CLASSIFIER	84.85%	
4	SUPPORT VECTOR MACHINE	67.69%	GridSearch Cross Validation is applied on this model after which the accuracy should improve.
5	DECISION TREES	88.24%	This accuracy is received after hyperparameter tuning with RandomSearch Cross Validation. After K-fold CV hyperparameter tuning on this model, the accuracy received is 87.81%.

PROJECT 2:

SENTIMENT ANALYSIS

Sentiment analysis is an **analytical approach to natural language processing (NLP) that uses statistics, and other machine learning algorithms to determine the emotional tone/ context behind a body of text**. It is widely used to understand and evaluate customer reactions, online reviews, customer ratings, social media posts, and other content.

PROBLEM STATEMENT:

Here, we are provided with Train, Dev and Test datasets of customer reviews of an airline company. We have to perform data cleaning, data preprocessing, apply shallow and deep classifiers, use ensembled approaches, apply machine learning techniques and perform data augmentation, if necessary, to predict the sentiments of the Test dataset.

For shallow classifier, the Naïve Bayes classifier is applied, for Deep Learning, SNN, CNN and LSTM (RNN) is tried.

Code 1 demonstrates the entire project with Naïve Bayes classifier, and code 2 demonstrates the entire project with SNN.

CODE 1 with NAÏVE BAYES MODEL:

DATA LOADING:

At first, all the required libraries are imported and the training and validation datasets are loaded with Pandas dataframes.

```
[ ]  import numpy as np
     import pandas as pd
     import re
     import nltk
     nltk.download('stopwords')
     from nltk.corpus import stopwords
     from nltk.stem.porter import PorterStemmer
     from sklearn.feature_extraction.text import CountVectorizer
```

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

```
[ ]  ps = PorterStemmer()
     all_stopwords = stopwords.words('english')
     all_stopwords.remove('not')
```

```
[ ]  dataTrain = pd.read_csv('/content/drive/MyDrive/Sentiment Analysis Project final/Tweets_train.csv')
     dataValidate = pd.read_csv('/content/drive/MyDrive/Sentiment Analysis Project final/Tweets_dev.csv')
```

```
[ ]  print(dataTrain.shape)
     print(dataValidate.shape)
```

```
(11858, 3)
(1318, 3)
```

DATA PRE-PROCESSING

The “text” in both the training and validation datasets are cleaned and processed here. The text is manipulated using regex, the entire text is changed to lowercase, the words in the texts are stemmed and all the stopwords are cleaned.

```
corpus_train=[]
for x in range(0, len (dataTrain)):
    dataClean = re.sub('[^a-zA-Z]',' ',dataTrain['text'][x])
    dataClean = dataClean.lower()
    dataClean = dataClean.split()
    dataClean = [ps.stem(word) for word in dataClean if not word in set(all_stopwords)]
    dataClean=' '.join(dataClean)
    corpus_train.append(dataClean)
corpus_train
```

```
[ ]  corpus_validate=[]
     for i in range(0, len (dataValidate)):
         dataClean = re.sub('[^a-zA-Z]',' ',dataValidate['text'][i])
         dataClean = dataClean.lower()
         dataClean = dataClean.split()
         dataClean = [ps.stem(word) for word in dataClean if not word in set(all_stopwords)]
         dataClean=' '.join(dataClean)
         corpus_validate.append(dataClean)
     corpus_validate
```

CountVectorizer is used to convert the text into token counts and represent it in a sparse matrix and same is dumped as a pkl file.

```
[34] cv_Train = CountVectorizer(max_features=500)
     x= cv_Train.fit_transform(corpus_train).toarray()
     y = list(dataTrain.iloc[:,-1].values)

     cv_Validate = CountVectorizer(max_features=500)
     x1=cv_Validate.fit_transform(corpus_validate).toarray()
     y1= list(dataValidate.iloc[:,-1].values)
```

```
import pickle

bow_path = 'DataTrain.pkl'
pickle.dump(cv_Train, open(bow_path,"wb"))

bow_path1 = 'DataValidate.pkl'
pickle.dump(cv_Validate, open(bow_path1,"wb"))
```

MODEL TRAINING

The Naïve Bayes classifier is used to train the model. Naïve Bayes classifier is based on the Bayes theorem. It is **a probabilistic machine learning model which is used for classification task**s.

```
[26] from sklearn.naive_bayes import GaussianNB
     classifier = GaussianNB()
     classifier.fit(x,y)

     import joblib
     joblib.dump(classifier,"NaiveBayesModel")
```

```
['NaiveBayesModel']
```

```
[27] y_prediction = classifier.predict(x1)
```

```
[28] from sklearn.metrics import confusion_matrix, accuracy_score
     cm = confusion_matrix(y1,y_prediction)
     print(cm)

     print(accuracy_score(y1,y_prediction))
```

```
[[536 109 181]
 [113  80  86]
 [ 83  55  75]]
0.5242792109256449
```

The Naives Bayes classifier model is stored using joblib for future use. In the y_prediction variable as shown in the code, the predicted sentiments are stored based on the review texts in the classifier. Finally, in the confusion matrix, we can see all the false positives, false negatives, true positives and true negatives. The accuracy of this model is calculated as 52.42%.

PREDICTION ON NEW TEST DATASETS

Now, the test dataset is loaded and the same data cleaning and pre-processing tasks are performed on the test dataset texts.

```
[29] dataTest = pd.read_csv('/content/drive/MyDrive/Sentiment Analysis Project final/New_Tweets_test-1.csv')
     print(dataTest.shape)

     (1464, 4)
```

```
corpus_Test=[]
for i in range(0, len (dataTest)):
    dataClean = re.sub('[^a-zA-Z]',' ',dataTest['text'][i])
    dataClean = dataClean.lower()
    dataClean = dataClean.split()
    dataClean = [ps.stem(word) for word in dataClean if not word in set(all_stopwords)]
    dataClean=' '.join(dataClean)
    corpus_Test.append(dataClean)
print(corpus_Test)

['americanair need refund', 'usairway cancel flightlat delay caus miss connect flight anoth night not home thank pathet', 'jetblu thar
```

Previously used Naïve Bayes classifer model is used for predicting the sentiments of the text of the fresh testing dataset.

```
[42] classifier = joblib.load("NaiveBayesModel")
     count_true = 0
     y_predictionTest = classifier.predict(X_fresh)
     y_predictionTest

     array(['neutral', 'negative', 'positive', ..., 'positive', 'neutral',
            'negative'], dtype='<U8')
```

```
y2= list(dataTest.iloc[:,-1].values)
y2

 'neutral',
 'positive',
 'positive',
 'negative',
 'negative',
 'negative',
 'negative',
 'neutral',
 'neutral',
 'negative',
 'negative',
 'negative',
 'negative',
 'negative',
 'negative',
 'negative',
 'negative',
 'neutral',
 'negative',
```

```
[ ] print(accuracy_score(y2,y_predictionTest))

     0.5478142076502732
```

The accuracy on the test dataset is calculated as 54.78%.

CODE 2 WITH SIMPLE NEURAL NETWORKS

ENVIRONMENT SET UP

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

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

```
[ ] pip install tensorflow
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tensorflow in /usr/local/lib/python3.8/dist-packages (2.9.2)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (14.0.6)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (3.1.0)
Requirement already satisfied: tensorflow-estimator<2.10.0,>=2.9.0rc0 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (2.9.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (4.4.0)
Requirement already satisfied: keras-preprocessing>=1.1.1 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (1.1.2)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (1.3.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (1.21.6)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (2.1.1)
Requirement already satisfied: keras<2.10.0,>=2.9.0rc0 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (2.9.0)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (0.28.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (1.51.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.8/dist-packages (from tensorflow) (21.3)
Requirement already satisfied: tensorboard<2.10,>=2.9 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (2.9.1)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (0.4.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (1.15.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.8/dist-packages (from tensorflow) (57.4.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.8/dist-packages (from tensorflow) (1.14.1)
```

All necessary libraries are imported below.

```
[ ] import numpy as np
    import pandas as pd
    import re
    import nltk
    nltk.download('stopwords')
    from nltk.corpus import stopwords
    stopwords_list = set(stopwords.words('english'))
    from numpy import array
    from numpy import asarray
    from numpy import zeros
    from imblearn.over_sampling import RandomOverSampler
    from imblearn.over_sampling import SMOTE
    from nltk.stem.porter import PorterStemmer
    ps = PorterStemmer()
```

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

```
[ ] from keras.preprocessing.text import one_hot, Tokenizer
    from keras_preprocessing.sequence import pad_sequences
    from keras.models import Sequential
    from keras.layers.core import Activation, Dropout, Dense
    from keras.layers import Flatten, GlobalMaxPooling1D, Embedding, Conv1D, LSTM
```

DATA LOADING

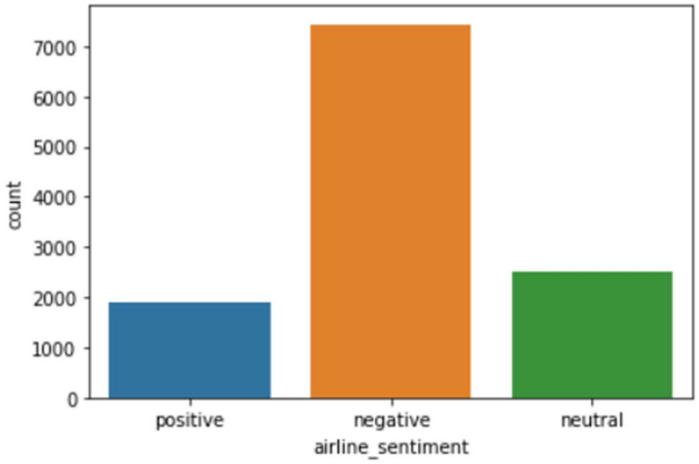
```
[ ] dataTrain = pd.read_csv("/content/drive/MyDrive/Sentiment Analysis Project final/Tweets_train.csv")
    print(dataTrain.head())
    dataValidate = pd.read_csv("/content/drive/MyDrive/Sentiment Analysis Project final/Tweets_dev.csv")
    print(dataValidate.shape)
```

```
             tweet_id                                               text  \
0  569179849518161920                  @united you're good. Thank you!
1  569835751275433984  @AmericanAir way to ruin a vacation, my brothe...
2  568588936852799488  @JetBlue yes thankfully! Catering just got her...
3  569525116725567491  @USAirways The automated message isn't helpful...
4  568807823187976193  @JetBlue I'm #MakingLoveOutofNothingAtAll on m...

  airline_sentiment
0          positive
1          negative
2          positive
3          negative
4          positive
(1318, 3)
```

VISUALIZING LOADED DATASET

```
[ ] import seaborn as sns
    sns.countplot(x='airline_sentiment', data = dataTrain)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f93d3fd2910>
```



DATA PRE-PROCESSING

Data pre-processing is done much similar to what has been done in code 1.

```
[ ]  TAG_RE = re.compile(r'<[^>]+>')
     def remove_tags(text):
         return TAG_RE.sub(' ',text)
```

```
[ ]  def preprocessing(sentence):
         statement = sentence.lower()
         statement = remove_tags(statement)
         statement = re.sub('[^a-zA-Z]',' ',statement)
         statement = re.sub(r'\s+[a-zA-Z]\s',' ',statement)
         statement = re.sub(r'\s+', ' ', statement)
         #pattern = re.compile(r'\b(' + r'|'.join(stopwords_list) + r')\b\s*')
         statement = [ps.stem(word) for word in statement if not word in set(stopwords_list)]
         statement = ''.join(statement)
         return statement
```

HTML tags, single characters, numbers, punctuations, multiple spaces, and stopwords are removed from the texts.

```
[ ]  corpus_Train = []
     sentences = list(dataTrain['text'])
     for x in sentences:
         corpus_Train.append(preprocessing(x))
     corpus_Train
```

```
[' une u re g hnk u ',
 ' ercnr w  run vcn  brher h clle ll ngh n h ulple plce n lne nl  ge e r n cll bck',
 ' jeblue e hnkfull cerng ju g here n nw he re lng bu ver frure w uppe  be here b ',
 ' urw he ue ege n helpful n  pble  pek wh hun rgh nw eperel nee ur luggge ',
 ' jeblue  knglveufnhngll n  brnlveffr  lx hp c khruf w',
 ' ercnr ju wnle he pp fr phne nce rnk cupn bu nhng  ple geng re  fl ',
 ' une n p lnk  n rcle h verfe he nl cpln wh cen ng he e hng hrr',
 ' jeblue  n ure f u cn  nhng  help e wh h bu f u cn h be gre ',
 ' ercnr  bggge  l  flgh cncelle flghle n ge n ccn n even  nher rlne  le ne free hel ',
 ' urw chrle ff nee prer ppne n repne  hp l f pe ff peple here ',
 ' urw ugge u flure ke huge nn  he u chrle nc  he prve gre cuer ervce  unlke u ',
 ' ercnr plee expln wh  c l he e prce f full runrp fr  nl chnge he eprure e f flgh ',
 ' uhwer  when ur plne en wrk he luxure p fr g w bu u keep  ne nee h bz el ',
'elee ur ccun jeblue',
 ' vrgnerc hve n nereng flng wh u fer h wll cncelle flgh  nex fur flgh plnne neverflvrgnfrbune',
 ' vrgnerc pple ver week g hven her bck n  flng h week ppne',
 ' ercnr he  ell h ur luggge e ne he plne lk h ve hp c kkhhqcp',
 ' uhwer ee lke u cul ke re ne b penng up e n re erble flgh ',
 ' uhwer ll re f we lre hve ur wnner cngr ',
 ' uhwer everhng k h   r cll fr he  n h e ve been n hl fr hr ll hng up n r gn ',
 ' ercnr h exr hur f rvel e e vcn e n nw u gu re eng wh  prfenl lfe ',
 ' une n ce u re reng h u rgh nw  n verze crrn kng up  uch r enfrce ur rule ',
 ' uhwer ju  l ngh hnk fr fllwng up ',
 ' urw l h n help e   eeng  wrk  eeng   preenn w n here jr rke gn e ',
```

```
corpus_Dev = []
sentences = list(dataValidate['text'])
for x in sentences:
    corpus_Dev.append(preprocessing(x))
corpus_Dev
```

```
 ' vrgnerc  ne f he rveler ffece b he bn r  hcke  vrgn cplee ph wr her cuer ',
 ' urw rke gn l bg n w f he hewr',
 ' une le le le ll ng  he ge hve n ve n nch hp c lulgnweffh',
 ' uhwer bg fl free  wh wh ll he crr n pckngk brnregncc',
 ' une ur helpful gen n club helpe  ju u bggge fee n ngh f  lfe p ng bune wh lverrw',
 ' urw w l  w fwre bug l  cll bck h  nw ur e  bu  hnle n cll h  clwn he ',
 ' urw rbprce hw cn he r gn  wll be hur befre he ge help cnngenc pln nn exen ',
 ' urw ve been wng fr ek fr n n hl n n cunng ph f ek gen n phne n flerng ',
 ' urw lre re hw bu cnf v ',
 ' uhwer  h ccun even wre f he ennrgn cne he ne ure fucknq up fr e ',
 ' urw clle he bv ek he h re nfrn  ex ur e n ke  e fr he cuer  cce  ',
 ' jeblue n he n hve re nf w re nfure b he w prevu rep ree e hw he cn pbl wrk  jeblue rep ',
 ' urw churn re ncreng b he ecn',
 ' uhwer he uhwe help e fn flgh fr nhvlle whngn c r rlegh fr uner n rch plee ',
 ' uhwer  flgh  kc h been cncelle flghe n he nex vlble n ll ue wh  uppe  fr ngh ',
 ' ercnr elvere  ele bg  he wrng plce hr n hl eer g hr  ll n bg  hp c qkkjfb',
 ' ercnr p enng e bck  cuer reln hve hn wren leer  he en n leer clle he n help ',
 ' urw ercnr expln hw ce cn ge free upgre n u  n exe pl bu n ex rw eng u png cnfue',
 ' une hul hve been ble  ge e gn whu png re cu ec hul hve ne h fr e',
 ' une ng  f vcn  ee  hubn becue f h el bu uff hppen n ge h ju wh  w hnle beer',
 ' ercnr u n repn  u weee he e cnne wee u re ellng everne ele ',
 ' ercnr e ju n h w prculrl prl hnle un ',
 ' jeblue ur fllerg cun f u wn ur cnnue ll u nee  beer hn h perhp e rnng  n ',
 ' une hnk fr ur help g e wh nee bu  n ue wh urn gue  whle lnger ',
 ' jeblue h c hnkng bu uer wh new nnucke ervce rkewch hp c  wguk r',
```

The class labels are converted to numerical forms:

1 for “positive”, 0 for “negative” and 2 for “neutral” sentiments

```
[ ]  sentiment_Train = dataTrain['airline_sentiment']
     sentiment_Train = np.array(list(map(lambda x:1 if x== "positive" else (0 if x == "negative" else 2), sentiment_Train)))
     print(sentiment_Train)
```

```
[1 0 1 ... 0 0 0]
```

```
[ ]  sentiment_validate = dataValidate['airline_sentiment']
     sentiment_validate = np.array(list(map(lambda x:1 if x== "positive" else (0 if x == "negative" else 2), sentiment_validate)))
     print(sentiment_validate)
```

```
[1 1 0 ... 0 0 1]
```

PREPARING EMBEDDING LAYER

The textual data in the datasets are already changed to numerical in the previous steps. The Embedding layer **takes this integer-encoded vocabulary and searches for the respective embedding vector for each word-index**. These vectors are learned as the model trains. The vectors add a dimension to the output array. The resulting dimensions are: (batch, sequence, embedding). – Certains portions taken from tensorflow.org

```
[ ]  word_tokenizer = Tokenizer()
     word_tokenizer.fit_on_texts(corpus_Train)
     word_tokenizer.fit_on_texts(corpus_Dev)
```

```
[ ]  X_train = word_tokenizer.texts_to_sequences(corpus_Train)
     X_dev = word_tokenizer.texts_to_sequences(corpus_Dev)
```

```
[ ]  vocab_length = len(word_tokenizer.word_index) +1
     print(vocab_length)
```

```
     7133
```

```
[ ]  max_len = 1100
     X_train = pad_sequences(X_train,padding = 'post', maxlen =max_len)
     X_dev = pad_sequences(X_dev,padding = 'post', maxlen =max_len)
     print(type(X_train))
```

```
     <class 'numpy.ndarray'>
```

For using pre-trained word embeddings, we load the GloVe file.

```
[ ]  embeddings_dictonary = dict()
     glove_file = open('/content/drive/MyDrive/Sentiment Analysis Project final/a2_glove.6B.100d.txt',encoding = "utf8")

     for line in glove_file:
         records = line.split()
         word = records[0]
         vector_dimensions = asarray(records[1:], dtype ='float32')
         embeddings_dictonary[word] = vector_dimensions
     glove_file.close()

     embedding_matrix = zeros((vocab_length,100))
     for word, index in word_tokenizer.word_index.items():
         embedding_vector = embeddings_dictonary.get(word)
         if embedding_vector is not None:
             embedding_matrix[index] = embedding_vector

     print(embedding_matrix.shape)
```

```
     (7133, 100)
```

MODEL TRAINING

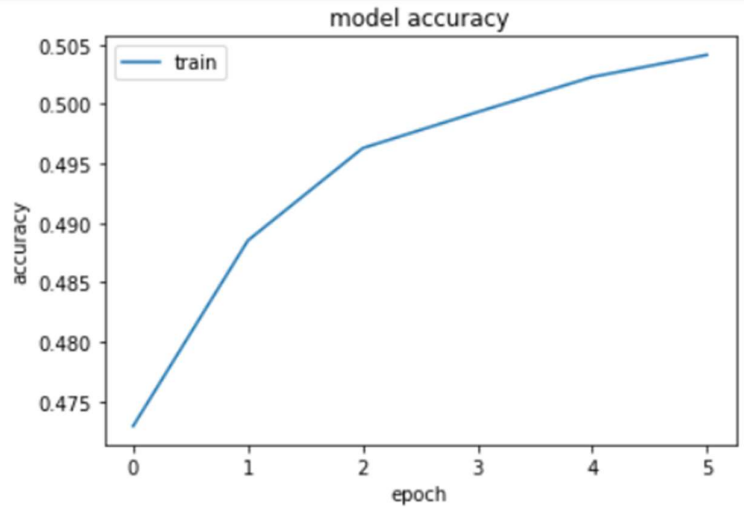
```
[ ]  snn = Sequential()
     embedding_layer = Embedding(vocab_length,100,weights = [embedding_matrix],input_length=max_len, trainable = False)
     snn.add(embedding_layer)
     snn.add(Flatten())
     snn.add(Dense(1,activation = "sigmoid"))

     snn.compile(optimizer= "adam" ,loss = "binary_crossentropy", metrics = ['acc'])
     print(snn.summary())

     snn_history = snn.fit(X_train,sentiment_Train,batch_size =128,epochs =6,verbose = 1)
     print(snn_history)

     score = snn.evaluate(X_dev,sentiment_validate,verbose=1)
     print("Score is:  ", score[0])
     print("Accuracy ", score[1])
```

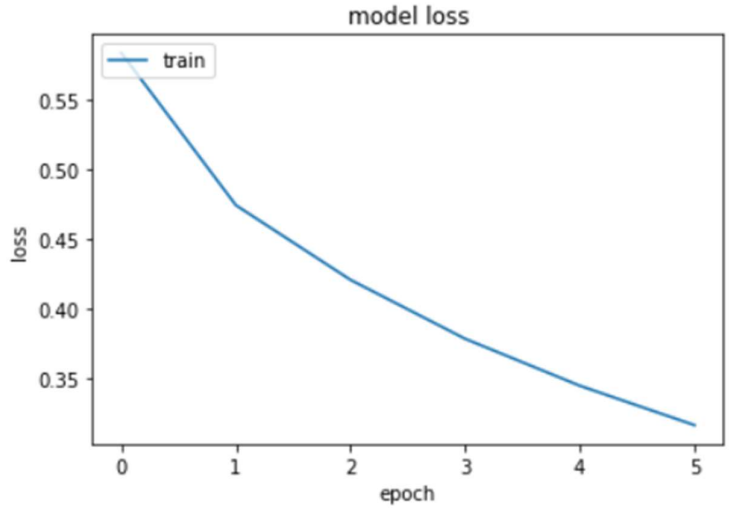
```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_8 (Embedding)     (None, 1100, 100)         713300

 flatten_5 (Flatten)         (None, 110000)            0

 dense_7 (Dense)             (None, 1)                 110001

=================================================================
Total params: 823,301
_________________________________________________________________
None
Epoch 1/6
93/93 [==============================] - 4s 37ms/step - loss: 0.5830 - acc: 0.4729
Epoch 2/6
93/93 [==============================] - 3s 36ms/step - loss: 0.4741 - acc: 0.4885
Epoch 3/6
93/93 [==============================] - 3s 36ms/step - loss: 0.4207 - acc: 0.4963
Epoch 4/6
93/93 [==============================] - 3s 33ms/step - loss: 0.3786 - acc: 0.4993
Epoch 5/6
93/93 [==============================] - 3s 33ms/step - loss: 0.3450 - acc: 0.5023
Epoch 6/6
93/93 [==============================] - 3s 37ms/step - loss: 0.3169 - acc: 0.5041
<keras.callbacks.History object at 0x7f93d4504df0>
42/42 [==============================] - 1s 10ms/step - loss: 0.4567 - acc: 0.5152
Score is:   0.45665842294692993
Accuracy  0.5151745080947876
```

The accuracy received in this model is 51.51%.

RESULTS PLOTTING

```
[ ]  import matplotlib.pyplot as plt

     plt.plot(snn_history.history['acc'])


     plt.title('model accuracy')
     plt.ylabel('accuracy')
     plt.xlabel('epoch')
     plt.legend(['train','test'], loc = 'upper left')
     plt.show()

     plt.plot(snn_history.history['loss'])


     plt.title('model loss')
     plt.ylabel('loss')
     plt.xlabel('epoch')
     plt.legend(['train','test'], loc = 'upper left')
     plt.show()
```





PREDICTION ON NEW TEST DATASETS

The test dataset is loaded and all the pre-processing steps previously done on the train and validation datasets are repeated on the test dataset.

```
[ ] dataTest = pd.read_csv("/content/drive/MyDrive/Sentiment Analysis Project final/New_Tweets_test-1.csv")
```

```
[ ] unseen_reviews = dataTest['text']

    unseen_processed = []
    for review in unseen_reviews:
        unseen_processed.append(preprocessing(review))
    unseen_processed
```

```
[' ercnr nee refun ',
 ' urw fer cncelle flghln n el cung e   cnnecng flgh nher ngh n beng he hnk phec',
 ' jeblue hnk  uch cn w  fl wh u gu ',
 ' une hve never been re frure hn  cnvern wh une wh cn pek pnh  he curer cpn h h  bg ',
 ' urw he wr hl e crz gen hrrble n ccunbl uruck',
 ' ercnr  pleure nex  flgh h wene  ln l fr exhbn ee u hen ',
 ' une frnkl wre cuer ervce ever prble wll hppen hw u el efne cpn never gn une ',
 ' uhwer u nee  ge ur c geher u new h rnng  ur plne w lfuncnng e ve been ele e ',
 ' ercnr n he cncelle flghe  flgh n lef e wh n help  fn hel   n lep n n rpr fr ngh ',
 ' jeblue flgh fr b  p keff n le flgh',
 ' une  r xweekl flgh fr newrlen  cncun n beween  ep vgeek',
 ' jeblue rw hr nere wn n jnur jblu leekne hp c znujp bv',
```

Now, previously prepared Simple neural networks ML models is used to predict the sentiments of the new dataset customer reviews.

```
[ ] sentiment_Test = snn.predict(X_test)
    X_test
```

```
46/46 [==============================] - 0s 10ms/step
array([[ 12,  61, 190, ...,   0,   0,   0],
       [ 11,  74,  36, ...,   0,   0,   0],
       [ 17,  20, 111, ...,   0,   0,   0],
       ...,
       [ 16,  39,   6, ...,   0,   0,   0],
       [ 12,   9, 333, ...,   0,   0,   0],
       [ 12,  52,  24, ...,   0,   0,   0]], dtype=int32)
```

The results are displayed below:

```
[ ] dataTest['Predicted airline_sentiment'] = sentiment_Test

    df_prediction_sentiments = pd.DataFrame(dataTest['Predicted airline_sentiment'], columns = ['Predicted airline_sentiment'])
    df_tweetid                = pd.DataFrame(dataTest['tweet_id'], columns = ['tweet_id'])
    df_text          = pd.DataFrame(dataTest['text'], columns = ['text'])
    df_airline_sentiment           = pd.DataFrame(dataTest['airline_sentiment'], columns = ['airline_sentiment'])


    dfx=pd.concat([df_tweetid, df_text,df_airline_sentiment, df_prediction_sentiments],axis =1)

    dfx.head(5)
```

	tweet_id	text	airline_sentiment	Predicted airline_sentiment
0	570252000000000000	@AmericanAir I need refund.	negative	neutral
1	568173000000000000	@USAirways after 3 Cancelled Flightlations and...	negative	neutral
2	569321000000000000	@JetBlue thanks so much. Can't wait to fly wit...	positive	neutral
3	569503000000000000	@united I have never been more frustrated than...	negative	neutral
4	568981000000000000	@USAirways - the worst! Hold time crazy, agent...	negative	neutral

The results will be better had the airline _sentiments in the training dataset been more equally distributed. Data augmentation is hard to implement on texts. Smote was tried but there were issues as well. Had more methods, for equal distribution been tried, accuracy would have improved.

Other than SNN model, codes for CNN model and LSTM model are also uploaded in NESS.

In both project 1 & 2, certain helps in the code, report explanation are taken from the below mentioned websites:

Wikipedia

https://machinelearningmastery.com/

https://scikit-learn.org/

tensorflow.org,etc.

THANK YOU!