BIGDATA COURSEWORK-INDIVIDUAL ASSIGNMENT

PREDICTIVE MODELLING TO ASSESS THE SEVERITY OF ACCIDENTS

TABLE OF CONTENTS

1. INTRODUCTION : BUSINESS OBJECTIVE AND ITS CONTEXT

Emerging risks are harming businesses more than ever in a world, that is changing quickly. When it comes to managing risks, organisations need to take a comprehensive strategy since it may help them achieve long-term success. **Utilizing data and technology** to bring risk to the forefront of decision-making processes is essential.

Car insurance businesses work in the risk industry. Insurance claims are significant for the insurance company. For insurers, high-risk drivers pose the greatest financial hazards. The risks are collectively determined by experience, age,vehicle age, regulations, etc. **Accident Severity** is one of the main factors in determining the risks.

ML is a key tool in loss prediction and risk management because it can quickly identify possibly anomalous or unexpected activities using data and algorithms.

This project's aim is to create a prediction model that a car insurance provider may use to assess the severity of accidents that a car driver who is seeking for insurance is likely to be involved in. The auto insurance provider will then use that risk to calculate the premium and coverages for that particular driver based on his driving patterns and behaviours.

2.DATASET PREPARATION

Here, the libraries will be imported and the dataframes which have been created in the group part will be uploaded.

2.1 IMPORTING LIBRARIES

In [ ]:
```
#Importing libraries for data loading and data manipulation
# Preparing the environment

import time
import numpy as np
import pandas as pd
```

```
#Library for Plotting
import seaborn as sns
sns.set(style="darkgrid")
import matplotlib.pyplot as plt
%matplotlib inline
```

2.2 LOADING THE DATAFRAMES

At the end of group part, 2 dataframes; 1 for Training Set and one for Testing set was created and were exported into csv files.

In [ ]:
```
# loading the csv files
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving testset.csv to testset.csv
Saving trainset.csv to trainset.csv
```

In [ ]:
```
# loading the 2 files in dataframes.
import io
# loading training set
dataframe1 = io.BytesIO(uploaded['trainset.csv'])
#dropping the index column
trainset= pd.read_csv(dataframe1 , index_col=0)

# loading testing set
dataframe2 = io.BytesIO(uploaded['testset.csv'])
# dropping the index column
testset= pd.read_csv(dataframe2 , index_col=0)
```

2.3 VALIDATING THE 2 DATAFRAMES

The training set and testing set would be validated by checking their shapes, data types and the first 10 rows.

In [ ]:
```
# checking the shape of the training set
trainset.shape
```

Out[ ]:
```
(13643, 11)
```

The training set has 13643 rows and 11 columns.

In [ ]:
```
# checking the datatypes of the training set
trainset.dtypes
```

Out[ ]:
```
accident_severity                 object
age_of_driver                    float64
engine_capacity_cc               float64
age_of_vehicle                   float64
urban_or_rural_area_Urban        float64
vehicle_left_hand_drive_Yes      float64
sex_of_driver_Male               float64
propulsion_code_Heavy oil        float64
propulsion_code_Hybrid electric  float64
propulsion_code_Petrol           float64
driver_home_area_type_Urban area float64
dtype: object
```

There are **3 Numerical variables age_of_driver,age_of_vehicle & engine_capacity_cc . The categorical variables are urban_or_rural_area, vehicle_left_hand_drive,sex_of_driver,propulsion_code and driver_home_area_type.** All the categorical variables have been converted to dummy variables.

In [ ]:
```
# checking the first 10 rows of the training set
trainset.head(10)
```

Out[ ]:

	accident_severity	age_of_driver	engine_capacity_cc	age_of_vehicle	urban_or_rural_area_Urban	vehicle_left_hand_drive_Yes	sex_of_driver_Male	pr
44923	Slight	42.0	1360.0	10.0	1.0	0.0	0.0	
110490	Slight	74.0	1200.0	4.0	1.0	0.0	0.0	
56293	Slight	50.0	1560.0	10.0	1.0	0.0	1.0	
127729	Slight	18.0	1870.0	18.0	1.0	0.0	1.0	
42831	Slight	33.0	1248.0	5.0	0.0	0.0	0.0	
58762	Slight	65.0	1995.0	1.0	1.0	0.0	1.0	
152658	Slight	41.0	1998.0	16.0	0.0	0.0	1.0	
147432	Serious	52.0	2400.0	11.0	0.0	0.0	1.0	
43302	Slight	25.0	998.0	11.0	0.0	0.0	0.0	
156490	Slight	68.0	1582.0	5.0	1.0	0.0	1.0	

In [ ]:
```
# validating the testing set
testset.shape
```

Out[ ]:
```
(3414, 11)
```

In [ ]:
```
# checking the datatypes of the testing set
testset.dtypes
```

Out[ ]:
```
accident_severity                   object
age_of_driver                      float64
engine_capacity_cc                 float64
age_of_vehicle                     float64
urban_or_rural_area_Urban          float64
vehicle_left_hand_drive_Yes        float64
sex_of_driver_Male                 float64
propulsion_code_Heavy oil          float64
propulsion_code_Hybrid electric    float64
propulsion_code_Petrol             float64
driver_home_area_type_Urban area   float64
dtype: object
```

In [ ]:
```
# checking the first 10 rows of the testing set
testset.head(10)
```

Out[ ]:

	accident_severity	age_of_driver	engine_capacity_cc	age_of_vehicle	urban_or_rural_area_Urban	vehicle_left_hand_drive_Yes	sex_of_driver_Male	pr
19933	Slight	17.0	1198.0	11.0	1.0	0.0	1.0	
51085	Slight	60.0	1799.0	15.0	1.0	0.0	1.0	
41521	Slight	24.0	998.0	12.0	0.0	0.0	1.0	
64094	Slight	51.0	1984.0	2.0	0.0	0.0	1.0	
84495	Slight	36.0	2204.0	11.0	1.0	0.0	0.0	
63699	Slight	44.0	1598.0	14.0	1.0	0.0	1.0	
135637	Slight	49.0	1798.0	5.0	1.0	0.0	1.0	
93361	Slight	57.0	1560.0	4.0	0.0	0.0	1.0	
7470	Serious	32.0	1339.0	13.0	1.0	0.0	0.0	
116727	Slight	33.0	1086.0	17.0	1.0	0.0	0.0	

3.FEATURE ENGINEERING

It consists of Feature Selection, Feature Transformation and Feature Scaling.

Feature Selection means selecting the important features based on their relationship with the target variable.

Feature transformation will be done to remove the skewness of the data

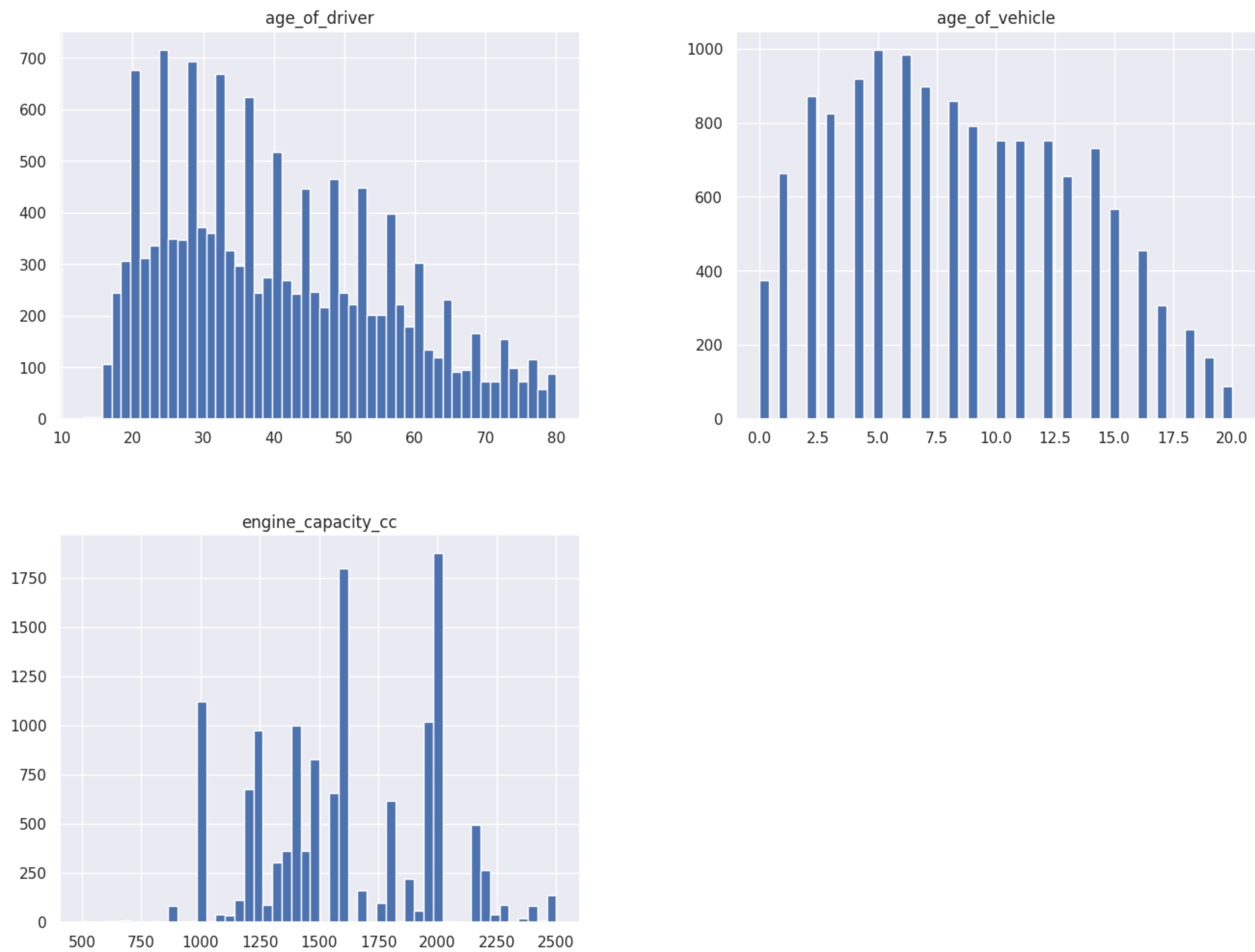
As,it was observed during the EDA that **there was a difference between the scales of age of car, age of driver and engine capacity, so Feature Scaling will be carried out to normalize the range of predictors in both the training and testing set .**

3.1 FEATURE SELECTION.

This step was carried out in the **group part where the variable "Driver_IMd Decile" was dropped from both the training and testing sets .** It was dropped on the basis of **Chi Square test of Independence which highlighted the independence of this variable and the target variable.**

3.2 INSPECTING THE DISTRIBUTION OF NUMERICAL VARIABLES

In [ ]:
```
# selecting only the numerical variables
attributes = ["age_of_driver","age_of_vehicle","engine_capacity_cc"]

# plotting the histogram to gain insight about the numerical variables.
dummy = (trainset[attributes].hist(bins=50, figsize=(16,12)))
```



Here, we can observe some measure of skewness in all the three variables. The measures of skewness will be calculated for all the three variables.

In [ ]:
```
# calculating the skewness measures
trainset[attributes].agg(['skew']).transpose()
```

Out[ ]:

	skew
age_of_driver	0.517375
age_of_vehicle	0.265421
engine_capacity_cc	0.102670

SKEWNESS MEASURES

Fairly Symmetrical -0.5 to 0.5

Moderate Skewed -0.5 to -1.0 and 0.5 to 1.0

Highly Skewed < -1.0 and > 1.0

On comparing the skewness measures of the variables with the above range, it can be observed that the **variable, age_of_driver is moderately skewed** whereas the rest 2 are **fairly symmetrical.**

3.3 FEATURE TRANSFORMATION(LOG TRANSFORMATION)

Here, it can be observed that all the **numerical variables** are **skewed. Thus,the variables will be transformed using log transformation.**

All the transformations done on the training set will be done on the testing set as well, because the training set should be a TRUE REPRESENATIVE of the testing set.

In [ ]:
```
# transforming the numerical variables in the training set using log transformation to remove skewness
for x in ["age_of_driver","age_of_vehicle","engine_capacity_cc"]:
    # add 1 so that 0s remain 0s
    trainset[x] = np.log(trainset[x] + 1)

# doing the same transformations on the testing set
    testset[x] = np.log(testset[x] + 1)
```

3.4 FEATURE SCALING

All the numerical variables have very different scales (e.g., age of car ranges between 0 and 20, while engine capacity value between 500 and 2500).**The variables will be scaled to be within the same range.**

The scaling will be done on the training set and then it will be used to transform both the training set and the test set by fitting and transforming a standard scaler.

In [ ]:
```
# fitting and transforming a standard scaler
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# take the target variable out before scaling
trainset_target = trainset["accident_severity"].values
trainset_predictors = trainset.drop("accident_severity", axis=1)

# fit_transform returns a NumPy array, so we need to put it back
# into a Pandas dataframe
scaled_vals = scaler.fit_transform(trainset_predictors)
trainset = pd.DataFrame(scaled_vals, columns=trainset_predictors.columns)

# put the non-scaled target back in
trainset['accident_severity'] = trainset_target

# inspect the data
trainset.head()
```

Out[ ]:

	age_of_driver	engine_capacity_cc	age_of_vehicle	urban_or_rural_area_Urban	vehicle_left_hand_drive_Yes	sex_of_driver_Male	propulsion_code_Heavy oil	pı
0	0.278658	-0.583338	0.519309	0.764561	-0.094989	-1.281583	-0.868750	
1	1.731647	-1.113439	-0.626453	0.764561	-0.094989	-1.281583	-0.868750	
2	0.724321	-0.002197	0.519309	0.764561	-0.094989	0.780285	1.151079	
3	-1.854672	0.765609	1.313529	0.764561	-0.094989	0.780285	1.151079	
4	-0.334729	-0.947334	-0.361509	-1.307941	-0.094989	-1.281583	-0.868750	

The **test data** will also be transformed using the **fitted scaler**.

In [ ]:
```
# transforming the test data
testset_target = testset["accident_severity"].values
testset_predictors = testset.drop("accident_severity", axis=1)

scaled_vals = scaler.transform(testset_predictors)
testset = pd.DataFrame(scaled_vals, columns=testset_predictors.columns)

# put the non-scaled target back in
testset['accident_severity'] = testset_target

testset.head()
```

Out[ ]:

	age_of_driver	engine_capacity_cc	age_of_vehicle	urban_or_rural_area_Urban	vehicle_left_hand_drive_Yes	sex_of_driver_Male	propulsion_code_Heavy oil	pı
0	-1.995892	-1.120504	0.645751	0.764561	-0.094989	0.780285	-0.868750	
1	1.191984	0.601633	1.063802	0.764561	-0.094989	0.780285	-0.868750	
2	-1.137860	-1.893998	0.762067	-1.307941	-0.094989	0.780285	-0.868750	
3	0.775040	1.016305	-1.368768	-1.307941	-0.094989	0.780285	-0.868750	
4	-0.113870	1.461818	0.645751	0.764561	-0.094989	-1.281583	1.151079	

All the required preprocessing steps have been taken and now the model building process can start.

4.TARGET VARIABLE ASSESSMENT AND TREATMENT

The target variable is **Accident Severity** which is a categorical variable and at the time of EDA we observed that the distribution of the observations in the different categories is not same.

In [ ]:
```
# counting the number of instances belonging to each class
trainset["accident_severity"].value_counts()
```

Out[ ]:
```
Slight     10867
Serious     2606
Fatal        170
Name: accident_severity, dtype: int64
```

The number of observations in the "SLIGHT" category is quite high as compared to the other 2 categories, so it will be sensible to merge Serious and Fatal.

In [ ]:
```
# merging Serious and Fatal by replacing Fatal with Serious
trainset['accident_severity'] = trainset['accident_severity'].replace({'Fatal': 'Serious'})
testset['accident_severity'] = testset['accident_severity'].replace({'Fatal': 'Serious'})

# validating the change on the trainset
trainset['accident_severity'].value_counts()
```

Out[ ]:
```
Slight     10867
Serious     2776
Name: accident_severity, dtype: int64
```

In [ ]:
```
# validating the change on the testset
testset['accident_severity'].value_counts()
```

Out[ ]:
```
Slight     2728
Serious     686
Name: accident_severity, dtype: int64
```

5. MODEL BUILDING

In this part,differents models will be created and trained on the training set and after comparing their results, the best ones will be picked up to evaluate on the test set.

5.0(a) CREATING SEPARATE ARRAYS

Here, separate arrays for the predictors and targets will be created for both the training and testing set.

In [ ]:
```
# creating separate arrays for the predictors (Xtrain) and for the target (ytrain)in training set.
# drop labels for training set, but keep all others
Xtrain = trainset.drop("accident_severity", axis=1)
ytrain = trainset["accident_severity"].copy()

# validating the dataframes
print(f"The shape of Xtrain is {Xtrain.shape} and the shape of ytrain is {ytrain.shape}")
```

```
The shape of Xtrain is (13643, 10) and the shape of ytrain is (13643,)
```

In [ ]:
```
# creating separate arrays for the predictors (Xtest) and for the target (ytest)in testing set.
# drop labels for testing set, but keep all others
Xtest = testset.drop("accident_severity", axis=1)
ytest = testset["accident_severity"].copy()

# validating the dataframes
print(f"The shape of Xtest is {Xtest.shape} and the shape of ytest is {ytest.shape}")
```

```
The shape of Xtest is (3414, 10) and the shape of ytest is (3414,)
```

5.0(b) CROSS VALIDATION AND HYPERPARAMETER TUNING

CROSS VALIDATION: Here, we will be using the cross validation technique to provide an insight into how well a model will perform on new and unseen dataset.It will also help in understanding that **whether the model is overfitting the data or not**. **k-1 cross validation will be used .**

HYPERPARAMETER TUNING: A hyperparameter is a parameter whose value is chosen before training the models. Changing the values of hyperparameters to gain the optimal results is known as hyperparameter tuning. Here,both **Exhaustive Grid Search** and **Randomized Grid Search** will be used for **hyperparameter tuning.**

5.1 BASELINE MODEL

A **majority class classifier(MODE)** will be considered as a baseline, i.e., the most common class label in the training set will be found out and will always be taken as an output as a prediction.

In [ ]:
```
# counting the number of instances belonging to each class
trainset["accident_severity"].value_counts()
```

Out[ ]:
```
Slight     10867
Serious     2776
Name: accident_severity, dtype: int64
```

In [ ]:
```
# total size of the training set
trainset.shape[0]
```

Out[ ]:
```
13643
```

It is very important to note that there is a very **high class imbalance** which was also noticed in the initial steps of our creating the project. It will be dealt with after comparing the performance of the models created on unbalanced data and the balanced data

The baseline classifier will output **SLIGHT** for all predictions. We will use macro-averaging in this project (precision, recall and F-score are evaluated in each class separately and then averaged across classes).

In [ ]:
```
# importing the required libraries

from sklearn.metrics import precision_recall_fscore_support, classification_report
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.model_selection import cross_val_predict
```

In [ ]:
```
# creating the necessary functions to evaluate the performance
def evaluate_model(model, ytest, Xtest):
    """Given a trained model and test data, generate predictions
    and print a report with evaluation results
    """
    yhat = model.predict(Xtest)
    print(classification_report(ytest, yhat, zero_division=0))


def print_cv_results(grid_search, col_width=100, max_rows=10):
    """Given a grid search object, print a table with the
    cross-validation results
    """
    results = pd.DataFrame(grid_search.cv_results_
                          )[['params', 'mean_train_score', 'mean_test_score']]
    results["diff, %"] = 100*(results["mean_train_score"]-results["mean_test_score"]
                                                     )/results["mean_train_score"]

    pd.set_option('display.max_colwidth', col_width)
    pd.set_option('display.min_rows', max_rows)
    pd.set_option('display.max_rows', max_rows)
    display(results.sort_values('mean_test_score', ascending=False))
```

In [ ]:
```
from sklearn.dummy import DummyClassifier

# creating the BASELINE MODEL by considering the mode of the target variable as the predicted value
dummy_clf = DummyClassifier(strategy="most_frequent")
dummy_clf.fit(Xtrain, ytrain)
yhat_train = dummy_clf.predict(Xtrain)

# evaluating the Baseline Model's performance on training set
evaluate_model(dummy_clf, ytrain, Xtrain)
```

```
              precision    recall  f1-score   support

     Serious       0.00      0.00      0.00      2776
      Slight       0.80      1.00      0.89     10867

    accuracy                           0.80     13643
   macro avg       0.40      0.50      0.44     13643
weighted avg       0.63      0.80      0.71     13643
```

The **f score for the Baseline Model is 0.44** ,which is quite good for a baseline model. **The performance of the Baseline Model for the data with high class imbalance is fairly high.

Thus, a more complex model **"RANDOM FOREST"** will be built to check the improvement of the performance from the baseline model.

5.2 RANDOM FORESTS

Random Forests work by training many Decision Trees on random subsets of the features, then averaging out their predictions.

In [ ]:
```
# building a random forest model using GRID SEARCH for HYPERPARAMETER TUNING
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# specify the hyperparameters and their values
# 3 x 2  = 6 combinations in the grid
param_grid = [
    {'n_estimators': [100, 200, 500],
     'max_depth': [5, None]},
]

forest = RandomForestClassifier(random_state=7)

# using a 10-fold cross-validation
rf_grid_search = GridSearchCV(forest, param_grid, cv=10,
                              scoring='f1_macro',
                              return_train_score=True)

rf_grid_search.fit(Xtrain, ytrain)
```

Out[ ]:

```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

In [ ]:

```
# printing training and validation RMSE
print_cv_results(rf_grid_search, col_width=100)
```

	params	mean_train_score	mean_test_score	diff, %
3	{'max_depth': None, 'n_estimators': 100}	0.975014	0.483814	50.378775
5	{'max_depth': None, 'n_estimators': 500}	0.974893	0.480972	50.664110
4	{'max_depth': None, 'n_estimators': 200}	0.974941	0.480824	50.681687
0	{'max_depth': 5, 'n_estimators': 100}	0.443454	0.443370	0.018948
1	{'max_depth': 5, 'n_estimators': 200}	0.443370	0.443370	0.000001
2	{'max_depth': 5, 'n_estimators': 500}	0.443412	0.443370	0.009479

The F-score of the Baseline and Random Forest Model are nearly equal.

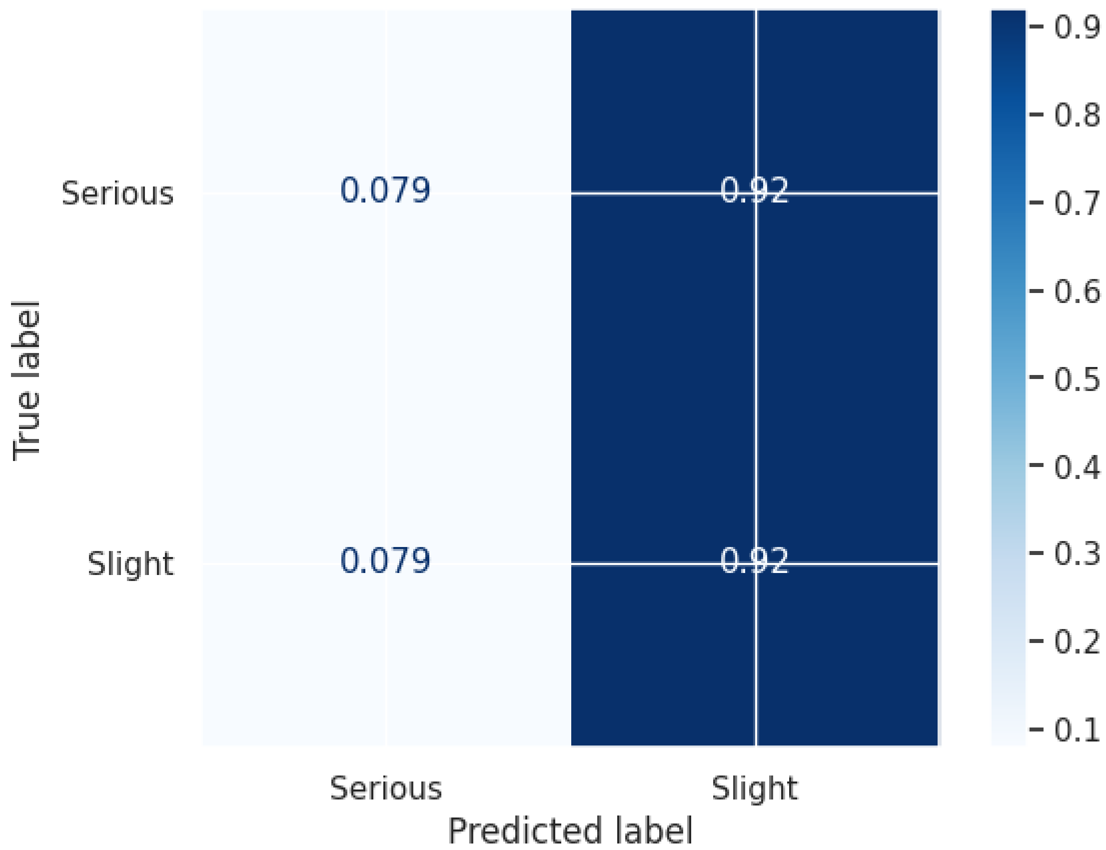
Also, the model has overfitted the data where the depth of the model is unconstrained.

A confusion matrix will be built to see what errors the model has made.

In [ ]:

```
# cross-validation confusion matrix, training data
yhat = cross_val_predict(rf_grid_search.best_estimator_, Xtrain, ytrain, cv=10)
ConfusionMatrixDisplay.from_predictions(ytrain, yhat,
                                        labels=rf_grid_search.best_estimator_.classes_,
                                        normalize="true",
                                        cmap=plt.cm.Blues)
```

Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff618d72d00>

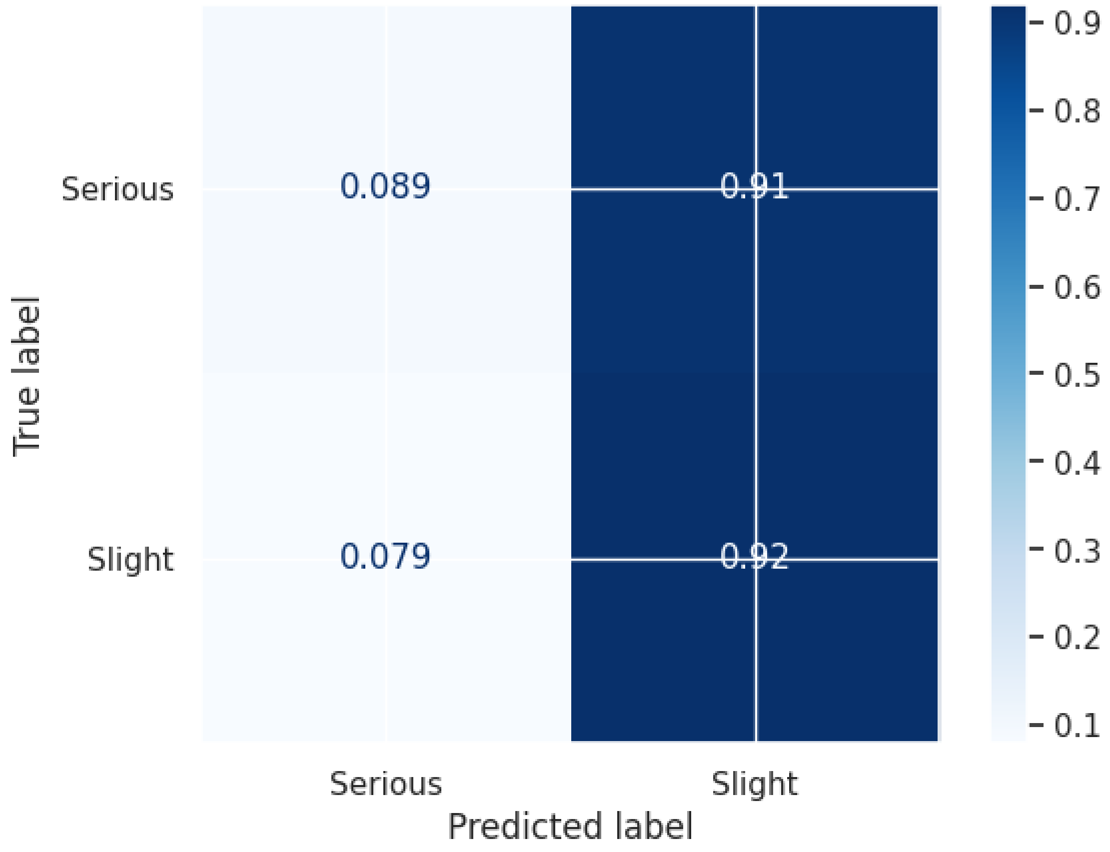


Here from the confusion matrix we can observe that the model has assigned **around 92% of all test instances - of both the classes to the "SLIGHT CLASS"**. The confusion matrix on the testing set will also be obtained to gain more insights.

In [ ]:

```
# Confusion matrix on the testing set
ConfusionMatrixDisplay.from_estimator(rf_grid_search.best_estimator_, Xtest, ytest,
                                  display_labels=rf_grid_search.best_estimator_.classes_,
                                  cmap=plt.cm.Blues,
                                  normalize='true')
```

Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff618d722e0>



Almost similar results can be observed from the testing set also. **This is definitely a consequence of the training data being very imbalanced.** Before moving any further we will first deal with imbalance in the data and then proceed with building rest of the models.

DEALING WITH CLASS IMBALANCE

Here, we will use three techniques to balance the classes and choose the best out of them.

1. RANDOM OVERSAMPLING
2. SMOTE
3. RANDOM UNDERSAMPLING

Here, we will use Exhaustive Grid Search for doing hyperparameter tuning for Random Forests **. While using these Sampling techniques we can also perform Hyperparameter Tuning for Sampling Strategy. Thus, we will be using Pipeline Class of Imblearn which arranges the different pre-processing steps in a pipeline.**

1. RANDOM OVERSAMPLING.

The copies of instances of the minority class will be added to the training data, with replacement.

In [ ]:
```
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import RandomOverSampler

# arranging the different processing steps in a pipeline
pipeline = Pipeline([
        ('ros', RandomOverSampler(random_state=7)),
        ('rfc', RandomForestClassifier(random_state=7))
    ])

# performing the hyperparameter tuning to find the optimal results.
# sampling strategy : degree of oversampling the minority class
# sampling strategy 0.5 : the instances in minority class will be half that of the majority class
# sampling strategy 1 : the instances in the minority class will be equal to the majority class.
# specifying the hyperparameters and their values
# 3 x 3 x 2  = 18 combinations in the grid
param_grid = [
    {
        'ros__sampling_strategy': [0.5, 0.75, 1.0],
        'rfc__n_estimators': [100, 200, 500],
        'rfc__max_depth': [5, None]
    },
]

# using a 10 fold cross validation
ros_grid_search = GridSearchCV(pipeline, param_grid, cv=10,
                               scoring='f1_macro',
                               return_train_score=True)

# fitting the model on the training set
ros_grid_search.fit(Xtrain, ytrain)
```

Out[ ]:

```
GridSearchCV
estimator: Pipeline
RandomOverSampler
RandomForestClassifier
```

In [ ]:

```
# printing the cv scores
print_cv_results(ros_grid_search, col_width=100)
```

	params	mean_train_score	mean_test_score	diff, %
5	{'rfc__max_depth': 5, 'rfc__n_estimators': 200, 'ros__sampling_strategy': 1.0}	0.542741	0.509815	6.066463
8	{'rfc__max_depth': 5, 'rfc__n_estimators': 500, 'ros__sampling_strategy': 1.0}	0.542782	0.509011	6.221794
2	{'rfc__max_depth': 5, 'rfc__n_estimators': 100, 'ros__sampling_strategy': 1.0}	0.541309	0.508733	6.018022
11	{'rfc__max_depth': None, 'rfc__n_estimators': 100, 'ros__sampling_strategy': 1.0}	0.974604	0.501465	48.546841
1	{'rfc__max_depth': 5, 'rfc__n_estimators': 100, 'ros__sampling_strategy': 0.75}	0.528353	0.501355	5.109736
...	...	...	...	...
15	{'rfc__max_depth': None, 'rfc__n_estimators': 500, 'ros__sampling_strategy': 0.5}	0.975222	0.496504	49.088092
9	{'rfc__max_depth': None, 'rfc__n_estimators': 100, 'ros__sampling_strategy': 0.5}	0.975225	0.495023	49.240129
0	{'rfc__max_depth': 5, 'rfc__n_estimators': 100, 'ros__sampling_strategy': 0.5}	0.453971	0.449330	1.022416
6	{'rfc__max_depth': 5, 'rfc__n_estimators': 500, 'ros__sampling_strategy': 0.5}	0.453340	0.448261	1.120237
3	{'rfc__max_depth': 5, 'rfc__n_estimators': 200, 'ros__sampling_strategy': 0.5}	0.452609	0.447872	1.046525

18 rows × 4 columns

In [ ]:

```
ros_grid_search.best_score_
```

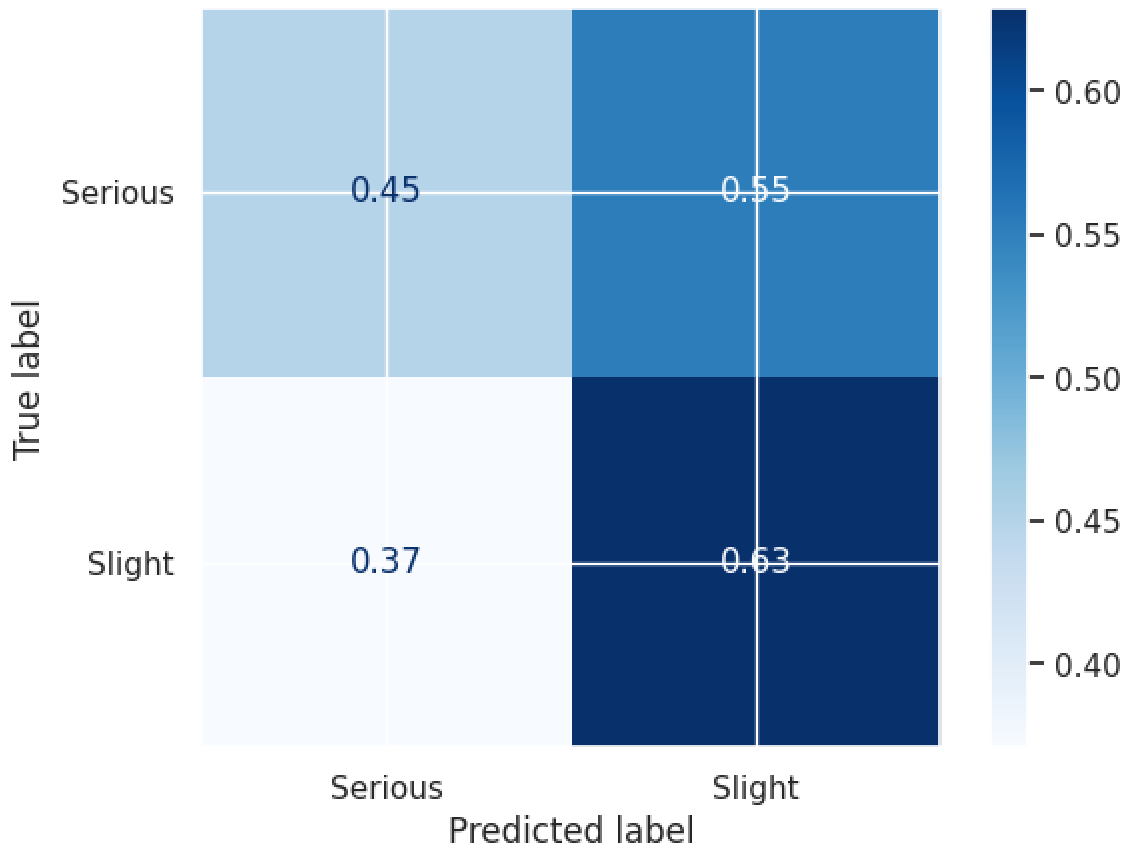
Out[ ]: 0.5098153627145449

Random oversampling produces a cross-validation F-score of 0.509, an improvement on the classifier trained on unbalanced data, which has the F-score of 0.44.

In [ ]:

```
# cross-validation confusion matrix on the training data
yhat = cross_val_predict(ros_grid_search.best_estimator_, Xtrain, ytrain, cv=10)

ConfusionMatrixDisplay.from_predictions(ytrain, yhat,
                                        labels=ros_grid_search.best_estimator_.classes_,
                                        normalize="true",
                                        cmap=plt.cm.Blues)
```

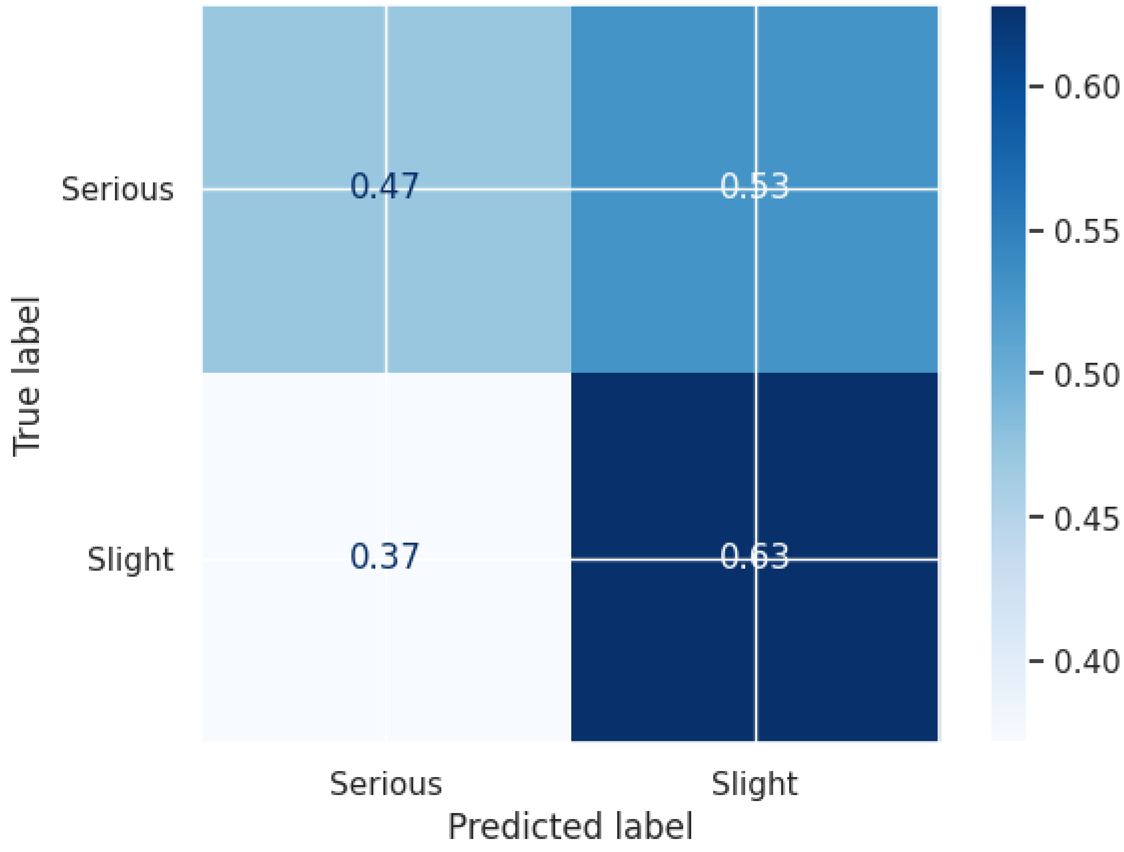
Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff61891df70>



In [ ]:

```
# confusion matrix on the test data
ConfusionMatrixDisplay.from_estimator(ros_grid_search.best_estimator_, Xtest, ytest,
                                      display_labels=ros_grid_search.best_estimator_.classes_,
                                      cmap=plt.cm.Blues,
                                      normalize='true')
```

Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff61892ae20>



47% of the minority class **(SERIOUS)** instances are correctly classified, opposed to just 8% which was obtained by Random Forest Model built on the unbalanced data.

The best minority-majority balance proves to be 1.0 implying that there are same instances of the majority and minority class.

2.SMOTE

SMOTE oversamples the minority class by generating synthetic training data of a minority class.

In [ ]:
```
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE

pipeline = Pipeline([
        ('smote', SMOTE(random_state=7)),
        ('rfc', RandomForestClassifier(random_state=7))
    ])

# specifying the hyperparameters and their values
param_grid = [
    {
        'smote__sampling_strategy': [0.5, 0.75, 1.0],
        'rfc__n_estimators': [100, 200, 500],
        'rfc__max_depth': [5, None]
    },
]

os_grid_search = GridSearchCV(pipeline, param_grid, cv=10,
                              scoring='f1_macro',
                              return_train_score=True)

os_grid_search.fit(Xtrain, ytrain)
```

Out[ ]:
```
▸ GridSearchCV
▸ estimator: Pipeline
    ▸ SMOTE
▸ RandomForestClassifier
```

In [ ]:
```
# printing the cv test scores
print_cv_results(os_grid_search, col_width=100)
```

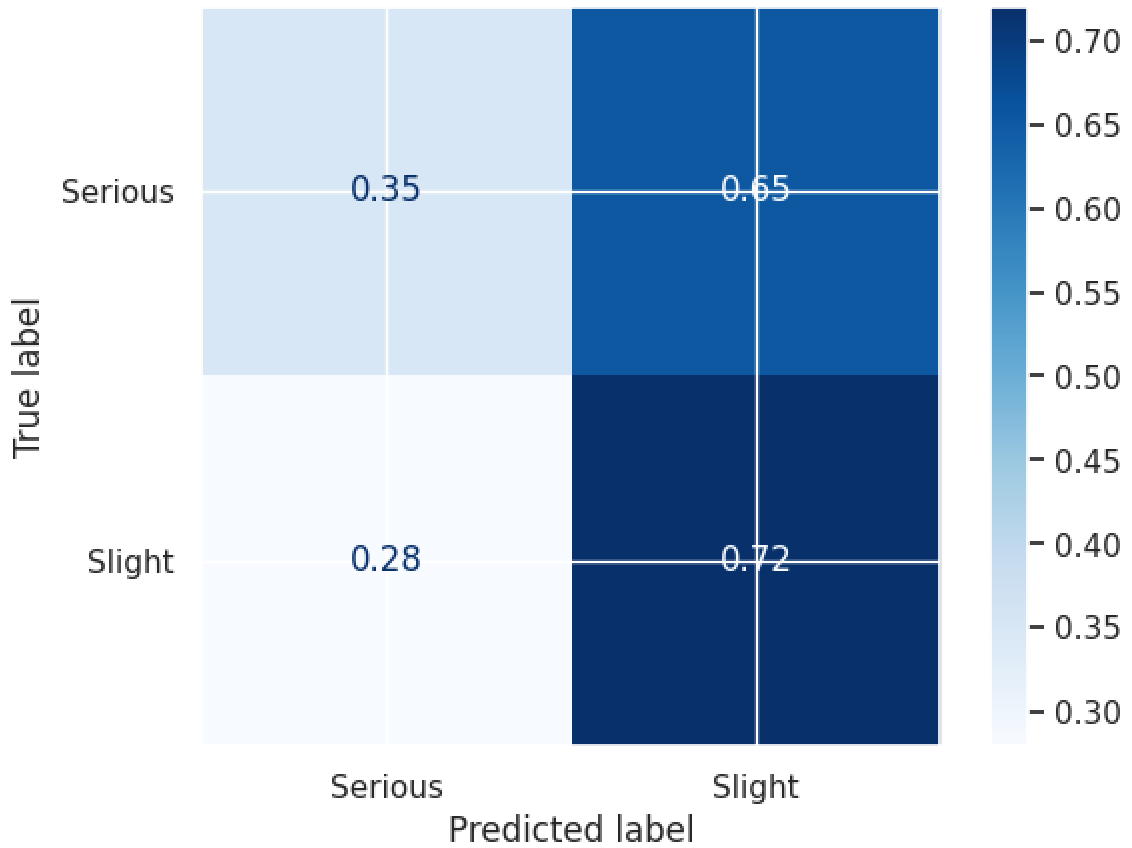
	params	mean_train_score	mean_test_score	diff, %
8	{'rfc__max_depth': 5, 'rfc__n_estimators': 500, 'smote__sampling_strategy': 1.0}	0.549141	0.523709	4.631247
5	{'rfc__max_depth': 5, 'rfc__n_estimators': 200, 'smote__sampling_strategy': 1.0}	0.549149	0.521296	5.072036
2	{'rfc__max_depth': 5, 'rfc__n_estimators': 100, 'smote__sampling_strategy': 1.0}	0.548038	0.519812	5.150411
14	{'rfc__max_depth': None, 'rfc__n_estimators': 200, 'smote__sampling_strategy': 1.0}	0.975248	0.503987	48.322182
17	{'rfc__max_depth': None, 'rfc__n_estimators': 500, 'smote__sampling_strategy': 1.0}	0.975231	0.503451	48.376167
...	...	...	...	...
4	{'rfc__max_depth': 5, 'rfc__n_estimators': 200, 'smote__sampling_strategy': 0.75}	0.502735	0.486480	3.233140
7	{'rfc__max_depth': 5, 'rfc__n_estimators': 500, 'smote__sampling_strategy': 0.75}	0.502748	0.485743	3.382445
3	{'rfc__max_depth': 5, 'rfc__n_estimators': 200, 'smote__sampling_strategy': 0.5}	0.443578	0.443370	0.046768
0	{'rfc__max_depth': 5, 'rfc__n_estimators': 100, 'smote__sampling_strategy': 0.5}	0.443829	0.443370	0.103452
6	{'rfc__max_depth': 5, 'rfc__n_estimators': 500, 'smote__sampling_strategy': 0.5}	0.443619	0.443347	0.061350

18 rows × 4 columns

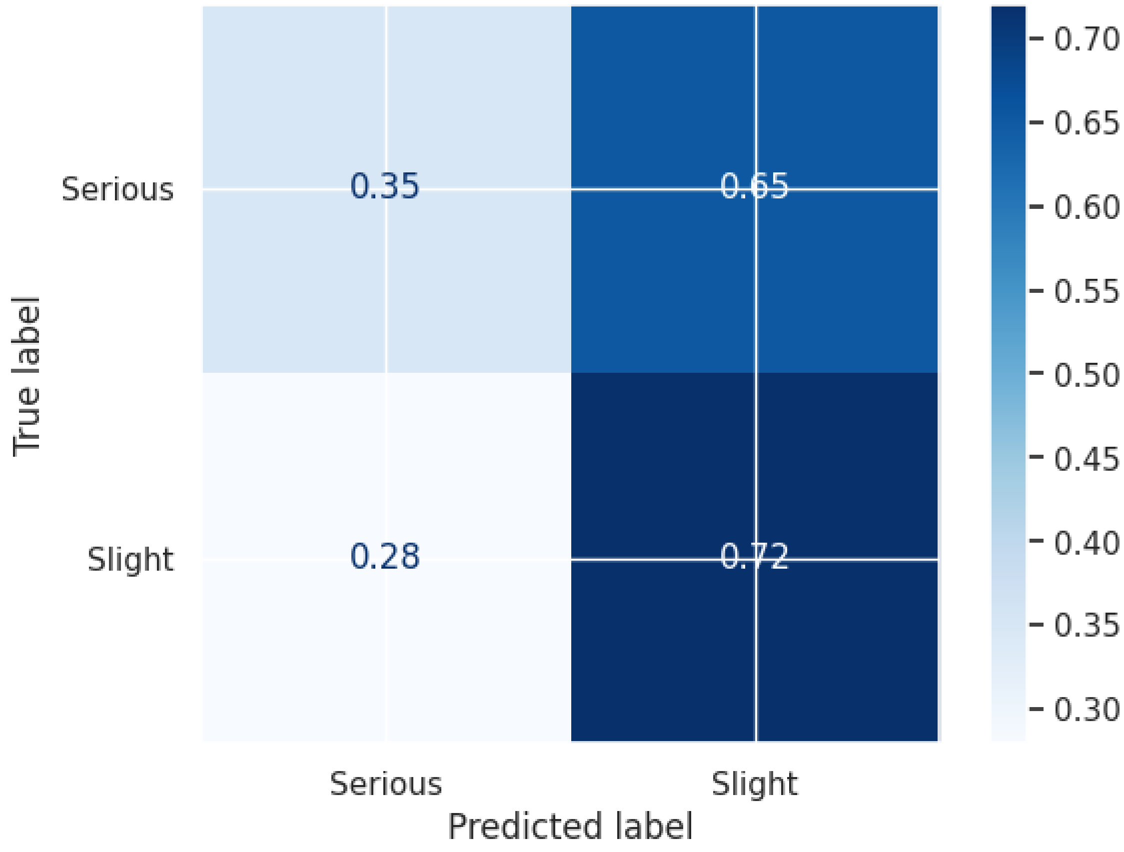
In [ ]:
```
os_grid_search.best_score_
```
Out[ ]:
```
0.5237092190753238
```

The model's performance is slightly better than what was obtained by the Random Oversampling.

In [ ]:
```
# cross-validation confusion matrix on the training data
yhat = cross_val_predict(os_grid_search.best_estimator_, Xtrain, ytrain, cv=10)

ConfusionMatrixDisplay.from_predictions(ytrain, yhat,
                                        labels=os_grid_search.best_estimator_.classes_,
                                        normalize="true",
                                        cmap=plt.cm.Blues)
```
Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff6185e39a0>
```



In [ ]:
```
# cross-validation confusion matrix on the training data
yhat = cross_val_predict(os_grid_search.best_estimator_, Xtrain, ytrain, cv=10)

ConfusionMatrixDisplay.from_predictions(ytrain, yhat,
                                        labels=os_grid_search.best_estimator_.classes_,
                                        normalize="true",
                                        cmap=plt.cm.Blues)
```
Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff6183b45e0>
```



The model still has a tendency to classify most test instances as "SLIGHT", its ratio is much smaller than before: 65% of "SERIOUS" instances were classified as "SLIGHT". This is a big reduction compared to 93% of the classifier trained on unbalanced data.

3.RANDOM UNDERSAMPLING

In this technique, some training instances of the majority class from the data will be removed.

In [ ]:
```
from imblearn.under_sampling import RandomUnderSampler

pipeline = Pipeline([
        ('rus', RandomUnderSampler(random_state=7)),
        ('rfc', RandomForestClassifier(random_state=7))
    ])

# specifying the hyperparameters
param_grid = [
    {
        'rus__sampling_strategy': [0.5, 0.75, 1.0],
        'rfc__n_estimators': [100, 200, 500],
        'rfc__max_depth': [5, None]
    },
]

rus_grid_search = GridSearchCV(pipeline, param_grid, cv=10,
                               scoring='f1_macro',
                               return_train_score=True)

rus_grid_search.fit(Xtrain, ytrain)
```

Out[ ]:
```
GridSearchCV
estimator: Pipeline
RandomUnderSampler
RandomForestClassifier
```

In [ ]:
```
# printing the cross validation results
print_cv_results(rus_grid_search, col_width=100)
```

	params	mean_train_score	mean_test_score	diff, %
9	{'rfc__max_depth': None, 'rfc__n_estimators': 100, 'rus__sampling_strategy': 0.5}	0.875638	0.509607	41.801679
12	{'rfc__max_depth': None, 'rfc__n_estimators': 200, 'rus__sampling_strategy': 0.5}	0.877421	0.506538	42.269645
1	{'rfc__max_depth': 5, 'rfc__n_estimators': 100, 'rus__sampling_strategy': 0.75}	0.530435	0.505747	4.654296
15	{'rfc__max_depth': None, 'rfc__n_estimators': 500, 'rus__sampling_strategy': 0.5}	0.879144	0.505071	42.549688
8	{'rfc__max_depth': 5, 'rfc__n_estimators': 500, 'rus__sampling_strategy': 1.0}	0.532170	0.504275	5.241841
...	...	...	...	...
14	{'rfc__max_depth': None, 'rfc__n_estimators': 200, 'rus__sampling_strategy': 1.0}	0.674862	0.463534	31.314297
11	{'rfc__max_depth': None, 'rfc__n_estimators': 100, 'rus__sampling_strategy': 1.0}	0.672495	0.458202	31.865253
0	{'rfc__max_depth': 5, 'rfc__n_estimators': 100, 'rus__sampling_strategy': 0.5}	0.449734	0.446419	0.737029
6	{'rfc__max_depth': 5, 'rfc__n_estimators': 500, 'rus__sampling_strategy': 0.5}	0.450106	0.446035	0.904493
3	{'rfc__max_depth': 5, 'rfc__n_estimators': 200, 'rus__sampling_strategy': 0.5}	0.449269	0.445710	0.792222

18 rows × 4 columns

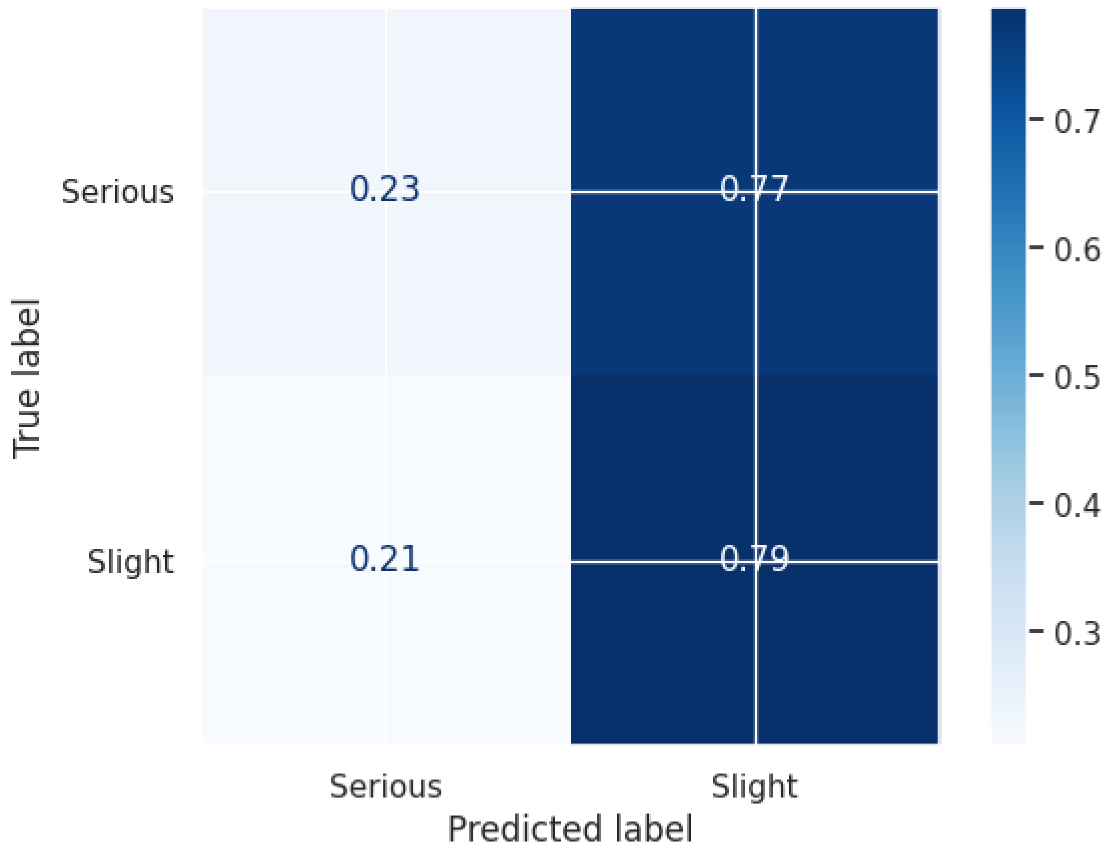
The f1 score is same as the one obtained with Random Oversampling.

In [ ]:
```
rus_grid_search.best_score_
```

Out[ ]:
```
0.5096067927723548
```

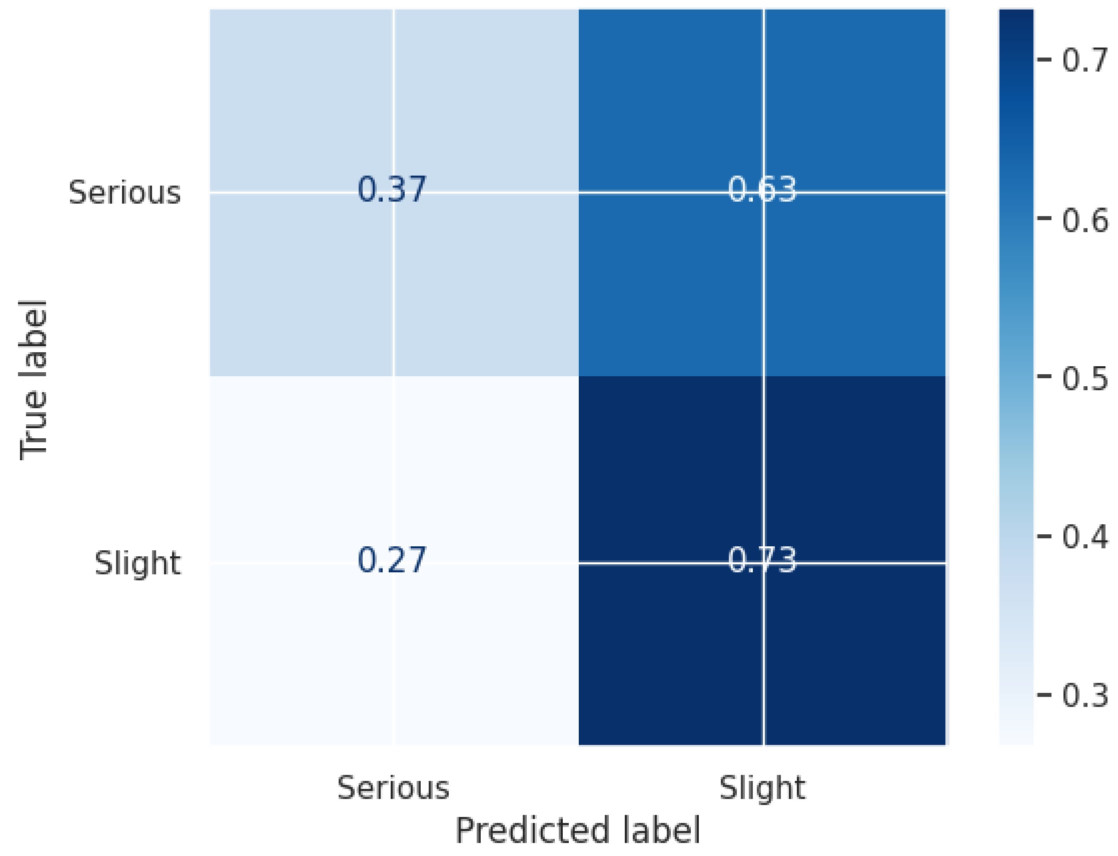
In [ ]:
```
# cross-validation confusion matrix on the training data
yhat = cross_val_predict(rus_grid_search.best_estimator_, Xtrain, ytrain, cv=10)

ConfusionMatrixDisplay.from_predictions(ytrain, yhat,
                                        labels=rus_grid_search.best_estimator_.classes_,
                                        normalize="true",
                                        cmap=plt.cm.Blues)
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff6189f8610>
```



In [ ]:
```
# confusion matrix on the test data
ConfusionMatrixDisplay.from_estimator(os_grid_search.best_estimator_, Xtest, ytest,
                                      display_labels=os_grid_search.best_estimator_.classes_,
                                      cmap=plt.cm.Blues,
                                      normalize='true')
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7ff6182094f0>
```



The f-score for this model is also pretty much similar to what was obtained by Random Oversampling.

SMOTE(0.52), by far has a better performance as compared to the unbalanced data(0.44) and slightly better as compared with the other 2 models built by Random Oversampling and Undersampling(0.50).

We will be using the **SMOTE technique** to deal with the class imbalance and start building the models.

5.2 RANDOM FORESTS(Contd.)

In [ ]:
```
# finding the best estimator
os_grid_search.best_estimator_
```

Out[ ]:
```
Pipeline
  SMOTE
  RandomForestClassifier
```

The best hyperparameters prove to be n_estimators=500, max_depth=5 and sampling strategy =1.

In [ ]:
```
# finding the best f1 score
os_grid_search.best_score_
```

Out[ ]:
```
0.5237092190753238
```

The best f1 score is **0.52**

The results of the best model in each split will be recorded, for future reference.

In [ ]:
```
# recording the results of the best model
best_model_index = os_grid_search.cv_results_["rank_test_score"].tolist().index(1)
best_model_index
```

Out[ ]:
```
8
```

The F-scores achieved by the best model in each fold will be stored, in order to run a t-test to compare the mean F-score with the mean scores of other methods.

In [ ]:
```
rf_split_test_scores = []
for x in range(5):
    # extract f-score of the best model (index='best_model_index') from each of the 5 splits
    val = os_grid_search.cv_results_[f"split{x}_test_score"][best_model_index]
    rf_split_test_scores.append(val)
```

In [ ]:
```
# printing the cv test scores
print_cv_results(os_grid_search, col_width=100)
```

	params	mean_train_score	mean_test_score	diff, %
8	{'rfc__max_depth': 5, 'rfc__n_estimators': 500, 'smote__sampling_strategy': 1.0}	0.549141	0.523709	4.631247
5	{'rfc__max_depth': 5, 'rfc__n_estimators': 200, 'smote__sampling_strategy': 1.0}	0.549149	0.521296	5.072036
2	{'rfc__max_depth': 5, 'rfc__n_estimators': 100, 'smote__sampling_strategy': 1.0}	0.548038	0.519812	5.150411
14	{'rfc__max_depth': None, 'rfc__n_estimators': 200, 'smote__sampling_strategy': 1.0}	0.975248	0.503987	48.322182
17	{'rfc__max_depth': None, 'rfc__n_estimators': 500, 'smote__sampling_strategy': 1.0}	0.975231	0.503451	48.376167
...	...	...	...	...
4	{'rfc__max_depth': 5, 'rfc__n_estimators': 200, 'smote__sampling_strategy': 0.75}	0.502735	0.486480	3.233140
7	{'rfc__max_depth': 5, 'rfc__n_estimators': 500, 'smote__sampling_strategy': 0.75}	0.502748	0.485743	3.382445
3	{'rfc__max_depth': 5, 'rfc__n_estimators': 200, 'smote__sampling_strategy': 0.5}	0.443578	0.443370	0.046768
0	{'rfc__max_depth': 5, 'rfc__n_estimators': 100, 'smote__sampling_strategy': 0.5}	0.443829	0.443370	0.103452
6	{'rfc__max_depth': 5, 'rfc__n_estimators': 500, 'smote__sampling_strategy': 0.5}	0.443619	0.443347	0.061350

18 rows × 4 columns

The performance of Random Forest classifiers doesn't vary a lot across the runs, on the validation set. There is some evidence of overfitting with the unconstrained max.depth i.e. the performance on training parts is considerably better than on the validation part. This suggests we may gain further improvements by regularizing the model with other hyperparameters of the RF method and by trying higher values for max.depth.

As the models take a long time to train, they will be saved to disk for future reading.The dump function from the built-in Python module joblib will be used for the same.

In [ ]:
```
# saving the model to the disk.
import os
from joblib import dump

# create a folder where all trained models will be kept
if not os.path.exists("models"):
    os.makedirs("models")

dump(os_grid_search.best_estimator_, 'models/rf-clf.joblib')
```

Out[ ]:
```
['models/rf-clf.joblib']
```

5.3 SUPPORT VECTOR MACHINES

SVM finds a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.

In [ ]:
```
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE
from sklearn.svm import LinearSVC

# creating a pipeline for the processing
pipeline = Pipeline([
        ('smote', SMOTE(random_state=7)),
        ('lsvm', LinearSVC(random_state=7, max_iter=5000))
    ])

# specifying the hyperparameters for SMOTE and Linear support vector machines.
# 3 x 5 = 15 hyperparameters in the grid
param_grid = [
    {
        'smote__sampling_strategy': [0.5, 0.75, 1.0],
        'lsvm__C': [0.001, 0.01, 0.1, 1, 10],
        },
]

os_grid_search = GridSearchCV(pipeline, param_grid, cv=10,
                              scoring='f1_macro',
                              return_train_score=True)

# fitting the model on the training set
os_grid_search.fit(Xtrain, ytrain)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
```

```
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

Out[ ]:

```
GridSearchCV
estimator: Pipeline
  SMOTE
  LinearSVC
```

In [ ]:

```python
# finding the best Linear SVM
os_grid_search.best_estimator_
```

Out[ ]:

```
Pipeline
  SMOTE
  LinearSVC
```

The hyperparameters for the best estimator came out to be as C=0.001 with 0.75 sampling strategy.

In [ ]:

```python
# finding the best scores
os_grid_search.best_score_
```

Out[ ]:

```
0.5101532784458767
```

There is **no improvement** on the performance of the model as compared to the RANDOM FORESTS.

In [ ]:

```python
# printing the cv test scores
print_cv_results(os_grid_search, col_width=100)
```

	params	mean_train_score	mean_test_score	diff, %
1	{'lsvm__C': 0.001, 'smote__sampling_strategy': 0.75}	0.516819	0.510153	1.289827
10	{'lsvm__C': 1, 'smote__sampling_strategy': 0.75}	0.516902	0.509531	1.426071
4	{'lsvm__C': 0.01, 'smote__sampling_strategy': 0.75}	0.516547	0.509425	1.378799
7	{'lsvm__C': 0.1, 'smote__sampling_strategy': 0.75}	0.516905	0.509145	1.501333
13	{'lsvm__C': 10, 'smote__sampling_strategy': 0.75}	0.516700	0.508560	1.575349
...	...	...	...	...
0	{'lsvm__C': 0.001, 'smote__sampling_strategy': 0.5}	0.443370	0.443370	0.000001
3	{'lsvm__C': 0.01, 'smote__sampling_strategy': 0.5}	0.443370	0.443370	0.000001
6	{'lsvm__C': 0.1, 'smote__sampling_strategy': 0.5}	0.443370	0.443370	0.000001
9	{'lsvm__C': 1, 'smote__sampling_strategy': 0.5}	0.443370	0.443370	0.000001
12	{'lsvm__C': 10, 'smote__sampling_strategy': 0.5}	0.443423	0.443347	0.017169

15 rows × 4 columns

Fine-tuning C of the linear SVM method does not seem to produce much effect on the quality of the model.

5.4 RADIAL BASIS FUNCTION

A radial basis function network is a type of supervised artificial neural network that functions as a nonlinear classifier.

In [ ]:

```python
%%time
from sklearn.svm import SVC
from sklearn.model_selection import RandomizedSearchCV
pipeline = Pipeline([
        ('smote', SMOTE(random_state=7)),
        ('svm', SVC(random_state=7, kernel='rbf'))
    ])
# specify the hyperparameters and their values
# 3 x 5 x 2 = 30 combinations in the grid
param_grid = [
    {
        'smote__sampling_strategy': [0.5, 0.75, 1.0],
        'svm__C': [0.01, 0.1, 1, 10,100],
        'svm__gamma':["auto",0.1],
```

```
        },
]

os_grid_search = RandomizedSearchCV(pipeline, param_grid, cv=10,
                            scoring='f1_macro',
                            return_train_score=True,verbose=2,n_iter=6)

os_grid_search.fit(Xtrain, ytrain)
```

```
Fitting 10 folds for each of 6 candidates, totalling 60 fits
[CV] END smote__sampling_strategy=1.0, svm__C=0.01, svm__gamma=auto; total time=  23.1s
[CV] END smote__sampling_strategy=1.0, svm__C=0.01, svm__gamma=auto; total time=  22.7s
[CV] END smote__sampling_strategy=1.0, svm__C=0.01, svm__gamma=auto; total time=  22.4s
[CV] END smote__sampling_strategy=1.0, svm__C=0.01, svm__gamma=auto; total time=  21.3s
[CV] END smote__sampling_strategy=1.0, svm__C=0.01, svm__gamma=auto; total time=  21.6s
[CV] END smote__sampling_strategy=1.0, svm__C=0.01, svm__gamma=auto; total time=  21.3s
[CV] END smote__sampling_strategy=1.0, svm__C=0.01, svm__gamma=auto; total time=  21.7s
[CV] END smote__sampling_strategy=1.0, svm__C=0.01, svm__gamma=auto; total time=  22.3s
[CV] END smote__sampling_strategy=1.0, svm__C=0.01, svm__gamma=auto; total time=  22.7s
[CV] END smote__sampling_strategy=1.0, svm__C=0.01, svm__gamma=auto; total time=  22.5s
[CV] END smote__sampling_strategy=1.0, svm__C=0.01, svm__gamma=0.1; total time=  22.8s
[CV] END smote__sampling_strategy=1.0, svm__C=0.01, svm__gamma=0.1; total time=  22.3s
[CV] END smote__sampling_strategy=1.0, svm__C=0.01, svm__gamma=0.1; total time=  22.4s
[CV] END smote__sampling_strategy=1.0, svm__C=0.01, svm__gamma=0.1; total time=  22.2s
[CV] END smote__sampling_strategy=1.0, svm__C=0.01, svm__gamma=0.1; total time=  22.5s
[CV] END smote__sampling_strategy=1.0, svm__C=0.01, svm__gamma=0.1; total time=  22.6s
[CV] END smote__sampling_strategy=1.0, svm__C=0.01, svm__gamma=0.1; total time=  22.7s
[CV] END smote__sampling_strategy=1.0, svm__C=0.01, svm__gamma=0.1; total time=  23.0s
[CV] END smote__sampling_strategy=1.0, svm__C=0.01, svm__gamma=0.1; total time=  22.6s
[CV] END smote__sampling_strategy=1.0, svm__C=0.01, svm__gamma=0.1; total time=  21.8s
[CV] END smote__sampling_strategy=0.5, svm__C=0.01, svm__gamma=auto; total time=   8.7s
[CV] END smote__sampling_strategy=0.5, svm__C=0.01, svm__gamma=auto; total time=   9.6s
[CV] END smote__sampling_strategy=0.5, svm__C=0.01, svm__gamma=auto; total time=   9.7s
[CV] END smote__sampling_strategy=0.5, svm__C=0.01, svm__gamma=auto; total time=   9.8s
[CV] END smote__sampling_strategy=0.5, svm__C=0.01, svm__gamma=auto; total time=   9.6s
[CV] END smote__sampling_strategy=0.5, svm__C=0.01, svm__gamma=auto; total time=   8.8s
[CV] END smote__sampling_strategy=0.5, svm__C=0.01, svm__gamma=auto; total time=   9.4s
[CV] END smote__sampling_strategy=0.5, svm__C=0.01, svm__gamma=auto; total time=   9.6s
[CV] END smote__sampling_strategy=0.5, svm__C=0.01, svm__gamma=auto; total time=   8.6s
[CV] END smote__sampling_strategy=0.5, svm__C=0.01, svm__gamma=auto; total time=   9.6s
[CV] END smote__sampling_strategy=0.5, svm__C=100, svm__gamma=0.1; total time= 1.4min
[CV] END smote__sampling_strategy=0.5, svm__C=100, svm__gamma=0.1; total time= 1.8min
[CV] END smote__sampling_strategy=0.5, svm__C=100, svm__gamma=0.1; total time= 1.8min
[CV] END smote__sampling_strategy=0.5, svm__C=100, svm__gamma=0.1; total time= 2.0min
[CV] END smote__sampling_strategy=0.5, svm__C=100, svm__gamma=0.1; total time= 1.9min
[CV] END smote__sampling_strategy=0.5, svm__C=100, svm__gamma=0.1; total time= 2.0min
[CV] END smote__sampling_strategy=0.5, svm__C=100, svm__gamma=0.1; total time= 1.8min
[CV] END smote__sampling_strategy=0.5, svm__C=100, svm__gamma=0.1; total time= 1.5min
[CV] END smote__sampling_strategy=0.5, svm__C=100, svm__gamma=0.1; total time= 1.6min
[CV] END smote__sampling_strategy=0.5, svm__C=100, svm__gamma=0.1; total time= 1.8min
[CV] END smote__sampling_strategy=0.75, svm__C=1, svm__gamma=auto; total time=  19.4s
[CV] END smote__sampling_strategy=0.75, svm__C=1, svm__gamma=auto; total time=  19.1s
[CV] END smote__sampling_strategy=0.75, svm__C=1, svm__gamma=auto; total time=  19.5s
[CV] END smote__sampling_strategy=0.75, svm__C=1, svm__gamma=auto; total time=  18.5s
[CV] END smote__sampling_strategy=0.75, svm__C=1, svm__gamma=auto; total time=  19.4s
[CV] END smote__sampling_strategy=0.75, svm__C=1, svm__gamma=auto; total time=  19.6s
[CV] END smote__sampling_strategy=0.75, svm__C=1, svm__gamma=auto; total time=  18.4s
[CV] END smote__sampling_strategy=0.75, svm__C=1, svm__gamma=auto; total time=  19.5s
[CV] END smote__sampling_strategy=0.75, svm__C=1, svm__gamma=auto; total time=  18.3s
[CV] END smote__sampling_strategy=0.75, svm__C=1, svm__gamma=auto; total time=  18.6s
[CV] END smote__sampling_strategy=0.5, svm__C=1, svm__gamma=auto; total time=  18.9s
[CV] END smote__sampling_strategy=0.5, svm__C=1, svm__gamma=auto; total time=  18.7s
[CV] END smote__sampling_strategy=0.5, svm__C=1, svm__gamma=auto; total time=  17.7s
[CV] END smote__sampling_strategy=0.5, svm__C=1, svm__gamma=auto; total time=  17.1s
[CV] END smote__sampling_strategy=0.5, svm__C=1, svm__gamma=auto; total time=  16.9s
[CV] END smote__sampling_strategy=0.5, svm__C=1, svm__gamma=auto; total time=  16.9s
[CV] END smote__sampling_strategy=0.5, svm__C=1, svm__gamma=auto; total time=  16.9s
[CV] END smote__sampling_strategy=0.5, svm__C=1, svm__gamma=auto; total time=  17.8s
[CV] END smote__sampling_strategy=0.5, svm__C=1, svm__gamma=auto; total time=  17.5s
[CV] END smote__sampling_strategy=0.5, svm__C=1, svm__gamma=auto; total time=  17.6s
CPU times: user 44min 16s, sys: 2.66 s, total: 44min 18s
Wall time: 44min 25s
```

Out[ ]:

▸ **RandomizedSearchCV**

▸ **estimator: Pipeline**

▸ SMOTE

▸ SVC

In [ ]:

```
# finding the best estimator
os_grid_search.best_estimator_
```

Out[ ]:

▸ **Pipeline**

▸ SMOTE

▸ SVC

The best estimator is with the hyperparameters of C= 1, gamma =0.1 and sampling strategy = 0.75.

In [ ]:
```
# finding the best score
os_grid_search.best_score_
```

Out[ ]:
```
0.5256218347921497
```

No substantial improvement is seen on the model's performance.

In [ ]:
```
# recording the results of the best model.
best_model_index = os_grid_search.cv_results_["rank_test_score"].tolist().index(1)
best_model_index
```

Out[ ]:
```
4
```

In [ ]:
```
# printing the cv test scores
print_cv_results(os_grid_search, col_width=100)
```

	params	mean_train_score	mean_test_score	diff, %
4	{'svm__gamma': 'auto', 'svm__C': 1, 'smote__sampling_strategy': 0.75}	0.548126	0.525622	4.105697
0	{'svm__gamma': 'auto', 'svm__C': 0.01, 'smote__sampling_strategy': 1.0}	0.512723	0.503288	1.840096
1	{'svm__gamma': 0.1, 'svm__C': 0.01, 'smote__sampling_strategy': 1.0}	0.512723	0.503288	1.840096
3	{'svm__gamma': 0.1, 'svm__C': 100, 'smote__sampling_strategy': 0.5}	0.516456	0.469342	9.122667
5	{'svm__gamma': 'auto', 'svm__C': 1, 'smote__sampling_strategy': 0.5}	0.447638	0.444053	0.800825
2	{'svm__gamma': 'auto', 'svm__C': 0.01, 'smote__sampling_strategy': 0.5}	0.443370	0.443370	0.000001

The f-score is almost same as compared to the Random Forests model. We can check for statistical significance of the difference.

As before, let's retrieve the scores achieved by the best model on each fold.

Looking at the classification accuracy scores of each model, we notice that they are strongly affected by the settings for C: values below 0.1 result in very poor scores.

The most promising results are achieved with C=1 and gamma set to 0.01. Also, it is observed that the performance of the model is decreasing with an increase in C.

In [ ]:
```
# obtaining the f-scores of the best models in each split

svmrbf_split_test_scores = []
for x in range(5):
    # extract f-score of the best model (at index=best_model_index) from each of the 5 splits
    val = os_grid_search.cv_results_[f"split{x}_test_score"][best_model_index]
    svmrbf_split_test_scores.append(val)
```

An independent samples t-test will be carried out to compare the mean F-scores of the best RF classifier and the best RBF SVM classifier, obtained by averaging across the folds.

In [ ]:
```
# obtaining the mean F scores for the best RF classifier and best RBF SVM classifier
print(f"Mean F-score of RF across the folds: {np.array(rf_split_test_scores).mean()}")
print(f"Mean F-score of RBF SVM across the folds: {np.array(svmrbf_split_test_scores).mean()}")
```

```
Mean F-score of RF across the folds: 0.5210322110291331
Mean F-score of RBF SVM across the folds: 0.5290731568369268
```

In [ ]:
```
# performing the t-test to check that whether there is a significant difference between the 2 means or not.
# Null Hypothesis : $H_0$ : There is no significant difference between the means.
# Alternative Hypothesis : $H_1$ : There is a significant difference between the means.

# return the t-score and a two-tailed p-value
from scipy.stats import ttest_ind

#printing the scores
ttest_ind(rf_split_test_scores, svmrbf_split_test_scores)
```

Out[ ]:
```
Ttest_indResult(statistic=-0.7812340002272946, pvalue=0.45715868802386805)
```

As p value is greater than 0.05 , we reject the null hypothesis thus by concluding that there is no significant difference between the means.

In [ ]:
```
# saving the model to the disk
import os
from joblib import dump

# create a folder where all trained models will be kept
if not os.path.exists("models"):
    os.makedirs("models")

dump(os_grid_search.best_estimator_, 'models/svm-rbf-clf.joblib')
```

Out[ ]:
```
['models/svm-rbf-clf.joblib']
```

5.5 POLYNOMIAL SVM

It uses a polynomial function to map the data into a higher-dimensional space.

In [ ]:

```
%%time
from sklearn.svm import SVC
from sklearn.model_selection import RandomizedSearchCV

pipeline = Pipeline([
        ('smote', SMOTE(random_state=7)),
        ('svm_poly',SVC(random_state=7, kernel='poly',gamma="scale",degree=2))
    ])
# specify the hyperparameters and their values
# 3 x 5  = 15 combinations in the grid
param_grid = [
    {
        'smote__sampling_strategy': [0.5, 0.75, 1.0],
        'svm_poly__C': [0.01, 0.1, 1, 10,100],
        },
]

os_grid_search = RandomizedSearchCV(pipeline, param_grid, cv=5,
                              scoring='f1_macro',
                              return_train_score=True,verbose=2,n_iter=6)

os_grid_search.fit(Xtrain, ytrain)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
[CV] END .....smote__sampling_strategy=0.75, svm_poly__C=0.1; total time=  12.5s
[CV] END .....smote__sampling_strategy=0.75, svm_poly__C=0.1; total time=  12.1s
[CV] END .....smote__sampling_strategy=0.75, svm_poly__C=0.1; total time=  11.1s
[CV] END .....smote__sampling_strategy=0.75, svm_poly__C=0.1; total time=  12.8s
[CV] END .....smote__sampling_strategy=0.75, svm_poly__C=0.1; total time=  11.5s
[CV] END .......smote__sampling_strategy=0.75, svm_poly__C=1; total time=  17.9s
[CV] END .......smote__sampling_strategy=0.75, svm_poly__C=1; total time=  15.5s
[CV] END .......smote__sampling_strategy=0.75, svm_poly__C=1; total time=  14.2s
[CV] END .......smote__sampling_strategy=0.75, svm_poly__C=1; total time=  20.2s
[CV] END .......smote__sampling_strategy=0.75, svm_poly__C=1; total time=  11.4s
[CV] END .......smote__sampling_strategy=1.0, svm_poly__C=10; total time=  53.5s
[CV] END .......smote__sampling_strategy=1.0, svm_poly__C=10; total time=  48.8s
[CV] END .......smote__sampling_strategy=1.0, svm_poly__C=10; total time= 4.5min
[CV] END .......smote__sampling_strategy=1.0, svm_poly__C=10; total time=  25.9s
[CV] END .......smote__sampling_strategy=1.0, svm_poly__C=10; total time= 4.6min
[CV] END .....smote__sampling_strategy=1.0, svm_poly__C=0.01; total time=  16.1s
[CV] END .....smote__sampling_strategy=1.0, svm_poly__C=0.01; total time=  14.9s
[CV] END .....smote__sampling_strategy=1.0, svm_poly__C=0.01; total time=  14.9s
[CV] END .....smote__sampling_strategy=1.0, svm_poly__C=0.01; total time=  15.2s
[CV] END .....smote__sampling_strategy=1.0, svm_poly__C=0.01; total time=  14.8s
[CV] END ........smote__sampling_strategy=1.0, svm_poly__C=1; total time=  18.1s
[CV] END ........smote__sampling_strategy=1.0, svm_poly__C=1; total time=  16.7s
[CV] END ........smote__sampling_strategy=1.0, svm_poly__C=1; total time=  21.8s
[CV] END ........smote__sampling_strategy=1.0, svm_poly__C=1; total time=  16.1s
[CV] END ........smote__sampling_strategy=1.0, svm_poly__C=1; total time=  25.4s
[CV] END ......smote__sampling_strategy=1.0, svm_poly__C=100; total time= 6.7min
[CV] END ......smote__sampling_strategy=1.0, svm_poly__C=100; total time=10.2min
[CV] END ......smote__sampling_strategy=1.0, svm_poly__C=100; total time=37.2min
[CV] END ......smote__sampling_strategy=1.0, svm_poly__C=100; total time= 5.1min
[CV] END ......smote__sampling_strategy=1.0, svm_poly__C=100; total time=33.7min
CPU times: user 1h 52min 11s, sys: 4.11 s, total: 1h 52min 15s
Wall time: 1h 52min 26s
```

Out[ ]:

```
▸ RandomizedSearchCV
▸ estimator: Pipeline
    ▸ SMOTE
    ▸ SVC
```

In [ ]:

```
print(os_grid_search.best_estimator_)
print(os_grid_search.best_score_)
```

```
Pipeline(steps=[('smote', SMOTE(random_state=7, sampling_strategy=1.0)),
                ('svm_poly',
                 SVC(C=0.01, degree=2, kernel='poly', random_state=7))])
0.5143237126076285
```

No further improvement is observed in the accuracy score.

In [ ]:

```
# printing the cv test scores
print_cv_results(os_grid_search, col_width=100)
```

	params	mean_train_score	mean_test_score	diff, %
3	{'svm_poly__C': 0.01, 'smote__sampling_strategy': 1.0}	0.526375	0.514324	2.289449
4	{'svm_poly__C': 1, 'smote__sampling_strategy': 1.0}	0.516515	0.508429	1.565484
5	{'svm_poly__C': 100, 'smote__sampling_strategy': 1.0}	0.505713	0.497672	1.589966
2	{'svm_poly__C': 10, 'smote__sampling_strategy': 1.0}	0.505695	0.497572	1.606200
1	{'svm_poly__C': 1, 'smote__sampling_strategy': 0.75}	0.478734	0.473168	1.162615
0	{'svm_poly__C': 0.1, 'smote__sampling_strategy': 0.75}	0.445669	0.444786	0.198073

In [ ]:
```
# saving the model to the disk
import os
from joblib import dump

# create a folder where all trained models will be kept
if not os.path.exists("models"):
    os.makedirs("models")

dump(os_grid_search.best_estimator_, 'models/svm-poly-clf.joblib')
```

Out[ ]:
```
['models/svm-poly-clf.joblib']
```

6. EVALUATING THE BEST MODELS

The best model so obtained is the Random Forests with an accuracy score of 52%. However, polynomial SVM has also almost the similar result. We'll evaluate the Random Forest and the SVM with a rbf kernel on the test set.

First, load the models from disk:

In [ ]:
```
# importing the best models
from joblib import load

best_rf = load("models/rf-clf.joblib")
best_svm = load("models/svm-poly-clf.joblib")
```

RANDOM FORESTS

In [ ]:
```
from sklearn.metrics import precision_recall_fscore_support

# rf
yhat = best_rf.predict(Xtest)

# micro-averaged precision, recall and f-score
p, r, f, s = precision_recall_fscore_support(ytest, yhat, average="macro")
print("Random Forest:")
print(f"Precision: {p}")
print(f"Recall: {r}")
print(f"F score: {f}")
```

```
Random Forest:
Precision: 0.5410583372704132
Recall: 0.5524289732650496
F score: 0.5404581289238292
```

POLYNOMIAL SVM

In [ ]:
```
from sklearn.metrics import precision_recall_fscore_support

# rf
yhat = best_svm.predict(Xtest)

# micro-averaged precision, recall and f-score
p, r, f, s = precision_recall_fscore_support(ytest, yhat, average="macro")
print("Polynomial SVM:")
print(f"Precision: {p}")
print(f"Recall: {r}")
print(f"F score: {f}")
```

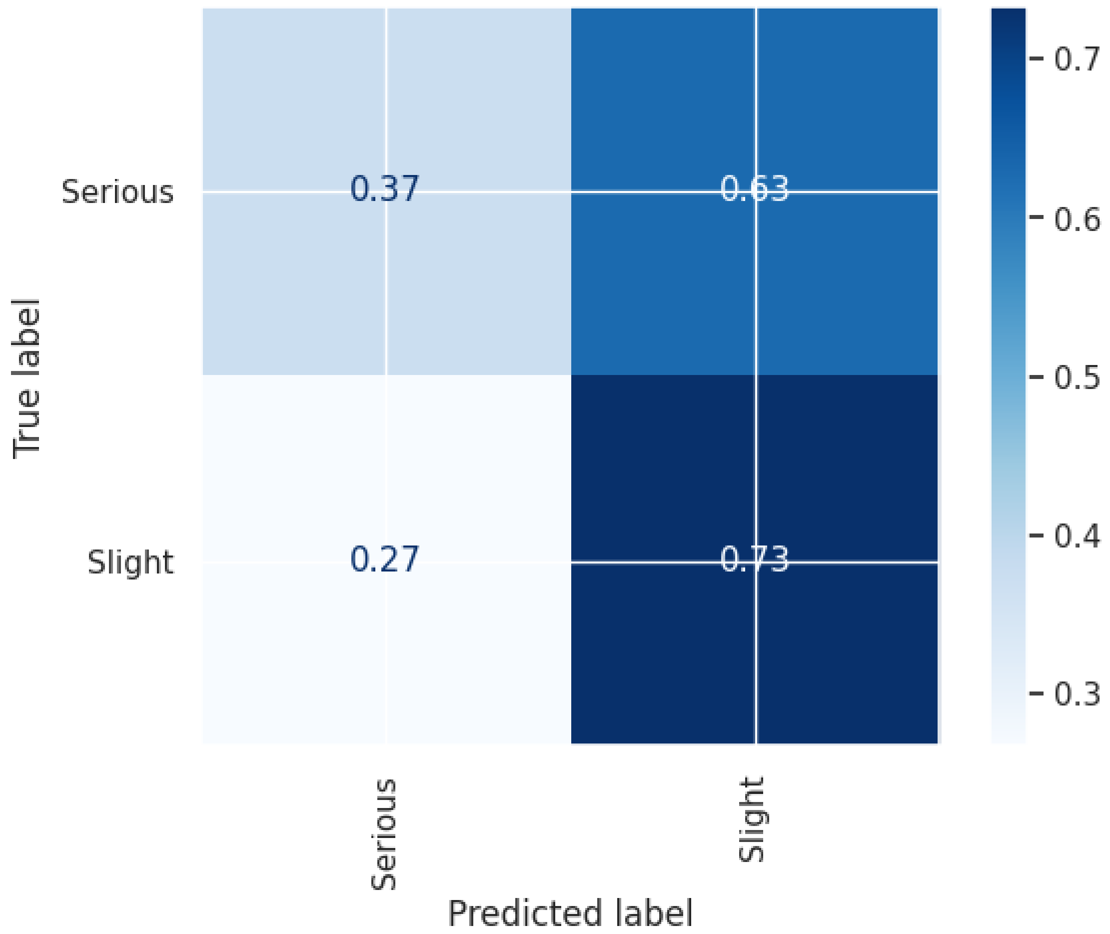
```
Polynomial SVM:
Precision: 0.49487812336187315
Recall: 0.49548895804656173
F score: 0.49444318362344825
```

Thus, it is observed that RF classifier has a better performance than Polynomial SVM, as was also observed during cross-validation. Even though, the accuaracy measure is low, the Random Forest performed better of the two.

In [ ]:
```
# Plotting a confusion matrix for Random Forests to perform error analysis
from sklearn.metrics import ConfusionMatrixDisplay
```

```
ConfusionMatrixDisplay.from_predictions(ytest, yhat, labels=best_rf.classes_,
                                        xticks_rotation="vertical", normalize="true",
                                        cmap=plt.cm.Blues)
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fc1420271c0>
```



7.CONCLUSIONS : KEY FINDINGS AND POSSIBLE FUTURE IMPROVEMENTS

The RF model gave a fair accuracy score of 54%. **From the error analysis we can see that the model is fairly classifying the "Slight" class (73% of true instances of "Slight" were classified as "Slight"). However, it has poorly classified the "Serious" class (only 37% were classified correctly)**

This poor classification of Serious Category,(which was an underrepresented class in the dataset) can be attributed to the CLASS IMBALANCE. Although sampling techniques were used to sort out the issue, but it is evident that the solution is not optimal.

FUTURE IMPROVEMENTS: Based on the above results, following recommendations can be given.

1. Adding more balanced and meaningful data can improve the accuracy of the model.
2. Use of bagging and boosting techniques will also be beneficial.
3. In place of Randomized Search, Exhaustive Search can be used for the hyperparameter tuning to get better results.
4. Extensive use of Feature Engineering and Feature selection can also help in deceloping a better model.

References:

1. Pekar, V. (2022). Big Data for Decision Making. Lecture examples and exercises. (Version 1.0.0). URL: https://github.com/vpekar/bd4dm

2. Pekar, V. (2022). Big Data for Decision Making. Lecture examples and exercises. (Version 1.0.0). URL: https://github.com/vpekar/bd4dm