Submission by:

1. Aviel Kelelev (Id: 323504670)
2. Daniel Adam (Id: 342475639)

Assignment 2: Machine Learning Problem

Pre experiment phase:

- 1) Preliminary analysis

Experiment phase:

- 2) Preprocessing
- 3) Metrics selection
- 4) Model training and evaluation
- 5) Hyperparameter tuning and best model selection

Post experiment phase:

- 6) Final evaluation on test set

1) Preliminary data analysis:

[ ] ↳ *12 cells hidden*

2)Preprocessing

[ ] ↳ *5 cells hidden*

3) Metrics selection

[ ] ↳ *2 cells hidden*

4) Model training and evaluation

We'll train the following models:

- Decision Tree
- Random Forest
- GBDT - Gradient Boost Decision Tree
- Logistic Regression
- K-nearest Neighbors

Hence the dataset is provided with labels/targets we won't train a clustering models.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier


non_sensitive_models = [
          ('Decision Tree',DecisionTreeClassifier(random_state=42)),
          ('Random Forrest', RandomForestClassifier(random_state=42)),
           ('Gradien Boost Decision Tree',GradientBoostingClassifier(learning_rate=1, random_state=42)),
        ]

sensitive_models = [
          ('Logistic Regression', LogisticRegression(max_iter=1000)),
          ('KNN', KNeighborsClassifier())
        ]

target_names = ['0', '1', '2']
def eval_model(model, x_eval, y_eval, accuracies, cv):
  y_dev_pred = clf.predict(x_eval)
  cv_scores = cross_val_score(clf, X_train, y_train,scoring='accuracy', cv=cv, n_jobs=-1)
  accuracy = accuracy_score(y_eval, y_dev_pred)
  if (accuracies.get('model') is None) or accuracy > accuracies.get('model'):
```

```
    accuracies[model] = accuracy

  #print results
  print(f'Classifier                      : {model}')
  print(f'Accuracy                        : {accuracy}')
  print(f'Weighted F1-score is            : {f1_score(y_eval, y_dev_pred ,average= "weighted")}')
  print(f'cross-validation mean accuracies : {cv_scores.mean()}')
  print(f'classification report\n {classification_report(y_eval, y_dev_pred, target_names=target_names)}')
  print(f'confusion matrix\n {confusion_matrix(y_eval, y_dev_pred)}', '\n\n')
```

4.1) Training Decision Tree, Random Forrest and Gradient Boost with imbalanced data

```
accuracies_non_sensitive_models_imbalanced = {}
for _,model in non_sensitive_models:
        clf = model.fit(X_train, y_train)
        eval_model(clf,X_dev, y_dev, accuracies_non_sensitive_models_imbalanced,  cv=5)
print(f'Accuracies are: {accuracies_non_sensitive_models_imbalanced}\n')
```

```
               1       0.62      0.63      0.63      3373
               2       0.83      0.83      0.83      8869

        accuracy                           0.72     13512
       macro avg       0.57      0.57      0.57     13512
    weighted avg       0.73      0.72      0.72     13512

    confusion matrix
     [[ 314  391  565]
     [ 359 2112  902]
     [ 631  879 7359]]


    Classifier                      : RandomForestClassifier(random_state=42)
    Accuracy                        : 0.8120189461219657
    Weighted F1-score is            : 0.7846647140806325
    cross-validation mean accuracies : 0.8004836709211679
    classification report
                  precision    recall  f1-score   support

               0       0.51      0.12      0.20      1270
               1       0.78      0.68      0.73      3373
               2       0.83      0.96      0.89      8869

        accuracy                           0.81     13512
       macro avg       0.71      0.59      0.61     13512
    weighted avg       0.79      0.81      0.78     13512

    confusion matrix
     [[ 155  340  775]
     [  91 2299  983]
     [  60  291 8518]]


    Classifier                      : GradientBoostingClassifier(learning_rate=1, random_state=42)
    Accuracy                        : 0.8108348134991119
    Weighted F1-score is            : 0.7878029405074702
    cross-validation mean accuracies : 0.8055659087949356
    classification report
                  precision    recall  f1-score   support

               0       0.49      0.14      0.21      1270
               1       0.74      0.74      0.74      3373
               2       0.85      0.94      0.89      8869

        accuracy                           0.81     13512
       macro avg       0.69      0.60      0.61     13512
    weighted avg       0.79      0.81      0.79     13512

    confusion matrix
     [[ 174  379  717]
     [  86 2483  804]
     [  94  476 8299]]


    Accuracies are: {DecisionTreeClassifier(random_state=42): 0.7241711071640023, RandomForestClassifier(random_state=42): 0.8120189
```

4.2) Training Decision Tree, Random Forrest and Gradient Boost with balanced data

```
accuracies_non_sensitive_models_balanced={}
#train with undersampled balanced data and "normal" cross validation
print('Training and evaluating models after undersampling and with cross validation \n\n')
for _, model in non_sensitive_models:
        clf = model.fit(X_train_transformed, y_train_balanced)
```

```
        eval_model(clf, X_dev_transformed, y_dev,accuracies_non_sensitive_models_balanced , cv=5)
print(f'Accuracies are: {accuracies_non_sensitive_models_balanced}\n')
```

```
              1       0.52      0.57      0.54      3373
              2       0.87      0.58      0.70      8869

       accuracy                           0.57     13512
      macro avg       0.52      0.55      0.50     13512
   weighted avg       0.71      0.57      0.62     13512

   confusion matrix
    [[ 640  365  265]
    [ 905 1934  534]
    [2235 1455 5179]]


   Classifier                      : RandomForestClassifier(random_state=42)
   Accuracy                        : 0.6718472468916519
   Weighted F1-score is            : 0.7068279105570515
   cross-validation mean accuracies : 0.8004836709211679
   classification report
                 precision    recall  f1-score   support

              0       0.23      0.61      0.34      1270
              1       0.63      0.68      0.65      3373
              2       0.92      0.68      0.78      8869

       accuracy                           0.67     13512
      macro avg       0.59      0.66      0.59     13512
   weighted avg       0.78      0.67      0.71     13512

   confusion matrix
    [[ 775  293  202]
    [ 766 2293  314]
    [1777 1082 6010]]


   Classifier                      : GradientBoostingClassifier(learning_rate=1, random_state=42)
   Accuracy                        : 0.7020426287744227
   Weighted F1-score is            : 0.7337740432527801
   cross-validation mean accuracies : 0.8055659087949356
   classification report
                 precision    recall  f1-score   support

              0       0.25      0.62      0.36      1270
              1       0.67      0.70      0.68      3373
              2       0.93      0.71      0.81      8869

       accuracy                           0.70     13512
      macro avg       0.62      0.68      0.62     13512
   weighted avg       0.80      0.70      0.73     13512

   confusion matrix
    [[ 792  275  203]
    [ 716 2356  301]
    [1634  897 6338]]


   Accuracies are: {DecisionTreeClassifier(random_state=42): 0.5737862640615748, RandomForestClassifier(random_state=42): 0.6718472
```

We see that the tree algorithms performed worse on the balanced data after undersampling, even when tree algorithms are generally not sensitive to imbalanced or balanced data. This may be a result of the "nature" of given data and altering the original data distribution may lead to decrease of accuracy which may be caused due to loss of information, class noise etc. after the undersampling.

4.3) Training separately classifiers sensitive to imbalanced and not stadardized data.

```
accuracies_sensitive_models_={}
#train with undersampled balanced data and "normal" cross validation
print('Training and evaluating models after undersampling and with cross validation \n\n')
for _, model in sensitive_models:
        clf = model.fit(X_train_transformed_balanced, y_train_balanced)
        eval_model(clf, X_dev_transformed, y_dev,accuracies_sensitive_models_ , cv=5)
print(f'Accuracies are: {accuracies_sensitive_models_}\n')

#train with repeated stratified 5-fold cross validation
rep_strat_kfold = RepeatedStratifiedKFold(n_repeats=2, n_splits=5, random_state=42)
print('Training and evaluating models with repeated stratified 5-fold cross validation \n\n')
for _, model in sensitive_models:
        clf = model.fit(X_train_transformed, y_train)
        eval_model(clf, X_dev_transformed, y_dev,accuracies_sensitive_models_ , cv=rep_strat_kfold)
print(f'Accuracies are: {accuracies_sensitive_models_}\n')
```

```
Classifier                      : LogisticRegression(max_iter=1000)
Accuracy                        : 0.6597098875074008
Weighted F1-score is            : 0.5348705144169478
cross-validation mean accuracies : 0.6610539377547483
classification report
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1270
           1       0.58      0.03      0.06      3373
           2       0.66      0.99      0.79      8869

    accuracy                           0.66     13512
   macro avg       0.41      0.34      0.28     13512
weighted avg       0.58      0.66      0.53     13512

confusion matrix
 [[   0   19 1251]
 [   0   98 3275]
 [   0   53 8816]]


/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to

Classifier                      : KNeighborsClassifier()
Accuracy                        : 0.7259473060982831
Weighted F1-score is            : 0.7124183744273278
cross-validation mean accuracies : 0.7359560329154189
classification report
              precision    recall  f1-score   support

           0       0.31      0.23      0.26      1270
           1       0.62      0.50      0.55      3373
           2       0.80      0.88      0.84      8869

    accuracy                           0.73     13512
   macro avg       0.57      0.54      0.55     13512
weighted avg       0.71      0.73      0.71     13512

confusion matrix
 [[ 291  344  635]
 [ 316 1694 1363]
 [ 335  710 7824]]


Accuracies are: {LogisticRegression(max_iter=1000): 0.6597098875074008, KNeighborsClassifier(): 0.7259473060982831}
```

We see that KNN and Logistic Regression both performed much better with repeated stratified 5-fold cross validation then with the undersampled data. This could be explained since repeated stratified k-fold cross validation:

- provides a more comprehensive evaluation by considering multiple iterations of training and testing on different subsets of the data.
- reduces the potential bias introduced by undersampling, where some information from the majority class may be lost.
- allows the models to learn from a wider range of samples, including both majority and minority class instances, which can lead to better performance.

5) Hyperparamter tuning and best model selection

5.1) Tuning hyperparameters of best 2 models of section 4:

Explanation:

> We found Random Forest Classifier and Gradien Boost Classifier performing best on dev-set. Thus, tuning hyperparameters of both, trying to improve performance on dev-set and finally choosing best classifier on dev-set.

```
overall_accuracies = [accuracies_non_sensitive_models_imbalanced, accuracies_non_sensitive_models_balanced, accuracies_sensitive_models_]
def get_best_classifiers(overall_accuracies):
    top_models = []
    top_accuracies = [0.0, 0.0]
```

```
    for dictionary in overall_accuracies:
        for model, accuracy in dictionary.items():
            # Update the top two models if the current model has a higher accuracy
            if accuracy > top_accuracies[0]:
                top_models.insert(0, model)
                top_accuracies.insert(0, accuracy)
            elif accuracy > top_accuracies[1]:
                top_models.insert(1, model)
                top_accuracies.insert(1, accuracy)

    return top_models[:2], top_accuracies[:2]


for accuracy in overall_accuracies:
  print(f'Accuracies of non sensitive models with imbalanced data: {accuracy}')

best_models, best_accuracies = get_best_classifiers(overall_accuracies)
# Print maximum accuracy and corresponding model
print("\n\n2 best accuracies  :", best_accuracies)
print("2 best classifiers :", best_models)
```

```
Accuracies of non sensitive models with imbalanced data: {DecisionTreeClassifier(random_state=42): 0.7241711071640023, RandomForest
Accuracies of non sensitive models with imbalanced data: {DecisionTreeClassifier(random_state=42): 0.5737862640615748, RandomForest
Accuracies of non sensitive models with imbalanced data: {LogisticRegression(max_iter=1000): 0.4126702190645352, KNeighborsClassifi


2 best accuracies  : [0.8120189461219657, 0.8108348134991119]
2 best classifiers : [RandomForestClassifier(random_state=42), GradientBoostingClassifier(learning_rate=1, random_state=42)]
```

Tuning following hyperparamter with optuna:

- n_estimators
- max_depth
- max_features
- min_samples_split
- min_samples_leaf
- criterion

Explanation of hyperparameters and how they may affect performance:

- n_estimators:

  > Increasing number of Decision Trees can improve models perfomance by reducing overfitting, but can also increase training time.

- max_depth:

  > It controls the maximum depth of each decision tree. A deeper tree can capture more complex relationships in the data, but it may also lead to overfitting. Setting an appropriate max_depth value helps balance model complexity and generalization.

- max_features:

  > It determines the maximum number of features to consider when looking for the best split at each node. Limiting the number of features can improve the diversity and randomness of the trees, reducing the likelihood of overfitting.

- min_samples_split:

  > It specifies the minimum number of samples required to split an internal node during the construction of each decision tree. Increasing this value can prevent overfitting by enforcing a minimum number of samples required for a split.

- min_samples_leaf:

  > It sets the minimum number of samples required to be at a leaf node. Similar to min_samples_split, increasing this value helps prevent overfitting by requiring a minimum number of samples in each leaf.

- criterion:

  > It defines the function used to measure the quality of a split. Gini impurity measures the probability of misclassifying a randomly chosen element, while entropy measures the information gain in terms of the reduction in uncertainty.

- learning_rate:

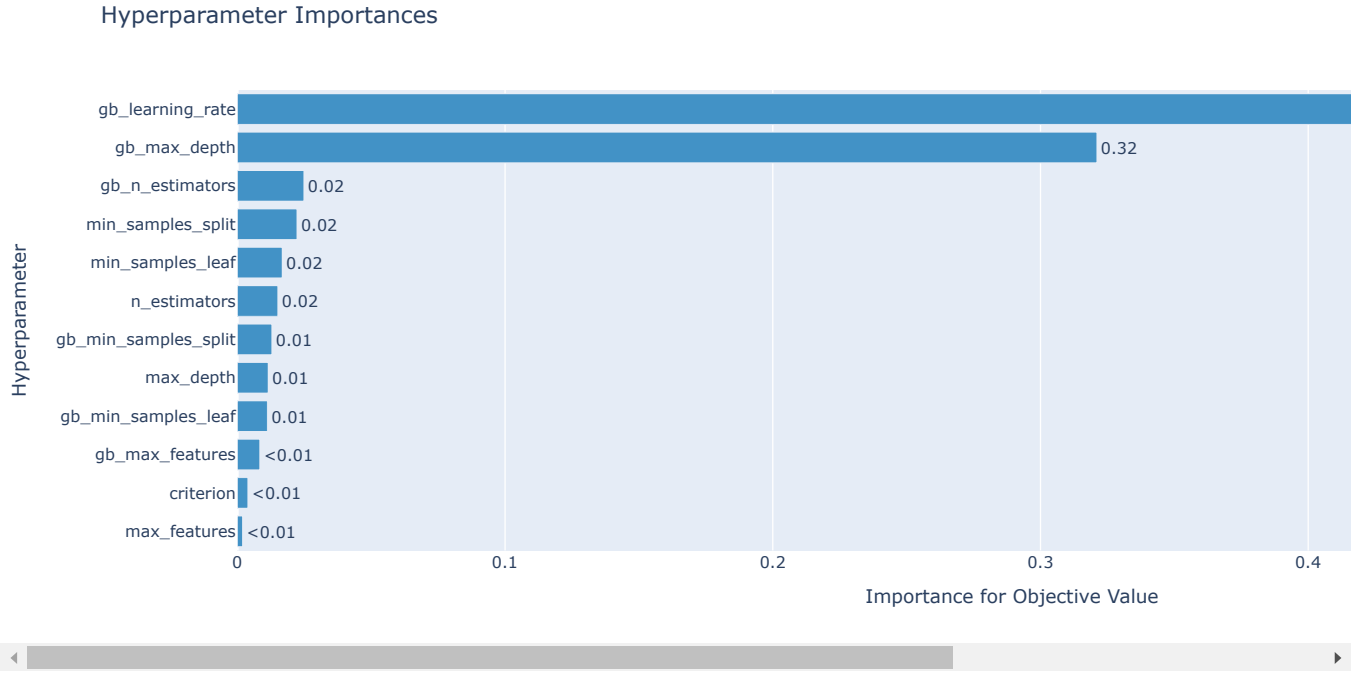
> Determines the contribution of each tree to the final ensemble. Smaller learning rates can lead to better generalization and more robust models, as they prevent overfitting by reducing the impact of individual trees. Higher learning rates can lead to faster convergence and improved training set performance, but they may also increase the risk of overfitting.

```
!pip install optuna
import optuna


best_model = None
best_params = None
best_accuracy = 0.0

def objective(trial):
    global best_model
    global best_params
    global best_accuracy

    # Define the hyperparameter search spaces for the first model (Random Forest)
    rf_params = {
        'n_estimators': trial.suggest_int('n_estimators', 100, 500, step=100),
        'max_depth': trial.suggest_int('max_depth', 5, 20),
        'min_samples_split': trial.suggest_int('min_samples_split', 2, 10),
        'min_samples_leaf': trial.suggest_int('min_samples_leaf', 1, 10),
        'max_features': trial.suggest_categorical('max_features', ['sqrt', 'log2']),
        'criterion': trial.suggest_categorical('criterion', ['gini', 'entropy'])
    }

    rf_model = best_models[0]
    rf_model.set_params(**rf_params)


    # Define the hyperparameter search spaces for the second model (Gradient Boosting)
    gb_params = {
    'n_estimators': trial.suggest_int('gb_n_estimators', 100, 500, step=100),
    'learning_rate': trial.suggest_loguniform('gb_learning_rate', 0.001, 0.1),
    'max_depth': trial.suggest_int('gb_max_depth', 3, 10),
    'min_samples_split': trial.suggest_int('gb_min_samples_split', 2, 10),
    'min_samples_leaf': trial.suggest_int('gb_min_samples_leaf', 1, 10),
    'max_features': trial.suggest_categorical('gb_max_features', ['sqrt', 'log2']),
}

    gb_model = best_models[1]
    gb_model.set_params(**gb_params)

    # Train and evaluate both models
    rf_model.fit(X_train, y_train)
    gb_model.fit(X_train, y_train)

    rf_accuracy = rf_model.score(X_dev, y_dev)
    gb_accuracy = gb_model.score(X_dev, y_dev)

    # Update the best model and parameters if necessary
    if rf_accuracy > best_accuracy:
        best_model = rf_model
        best_params = rf_params
        best_accuracy = rf_accuracy
    if gb_accuracy > best_accuracy:
        best_model = gb_model
        best_params = gb_params
        best_accuracy = gb_accuracy

    # Return the maximum accuracy of the two models
    return max(rf_accuracy, gb_accuracy)

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)
```

```
  'learning_rate': trial.suggest_loguniform('gb_learning_rate', 0.001, 0.1),
[I 2023-05-21 19:50:03,273] Trial 63 finished with value: 0.8465068087625814 and parameters: {'n_estimators': 500, 'max_depth':
<ipython-input-19-0cab925a892e>:26: FutureWarning: suggest_loguniform has been deprecated in v3.0.0. This feature will be remove
  'learning_rate': trial.suggest_loguniform('gb_learning_rate', 0.001, 0.1),
[I 2023-05-21 19:51:28,317] Trial 64 finished with value: 0.848283007696862 and parameters: {'n_estimators': 500, 'max_depth': 1
<ipython-input-19-0cab925a892e>:26: FutureWarning: suggest_loguniform has been deprecated in v3.0.0. This feature will be remove
  'learning_rate': trial.suggest_loguniform('gb_learning_rate', 0.001, 0.1),
[I 2023-05-21 19:53:03,290] Trial 65 finished with value: 0.8501332149200711 and parameters: {'n_estimators': 500, 'max_depth':
<ipython-input-19-0cab925a892e>:26: FutureWarning: suggest_loguniform has been deprecated in v3.0.0. This feature will be remove
  'learning_rate': trial.suggest_loguniform('gb_learning_rate', 0.001, 0.1),
[I 2023-05-21 19:54:37,920] Trial 66 finished with value: 0.8441385435168739 and parameters: {'n_estimators': 500, 'max_depth':
<ipython-input-19-0cab925a892e>:26: FutureWarning: suggest_loguniform has been deprecated in v3.0.0. This feature will be remove
  'learning_rate': trial.suggest_loguniform('gb_learning_rate', 0.001, 0.1),
[I 2023-05-21 19:55:59,365] Trial 67 finished with value: 0.8474689165186501 and parameters: {'n_estimators': 400, 'max_depth':
<ipython-input-19-0cab925a892e>:26: FutureWarning: suggest_loguniform has been deprecated in v3.0.0. This feature will be remove
  'learning_rate': trial.suggest_loguniform('gb_learning_rate', 0.001, 0.1),
[I 2023-05-21 19:57:35,039] Trial 68 finished with value: 0.8480609828300769 and parameters: {'n_estimators': 500, 'max_depth':
<ipython-input-19-0cab925a892e>:26: FutureWarning: suggest_loguniform has been deprecated in v3.0.0. This feature will be remove
  'learning_rate': trial.suggest_loguniform('gb_learning_rate', 0.001, 0.1),
[I 2023-05-21 19:58:18,943] Trial 69 finished with value: 0.8276346950858496 and parameters: {'n_estimators': 500, 'max_depth':
<ipython-input-19-0cab925a892e>:26: FutureWarning: suggest_loguniform has been deprecated in v3.0.0. This feature will be remove
  'learning_rate': trial.suggest_loguniform('gb_learning_rate', 0.001, 0.1),
[I 2023-05-21 19:59:34,395] Trial 70 finished with value: 0.8370337477797514 and parameters: {'n_estimators': 300, 'max_depth':
<ipython-input-19-0cab925a892e>:26: FutureWarning: suggest_loguniform has been deprecated in v3.0.0. This feature will be remove
  'learning_rate': trial.suggest_loguniform('gb_learning_rate', 0.001, 0.1),
[I 2023-05-21 20:01:12,455] Trial 71 finished with value: 0.8479869745411486 and parameters: {'n_estimators': 500, 'max_depth':
<ipython-input-19-0cab925a892e>:26: FutureWarning: suggest_loguniform has been deprecated in v3.0.0. This feature will be remove
  'learning_rate': trial.suggest_loguniform('gb_learning_rate', 0.001, 0.1),
[I 2023-05-21 20:02:49,664] Trial 72 finished with value: 0.8511693309650681 and parameters: {'n_estimators': 500, 'max_depth':
<ipython-input-19-0cab925a892e>:26: FutureWarning: suggest_loguniform has been deprecated in v3.0.0. This feature will be remove
  'learning_rate': trial.suggest_loguniform('gb_learning_rate', 0.001, 0.1),
[I 2023-05-21 20:04:25,556] Trial 73 finished with value: 0.8519094138543517 and parameters: {'n_estimators': 500, 'max_depth':
<ipython-input-19-0cab925a892e>:26: FutureWarning: suggest_loguniform has been deprecated in v3.0.0. This feature will be remove
  'learning_rate': trial.suggest_loguniform('gb_learning_rate', 0.001, 0.1),
[I 2023-05-21 20:06:01,831] Trial 74 finished with value: 0.8515393724097099 and parameters: {'n_estimators': 500, 'max_depth':
<ipython-input-19-0cab925a892e>:26: FutureWarning: suggest_loguniform has been deprecated in v3.0.0. This feature will be remove
  'learning_rate': trial.suggest_loguniform('gb_learning_rate', 0.001, 0.1),
[I 2023-05-21 20:07:36,734] Trial 75 finished with value: 0.8507992895204263 and parameters: {'n_estimators': 500, 'max_depth':
<ipython-input-19-0cab925a892e>:26: FutureWarning: suggest_loguniform has been deprecated in v3.0.0. This feature will be remove
  'learning_rate': trial.suggest_loguniform('gb_learning_rate', 0.001, 0.1),
[I 2023-05-21 20:09:10,300] Trial 76 finished with value: 0.8499851983422143 and parameters: {'n_estimators': 500, 'max_depth':
```

Visualization of hyperparamter tuning

```
from optuna.visualization import plot_optimization_history, plot_slice, plot_param_importances
plot_optimization_history(study)
# Visualize the hyperparameter slice
plot_slice(study)
# Visualize the importance of hyperparameters
plot_param_importances(study)
```



Explanation of model performance after hyperparamter tuning:

> We can clearly see that learning rate is the most important hyperparameter for increasing improvement of model performance, strongly followed by the second most important hyperparamter, max_depth.

5.2) Selecting best classifier

```
print(f'\nSelected best model is')
print(best_model)
```

```
Selected best model is
GradientBoostingClassifier(learning_rate=0.09759345772168124, max_depth=10,
                           max_features='sqrt', min_samples_leaf=10,
                           min_samples_split=6, n_estimators=500)
```

Evaluation on dev-set of best model with best hyperparamters

```
# Refit best model with best hyperparameters
best_model.set_params(**best_params)
best_model.fit(X_train, y_train)
y_pred_dev = best_model.predict(X_dev)
```

```
Best Hyperparameters  : {'n_estimators': 500, 'learning_rate': 0.09759345772168124, 'max_depth': 10, 'min_samples_split': 6, 'min_s
Accuracy              : 0.8522794552989935
Weighted F1-score     : 0.8425298394118671
classification report :
               precision    recall  f1-score   support

           0       0.54      0.32      0.40      1270
           1       0.81      0.80      0.81      3373
           2       0.89      0.95      0.92      8869

    accuracy                           0.85     13512
   macro avg       0.75      0.69      0.71     13512
weighted avg       0.84      0.85      0.84     13512

confusion matrix:
 [[ 410  330  530]
 [ 186 2713  474]
 [ 168  308 8393]]
```

```
print(f'Best Hyperparameters  : {best_params}')
print(f'Accuracy              : {accuracy_score(y_dev, y_pred_dev)}')
print(f'Weighted F1-score     : {f1_score(y_dev, y_pred_dev ,average= "weighted")}')
print("classification report :\n", classification_report(y_dev, y_pred_dev, target_names=target_names))
print(f'confusion matrix:\n {confusion_matrix(y_dev, y_pred_dev)}')
```

Explanation of model performance after hyperparamter tuning:

> After hyperparameter tuning, the Gradient Boost Classifier achieved the best performance on the dev-set with an overall great increasing in all metrices.

6) Final evaluation on test set

```
#evaluate best model on test set
y_test_pred = best_model.predict(X_test)
print(f'Final evaluation of {best_model} on test data\n')
print(f'Accuracy on test data              : {accuracy_score(y_test, y_test_pred)}')
print(f'Weighted F1-score on test data is  : {f1_score(y_test, y_test_pred ,average= "weighted")}')
print("classification report:\n", classification_report(y_test, y_test_pred, target_names=target_names))
```

```
Final evaluation of GradientBoostingClassifier(learning_rate=0.09759345772168124, max_depth=10,
                           max_features='sqrt', min_samples_leaf=10,
                           min_samples_split=6, n_estimators=500) on test data

Accuracy on test data              : 0.8442865600947306
Weighted F1-score on test data is  : 0.8330359935714399
classification report:
               precision    recall  f1-score   support

           0       0.49      0.28      0.36      1262
           1       0.80      0.79      0.80      3380
           2       0.89      0.94      0.91      8870
```

```
              accuracy                           0.84     13512
             macro avg       0.73      0.67      0.69     13512
          weighted avg       0.83      0.84      0.83     13512
```

Final analysis of performance on test-data:

> The best model, GradientBoostingClassifier(learning_rate=0.09759345772168124, max_depth=10,max_features='sqrt', min_samples_leaf=10,min_samples_split=6, n_estimators=500) , was evaluated on the test data. The test-set accuracy of the model was found to be 0.844, indicating that it performed well on real, unseen data. Not surprisingly the highest F1-score has been achieved for class 2, then class 1 and F1-score for class 0 is reasonable as well, which has least number of samples also in test-set Analyzing the classification report, we observe that the model exhibited varying performance for each class on the test data. The performance on the test data aligned with the performance on the dev-set, confirming the generalizability of our model.