COMPLEX ENGINEERING PROBLEM
MACHINE LEARNING
(CS-324)

SUBMITTED BY:	RIMSHA ISHTAQ (CS-20039) HABIBA ASIF (CS-20045) ALAINA (CS-20056)
COURSE:	MACHINE LEARNING
CODE:	CS-324
BATCH	2020
YEAR	THIRD YEAR
SUBMITTED TO:	MS. MEHWISH RAZA

DEPARTMENT OF COMPUTER & INFORMATION SYSTEMS ENGINEERING
BACHELORS IN COMPUTER SYSTEMS ENGINEERING
Course Code: CS-324
Course Title: Machine Learning
Complex Engineering Problem
TE Batch 2020, Spring Semester 2023
Grading Rubric
TERM PROJECT
Group Members:

Student No.	Name	Roll No.
S1	**Rimsha Ishtiaq**	**CS-20039**
S2	**Habiba Asif**	**CS-20045**
S3	**Alaina**	**CS-20056**

CRITERIA AND SCALES				Marks Obtained		
				S1	**S2**	**S3**
Criterion 1: Does the application meet the desired specifications and produce the desired outputs? (CPA-1, CPA-2, CPA-3) **[8 marks]**						
1	2	3	4			
The application does not meet the desired specifications and is producing incorrect outputs.	The application partially meets the desired specifications and is producing incorrect or partially correct outputs.	The application meets the desired specifications but is producing incorrect or partially correct outputs.	The application meets all the desired specifications and is producing correct outputs.			
Criterion 2: How well is the code organization? **[2 marks]**						
1	2	3	4			
The code is poorly organized and very difficult to read.	The code is readable only to someone who knows what it is supposed to be doing.	Some part of the code is well organized, while some part is difficult to follow.	The code is well organized and very easy to follow.			
Criterion 3: Does the report adhere to the given format and requirements? **[6 marks]**						
1	2	3	4			
The report does not contain the required information and is formatted poorly.	The report contains the required information only partially but is formatted well.	The report contains all the required information but is formatted poorly.	The report contains all the required information and completely adheres to the given format.			
Criterion 4: How does the student performed individually and as a team member? (CPA-1, CPA-2, CPA-3) **[4 marks]**						
1	2	3	4			
The student did not work on the assigned task.	The student worked on the assigned task, and accomplished goals partially.	The student worked on the assigned task, and accomplished goals satisfactorily.	The student worked on the assigned task, and accomplished goals beyond expectations.			

Final Score = (Criterial_1_score x 2) + (Criteria_2_score / 2) + (Criteria_3_score x (3/2)) + (Criteria_4_score)
= ______________________

Table of Contents

Introduction:

The problem given on hand is to explore and apply all the machine learning techniques from cleaning data to visualizing the results of multiple models for comparison of best models for the given dataset. The main key features of the dataset given are, the data has more than 30000 examples and the features are multiple also the output is supposed to be generated in multiclass. For this pupose we have applied all the techniques to preprocessed data, scale the data, model training and visualizing the models as following:

Preprocessing:

Preprocessing refers to the steps taken to clean, transform, and prepare raw data before it can be used for analysis or modeling. Here are some short steps for preprocessing data:

1. Identifying Features and Target

- Import the necessary libraries: Start by importing the required libraries such as pandas, NumPy, or scikit-learn, which are commonly used for data preprocessing tasks.
- Load the data: Read the raw data into programming environment using the appropriate functions or methods. The data was in CSV format.
- Initially 9 features were identified including:
- Summary was taken as target which consisted of 27 number of classes.

2. Handle missing values:

- There are no missing values for except for ‘Precip Type’ which has 517 missing values.
- As 'Precip Type' is a categorical variable, so we can't fill missing values with mean/med/mode but fill based on the assumption that adjacent observations are similar to one another using ‘ffill’ method.

```
non_null_dataset=dataset.fillna(method='ffill')
non_null_dataset
```

```
dataset.isnull().sum()

Formatted Date               0
Summary                      0
Precip Type                517
Temperature (C)              0
Apparent Temperature (C)     0
Humidity                     0
Wind Speed (km/h)            0
Wind Bearing (degrees)       0
Visibility (km)              0
Loud Cover                   0
Pressure (millibars)         0
Daily Summary                0
dtype: int64
```

3. Encoding categorical variables: Convert categorical variables into numerical representations that can be understood by machine learning algorithms.

- This was done on 'PrecipType' categorical variable using label encoding.
- Target was remained as categorical.

4. Feature engineering: Created new features or transformed existing ones to enhance the predictive power of the data.

- Collinearity test was carried out on features which resulted in removing 'Apparent temperature' feature which was causing collinearity.
- Grouping suitable classes; grouped classes into 4 namely, Partly Cloudy, Mostly Cloudy, Overcast, Non-Cloudy
- Feature Scaling using standardization

5. Split the data: Split the preprocessed data into training and testing sets. The training set is used to build the model, while the testing set is used to evaluate its performance which ensures an unbiased evaluation of the model's capabilities.

Model Implementation:

The models selected for implementation includes:

- Knn (non-parametric)
- Logistic Regression (parametric)
- ANN

Best Model Selection:

As the model are implemented, their hyperparameter selection is not an easy task. For that k-fold cross validation is used to ensure that the best hyperparameters according to the chosen evaluation metric are selected.

KNeighborsClassifier (KNN):

The K-Nearest Neighbors (KNN) algorithm is a non-parametric and instance-based classification algorithm. It does not involve explicit model training but instead stores the entire training dataset in memory for prediction. KNN makes predictions based on the similarity or proximity between instances in the feature space.

KNN classifier with its possible parameters is defined below:

Although, there are many hyperparameters, we choose to tune the following: n_neighbors, p , weights.

KNeighborsClassifier(*n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None*)

GridSearch on KNN:

```
# Define the hyperparameters and their possible values
hyperparameters = {
    'n_neighbors': np.arange(1,30,2) ,
    'weights': ['uniform', 'distance'],
    'p': [1, 2]
}
```

In order to choose the best value for the hyperparameters, we applied the Grid search technique.

Values of the hyperparameters are defined below:

- n_neigbors:

It is a hyperparameter that determines the number of neighbors to consider when making predictions for a new instance. Its default value is 5.

By hit-and-trial, we take range as (1,30,2)

- weights:

  It specifies how the neighbor's contributions are weighted when making predictions.

  Its possible values are: 'distance' and 'uniform'.

  ✓ **distance:** weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.

  ✓ **uniform:** uniform weights. All points in each neighborhood are weighted equally.

- p:

  Power parameter for the Minkowski metric.

  Its possible values are: 1 and 2

  ✓ p=1: Manhattan distance

  ✓ p=2: Euclidean distance

This Grid Search selects the following values as the best values for the hyperparameters mentioned above:

```
Best Hyperparameters:{'n_neighbors': 29, 'p': 1, 'weights': 'distance'}
```

Implementing the model as Function:

A function is defined as 'knn_classifier', in which KNN model is defined and trained.

This function calculates the roc_auc_curves value for one-vs-rest and one-vs-one techniques.

This also gives us the confusion matrix, precision and recall values for the model.

KNN Model 1 (Best model):

In KNN model 1, we use the best values of the hyperparameters obtained by Grid Search.

```
ROC AUC score:
 Train Set:
ovr:  0.81   ovo:  0.82
ROC AUC score:
 Test Set:
ovr:  0.79   ovo:  0.8

confusion_matrix:
 [[2553  125  849 2310]
 [ 397 1896  203 1151]
 [1197   53 1609  523]
 [1591  266  346 4222]]
```

```
precision_score:  [0.44 0.81 0.54 0.51]
recall_score:  [0.44 0.52 0.48 0.66]
               precision    recall  f1-score   support

Mostly Cloudy       0.44      0.44      0.44      5837
        Other       0.81      0.52      0.63      3647
     Overcast       0.54      0.48      0.50      3382
Partly Cloudy       0.51      0.66      0.58      6425

     accuracy                           0.53     19291
    macro avg       0.58      0.52      0.54     19291
 weighted avg       0.55      0.53      0.53     19291

accuracy 0.53
classificationscore(testset):  0.53
classificationscore(trainset):  0.56
```

KNN Model 2:

In this the combination of other values for the hyperparameters is used:

```
ROC AUC score:
 Train Set:
ovr:  0.86   ovo:  0.87
ROC AUC score:
 Test Set:
ovr:  0.81   ovo:  0.81

confusion_matrix:
 [[2899  270  635 1873]
 [ 411 2103  189  950]
 [1227  234 1621  392]
 [1698  472  294 4023]]
```

```
precision_score:  [0.48 0.71 0.61 0.55]
recall_score:  [0.5  0.58 0.46 0.66]
               precision    recall  f1-score   support

Mostly Cloudy       0.48      0.50      0.49      5677
        Other       0.71      0.58      0.64      3653
     Overcast       0.61      0.46      0.53      3474
Partly Cloudy       0.55      0.66      0.60      6487

     accuracy                           0.56     19291
    macro avg       0.59      0.55      0.56     19291
 weighted avg       0.57      0.56      0.56     19291

accuracy 0.56
classificationscore(testset):  0.56
classificationscore(trainset):  0.61
```

KNN Model 3:

In this the combination of other values for the hyperparameters is used:

```
ROC AUC score:
 Train Set:
ovr:  0.85   ovo:  0.86
ROC AUC score:
 Test Set:
ovr:  0.81   ovo:  0.82

confusion_matrix:
 [[2823  244  602 2008]
 [ 347 2125  170 1011]
 [1196  225 1613  440]
 [1556  415  263 4253]]
```

```
precision_score:  [0.48 0.71 0.61 0.55]
recall_score:  [0.5  0.58 0.46 0.66]
               precision    recall  f1-score   support

Mostly Cloudy       0.48      0.50      0.49      5677
        Other       0.71      0.58      0.64      3653
     Overcast       0.61      0.46      0.53      3474
Partly Cloudy       0.55      0.66      0.60      6487

     accuracy                           0.56     19291
    macro avg       0.59      0.55      0.56     19291
 weighted avg       0.57      0.56      0.56     19291

accuracy 0.56
classificationscore(testset):  0.56
classificationscore(trainset):  0.61
```

Decision Tree:

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.

DecisionTree Classifier with its possible hyperparameters is defined below:

```
DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0)
```

Although, there are many hyperparameters, we choose to tune the following: max_depth, min_samples_split, min_samples_leaf, criterion.

GridSearch on Decision Tree Classifier

```python
# Define the hyperparameters to tune
param_grid = {
    'max_depth': [None, 5, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],

    'criterion': ['gini', 'entropy']
}
```

In order to choose the best value for the hyperparameters , we applied the Grid search technique.

Values of the hyperparameters are defined below:

- max_depth:
  The maximum depth of the tree. It limits the number of nodes and splits in the tree, helping to control overfitting. Options include are: None, 5 and 10
- min_samples_split:
  The minimum number of samples required to split an internal node. Options include are: [2,5,10]
- min_samples_leaf:
  The minimum number of samples required to be at a leaf node. Options include are: [1,2,4]
- criterion
  The function used to measure the quality of a split. Common criteria are "gini" for Gini impurity and "entropy" for information gain.

DecisionTreeClassifier Model 1 (Best model):

In model 1, we use the best values of the hyperparameters obtained by Grid Search.

```
ROC AUC score:
 Train Set:
ovr:   0.81    ovo:   0.82
ROC AUC score:
 Test Set:
ovr:   0.81    ovo:   0.82

confusion_matrix:
 [[2853  116  778 1930]
 [ 389 1982  180 1102]
 [1239   38 1743  454]
 [1806  246  353 4082]]

precision_score:   [0.45 0.83 0.57 0.54]
recall_score:   [0.5   0.54 0.5   0.63]
               precision    recall  f1-score   support

Mostly Cloudy       0.45      0.50      0.48      5677
        Other       0.83      0.54      0.66      3653
     Overcast       0.57      0.50      0.53      3474
Partly Cloudy       0.54      0.63      0.58      6487

     accuracy                           0.55     19291
    macro avg       0.60      0.54      0.56     19291
 weighted avg       0.58      0.55      0.56     19291

accuracy 0.55
classificationscore(testset):   0.55
classificationscore(trainset):   0.55
```

DecisionTreeClassifier Model 2:

In this the combination of other values for the hyperparameters is used:

```
ROC AUC score:
 Train Set:
ovr:   0.81    ovo:   0.82
ROC AUC score:
 Test Set:
ovr:   0.79    ovo:   0.8

confusion_matrix:
 [[2533  145  655 2344]
 [ 352 1962  170 1169]
 [1300   49 1524  601]
 [1574  306  312 4295]]

precision_score:   [0.44 0.8  0.57 0.51]
recall_score:   [0.45 0.54 0.44 0.66]
               precision    recall  f1-score   support

Mostly Cloudy       0.44      0.45      0.44      5677
        Other       0.80      0.54      0.64      3653
     Overcast       0.57      0.44      0.50      3474
Partly Cloudy       0.51      0.66      0.58      6487

     accuracy                           0.53     19291
    macro avg       0.58      0.52      0.54     19291
 weighted avg       0.56      0.53      0.54     19291

accuracy 0.53
classificationscore(testset):   0.53
classificationscore(trainset):   0.56
```

DecisionTreeClassifier Model 3:

In this the combination of other values for the hyperparameters is used:

```
ROC AUC score:
 Train Set:
ovr:   0.97    ovo:   0.97
ROC AUC score:
 Test Set:
ovr:   0.74    ovo:   0.75

confusion_matrix:
 [[2819  389  874 1595]
 [ 480 2316  160  697]
 [1108  167 1781  418]
 [1935  828  501 3223]]

precision_score:   [0.44 0.63 0.54 0.54]
recall_score:   [0.5   0.63 0.51 0.5 ]
               precision    recall  f1-score   support

Mostly Cloudy       0.44      0.50      0.47      5677
        Other       0.63      0.63      0.63      3653
     Overcast       0.54      0.51      0.52      3474
Partly Cloudy       0.54      0.50      0.52      6487

     accuracy                           0.53     19291
    macro avg       0.54      0.54      0.54     19291
 weighted avg       0.53      0.53      0.53     19291

accuracy 0.53
classificationscore(testset):   0.53
classificationscore(trainset):   0.83
```

Logistic Regression:

Logistic regression is a statistical model used for binary classification problems, where the dependent variable takes two possible values.Logistic regression can be extended to handle multiclass classification problems through various strategies, such as one-vs-rest (also known as one-vs-all).

The goal of logistic regression is to estimate the probability of the dependent variable belonging to a specific class based on the values of the independent variables.

Logistic Regression with its possible hyperparameters is defined below:

Although, there are many hyperparameters, we choose to tune the following: C, penalty, solver.

```
LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, cl
ass_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=
```

GridSearch on Logistic Regression:

In order to choose the best value for the hyperparameters , we applied the Grid search technique.

Values of the hyperparameters are defined below:

```
# Define the hyperparameters to tune
param_grid = {
    'penalty': ['l2'],
    'C': [0.1,0.001, 0.01],
    'solver': ['lbfgs', 'newton-cg']
}
```

- C:
  This is Inverse of regularization strength; must be a positive float and smaller values specify stronger regularization.
  By some research and hit-and-trial, we select 0.1, 1.0 and 10.0 as the options.
- penalty:
  The penalty hyperparameter specifies the type of regularization used in multiclass logistic regression. Options include":
  - ✓ 'l1' (L1 regularization)
  - ✓ 'l2' (L2 regularization)
  - ✓ 'elasticnet' (combination of L1 and L2 regularization)
  - ✓ 'none' (no regularization).

  But because of the compatibility issue we select only ‘l2’
- solver:
  The solver algorithm determines the optimization algorithm used to estimate the coefficients in multinomial logistic regression.
  Common choices include 'newton-cg', 'lbfgs', 'sag', and 'saga'.
  Because l2 is only compatible with ‘lbfgs’ and ‘newton-cg’, we select these as the options.

This Grid Search selects the following values as the best values for the hyperparameters mentioned above: In addition to these hyperparameters, we set:

```
Best Hyperparameters:{ {'C': 10.0, 'penalty': 'l2', 'solver': 'newton-
cg'}
```

multi-class = auto and max-iter = 1000

Logistic Regression Model 1 (Best model):

In model 1, we use the best values of the hyperparameters obtained by Grid Search.

```
ROC AUC score:
 Train Set:
ovr:  0.86   ovo:  0.87
ROC AUC score:
 Test Set:
ovr:  0.8   ovo:  0.81

confusion_matrix:
 [[2861  257  642 1958]
 [ 430 2127  185  917]
 [1137  210 1650  437]
 [1695  491  293 4001]]

precision_score:  [0.47 0.69 0.6  0.55]
recall_score:  [0.5  0.58 0.48 0.62]
               precision    recall  f1-score   support

Mostly Cloudy       0.47      0.50      0.48      5718
        Other       0.69      0.58      0.63      3659
     Overcast       0.60      0.48      0.53      3434
Partly Cloudy       0.55      0.62      0.58      6480

     accuracy                           0.55     19291
    macro avg       0.57      0.54      0.56     19291
 weighted avg       0.56      0.55      0.55     19291

accuracy 0.55
classificationscore(testset):  0.55
classificationscore(trainset):  0.62
```

Logistic Regression Model 2:

In this the combination of other values for the hyperparameters is used:

```
ROC AUC score:
 Train Set:
ovr:  0.72   ovo:  0.72
ROC AUC score:
 Test Set:
ovr:  0.72   ovo:  0.72

confusion_matrix:
 [[2074  571  566 2466]
 [ 360 1713  346 1234]
 [1181  695 1103  495]
 [1471  503  279 4234]]

precision_score:  [0.41 0.49 0.48 0.5 ]
recall_score:  [0.37 0.47 0.32 0.65]
               precision    recall  f1-score   support

Mostly Cloudy       0.41      0.37      0.39      5677
        Other       0.49      0.47      0.48      3653
     Overcast       0.48      0.32      0.38      3474
Partly Cloudy       0.50      0.65      0.57      6487

     accuracy                           0.47     19291
    macro avg       0.47      0.45      0.45     19291
 weighted avg       0.47      0.47      0.46     19291

accuracy 0.47
classificationscore(testset):  0.47
classificationscore(trainset):  0.47
```

Logistic Regression Model 3:

In this the combination of other values for the hyperparameters is used:

```
ROC AUC score:
 Train Set:
ovr:  0.72   ovo:  0.72
ROC AUC score:
 Test Set:
ovr:  0.72   ovo:  0.72

confusion_matrix:
 [[2071  570  572 2464]
 [ 363 1712  346 1232]
 [1174  694 1112  494]
 [1475  504  282 4226]]

precision_score:  [0.41 0.49 0.48 0.5 ]
recall_score:  [0.36 0.47 0.32 0.65]
               precision    recall  f1-score   support

Mostly Cloudy       0.41      0.36      0.38      5677
        Other       0.49      0.47      0.48      3653
     Overcast       0.48      0.32      0.38      3474
Partly Cloudy       0.50      0.65      0.57      6487

     accuracy                           0.47     19291
    macro avg       0.47      0.45      0.45     19291
 weighted avg       0.47      0.47      0.46     19291

accuracy 0.47
classificationscore(testset):  0.47
classificationscore(trainset):  0.47
```

Multiple Layer Perceptron (MLP):

The Multilayer Perceptron (MLP) is a feedforward artificial neural network that consists of multiple layers of interconnected neurons. It is a versatile model widely used for both regression and classification tasks.

MLP with its possible hyperparameters is defined below:

MLPClassifier(*hidden_layer_sizes=(100,), activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=Non*

As we can see there are many hyperparameters, we choose to tune the following: 'hidden_layer', 'activation', 'solver', 'alpha', 'learning_rate'

GridSearch on MLP Classifier:

```
# Define the parameter grid for GridSearchCV
param_grid = {
    'hidden_layer_sizes': [(100,), (100, 50), (50, 50)],
    'activation': ['relu', 'tanh'],
    'solver': ['adam', 'sgd'],
    'alpha': [0.0001, 0.001, 0.01],
    'learning_rate': ['constant', 'invscaling', 'adaptive']
}
```

In order to choose the best value for the hyperparameters , we applied the Grid search technique.

Values of the hyperparameters are defined below:

- hidden_layer:

  MLP can have one or more hidden layers between the input and output layers. Each hidden layer consists of multiple neurons that perform nonlinear transformations on the input data.
  Here we added two hidden layers, and by hit-and-trial method take the options for the values as:
  (100,) , (100,50), (50,50)
- activation:

  Activation functions introduce nonlinearity to the MLP, enabling it to learn complex patterns and relationships in the data. Following activation functions are taken for the model

  ✓ relu: the rectified linear unit function, returns f(x) = max(0, x)
  ✓ tanh: the hyperbolic tan function, returns f(x) = tanh(x).
- solver:

This hyperparameter refers to the optimization algorithm used to train the neural network by minimizing the loss function. The solver determines how the weights are updated during the learning process.

The solver hyperparameter options we used are:

- ✓ sgd (Stochastic Gradient Descent): This solver performs updates to the model's weights based on a randomly selected subset of training samples (mini-batches).
- ✓ Adam (Adaptive Moment Estimation): It adapts the learning rates for each parameter based on estimates of the first and second moments of the gradients.

- alpha:

  It controls the regularization strength applied to the neural network model.
  Values selected are: [0.0001, 0.001, 0.01]
  All these values are selected by some research and hit-and-trial method.
- learning_rate:

  It determines the step size taken during the optimization process to update the weights of the neural network.

  The different options are as follows:

  - ✓ constant: this is a constant learning rate given by 'learning_rate_init'.
  - ✓ invscaling: It gradually decreases the learning rate at each time step 't' using an inverse scaling exponent of 'power_t'.

    effective_learning_rate = learning_rate_init / pow(t, power_t)

  - ✓ adaptive: It keeps the learning rate constant to 'learning_rate_init' as long as training loss keeps decreasing.

<u>MLP Classifier Model 1 (Best model):</u>

In model 1, we use the best values of the hyperparameters obtained by Grid Search.

```
ROC AUC score:
 Train Set:
ovr:  0.85   ovo:  0.86
ROC AUC score:
 Test Set:
ovr:  0.83   ovo:  0.84

confusion_matrix:
 [[2539  165  996 2018]
 [ 246 2122  211 1080]
 [ 873   77 2079  405]
 [1379  425  420 4256]]

precision_score:  [0.5  0.76 0.56 0.55]
recall_score:  [0.44 0.58 0.61 0.66]
               precision    recall  f1-score   support

Mostly Cloudy       0.50      0.44      0.47      5718
        Other       0.76      0.58      0.66      3659
     Overcast       0.56      0.61      0.58      3434
Partly Cloudy       0.55      0.66      0.60      6480

     accuracy                           0.57     19291
    macro avg       0.59      0.57      0.58     19291
 weighted avg       0.58      0.57      0.57     19291

accuracy 0.57
classificationscore(testset):  0.57
classificationscore(trainset):  0.6
```

<u>MLP Classifier Model 2:</u>

In this the combination of other values for the hyperparameters is used:

```
ROC AUC score:
 Train Set:
ovr:  0.85   ovo:  0.86
ROC AUC score:
 Test Set:
ovr:  0.83   ovo:  0.84

confusion_matrix:
 [[2539  165  996 2018]
 [ 246 2122  211 1080]
 [ 873   77 2079  405]
 [1379  425  420 4256]]

precision_score:  [0.5  0.76 0.56 0.55]
recall_score:  [0.44 0.58 0.61 0.66]
               precision    recall  f1-score   support

Mostly Cloudy       0.50      0.44      0.47      5718
        Other       0.76      0.58      0.66      3659
     Overcast       0.56      0.61      0.58      3434
Partly Cloudy       0.55      0.66      0.60      6480

     accuracy                           0.57     19291
    macro avg       0.59      0.57      0.58     19291
 weighted avg       0.58      0.57      0.57     19291

accuracy 0.57
classificationscore(testset):  0.57
classificationscore(trainset):  0.6
```

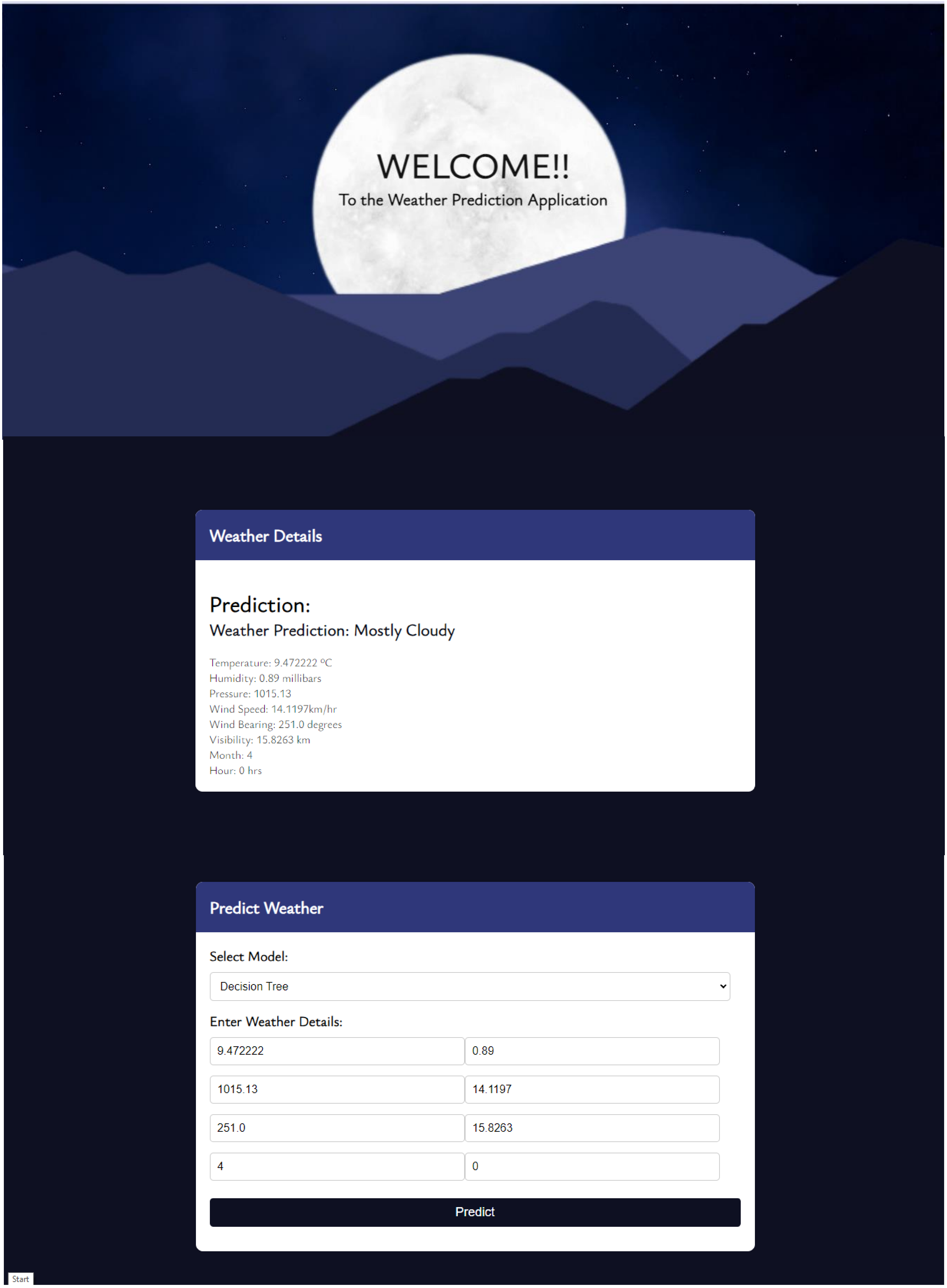
<u>MLP Classifier Model 3:</u>

In this the combination of other values for the hyperparameters is used:

```
ROC AUC score:
 Train Set:
ovr:  0.85   ovo:  0.86
ROC AUC score:
 Test Set:
ovr:  0.83   ovo:  0.84

confusion_matrix:
 [[2539  165  996 2018]
 [ 246 2122  211 1080]
 [ 873   77 2079  405]
 [1379  425  420 4256]]

precision_score:  [0.5  0.76 0.56 0.55]
recall_score:  [0.44 0.58 0.61 0.66]
               precision    recall  f1-score   support

Mostly Cloudy       0.50      0.44      0.47      5718
        Other       0.76      0.58      0.66      3659
     Overcast       0.56      0.61      0.58      3434
Partly Cloudy       0.55      0.66      0.60      6480

     accuracy                           0.57     19291
    macro avg       0.59      0.57      0.58     19291
 weighted avg       0.58      0.57      0.57     19291

accuracy 0.57
classificationscore(testset):  0.57
classificationscore(trainset):  0.6
```

User Interface:

For user interface, flask app is used to show the results



Tabular Comparison:

	Knn_model1	Knn_model2	Knn_model3	DecisonTree_model1	DecisonTree_model2	DecisonTree_model3	MLP_Classifier_model1	MLP_Classifier_model2	MLP_Classifier_model3
Train roc_auc score	0.82	0.87	0.86	0.82	0.82	0.97	0.86	.86	0.86
Test roc_auc score	0.82	0.81	0.82	0.82	0.80	0.75	0.84	0.84	0.84
Precision score	[0.45, 0.83, 0.57, 0.54]	[0.47, 0.69, 0.6, 0.55]	[0.48, 0.71, 0.61, 0.55]	[0.45, 0.83, 0.57, 0.54]	[0.44,0.8, 0.57, 0.51]	[0.44, 0.63, 0.54, 0.54]	[0.5, 0.76, 0.56, 0.55]	[0.5, 0.76, 0.56, 0.55]	[0.5, 0.76, 0.56, 0.55]
Recall_score	[0.5 0.54 0.5 0.63]	[0.47 0.69 0.6 0.55]	[0.5 0.58 0.46 0.66]	[0.5 0.54 0.5 0.63]	[0.45 0.54 0.44 0.66]	[0.5, 0.63, 0.51, 0.5]	[0.44, 0.58, 0.61, 0.66]	[0.44, 0.58, 0.61, 0.66]	[0.44, 0.58, 0.61, 0.66]
Accuracy	0.55	0.55	0.56	0.55	0.53	0.53	0.57	0.57	0.57
Test classification score	0.55	0.55	0.56	0.55	0.53	0.53	0.57	0.57	0.57
Train classification score	0.55	0.62	0.67	0.55	0.56	0.83	0.60	0.60	0.60

Graphical comparisons: