Bilkent University
Department of Electrical and Electronics Engineering

EEE 485 - Statistical Learning and Data Analytics
Project First Report
Spring 2021

Smart Weather Teller

Group 12 Members

Meltem Toprak 21502044

Şevki Gavrem Kulkuloğlu 21601793

April 2, 2021

1 INTRODUCTION

The purpose of this project is to solve a multiclass classification problem which is predicting weather conditions. To solve this problem, some supervised and unsupervised machine learning algorithms will be constructed and the US Weather Events (2016 - 2020) dataset will be used. Then, the performance of the classifiers will be evaluated and compared. Also, Python will be used as the programming language. Since the weather conditions affect human lives and plans directly, predicting weather for a certain date is significant. For instance, the renewable energy power plants have increased and they mostly depend on the weather condition. For this reason, many energy companies need daily, weekly, or monthly weather predictions to calculate an adequate amount of energy to sell daily. Otherwise, they get a penalty due to not selling energy properly so predicting weather has a significant role in the renewable energy sector. Similarly, the air traffic is affected from the weather events. Therefore, there are many areas that are directly dependent on the weather conditions.

2 DESCRIPTION OF DATASET

Dataset: US Weather Events (2016 - 2020) [1]

In this dataset, there are weather events from 49 states of the United States of America. It includes 6.3 million events in total. Weather events (labels) are severe-cold, fog, hail, rain, snow, storm and other precipications. Some events can be regarded as an extreme event like the storm while some events like rain and snow can be regarded as regular events. Data is collected from 2,071 airport-based weather stations across the USA from January 2016 to December 2020 [1].

The columns of the dataset are EventId, Type, Severity, StartTime(UTC), EndTime(UTC), TimeZone, AirportCode, LocationLat, LocationLng, City. It includes a total of 9 features with 6 strings, 2 decimals and 2 dates and 7 classes of weather events specified as Type. Also, it has 6274206 samples.

Shape of the data: 6274206 x 10 array(['Cold', 'Fog', 'Hail', 'Precipitation', 'Rain', 'Snow', 'Storm'], dtype=object) array([169182, 1385264, 2627, 96684, 3752341, 827555, 40553])

3 METHODOLOGY

Preprocess

The preprocess code of the data is taken from Kaggle [2]. In the implementation, the NaN values are checked first after uploading data as a data frame. According to this, it is observed that the 'City' feature has 9087, 'ZipCode' feature has 38685 NaN values in total in the dataset. The NaN values

are replaced using the mean of the column. The data is grouped into Series(columns) according to 'City' and 'Type'. The number of types depending on the cities are collected and they are merged with the data.'StartTime(UTC)' and 'EndTime(UTC)' features are converted into datetime object. In this way, 'Start_year', 'Start_month','Start_week', 'Start_weekday','Start_day', 'end_year', 'end_month', 'end_week', 'end_weekday', 'end_day' features are obtained from the datetime objects. Then, 'Type', 'StartTime(UTC)', 'EndTime(UTC)' are removed from the data and 'Type' is separated as labels [2]. Then, training and test data are separated and converted into numpy arrays. The labels which are in string type are converted to integers 1-7.

	EventId	Type	Severity	StartTime(UTC)	EndTime(UTC)	TimeZone	AirportCode	LocationLat	LocationLng	City	County	State	ZipCode
0	W-3339557	Snow	Light	2016-12-07 14:30:00	2016-12-07 19:56:00	US/Central	KEAU	44.8667	-91.4880	Eau Claire	Chippewa	WI	54703.0
1	W-5208237	Rain	Light	2016-01-07 16:56:00	2016-01-07 17:56:00	US/Pacific	KNLC	36.3024	-119.9397	Lemoore	Kings	CA	93245.0
2	W-4424597	Rain	Light	2017-09-04 23:15:00	2017-09-04 23:55:00	US/Central	KLUM	44.8936	-91.8665	Menomonie city	Dunn	WI	54751.0
3	W-2865395	Fog	Severe	2020-10-07 08:53:00	2020-10-07 09:53:00	US/Pacific	KMMV	45.1945	-123.1361	McMinnville	Yamhill	OR	97128.0
4	W-2661943	Fog	Severe	2016-11-26 12:50:00	2016-11-26 15:30:00	US/Eastern	K2G4	39.5827	-79.3418	Accident	Garrett	MD	21520.0

Figure 1: Raw data

We looked over our preprocess from start. Since we had 20 features in the original dataset, some redundant features are dropped. We have reduced our feature size to 5: 'Severity', 'LocationLat', 'LocationLng', 'Start_date', 'Duration'. Severity', 'LocationLat' and 'LocationLng' are directly coming from the raw dataset. Using 'StartTime(UTC)' and 'EndTime(UTC)' we drive out 'Start_date' and 'Duration' features. We deal with time in hours. Both of them converted to hours of the year. 'Start_date' is referring the starting hour of the weather event. 'Duration' refers to the duration of the event after it started. Labels are the same as previous (string type are converted to integers 1-7). Raw data is shuffled, 10k sampled dataset is taken from it and preprocessed.

	Severity	LocationLat	LocationLng	Start_date	Duration	Labels
0	1	44.8667	-91.4880	8198.500000	5.433333	Snow
1	1	36.3024	-119.9397	160.933333	1.000000	Rain
2	1	44.8936	-91.8665	5927.250000	0.666667	Rain
3	4	45.1945	-123.1361	6728.883333	1.000000	Fog
4	4	39.5827	-79.3418	7932.833333	2.666667	Fog

Figure 2: Preproccessed data

Principal Component Analysis (PCA)

PCA is a statistical procedure that allows you to summarize the large data with small indices which is easy to analyze and visualize[3]. That's the reason why we have using PCA to visualize our data. Main purpose is minimizing the data with minimum loss of information. We start with normalization of data to prevent big values to affect the result with more weight than other small values. Then, we calculate the correlation matrix. We can find the eigenvectors and eigenvalues from there[4]. Biggest eigenvalues are representing the most important corresponding eigenvectors, which become our principal components that represent the data. When we use n eigenvectors its stated as n-component PCA. Its performance of representing the data is calculated by explained variance. It is calculated by sum of corresponding n

eigenvalues of n eigenvectors to sum of all eigenvalues. In our project, we have used zero-mean and unit variance normalization. 2D and 3D plots have been plotted for visualization of data with first two and first three principal components.

Logistic Regression

The purpose of the Logistic Regression is to find a relation between features and probability of outcome.It's a classification algorithm, where the response variable is categorical. It fits to our a data by this perspective we have chosen this algorithm. Ordinal Logistic Regression is implemented in this project, since there are 7 possible responses [5] which are cold, fog, precipitation, rain, snow, storm. Since these are strings, we have represented them as integers.

We start with sigmoid(logistic) function, which gives output probability between 0 and 1.

$$sigmoid(z) = \frac{1}{1+e^{(-z)}} \tag{1}$$

Where the input of the sigmoid function (z), is the dot product of the data and weight matrix. Out goal is minimizing the loss function by updating weight matrix in each iteration, which is called fitting. Using the derivative of loss function, which is called gradient descent, we can decide this update of weights. Weight matrix is updated by subtracting learning rate time gradient descent from itself in each iteration. We are also updating the intercept point in each iteration by subtracting mean of difference between predict and target times learning rate from itself. After n iterations Logistic Regression model is trained, weight matrix and intercept point is obtained. Finally, to use the model, we take dot product of test data and weight matrix and sum with intercept point. The result of this is given as input to *sigmoid* function to obtain predicts. To measure the accuracy score, we is checking the predicts for classification boundaries. Iteration continues until weights converged.

K-Nearest Neighbors (KNN) Classifier

KNN classifier is a non-parametric supervised algorithm, therefore, it is flexible and easy to implement. It is based on observing the conditional probability of Y given X. Then, it predicts the classes with the largest probability obtained depending on the given observation [6]. In the implementation, the algorithm takes k as a parameter and observes k closest data points to the sample by looking at their labels. Then, it gives the mode of the labels (the most frequent label) as the prediction [7].

- k: Number of neighboring samples
- p: Minkowski distance given as follows

$$(D,Y) = (\sum_{i=1}^{n}(|x_i - y_i|)^p)^{1/p} \tag{2}$$

Moreover, selecting the correct k and p values for the algorithm is an important issue. As the value of k

increases the algorithm becomes more stable. However, using large k and p values makes the algorithm work slower especially in large datasets. The optimal k and p values are determined with a grid search method using validation data in our project [7].

K-Fold Cross Validation

K-Fold Cross Validation is used to evaluate the model with a specific dataset. It takes k as a parameter and randomly divides the dataset with n samples into k folds. The size of each fold becomes n/k. These folds are called validation sets. One of them is taken as test data and the other remaining data with size (n-n/k) becomes training data. In this way, the model is trained, tested and the accuracy score is obtained. This procedure is repeated until all the folds are tested. Then, accuracy scores are summed and divided into k. In this way, the validation score is obtained for the specific data and model [6].

K-Fold Cross Validation method is more accurate while evaluating the model since the dataset is shuffled and the accuracy is averaged [6].

Random Forest Classifier

Random Forest Classifier is the improved version of bagged trees. The implementation of the Random Forest Classifier depends on the decision trees. Multiple trees are built on "bootstrapped training samples" and they are trained [6]. In the evaluation of the splits, gini index is used as the cost function. Each tree gives an estimation. The final prediction is obtained using the subsamples of the training dataset. In the implementation, it takes the following parameters:

- n_trees: Number of trees
- max_depth:Maximum depth for each tree to stop splitting
- min_size:To determine leaf nodes for left and right nodes [8]
- sample_size:To determine subsample sizes
- n_feats: Depth of each tree

As an advantage of this classifier, since multiple trees are involved in the prediction procedure, more accurate results are obtained. Also, single decision trees are inclined to overfitting, however, Random Forest prevents it. Therefore, its accuracy is high compared to a decision tree [8]. The implementation of the Random Forest Classifier is inspired from Machine Learning Mastery website[9] and some parts of the code is taken from the same website.

Evaluation of the Algorithms

For evaluation, a multiclass confusion matrix are created for each algorithm. Based on these confusion matrices, true positives (TP), false positives (FP), true negatives (TN), false negatives (FN) can be determined. Then, accuracy, precision, recall score metrics can be found below [10]. Also, the K-Fold Cross Validation results plotted based on accuracy to compare the algorithms.

$$Accuracy = \frac{TN+TP}{TN+TP+FN+FP} \tag{3}$$

$$P = \frac{TP}{TP+FP} \tag{4}$$

$$R = \frac{TP}{TP+FN} \tag{5}$$

The precision, recall score metrics are important since sometimes accuracy is not a useful metric in determining performance of an algorithm. This happens due to imbalanced classification. In such cases, we may need precision and recall score metrics [10]. Since the project is based on multiclass classifications, we thought that these metric may provide more accurate evaluation results. For Area Under the ROC Curve to be plotted TPR (True Positive Rate) and FPR (False Positive Rate) is needed.

$$TPR = \frac{TP}{TP+FN} \tag{6}$$

$$FPR = \frac{FP}{FP+TN} \tag{7}$$

TPR and FPR values are collected as pairs for various thresholds against the class probabilities.AUROC is plotted according to these pairs. Area under the curve is calculated by taking the integral of the plot[11].

4 GANTT CHART

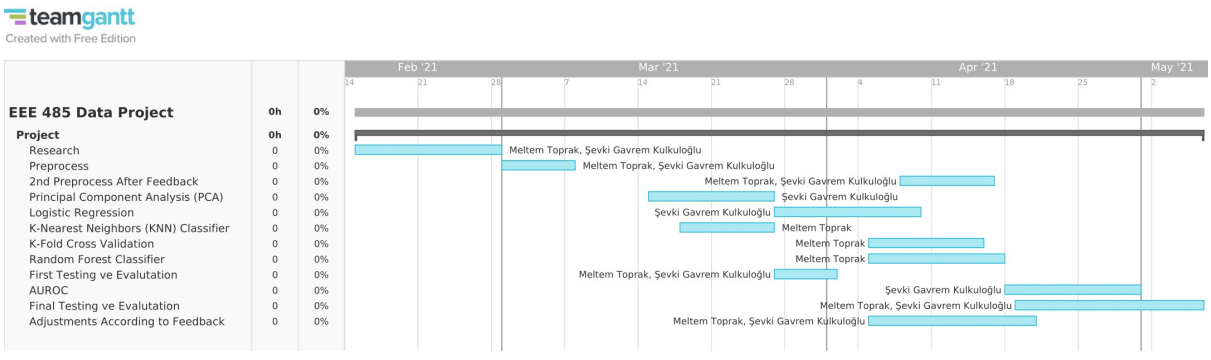
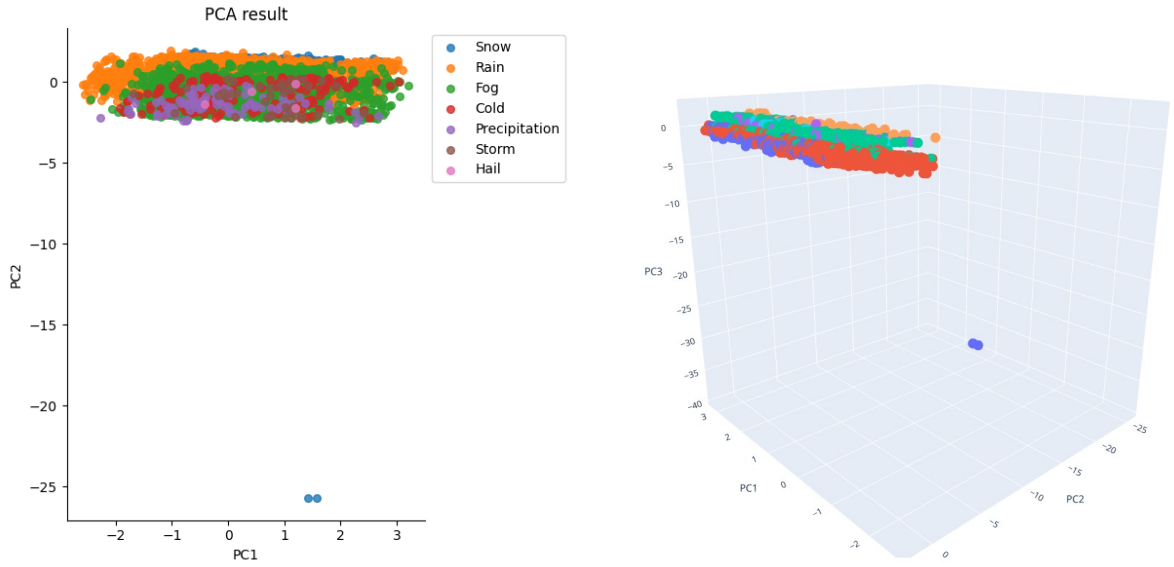


Figure 3: Gannt Chart

5 SIMULATION RESULTS



(a) 2D PCA plot

(b) 3D PCA plot

Figure 4: 2D and 3D PCA plots with zero mean unit variance

```
Explained varience for 2 =  0.44334185174676055
Explained varience for 3 =  0.647335955819219
```

Figure 5: Explained variance for zero mean unit variance of PCA

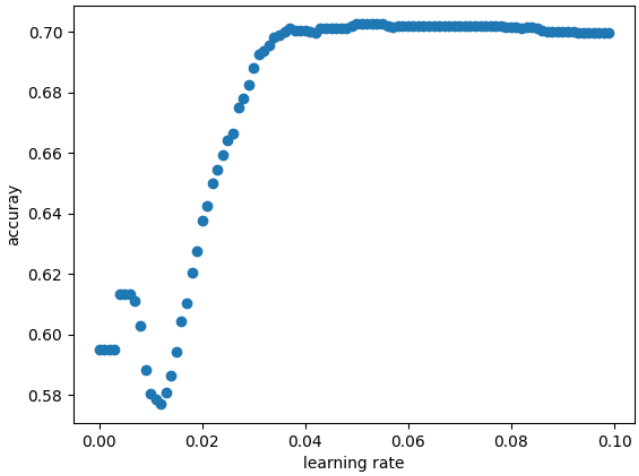


Figure 6: Logistic Regression accuracy vs Learning rate plot

```
Confusion Matrix for Logistic
 regression at learning rate  = 0.05
[[   0   64    3    0    0    0    0]
 [   0  293   69   79    0    0    0]
 [   0   45    0    0    0    0    0]
 [   0    0   98 1035   35    0    0]
 [   0    0    8  251    8    0    0]
 [   0   11    1    0    0    0    0]
 [   0    0    0    0    0    0    0]]
```

Figure 7: Confusion Matrix of Logistis Regression learning rate = 0.06

```
Cross Validation Scores: [0.66, 0.69, 0.64, 0.695, 0.75]
Average Cross Validation Scores: 0.687
```

Figure 8: Accuracy of Logistic Regression

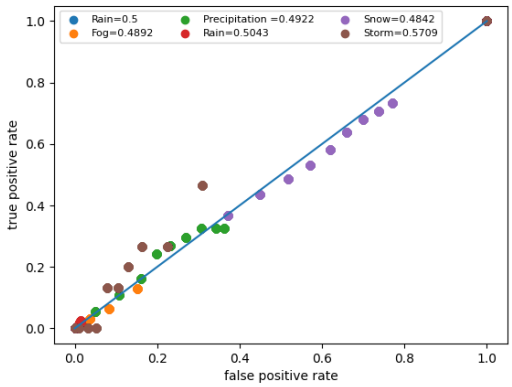
	k=3	k=5	k=7	k=8	k=9	k=10	k=11	k=12
p=1	80.25	81.8	81.95	81.45	82.05	81.65	81.6	81.25
p=2	80.9	81.5	82.35	81.8	82.05	81.75	81.8	81.85
p=3	80.3	81.3	81.65	81.35	82.15	82.05	82.25	81.5
p=4	79.35	80.8	81.5	80.9	81.45	81.65	81.7	81.2

Figure 9: Different hyperparameters for Knn and accuracy values

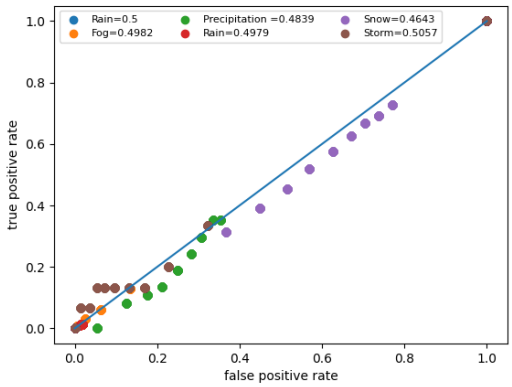
n_trees	max_depth	sample_size	Accuracy(%)	Training Time (seconds)
5	5	1	75.0	5.11
5	5	2	79.0	17.03
5	5	3	76.0	40.62
5	10	1	70.0	6.04
5	10	2	71.0	24.55
5	10	3	73.0	45.89
5	15	1	72.0	5.32
5	15	2	68.0	23.94
5	15	3	70.0	62.71
10	5	1	75.0	10.93
10	5	2	73.0	41.50
10	5	3	73.0	101.26
10	10	1	73.0	13.22
10	10	2	80.0	49.05
10	10	3	70.0	100.18
10	15	1	71.0	11.81
10	15	2	75.0	47.77
10	15	3	71.0	111.08
15	5	1	72.0	15.36
15	5	2	75.0	78.22
15	5	3	78.0	138.69
15	10	1	69.0	17.21
15	10	2	69.0	75.93
15	10	3	73.0	161.05
15	15	1	77.0	17.53
15	15	2	70.0	70.26
15	15	3	72.0	163.69

Figure 10: Different hyperparameters for random forest and the accuracy values

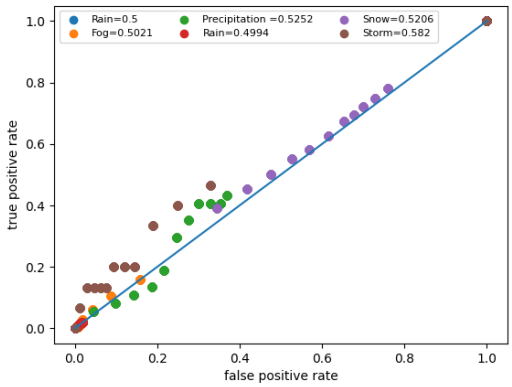
Simulation Setup: We have used Python programming language, Pycharm and Google Colab as environments.



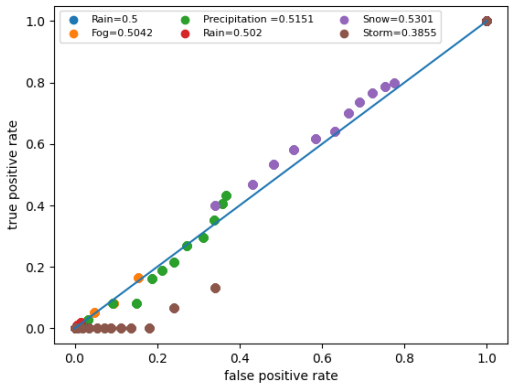
(a) AUROC for KNN k = 9 and p = 2



(b) AUROC for KNN k = 9 and p = 4



(c) AUROC for KNN k = 11 and p = 2



(d) AUROC for KNN k = 11 and p = 4

Figure 11: AUROC plots for KNN k = 9 , k = 11 and p = 2 , p = 4

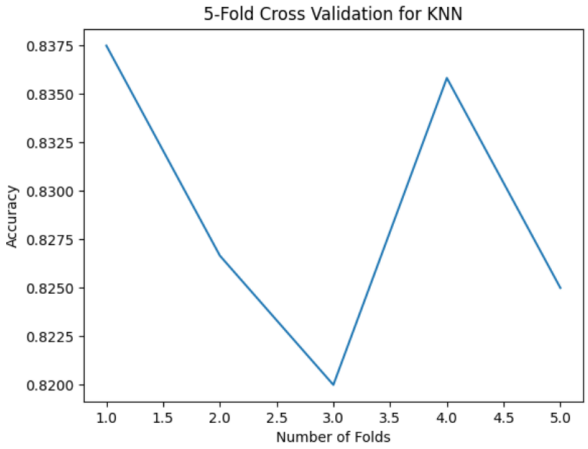


Figure 12: Accuracy of KNN vs number of fold (k fold cross validation)

```
Cross Validation Scores: [0.8375, 0.8266666666666667, 0.82, 0.8358333333333333, 0.825]
Average Cross Validation Scores: 0.829
```

Figure 13: Accuracy of KNN

```
Cross Validation Scores: [82.83333333333334, 82.58333333333333, 85.5, 83.25, 82.58333333333333]
Average Cross Validation Scores: 83.35
```

Figure 14: Accuracy of Random Forest

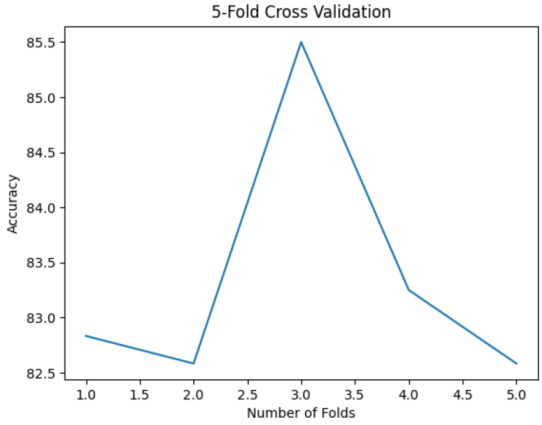


Figure 15: Accuracy Random Forest vs number of fold (k fold cross validation)

Model	Training Time (sec)	Accuracy
Logistic Regression	2.72	70%
KNN	247.89	84%
Random Forest	6594.5	87%

Figure 16: Models, Training time and Accuracy's

Validation: For validation of the performance the methods we will be using precision, recall, F1 score and accuracy. K-fold cross validation method is used. The ratio the dataset is as follows: train: 60%, validation: 20%, test: 20%

KNN Results: A grid search method is applied to choose k and p values that give the best accuracy score. The k values are determined as [3, 5, 7, 8, 9, 10, 11, 12] and p values are determined as [1, 2, 3, 4]. After the implementation the accuracy scores obtained for KNN algorithm as in Figure 9. Based on these results, the best k and p values are chosen as k=9 and p=2. The training takes no more than 90 seconds and the prediction time takes ~ 0.75 seconds.

AUROC's: Even tough some time we can see >0.6 scores bu in general AUROC scores are around 0.5. It is slightly better than the random number generator No Skill line).This is multiclass data we are dealing with and 7 labels are three. Because of that classes individually cannot have high TPR's and FPR's. For each class, they can have 6 times more false compared to binary classification.

Discussion Explained variance is not high enough to represent the dataset thoroughly. However, looking at the PCA plots we can see that even tough data points are gathered around and form a rectangular parallelepiped, the colors seem to be linearly separable mostly. In Logistic Regression model, until

difference between weights are 10^{-6} and learning rate at 0.06 it takes around ~ 2.7218 seconds to train and around ~ 0.15 seconds to predict it.

As it can be seen from the results, Logistic Regression has the shortest training time though it gives the worst accuracy. Random Forest gives the best accuracy but the implementation of the algorithm is expensive. KNN is better than Logistic Regression in terms of accuracy and is not good as Random Forest in terms of accuracy. Depending on these, KNN is chosen as the best algorithm to solve this classification problem.

6 CHALLENGES

First of all, as we observed from PCA plots, the data is not linearly separable. Also, there is very few number of label 'Hail'. Therefore, the algorithms cannot enough data points to learn it. For PCA algorithm we could not sure to with normalization method we need to choose since both gives different results. For Logistic Regression the challenge was the determining the boundaries of the classes to measure the accuracy of the model. For the time being it gives reasonable result but we are still working on it. For the KNN, as the k number increases, the algorithm gets slower but the accuracy gets better. Also, finding the best hyperparameters for the Random Forest Classifier was challenging since it takes 5 hyperparameters as input. Generally, as the values of the parameters increase, the accuracy gets better but the training time gets longer and the algorithm becomes more complex.

7 CONCLUSION

In this project, we have learned visualizing the multiclass dataset with PCA. We have investigated learning algorithms and selected the ones that are suitable for solving the multiclass classification problem. For the classification, we have found Logistic Regression, KNN and Random Forest. In the evaluation of the models, we have considered performance metrics and training times for efficiency. Also, we have learned how to do preprocess for the data properly so that the models give better results. We have learned the implementation of 5-Fold Cross Validation and its importance in terms of the distribution of the datasets and accuracy. The hyperparameter tuning is also done for selecting best values to make models more efficient. Then, we have learned how to interpret the area under ROC curve and its impotance in evaluating the algorithms.

As a result, we have observed that KNN algorithm is the optimal solution to solve this multiclass classification problem in terms of efficiency and performance metric. With KNN model, we can predict the weather in a certain location with 84% accuracy.

As a future work, the Random Forest algorithm may be developed by changing the hyperparameters and using a better decision tree structure.

References

[1] *Us weather events (2016 - 2020)*, Feb. 2021. [Online]. Available: https://www.kaggle.com/sobhanmoosavi/us-weather-events.

[2] *Weather$_p$rediction*), Feb. 2021. [Online]. Available: https://www.kaggle.com/bcantt/weather-prediction.

[3] *What is principal component analysis (pca) and how it is used?* Aug. 2020. [Online]. Available: https://www.sartorius.com/en/knowledge/science-snippets/what-is-principal-component-analysis-pca-and-how-it-is-used-507186.

[4] *The mathematics behind principal component analysis*, Dec. 2018. [Online]. Available: https://towardsdatascience.com/the-mathematics-behind-principal-component-analysis-fff2d7f4b643.

[5] *Logistic regression — detailed overview*, Mar. 2018. [Online]. Available: https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc.

[6] T. H. Gareth James Daniela Witten and R. Tibshirani, "An introduction to statistical learning with applications in r," *International Journal of Computer Applications*, 2013. DOI: 10.1007/978-1-4614-7138-7.

[7] *Machine learning basics with the k-nearest neighbors algorithm)*, Sep. 2018. [Online]. Available: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.

[8] *Random forests and decision trees from scratch in python*, Oct. 2018. [Online]. Available: https://towardsdatascience.com/random-forests-and-decision-trees-from-scratch-in-python-3e4fa5ae4249.

[9] *How to implement random forest from scratch in python*, Now 2016. [Online]. Available: https://machinelearningmastery.com/implement-random-forest-scratch-python/.

[10] *Performance metrics: Confusion matrix, precision, recall, and f1 score*, Sep. 2020. [Online]. Available: https://towardsdatascience.com/performance-metrics-confusion-matrix-precision-recall-and-f1-score-a8fe076a2262.

[11] *A gentle introduction to roc curve and auc in machine learning*, Dec. 2020. [Online]. Available: https://sefiks.com/2020/12/10/a-gentle-introduction-to-roc-curve-and-auc/.

8 APPENDIX

```
================PCA CLASS================
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px


class PCA:

    def fit(x_train,y_train):
        # y_train = y_train['Type'].tolist()
        #Normalization
        x_train = (x_train - x_train.mean()) / x_train.std(ddof=0)

        df_corr = (1 / x_train.shape[0]) * x_train.T.dot(x_train)

        # plt.figure(figsize=(10, 10))
        # sns.heatmap(df_corr, vmax=1, square=True, annot=True)
        # plt.title('Correlation matrix')
        # plt.show()

        u,s,v = np.linalg.svd(df_corr)
        eig_values, eig_vectors = s, u

        # first seven eigen vectors is enough
        x=0;
        for i in range(2):
             x = x+eig_values[i]
             # print(eig_values[i])

        print("Explained varience for "+str(i+1) + " = ", x/sum(eig_values))
        print("Explained varience for "+str(i+2)+ " = ", (x+eig_values[2])/sum(eig_values))



        # Finding the principal components
        pc1 = x_train.dot(eig_vectors[:,0])
        pc2 = x_train.dot(eig_vectors[:,1])
        pc3 = x_train.dot(eig_vectors[:,2])

        col= ['PC1','PC2','PC3']
        global pcs
        pcs = pd.concat([pc1, pc2,pc3], axis=1)
        pcs.columns = col
        pcs['label'] = y_train
```

```
        return pcs

    def plot_2D():

        # result = pd.DataFrame(pc1, columns=['PC1'])
        # result['PC2'] = pc2

        # print(result)
        fig2=sns.lmplot('PC1', 'PC2', data= pcs, fit_reg=False,  # x-axis, y-axis, data, no line
                   scatter_kws={"s": 30},height=6, aspect=1, hue="label",legend=False) # color

        # title
        plt.legend(bbox_to_anchor=(1, 1), loc=2)

        # plt.legend(bbox_to_anchor=(1.01, 1),borderaxespad=0)
        plt.title('PCA result')
        plt.tight_layout()
        plt.show()
        fig2.savefig('pca2d.png', bbox_inches='tight')

    def plot_3D():
        fig = px.scatter_3d(pcs, x='PC1', y='PC2', z='PC3',color= 'label')
        fig.show()


==========test PCA============
from models.PCA import PCA
import pandas as pd
from sklearn import preprocessing


def dataset_minmax(dataset):
    minmax = list()
    for column in dataset.columns[0:]:
        col_values = dataset[column].values
        value_min = min(col_values)
        value_max = max(col_values)
        minmax.append([value_min, value_max])
    return minmax

def normalize_dataset(dataset, minmax):
    i=0
    for column in dataset.columns[0:]:
        dataset[column] = (dataset[column] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])
        i=i+1


df = pd.read_csv(r'C:\Users\sevki\PycharmProjects\485data\Data\weather_dataset1')
df = df.drop('Unnamed: 0', 1)

# print(df['Severity'].unique())
```

```
le = preprocessing.LabelEncoder()

df['Severity'] = df['Severity'].astype(str)
le.fit(df['Severity'])
df['Severity'] = le.transform(df['Severity'])

x_train = df.iloc[:6000,:]
# x_test = df.iloc[4500:,:]

y_train = x_train['Labels']
# y_test = x_test['Labels']
x_train = x_train.drop('Labels', 1)
# x_test = x_test.drop('Labels', 1)

pd.set_option('display.max_columns', None)

p1 = PCA
pcs = p1.fit(x_train,y_train)
p1.plot_2D()
p1.plot_3D()

=======LOGISTIC REGRESSION CLASS==========
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
import time


class Logistic_Regression:

    def sigmoid(X, weight, b):
        z = np.array(np.dot(X, weight), dtype=np.float32) + b
        global y_predict
        y_predict =  1 / (1 + np.exp(-z))
        return y_predict

    # def square_loss(self, y_pred, target):
    #   return (np.mean(pow(y_pred - target,2)))


    def square_loss(y_pred, y_target):
        loss = (-1 * y_target* np.log(y_pred) - (1 - y_target) * np.log(1 - y_pred)).mean()
        return loss

    def log_likelihood(X, y, weight):
        z = np.dot(X, weight)
        log_like = np.sum(y * z - np.log(1 + np.exp(z)))
        return log_like
```

```
def gradient_descent(X, y_pred, y_target):
    return np.dot(X.T, (y_pred - y_target)) / y_target.shape[0]


def update_weight_loss( weight, lr, gradient):
    return weight - np.dot(lr, gradient.T)

def dataset_minmax(X):
    minmax = list()
    for column in X.columns[0:]:
        col_values = X[column].values
        value_min = min(col_values)
        value_max = max(col_values)
        minmax.append([value_min, value_max])
    return minmax

def normalize_dataset(dataset, minmax):
    i = 0
    for column in dataset.columns[0:]:
        dataset[column] = (dataset[column] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])
        i = i + 1

def fit(self, X, y_train, lr):
    start_time = time.time()
    global theta, intercept
    theta = np.zeros(X.shape[1])
    intercept = 0.1
    h_prime = 0
    change = 1
    num_iter = 0
    while abs(change) > 0.00001:
        h = self.sigmoid(X, theta, intercept)
        gradient = self.gradient_descent(X, h, y_train)
        theta = self.update_weight_loss(theta, lr, gradient)
        gradient_intercept = np.mean(h - y_train)
        intercept = intercept - lr * gradient_intercept
        change = np.mean(h - h_prime)
        h_prime = h
        num_iter = num_iter +1
    print("Training time (Log Reg using Gradient descent):" + str(time.time() - start_time) + " seconds")
    print("Learning rate: {}\nIteration: {}".format(lr, num_iter))

def predict(self,X,y_test):
    result = self.sigmoid(X, theta, intercept)
    for i in range(len(result)):
        result[i] = round(result[i], 2)

    y_test = np.float32(y_test)

    tr = 0
    fls = 0
```

```
        for i in range(len(result)):
            if abs((result[i]) - y_test[i]) >= 0.07:
                fls = fls + 1
            else:
                tr = tr + 1
        # print(tr, fls)
        print('score', round(tr / (tr + fls),2))



        deca =[0.14, 0.29, 0.43, 0.57, 0.71, 0.86]
        for i in range(len(result)):
            if result[i] < (deca[0]+deca[1])/2 :
                result[i] = deca[0]
            for j in range(len(deca)-2):
                # print(j)
                if  (result[i] >= (deca[j]+deca[j+1])/2 ) and ( result[i] < (deca[j+1]+deca[j+2])/2):
                    result[i] = deca[j+1]
            if result[i] >= (deca[4]+deca[5])/2:
                result[i] = deca[5]

        tr = 0
        fls = 0
        for i in range(len(result)):
            if abs((result[i]) - y_test[i]) >= 0.07:
                fls = fls + 1
            else:
                tr = tr + 1
        # print(tr, fls)
        print('score', round(tr / (tr + fls),2))


        print( (y_test == result).sum() / float(len(y_test)))
        print(round((np.sum(result == y_test) / y_test.shape[0]),2))
        return  ((y_test == result).sum() / float(len(y_test))) , result


======== Test Logistic Regression ==============
import pandas as pd
from models.Logistic_Regression import  Logistic_Regression
from Label_Encoder import Label_Encoder
from sklearn import preprocessing
import numpy as np
import matplotlib.pyplot as plt


def confusion_matrix( y_pred, y_test):
    # contructing empty confusion matrix with shape(7,7)
    conf_matrix = np.zeros((7, 7))

    for i, j in zip(y_test, y_pred):
```

```
        conf_matrix[i-1][j-1] = conf_matrix[i-1][j-1] + 1

    return conf_matrix.astype(int)

df = pd.read_csv(r'C:\Users\sevki\PycharmProjects\485data\Data\weather_dataset1')
df = df.drop('Unnamed: 0', 1)
# df = df.sample(frac=1)
# print(df['Severity'].unique())

le = preprocessing.LabelEncoder()

df['Severity'] = df['Severity'].astype(str)
le.fit(df['Severity'])
df['Severity'] = le.transform(df['Severity'])

x_train = df.iloc[:6000, :]
x_validation = df.iloc[6000:8000, :]
x_test = df.iloc[8000:, :]

y_train = x_train['Labels']
y_test = x_test['Labels']
y_validation = x_validation['Labels']
x_train = x_train.drop('Labels', 1)
x_validation = x_validation.drop('Labels', 1)
x_test = x_test.drop('Labels', 1)

pd.set_option('display.max_columns', None)

label = y_train.copy()
label2 = y_test.copy()


y_train = Label_Encoder(y_train)
y_test = Label_Encoder(y_test)
y_validation = Label_Encoder(y_validation)


x_train = (x_train - x_train.mean()) / x_train.std(ddof=0)
x_test = (x_test - x_test.mean()) / x_test.std(ddof=0)
x_validation = (x_validation - x_validation.mean()) / x_validation.std(ddof=0)

y_test2= y_test
y_train = y_train/7
y_test = y_test/7
y_validation2 = y_validation
y_validation = y_validation/7



for i  in range(len(y_train)):
    y_train[i] =round(y_train[i],2)
```

```
for i in range(len(y_test)):
    y_test[i] = round(y_test[i], 2)
for i in range(len(y_validation)):
    y_validation[i] = round(y_validation[i], 2)

log_res = Logistic_Regression


lr_rates = ist(np.array(list(range(0,100,1)))/1000)
points = []
for lr_rate in lr_rates:

    log_res.fit(log_res,x_train,y_train,lr_rate)
    acc ,y_pred = log_res.predict(log_res,x_test,y_test)
    acc = acc * 100
    points.append([acc, lr_rate])

df1=pd.DataFrame(points,columns=["x","y"])
cl1 = plt.scatter(df1.y,df1.x)
plt.xlabel('learning rate')
plt.ylabel('accuray (%)')
# plt.show()

y_pred = np.around(y_pred*7)
y_pred = y_pred.astype(int)

print("Accuracy =", acc)
print("Confusion Matrix for Logistic \n regression at learning rate  = " + str(lr_rates[0]))
print((confusion_matrix(y_pred,y_test2)))



========LABEL ENCODER ==================
import numpy as np
import pandas as pd

def Label_Encoder(y):
    y=y.values
    #arranging labels
    y=np.where(y == 'Cold', 1, y)
    y=np.where(y == 'Fog', 2, y)
    y=np.where(y == 'Precipitation', 3, y)
    y=np.where(y == 'Rain', 4, y)
    y=np.where(y == 'Snow', 5, y)
    y=np.where(y == 'Storm', 6, y)
    y=np.where(y == 'Hail', 7, y)


    return y
```

```
================ KNN ======================
class KNN:
    def __init__(self, k, p):
        self.k = k  # number of samples nearby
        self.p = p  # distance metric
        # self.distance=distance

    def find_distance(self, X_train, X_test):
        # the length of the datasets
        test_length = X_test.shape[0]
        train_length = X_train.shape[0]

        # Initilizing distance vector
        distance = np.zeros((test_length, train_length))

        # Computing distance between two points
        for i in range(test_length):
            for j in range(train_length):
                distance[i, j] = sum(abs(X_test[i, :] - X_train[j, :]) ** self.p)
                distance[i, j] = (distance[i, j]) ** (1 / self.p)

        return distance

    def fit(self, X_train, y_train, X_test):
        self.y_train = y_train
        self.distance = self.find_distance(X_train, X_test)
        # return distances

    def predict(self):
        test_length = self.distance.shape[0]  # distance length
        y_predicted = np.zeros(test_length)

        for i in range(test_length):
            y_indices = self.distance[i, :]
            y_indices = np.argsort(y_indices)
            k_closest_classes = self.y_train[y_indices[:self.k]]
            k_closest_classes = k_closest_classes.astype(int)  # coverting it to int type
            frequencies = np.bincount(k_closest_classes.flatten())  # counting the frequency of the classes
            y_predicted[i] = np.argmax(frequencies)  # the most frequent class
        return y_predicted

    def accuracy(self, y_pred, y_validation):
        y_pred = y_pred.reshape(y_validation.shape[0], 1)
        #
        y_pred = y_pred.astype(int)
        y_pred = np.concatenate(y_pred)
        y_validation = y_validation.astype(int)

        return np.sum(y_pred == y_validation.flatten()) / y_validation.shape[0]
```

```
    def confusion_matrix(self, y_pred, y_test):
        # contructing empty confusion matrix with shape(7,7)
        conf_matrix = np.zeros((7, 7))
        for i, j in zip(y_test, y_pred):
            conf_matrix[i - 1][j - 1] = conf_matrix[i - 1][j - 1] + 1
        return conf_matrix.astype(int)

if __name__ == '__main__':
    # loading data

    X_train = pd.read_csv(X_train_path)
    X_validation = pd.read_csv(X_validation_path)
    y_train = pd.read_csv(y_train_path)
    y_validation = pd.read_csv(y_validation_path)

    X_train = X_train.drop('Unnamed: 0', 1)
    X_validation = X_validation.drop('Unnamed: 0', 1)
    y_train = y_train.drop('Unnamed: 0', 1)
    y_validation = y_validation.drop('Unnamed: 0', 1)

    # converting dataframes into numpy arrays
    X_train = X_train.to_numpy()
    X_validation = X_validation.to_numpy()
    y_train = y_train['0'].values
    y_validation = y_validation['0'].values

    print('X_train.shape', X_train.shape)
    print('y_train.shape', y_train.shape)

    X_test = pd.read_csv(X_test_path)
    y_test = pd.read_csv(y_test_path)
    X_test = X_test.drop('Unnamed: 0', 1)
    y_test = y_test.drop('Unnamed: 0', 1)
    X_test = X_test.to_numpy()
    y_test = y_test['0'].values

    knn = KNN(k=9, p=2)
    # training
    Start = time.time()
    knn.fit(X_train, y_train, X_test)
    print('Training Time: ', time.time() - Start)
    #
    # predicting
    Start = time.time()
    y_pred = knn.predict()
    print('Prediction Time: ', time.time() - Start)
    y_pred = y_pred.reshape(y_test.shape[0], 1)
    #
    y_pred = y_pred.astype(int)
    y_pred = np.concatenate(y_pred)
    y_test = y_test.astype(int)
```

```
    print('np.unique(y_validation)',np.unique(y_test))
    print('np.unique(y_pred)', np.unique(y_pred))
    # accuracy
    print('Accuracy:', np.sum(y_pred == y_test.flatten()) / y_test.shape[0])
    print('Accuracy:', knn.accuracy(y_pred, y_test))
    conf_matr = knn.confusion_matrix(y_pred.astype(int), y_test.astype(int))
    print(conf_matr)
   # Evaluating the algorithm depending on different k and p values
    k_values = [3, 5, 7, 8, 9, 10, 11, 12]
    p_values = [1, 2, 3, 4]

    for k in k_values:
        for p in p_values:
            print('================================================')
            print('Results for k =', k, 'and p =', p)
            knn = KNN(k=k, p=p)
            # training
            Start = time.time()
            knn.fit(X_train, y_train, X_validation)
            print('Training Time: ', time.time() - Start)
            #
            # predicting
            Start = time.time()
            y_pred = knn.predict()
            print('Prediction Time: ', time.time() - Start)
            print('Accuracy:', knn.accuracy(y_pred, y_validation))
            conf_matr = knn.confusion_matrix(y_pred.astype(int), y_validation.astype(int))
            print(conf_matr)


================= Random Forest ======================
from random import seed
from random import randrange
from math import sqrt
import pandas as pd
import numpy as np
import time


class RandomForest:

    def __init__(self, max_depth, min_size, sample_size, n_trees, n_feats):
        self.max_depth=max_depth
        self.min_size=min_size
        self.sample_size=sample_size
        self.n_trees=n_trees
        self.n_feats=n_feats

    # Calculate accuracy percentage
    def accuracy_metric(self,actual, predicted):
        correct = 0
```

```
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0

# Splitting the data as left and right
def split_into_two(self,data, index, value):
    l = []  # left
    r = []  # right
    for sample in data:
        if sample[index] < value:
            l.append(sample)
        else:
            r.append(sample)
    return l, r

#Calculating the purity of the data
def gini(self, splits, classes):
    n_instances = []
    for s in splits:
        n_instances.append(len(s))
    n_instances = float(sum(n_instances))
    g_index = 0#weighted gini index
    for split in splits:
        if float(len(split)) != 0:
            res = 0.0
            for class_val in classes:
                probs = []
                for row in split:
                    probs.append(row[-1])
                probs = probs.count(class_val) / float(len(split))
                res = res + probs ** 2
            g_index = g_index + (1.0 - res) * (float(len(split)) / n_instances)
    return g_index

# getting a split node
def best_split(self,data, n_features):
    feature_size = len(data[0] )-1
    # print('data[0]', data[0] )
    # print('data[0]-1',data[0]-1)
    labels = []
    for i in range(len(data)):
        labels.append(data[i][-1])
    class_values = np.unique(labels).tolist()

    best_gini_score = 1  # gini index should be lower than 1
    features = []

    while len(features) < n_features:  # SHUFFLING GIBI BI SEY
        i = randrange(feature_size)
        if i not in features:
```

```
            features.append(i)
    for i in features:
        for row in data:
            splits = self.split_into_two(data, i, row[i])
            g_index = self.gini(splits, class_values)
            if g_index < best_gini_score:
                best_index = i
                best_value = row[i]
                best_gini_score = g_index
                best_splits = splits
    dict = {'index': best_index, 'value': best_value, 'splits': best_splits}
    # print('best_index',best_index)
    return dict

# Create a terminal node value
def leaf_node(self,group):
    res = []
    for row in group:
        res.append(row[-1])
    freq = np.bincount(res)  # counting the frequency of the classes
    return np.argmax(freq)  # the most frequent class

# Create child splits for a node or make terminal
def split(self,dec_node, max_depth, min_size, n_features, depth):
    l_node, r_node = dec_node['splits']
    length_left = len(l_node)
    lenght_right = len(r_node)
    del (dec_node['splits'])

    if not l_node or not r_node:
        dec_node['left'] = self.leaf_node(l_node + r_node)
        dec_node['right'] = self.leaf_node(l_node + r_node)
        return
    # max depth
    if depth >= max_depth:
        dec_node['left'] = self.leaf_node(l_node)
        dec_node['right'] = self.leaf_node(r_node)
        return
    #left node
    if length_left <= min_size:
        dec_node['left'] = self.leaf_node(l_node)
    else:
        dec_node['left'] = self.best_split(l_node, n_features)
        self.split(dec_node['left'], max_depth, min_size, n_features, depth + 1)
    # right node
    if lenght_right <= min_size:
        dec_node['right'] = self.leaf_node(r_node)
    else:
        dec_node['right'] = self.best_split(r_node, n_features)
        self.split(dec_node['right'], max_depth, min_size, n_features, depth + 1)
```

```
# growing a tree
def grow_tree(self,train, max_depth, min_size, n_features):
    tree_root = self.best_split(train, n_features)
    self.split(tree_root, max_depth, min_size, n_features, 1)
    return tree_root


def predict(self, node, row):
    if row[node['index']] < node['value']:
        if isinstance(node['left'], dict):
            return self.predict(node['left'], row)
        else:
            return node['left']
    else:
        if isinstance(node['right'], dict):
            return self.predict(node['right'], row)
        else:
            return node['right']


def bagging_pred(self,trees, row):
    pred = []
    for t in trees:
        pred.append(self.predict(t, row))
    freq = np.bincount(pred)  # counting the frequency of the classes
    return np.argmax(freq)  # the most frequent class

# fit
def random_forest_fit(self,train, y_train):
    self.trees=[]
    y_train = np.array(y_train)
    y_train = y_train.reshape(y_train.shape[0], 1)
    train = np.append(train, y_train, axis=1)
    trees = []
    for i in range(self.n_trees):

        sample = []
        n_sample = round(len(train) * self.sample_size)
        while len(sample) < n_sample:
            i = randrange(len(train))
            sample.append(train[i])
        tree = self.grow_tree(sample, self.max_depth, self.min_size, self.n_feats)
        self.trees.append(tree)


def random_forest_predict(self,X_test):
    # predict
    predictions = []
    # print('random_forest_predict')
    for row in X_test:
        # print('row',row)
```

```
            predictions.append(self.bagging_pred(self.trees, row))
        return (predictions)
        #     ======================
    def confusion_matrix(self, y_pred, y_test):
        # contructing empty confusion matrix with shape(6,6)
        conf_matrix = np.zeros((7, 7))
        for i, j in zip(y_test, y_pred):
            conf_matrix[i - 1][j - 1] = conf_matrix[i - 1][j - 1] + 1
        return conf_matrix.astype(int)
if __name__ == '__main__':
    # loading data

    X_train = pd.read_csv(X_train_path)
    X_test = pd.read_csv(X_test_path)
    X_validation = pd.read_csv(X_validation_path)
    y_test=pd.read_csv(y_test_path)
    y_train = pd.read_csv(y_train_path)
    y_validation = pd.read_csv(y_validation_path)

    X_train = X_train.drop('Unnamed: 0', 1)
    X_validation = X_validation.drop('Unnamed: 0', 1)
    y_train = y_train.drop('Unnamed: 0', 1)
    y_validation = y_validation.drop('Unnamed: 0', 1)
    X_test = X_test.drop('Unnamed: 0', 1)
    y_test = y_test.drop('Unnamed: 0', 1)

    # converting dataframes into numpy arrays
    X_test = X_test.to_numpy()
    X_train = X_train.to_numpy()
    X_validation = X_validation.to_numpy()
    y_train = y_train['0'].values
    y_test = y_test['0'].values
    y_validation = y_validation['0'].values

    print('X_test.shape', X_test.shape)
    print('y_test.shape', y_test.shape)
    # X_train=X_train[0:400,:]
    # y_train=y_train[0:400]
    # X_validation=X_validation[0:100,:]
    # y_validation=y_validation[0:100]
    # X_test=X_test[0:100,:]
    # y_test=y_test[0:100]

    max_depth = 10
    min_size = 1
    sample_size = 2.0
    n_feats = int(sqrt(len(X_train[0]) - 1))

    clf = RandomForest(max_depth, min_size, sample_size, n_trees=10, n_feats=n_feats)
    # FIT PREDICT
    Start = time.time()
```

```
    clf.random_forest_fit(X_train, y_train)
    print('Training Time: ', time.time() - Start)

    Start = time.time()
    predictions = clf.random_forest_predict(X_test)
    print('Prediction Time: ', time.time() - Start)
    predictions=np.array(predictions)
    print(clf.accuracy_metric(y_test, predictions))
    conf_matr = clf.confusion_matrix(predictions.astype(int), y_test.astype(int))
    print(conf_matr)

# Evaluating the algorithm depending on different parameter values
    trees = [5, 10, 15]
    max_depth = [5, 10, 15]
    sample_size=[1, 2, 3]

    for t in trees:
        for m in max_depth:
            for s in sample_size:
                print('================================================')
                print('Results for n_trees =', t, ', max_depth=', m,'sample_size=',s,)
                clf = RandomForest(max_depth=m, min_size=min_size, sample_size=s, n_trees=t, n_feats=n_feats)
                # training
                Start = time.time()
                clf.random_forest_fit(X_train, y_train)
                print('Training Time: ', time.time() - Start)
                #
                # predicting
                Start = time.time()
                y_pred = clf.random_forest_predict( X_validation)
                print('Prediction Time: ', time.time() - Start)
                print('Accuracy:', clf.accuracy_metric(y_validation,y_pred))
                y_pred=np.array(y_pred)

================= 5 Fold Cross Validation ======================
def cross_validation_split(data,labels, n_folds):
    dataset_split = []
    dataset_split_labels=[]

    fold_size = data.shape[0] // n_folds

    # print(fold_size)
    for i in range(n_folds):
        fold = []
        fold_labels = []

        while len(fold) < fold_size:
            random_index = randrange(data.shape[0])
            fold.append(data[random_index])
            fold_labels.append(labels[random_index])
            data=np.delete(data, [random_index], axis=0)
```

```
        dataset_split.append(fold)
        dataset_split_labels.append(fold_labels)

    return dataset_split,dataset_split_labels


n_folds = 5
max_depth = 10
min_size = 1
sample_size = 1.0
n_feats = int(sqrt(len(X_new[0]) - 1))  # MODIFY
def evaluate(dataset,labels, n_folds):
    folds, fold_labels = cross_validation_split(dataset,labels, n_folds)
    results = []
    print(len(folds))
    # print(folds)
    for i in range(len(folds)):
        # print('fold',fold)
        print('========================')
        X_train = []
        y_train=[]
        for j in range(len(folds)):
            if j == i:
                X_test =folds[j]
                y_test=fold_labels[j]
            else:
                X_train.append(folds[j])
                y_train.append(fold_labels[j])
        X_train = sum(X_train, [])
        y_train=sum(y_train, [])
        clf = RandomForest(max_depth=10, min_size=min_size,

        sample_size=2, n_trees=10, n_feats=n_feats)  # n_trees=5
        # #####FIT PREDICT CHECK
        clf.random_forest_fit(X_train, y_train)
        predictions = clf.random_forest_predict(X_test)
        results.append(clf.accuracy_metric(y_test, predictions))
    print('Cross Validation Scores:',results)
    print('Average Cross Validation Scores:', sum(results)/n_folds)
    return results


results=evaluate(X_new,y_new, 5)
folds=[1,2,3,4,5]
# importing the required module
import matplotlib.pyplot as plt

# x axis values
x = folds
# corresponding y axis values
```

```
y = results

# plotting the points
plt.plot(x, y)

# naming the x axis
plt.xlabel('Number of Folds')
# naming the y axis
plt.ylabel('Accuracy')

# giving a title to my graph
plt.title('5-Fold Cross Validation for Random Forest')

# function to show the plot
plt.show()


================ AUROC ============================
import pandas as pd
from sklearn import preprocessing
from models.Label_Encoder import Label_Encoder
import numpy as np
import matplotlib.pyplot as plt
import pickle



knn_pred = pd.read_csv(r'/Data/knn_predict.csv')
auc_pred = pd.read_csv(r'/Data/auc-case-predictions.csv')

df = pd.read_csv(r'/Data/weather_dataset1')

# print(df['Severity'].unique())
df = df.drop('Unnamed: 0', 1)

le = preprocessing.LabelEncoder()
df['Severity'] = df['Severity'].astype(str)
le.fit(df['Severity'])
df['Severity'] = le.transform(df['Severity'])

x_validation = df.iloc[6000:8000, :]

y_validation = x_validation['Labels']
y_validation = Label_Encoder(y_validation)

with open("../scractess/test_1_9_ 4.txt", "rb") as fp:   # Unpickling
    predict = pickle.load(fp)


for i in range(len(predict)):
    if len(predict[i])<7:
```

```
        point = len(predict[i])
        for j in range(7-point):
            # predict[i][j+point] = 0
            predict[i]= np.append(predict[i], np.array([0]))


# for i in predict:
#     print(i)
roc_points1 = []
roc_points2 = []
roc_points3 = []
roc_points4 = []
roc_points5 = []
roc_points6 = []


acc1 = acc2 = acc3 = acc4 = acc5 = acc6 = []
#
# y_test = auc_pred['actual']
# predict = auc_pred['prediction']

tresholds = list(np.array(list(range(-100,1000,1)))/800)
for treshold in tresholds:
    tp =[0] * 7; tn = [0] * 7;    fp = [0] * 7;    fn = [0] * 7; tpr = [0] * 7; fpr = [0] * 7;
    for  i in range(len(y_validation)):
        for j in range(len(predict[i])):
            y_pred = predict[i][j]
            if y_pred >= treshold:
                prediction_class = 1
            else:
                prediction_class = 0

            if prediction_class == 1 and y_validation[i] == j+1:
                tp[j]= tp[j] + 1
            elif  y_validation[i] == j+1 and prediction_class ==0:
                fn[j] = fn[j]+1
            elif  y_validation[i] != j+1 and prediction_class == 1:
                fp[j] = fp[j] + 1
            elif  y_validation[i] != j+1 and prediction_class ==0:
                tn[j] = tn[j] + 1
    for k in range(6):
        tpr[k] = tp[k] / (tp[k] + fn[k])
        fpr[k] = fp[k] / (fp[k]+tn[k])
        if k == 0:
            roc_points1.append([tpr[0],fpr[0]])
        elif k == 1:
            roc_points2.append([tpr[1], fpr[1]])
        elif k == 2:
            roc_points3.append([tpr[2], fpr[2]])
        elif k == 3:
            roc_points4.append([tpr[3], fpr[3]])
```

```
        elif k == 4:
            roc_points5.append([tpr[4], fpr[4]])
        elif k == 5:
            roc_points6.append([tpr[5], fpr[5]])



df1=pd.DataFrame(roc_points1,columns=["x","y"])
df2=pd.DataFrame(roc_points2,columns=["x","y"])
df3=pd.DataFrame(roc_points3,columns=["x","y"])
df4=pd.DataFrame(roc_points4,columns=["x","y"])
df5=pd.DataFrame(roc_points5,columns=["x","y"])
df6=pd.DataFrame(roc_points6,columns=["x","y"])

cl1 = plt.scatter(df1.y,df1.x)
cl2 = plt.scatter(df2.y,df2.x)
cl3 = plt.scatter(df3.y,df3.x)
cl4 = plt.scatter(df4.y,df4.x)
cl5 = plt.scatter(df5.y,df5.x)
cl6 = plt.scatter(df6.y,df6.x)
plt.plot([0,1])

one = (round(abs(np.trapz(df1.x,df1.y)),4))
two = (round(abs(np.trapz(df2.x,df2.y)),4))
three = (round(abs(np.trapz(df3.x,df3.y)),4))
four = (round(abs(np.trapz(df4.x,df4.y)),4))
five = (round(abs(np.trapz(df5.x,df5.y)),4))
six = (round(abs(np.trapz(df6.x,df6.y)),4))
print(1, one)
print(2, two)
print(3, three)
print(4, four)
print(5, five)
print(6, six)

#

plt.legend((cl1, cl2 , cl3, cl4, cl5, cl6),
           ('Rain=' + str(one),'Fog=' +str(two), 'Precipitation ='

           +str(three), 'Rain='+str(four), 'Snow='+str(five)
            , 'Storm='+str(six)),
           scatterpoints=1,
           loc='upper left',
           ncol=3,
           fontsize=8)
plt.xlabel('false positive rate')
plt.ylabel('true positive rate')
plt.show()
```