Komal Diwe
DATA SCIENCE ENTHUSIAST
github.com/ksdiwe komaldiwe28@gmail.com

Logistic Regression

Logistic Regression on Algerian Forest Fire

Logistic Regression

Logistic regression is one such regression algorithm that can be used for performing classification problems. It calculates the probability that a given value belongs to a specific class. If the probability is more than 50%, it assigns the value to that particular class else if the probability is less than 50%, the value is assigned to the other class. Therefore, we can say that logistic regression acts as a binary classifier.

For linear regression, the model is defined by:

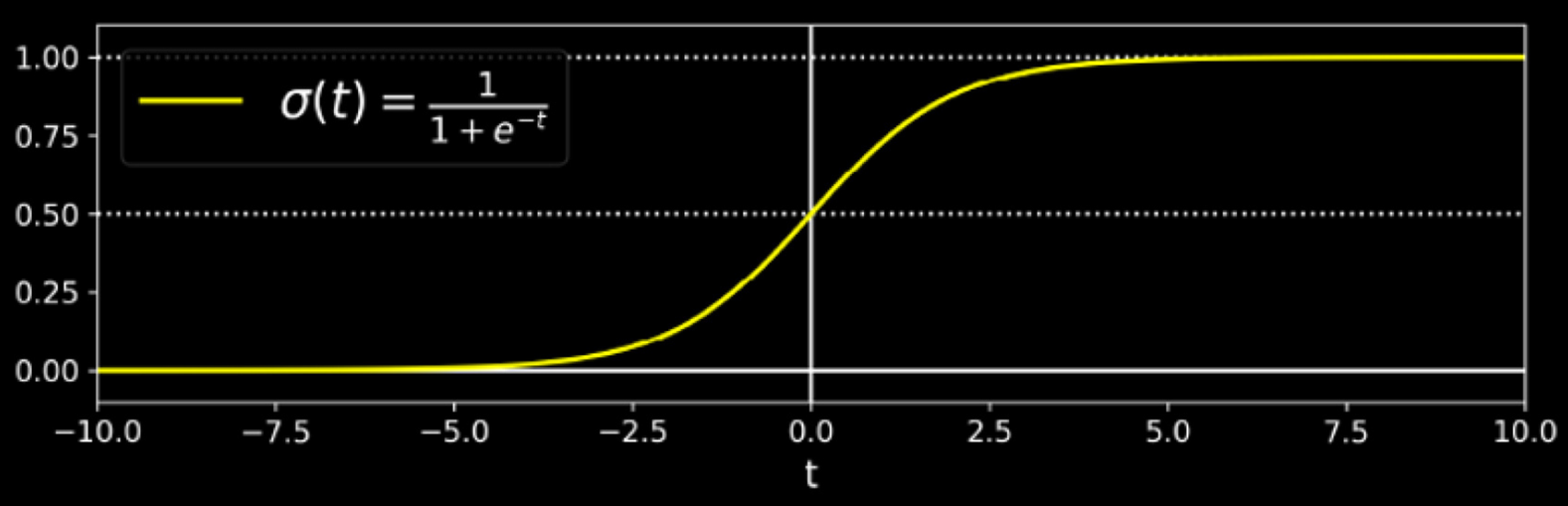
$$y = \beta_0 + \beta_1 x - (i)$$

and for logistic regression, we calculate probability, i.e. y is the probability of a given variable x belonging to a certain class. Thus, it is obvious that the value of y should lie between 0 and 1.

But, when we use equation(i) to calculate probability, we would get values less than 0 as well as greater than 1. That doesn't make any sense.
So, we need to use such an equation that always gives values between 0 and 1, as we desire while calculating the probability.

Sigmoid Function

We use the sigmoid function as the underlying function in Logistic regression. Mathematically and graphically, it is shown as:



Why do we use the Sigmoid Function?

1. The sigmoid function's range is bounded between 0 and 1. Thus it's useful in calculating the probability of the Logistic function.
2. Its derivative is easy to calculate than other functions which are useful during gradient descent calculation.
3. It is a simple way of introducing non-linearity to the model.

Although there are other functions as well, which can be used, sigmoid is the most common function used for logistic regression. We will talk about the rest of the functions in the neural network section.

The logistic function is given as:

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}.$$

Let's see some manipulation with the logistic function:

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

$$\therefore \quad p(x) = \frac{e^{(\beta_0 + \beta_1 x)}}{1 + e^{(\beta_0 + \beta_1 x)}} \qquad \text{—— (1)}$$

let's do some manipulation with eqn (1)

$$1 - p(x) = 1 - \frac{e^{h(\theta)}}{1 + e^{h(\theta)}}$$

where, $h(\theta) = \beta_0 + \beta_1 x$

$$\therefore \quad 1 - p(x) = \frac{1}{1 + e^{h(\theta)}} = \frac{1}{1 + e^{(\beta_0 + \beta_1 x)}} \qquad \text{—— (2)}$$

If we divide (1) by eqn (2),

$$\frac{p(x)}{1 - p(x)} = e^{(\beta_0 + \beta_1 x)}$$

taking log on both sides

$$\log\left(\frac{p(x)}{1 - p(x)}\right) = \beta_0 + \beta_1 x$$

logit function.

We can see that the logit function is linear in terms of x.

Prediction

$$y = \begin{cases} 0, if p(x) < 0.5 \\ 1, if p(x) >= 0.5 \end{cases}$$

Cost Function

for a single training instance

$$cost(p(x), y) = \begin{cases} -\log(p(x)), & \text{if } y=1 \\ -\log(1-p(x)), & \text{if } y=0 \end{cases}$$

why this cost function?

- if $p(x)=1$ and $y=1$, cost $=0$
  $p(x)=0$ and $y=0$, cost $=0$

but,

if $p(x)=0$ and $y=1$, cost $\to \infty$
$\because \log 0 = \infty$

if $p(x)=1$ and $y=0$, cost $\to \infty$.

The cost function for the whole training set is given as :

$$J(\theta) = -\frac{1}{m}\Sigma_{i=1}^{m}\left[y^{(i)}log\left(\hat{p}^{(i)}\right) + \left(1-y^{(i)}\right)log\left(1-\hat{p}^{(i)}\right)\right]$$

The values of parameters (θ) for which the cost function is minimum are calculated using the gradient descent (as discussed in the Linear Regression section) algorithm. The partial derivative for the cost function is given as :

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma\left(\theta^T\mathbf{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

Multiple Logistic Function

We can generalize the simple logistic function for multiple features as:

$$p(X) = \frac{e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}{1+e^{\beta_0+\beta_1 X_1+\cdots+\beta_p X_p}}.$$

And the logit function can be written as:

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p,$$

The coefficients are calculated the same we did for simple logistic functions, by passing the above equation into the cost function.

Just like we did in multilinear regression, we will check for correlation between different features for Multi logistics as well.

We will see how we implement all the above concepts through a practical example.

Multinomial Logistics Regression(Number of Labels >2)

Many times, there are classification problems where the number of classes is greater than 2.
We can extend Logistic regression for multi-class classification. The logic is simple; we train our logistic model for each class and calculate the probability(hθx) that a specific feature belongs to that class. Once we have trained the model for all the classes, we predict a new value's class by choosing that class for which the probability(hθx) is maximum.

Although we have libraries that we can use to perform multinomial logistic regression, we rarely use logistic regression for classification problems where the number of classes is more than 2.
There are many other classification models for such scenarios. We will see more of that in the coming lectures.

Learning Algorithm

The learning algorithm is how we search the set of possible hypotheses for the best parameterization (in this case the weight vector $\mathbf{w}$. This search is an optimization problem looking for the hypothesis that optimizes an error measure.

There is no sophisticated closed-form solution like least-squares linear, so we will use gradient descent instead.
Specifically, we will use batch gradient descent which calculates the gradient from all data points in the data set.

Luckily, our "cross-entropy" error measure is convex so there is only one minimum. Thus the minimum we arrive at is the global minimum.

To learn we're going to minimize the following error measure using batch gradient descent.

$$e(h(\mathbf{x}_n), y_n) = \ln\left(1 + e^{-y_n\ \mathbf{w}^T \mathbf{x}_n}\right)$$

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} e(h(\mathbf{x}_n), y_n) = \frac{1}{N}\sum_{n=1}^{N} \ln\left(1 + e^{-y_n\ \mathbf{w}^T \mathbf{x}_n}\right)$$

We'll need the derivative of the point loss function and possibly some abuse of notation.

$$\frac{d}{d\mathbf{w}} e(h(\mathbf{x}_n), y_n) = \frac{-y_n\ \mathbf{x}_n\ e^{-y_n \mathbf{w}^T \mathbf{x}_n}}{1 + e^{-y_n \mathbf{w}^T \mathbf{x}_n}} = -\frac{y_n\ \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^T \mathbf{x}_n}}$$

With the point loss derivative we can determine the gradient of the in-sample error:

$$\nabla E_{\text{in}}(\mathbf{w}) = \frac{d}{d\mathbf{w}}\left[\frac{1}{N}\sum_{n=1}^{N} e(h(\mathbf{x}_n), y_n)\right] \tag{1}$$

$$= \frac{1}{N}\sum_{n=1}^{N} \frac{d}{d\mathbf{w}} e(h(\mathbf{x}_n), y_n) \tag{2}$$

$$= \frac{1}{N}\sum_{n=1}^{N} \left(-\frac{y_n\ \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^T \mathbf{x}_n}}\right) \tag{3}$$

$$= -\frac{1}{N}\sum_{n=1}^{N} \frac{y_n\ \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^T \mathbf{x}_n}} \tag{4}$$

Our weight update rule per batch gradient descent becomes

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \eta\ \nabla E_{\text{in}}(\mathbf{w}_i) \tag{5}$$

$$= \mathbf{w}_i - \eta\ \left(-\frac{1}{N}\sum_{n=1}^{N} \frac{y_n\ \mathbf{x}_n}{1 + e^{y_n \mathbf{w}_i^T \mathbf{x}_n}}\right) \tag{6}$$

$$= \mathbf{w}_i + \eta\ \left(\frac{1}{N}\sum_{n=1}^{N} \frac{y_n\ \mathbf{x}_n}{1 + e^{y_n \mathbf{w}_i^T \mathbf{x}_n}}\right) \tag{7}$$

where η is the learning rate.

Evaluation of the Classification Model

In machine learning, once we have a result of the classification problem, how do we measure how accurate our classification is?
For a regression problem, we have different metrics like R Squared score, Mean Squared Error, etc. what are the metrics to measure the credibility of a classification model?

Metrics

In a regression problem, the accuracy is generally measured in terms of the difference between the actual values and the predicted values.
In a classification problem, the credibility of the model is measured using the confusion matrix generated, i.e., how accurately the true positives and true negatives were predicted.
The different metrics used for this purpose are:

- Accuracy
- Recall
- Precision
- F1 Score
- Specificity
- AUC(Area Under the Curve)
- RUC(Receiver Operator Characteristic)

Python Implementation

```
# Pandas and Numpy
import pandas as pd
import numpy as np

# Visualisation libraries
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# For Q-Q Plot
import scipy.stats as stats

# To ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Machine Learning libraries
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

# To be able to see maximum columns on screen
pd.set_option('display.max_columns', 500)

# To save the model
import pickle
```

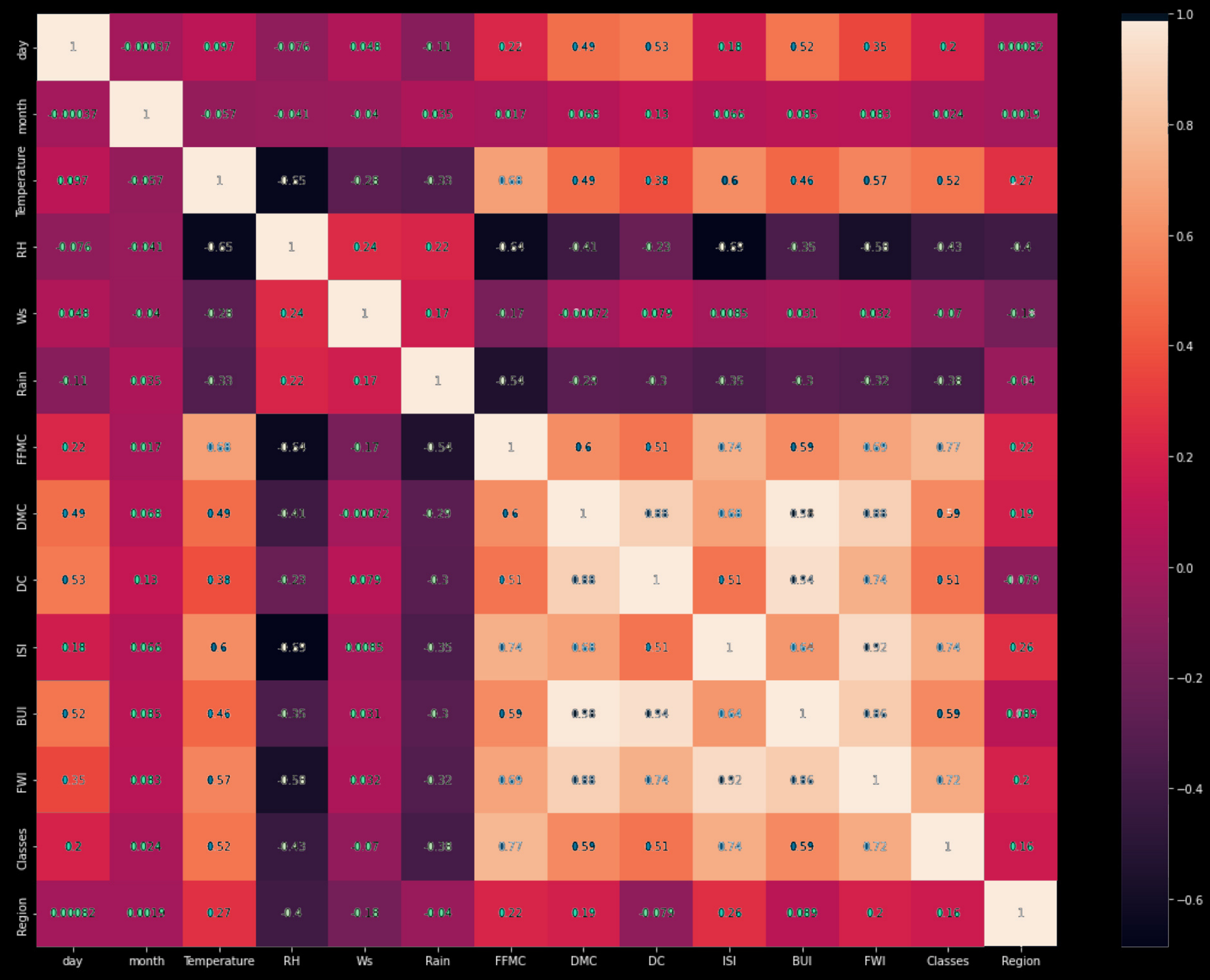
Dataset looks like:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
0	01	06	2012	29	57	18	0	65.7	3.4	7.6	1.3	3.4	0.5	not fire
1	02	06	2012	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4	not fire
2	03	06	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not fire
3	04	06	2012	25	89	13	2.5	28.6	1.3	6.9	0	1.7	0	not fire
4	05	06	2012	27	77	16	0	64.8	3	14.2	1.2	3.9	0.5	not fire

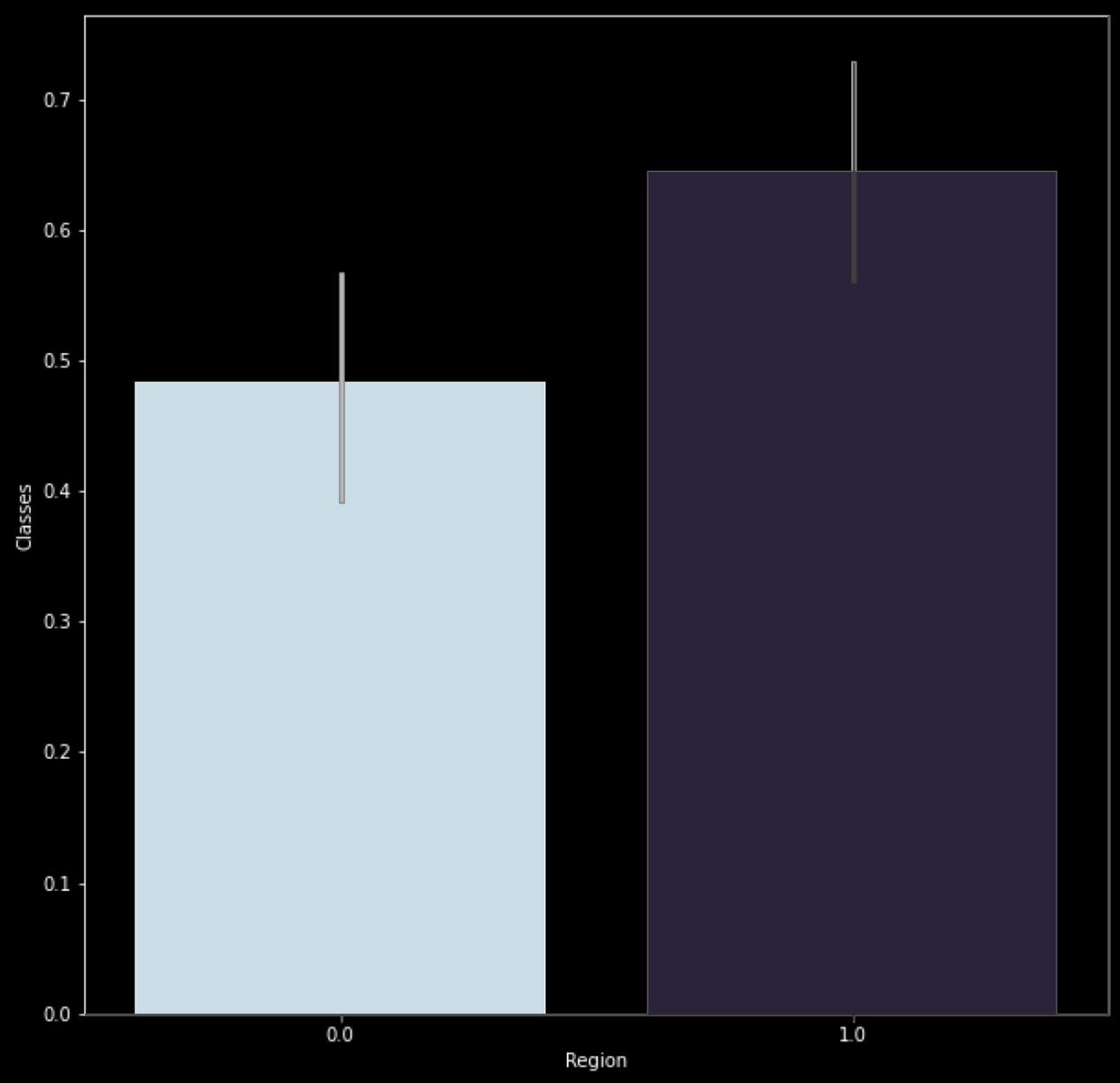
Description of Dataset:

1. The dataset includes 244 instances that regroup a data of two regions of Algeria, namely the Bejaia region located in the northeast of Algeria and the Sidi Bel-abbes region located in the northwest of Algeria.
2. 122 instances for each region.
3. The period from June 2012 to September 2012.
4. The dataset includes 11 attribues and 1 output attribute (classes)
5. The 244 instances have been classified into the fire (138 classes) and not fire (106 classes) classes.

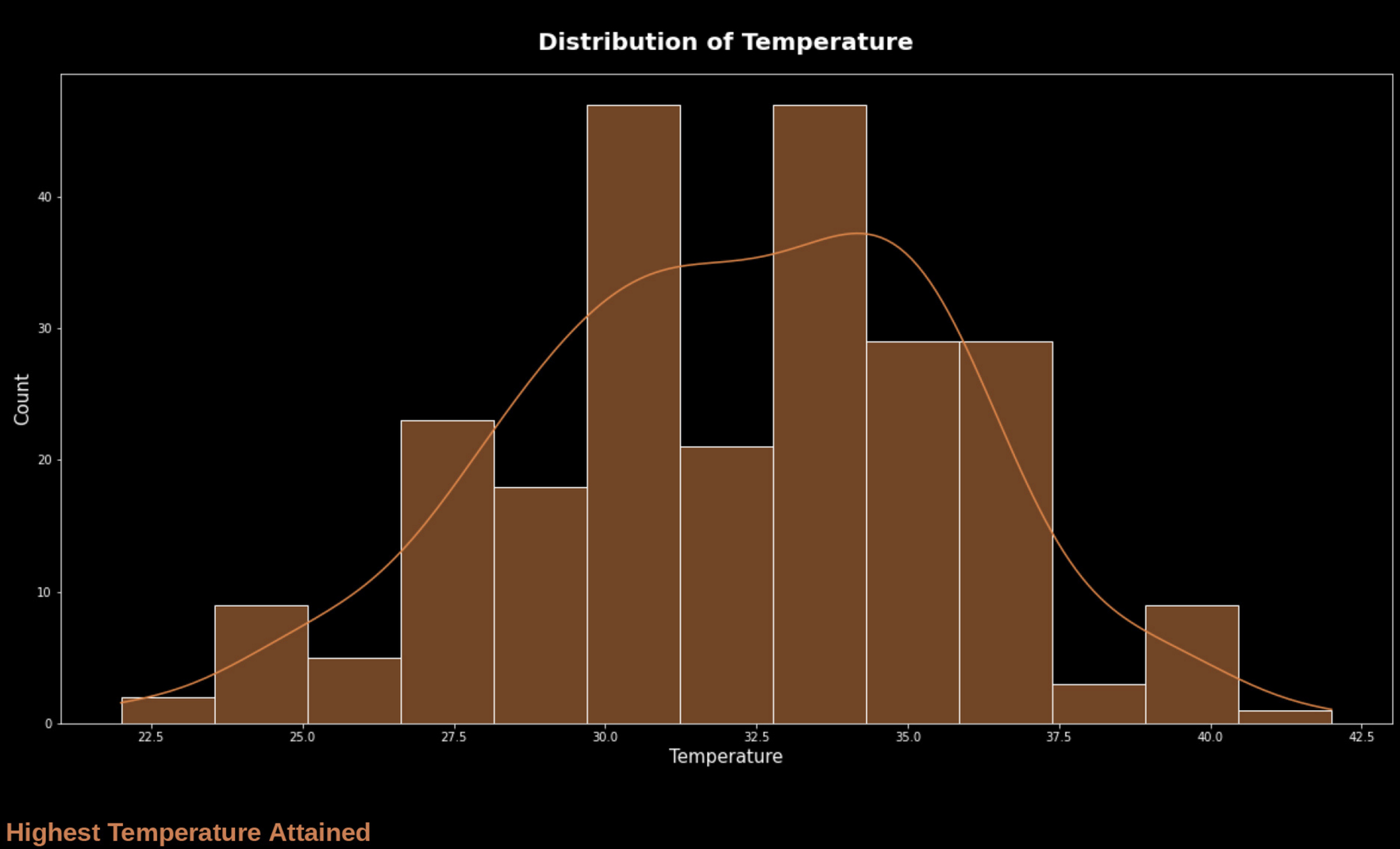
Correlation between attributes:



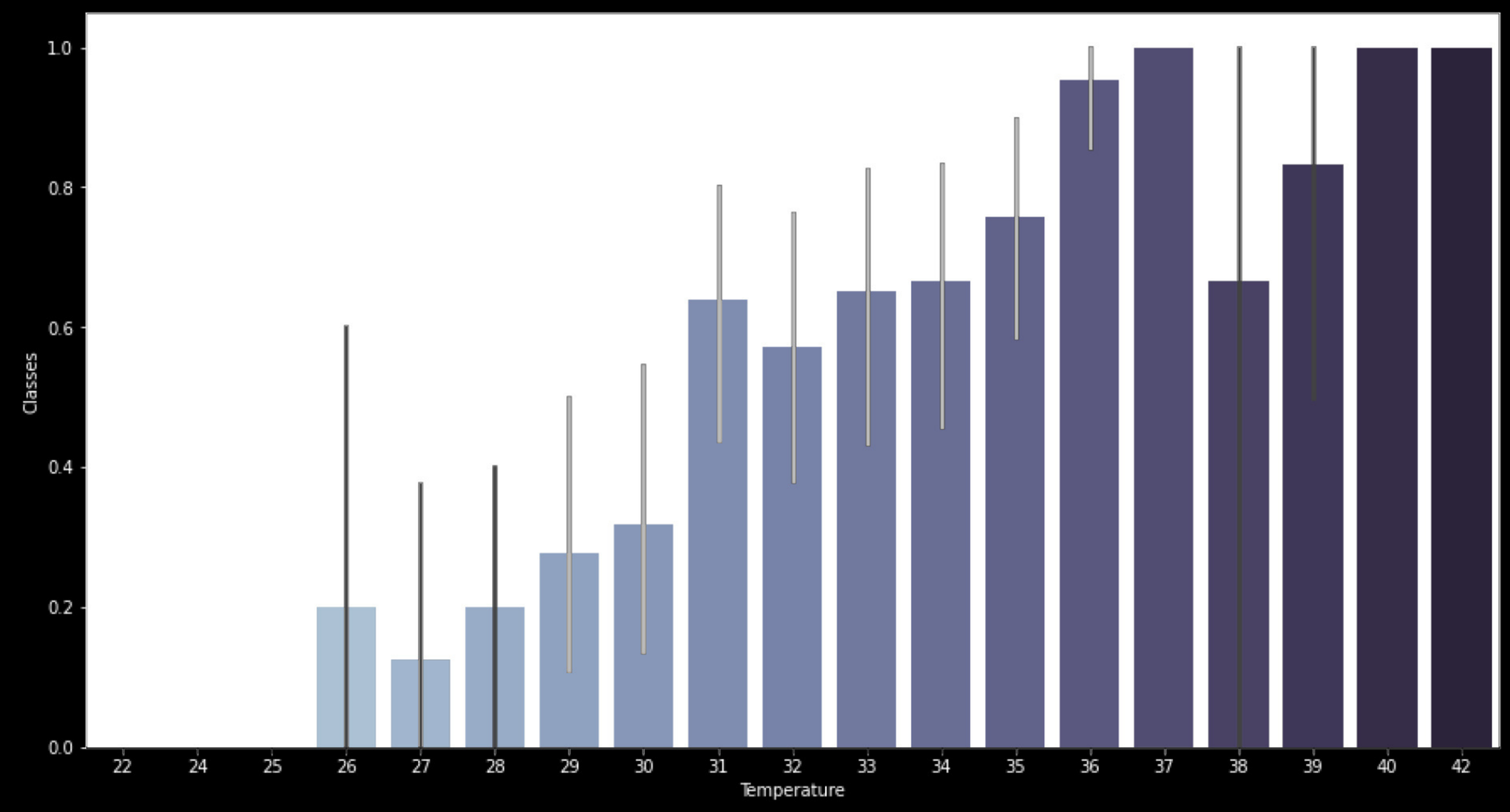
Visualization of Target feature



Visualization of Temperature Feature



Highest Temperature Attained



For more Visuals You can check here:

Machine-Learning-Implementation/Logistic Regression at main · ksdiwe/Machine-Learning-Implementation

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

https://github.com/ksdiwe/Machine-Learning-Implementation/tree/main/Logistic%20Regression

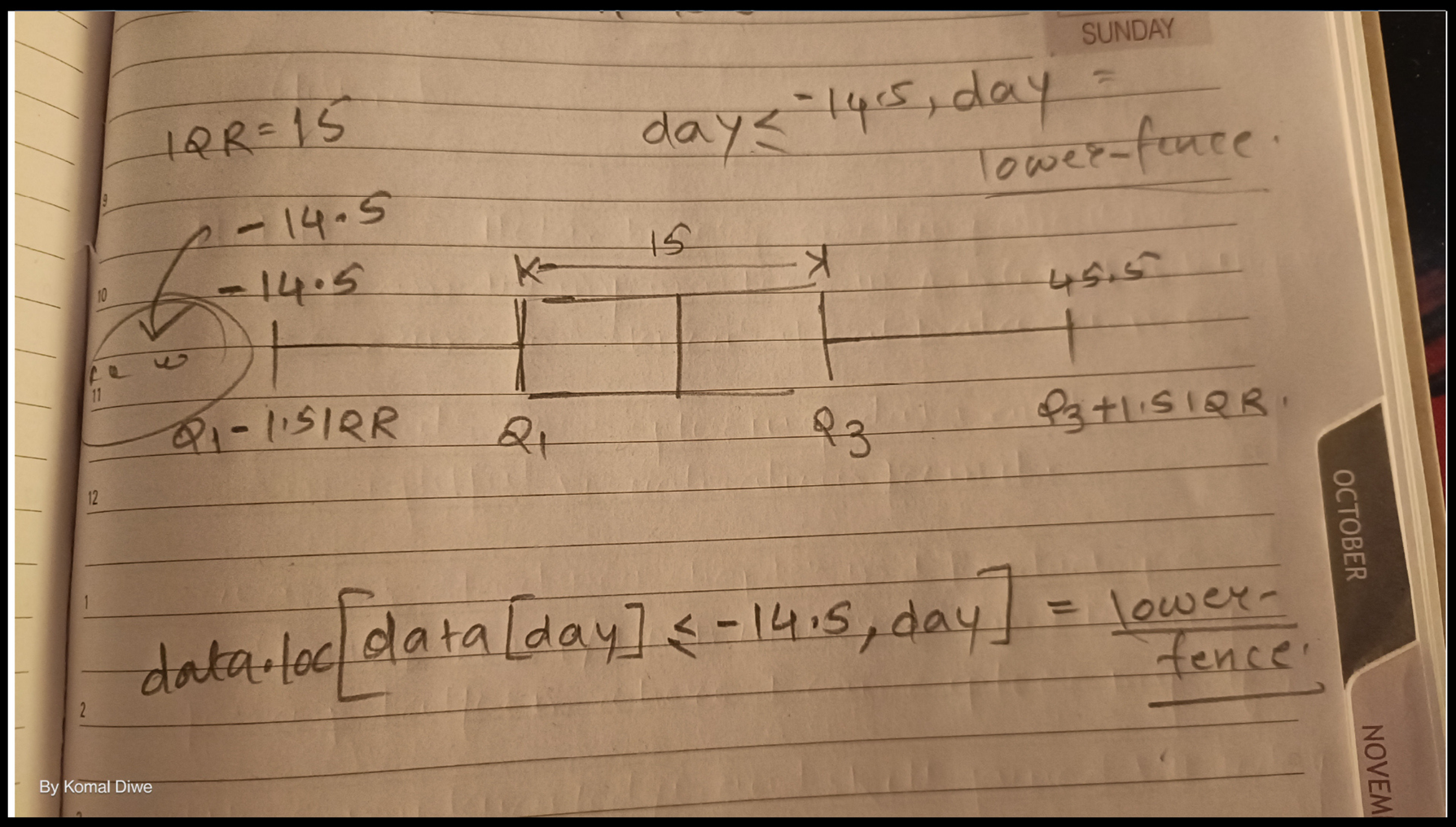
ksdiwe/**Machine-Learning-Implementation**

1 Contributor · 0 Issues · 2 Stars · 0 Forks

Handling Outliers

```python
def find_boundaries(data,column):
    IQR = data[column].quantile(0.75) - data[column].quantile(0.25)
    lower_boundary = data[column].quantile(0.25) - (1.5 * IQR)
    higher_boundary = data[column].quantile(0.75) + (1.5 * IQR)
    print(column, "---", "IQR --->",IQR)
    print("Lower Boundary:",lower_boundary)
    print("Higher Boundary:",  higher_boundary)
    print("-----------------------------------------------------------------------")
    data.loc[data[column] <= lower_boundary, column] = lower_boundary
    data.loc[data[column] >= higher_boundary, column] =  higher_boundary
```

For understanding function :



Logistic Regression Training Model

```
# train, test split data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state=42)

from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
from sklearn.model_selection import GridSearchCV
parameter = {'penalty':['l1','l2','elasticnet'],'C':[1,2,3,4,5,6,10,20,30,40,50],'max_iter':[100,200,300]}


classifier_regressor = GridSearchCV(classifier, param_grid=parameter, scoring='accuracy', cv=5)
# standardizing or feature selection

classifier_regressor.fit(X_train,y_train)
print(classifier_regressor.best_params_)
print(classifier_regressor.best_score_)
```

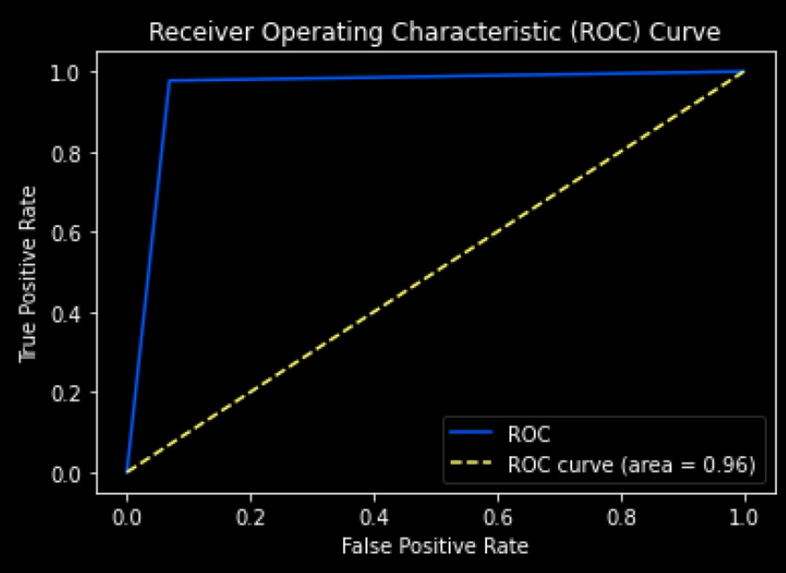
Prediction

```
y_pred = classifier_regressor.predict(X_test)

# accuracy score
from sklearn.metrics import accuracy_score, classification_report
score = accuracy_score(y_pred, y_test)

#classification report
print(classification_report(y_pred,y_test))
```

Performance Metrics

```
#confusion matrix
conf_Max = confusion_matrix(y_pred,y_test)

true_positive = conf_Max[0][0]
false_positive = conf_Max[0][1]
false_negative = conf_Max[1][0]
true_negative = conf_Max[1][1]

#accuracy
Accuracy = (true_positive + true_negative) / (true_positive +false_positive + false_negative + true_negative)

#precision
Precision = true_positive/(true_positive+false_positive)

#recall
Recall = true_positive/(true_positive+false_negative)

#F1-score
F1_Score = 2*(Recall * Precision) / (Recall + Precision)

#AUC
auc = roc_auc_score(y_pred, y_test)

#ROC
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
```

ROC Curve



Now, I created an Inbalance dataset from the original balanced dataset by splitting data in 9:1 ratio using train test split.

```
# creating imbalance
# 1. splitting data in 9:1 percent ratio using train test split
X1 = pd.DataFrame(df, columns = ['day', 'month','Temperature','RH','Ws','Rain','FFMC','DMC','DC','ISI','BUI','FWI','Region'])
y1 = pd.DataFrame(df, columns = ['Classes'])

X_train_imb, X_test_imb, y_train_imb, y_test_imb = train_test_split(X1, y1, test_size=0.10, random_state=17)
```

Replaced all 1’s in y_train as 0 and vice versa to create imbalance.

So, after cleaning and modeling, I came to the conclusion that

Performance of Logistic Model on Original Dataset:

	precision	recall	f1-score	support
0.0	0.93	0.96	0.95	28
1.0	0.98	0.96	0.97	45
accuracy			0.96	73
macro avg	0.95	0.96	0.96	73
weighted avg	0.96	0.96	0.96	73

Performance of Logistic Model on Balanced dataset which was created from Imbalanced dataset:

	precision	recall	f1-score	support
0	1.00	0.87	0.93	87
1	0.83	1.00	0.90	52
accuracy			0.92	139
macro avg	0.91	0.94	0.92	139
weighted avg	0.93	0.92	0.92	139

Observation:

- It seems that the model is `good` when `we predict from the original dataset`
- It seems that the model is very `bad` when we try to `predict from a balanced(created from an imbalanced dataset )`
- Also, the area under the curve `(AUC)` of the `original dataset` is `more`, so that model is good

For more in-depth details: 👇

GitHub

Machine-Learning-Implementation/Logistic Regression at main · ksdiwe/Machine-Learning-Implementation

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

https://github.com/ksdiwe/Machine-Learning-Implementation/tree/main/Logistic%20Regression

ksdiwe/**Machine-Learning-Implementation**

1 Contributor · 0 Issues · 2 Stars · 0 Forks