

Name - Akash Kumar Singh
Reg No - RA1911003010667

AI LAB
EXPERIMENT NO: 10
Implementation of a learning
algorithm

WORKING PRINCIPLE:-

Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis). To calculate best-fit line linear regression uses a traditional slope-intercept form. A regression line can be a Positive Linear Relationship or a Negative Linear Relationship. The goal of the linear regression algorithm is to get the best values for a_0 and a_1 to find the best fit line and the best fit line should have the least error. In Linear Regression, Mean Squared Error (MSE) cost function is used, which helps to figure out the best possible values for a_0 and a_1 , which provides the best fit line for the data points. Using the MSE function, we will change the values of a_0 and a_1 such that the MSE value settles at the minima. Gradient descent is a method of updating a_0 and a_1 to minimize the cost function(MSE).

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving **regression and classification problems** too.

The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by **learning simple decision rules** inferred from prior data(training data).

In Decision Trees, for predicting a class label for a record we start from the **root** of the tree. We compare the values of the root attribute with the record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

Types of Decision Trees

Types of decision trees are based on the type of target variable we have.

It can be of two types:

1. **Categorical Variable Decision Tree:** Decision Tree which has a categorical target variable then it called a **Categorical variable decision tree**.
2. **Continuous Variable Decision Tree:** Decision Tree has a continuous target variable then it is called **Continuous Variable Decision Tree**.

CODE:-

```
pip install termcolor
import pandas as pd # data processing
import numpy as np # working with arrays
import matplotlib.pyplot as plt # visualization
from termcolor import colored as cl # text customization
import itertools # advanced tools
from sklearn.preprocessing import StandardScaler # data normalization
from sklearn.model_selection import train_test_split # data split
from sklearn.tree import DecisionTreeClassifier # Decision tree algorithm
from sklearn.neighbors import KNeighborsClassifier # KNN algorithm
from sklearn.linear_model import LogisticRegression # Logistic regression
algorithm
from sklearn.svm import SVC # SVM algorithm
from sklearn.ensemble import RandomForestClassifier # Random forest tree
algorithm
from xgboost import XGBClassifier # XGBoost algorithm
from sklearn.metrics import confusion_matrix # evaluation metric
from sklearn.metrics import accuracy_score # evaluation metric
from sklearn.metrics import f1_score # evaluation metric
df = pd.read_csv('creditcard.csv')
df.drop('Time', axis = 1, inplace = True)
[{"metadata":{"trusted":false},"cell_type":"code","source":"print(df.head(
))\n","execution_count":12,"outputs":[{"name":"stdout","output_type":"stre
am","text":"          V1          V2          V3          V4          V5          V6
```

```

V7  \\n0 -1.359807 -0.072781 2.536347 1.378155 -0.338321 0.462388
0.239599  \n1 1.191857 0.266151 0.166480 0.448154 0.060018 -0.082361
-0.078803  \n2 -1.358354 -1.340163 1.773209 0.379780 -0.503198
1.800499 0.791461  \n3 -0.966272 -0.185226 1.792993 -0.863291 -0.010309
1.247203 0.237609  \n4 -1.158233 0.877737 1.548718 0.403034 -0.407193
0.095921 0.592941  \n\n      V8      V9      V10 ...      V21
V22      V23      V24  \\n0 0.098698 0.363787 0.090794 ... -
0.018307 0.277838 -0.110474 0.066928  \n1 0.085102 -0.255425 -0.166974
... -0.225775 -0.638672 0.101288 -0.339846  \n2 0.247676 -1.514654
0.207643 ... 0.247998 0.771679 0.909412 -0.689281  \n3 0.377436 -
1.387024 -0.054952 ... -0.108300 0.005274 -0.190321 -1.175575  \n4 -
0.270533 0.817739 0.753074 ... -0.009431 0.798278 -0.137458 0.141267
\n\n      V25      V26      V27      V28 Amount Class  \n0
0.128539 -0.189115 0.133558 -0.021053 149.62      0  \n1 0.167170
0.125895 -0.008983 0.014724 2.69      0  \n2 -0.327642 -0.139097 -
0.055353 -0.059752 378.66      0  \n3 0.647376 -0.221929 0.062723
0.061458 123.50      0  \n4 -0.206010 0.502292 0.219422 0.215153
69.99      0  \n\n[5 rows x 30 columns]\n"]]]]

```

```
cases = len(df)
```

```
nonfraud_count = len(df[df.Class == 0])
```

```
fraud_count = len(df[df.Class == 1])
```

```
fraud_percentage = round(fraud_count/nonfraud_count*100, 2)
```

```
print(cl('CASE COUNT', attrs = ['bold']))
```

```
print(cl('-----', attrs = ['bold']))
```

```
print(cl('Total number of cases are {}'.format(cases), attrs = ['bold']))
```

```
print(cl('Number of Non-fraud cases are {}'.format(nonfraud_count), attrs =
['bold']))
```

```
print(cl('Number of Non-fraud cases are {}'.format(fraud_count), attrs =
['bold']))
```

```
print(cl('Percentage of fraud cases is {}'.format(fraud_percentage), attrs =
['bold']))
```

```
print(cl('-----', attrs = ['bold']))
```

```
nonfraud_cases = df[df.Class == 0]
```

```
fraud_cases = df[df.Class == 1]
```

```
print(cl('CASE AMOUNT STATISTICS', attrs = ['bold']))
```

```
print(cl('-----', attrs = ['bold']))
```

```
print(cl('NON-FRAUD CASE AMOUNT STATS', attrs = ['bold']))
```

```
print(nonfraud_cases.Amount.describe())
```

```
print(cl('-----', attrs = ['bold']))
```

```
print(cl('FRAUD CASE AMOUNT STATS', attrs = ['bold']))
```

```
print(fraud_cases.Amount.describe())
```

```
print(cl('-----', attrs = ['bold']))
```

```
X = df.drop('Class', axis = 1).values
```

```
y = df['Class'].values
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
```

```
random_state = 0)
```

```
print(cl('X_train samples : ', attrs = ['bold']), X_train[:1])  
print(cl('X_test samples : ', attrs = ['bold']), X_test[0:1])  
print(cl('y_train samples : ', attrs = ['bold']), y_train[0:10])  
print(cl('y_test samples : ', attrs = ['bold']), y_test[0:10])
```

```
from sklearn.preprocessing import StandardScaler  
tree_model = DecisionTreeClassifier(max_depth = 4, criterion = 'entropy')  
tree_model.fit(X_train, y_train)  
tree_yhat = tree_model.predict(X_test)
```

2. K-Nearest Neighbors

```
n = 5
```

```
knn = KNeighborsClassifier(n_neighbors = n)  
knn.fit(X_train, y_train)  
knn_yhat = knn.predict(X_test)
```

3. Logistic Regression

```
lr = LogisticRegression()  
lr.fit(X_train, y_train)  
lr_yhat = lr.predict(X_test)
```

4. SVM

```
svm = SVC()  
svm.fit(X_train, y_train)  
svm_yhat = svm.predict(X_test)
```

5. Random Forest Tree

```
rf = RandomForestClassifier(max_depth = 4)  
rf.fit(X_train, y_train)  
rf_yhat = rf.predict(X_test)
```

6. XGBoost

```
xgb = XGBClassifier(max_depth = 4)  
xgb.fit(X_train, y_train)  
xgb_yhat = xgb.predict(X_test)  
print(cl('ACCURACY SCORE', attrs = ['bold']))
```

```

print(cl('-----', attrs =
['bold']))
print(cl('Accuracy score of the Decision Tree model is
{}'.format(accuracy_score(y_test, tree_yhat)), attrs = ['bold']))
print(cl('-----', attrs =
['bold']))
print(cl('Accuracy score of the KNN model is {}'.format(accuracy_score(y_test,
knn_yhat)), attrs = ['bold'], color = 'green'))
print(cl('-----', attrs =
['bold']))
print(cl('Accuracy score of the Logistic Regression model is
{}'.format(accuracy_score(y_test, lr_yhat)), attrs = ['bold'], color = 'red'))
print(cl('-----', attrs =
['bold']))
print(cl('Accuracy score of the SVM model is {}'.format(accuracy_score(y_test,
svm_yhat)), attrs = ['bold']))
print(cl('-----', attrs =
['bold']))
print(cl('Accuracy score of the Random Forest Tree model is
{}'.format(accuracy_score(y_test, rf_yhat)), attrs = ['bold']))
print(cl('-----', attrs =
['bold']))
print(cl('Accuracy score of the XGBoost model is
{}'.format(accuracy_score(y_test, xgb_yhat)), attrs = ['bold']))
print(cl('-----', attrs =
['bold']))
def plot_confusion_matrix(cm, classes, title, normalize = False, cmap =
plt.cm.Blues):
    title = 'Confusion Matrix of {}'.format(title)
    if normalize:
        cm = cm.astype(float) / cm.sum(axis=1)[:, np.newaxis]

    plt.imshow(cm, interpolation = 'nearest', cmap = cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation = 45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
            horizontalalignment = 'center',

```

```

        color = 'white' if cm[i, j] > thresh else 'black')

plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')

# Compute confusion matrix for the models

tree_matrix = confusion_matrix(y_test, tree_yhat, labels = [0, 1]) # Decision
Tree
knn_matrix = confusion_matrix(y_test, knn_yhat, labels = [0, 1]) # K-Nearest
Neighbors
lr_matrix = confusion_matrix(y_test, lr_yhat, labels = [0, 1]) # Logistic
Regression
svm_matrix = confusion_matrix(y_test, svm_yhat, labels = [0, 1]) # Support
Vector Machine
rf_matrix = confusion_matrix(y_test, rf_yhat, labels = [0, 1]) # Random Forest
Tree
xgb_matrix = confusion_matrix(y_test, xgb_yhat, labels = [0, 1]) # XGBoost

# Plot the confusion matrix

plt.rcParams['figure.figsize'] = (6, 6)
# 1. Decision tree

tree_cm_plot = plot_confusion_matrix(tree_matrix,
                                     classes = ['Non-Default(0)', 'Default(1)'],
                                     normalize = False, title = 'Decision Tree')
plt.savefig('tree_cm_plot.png')
plt.show()

# 2. K-Nearest Neighbors

knn_cm_plot = plot_confusion_matrix(knn_matrix,
                                    classes = ['Non-Default(0)', 'Default(1)'],
                                    normalize = False, title = 'KNN')
plt.savefig('knn_cm_plot.png')
plt.show()

# 3. Logistic regression

lr_cm_plot = plot_confusion_matrix(lr_matrix,
                                   classes = ['Non-Default(0)', 'Default(1)'],
                                   normalize = False, title = 'Logistic Regression')

```

```
plt.savefig('lr_cm_plot.png')
plt.show()
```

4. Support Vector Machine

```
svm_cm_plot = plot_confusion_matrix(svm_matrix,
                                     classes = ['Non-Default(0)', 'Default(1)'],
                                     normalize = False, title = 'SVM')
plt.savefig('svm_cm_plot.png')
plt.show()
```

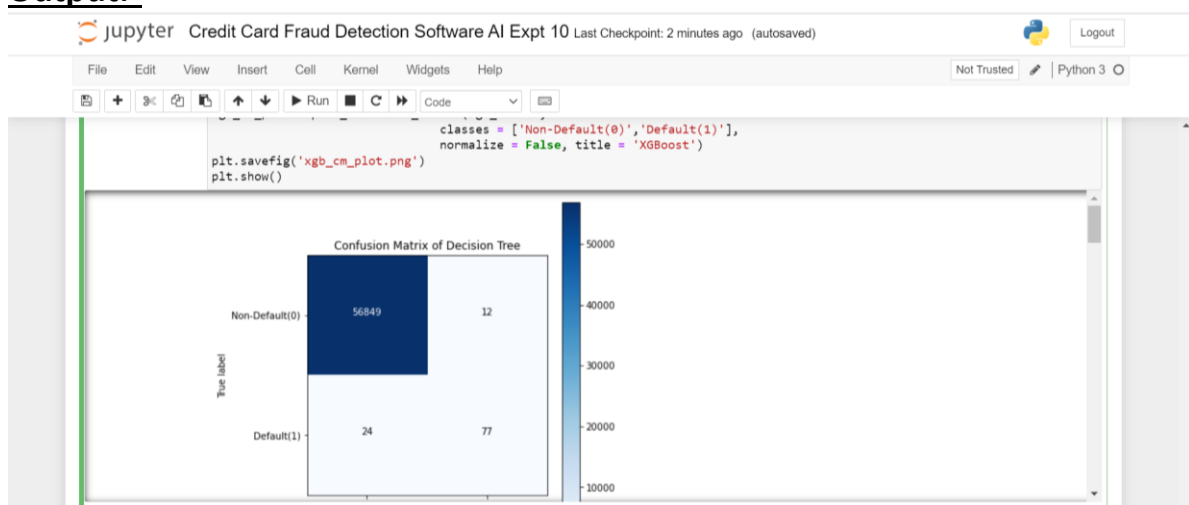
5. Random forest tree

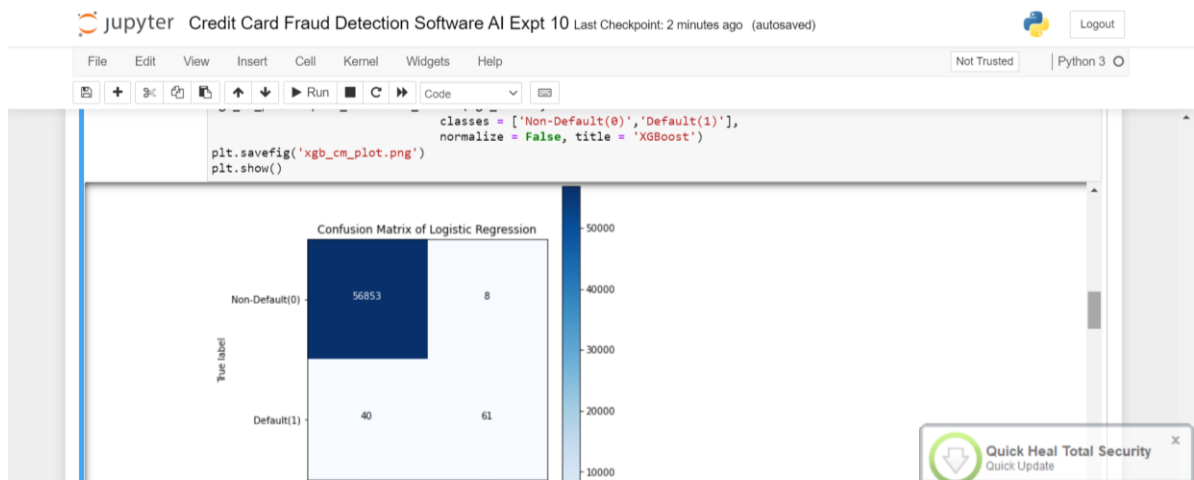
```
rf_cm_plot = plot_confusion_matrix(rf_matrix,
                                    classes = ['Non-Default(0)', 'Default(1)'],
                                    normalize = False, title = 'Random Forest Tree')
plt.savefig('rf_cm_plot.png')
plt.show()
```

6. XGBoost

```
xgb_cm_plot = plot_confusion_matrix(xgb_matrix,
                                    classes = ['Non-Default(0)', 'Default(1)'],
                                    normalize = False, title = 'XGBoost')
plt.savefig('xgb_cm_plot.png')
plt.show()
```

Output:-





RESULT:-

Hence, the Implementation of a machine learning algorithm is done successfully.