UCS 2611 Machine Learning Lab

Lab Test for Machine Learning : Predict Census Income

Name : **Mega V** Roll No : **3122 21 5001 51**

Aim

To Develop a python program to predict the income of a person using all the classification models (LR, PLA, MLP, KNN, SVM, Naïve Bayes) you have learnt. Interpret the model which works better for this dataset. Visualize the features from the dataset and interpret the results obtained by the model using Matplotlib library

Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.preprocessing import LabelEncoder

from sklearn.linear_model import Perceptron
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
```

```
1. Loading the dataset.                                  Using Pandas
2. Pre-Processing                                        Encoding ( Lable Encoder
) , Normalization, Handling missing values
3. Exploratory Data Analysis.                            Data Description and
Data Frequency Graph
4. Feature Engineering techniques.                       Select K Best
5. Split the data into training, testing sets.           Training and Testing
dataset spliting
```

```
6. Train the model.                                  PLA, MLP, KNN, SVM, Naïve
Bayes) yo
7. Test the model.                                   PLA, MLP, KNN, SVM, Naïve
Bayes) yo
8. Measure the performance of the trained model.    Training and Testing
Accuracy
9. Represent the training and testing results using ROC curves.   ROC Curve
10. Results.  Neither Overfitting nor underfitting
```

```
df=pd.read_csv("/home/test2/Desktop/Test/AdultCensusIncome.csv")
print("The Shape of The Dataset is : ",df.shape)
```

```
The Shape of The Dataset is :  (32561, 15)
```

```
print(df.isnull().sum())
```

```
age               0
workclass         0
fnlwgt            0
education         0
education.num     0
marital.status    0
occupation        0
relationship      0
race              0
sex               0
capital.gain      0
capital.loss      0
hours.per.week    0
native.country    0
income            0
dtype: int64
```

```
df.dtypes
```

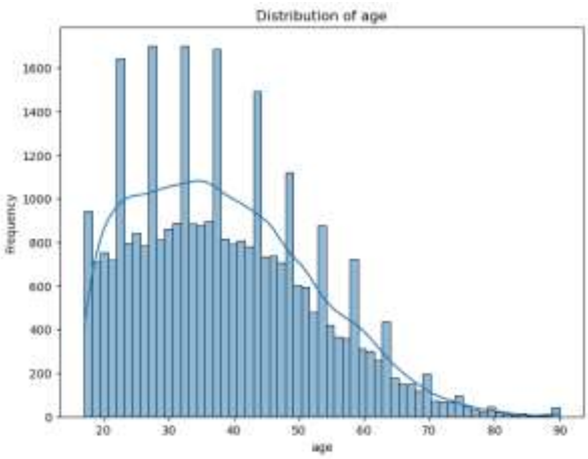
```
age                int64
workclass         object
fnlwgt             int64
education         object
education.num      int64
marital.status    object
occupation        object
relationship      object
race              object
sex               object
capital.gain       int64
capital.loss       int64
hours.per.week     int64
native.country    object
income            object
dtype: object
```

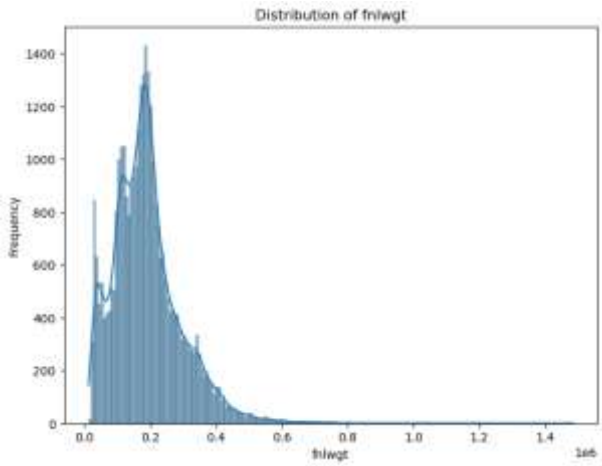
```
df=df.drop(columns=["marital.status","relationship","race","native.country"])
df.info()
```

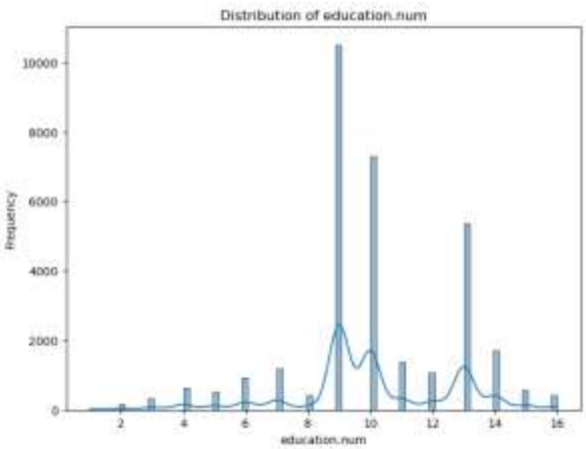
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64
 3   education       32561 non-null  object
 4   education.num   32561 non-null  int64
 5   occupation      32561 non-null  object
 6   sex             32561 non-null  object
 7   capital.gain    32561 non-null  int64
 8   capital.loss    32561 non-null  int64
 9   hours.per.week  32561 non-null  int64
 10  income          32561 non-null  object
dtypes: int64(6), object(5)
memory usage: 2.7+ MB
```

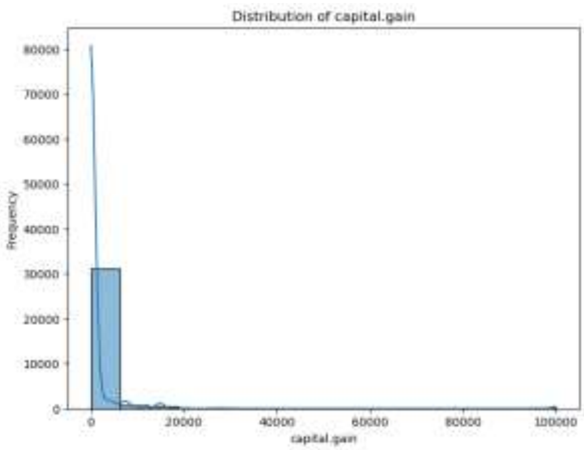
```
data = pd.read_csv("/home/test2/Desktop/Test/AdultCensusIncome.csv")

print(data.isnull().sum())

numeric_columns = data.select_dtypes(include=['int64', 'float64']).columns
for col in numeric_columns:
    plt.figure(figsize=(8, 6))
    sns.histplot(data[col], kde=True)
    plt.title(f"Distribution of {col}")
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.show()

sns.pairplot(data)
plt.show()
```

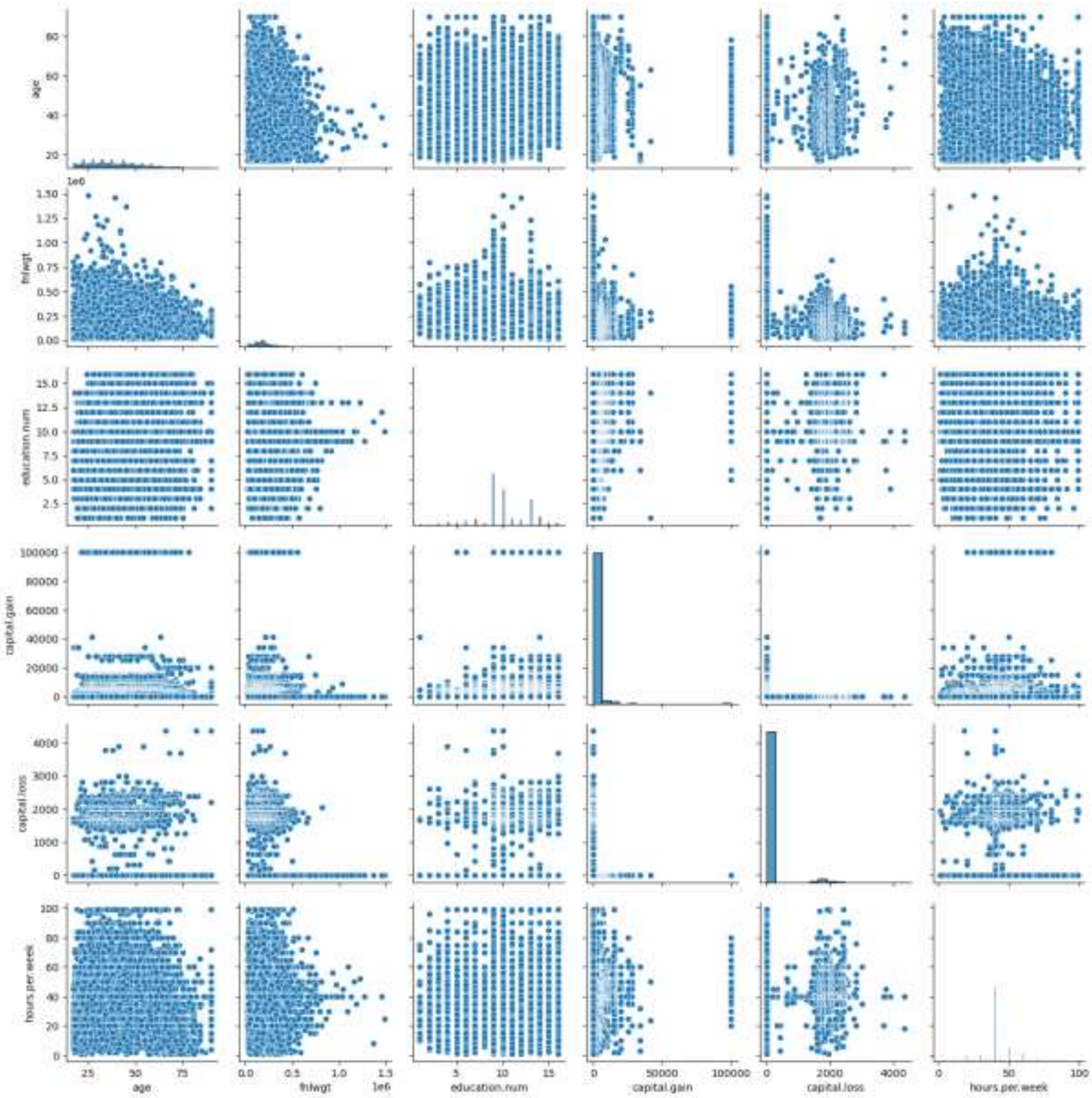
```
age               0
workclass         0
fnlwgt            0
education         0
education.num     0
marital.status    0
occupation        0
relationship      0
race              0
sex               0
capital.gain      0
capital.loss      0
hours.per.week    0
native.country    0
income            0
dtype: int64
```












```
X_train,X_test,Y_train,Y_test=train_test_split(x,y,test_size=0.3,random_state=
42)
```

```
label_encoder = LabelEncoder()
X_train_encoded = X_train.copy()
X_test_encoded = X_test.copy()
for col in X_train_encoded.select_dtypes(include=["object"]).columns:
    X_train_encoded[col] = label_encoder.fit_transform(X_train[col])
    X_test_encoded[col] = label_encoder.transform(X_test[col])

k_best = SelectKBest(score_func=chi2, k=5)
X_train_selected = k_best.fit_transform(X_train_encoded, y_train)
```

```
selected_feature_indices = k_best.get_support(indices=True)

selected_features = X_train_encoded.columns[selected_feature_indices]

print("Selected Features:", selected_features)
```

```
Selected Features: Index(['age', 'fnlwgt', 'capital.gain', 'capital.loss', 'hours.per.week'], dtype='object')
```

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X_train_selected_normalized = X_train_encoded.iloc[:,
selected_feature_indices].copy()
X_test_selected_normalized = X_test_encoded.iloc[:,
selected_feature_indices].copy()

X_train_selected_normalized =
scaler.fit_transform(X_train_selected_normalized)

X_test_selected_normalized = scaler.transform(X_test_selected_normalized)
```

```
X_train=X_train_selected_normalized
X_test=X_test_selected_normalized
```

```
print(X_train.shape)
print(X_test.shape)
```

```
pla_classifier = Perceptron()

pla_classifier.fit(X_train, y_train)

pla_predictions = pla_classifier.predict(X_test)

pla_accuracy = pla_classifier.score(X_test, y_test)
pla_training_accuracy = pla_classifier.score(X_train, y_train)

print("Perceptron Training Accuracy : ", pla_training_accuracy)
print("Perceptron Testing Accuracy  : ", pla_accuracy)
```

```
Perceptron Training Accuracy :  0.7947086697086697
Perceptron Testing Accuracy  :  0.7984440577336472
```

```
MLA
```

```
mlp_classifier = MLPClassifier()

mlp_classifier.fit(X_train, y_train)
mlp_predictions = mlp_classifier.predict(X_test)

mlp_accuracy = mlp_classifier.score(X_test, y_test)

mlp_classifier.fit(X_train, y_train)
mlp_training_accuracy = mlp_classifier.score(X_train, y_train)

print("MLP Training Accuracy : ", mlp_training_accuracy)
print("MLP Testing Accuracy  : ", mlp_accuracy)
```

```
MLP Training Accuracy :  0.8067304317304317
MLP Testing Accuracy  :  0.8094994369945747
```

KNN

```
knn_classifier = KNeighborsClassifier()

knn_classifier.fit(X_train, y_train)
knn_predictions = knn_classifier.predict(X_test)

knn_accuracy = knn_classifier.score(X_test, y_test)

knn_classifier.fit(X_train, y_train)
knn_training_accuracy = knn_classifier.score(X_train, y_train)

print("KNN Training Accuracy : ", knn_training_accuracy)
print("KNN Testing Accuracy  : ", knn_accuracy)
```

```
KNN Training Accuracy :  0.8415233415233415
KNN Testing Accuracy  :  0.788002866209438
```

SVM

```
from sklearn.svm import SVC

svm_kernels = ['linear', 'poly', 'rbf', 'sigmoid']
svm_classifiers = {}

for kernel in svm_kernels:
    svm_classifier = SVC(kernel=kernel)
    svm_classifier.fit(X_train, y_train)
    svm_predictions = svm_classifier.predict(X_test)
    svm_accuracy = svm_classifier.score(X_test, y_test)
    svm_classifiers[kernel] = {'model': svm_classifier, 'accuracy':
svm_accuracy}

for kernel, info in svm_classifiers.items():
    print(f"SVM ({kernel.capitalize()} Kernel) Accuracy: {info['accuracy']}")
```

```
SVM (Linear Kernel) Accuracy: 0.8029481011362473
SVM (Poly Kernel) Accuracy: 0.8100112601085065
SVM (Rbf Kernel) Accuracy: 0.8117514586958747
SVM (Sigmoid Kernel) Accuracy: 0.5492885658716348
```

KMeans

```
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)

nb_predictions = nb_classifier.predict(X_test)
nb_accuracy = nb_classifier.score(X_test, y_test)
nb_classifier.fit(X_train, y_train)
nb_training_accuracy = nb_classifier.score(X_train, y_train)

print("Naïve Bayes Training Accuracy  : ", nb_training_accuracy)
print("Naive Bayes Testing Accuracy   : ", nb_accuracy)
```

```
Naïve Bayes Training Accuracy  :  0.7951474201474201
Naive Bayes Testing Accuracy   :  0.7958849421639881
```

```
print("Perceptron Accuracy   : ", round(pla_accuracy*100,2))
print("MLP Accuracy          : ", round(mlp_accuracy*100,2))
print("KNN Accuracy          : ", round(knn_accuracy*100,2))
print("Naive Bayes Accuracy  : ", round(nb_accuracy*100,2))
```

```
print()
print()

for kernel, info in svm_classifiers.items():
    print(f"SVM ({kernel.capitalize()} Kernel) Accuracy :
{round(info['accuracy']*100,2)}")
```

```
Perceptron Accuracy   :  79.84
MLP Accuracy          :  81.0
KNN Accuracy          :  78.8
Naive Bayes Accuracy  :  79.59


SVM (Linear Kernel) Accuracy : 80.29
SVM (Poly Kernel) Accuracy : 81.0
SVM (Rbf Kernel) Accuracy : 81.18
SVM (Sigmoid Kernel) Accuracy : 54.93
```

Comparison Of Models

From the Above results all the models are having the accuracy nearly 75 - 80 percentage. So all the models are works good for the given Salaray predicting Dataset.

From the abovce models The SVM Using the RBF Kernal comparitively has the higher Accuracy. and then the MLP model is very near to the SVM using RBF. The both are having the same results.

The SVM using the other two kernals like Linear and poly are also works well like the above two models.

The Perceptron and the KNN and Naive Bayes comparitively have the less accuracy then the SVM using RBF.

Thge worst model is the model SVM using the Sigmoid Kernal. This has only the 55 percentage accuracy.

Best Models :

`1)  SVM ( RBF )`
`2)  MLP`
`3)  SVM ( Poly )`
`4)  SVM ( Linear )`
`5)  Perceptron`
`6)  KNN`
`7)  SVM ( Sigmoid )`

Therefor By comparitively

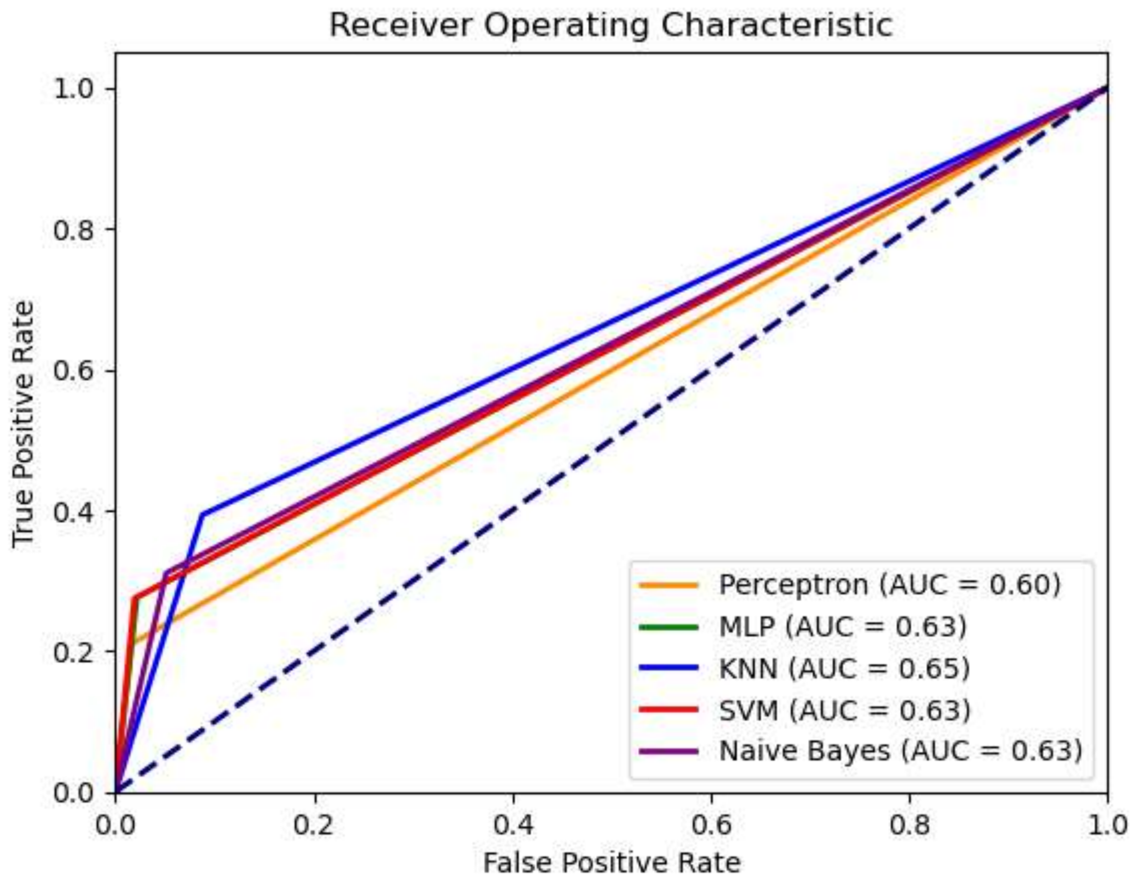
`    1 ) Best Model   :  SVM ( RBF ) and MLP`

`    2 ) Worst Model  :  SVM ( Sigmoid )`

```
import numpy as np
```

```python
y_test_binary = y_test.replace({'<=50K': 0, '>50K': 1})

pla_predictions_numeric = np.where(pla_predictions == '<=50K', 0, 1)
mlp_predictions_numeric = np.where(mlp_predictions == '<=50K', 0, 1)
knn_predictions_numeric = np.where(knn_predictions == '<=50K', 0, 1)
svm_predictions_numeric = np.where(svm_predictions == '<=50K', 0, 1)
nb_predictions_numeric = np.where(nb_predictions == '<=50K', 0, 1)

pla_fpr, pla_tpr, _ = roc_curve(y_test_binary, pla_predictions_numeric)
pla_roc_auc = auc(pla_fpr, pla_tpr)

mlp_fpr, mlp_tpr, _ = roc_curve(y_test_binary, mlp_predictions_numeric)
mlp_roc_auc = auc(mlp_fpr, mlp_tpr)

knn_fpr, knn_tpr, _ = roc_curve(y_test_binary, knn_predictions_numeric)
knn_roc_auc = auc(knn_fpr, knn_tpr)

svm_fpr, svm_tpr, _ = roc_curve(y_test_binary, svm_predictions_numeric)
svm_roc_auc = auc(svm_fpr, svm_tpr)

nb_fpr, nb_tpr, _ = roc_curve(y_test_binary, nb_predictions_numeric)
nb_roc_auc = auc(nb_fpr, nb_tpr)
```

```python
plt.figure()
plt.plot(pla_fpr, pla_tpr, color='darkorange', lw=2, label='Perceptron (AUC =
%0.2f)' % pla_roc_auc)
plt.plot(mlp_fpr, mlp_tpr, color='green', lw=2, label='MLP (AUC = %0.2f)' %
mlp_roc_auc)
plt.plot(knn_fpr, knn_tpr, color='blue', lw=2, label='KNN (AUC = %0.2f)' %
knn_roc_auc)
plt.plot(svm_fpr, svm_tpr, color='red', lw=2, label='SVM (AUC = %0.2f)' %
svm_roc_auc)
plt.plot(nb_fpr, nb_tpr, color='purple', lw=2, label='Naive Bayes (AUC =
%0.2f)' % nb_roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```




```
ROC Curve

AUC = 1: The classifier has excellent performance
AUC > 0.5: The classifier performs better than random guessing
AUC = 0.5: The classifier performs no better than random guessing.
AUC < 0.5: The classifier's performance is worse than random guessing

From the ROC Curve we can classify that all the models are works better that
the Random Classifier.
```

```
Comparison Of Models

From the Above results all the models are having the accuracy nearly 75 - 80
percentage.
So all the models are works good for the given Salaray predicting Dataset.
```

```
From the abovce models The SVM Using the RBF Kernal comparitively has the
higher Accuracy. and then the MLP model is very near to the SVM using RBF. The
both are having the same results.

The SVM using the other two kernals like Linear and poly are also works well
like the above two models.

The Perceptron and the KNN and Naive Bayes comparitively have the less
accuracy then the SVM using RBF.

Thge worst model is the model SVM using the Sigmoid Kernal. This has only the
55 percentage accuracy.

Best Models :

    1)  SVM ( RBF )
    2)  MLP
    3)  SVM ( Poly )
    4)  SVM ( Linear )
    5)  Perceptron
    6)  KNN
    7)  SVM ( Sigmoid )

Therefor By comparitively

        1 ) Best Model   :  SVM ( RBF ) and MLP

        2 ) Worst Model  :  SVM ( Sigmoid )
```

```
Overfitting and Underfitting

Perceptron Training Accuracy :  0.7947086697086697
Perceptron Testing Accuracy  :  0.7984440577336472

MLP Training Accuracy :  0.8067304317304317
MLP Testing Accuracy  :  0.8094994369945747

KNN Training Accuracy :  0.8415233415233415
KNN Testing Accuracy  :  0.788002866209438

Naïve Bayes Training Accuracy  :  0.7951474201474201
Naive Bayes Testing Accuracy   :  0.7958849421639881
```

```
From the aboce results all the models nearly have the same traing and testoing
accuracies. So the models are neither Overfitting Nor Underfitting
```

Learning Outcome

Understanding the importance of data preprocessing steps like encoding categorical variables, handling missing values, and normalization for building effective machine learning models.

Utilizing feature selection techniques like SelectKBest to choose the most relevant features for improving model performance.

Experimenting with different classification algorithms like Perceptron, MLP, KNN, SVM, and Naïve Bayes to compare their performance on the given dataset.

Assessing model performance using metrics such as accuracy and ROC curves, understanding the significance of ROC curves and AUC scores in evaluating classifier performance.

Analyzing the results obtained from different models and interpreting their accuracies, identifying which models perform better and which ones need improvement.

Github Link

https://github.com/MegaVenkatachalam/Machine-Learning-Laboratory/tree/main/Lab%20test