# gender-identification-project

September 2, 2024

## 1 Regularization - Case Study

Classification by applying Logistic Regression with the Regularization technique (Lasso (L1), Ridge (L2), and Elastic Net) (Identify the Gender based on the data)

One of the organizations is looking to classify the gender based on the various facial features like nose, forehead, etc. They are having the online business and wanted to implement the same so that if any new customer or individual comes into their website, they will be asked to turn on the camera so that their face measurement can be taken using which the organization wants to create a model. Once a new customer comes in, the machine will be able to predict if the customer is Male or Female.

I have to apply the regularization technique to solve this problem statement.

Problem Statement There is a business requirement where we need to classify the gender based on the various features using the concept of regularization. Here we are going to apply the L1 and L2 regularization and using logistic regression we are going to apply the same. There are around 5000 records on which we need to create the model and come up with the solution.

Data Dictionary long_hair - Length of hair forehead_width_cm - Forehead width of individual forehead_height_cm - Forehead height of individual nose_wide - Nose width of individual nose_long - Nose length of the individual lips_thin - lips structure of the individual distance_nose_to_lip_long - Distance from nose to lip gender - Dependent variables i.e. Gender

[147]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
import seaborn as sns
import plotly.express as px
```

[2]:
```
df = pd.read_csv("C:
↪\\Users\\sharm\\Downloads\\regularization\\gender_classification_v7.csv")
df.head()
```

[2]:

| | long_hair | forehead_width_cm | forehead_height_cm | nose_wide | nose_long | \ |
|---|---|---|---|---|---|---|
| 0 | 1 | 11.8 | 6.1 | 1 | 0 | |
| 1 | 0 | 14.0 | 5.4 | 0 | 0 | |
| 2 | 0 | 11.8 | 6.3 | 1 | 1 | |
| 3 | 0 | 14.4 | 6.1 | 0 | 1 | |

```
4          1               13.5                 5.9          0          0

   lips_thin  distance_nose_to_lip_long  gender
0          1                         1    Male
1          1                         0  Female
2          1                         1    Male
3          1                         1    Male
4          0                         0  Female
```

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5001 entries, 0 to 5000
Data columns (total 8 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   long_hair                  5001 non-null   int64
 1   forehead_width_cm          5001 non-null   float64
 2   forehead_height_cm         5001 non-null   float64
 3   nose_wide                  5001 non-null   int64
 4   nose_long                  5001 non-null   int64
 5   lips_thin                  5001 non-null   int64
 6   distance_nose_to_lip_long  5001 non-null   int64
 7   gender                     5001 non-null   object
dtypes: float64(2), int64(5), object(1)
memory usage: 312.7+ KB
```

[4]:
```
df.isnull().sum()
```

[4]:
```
long_hair                    0
forehead_width_cm            0
forehead_height_cm           0
nose_wide                    0
nose_long                    0
lips_thin                    0
distance_nose_to_lip_long    0
gender                       0
dtype: int64
```

[5]:
```
df.duplicated().sum()
```

[5]:
```
1768
```

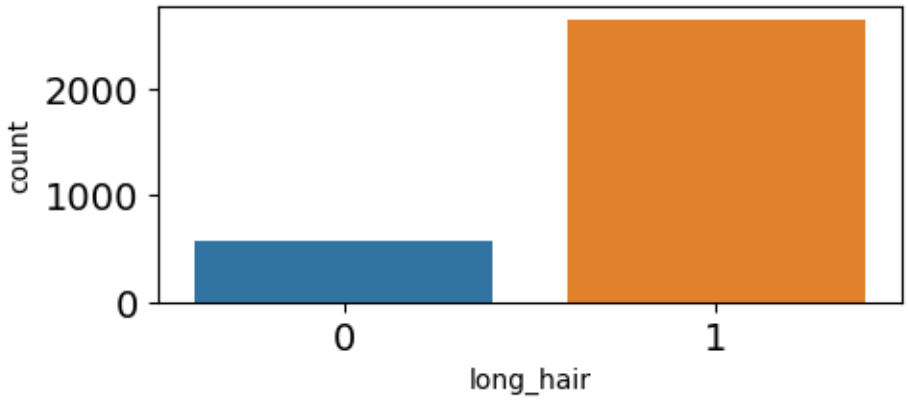
[6]:
```
df.drop_duplicates(inplace=True)
```

[7]:
```
for i in df.columns:
    print(i, df[i].nunique())
```

```
long_hair 2
forehead_width_cm 42
forehead_height_cm 21
nose_wide 2
nose_long 2
lips_thin 2
distance_nose_to_lip_long 2
gender 2
```

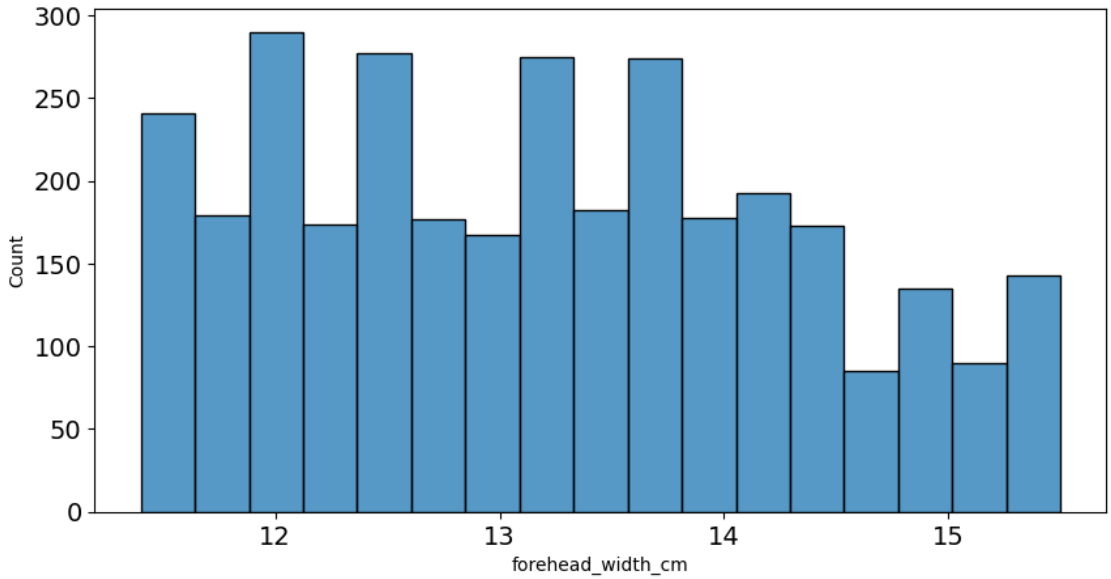
[8]:
```
plt.figure(figsize=(5,2))
plt.tick_params(axis='both', which='major', labelsize=14)
sns.countplot(x = df['long_hair'])
```

[8]: <Axes: xlabel='long_hair', ylabel='count'>




[9]:
```
plt.figure(figsize=(10,5))
plt.tick_params(axis='both', which='major', labelsize=14)
sns.histplot(x = df['forehead_width_cm'])
```

[9]: <Axes: xlabel='forehead_width_cm', ylabel='Count'>



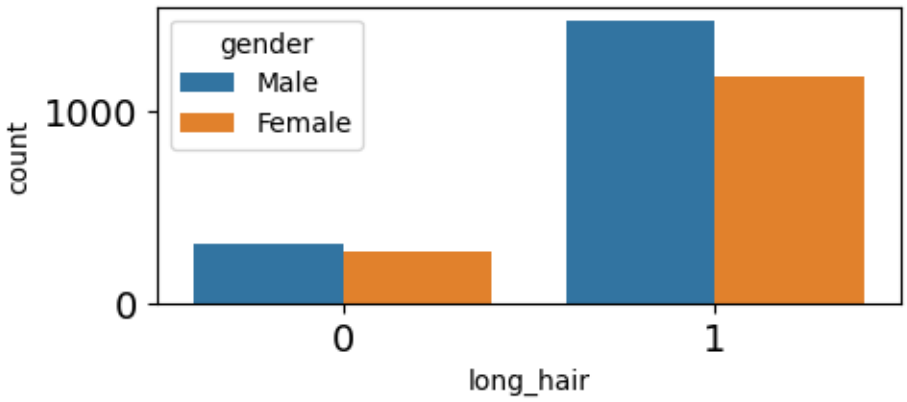

[ ]:

[10]:
```
plt.figure(figsize=(10,4))
fig = px.pie(df, names='gender', labels={'gender':'Gender'},␣
 ↪color_discrete_sequence = px.colors.sequential.RdBu)
fig.update_traces(textposition='inside', textinfo='percent+label')
fig.show()
```

```
<Figure size 1000x400 with 0 Axes>
```

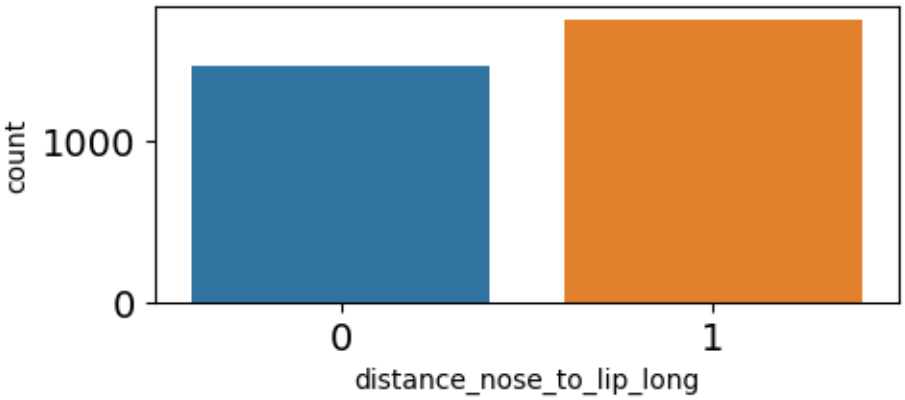
[11]:
```
plt.figure(figsize=(5,2))
plt.tick_params(axis='both', which='major', labelsize=14)
sns.countplot(hue = df['gender'], x= df['long_hair'])
```

[11]: <Axes: xlabel='long_hair', ylabel='count'>



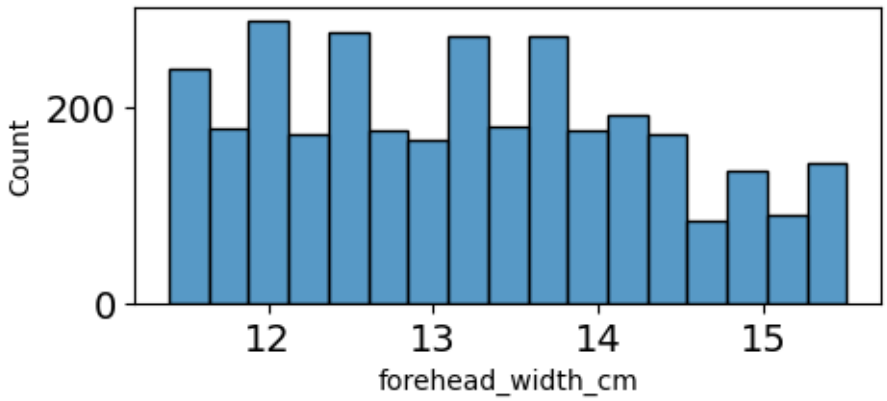
[12]:
```
plt.figure(figsize=(5,2))
plt.tick_params(axis='both', which='major', labelsize=14)
sns.countplot(x = df['distance_nose_to_lip_long'])
```

[12]: <Axes: xlabel='distance_nose_to_lip_long', ylabel='count'>




[13]:
```
plt.figure(figsize=(5,2))
plt.tick_params(axis='both', which='major', labelsize=14)
sns.histplot(x = df['forehead_width_cm'])
```

[13]: <Axes: xlabel='forehead_width_cm', ylabel='Count'>



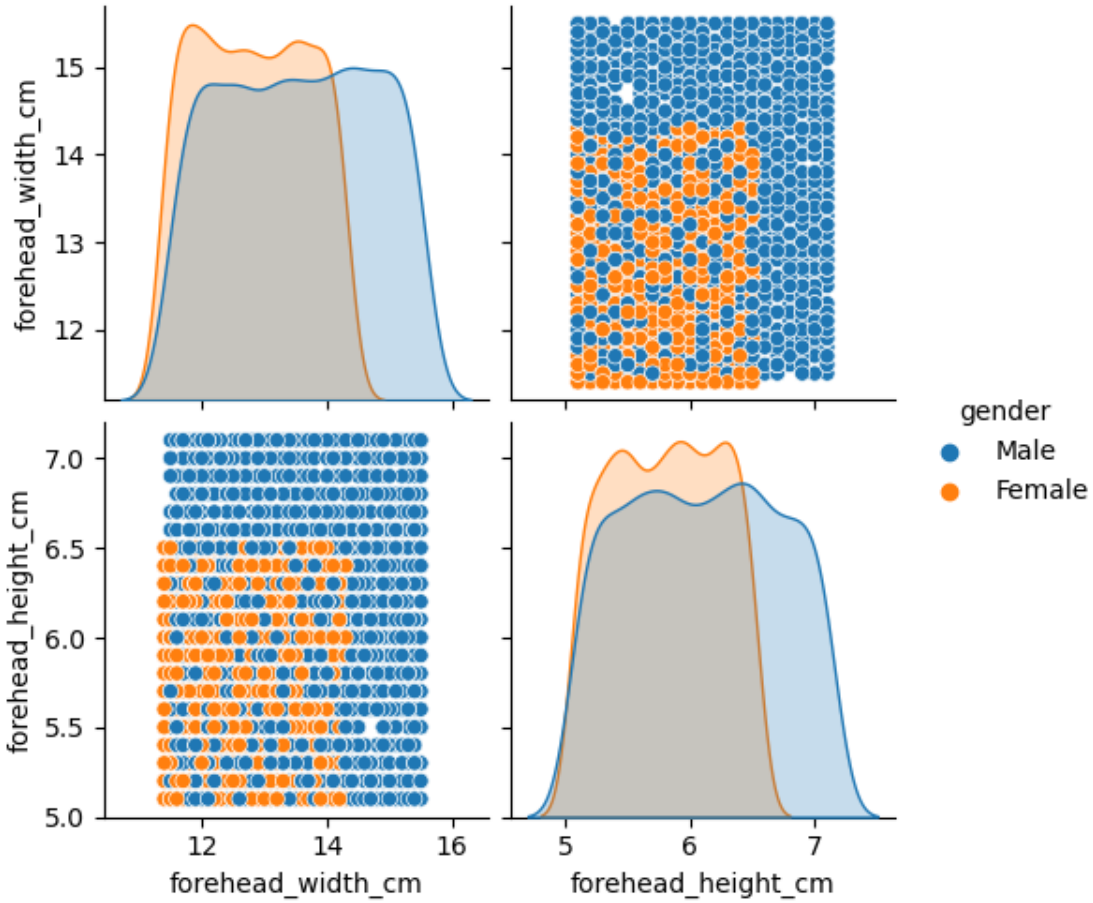

[14]:
```
fig = px.scatter(df,df['forehead_width_cm'], color='forehead_width_cm')
fig.show()
```

[16]:
```
plt.figure(figsize=(10,4))
sns.pairplot(df[['forehead_width_cm','forehead_height_cm', 'gender']],␣
 ↪hue='gender')
plt.show()
```

```
C:\ProgramData\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning:

The figure layout has changed to tight


<Figure size 1000x400 with 0 Axes>
```




[17]:
```
# Convert categorical to numerical
from sklearn.preprocessing import StandardScaler, LabelEncoder
le = LabelEncoder()
df['gender'] = le.fit_transform(df['gender'])
df
```

[17]:
```
      long_hair  forehead_width_cm  forehead_height_cm  nose_wide  nose_long  \
0             1               11.8                 6.1          1          0
1             0               14.0                 5.4          0          0
2             0               11.8                 6.3          1          1
3             0               14.4                 6.1          0          1
4             1               13.5                 5.9          0          0
…           …                  …                    …          …          …
4986          1               11.7                 6.1          1          1
4990          1               12.6                 5.7          0          0
4992          1               14.1                 7.0          1          1
4993          1               11.6                 5.9          0          0
4995          1               12.3                 6.9          0          1

      lips_thin  distance_nose_to_lip_long  gender
0             1                         1       1
1             1                         0       0
2             1                         1       1
3             1                         1       1
4             0                         0       0
…           …                         …       …
4986          0                         1       1
4990          1                         0       0
4992          1                         1       1
4993          0                         1       0
4995          1                         1       1

[3233 rows x 8 columns]
```

[27]:
```
# Separating the numerical and categorical columns
from sklearn.preprocessing import StandardScaler
def data_type(df):
 """
 Function to identify the numerical and categorical data columns
 :param dataset: Dataframe:return: list of numerical and categorical columns
 """
 numerical = []
 categorical = []
 for i in df.columns:
        if df[i].dtype == 'int64' or df[i].dtype =='float64':
            numerical.append(i)
        else:
            categorical.append(i)
            return numerical, categorical
numerical, categorical = data_type(df)
# Identifying the binary columns and ignoring them from scaling
def binary_columns(df):
 """
```

```
 Generates a list of binary columns in a dataframe.
 """
 binary_cols = []
 for col in df.select_dtypes(include=['int', 'float']).columns:
    unique_values = df[col].unique()
    if np.in1d(unique_values, [0, 1]).all():
        binary_cols.append(col)
        return binary_cols
binary_cols = binary_columns(df)
# Remove the binary columns from the numerical columns
numerical = [i for i in numerical if i not in binary_cols]
def encoding(df, categorical):
 """
 Function to automate the process of encoding the categorical data
 :param dataset: Dataframe
 :param categorical: List of categorical columns
 :return: Dataframe
 """
 for i in categorical:
        df[i] = df[i].astype('category')
        df[i] = df[i].cat.codes
        return df
df = encoding(df, categorical)
def feature_scaling(df, numerical):
 """
 Function to automate the process of feature scaling the numerical
data
:param dataset: Dataframe
 :param numerical: List of numerical columns
 :return: Dataframe
 """
 sc_x = StandardScaler()
 df[numerical] = sc_x.fit_transform(df[numerical])
 return df
df = feature_scaling(df, numerical)
df
```

[27]:

| | long_hair | forehead_width_cm | forehead_height_cm | nose_wide | nose_long | \ |
|---|---|---|---|---|---|---|
| 0 | 1 | -1.270095 | 0.243699 | 0.929772 | -1.122871 | |
| 1 | 0 | 0.701549 | -1.029713 | -1.075533 | -1.122871 | |
| 2 | 0 | -1.270095 | 0.607531 | 0.929772 | 0.890574 | |
| 3 | 0 | 1.060029 | 0.243699 | -1.075533 | 0.890574 | |
| 4 | 1 | 0.253448 | -0.120133 | -1.075533 | -1.122871 | |
| … | … | … | … | … | … | |
| 4986 | 1 | -1.359715 | 0.243699 | 0.929772 | 0.890574 | |
| 4990 | 1 | -0.553133 | -0.483965 | -1.075533 | -1.122871 | |
| 4992 | 1 | 0.791169 | 1.880943 | 0.929772 | 0.890574 | |

```
4993          1          -1.449335           -0.120133  -1.075533  -1.122871
4995          1          -0.821994            1.699027  -1.075533   0.890574

      lips_thin  distance_nose_to_lip_long  gender
0      0.925731                   0.913130       1
1      0.925731                  -1.095135       0
2      0.925731                   0.913130       1
3      0.925731                   0.913130       1
4     -1.080227                  -1.095135       0
…           …                          …       …
4986  -1.080227                   0.913130       1
4990   0.925731                  -1.095135       0
4992   0.925731                   0.913130       1
4993  -1.080227                   0.913130       0
4995   0.925731                   0.913130       1

[3233 rows x 8 columns]
```

[ ]:
```
from sklearn.model_selection import train_test_split
```

[54]:
```
# Extracting features and target variable
x = df.iloc[:, 0:7].values  # Features
y = df.iloc[:, -1].values   # Target variable (last column)

# Splitting the data into training and testing sets
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)

# Initializing and training the Logistic Regression model
from sklearn.linear_model import LogisticRegression
logmodel_ini = LogisticRegression()
logmodel_ini.fit(x_train, y_train)
LogisticRegression()
```

[54]:
```
LogisticRegression()
```

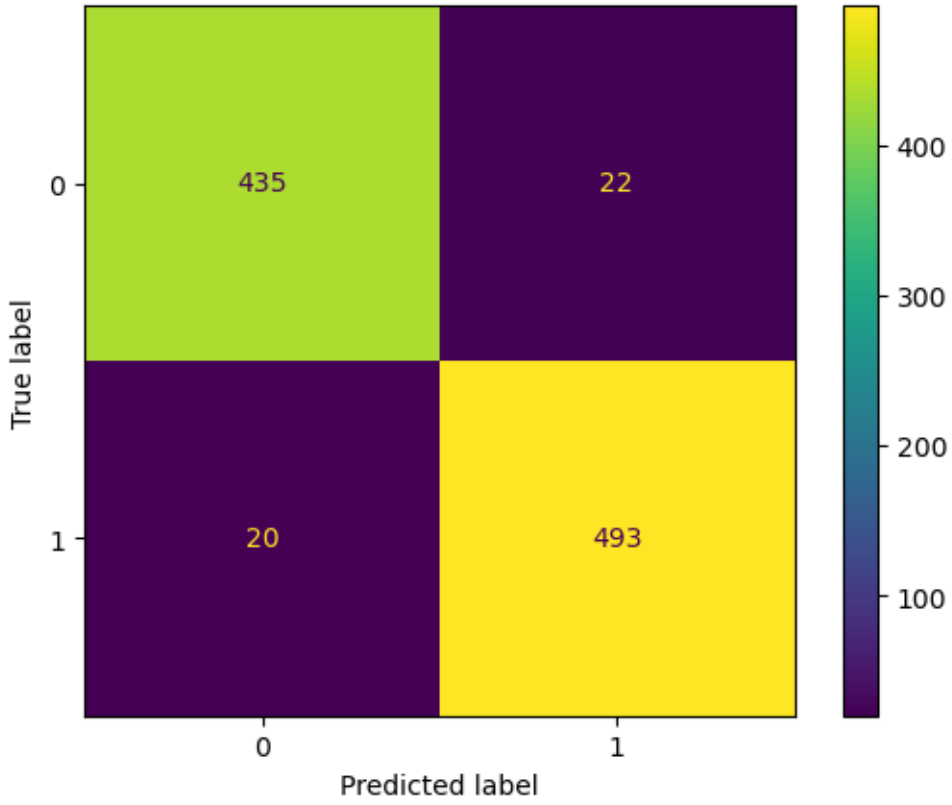
[57]:
```
y_predict = logmodel_ini.predict(x_test)
```

[62]:
```
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score, log_loss, roc_curve, auc
from sklearn.metrics import RocCurveDisplay, PrecisionRecallDisplay,␣
 ↪ConfusionMatrixDisplay
```

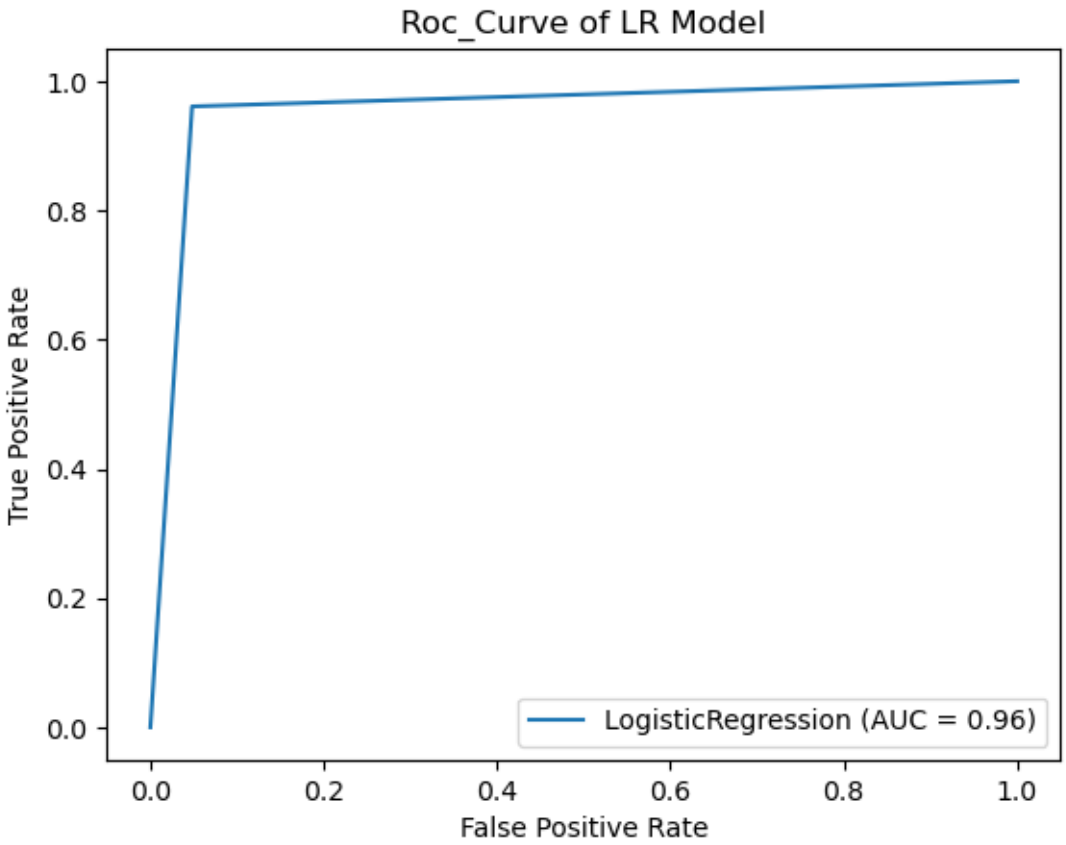
[63]:
```
accuracy_score(y_predict, y_test)
```

[63]:
```
0.9567010309278351
```

[68]:
```
cm = confusion_matrix(y_test, y_predict,labels=logmodel_ini.classes_)
# Create the ConfusionMatrixDisplay instance
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=logmodel_ini.
 ↪classes_)
# Plot the confusion matrix
disp.plot()
plt.show()
```




[70]:
```
# Roc Curve
fpr, tpr, threshold = roc_curve(y_test, y_predict)
roc_auc = auc(fpr,tpr)
Display = RocCurveDisplay(fpr=fpr, tpr=tpr, roc_auc=roc_auc, estimator_name =␣
 ↪'LogisticRegression')
Display.plot()
plt.title('Roc_Curve of LR Model')
plt.show()
```




## 2 F1 Score:

target_names = ["Negative(0)", "Positive(1)"] print(classification_report(y_test, y_predict, target_names = target_names))

[75]:
```
# Logg Loss
log_loss(y_test, logmodel_ini.predict(x_test))
```

[75]:
```
1.5606530333432171
```

## 3 Comparing the Training and Testing Accuracies

[78]:
```
y_predict_train = logmodel_ini.predict(x_train)
accuracy_score(y_train, y_predict_train)
```

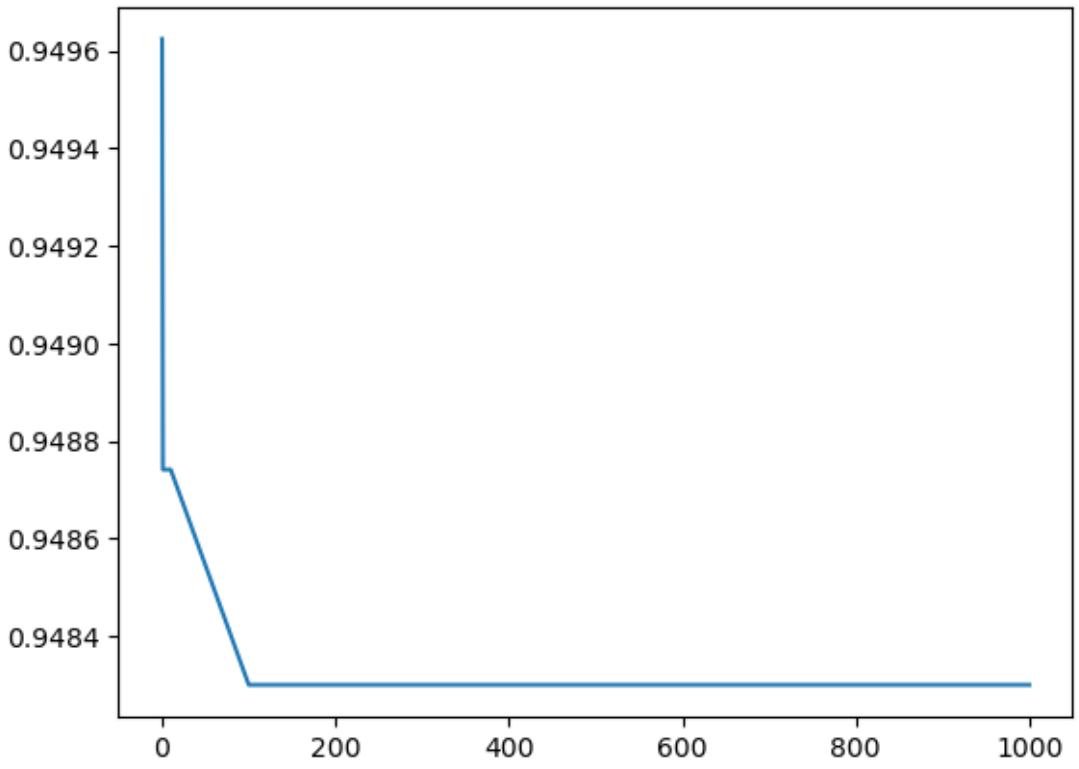
[78]:
```
0.9513919575784358
```

[79]:
```
accuracy_score(y_test, y_predict)
```

[79]: 0.9567010309278351

## 4 Applying K-Fold Cross Validation to find the best value of C (C = 1/Lambda)

[83]:
```
C = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
from sklearn.model_selection import cross_val_score
cv_score = []
for c in C:
    logmodel = LogisticRegression(C=c)
    scores = cross_val_score(logmodel, x_train,y_train,cv=3, scoring='accuracy')
    cv_score.append(scores.mean())
    cv_score
plt.plot(C, cv_score)
plt.show()
```




[86]:
```
#Applying the Logistic Regression on the training dataset
from sklearn.linear_model import LogisticRegression
logmodel_ridge = LogisticRegression(C=0.01,penalty='l2',solver='liblinear')
logmodel_ridge.fit(x_train,y_train)
LogisticRegression(C=0.01, solver='liblinear')
```

```
logmodel_ridge.coef_
```

[86]:
```
array([[0.14737008, 0.30399421, 0.29222935, 0.92015161, 0.84684995,
        0.84575585, 0.82622053]])
```

[87]:
```
#Running the model on the test dataset
y_pred_ridge = logmodel_ridge.predict(x_test)
```

[88]:
```
accuracy_score(y_test,y_pred_ridge)
```

[88]:
```
0.9536082474226805
```

[90]:
```
#Running the model on the test dataset
y_pred_train_ridge = logmodel_ridge.predict(x_train)
accuracy_score(y_train,y_pred_train_ridge)
```

[90]:
```
0.9518338488731772
```

[91]:
```
#Applying the Logistic Regression on the training dataset
from sklearn.linear_model import LogisticRegression
logmodel_lasso = LogisticRegression(C=0.01,penalty='l1',solver='liblinear')
logmodel_lasso.fit(x_train,y_train)
LogisticRegression(C=0.01, penalty='l1', solver='liblinear')
logmodel_lasso.coef_
```

[91]:
```
array([[0.        , 0.        , 0.        , 0.92104439, 0.81677827,
        0.8202117 , 0.80085455]])
```

[93]:
```
#Running the model on the test dataset
y_pred_lasso = logmodel_lasso.predict(x_test)
#Using accuracy score we are checking the accuracy on the testing dataset
accuracy_score(y_test,y_pred_lasso)
```

[93]:
```
0.9329896907216495
```

[94]:
```
y_pred_train_lasso = logmodel_lasso.predict(x_train)
accuracy_score(y_train,y_pred_train_lasso)
```

[94]:
```
0.93194874060981
```

## 5 Comparison of Lasso and Ridge models

[97]:
```
logmodel_lasso.coef_
```

[97]:
```
array([[0.        , 0.        , 0.        , 0.92104439, 0.81677827,
        0.8202117 , 0.80085455]])
```

[98]:
```
logmodel_ridge.coef_
```

[98]:
```
array([[0.14737008, 0.30399421, 0.29222935, 0.92015161, 0.84684995,
        0.84575585, 0.82622053]])
```

From the above coefficients we can say that the first feature i.e. long hair is not a very important variable that is the reason in the lasso logistic regression we can see the coefficient coming as 0, however, in the ridge we are seeing a lower coefficient. We also saw this in the exploratory data analysis that there are almost same number of males and females with long and short hairs.

[101]:
```
from sklearn.linear_model import LogisticRegression
logmodel_elasticnet = LogisticRegression(C=0.
 ↪01,penalty='elasticnet',solver='saga',l1_ratio=0.5)
logmodel_elasticnet.fit(x_train, y_train)
LogisticRegression(C=0.01, l1_ratio=0.5, penalty='elasticnet',solver='saga')
logmodel_elasticnet.coef_
```

[101]:
```
array([[0.        , 0.1757663 , 0.15911045, 0.91959258, 0.82995557,
        0.8319247 , 0.80924938]])
```

[103]:
```
y_pred_elasticnet = logmodel_elasticnet.predict(x_test)
accuracy_score(y_test,y_pred_elasticnet)
```

[103]:
```
0.9494845360824742
```

[104]:
```
y_pred_train_elasticnet = logmodel_elasticnet.predict(x_train)
accuracy_score(y_train,y_pred_train_elasticnet)
```

[104]:
```
0.9496243923994697
```

[133]:
```
from sklearn.metrics import roc_auc_score, precision_score, recall_score,␣
 ↪roc_auc_score, f1_score
```

[146]:
```
score_card = pd.DataFrame(columns=['model_name','Accuracy Score','Precision␣
 ↪Score','Recall Score','AUC Score','f1 Score'])
def update_score_card(y_test,y_predict,model_name):
    global score_card
    score_card = pd.concat([score_card, pd.DataFrame([{'model_name':model_name,␣
 ↪'Accuracy Score':accuracy_score(y_test, y_predict),'Precision Score':
 ↪precision_score(y_test, y_predict), 'Recall Score':recall_score(y_test,␣
 ↪y_predict),'AUC Score': roc_auc_score(y_test,y_predict), 'f1 Score':␣
 ↪f1_score(y_test,y_predict)}])], ignore_index = True)
update_score_card(y_test,y_predict,'initial_model')
update_score_card(y_test,y_pred_ridge,'Ridge Regression - L2 Reg')
update_score_card(y_test,y_pred_lasso,'Lasso Regression - L1 Reg')
update_score_card(y_test,y_pred_elasticnet,'Elastic Net - L1 and L2')
score_card
```

[146]:

| | model_name | Accuracy Score | Precision Score | Recall Score | AUC Score | f1 Score |
|---|---|---|---|---|---|---|
| 0 | initial_model | 0.956701 | 0.957282 | 0.961014 | 0.956437 | 0.959144 |
| 1 | Ridge Regression - L2 Reg | 0.953608 | 0.957031 | 0.955166 | 0.953513 | 0.956098 |
| 2 | Lasso Regression - L1 Reg | 0.932990 | 0.989083 | 0.883041 | 0.936050 | 0.933059 |
| 3 | Elastic Net - L1 and L2 | 0.949485 | 0.939394 | 0.966862 | 0.948420 | 0.952930 |

Interpretation: As per the above table, the initial model and the Elasticnet are giving the same accuracy score, however if the use case is to get the output faster i.e. low latency system we can go with L1 regularization

[ ]: