LogisticR

November 21, 2024

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[3]:
```
df = pd.read_csv('heart.csv')

# Print the number of records with and without heart disease.
print("Number of records in each label are")
print(df['target'].value_counts())

# Print the percentage of each label
print("\nPercentage of records in each label are")
print(df['target'].value_counts() * 100 / df.shape[0], "\n")

# Print the first five rows of Dataframe.
df.head()
```

```
Number of records in each label are
target
1    165
0    138
Name: count, dtype: int64

Percentage of records in each label are
target
1    54.455446
0    45.544554
Name: count, dtype: float64
```

[3]:
```
   Unnamed: 0  age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  \
0           0   63    1   3       145   233    1        0      150      0
1           1   37    1   2       130   250    0        1      187      0
2           2   41    0   1       130   204    0        0      172      0
3           3   56    1   1       120   236    0        1      178      0
4           4   57    0   0       120   354    0        1      163      1
```

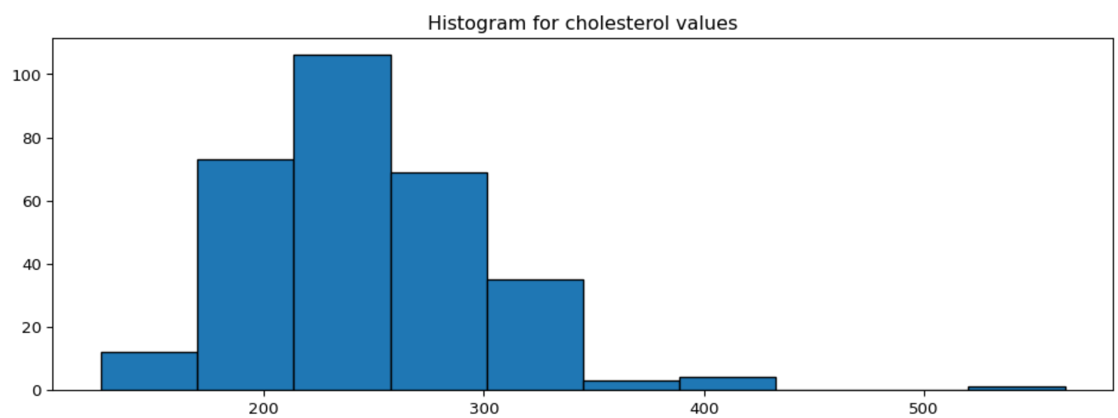
```
   oldpeak  slope  ca  thal  target
0      2.3      0   0     1       1
1      3.5      0   0     2       1
2      1.4      2   0     2       1
3      0.8      2   0     2       1
4      0.6      2   0     2       1
```

[4]:
```
def sigmoid(x):
    return pd.Series(1 / ( 1 + np.exp(-x)))
```

[5]:
```
df['chol'].describe()
```

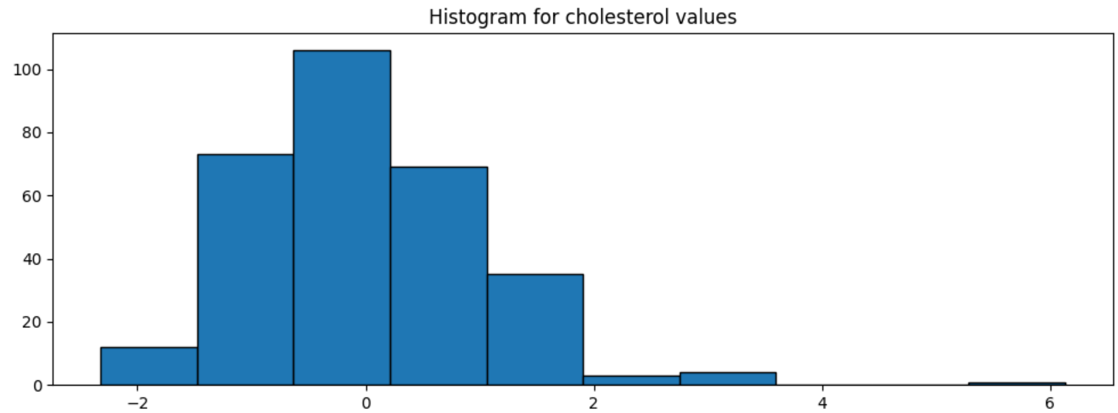
[5]:
```
count    303.000000
mean     246.264026
std       51.830751
min      126.000000
25%      211.000000
50%      240.000000
75%      274.500000
max      564.000000
Name: chol, dtype: float64
```

[6]:
```
plt.figure(figsize = (12,4), dpi = 96)
plt.title("Histogram for cholesterol values")
plt.hist(df['chol'], bins = 'sturges', edgecolor = 'black')
plt. show()
```



[7]:
```
def standard_scalar(series):
    new_series = (series - series.mean()) / series.std()
    return new_series
scaled_chol = standard_scalar(df['chol'])
```

```
plt.figure(figsize = (12,4))
plt.title("Histogram for cholesterol values")
plt.hist(scaled_chol, bins = 'sturges', edgecolor = 'black')
plt.show()
```



[8]:
```
chol_sig_output = sigmoid(df['chol'])
chol_sig_output.describe()
```

[8]:
```
count    303.0
mean       1.0
std        0.0
min        1.0
25%        1.0
50%        1.0
75%        1.0
max        1.0
Name: chol, dtype: float64
```

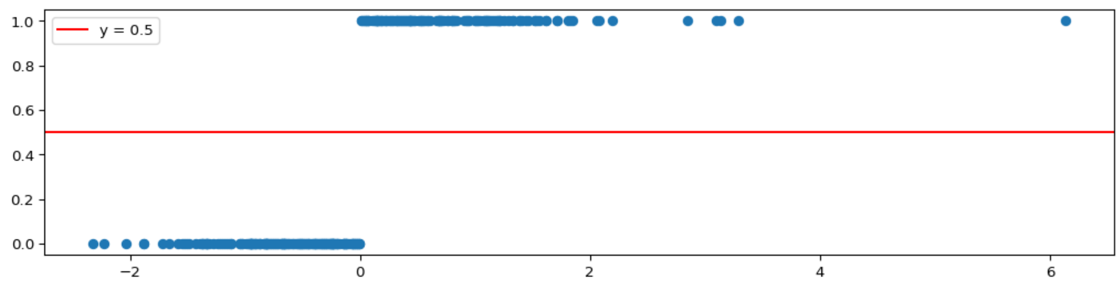
[9]:
```
scaled_chol_sig_output = sigmoid(scaled_chol)
scaled_chol_sig_output.describe()
```

[9]:
```
count    303.000000
mean       0.492837
std        0.198175
min        0.089454
25%        0.336179
50%        0.469823
75%        0.632919
max        0.997829
Name: chol, dtype: float64
```

[10]:
```
def predict(sig_output, threshold):
    y_pred = [ 1 if output >= threshold else 0 for output in sig_output]
    return pd.Series(y_pred)
```

[11]:
```
threshold = 0.5
heart_disease_pred = predict(scaled_chol_sig_output, threshold)

plt.figure(figsize=(13,3), dpi = 96)
plt.scatter(scaled_chol, heart_disease_pred)
plt.axhline(y = threshold, label = f'y = { threshold }', color = 'r')
plt. legend()
plt.show()
```



[12]:
```
print(f"Threshold value: {threshold}")
print(f"\nPredicted value counts:\n{heart_disease_pred.value_counts()}")
print(f"\nActual value counts:\n{df['target']. value_counts()}")
```

```
Threshold value: 0.5

Predicted value counts:
0    167
1    136
Name: count, dtype: int64

Actual value counts:
target
1    165
0    138
Name: count, dtype: int64
```

[13]:
```
from sklearn.metrics import confusion_matrix

print(confusion_matrix(df['target'], heart_disease_pred))
```

```
[[ 65  73]
 [102  63]]
```

[14]:
```python
from sklearn.metrics import classification_report

print(classification_report(df['target'], heart_disease_pred))
```

```
              precision    recall  f1-score   support

           0       0.39      0.47      0.43       138
           1       0.46      0.38      0.42       165

    accuracy                           0.42       303
   macro avg       0.43      0.43      0.42       303
weighted avg       0.43      0.42      0.42       303
```

[15]:
```python
#Split the training and testing data
from sklearn.model_selection import train_test_split

X = df.drop(columns = 'target')
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
↪random_state = 42)
```

[16]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

# Standardizing the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize Logistic Regression with increased max_iter
log_clf_1 = LogisticRegression(max_iter=200)

# Train the model on the scaled data
log_clf_1.fit(X_train_scaled, y_train)

# Print the training accuracy
print(f"Training Accuracy: {log_clf_1.score(X_train_scaled, y_train):.4f}")

# Predict the target values for the train set
y_train_pred = log_clf_1.predict(X_train_scaled)

# Confusion Matrix
print("\nConfusion Matrix:\n")
print(confusion_matrix(y_train, y_train_pred))

# Classification Report
```

```
print("\nClassification Report:\n")
print(classification_report(y_train, y_train_pred))
```

```
Training Accuracy: 0.9953

Confusion Matrix:

[[ 96   1]
 [  0 115]]

Classification Report:

              precision    recall  f1-score   support

           0       1.00      0.99      0.99        97
           1       0.99      1.00      1.00       115

    accuracy                           1.00       212
   macro avg       1.00      0.99      1.00       212
weighted avg       1.00      1.00      1.00       212
```

[17]:

```
#Normalise the train and test data-frames using the standard normalisation␣
↪method.
def standard_scaler(series):
  new_series = (series - series.mean()) / series.std()
  return new_series

norm_X_train = X_train.apply(standard_scaler, axis = 0)
norm_X_test = X_test.apply(standard_scaler, axis = 0)

norm_X_train.describe()
```

[17]:

```
         Unnamed: 0           age           sex            cp      trestbps  \
count  2.120000e+02  2.120000e+02  2.120000e+02  2.120000e+02  2.120000e+02
mean   1.508228e-16  1.864337e-16  1.298751e-16  2.251867e-17  5.697748e-16
std    1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00
min   -1.728945e+00 -2.757098e+00 -1.391141e+00 -9.778484e-01 -2.142798e+00
25%   -8.899408e-01 -7.177485e-01 -1.391141e+00 -9.778484e-01 -6.152369e-01
50%    2.326080e-02  7.080006e-02  7.154438e-01 -1.364440e-02 -2.771338e-02
75%    8.308735e-01  7.233920e-01  7.154438e-01  9.505596e-01  5.598102e-01
max    1.718391e+00  2.463637e+00  7.154438e-01  1.914764e+00  3.614933e+00

               chol           fbs       restecg       thalach         exang  \
count  2.120000e+02  2.120000e+02  2.120000e+02  2.120000e+02  2.120000e+02
mean   1.424437e-16 -5.812960e-17 -1.005485e-16  3.058350e-16  9.216946e-17
std    1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00
```

```
min   -2.129975e+00 -3.811266e-01 -1.029172e+00 -2.731467e+00 -6.855616e-01
25%   -6.649586e-01 -3.811266e-01 -1.029172e+00 -6.547229e-01 -6.855616e-01
50%   -1.338901e-01 -3.811266e-01  8.680843e-01  1.693821e-01 -6.855616e-01
75%    5.162111e-01 -3.811266e-01  8.680843e-01  7.847138e-01  1.451778e+00
max    5.799427e+00  2.611423e+00  2.765341e+00  2.279091e+00  1.451778e+00

            oldpeak         slope            ca          thal
count  2.120000e+02  2.120000e+02  2.120000e+02  2.120000e+02
mean   7.541138e-17  5.865329e-17  7.960090e-17  3.770569e-17
std    1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00
min   -9.289910e-01 -2.305793e+00 -6.746937e-01 -3.912465e+00
25%   -9.289910e-01 -6.763660e-01 -6.746937e-01 -5.475864e-01
50%   -1.961683e-01 -6.763660e-01 -6.746937e-01 -5.475864e-01
75%    5.366543e-01  9.530612e-01  3.770347e-01  1.134853e+00
max    4.200768e+00  9.530612e-01  3.532220e+00  1.134853e+00
```

[18]:
```
norm_X_test.describe()
```

[18]:
```
         Unnamed: 0           age           sex            cp      trestbps  \
count  9.100000e+01  9.100000e+01  9.100000e+01  9.100000e+01  9.100000e+01
mean  -1.249001e-16 -2.147245e-16 -1.390829e-16 -1.952040e-17 -6.868742e-16
std    1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00
min   -1.644808e+00 -2.301763e+00 -1.661622e+00 -8.425578e-01 -1.853721e+00
25%   -8.333169e-01 -8.354271e-01 -1.661622e+00 -8.425578e-01 -6.650121e-01
50%   -6.722410e-02  1.797284e-01  5.952080e-01 -8.425578e-01 -1.662530e-02
75%    9.258592e-01  6.309086e-01  5.952080e-01  1.123410e+00  4.696648e-01
max    1.703301e+00  2.435630e+00  5.952080e-01  2.106394e+00  3.549502e+00

               chol           fbs       restecg       thalach         exang  \
count  9.100000e+01  9.100000e+01  9.100000e+01  9.100000e+01  9.100000e+01
mean  -4.148086e-17  3.538073e-17 -4.880101e-18 -5.294910e-16 -1.049222e-16
std    1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00
min   -2.624853e+00 -4.938276e-01 -9.430373e-01 -3.319275e+00 -7.148350e-01
25%   -7.201088e-01 -4.938276e-01 -9.430373e-01 -6.418709e-01 -7.148350e-01
50%   -1.836075e-02 -4.938276e-01 -9.430373e-01  1.078023e-01 -7.148350e-01
75%    6.165541e-01 -4.938276e-01  9.639937e-01  6.432832e-01  1.383552e+00
max    3.679740e+00  2.002745e+00  2.871025e+00  1.864180e+00  1.383552e+00

            oldpeak         slope            ca          thal
count  9.100000e+01  9.100000e+01  9.100000e+01  9.100000e+01
mean   1.339037e-16 -1.244426e-16 -6.344132e-17  1.848338e-16
std    1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00
min   -8.367971e-01 -2.184053e+00 -8.102615e-01 -3.491486e+00
25%   -8.367971e-01 -5.812398e-01 -8.102615e-01 -4.364358e-01
50%   -3.799059e-01 -5.812398e-01 -8.102615e-01 -4.364358e-01
75%    5.719508e-01  1.021573e+00  9.246514e-01  1.091089e+00
max    3.884412e+00  1.021573e+00  2.659564e+00  1.091089e+00
```

[19]:
```
#Create a dictionary containing the different combination of features selected␣
 ↪by RFE and their corresponding f1-scores.
# Import the libraries
from sklearn.feature_selection import RFE
from sklearn.metrics import f1_score
from sklearn.linear_model import LogisticRegression

# Create the empty dictionary.
dict_rfe = {}

# Create a loop
for i in range(1, len(X_train.columns) + 1):
  lg_clf_2 = LogisticRegression()
  rfe = RFE(lg_clf_2,n_features_to_select=i) # 'i' is the number of features to␣
 ↪be selected by RFE to fit a logistic regression model on norm_X_train and␣
 ↪y_train.
  rfe.fit(norm_X_train, y_train)

  rfe_features = list(norm_X_train.columns[rfe.support_]) # A list of important␣
 ↪features chosen by RFE.
  rfe_X_train = norm_X_train[rfe_features]

  # Build a logistic regression model using the features selected by RFE.
  lg_clf_3 = LogisticRegression()
  lg_clf_3.fit(rfe_X_train, y_train)

  # Predicting 'y' values only for the test set as generally, they are␣
 ↪predicted quite accurately for the train set.
  y_test_pred = lg_clf_3.predict(norm_X_test[rfe_features])

  f1_scores_array = f1_score(y_test, y_test_pred, average = None)
  dict_rfe[i] = {"features": list(rfe_features), "f1_score": f1_scores_array} #␣
 ↪'i' is the number of features to be selected by RFE.
```

[20]:
```
#Print the dictionary created
dict_rfe
```

[20]:
```
{1: {'features': ['Unnamed: 0'], 'f1_score': array([0.98795181, 0.98989899])},
 2: {'features': ['Unnamed: 0', 'oldpeak'],
  'f1_score': array([0.98765432, 0.99009901])},
 3: {'features': ['Unnamed: 0', 'exang', 'oldpeak'],
  'f1_score': array([0.98765432, 0.99009901])},
 4: {'features': ['Unnamed: 0', 'exang', 'oldpeak', 'thal'],
  'f1_score': array([0.97560976, 0.98      ])},
 5: {'features': ['Unnamed: 0', 'restecg', 'exang', 'oldpeak', 'thal'],
  'f1_score': array([0.97560976, 0.98      ])},
 6: {'features': ['Unnamed: 0', 'sex', 'restecg', 'exang', 'oldpeak', 'thal'],
```

```
 'f1_score': array([0.96385542, 0.96969697])},
7: {'features': ['Unnamed: 0',
  'sex',
  'cp',
  'restecg',
  'exang',
  'oldpeak',
  'thal'],
 'f1_score': array([0.97560976, 0.98      ])},
8: {'features': ['Unnamed: 0',
  'sex',
  'cp',
  'restecg',
  'exang',
  'oldpeak',
  'ca',
  'thal'],
 'f1_score': array([0.96385542, 0.96969697])},
9: {'features': ['Unnamed: 0',
  'sex',
  'cp',
  'restecg',
  'exang',
  'oldpeak',
  'slope',
  'ca',
  'thal'],
 'f1_score': array([0.96385542, 0.96969697])},
10: {'features': ['Unnamed: 0',
  'sex',
  'cp',
  'chol',
  'restecg',
  'exang',
  'oldpeak',
  'slope',
  'ca',
  'thal'],
 'f1_score': array([0.96385542, 0.96969697])},
11: {'features': ['Unnamed: 0',
  'sex',
  'cp',
  'trestbps',
  'chol',
  'restecg',
  'exang',
  'oldpeak',
```

```
  'slope',
  'ca',
  'thal'],
 'f1_score': array([0.96385542, 0.96969697])},
12: {'features': ['Unnamed: 0',
  'sex',
  'cp',
  'trestbps',
  'chol',
  'fbs',
  'restecg',
  'exang',
  'oldpeak',
  'slope',
  'ca',
  'thal'],
 'f1_score': array([0.96385542, 0.96969697])},
13: {'features': ['Unnamed: 0',
  'sex',
  'cp',
  'trestbps',
  'chol',
  'fbs',
  'restecg',
  'thalach',
  'exang',
  'oldpeak',
  'slope',
  'ca',
  'thal'],
 'f1_score': array([0.96385542, 0.96969697])},
14: {'features': ['Unnamed: 0',
  'age',
  'sex',
  'cp',
  'trestbps',
  'chol',
  'fbs',
  'restecg',
  'thalach',
  'exang',
  'oldpeak',
  'slope',
  'ca',
  'thal'],
 'f1_score': array([0.96385542, 0.96969697])}}
```

[21]:
```
#Convert the dictionary to the dataframe
pd.options.display.max_colwidth = 100
f1_df = pd.DataFrame.from_dict(dict_rfe, orient = 'index')
f1_df
```

[21]:
```
                features  \
1
[Unnamed: 0]
2
[Unnamed: 0, oldpeak]
3
[Unnamed: 0, exang, oldpeak]
4                                                                     [Unnamed: 0,
exang, oldpeak, thal]
5                                                            [Unnamed: 0, restecg,
exang, oldpeak, thal]
6                                                       [Unnamed: 0, sex, restecg,
exang, oldpeak, thal]
7                                                   [Unnamed: 0, sex, cp, restecg,
exang, oldpeak, thal]
8                                               [Unnamed: 0, sex, cp, restecg,
exang, oldpeak, ca, thal]
9                                        [Unnamed: 0, sex, cp, restecg, exang,
oldpeak, slope, ca, thal]
10                                 [Unnamed: 0, sex, cp, chol, restecg, exang,
oldpeak, slope, ca, thal]
11                       [Unnamed: 0, sex, cp, trestbps, chol, restecg, exang,
oldpeak, slope, ca, thal]
12                  [Unnamed: 0, sex, cp, trestbps, chol, fbs, restecg, exang,
oldpeak, slope, ca, thal]
13         [Unnamed: 0, sex, cp, trestbps, chol, fbs, restecg, thalach, exang,
oldpeak, slope, ca, thal]
14    [Unnamed: 0, age, sex, cp, trestbps, chol, fbs, restecg, thalach, exang,
oldpeak, slope, ca, thal]

                                      f1_score
1      [0.9879518072289156, 0.98989898989899]
2    [0.9876543209876543, 0.9900990099009901]
3    [0.9876543209876543, 0.9900990099009901]
4                   [0.975609756097561, 0.98]
5                   [0.975609756097561, 0.98]
6     [0.963855421686747, 0.9696969696969697]
7                   [0.975609756097561, 0.98]
8     [0.963855421686747, 0.9696969696969697]
9     [0.963855421686747, 0.9696969696969697]
10    [0.963855421686747, 0.9696969696969697]
11    [0.963855421686747, 0.9696969696969697]
```

```
12    [0.963855421686747, 0.9696969696969697]
13    [0.963855421686747, 0.9696969696969697]
14    [0.963855421686747, 0.9696969696969697]
```

[22]:
```
#Logistic Regression with the ideal number of features.
lg_clf_4 = LogisticRegression()
rfe = RFE(lg_clf_4, n_features_to_select = 3)

rfe.fit(norm_X_train, y_train)

rfe_features = norm_X_train.columns[rfe.support_]
print(rfe_features)
final_X_train = norm_X_train[rfe_features]

lg_clf_4 = LogisticRegression()
lg_clf_4.fit(final_X_train, y_train)

y_test_predict = lg_clf_4.predict(norm_X_test[rfe_features])
final_f1_scores_array = f1_score(y_test, y_test_predict, average = None)
print(final_f1_scores_array)
```

```
Index(['Unnamed: 0', 'exang', 'oldpeak'], dtype='object')
[0.98765432 0.99009901]
```