Lab22_Class

November 19, 2020

1 Laboratory 22: Classification, Logistic Regression, and Discrete GOF Metrics

1.1 Full name: Samuel Balest

1.2 R#: 11727464

1.3 HEX: 0xb2f268

1.4 Title of the notebook: Lab 22

1.5 Date: 11/18/2020

[1]:
```
rn = 11727464
hrn = hex(rn)
print(hrn)
```

```
0xb2f268
```

1.6 For the last few sessions we have talked about simple linear regression …

1.6.1 We discussed …

- **The theory and implementation of simple linear regression in Python**
- **OLS and MLE methods for estimation of slope and intercept coefficients**

- **Errors (Noise, Variance, Bias) and their impacts on model's performance**
- **Confidence and prediction intervals**
- **And Multiple Linear Regressions**
- **What if we want to predict a discrete variable?**

  The general idea behind our efforts was to use a set of observed events (samples) to capture the relationship between one or more predictor (AKA input, indipendent) variables and an output (AKA response, dependent) variable. The nature of the dependent variables differentiates ***regression*** and ***classification*** problems.

  Regression problems have continuous and usually unbounded outputs. An example is when you're estimating the salary as a function of experience and education level. Or all the examples we have covered so far!

On the other hand, classification problems have discrete and finite outputs called classes or categories. For example, predicting if an employee is going to be promoted or not (true or false) is a classification problem. There are two main types of classification problems:

- Binary or binomial classification:

exactly two classes to choose between (usually 0 and 1, true and false, or positive and negative)

- Multiclass or multinomial classification:

three or more classes of the outputs to choose from

- **When Do We Need Classification?**

  We can apply classification in many fields of science and technology. For example, text classification algorithms are used to separate legitimate and spam emails, as well as positive and negative comments. Other examples involve medical applications, biological classification, credit scoring, and more.

1.7 Logistic Regression

- **What is logistic regression?** Logistic regression is a fundamental classification technique. It belongs to the group of linear classifiers and is somewhat similar to polynomial and linear regression. Logistic regression is fast and relatively uncomplicated, and it's convenient for users to interpret the results. Although it's essentially a method for binary classification, it can also be applied to multiclass problems.

  Logistic regression is a statistical method for predicting binary classes. The outcome or target variable is dichotomous in nature. Dichotomous means there are only two possible classes. For example, it can be used for cancer detection problems. It computes the probability of an event occurrence. Logistic regression can be considered a special case of linear regression where the target variable is categorical in nature. It uses a log of odds as the dependent variable. Logistic Regression predicts the probability of occurrence of a binary event utilizing a logit function. HOW? Remember the general format of the multiple linear regression model: Where, y is dependent variable and x1, x2 … and Xn are explanatory variables. This was, as you know by now, a linear function. There is another famous function known as the ***Sigmoid Function***, also called ***logistic function***. Here is the equation for the Sigmoid function: This image shows the sigmoid function (or S-shaped curve) of some variable : As you see, The sigmoid function has values very close to either 0 or 1 across most of its domain. It can take any real-valued number and map it into a value between 0 and 1. If the curve goes to positive infinity, y predicted will become 1, and if the curve goes to negative infinity, y predicted will become 0. This fact makes it suitable for application in classification methods since we are dealing with two discrete classes (labels, categories, …). If the output of the sigmoid function is more than 0.5, we can classify the outcome as 1 or YES, and if it is less than 0.5, we can classify it as 0 or NO. This cutoff value (threshold) is not always fixed at 0.5. If we apply the Sigmoid function on linear regression: Notice the difference between linear regression and logistic regression: logistic regression is estimated using Maximum Likelihood Estimation (MLE) approach. Maximizing the likelihood function determines the parameters that are most likely to produce the observed data.

  Let's work on an example in Python!

1.7.1 Example 1: Diagnosing Diabetes

The "diabetes.csv" dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. *Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.* #### The datasets consists of several medical predictor variables and one target variable, Outcome. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

Columns	Info.
Pregnancies	Number of times pregnant
Glucose	Plasma glucose concentration a 2 hours in an oral glucose tolerance test
BloodPressure	Diastolic blood pressure (mm Hg)
SkinThickness	Triceps skin fold thickness (mm)
Insulin	2-Hour serum insulin (mu U/ml)
BMI	Body mass index (weight in kg/(height in m)^2)
Diabetes pedigree	Diabetes pedigree function
Age	Age (years)
Outcome	Class variable (0 or 1) 268 of 768 are 1, the others are 0

Let's see if we can build a logistic regression model to accurately predict whether or not the patients in the dataset have diabetes or not? *Acknowledgements: Smith, J.W., Everhart, J.E., Dickson, W.C., Knowler, W.C., & Johannes, R.S. (1988). Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In Proceedings of the Symposium on Computer Applications and Medical Care (pp. 261–265). IEEE Computer Society Press.*

[2]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import sklearn.metrics as metrics
import seaborn as sns
%matplotlib inline
```

[3]:
```
# Import the dataset:
data = pd.read_csv("diabetes.csv")
data.rename(columns = {'Pregnancies':'pregnant', 'Glucose':
↪'glucose','BloodPressure':'bp','SkinThickness':'skin',
                        'Insulin        ':'Insulin','BMI':
↪'bmi','DiabetesPedigreeFunction':'pedigree','Age':'age',
                                'Outcome':'label'}, inplace = True)
data.head()
```

[3]:

	pregnant	glucose	bp	skin	Insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

[4]:

```python
data.describe()
```

[4]:

	pregnant	glucose	bp	skin	Insulin	bmi \
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000

	pedigree	age	label
count	768.000000	768.000000	768.000000
mean	0.471876	33.240885	0.348958
std	0.331329	11.760232	0.476951
min	0.078000	21.000000	0.000000
25%	0.243750	24.000000	0.000000
50%	0.372500	29.000000	0.000000
75%	0.626250	41.000000	1.000000
max	2.420000	81.000000	1.000000

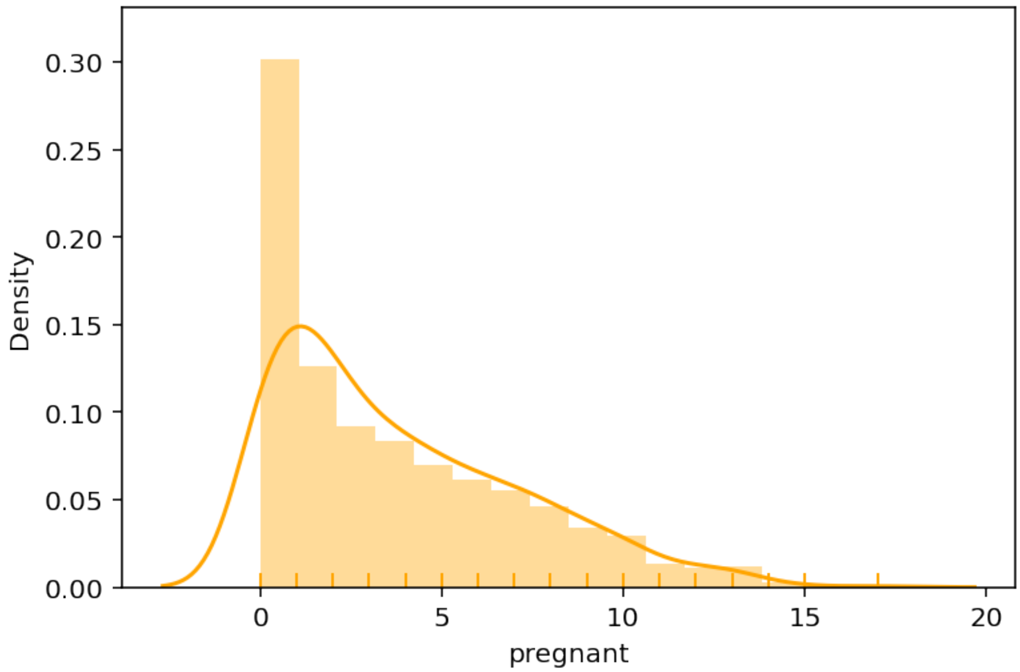
[5]:

```python
#Check some histograms
sns.distplot(data['pregnant'], kde = True, rug= True, color ='orange')
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2551:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2055:
FutureWarning: The `axis` variable is no longer used and will be removed.
Instead, assign variables directly to `x` or `y`.
  warnings.warn(msg, FutureWarning)
```

[5]: <AxesSubplot:xlabel='pregnant', ylabel='Density'>

[5]:

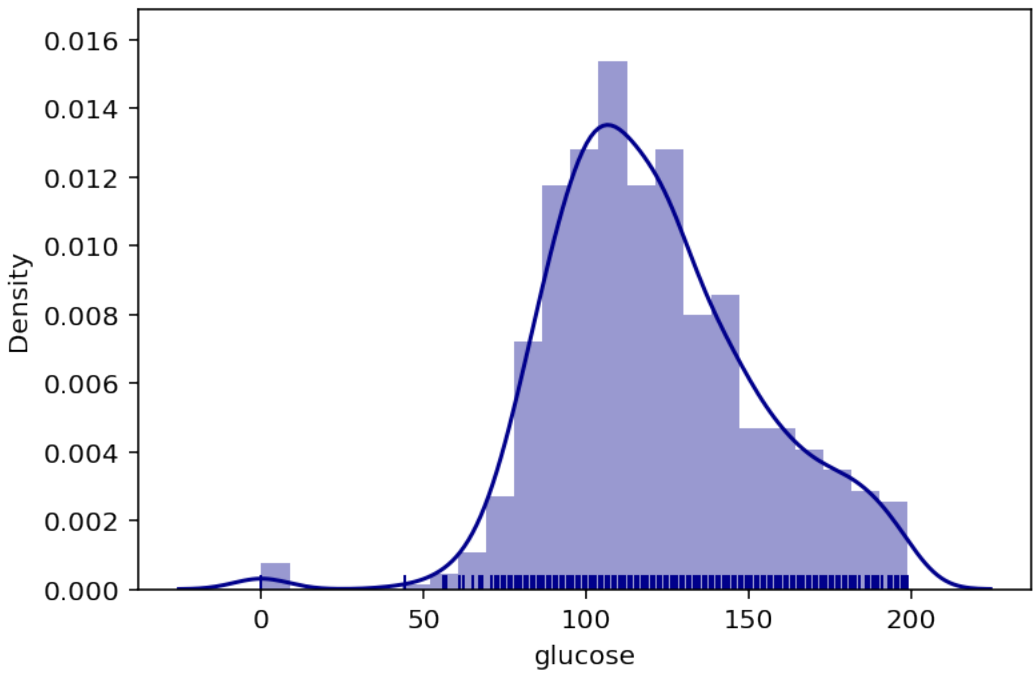


[6]:
```
sns.distplot(data['glucose'], kde = True, rug= True, color ='darkblue')
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2551:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2055:
FutureWarning: The `axis` variable is no longer used and will be removed.
Instead, assign variables directly to `x` or `y`.
  warnings.warn(msg, FutureWarning)
```

[6]:
```
<AxesSubplot:xlabel='glucose', ylabel='Density'>
```

[6]:

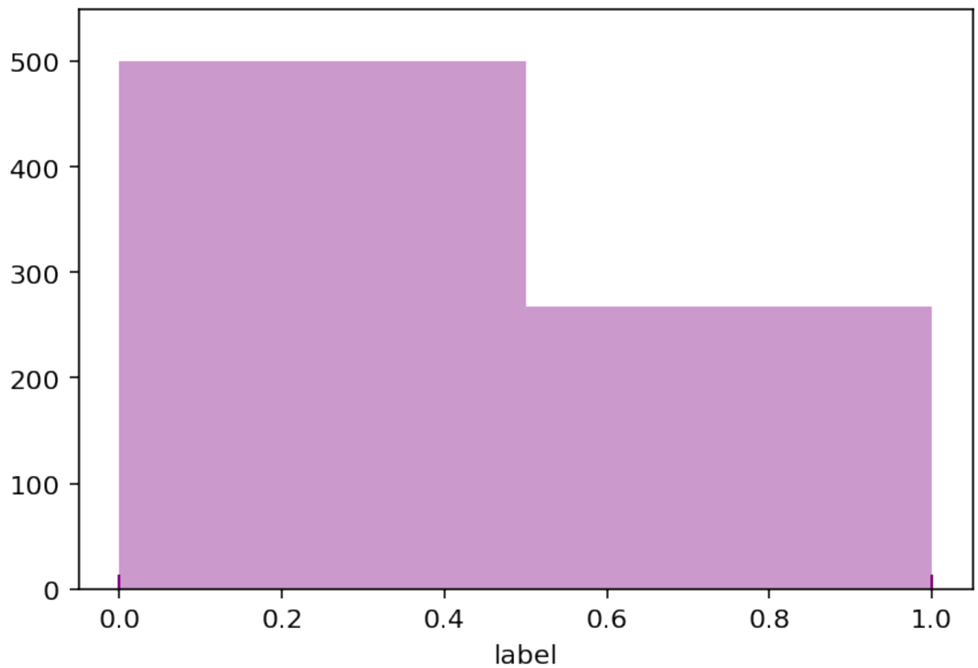


[7]:
```
sns.distplot(data['label'], kde = False, rug= True, color ='purple', bins=2)
```

```
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2551:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2055:
FutureWarning: The `axis` variable is no longer used and will be removed.
Instead, assign variables directly to `x` or `y`.
  warnings.warn(msg, FutureWarning)
```

[7]: <AxesSubplot:xlabel='label'>

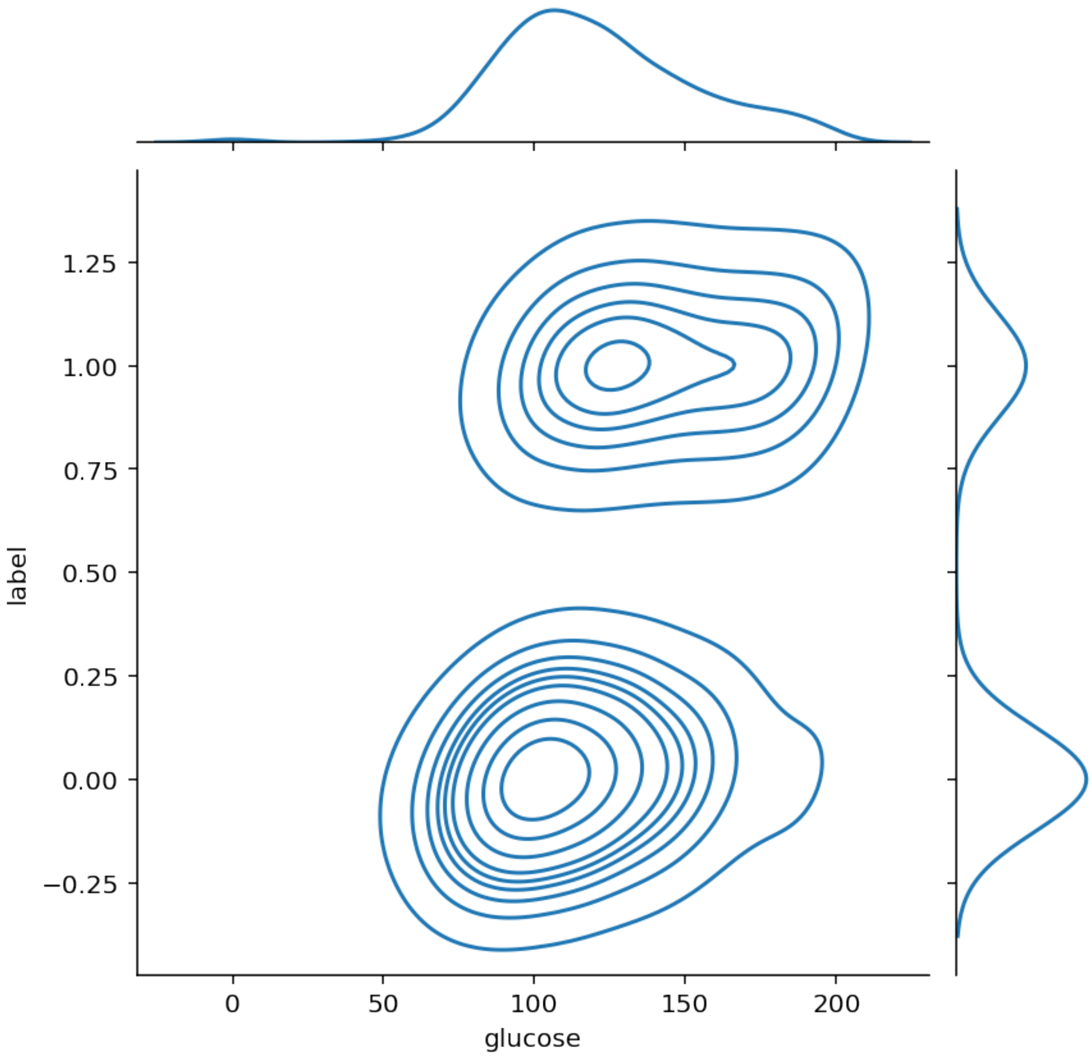
[7]:



[8]:
```
sns.jointplot(x ='glucose', y ='label', data = data, kind ='kde')
```

[8]: <seaborn.axisgrid.JointGrid at 0x7fd3a2202e80>

[8]:



Selecting Feature: Here, we need to divide the given columns into two types of variables dependent(or target variable) and independent variable(or feature variables or predictors).

[9]:
```
#split dataset in features and target variable
feature_cols = ['pregnant', 'glucose', 'bp', 'skin', 'Insulin', 'bmi',␣
 ↪'pedigree', 'age']
X = data[feature_cols] # Features
y = data.label # Target variable
```

Splitting Data: To understand model performance, dividing the dataset into a training set and a test set is a good strategy. Let's split dataset by using function train_test_split(). You need to pass 3 parameters: features, target, and test_set size. Additionally, you can use random_state to select records randomly. Here, the

Dataset is broken into two parts in a ratio of 75:25. It means 75% data will be used for model training and 25% for model testing:

[10]:
```
# split X and y 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.
 ↪25,random_state=0)
```

Model Development and Prediction: First, import the Logistic Regression module and create a Logistic Regression classifier object using LogisticRegression() function. Then, fit your model on the train set using fit() and perform prediction on the test set using predict().

[11]:
```
# import the class
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
logreg.fit(X_train,y_train)

y_pred=logreg.predict(X_test)
```

```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:762:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

- **How to assess the performance of logistic regression?**

  Binary classification has four possible types of results:

  - True negatives: correctly predicted negatives (zeros)
  - True positives: correctly predicted positives (ones)
  - False negatives: incorrectly predicted negatives (zeros)
  - False positives: incorrectly predicted positives (ones)

  We usually evaluate the performance of a classifier by comparing the actual and predicted outputsand counting the correct and incorrect predictions. A confusion matrix is a table that is used to evaluate the performance of a classification model.

  Some indicators of binary classifiers include the following:

  - The most straightforward indicator of classification accuracy is the ratio of the number of correct predictions to the total number of predictions (or observations).
  - The positive predictive value is the ratio of the number of true positives to the sum of the numbers of true and false positives.

- The negative predictive value is the ratio of the number of true negatives to the sum of the numbers of true and false negatives.
- The sensitivity (also known as recall or true positive rate) is the ratio of the number of true positives to the number of actual positives.
- The precision score quantifies the ability of a classifier to not label a negative example as positive. The precision score can be interpreted as the probability that a positive prediction made by the classifier is positive.
- The specificity (or true negative rate) is the ratio of the number of true negatives to the number of actual negatives.

The extent of importance of recall and precision depends on the problem. Achieving a high recall is more important than getting a high precision in cases like when we would like to detect as many heart patients as possible. For some other models, like classifying whether a bank customer is a loan defaulter or not, it is desirable to have a high precision since the bank wouldn't want to lose customers who were denied a loan based on the model's prediction that they would be defaulters. There are also a lot of situations where both precision and recall are equally important. Then we would aim for not only a high recall but a high precision as well. In such cases, we use something called F1-score. F1-score is the Harmonic mean of the Precision and Recall: This is easier to work with since now, instead of balancing precision and recall, we can just aim for a good F1-score and that would be indicative of a good Precision and a good Recall value as well.

Model Evaluation using Confusion Matrix: A confusion matrix is a table that is used to evaluate the performance of a classification model. You can also visualize the performance of an algorithm. The fundamental of a confusion matrix is the number of correct and incorrect predictions are summed up class-wise.

[12]:
```
# import the metrics class
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_pred, y_test)
cnf_matrix
```

[12]:
```
array([[115,  25],
       [ 15,  37]])
```

Here, you can see the confusion matrix in the form of the array object. The dimension of this matrix is 2*2 because this model is binary classification. You have two classes 0 and 1. Diagonal values represent accurate predictions, while non-diagonal elements are inaccurate predictions. In the output, 119 and 36 are actual predictions, and 26 and 11 are incorrect predictions.

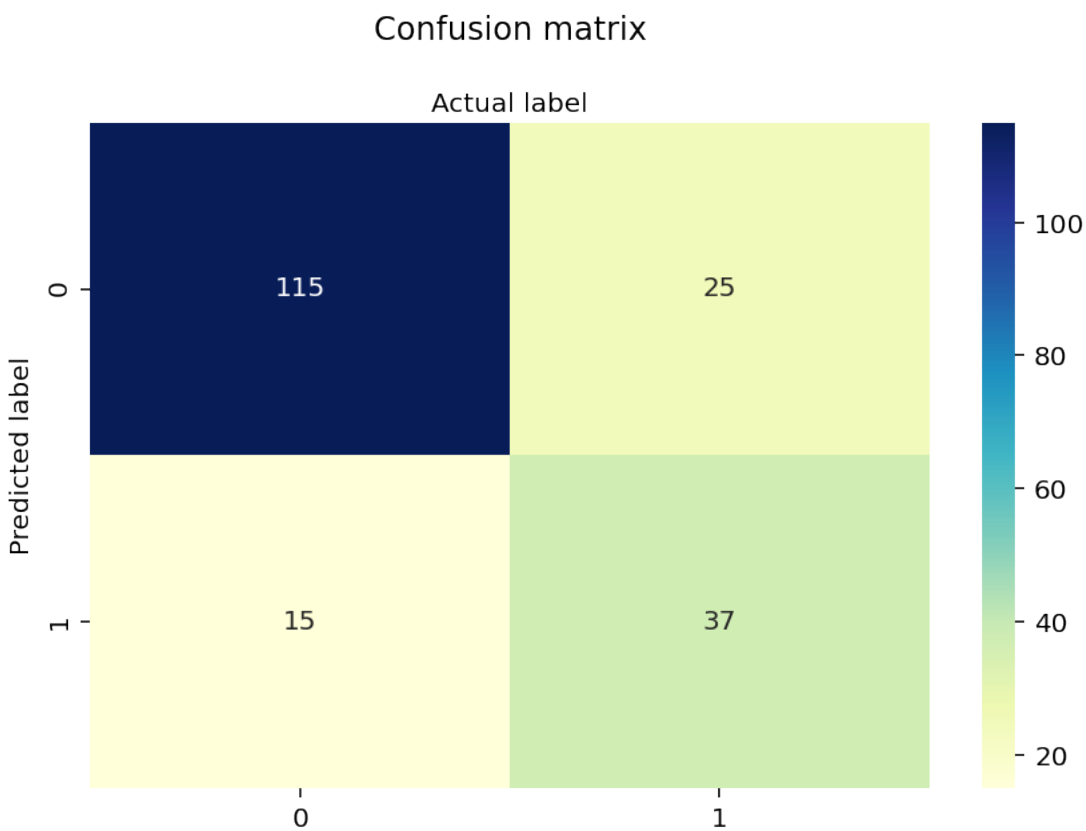
Visualizing Confusion Matrix using Heatmap: Let's visualize the results of the model in the form of a confusion matrix using matplotlib and seaborn.

[13]:
```
class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
```

```
# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Predicted label')
plt.xlabel('Actual label')
```

[13]: Text(0.5, 257.44, 'Actual label')

[13]:



Confusion Matrix Evaluation Metrics: Let's evaluate the model using model evaluation metrics such as accuracy, precision, and recall.

[14]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
print("F1-score:",metrics.f1_score(y_test, y_pred))
```

```
Accuracy: 0.7916666666666666
Precision: 0.7115384615384616
Recall: 0.5967741935483871
```

```
F1-score: 0.6491228070175439
```

[15]:
```python
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.88      0.85       130
           1       0.71      0.60      0.65        62

    accuracy                           0.79       192
   macro avg       0.77      0.74      0.75       192
weighted avg       0.79      0.79      0.79       192
```

This notebook was inspired by several blogposts including:

- **"Logistic Regression in Python"** by **Mirko Stojiljković** available at* https://realpython.com/logistic-regression-python/
- **"Understanding Logistic Regression in Python"** by **Avinash Navlani** available at* https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python
- **"Understanding Logistic Regression with Python: Practical Guide 1"** by **Mayank Tripathi** available at* https://datascience.foundation/sciencewhitepaper/understanding-logistic-regression-with-python-practical-guide-1
- **"Understanding Data Science Classification Metrics in Scikit-Learn in Python"** by **Andrew Long** available at* https://towardsdatascience.com/understanding-data-science-classification-metrics-in-scikit-learn-in-python-3bc336865019

Here are some great reads on these topics: - **"Example of Logistic Regression in Python"** available at* https://datatofish.com/logistic-regression-python/ - **"Building A Logistic Regression in Python, Step by Step"** by **Susan Li** available at* https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8 - **"How To Perform Logistic Regression In Python?"** by **Mohammad Waseem** available at* https://www.edureka.co/blog/logistic-regression-in-python/ - **"Logistic Regression in Python Using Scikit-learn"** by **Dhiraj K** available at* https://heartbeat.fritz.ai/logistic-regression-in-python-using-scikit-learn-d34e882eebb1 - **"ML | Logistic Regression using Python"** available at* https://www.geeksforgeeks.org/ml-logistic-regression-using-python/

Here are some great videos on these topics: - **"StatQuest: Logistic Regression"** by **StatQuest with Josh Starmer** available at* https://www.youtube.com/watch?v=yIYKR4sgzI8&list=PLblh5JKOoLUKxzEP5HA2d-Li7IJkHfXSe - **"Linear Regression vs Logistic Regression | Data Science Training | Edureka"** by **edureka!** available at* https://www.youtube.com/watch?v=OCwZyYH14uw - **"Logistic Regression in Python | Logistic Regression Example | Machine Learning Algorithms | Edureka"** by **edureka!** available at* https://www.youtube.com/watch?v=VCJdg7YBbAQ - **"How to evaluate a classifier in scikit-learn"** by **Data School** available at* https://www.youtube.com/watch?v=85dtiMz9tSo

1.7.2 Exercise 1: Wine Quality

The "winequality.csv" dataset is provided with information related to red vinho verde wine samples, from the north of Portugal. The goal is to model wine quality based on physicochemical tests. Follow the steps and answer the question. *Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.).*

Columns	Info.
fixed acidity	most acids involved with wine or fixed or nonvolatile (do not evaporate readily)
volatile acidity	the amount of acetic acid in wine, which at too high of levels can lead to an unpleasant, vinegar taste
citric acid	found in small quantities, citric acid can add 'freshness' and flavor to wines
residual sugar	the amount of sugar remaining after fermentation stops, it's rare to find wines with less than 1 gram/liter
chlorides	the amount of salt in the wine
free sulfur dioxide	the free form of SO2 exists in equilibrium between molecular SO2 (as a dissolved gas) and bisulfite ion
total sulfur dioxide	amount of free and bound forms of S02; in low concentrations, SO2 is mostly undetectable in wine
density	the density of water is close to that of water depending on the percent alcohol and sugar content
pH	describes how acidic or basic a wine is on a scale from 0 (very acidic) to 14 (very basic); most wines are between 3-4
sulphates	a wine additive which can contribute to sulfur dioxide gas (S02) levels, wich acts as an antimicrobial
alcohol	the percent alcohol content of the wine
quality (score between 0 and 10)	output variable (based on sensory data, score between 0 and 10)

The datasets consists of several Input variables (based on physicochemical tests).

Follow the steps and answer the following questions:

- Step1: Read the "winequality.csv" file as a dataframe. Change the column names to **('acidity_f','acidity_v','ca','rsugar','chlorides','sulfurd_f','sulfurd_t','density','ph','sulphates','alcol**

Explore the dataframe and in a markdown cell breifly describe the different variables in your own words.

- Step2: Use logistic regression and ('acidity_f', 'ca', 'chlorides', 'sulfurd_t', 'ph', 'alcohol') as predictors to predict the quality of wine. Use a 70/30 split for training and testing. Then, get the confusion matrix and use classification_report to describe the performance of your model. Also, get a heatmap and visually assess the predictions of your model. Explain the result of this analysis in a markdown cell.
- Step3: Use logistic regression and ('acidity_v', 'rsugar', 'sulfurd_f', 'density', 'sulphates') as predictors to predict the quality of wine. Use a 70/30 split for training and testing. Then, get the confusion matrix and use classification_report to describe the performance of your model. Also, get a heatmap and visually assess the predictions of your model. Explain the result of this analysis in a markdown cell.
- Step4: Use logistic regression and all the predictors to predict the quality of wine. Use a 70/30 split for training and testing. Then, get the confusion matrix and use classification_report to describe the performance of your model. Also, get a heatmap and visually assess the predictions of your model. Explain the result of this analysis in a markdown cell.
- Step5: Which model provides better results? what are some pros and cons associated with your winning model?

Acknowledgements: P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.

[16]:
```python
#Step1:
df=pd.read_csv('winequality.csv')
pd.read_csv('winequality.csv')
df.rename(columns={"fixed_acidity":"acidity_f","volatile_acidity":
↪"acidity_v","citric_acid":"ca","residual_sugar":"rsugar","chlorides":
↪"chlorides","free_sulfur_dioxide":"sulfurd_f","total_sulfur_dioxide":
↪"sulferd_t","density":"density","pH":"ph","sulphates":"sulphates","alcohol":
↪"alcohol","quality (score _0to10)":"qualityscore"}, inplace=True)

df.head()
```

[16]:
```
   acidity_f  acidity_v    ca  rsugar  chlorides  sulfurd_f  sulferd_t  \
0        7.4       0.70  0.00     1.9      0.076       11.0       34.0
1        7.8       0.88  0.00     2.6      0.098       25.0       67.0
2        7.8       0.76  0.04     2.3      0.092       15.0       54.0
3       11.2       0.28  0.56     1.9      0.075       17.0       60.0
4        7.4       0.70  0.00     1.9      0.076       11.0       34.0

   density    ph  sulphates  alcohol  qualityscore
0   0.9978  3.51       0.56      9.4             5
1   0.9968  3.20       0.68      9.8             5
2   0.9970  3.26       0.65      9.8             5
3   0.9980  3.16       0.58      9.8             6
```

```
4   0.9978  3.51        0.56      9.4             5
```

[17]:
```
df.describe()
```

[17]:
```
         acidity_f   acidity_v           ca       rsugar    chlorides  \
count  1599.000000  1599.000000  1599.000000  1599.000000  1599.000000
mean      8.319637     0.527821     0.270976     2.538806     0.087467
std       1.741096     0.179060     0.194801     1.409928     0.047065
min       4.600000     0.120000     0.000000     0.900000     0.012000
25%       7.100000     0.390000     0.090000     1.900000     0.070000
50%       7.900000     0.520000     0.260000     2.200000     0.079000
75%       9.200000     0.640000     0.420000     2.600000     0.090000
max      15.900000     1.580000     1.000000    15.500000     0.611000

         sulfurd_f   sulferd_t      density           ph    sulphates  \
count  1599.000000  1599.000000  1599.000000  1599.000000  1599.000000
mean     15.874922    46.467792     0.996747     3.311113     0.658149
std      10.460157    32.895324     0.001887     0.154386     0.169507
min       1.000000     6.000000     0.990070     2.740000     0.330000
25%       7.000000    22.000000     0.995600     3.210000     0.550000
50%      14.000000    38.000000     0.996750     3.310000     0.620000
75%      21.000000    62.000000     0.997835     3.400000     0.730000
max      72.000000   289.000000     1.003690     4.010000     2.000000

           alcohol  qualityscore
count  1599.000000   1599.000000
mean     10.422983      5.636023
std       1.065668      0.807569
min       8.400000      3.000000
25%       9.500000      5.000000
50%      10.200000      6.000000
75%      11.100000      6.000000
max      14.900000      8.000000
```

[18]:
```
#Step2:
feature_cols=['acidity_f', 'ca', 'chlorides', 'sulferd_t', 'ph', 'alcohol']
x=df[feature_cols]
y=df.qualityscore

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=0)

from sklearn.linear_model import LogisticRegression
logreg=LogisticRegression()
logreg.fit(x_train,y_train)
y_pred=logreg.predict(x_test)
```

```python
from sklearn import metrics
cnf_matrix=metrics.confusion_matrix(y_pred,y_test)
cnf_matrix
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Predicted label')
plt.xlabel('Actual label')

print(classification_report(y_test, y_pred))
```

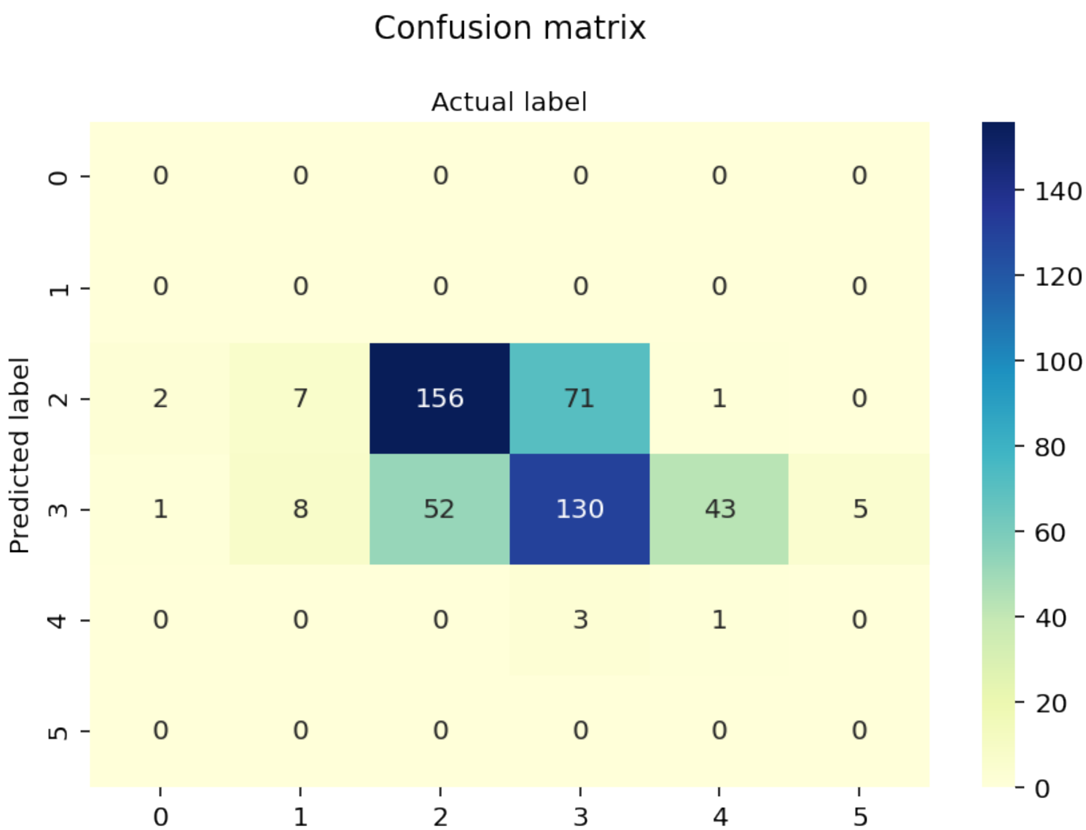
```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:762:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1221:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         3
           4       0.00      0.00      0.00        15
           5       0.66      0.75      0.70       208
           6       0.54      0.64      0.59       204
           7       0.25      0.02      0.04        45
           8       0.00      0.00      0.00         5

    accuracy                           0.60       480
   macro avg       0.24      0.23      0.22       480
weighted avg       0.54      0.60      0.56       480
```

[18]:



[19]:

```
#Step3:
feature_cols=['acidity_v', 'rsugar', 'sulfurd_f', 'density', 'sulphates']
x=df[feature_cols]
y=df.qualityscore

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=0)

from sklearn.linear_model import LogisticRegression
logreg=LogisticRegression()
logreg.fit(x_train,y_train)
y_pred=logreg.predict(x_test)

from sklearn import metrics
cnf_matrix=metrics.confusion_matrix(y_pred,y_test)
cnf_matrix
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
```

```
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Predicted label')
plt.xlabel('Actual label')

print(classification_report(y_test, y_pred))
```

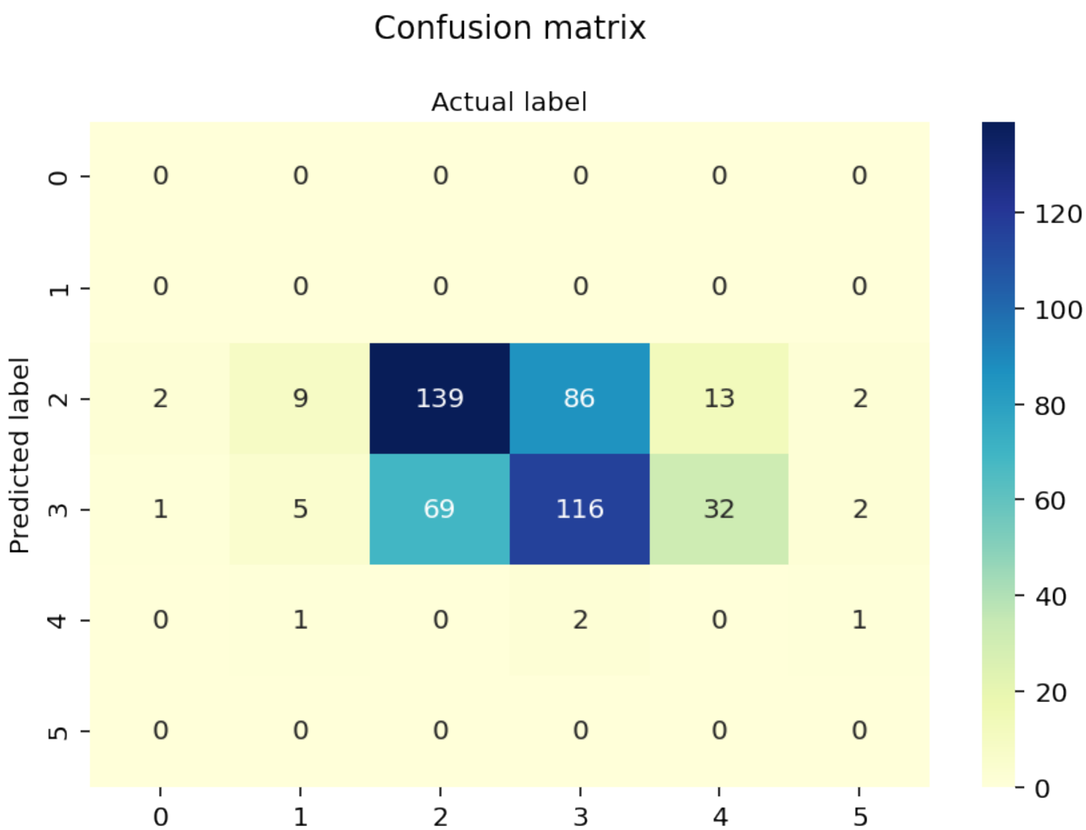
```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:762:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1221:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         3
           4       0.00      0.00      0.00        15
           5       0.55      0.67      0.61       208
           6       0.52      0.57      0.54       204
           7       0.00      0.00      0.00        45
           8       0.00      0.00      0.00         5

    accuracy                           0.53       480
   macro avg       0.18      0.21      0.19       480
weighted avg       0.46      0.53      0.49       480
```

[19]:



[20]:

```
#Step4:
feature_cols=['acidity_f','acidity_v','ca','rsugar','chlorides','sulfurd_f','sulferd_t','densi
x=df[feature_cols]
y=df.qualityscore

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=0)

from sklearn.linear_model import LogisticRegression
logreg=LogisticRegression()
logreg.fit(x_train,y_train)
y_pred=logreg.predict(x_test)

from sklearn import metrics
cnf_matrix=metrics.confusion_matrix(y_pred,y_test)
cnf_matrix
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
```

```
plt.yticks(tick_marks, class_names)
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Predicted label')
plt.xlabel('Actual label')

print(classification_report(y_test, y_pred))
```

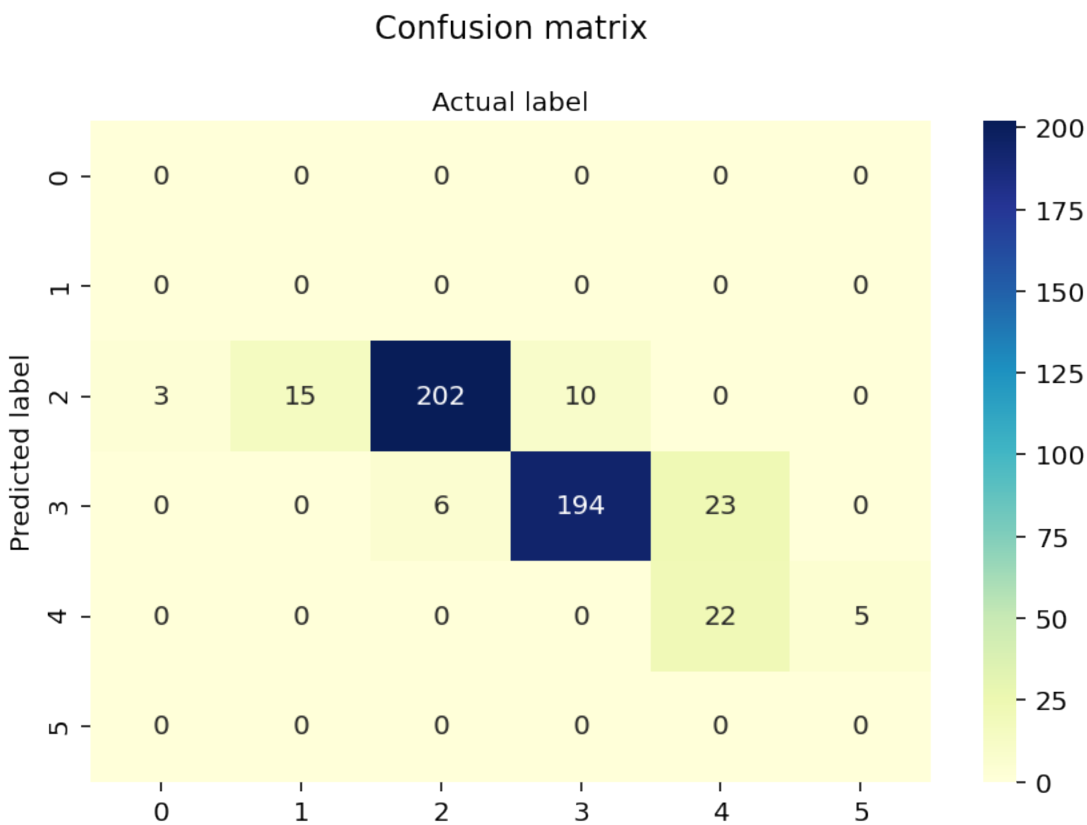
```
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logistic.py:762:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1221:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         3
           4       0.00      0.00      0.00        15
           5       0.88      0.97      0.92       208
           6       0.87      0.95      0.91       204
           7       0.81      0.49      0.61        45
           8       0.00      0.00      0.00         5

    accuracy                           0.87       480
   macro avg       0.43      0.40      0.41       480
weighted avg       0.83      0.87      0.84       480
```

[20]:



#Step5: I think the model used in step4 provided the best results because it had the highest accuracy value.