CSE_445_Project_Human_Activity_Recognition

June 11, 2021

[1]:
```python
print('hello world')
```

```
hello world
```

[2]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[3]:
```python
import numpy as np
import pandas as pd

import os
```

[4]:
```python
data = pd.read_csv(r"/content/drive/MyDrive/Datasets/Human Activity Recognition␣
↪UCI dataset/train.csv")
```

[5]:
```python
data.head()
```

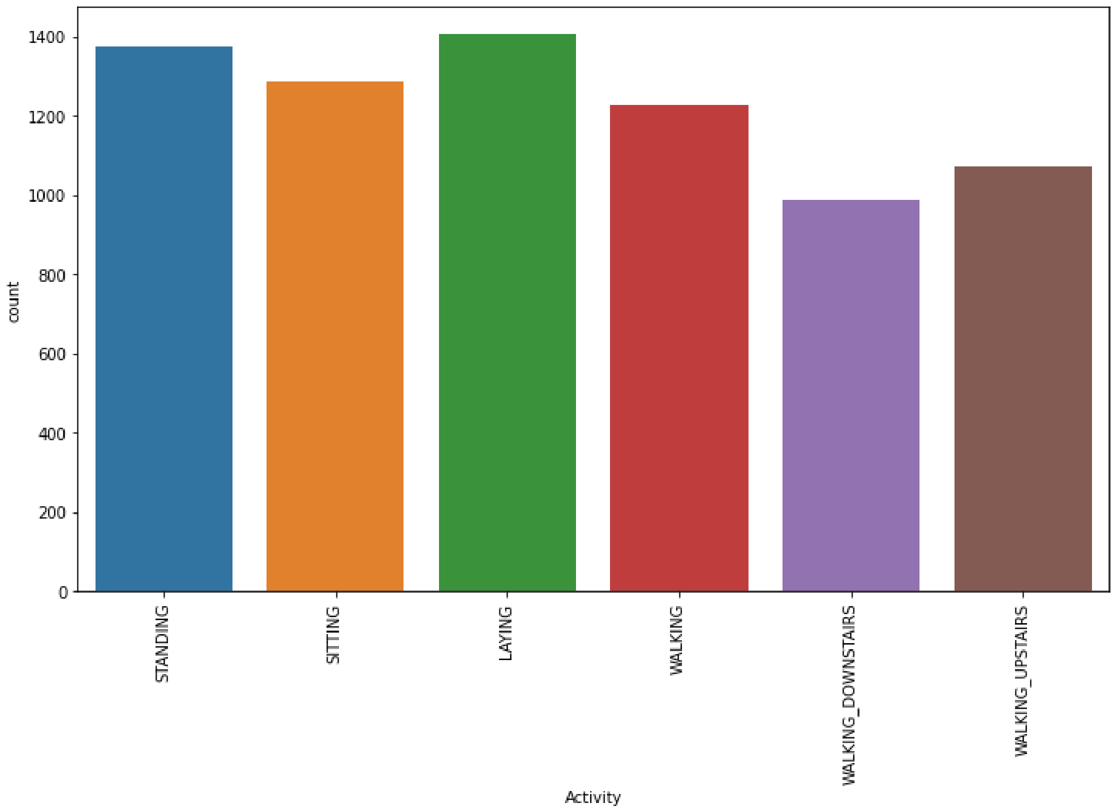
[5]:
```
   tBodyAcc-mean()-X  tBodyAcc-mean()-Y  ...  subject  Activity
0           0.288585          -0.020294  ...        1  STANDING
1           0.278419          -0.016411  ...        1  STANDING
2           0.279653          -0.019467  ...        1  STANDING
3           0.279174          -0.026201  ...        1  STANDING
4           0.276629          -0.016570  ...        1  STANDING

[5 rows x 563 columns]
```

[103]:
```python
data.shape
```

[103]:
```
(7352, 563)
```

1 Visualization of the Dataset

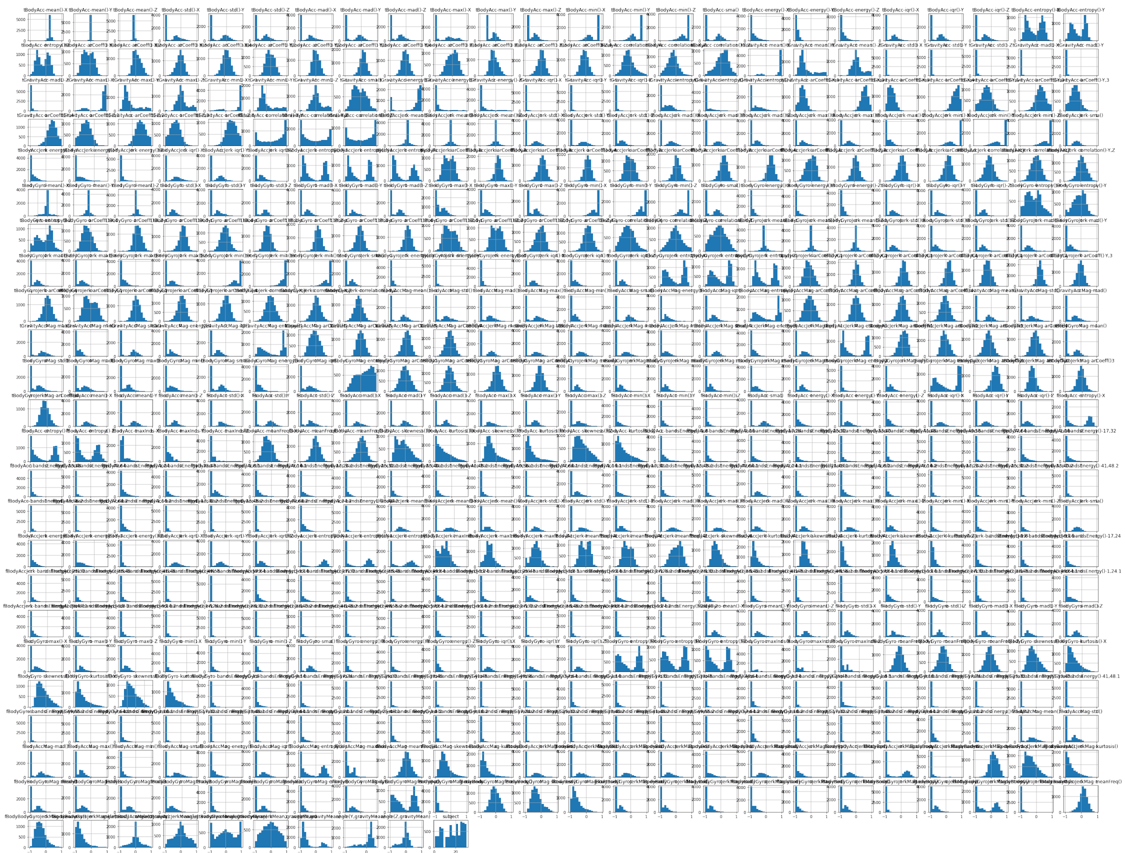
[6]:
```
# count = data['Activity'].value_counts()
# count.plot.bar()
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12,7))

ax = sns.countplot(x = "Activity", data = data)
plt.xticks(x = data["Activity"], rotation = 'vertical')
plt.show()
```



[7]:
```
# data2 = data.copy()
# x = data2.iloc[:-1]
# activity_count = np.array(x.value_counts())
# activity=sorted(x.unique())



# plt.figure(figsize=(10,10))
```

```python
# plt.pie(activity_count,labels=activity,autopct = '%0.2f');
```

[8]:
```python
data.hist(figsize=(50,40),bins = 15)
plt.title("Features Distribution")
plt.show()
```

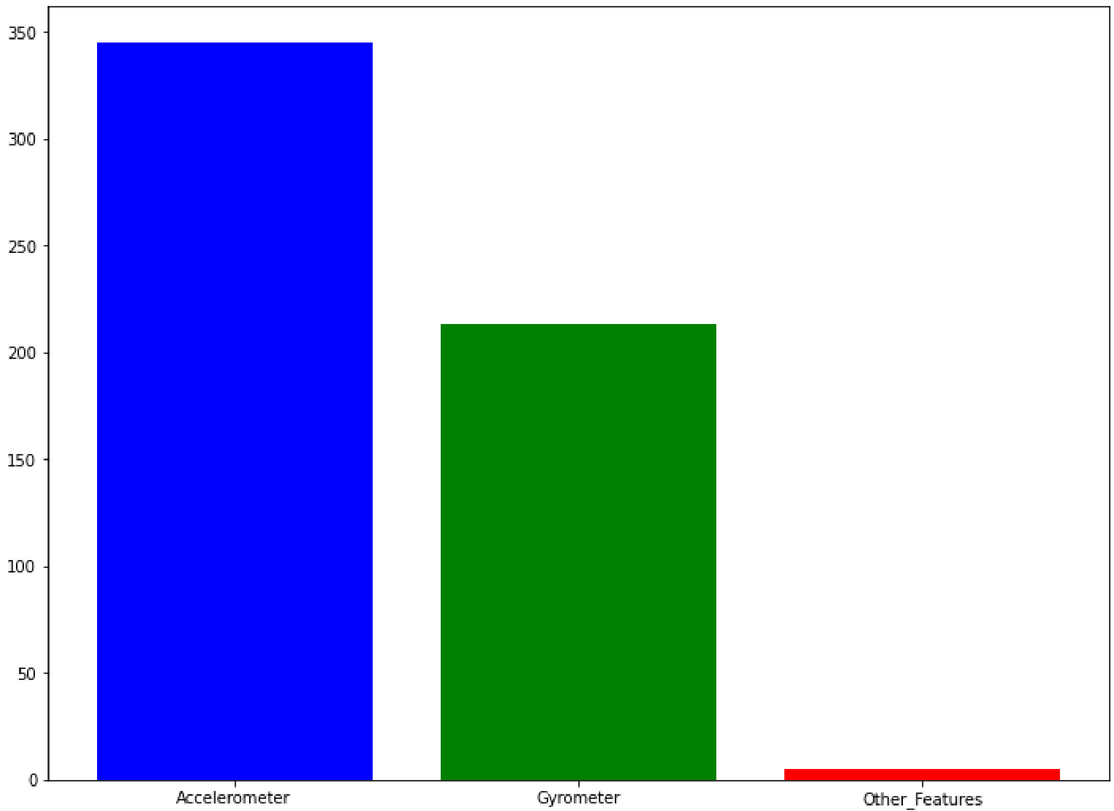


[9]:
```python
accelerometer_counter = 0
gyrometer_counter = 0
other_features = 0
for i in data.columns:
    if "Acc" in str(i):
      accelerometer_counter+=1
    elif "Gyro" in str(i):
      gyrometer_counter+=1
    else:
      other_features+=1
print(accelerometer_counter, gyrometer_counter, other_features)
```

```
345 213 5
```

[10]:
```
plt.figure(figsize = (12,9))

plt.bar(['Accelerometer','Gyrometer','Other_Features'], [accelerometer_counter,␣
↪gyrometer_counter, other_features],color = ['b','g','r'])
```

[10]: <BarContainer object of 3 artists>



[11]:
```
data.shape
```

[11]: (7352, 563)

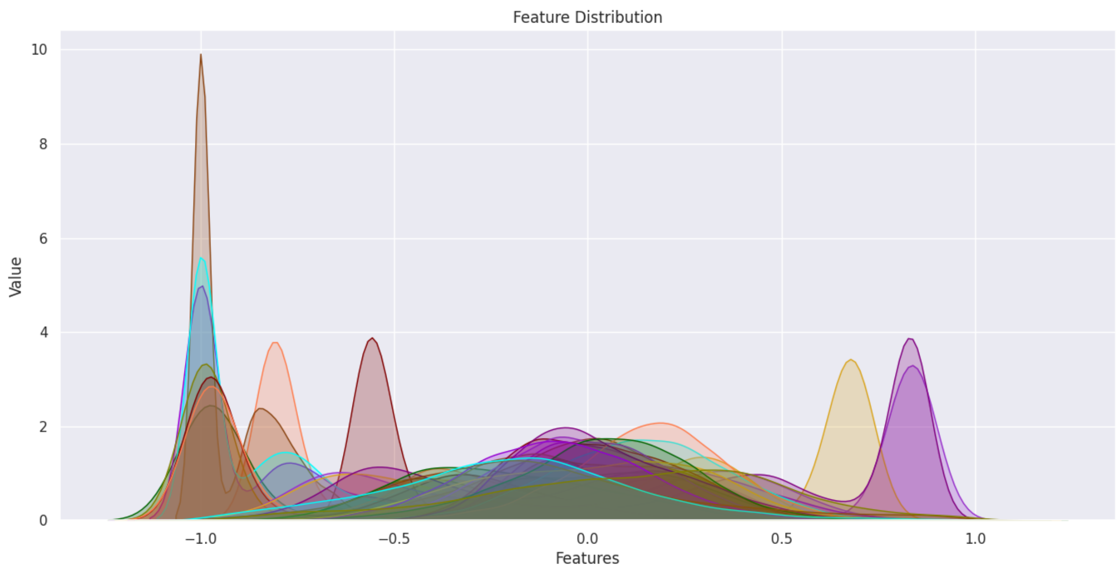
[12]:
```
# plt.figure(figsize=(15,15))
# p=sns.heatmap(data.corr(), annot=True,cmap='RdYlGn',center=0)
```

[13]:
```
data.describe()
```

[13]:
```
         tBodyAcc-mean()-X  tBodyAcc-mean()-Y  ...  angle(Z,gravityMean)
subject
count          7352.000000        7352.000000  ...           7352.000000
7352.000000
mean              0.274488          -0.017695  ...             -0.056515
```

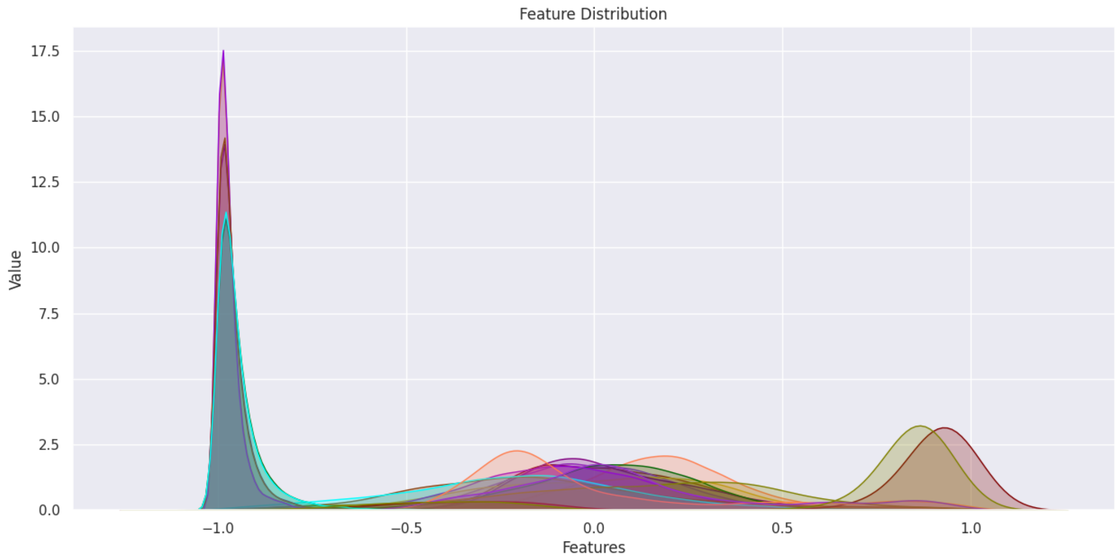
```
17.413085
std              0.070261           0.040811  ...                0.279122
8.975143
min             -1.000000          -1.000000  ...               -1.000000
1.000000
25%              0.262975          -0.024863  ...               -0.143414
8.000000
50%              0.277193          -0.017219  ...                0.003181
19.000000
75%              0.288461          -0.010783  ...                0.107659
26.000000
max              1.000000           1.000000  ...                1.000000
30.000000

[8 rows x 562 columns]
```

[101]:
```
sns.set(rc={'figure.figsize':(15,7)})
colours =␣
 ↪["maroon","coral","darkorchid","goldenrod","purple","darkgreen","darkviolet","saddlebrown","a
colours = colours*2
index = -1
for i in data.columns[10:40]:
    index = index + 1
    ax1 = sns.kdeplot(data[i] , shade=True, color=colours[index])
plt.xlabel("Features")
plt.ylabel("Value")
plt.title("Feature Distribution")
plt.grid(True)
plt.show(fig)
```



[99]:

```
sns.set(rc={'figure.figsize':(15,7)})
colours =␣
 ↪["maroon","coral","darkorchid","goldenrod","purple","darkgreen","darkviolet","saddlebrown","a
index = -1
for i in data.columns[30:50]:
    index = index + 1
    ax1 = sns.kdeplot(data[i] , shade=True, color=colours[index])
plt.xlabel("Features")
plt.ylabel("Value")
plt.title("Feature Distribution")
plt.grid(True)
plt.show(fig)
```



[14]:

```
data = data.dropna()
```

[15]:

```
mapping = {
    'LAYING' : 1,
    'STANDING' : 2,
    'SITTING' : 3,
    'WALKING': 4,
    'WALKING_UPSTAIRS':5,
    'WALKING_DOWNSTAIRS': 6
}
```

[15]:

```

```

[16]:
```
data2 = data.copy()

data2.drop('Activity',axis=1)
```

[16]:
```
      tBodyAcc-mean()-X  tBodyAcc-mean()-Y  ...  angle(Z,gravityMean)  subject
0              0.288585          -0.020294  ...             -0.058627        1
1              0.278419          -0.016411  ...             -0.054317        1
2              0.279653          -0.019467  ...             -0.049118        1
3              0.279174          -0.026201  ...             -0.047663        1
4              0.276629          -0.016570  ...             -0.043892        1
...                 ...                ...  ...                   ...      ...
7347           0.299665          -0.057193  ...              0.049819       30
7348           0.273853          -0.007749  ...              0.050053       30
7349           0.273387          -0.017011  ...              0.040811       30
7350           0.289654          -0.018843  ...              0.025339       30
7351           0.351503          -0.012423  ...              0.036695       30

[7352 rows x 562 columns]
```

[17]:
```
data.shape
```

[17]:
```
(7352, 563)
```

[18]:
```
training_accuracy_dict = {}
testing_accuracy_dict = {}
```

[18]:
```

```

1.1 Logistic Regression

[19]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
import matplotlib.pyplot as plt
```

[20]:
```
X = data.drop(labels = 'Activity',axis=1)
y = data['Activity'].replace(mapping).values
X_train,X_test, y_train, y_test = train_test_split(X,y,test_size=0.
 ↪3,random_state = 23)
```

[21]:
```
logReg = LogisticRegression()
logReg.fit(X_train,y_train)
y_pred=logReg.predict(X_test)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_logistic.py:940:
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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

[21]:

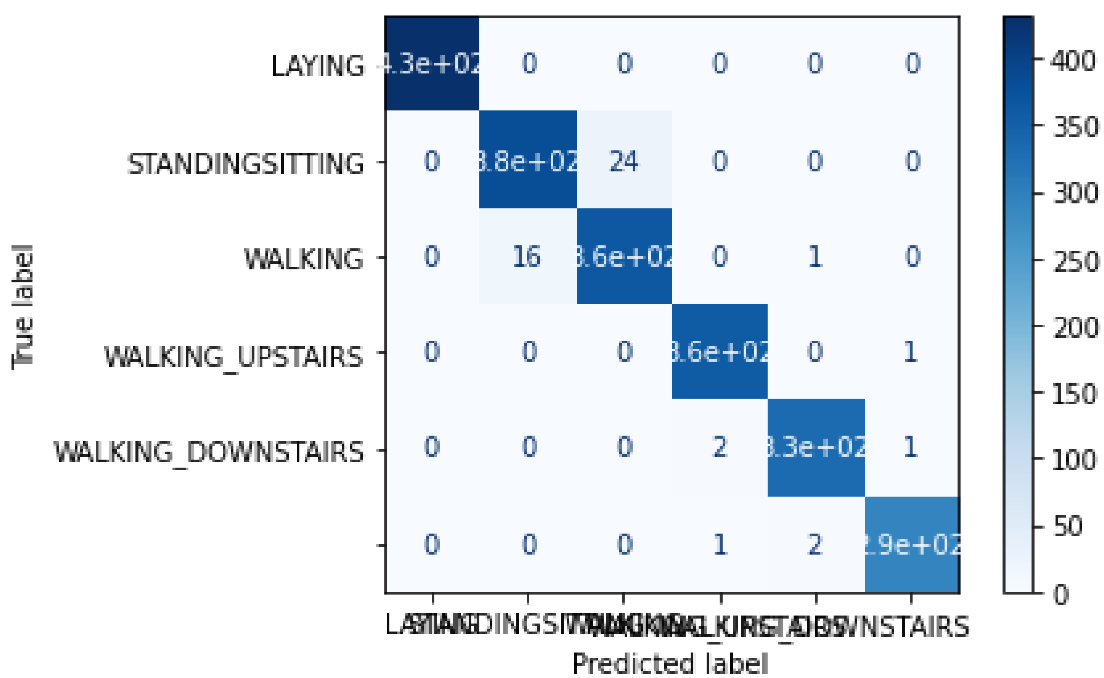
```

```

[22]:

```
cnf_matrix = confusion_matrix(y_test, y_pred)
```

[23]:

```
class_names = ['LAYING','STANDING''SITTING','WALKING','WALKING_UPSTAIRS',␣
↪'WALKING_DOWNSTAIRS']
disp = plot_confusion_matrix(logReg, X_test, y_test,
                                 display_labels=class_names,
                                 cmap=plt.cm.Blues)
```

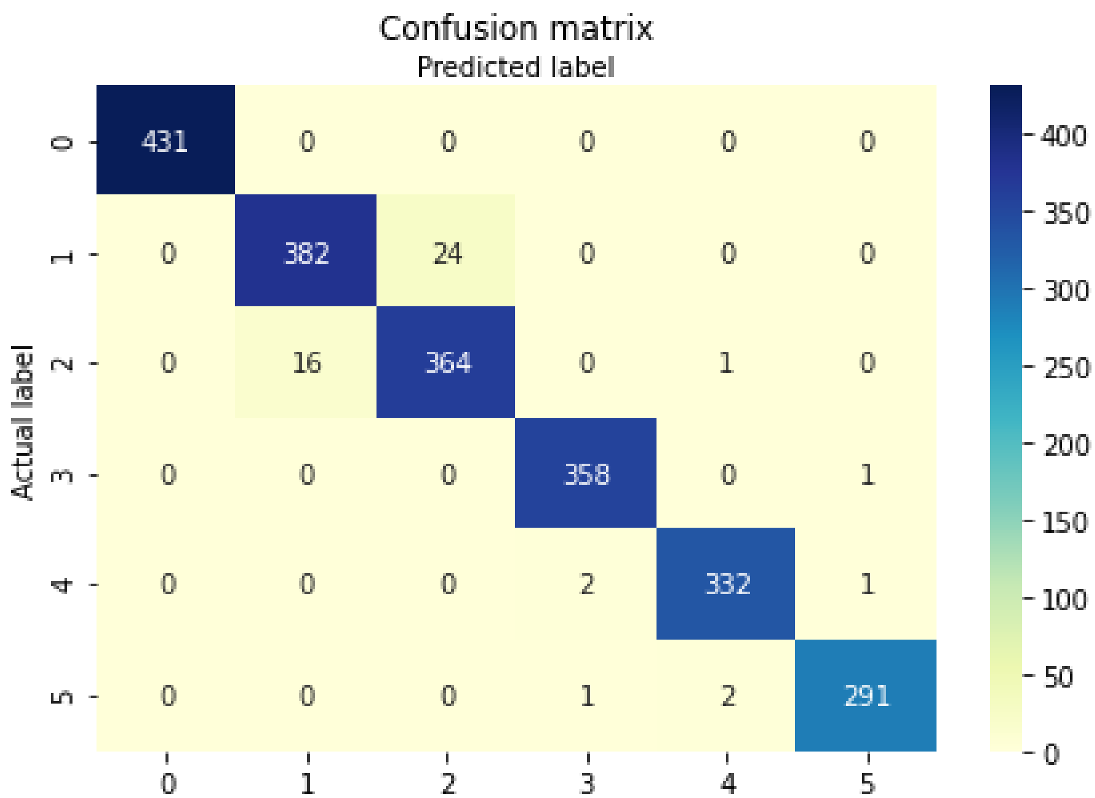


[24]:

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import metrics
%matplotlib inline
```

[25]:
```
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('Actual label')
plt.xlabel('Predicted label')
```

[25]: Text(0.5, 257.44, 'Predicted label')



2 Checking Model's Accuracy on Training Set (Logistic Regression)

[26]:
```
training_accuracy_dict["Logistic_Regression"] = metrics.accuracy_score(y_test,␣
 ↪y_pred)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9782411604714415
```

```
[27]: # X_validation = test_Data.drop(labels = 'Activity',axis=1)
      # y_validation = test_Data['Activity'].replace(mapping).values
      # y_pred_validation=logReg.predict(X_validation)
      # print("Accuracy:",metrics.accuracy_score(y_pred_validation, y_validation))
```

```
[28]: training_accuracy_dict
```

```
[28]: {'Logistic_Regression': 0.9782411604714415}
```

2.1 Checking Accuracy on The Test Set (Logistic Regression)

```
[29]: test_Data =  pd.read_csv(r"/content/drive/MyDrive/Datasets/Human Activity␣
        ↪Recognition UCI dataset/test.csv")

      X_validation = test_Data.drop(labels = 'Activity',axis=1)
      y_validation = test_Data['Activity'].replace(mapping).values
```

```
[30]: y_pred_validation=logReg.predict(X_validation)
```

```
[31]: testing_accuracy_dict["Logistic_Regression"] = metrics.
        ↪accuracy_score(y_pred_validation, y_validation)
      print("Accuracy:",metrics.accuracy_score(y_pred_validation, y_validation))
```

```
Accuracy: 0.9504580929759077
```

```
[32]: testing_accuracy_dict
```

```
[32]: {'Logistic_Regression': 0.9504580929759077}
```

#KNN (K-Nearest Neighbors)

```
[33]: import operator
      import matplotlib.pyplot as plt
```

```
[34]: from sklearn.neighbors import KNeighborsClassifier

      model = KNeighborsClassifier(n_neighbors=3)

      model.fit(X_train,y_train)
```

```
[34]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                           metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                           weights='uniform')
```

```
[35]: y_pred = model.predict(X_test)
```

3 Checking Model's Accuracy on Training Set (KNN)

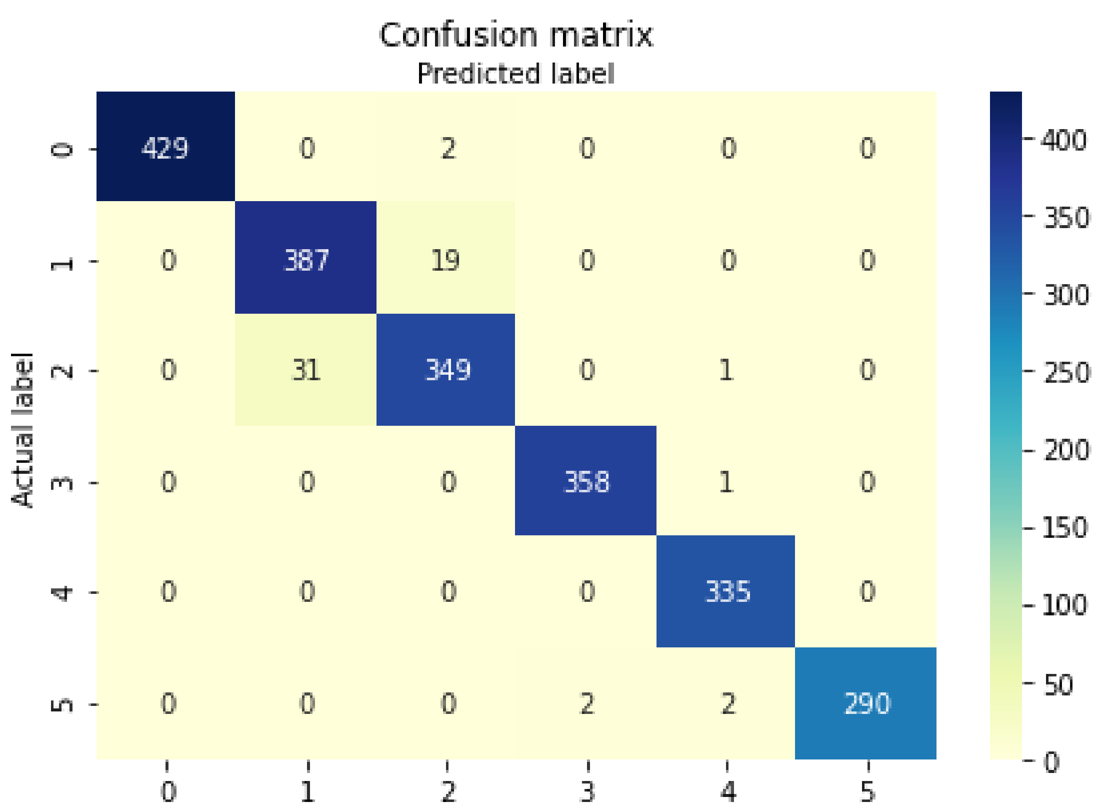
[36]:
```
from sklearn import metrics

training_accuracy_dict["KNN"] = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9737080689029919
```

[37]:
```
cnf_matrix = confusion_matrix(y_test, y_pred)
```

[38]:
```
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('Actual label')
plt.xlabel('Predicted label')
```

[38]:
```
Text(0.5, 257.44, 'Predicted label')
```



4 Checking Model's Accuracy on Test Set (KNN)

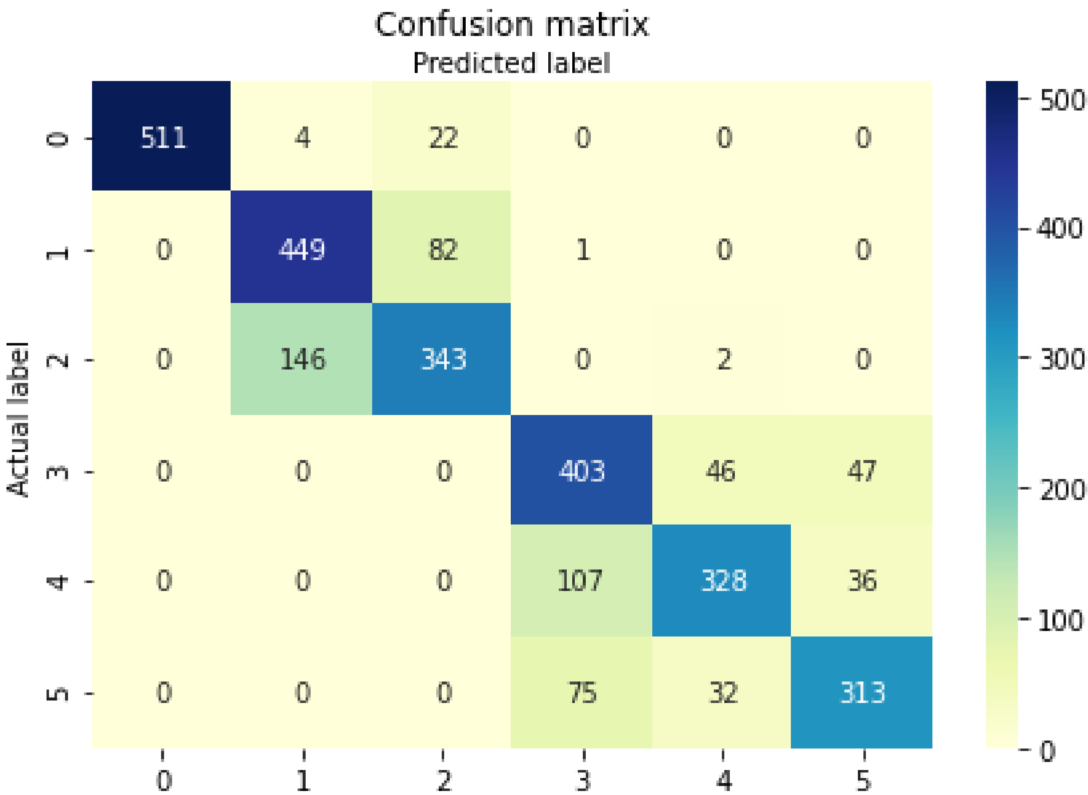
[39]:
```
X_validation = test_Data.drop(labels = 'Activity',axis=1)
y_validation = test_Data['Activity'].replace(mapping).values
y_pred_validation=model.predict(X_validation)

print("Accuracy:",metrics.accuracy_score(y_pred_validation, y_validation))

testing_accuracy_dict["KNN"] = metrics.accuracy_score(y_pred_validation,␣
 ↪y_validation)
```

```
Accuracy: 0.7964031218187988
```

[40]:
```
cnf_matrix = confusion_matrix(y_validation, y_pred_validation)
```

[41]:
```
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('Actual label')
plt.xlabel('Predicted label')
```

[41]:
```
Text(0.5, 257.44, 'Predicted label')
```



[42]:
```
testing_accuracy_dict
```

[42]: {'KNN': 0.7964031218187988, 'Logistic_Regression': 0.9504580929759077}

5 Naive Bayes Classifier

[43]:
```
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier
gnb_classifier = GaussianNB()

#Train the model using the training sets
gnb_classifier.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = gnb_classifier.predict(X_test)
```

6 Checking Model's Accuracy on Training Set (Naive Bayes)

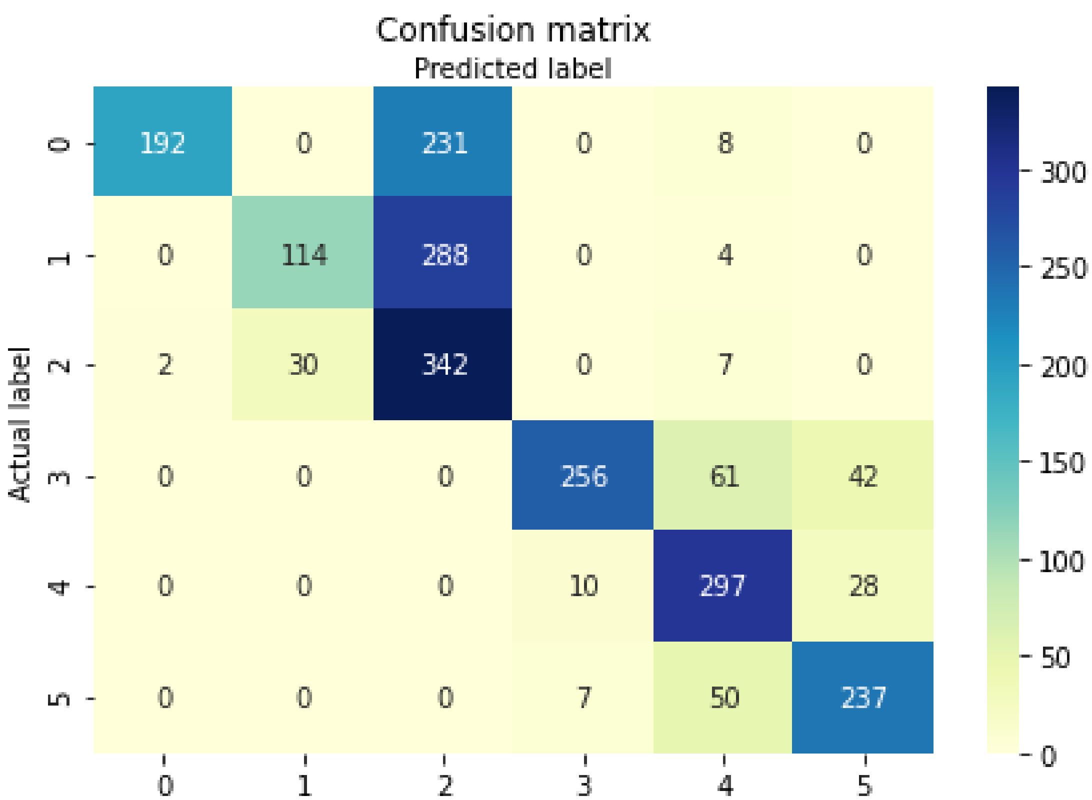
[44]:
```
training_accuracy_dict["Naive_Bayes"] = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.6518585675430644
```

[45]:
```
cnf_matrix = confusion_matrix(y_test, y_pred)
```

[46]:
```
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('Actual label')
plt.xlabel('Predicted label')
```

[46]:
```
Text(0.5, 257.44, 'Predicted label')
```



```
[47]: print("Precision Score : ",metrics.precision_score(y_test, y_pred,
                                           pos_label='positive',
                                           average='micro'))
      print("Recall Score : ",metrics.recall_score(y_test, y_pred,
                                           pos_label='positive',
                                           average='micro'))
```

```
Precision Score :  0.6518585675430644
Recall Score :  0.6518585675430644

/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1321:
UserWarning: Note that pos_label (set to 'positive') is ignored when average !=
'binary' (got 'micro'). You may use labels=[pos_label] to specify a single
positive class.
  % (pos_label, average), UserWarning)
```

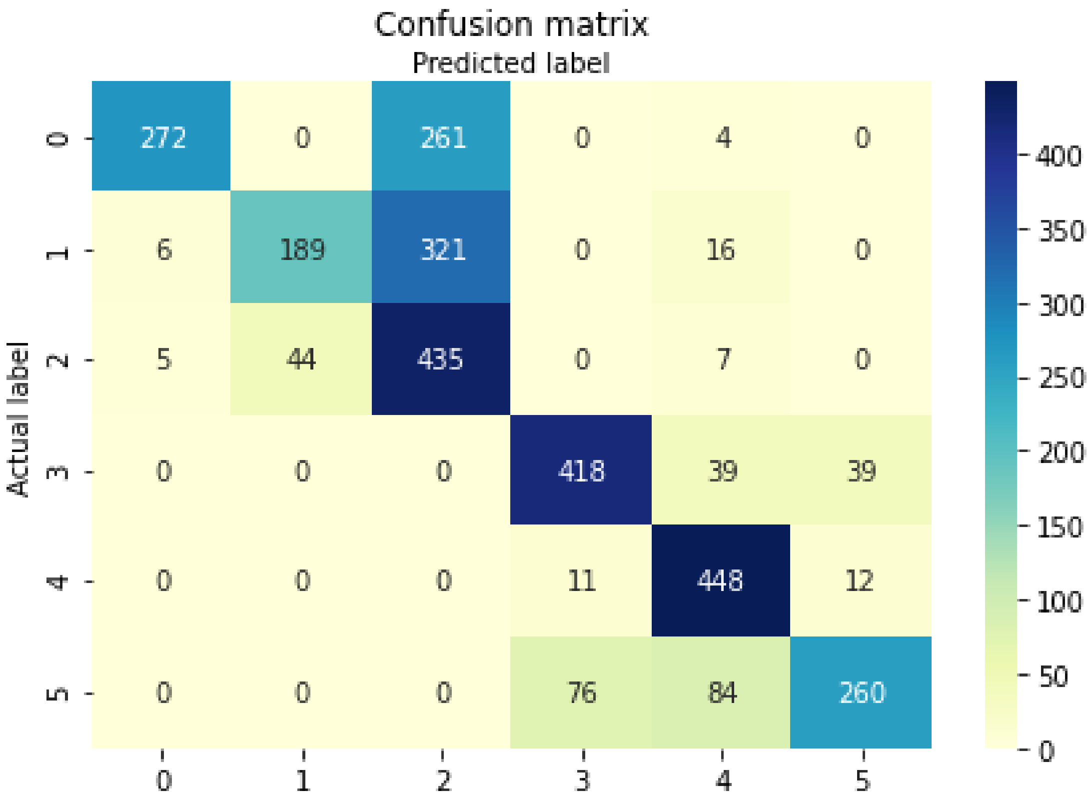
7 Checking Model's Accuracy on Test Set (Naive Bayes)

```
[48]: X_validation = test_Data.drop(labels = 'Activity',axis=1)
      y_validation = test_Data['Activity'].replace(mapping).values
      y_pred_validation=gnb_classifier.predict(X_validation)

      testing_accuracy_dict["Naive_Bayes"] = metrics.accuracy_score(y_pred_validation,␣
       ↪y_validation)


      print("Accuracy:",metrics.accuracy_score(y_pred_validation, y_validation))
```

```
Accuracy: 0.6861214794706482
```

```
[49]: cnf_matrix = confusion_matrix(y_validation, y_pred_validation)

      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('Actual label')
      plt.xlabel('Predicted label')
```

```
[49]: Text(0.5, 257.44, 'Predicted label')
```



[49]:

```

```

8 Decision Tree

[50]:

```
from sklearn.tree import DecisionTreeClassifier # Importing Decision Tree␣
↪Classifier
```

[51]:

```
dtc = DecisionTreeClassifier()
dtc = dtc.fit(X_train,y_train)
y_pred = dtc.predict(X_test)
```

9 Checking Model's Accuracy on Training Set (Decision Tree)

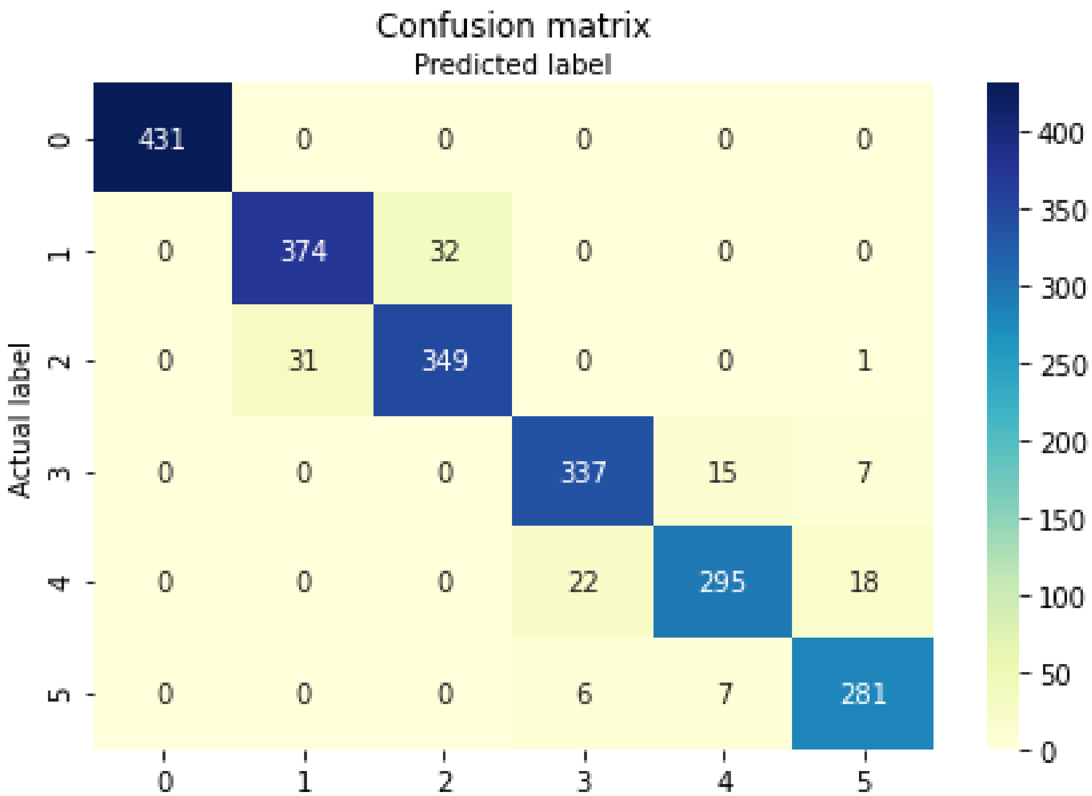
[52]:

```
training_accuracy_dict["Decision_Tree"] = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9369900271985494
```

[53]:
```
cnf_matrix = confusion_matrix(y_test, y_pred)
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('Actual label')
plt.xlabel('Predicted label')
```

[53]:
```
Text(0.5, 257.44, 'Predicted label')
```



Using attribute selection measure "Entropy"

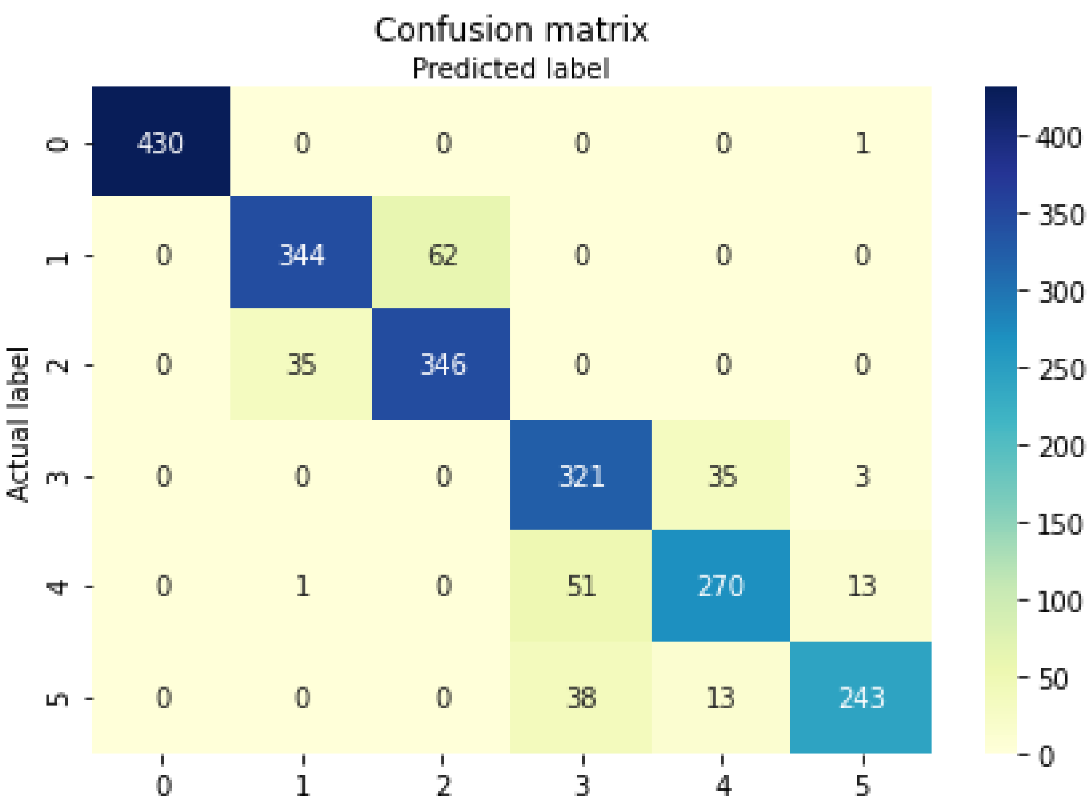
[54]:
```
dtc = DecisionTreeClassifier(criterion="entropy", max_depth=3)

# Train Decision Tree Classifer
dtc = dtc.fit(X_train,y_train)
```

```
#Predict the response for test dataset
y_pred = dtc.predict(X_test)


# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.885766092475068
```

[55]:
```
cnf_matrix = confusion_matrix(y_test, y_pred)
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('Actual label')
plt.xlabel('Predicted label')
```

[55]:
```
Text(0.5, 257.44, 'Predicted label')
```



[55]:

```

```

10 Checking Model's Accuracy on Test Set (Decision Tree)

[56]:

```
X_validation = test_Data.drop(labels = 'Activity',axis=1)
y_validation = test_Data['Activity'].replace(mapping).values
y_pred_validation = dtc.predict(X_validation)

testing_accuracy_dict["Decision_Tree"] = metrics.
 ↪accuracy_score(y_pred_validation, y_validation)


print("Accuracy:",metrics.accuracy_score(y_pred_validation, y_validation))
```

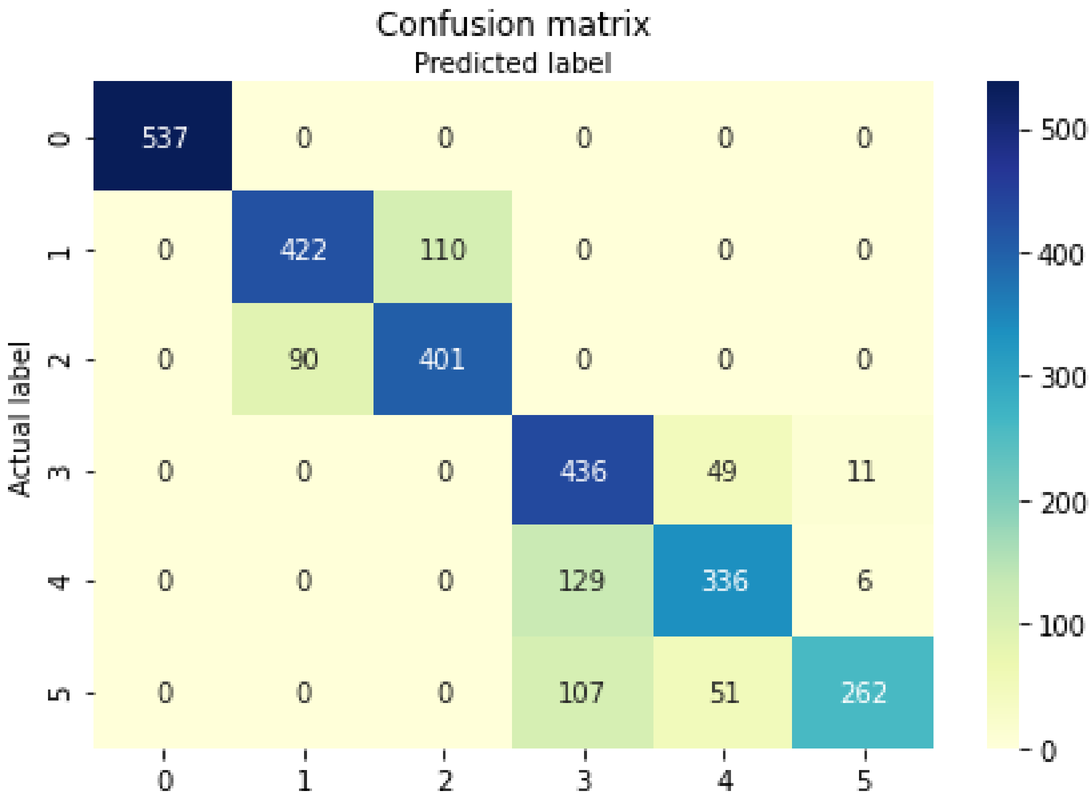
```
Accuracy: 0.8123515439429929
```

[57]:

```
cnf_matrix = confusion_matrix(y_validation, y_pred_validation)

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('Actual label')
plt.xlabel('Predicted label')
```

[57]:

```
Text(0.5, 257.44, 'Predicted label')
```



[57]:

[58]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
```

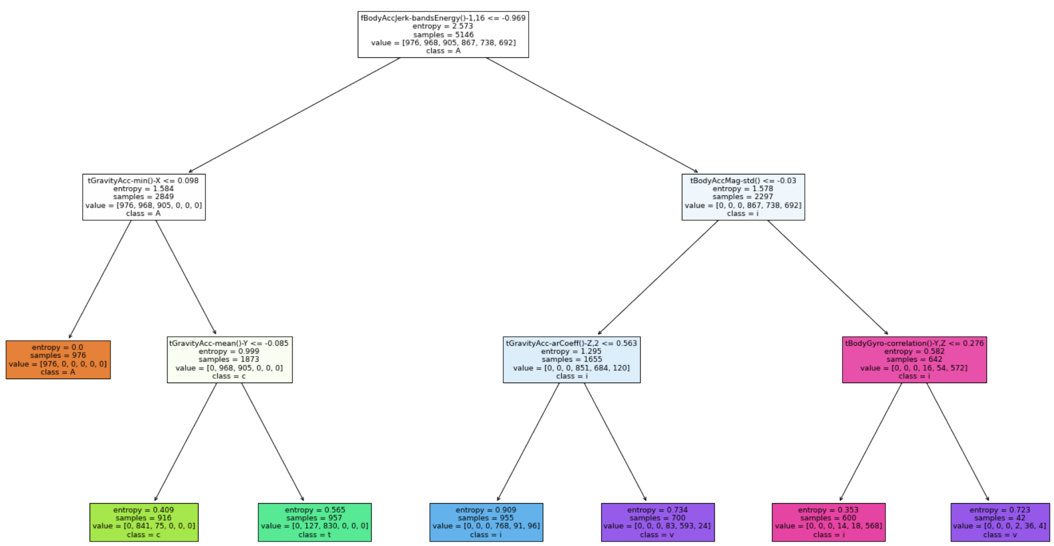
[59]:
```
text_representation = tree.export_text(dtc)
print(text_representation)
```

```
|--- feature_389 <= -0.97
|   |--- feature_52 <= 0.10
|   |   |--- class: 1
|   |--- feature_52 >  0.10
|   |   |--- feature_41 <= -0.08
|   |   |   |--- class: 2
|   |   |--- feature_41 >  -0.08
|   |   |   |--- class: 3
|--- feature_389 >  -0.97
|   |--- feature_201 <= -0.03
|   |   |--- feature_74 <= 0.56
|   |   |   |--- class: 4
|   |   |--- feature_74 >  0.56
|   |   |   |--- class: 5
```

```
|   |--- feature_201 >  -0.03
|   |   |--- feature_159 <= 0.28
|   |   |   |--- class: 6
|   |   |--- feature_159 >  0.28
|   |   |   |--- class: 5
```

[60]:
```python
x = list(data.columns)
feature_names = x[:-1]
target_name = x[-1]
```

[61]:
```python
fig = plt.figure(figsize=(25,15))
_ = tree.plot_tree(dtc,
                   feature_names=feature_names,
                   class_names=target_name,
                   filled=True)
```



11 Random Forest Classifier

[61]:
```python

```

[62]:
```python
from sklearn.ensemble import RandomForestClassifier

#Create a Gaussian Classifier
model =RandomForestClassifier(n_estimators=100)
```

```
#Train the model using the training sets y_pred=clf.predict(X_test)
model.fit(X_train,y_train)

y_pred=model.predict(X_test)
```

[62]:

12 Checking Model's Accuracy on Training Set (RFC)

[63]:
```
training_accuracy_dict["RFC"] = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.9777878513145966
```

[64]:
```
print("Precision Score : ",metrics.precision_score(y_test, y_pred,
                                           pos_label='positive',
                                           average='micro'))
print("Recall Score : ",metrics.recall_score(y_test, y_pred,
                                           pos_label='positive',
                                           average='micro'))
```

```
Precision Score :  0.9777878513145966
Recall Score :  0.9777878513145966
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1321:
UserWarning: Note that pos_label (set to 'positive') is ignored when average !=
'binary' (got 'micro'). You may use labels=[pos_label] to specify a single
positive class.
  % (pos_label, average), UserWarning)
```

[64]:

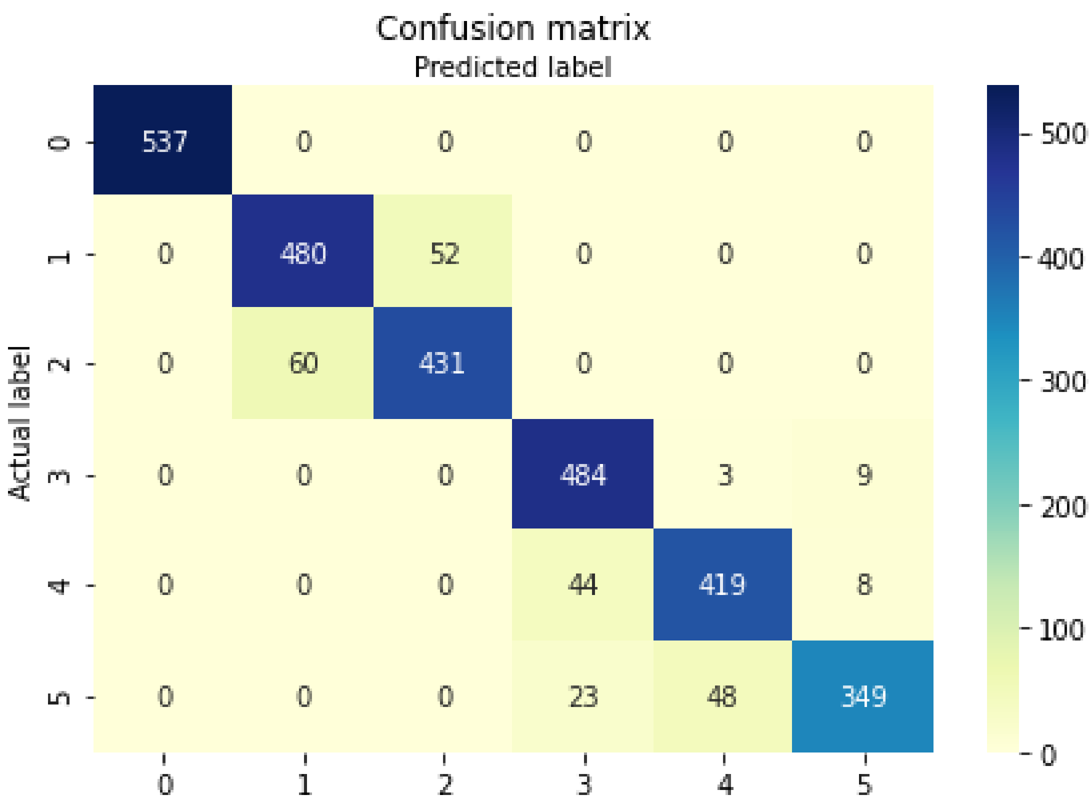
13 Checking Model's Accuracy on Test Set (RFC)

[65]:
```
X_validation = test_Data.drop(labels = 'Activity',axis=1)
y_validation = test_Data['Activity'].replace(mapping).values
y_pred_validation = model.predict(X_validation)

testing_accuracy_dict["RFC"] = metrics.accuracy_score(y_pred_validation,␣
 ↪y_validation)

print("Accuracy:",metrics.accuracy_score(y_pred_validation, y_validation))
```

```
Accuracy: 0.9161859518154055
```

[66]:
```
cnf_matrix = confusion_matrix(y_validation, y_pred_validation)

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('Actual label')
plt.xlabel('Predicted label')
```

[66]: Text(0.5, 257.44, 'Predicted label')



13.1 SVM (Support Vector Machines)

[67]:
```
from sklearn import svm

#Create a svm Classifier
svm_classifier = svm.SVC(kernel='linear') # Linear Kernel
```

```python
#Train the model using the training sets
svm_classifier.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = svm_classifier.predict(X_test)
```

14 Checking Model's Accuracy on Test Set (SVM)

[68]:
```python
training_accuracy_dict["SVM"] = metrics.accuracy_score(y_test, y_pred)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.985494106980961
```

[69]:
```python
print("Precision Score : ",metrics.precision_score(y_test, y_pred,
                                           pos_label='positive',
                                           average='micro'))
print("Recall Score : ",metrics.recall_score(y_test, y_pred,
                                           pos_label='positive',
                                           average='micro'))
```

```
Precision Score :  0.985494106980961
Recall Score :  0.985494106980961

/usr/local/lib/python3.6/dist-packages/sklearn/metrics/_classification.py:1321:
UserWarning: Note that pos_label (set to 'positive') is ignored when average !=
'binary' (got 'micro'). You may use labels=[pos_label] to specify a single
positive class.
  % (pos_label, average), UserWarning)
```

[69]:
```python

```

15 Checking Model's Accuracy on Test Set (SVM)

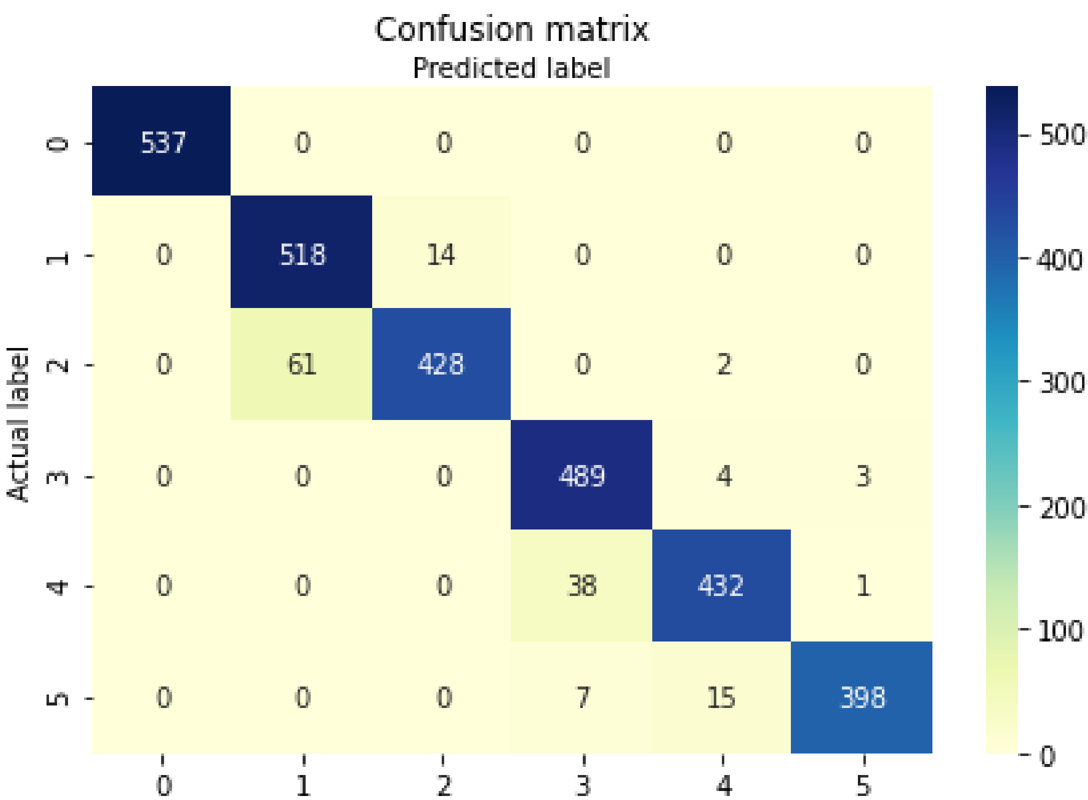
[70]:
```python
X_validation = test_Data.drop(labels = 'Activity',axis=1)
y_validation = test_Data['Activity'].replace(mapping).values
y_pred_validation = svm_classifier.predict(X_validation)

testing_accuracy_dict["SVM"] = metrics.accuracy_score(y_pred_validation,␣
 ↪y_validation)

print("Accuracy:",metrics.accuracy_score(y_pred_validation, y_validation))
```

```
Accuracy: 0.9507974211062097
```

[70]:

[71]:
```
cnf_matrix = confusion_matrix(y_validation, y_pred_validation)

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('Actual label')
plt.xlabel('Predicted label')
```

[71]: Text(0.5, 257.44, 'Predicted label')



16 ANN (Artifical Neural Network)

[72]:

```

```

[73]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils.np_utils import to_categorical
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

n_input = X_train.shape[1] # number of features
n_output = 6 # number of possible labels
n_samples = X_train.shape[0] # number of training samples
n_hidden_units = 40

Y_train = to_categorical(y_train)
Y_test = to_categorical(y_test)
print(Y_train.shape)
print(Y_test.shape)

def create_model():
    model = Sequential()
    model.add(Dense(n_hidden_units,
                    input_dim=n_input,
                    activation="relu"))
    model.add(Dense(n_hidden_units,
                    input_dim=n_input,
                    activation="relu"))
    model.add(Dense(n_output, activation="softmax"))

    # Compile model
    model.compile(loss="categorical_crossentropy", optimizer="adam",␣
 ↪metrics=['accuracy'])
    return model
```

```
(5146, 7)
(2206, 7)
```

[74]:

```
estimator = KerasClassifier(build_fn=create_model, epochs=20, batch_size=10,␣
 ↪verbose=False)
estimator.fit(X_train, y_train)
print("Score: {}".format(estimator.score(X_test, y_test)))
```

```
Score: 0.9868540167808533
```

[75]:
```python
score = estimator.score(X_test, y_test)
```

[76]:
```python
training_accuracy_dict['ANN'] = score
```

[77]:
```python
validation_score = estimator.score(X_validation, y_validation)
```

[78]:
```python
validation_score
```

[78]:
```
0.22192059457302094
```

[79]:
```python
testing_accuracy_dict['ANN'] = validation_score
```

[80]:
```python
testing_accuracy_dict
```

[80]:
```
{'ANN': 0.22192059457302094,
 'Decision_Tree': 0.8123515439429929,
 'KNN': 0.7964031218187988,
 'Logistic_Regression': 0.9504580929759077,
 'Naive_Bayes': 0.6861214794706482,
 'RFC': 0.9161859518154055,
 'SVM': 0.9507974211062097}
```

17 Results

[81]:
```python
training_accuracy_dict
```

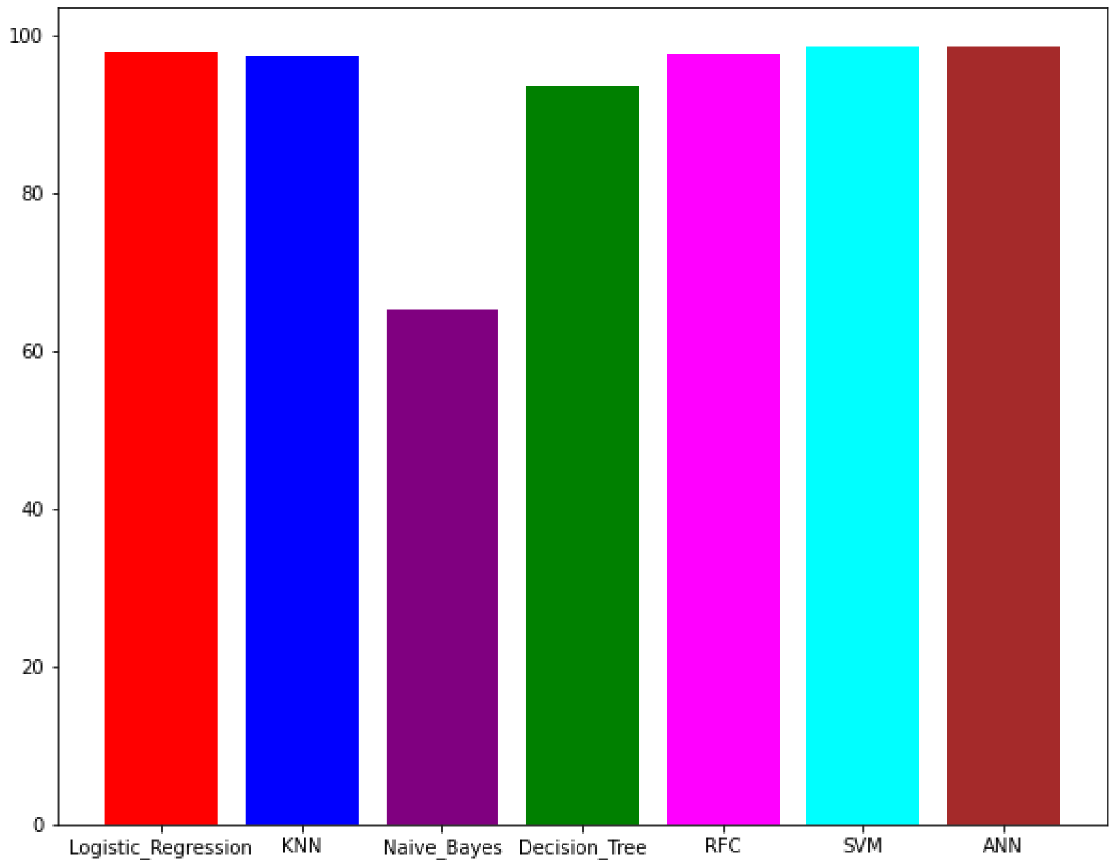
[81]:
```
{'ANN': 0.9868540167808533,
 'Decision_Tree': 0.9369900271985494,
 'KNN': 0.9737080689029919,
 'Logistic_Regression': 0.9782411604714415,
 'Naive_Bayes': 0.6518585675430644,
 'RFC': 0.9777878513145966,
 'SVM': 0.985494106980961}
```

[82]:
```python
testing_accuracy_dict
```

[82]:
```
{'ANN': 0.22192059457302094,
 'Decision_Tree': 0.8123515439429929,
 'KNN': 0.7964031218187988,
 'Logistic_Regression': 0.9504580929759077,
 'Naive_Bayes': 0.6861214794706482,
 'RFC': 0.9161859518154055,
 'SVM': 0.9507974211062097}
```

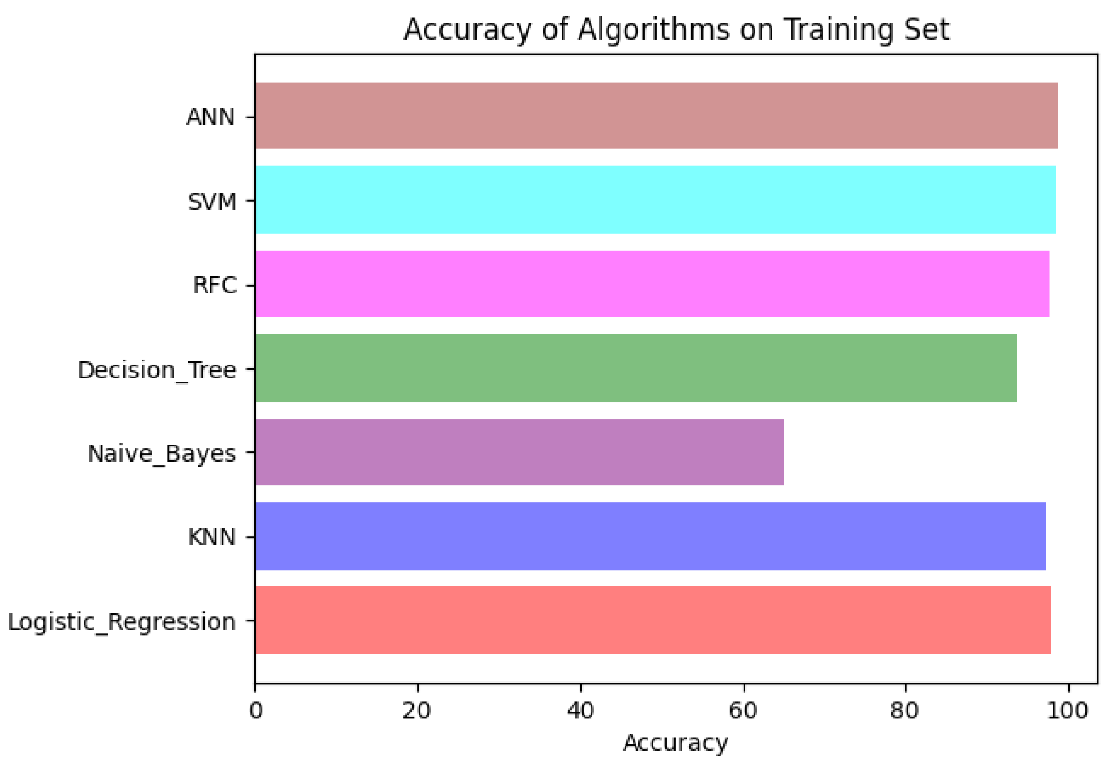
[83]:
```python
keys = training_accuracy_dict.keys()

values = training_accuracy_dict.values()
values = [i*100 for i in values]

plt.figure(figsize = (10,8))
plt.bar(keys, values,color=['red', 'blue', 'purple', 'green',␣
↪'fuchsia','cyan','brown'])
```

[83]: <BarContainer object of 7 artists>



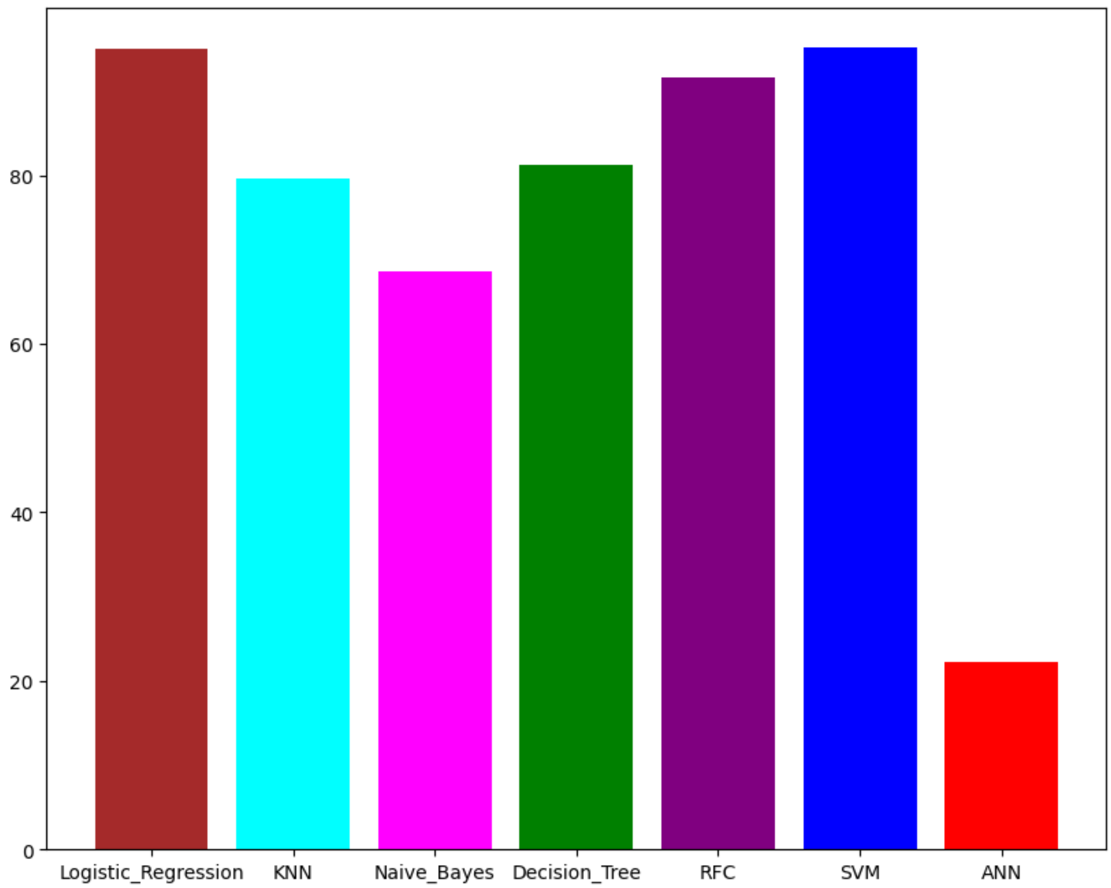
[84]:
```python
import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt


color = ['red', 'blue', 'purple', 'green', 'fuchsia','cyan','brown']
plt.barh(list(keys), values, align='center', alpha=0.5, color = color)
plt.yticks(list(keys))
plt.xlabel('Accuracy')
```

```
plt.title('Accuracy of Algorithms on Training Set')

plt.show()
```

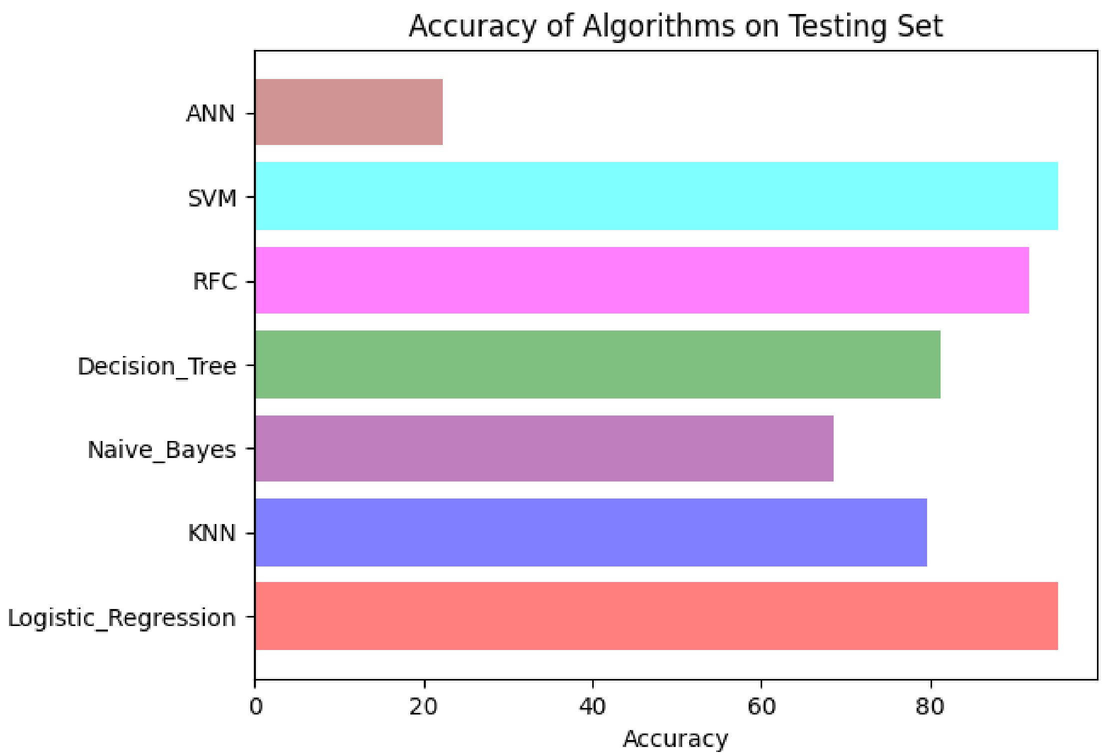


[85]:
```
keys = testing_accuracy_dict.keys()

values = testing_accuracy_dict.values()
values = [i*100 for i in values]

plt.figure(figsize = (10,8))
color = ['red', 'blue', 'purple', 'green', 'fuchsia','cyan','brown']
color = color[::-1]
plt.bar(keys, values,color=color)
```

[85]: <BarContainer object of 7 artists>



[86]:
```
import matplotlib.pyplot as plt; plt.rcdefaults()
import numpy as np
import matplotlib.pyplot as plt



plt.barh(list(keys), values, align='center', alpha=0.5,color=color[::-1])
plt.yticks(list(keys))
plt.xlabel('Accuracy')
plt.title('Accuracy of Algorithms on Testing Set')

plt.show()
```



[87]:
```
new_dict = training_accuracy_dict.copy()

for key, value in new_dict.items():
    new_dict[key] = str(value * 100)
    new_dict[key] = new_dict[key][:-12] + '%'


x = pd.DataFrame.from_dict(new_dict, orient='index')
x
```

[87]:

	0
Logistic_Regression	97.82%
KNN	97.37%
Naive_Bayes	65.18%
Decision_Tree	93.69%
RFC	97.77%
SVM	98.5%
ANN	98.68%

[88]:
```
new_dict = testing_accuracy_dict.copy()

for key, value in new_dict.items():
    new_dict[key] = str(value * 100)
```

```python
    new_dict[key] = new_dict[key][:-12] + '%'


x = pd.DataFrame.from_dict(new_dict, orient='index')
x
```

[88]:

	0
Logistic_Regression	95.04%
KNN	79.64%
Naive_Bayes	68.61%
Decision_Tree	81.23%
RFC	91.61%
SVM	95.07%
ANN	22.192%

[102]:

```python

```

[102]: (7352, 563)

[ ]:

```python

```