Machine learning Assignment 2

Sai Mounica Chenuri Venkata V Lakshmi Phalguna

Z963A577(outputs for 1st 250 samples of data)

Code:

```
from google.colab import drive
#drive.mount('/content/drive')
drive.mount('/content/drive')

from google.colab import files
import pandas as pd
import io
import matplotlib.pyplot as plt
from sklearn import linear_model
import numpy as np
import pandas as pd
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, MinMaxScaler, PolynomialFeatures, StandardScaler
from sklearn import preprocessing, metrics
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, f1_score

from sklearn.metrics import precision_score

from sklearn.model_selection import  GridSearchCV

from sklearn.neighbors import KNeighborsClassifier

from pandas import DataFrame

from sklearn.metrics import classification_report

from sklearn.tree import DecisionTreeClassifier


#uploading the data

input_data = pd.read_csv("/content/drive/My Drive/HW2ML.csv")

input_data.head()


# data.head()

input_data.isnull().any()

input_data['emotion'].unique()

df = DataFrame(input_data)

data_dictionay = {0:'anger', 1:'disgust', 2:'fear', 3:'happiness', 4: 'sadness', 5: 'surprise', 6: 'neutral'}

input_data['emotion'].value_counts()

input_data.dtypes
```

```
input_data.isnull().any()
input_data['emotion'].unique()
df = DataFrame(input_data)
data_dictionay = {0:'anger', 1:'disgust', 2:'fear', 3:'happiness', 4: 'sadness', 5: 'surprise', 6: 'neutral'}

input_data['emotion'].value_counts()

input_data.dtypes
```

```
emotion     int64
 Usage     object
 pixels    object
dtype: object
```

```
# assigning variables to the data

a = input_data['emotion']

B=input_data[' pixels'].apply(lambda x : np.array(x.split(' ')).astype(float))
```

B.shape

#printing pixel values

B = np.stack(B, axis=0)

print(B)

```
[11] # assigning variables to the data
     a = input_data['emotion']

     B=input_data[' pixels'].apply(lambda x : np.array(x.split(' ')).astype(float))

     B.shape

     (250,)

[12] #printing pixel values

     B = np.stack(B, axis=0)
     print(B)

     [[ 70.  80.  82. ... 106. 109.  82.]
      [151. 150. 147. ... 193. 183. 184.]
      [231. 212. 156. ...  88. 110. 152.]
      ...
      [ 10.  11.  11. ...  13.  16.  21.]
      [148. 151. 149. ... 184. 182. 180.]
      [ 43.  42.  36. ... 103.  97.  95.]]
```

#diving the data test and train set and printing the data

B_train, B_test, a_train, a_test = train_test_split(B, a, test_size=0.15, random_state=30)

print(B_train)

print(B_test)

print(a_train)

print(a_test)

#alling the lenghts of the data for easy computation

len(B_train) == len(a_train)

print(len(B_train))

print(len(a_train))

```
B_train, B_test, a_train, a_test = train_test_split(B, a, test_size=0.15, random_state=30)
print(B_train)
print(B_test)
print(a_train)
print(a_test)
```

```
[[241. 240. 196. ... 232. 233. 233.]
 [136. 137. 136. ... 137. 136. 135.]
 [206. 142.  59. ...  24.  46.  65.]
 ...
 [124. 101.  98. ... 141. 108.  55.]
 [ 44.  64.  55. ...  90. 110. 116.]
 [ 40.  46.  42. ...  17.  18.  18.]]
[[ 65.  85.  94. ... 190. 207. 210.]
 [ 65.  49.  54. ...  96.  99. 105.]
 [ 81. 106.  91. ... 103. 105. 105.]
 ...
 [ 42.  42.  37. ...  18.  12.   2.]
 [ 43.  43.  38. ...   4.   3.   3.]
 [  7.   5.   8. ... 153. 145. 135.]]
110    2
203    0
143    3
146    4
126    0
      ..
244    2
45     2
173    2
```

```
[14] #alling the lenghts of the data for easy computation

len(B_train)  == len(a_train)

print(len(B_train))
print(len(a_train))
```

```
212
212
```

SVM_Poly:

Code:

svm_poly_estimator = Pipeline(steps=[('scaler', StandardScaler()), ('estimator', SVC())])

svm_poly_params = [{'estimator__kernel': ["poly"],

'estimator__degree': [5,10,40],

'estimator__C': [0.1,10,1000],

'estimator__gamma': [.00001,.0001,.01,1,1.5]}]

svm_poly_grid = GridSearchCV(estimator=svm_poly_estimator,

param_grid=svm_poly_params,

cv=5,

n_jobs=-1,)

svm_poly_grid.fit(B_train, a_train)

print('svm_poly F1: ', f1_score(a,svm_poly_grid.predict(B), average='micro'))

print('best svm_poly_parameters', svm_poly_grid.best_estimator_)SVM_Rbf

"""# SVM RBF"""

svm_rbf_estimator = Pipeline(steps=[('scaler', StandardScaler()), ('estimator', SVC())])

svm_rbf_params = [{'estimator__kernel': ["rbf"],

    'estimator__degree': [5,10,40],

    'estimator__C': [0.1,10,1000],

    'estimator__gamma': [.00001,.0001,.01,1,1.5]}]

svm_rbf_grid = GridSearchCV(estimator=svm_rbf_estimator,

    param_grid=svm_rbf_params,

    cv=5,

    n_jobs=-1,)

svm_rbf_grid.fit(B_train, a_train)

print('svm_poly F1: ', f1_score(a,svm_rbf_grid.predict(B), average='micro'))

print('best svm_rbf_parameters', svm_rbf_grid.best_params_)

```
[31] svm_rbf_estimator = Pipeline(steps=[('scaler', StandardScaler()), ('estimator', SVC())])

     svm_rbf_params = [{'estimator__kernel': ["rbf"],
                        'estimator__degree': [5,10,40],
                        'estimator__C': [0.1,10,1000],
                        'estimator__gamma': [.00001,.0001,.01,1,1.5]}]

     svm_rbf_grid = GridSearchCV(estimator=svm_rbf_estimator,
                            param_grid=svm_rbf_params,
                            cv=5,
                            n_jobs=-1,)

     svm_rbf_grid.fit(B_train, a_train)


     print('svm_poly F1: ', f1_score(a,svm_rbf_grid.predict(B), average='micro'))
     print('best svm_rbf_parameters', svm_rbf_grid.best_params_)

     svm_poly F1:  0.868
     best svm_rbf_parameters {'estimator__C': 10, 'estimator__degree': 5, 'estimator__gamma': 0.0001, 'estimator__kernel': 'rbf'}
```

"""# Decision tree"""

dt_params = {

  'max_depth': [3,5,7,8,10]

}

decision_grid = GridSearchCV(DecisionTreeClassifier(),dt_params)

decision_grid.fit(B_train, a_train)

#best = decision_grid.best_estimator_

#print(best)s

predict_values = decision_grid.predict(B_test)

best.fit(B_train, a_train)

f1_score = f1_score(a_test, predict_values,average='micro')

print(f1_score)

print('best decision_grid ', decision_grid.best_params_)

```
#Decission tree

dt_params = {
    'max_depth': [3,5,7,8,10]
}


decision_grid = GridSearchCV(DecisionTreeClassifier(),dt_params)

decision_grid.fit(B_train, a_train)


# best = decision_grid.best_estimator_
# print(best)

predict_values = decision_grid.predict(B_test)

best.fit(B_train, a_train)

f1_score  = f1_score(a_test, predict_values,average='micro')
print(f1_score)
print('best decision_grid ', decision_grid.best_params_)
```

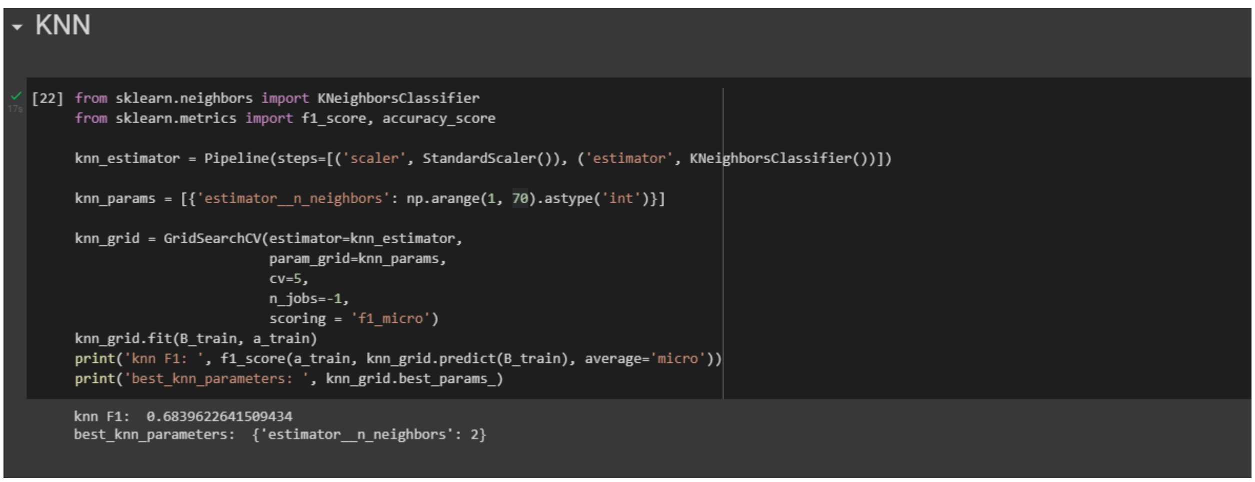
```
0.21052631578947367
best decision_grid  {'max_depth': 8}
```

"""# KNN"""

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import f1_score, accuracy_score

```
knn_estimator = Pipeline(steps=[('scaler', StandardScaler()), ('estimator', KNeighborsClassifier())])

knn_params = [{'estimator__n_neighbors': np.arange(1, 70).astype('int')}]

knn_grid = GridSearchCV(estimator=knn_estimator,

                    param_grid=knn_params,

                    cv=5,

                    n_jobs=-1,

                    scoring = 'f1_micro')

knn_grid.fit(B_train, a_train)

print('knn F1: ', f1_score(a_train, knn_grid.predict(B_train), average='micro'))

print('best_knn_parameters: ', knn_grid.best_params_)
```



KNN

```
[22] from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import f1_score, accuracy_score

     knn_estimator = Pipeline(steps=[('scaler', StandardScaler()), ('estimator', KNeighborsClassifier())])

     knn_params = [{'estimator__n_neighbors': np.arange(1, 70).astype('int')}]

     knn_grid = GridSearchCV(estimator=knn_estimator,
                             param_grid=knn_params,
                             cv=5,
                             n_jobs=-1,
                             scoring = 'f1_micro')
     knn_grid.fit(B_train, a_train)
     print('knn F1: ', f1_score(a_train, knn_grid.predict(B_train), average='micro'))
     print('best_knn_parameters: ', knn_grid.best_params_)

     knn F1:  0.6839622641509434
     best_knn_parameters:  {'estimator__n_neighbors': 2}
```