hw8-jack

May 15, 2022

Data Analytics

Homework 8

Name: ID:10546004

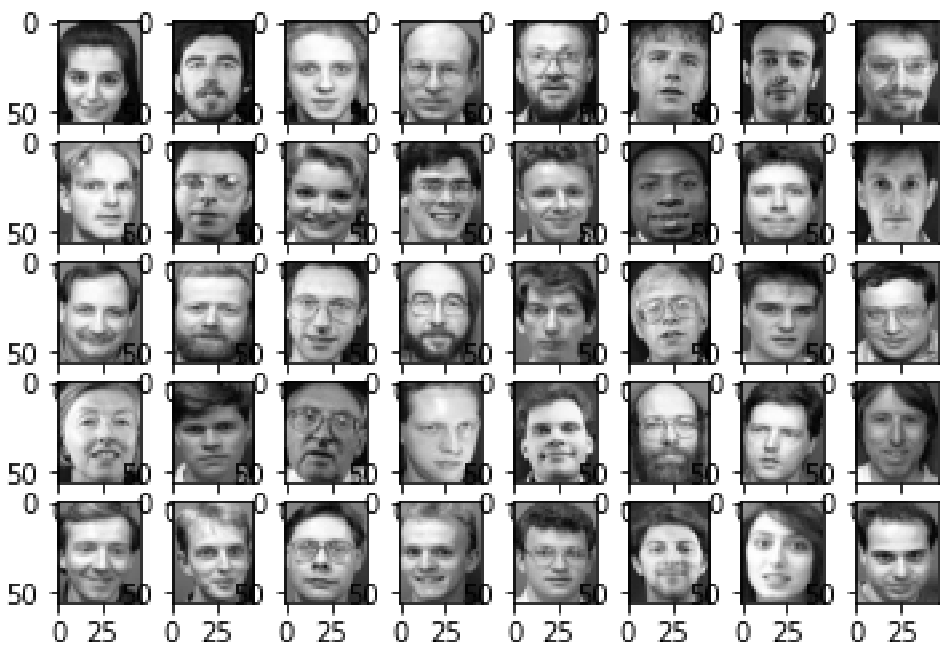
1 Q1

[ ]:
```
from zipfile import ZipFile
import numpy as np
import pandas as pd
from matplotlib import image
import matplotlib.pyplot as plt
from scipy import stats
import os
import re
import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import svm
%matplotlib inline
```

1.0.1 loading ORL faces data

[ ]:
```
# get current working directory
cwd= os.getcwd()

# add the read directory to the path
rd = os.path.join(cwd)
zip_file = ZipFile('ORL Faces.zip')
dfs = {png_file.filename: image.imread(zip_file.open(png_file.filename))
       for png_file in zip_file.infolist()
       if png_file.filename.endswith('.png')}
png_df = pd.DataFrame([dfs])

# define gender by hand, each value for 10 pictures
col_name = list(png_df.columns)
```

```
fig, ax = plt.subplots(5, 8)
genders = [0,1,1,1,1,1,1,1,
           1,1,0,1,1,1,1,1,
           1,1,1,1,1,1,1,1,
           0,1,1,1,1,1,1,1,
           1,1,1,1,1,1,0,1]
dic = {}
k = 0
# insert all gender to new row
for i in col_name:
    match = re.search(r'\w*\s*\w*\/\d*\_1.png',i)
    if match:
        #print(match.group(), k)
        plt.subplot(5,8,k+1)
        plt.imshow(png_df[i][0],cmap='gray')
        k = k +1
    dic[i]=[genders[k-1]]
```




```
[ ]: # flatten all pics data
df2 = pd.concat([png_df,pd.DataFrame(dic)]).reset_index(drop=True)
sample_df = df2.rename(index={0:'data',1:'gender'}).T
data_mat = np.array(np.ndarray.flatten(sample_df['data'][0]))
for i in range(1, len(sample_df)):
    data_mat = np.concatenate((data_mat,
         np.array(np.ndarray.flatten(sample_df['data'][i]))))
```

```
data_mat = data_mat.reshape(400,2576)
data_mat.shape
```

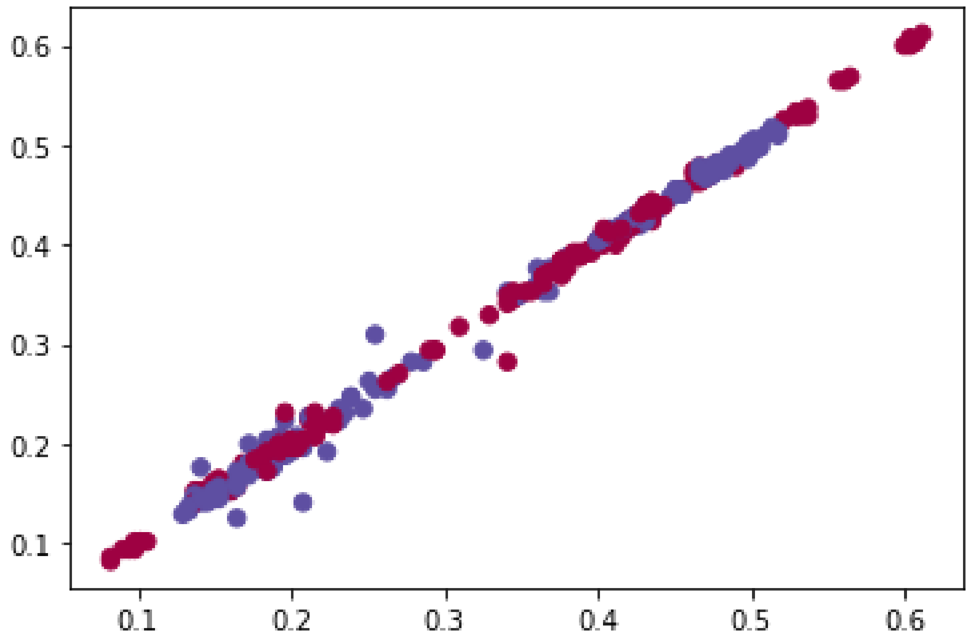
[ ]: (400, 2576)

```
[ ]: class KMeansClustering:
        def __init__(self, X, num_clusters,max_iterations=100):
            self.K = num_clusters
            self.max_iterations = 100
            self.num_examples = X.shape[0]
            self.num_features = X.shape[1]
            self.plot_figure = True

            # print((self.num_examples, self.num_features))

        def initialize_random_centroids(self, X):
            # print(self.num_features, X.shape)
            # print(np.zeros((self.K, self.num_features)))
            centroids = np.zeros((self.K, self.num_features))

            for k in range(self.K):
                centroid = X[np.random.choice(range(self.num_examples))]
                centroids[k] = centroid

            return centroids

        def create_clusters(self, X, centroids):
            # Will contain a list of the points that are associated with that␣
    ↪specific cluster
            clusters = [[] for _ in range(self.K)]

            # Loop through each point and check which is the closest cluster
            for point_idx, point in enumerate(X):
                closest_centroid = np.argmin(
                    np.sqrt(np.sum((point - centroids) ** 2, axis=1))
                )
                clusters[closest_centroid].append(point_idx)

            return clusters

        def calculate_new_centroids(self, clusters, X):
            # print(self.num_features, X.shape)

            centroids = np.zeros((self.K, self.num_features))
            for idx, cluster in enumerate(clusters):
                new_centroid = np.mean(X[cluster], axis=0)
                centroids[idx] = new_centroid
```

```
        return centroids

    def predict_cluster(self, clusters, X):
        y_pred = np.zeros(self.num_examples)

        for cluster_idx, cluster in enumerate(clusters):
            for sample_idx in cluster:
                y_pred[sample_idx] = cluster_idx

        return y_pred


    def plot_fig(self, X, y):
        plt.scatter(X[:, 0], X[:, 1], c=y, s=40, cmap=plt.cm.Spectral)
        plt.show()

    def fit(self, X):
        centroids = self.initialize_random_centroids(X)

        for it in range(self.max_iterations):
            clusters = self.create_clusters(X, centroids)

            previous_centroids = centroids
            centroids = self.calculate_new_centroids(clusters, X)

            diff = centroids - previous_centroids

            if not diff.any():
                print("Termination criterion satisfied")
                break

        # Get label predictions
        y_pred = self.predict_cluster(clusters, X)

        if self.plot_figure:
            self.plot_fig(X, y_pred)

        return y_pred
```

[ ]:
```
k_means_models = KMeansClustering(data_mat , 2)
y_pred = k_means_models.fit(data_mat)
```

```
Termination criterion satisfied
```



[ ]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(sample_df['gender'].
 ↪to_numpy(dtype=int),y_pred,normalize='true')
print('The confusion matrix for each cell are \n',np.round(cm,2) )
```

```
The confusion matrix for each cell are
 [[0.25 0.75]
 [0.48 0.52]]
```

1.0.2 Q1 results

From the upper confusion matrix, we know the `boy =1` , `girl = 1` is not good for cluster 2 group. The accuracy is :

$$\frac{TP + TF}{TP + FN + FP + TF} = \frac{.25 + .52}{.25 + .48 + .75 + .52} = 0.405$$

2 Q2

[ ]:
```
# Read Text Files with Pandas

col_names = ['mpg','cylinders','displacement','horsepower',
             'weight','acceleration','year','origin','car_name']
# read text file into pandas DataFrame
df = pd.read_fwf("auto-mpg.data.txt",header=None,names = col_names)
# df = pd.DataFrame(df1.to_numpy() , columns=col_names)
```

```
# display DataFrame
df = df[~df.isin({'?'}).any(1)]
print(df)
```

```
     mpg  cylinders  displacement horsepower  weight  acceleration  year  \
0    18.0          8         307.0      130.0  3504.0          12.0    70
1    15.0          8         350.0      165.0  3693.0          11.5    70
2    18.0          8         318.0      150.0  3436.0          11.0    70
3    16.0          8         304.0      150.0  3433.0          12.0    70
4    17.0          8         302.0      140.0  3449.0          10.5    70
..    …        …            …         …      …             …    …
393  27.0          4         140.0      86.00  2790.0          15.6    82
394  44.0          4          97.0      52.00  2130.0          24.6    82
395  32.0          4         135.0      84.00  2295.0          11.6    82
396  28.0          4         120.0      79.00  2625.0          18.6    82
397  31.0          4         119.0      82.00  2720.0          19.4    82

     origin                     car_name
0         1  "chevrolet chevelle malibu"
1         1          "buick skylark 320"
2         1         "plymouth satellite"
3         1              "amc rebel sst"
4         1                "ford torino"
..      …                          …
393       1            "ford mustang gl"
394       2                  "vw pickup"
395       1              "dodge rampage"
396       1                "ford ranger"
397       1                 "chevy s-10"

[392 rows x 9 columns]
```

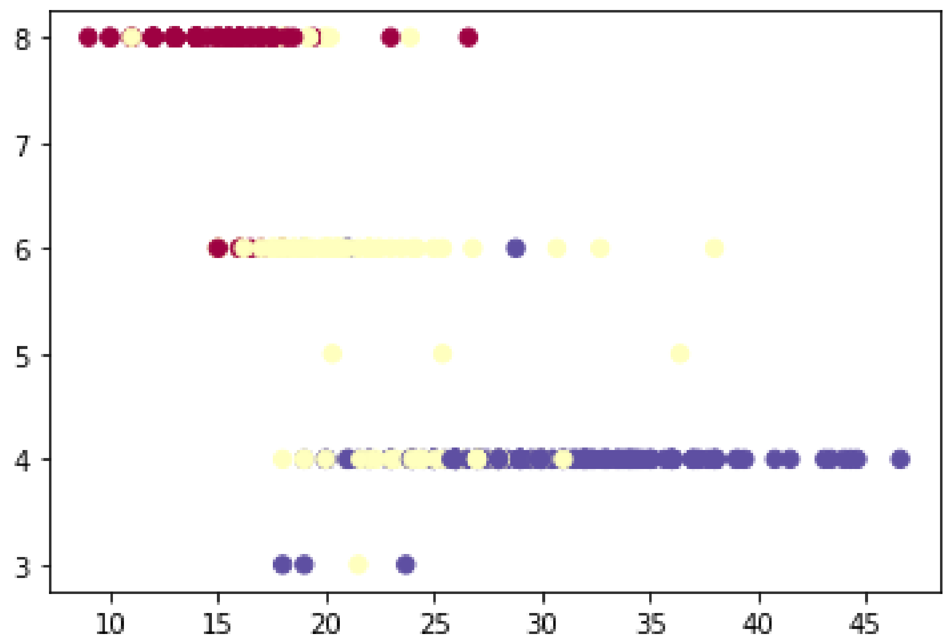
[ ]:
```
#define predictor and response variables
import itertools

m = df.loc[:,['mpg','cylinders','displacement','horsepower',
            'weight','acceleration', 'year']].astype(float).to_numpy()
X = m
y = df.loc[:,['origin'] ]
y= list(itertools.chain(*y.values))
```

[ ]:
```
from collections import Counter

# double check the unique value in y
print(f'the unique value for origin are \n')
unique = [k for k,v in Counter(y).items()]
print(unique)
```

```
the unique value for origin are

[1, 3, 2]
```

[ ]:
```python
k_means_models = KMeansClustering(X , 3, max_iterations=1e5)
y_pred = k_means_models.fit(X)
cm = confusion_matrix(np.array(y), y_pred.astype(int),normalize='true')
print('The k-means confusion matrix for each cell are \n',np.round(cm[1:4,0:
↪3],2) )
```

```
Termination criterion satisfied
```




```
The k-means confusion matrix for each cell are
 [[0.36 0.39 0.24]
 [0.01 0.25 0.74]
 [0.   0.11 0.89]]
```

[ ]:
```python
from sklearn.cluster import AgglomerativeClustering

cluster = AgglomerativeClustering(n_clusters=len(unique), affinity='euclidean',␣
↪linkage='ward')
y_pred = cluster.fit_predict(X) +1
cm = confusion_matrix(np.array(y), y_pred.astype(int),normalize='true')
print('The Hierarchical Clustering confusion matrix for each cell are \n',np.
↪round(cm,2) )
```

```
The Hierarchical Clustering confusion matrix for each cell are
```

```
[[0.53 0.36 0.11]
 [0.04 0.4  0.56]
 [0.   0.33 0.67]]
```

```
[ ]: from sklearn.cluster import DBSCAN

     DBSCAN_cluster = DBSCAN(eps=9, min_samples=3).fit(X)
     set(DBSCAN_cluster.labels_)
```

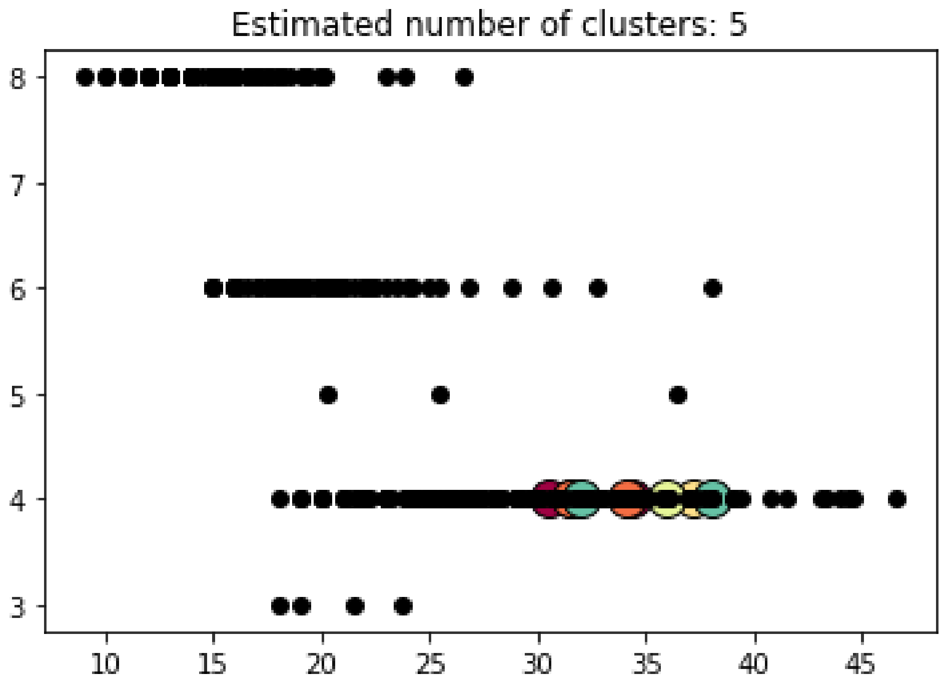
```
[ ]: {-1, 0, 1, 2, 3, 4}
```

```
[ ]: labels = DBSCAN_cluster.labels_
     n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
     print(f'The DBSCAN result have {n_clusters} clusters by droping the Noisy␣
      ↪samples')
```

```
The DBSCAN result have 5 clusters by droping the Noisy samples
```

After playing with many parameters, I found mmore than 3 levels.
I guess because this data set has correlation factors.
This unbalanced levels make difficult to compare the `origin` factor.
The visulization might easier to see.

```
[ ]: # Black removed and is used for noise instead.
     core_samples_mask = np.zeros_like(DBSCAN_cluster.labels_, dtype=bool)
     core_samples_mask[DBSCAN_cluster.core_sample_indices_] = True
     unique_labels = set(labels)
     colors = [plt.cm.Spectral(each)
               for each in np.linspace(0, 1, len(unique_labels))]
     for k, col in zip(unique_labels, colors):
         if k == -1:
             # Black used for noise.
             col = [0, 0, 0, 1]

         class_member_mask = (labels == k)

         xy = X[class_member_mask & core_samples_mask]
         plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
                  markeredgecolor='k', markersize=14)

         xy = X[class_member_mask & ~core_samples_mask]
         plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
                  markeredgecolor='k', markersize=6)

     plt.title('Estimated number of clusters: %d' % n_clusters)
     plt.show()
```



2.0.1 Logistic Regression results

[ ]:
```
#define predictor and response variables
m = df.loc[:,['mpg','cylinders','displacement','horsepower',
            'weight','acceleration', 'year']].astype(float).to_numpy()
X = m
y = df.loc[:,['origin'] ]
X_train, X_test, y_train, y_test = train_test_split(X,y,train_size=.
 ↪2,random_state=1)
LR = LogisticRegression().fit(X_train,y_train)
score = LR.score(X_test,y_test)
print('Accuracy: ', score)
```

```
Accuracy:  0.7643312101910829
```

```
/usr/lib/python3.10/site-packages/sklearn/utils/validation.py:993:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
/usr/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:814:
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(
```

2.0.2 -Nearest Neighbors results

```
[ ]: neigh = KNeighborsClassifier(n_neighbors=3).fit(X_train, y_train)
     print('Accuracy: ' , neigh.score(X_test,y_test))
```

```
Accuracy:  0.6146496815286624

/usr/lib/python3.10/site-packages/sklearn/neighbors/_classification.py:198:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples,), for example using
ravel().
  return self._fit(X, y)
```

2.0.3 Support vector machine results

```
[ ]: svm_model = svm.SVC(kernel='poly').fit(X_train, y_train)
     print('Accuracy: ' , svm_model.score(X_test,y_test))
```

```
Accuracy:  0.6878980891719745

/usr/lib/python3.10/site-packages/sklearn/utils/validation.py:993:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

2.0.4 Q2 Discussion

The supervised learning is beter than the unsupervised.
In AutoMPG data, the accuracy does not good compare to ORL face data.
Because we found `cylinders`,`displacement`,`horsepower` and `weight` are correlated factors, this will impact on model classification rate.