Module 6 Assignment 1: Digit Recognizer

Nadeem Patel

MSDS 422 Winter 2022, Section 55

Northwestern University, Practical Machine Learning

02/13/2022

When looking at the "MNIST Digit Recognizer" data, explaratory data analysis (EDA) provided extensive information regarding the pixel-value features and the labels. The labels appear to be digits that are represented by the 784 pixels, which can be considered the features. Additionally, EDA showed that there are 42,000 entries in the training dataset. When looking at the description of the training data, it could be was seen that the average label is around 4.46, the minimum is 0.00 and the maximum is 9.0, so it is clear that the digits represented in the images are from 0 to 9. Finally, the view of the images showed that each digit is written differently for every number, which means there is a possibility numbers could be mix up. For the most part, however, it appeared that every number was clearly written.

The training data was adjusted so the labels were dropped, and a made its own dataset. Then, the RandomForestClassifier function form Python was used with n_estimators=784 as the baseline. It took a little over four minutes to run the model. When performing cross-validation, the F1 score, precision score, and the recall score were all around 0.966. After training the model, it was run on the testing dataset, and the cross-validation took around nine minutes that gave a F1 score, precision score, and recall score all around 0.981. A confusion matrix was also built based on the testing set to get an idea of pixels.

The training and testing datasets were combined before executing principal components analysis (PCA), generating principal components that represent 95% of the variability in the explanatory variables, which took 14.65 seconds. The RandomForestClassifier was run again with n_estimators=154, which represents the principal components acquired from the 95% variability. This model took about two minutes to run, and the cross-validation took around seven minutes. The F1 score, precision score, and recall score based on the cross-validation scores came out to be around 0.942. When running the model on the testing set, and the cross-validation took around two minutes to run. The F1 score, precision score, and recall score all came out to be around 0.957.

For k-means clustering, the training values were reshaped, and the observations were grouped into one of 10 categories before assigning labels. Through this, the cluster labels and predicted labels were obtained. The cross-validation was run on the new labels, which took a little over a minute. The F1 score and the recall score was around 0.0643 while the precision score was around 0.080.

The flaw in the experiment was that PCA was performed on combined data rather than training and testing data sets separately. This way, the principal components can be saved after running PCA on the training set, and the test set can be transformed separately. The RandomForestClassifier was run again with n_estimators=154, which took around two minutes. The model was then run on the new testing set, and the cross-validation was around two minutes as well. The F1 score, the precision score and recall score were all around 0.955. This process was run all over once more, but the data was scaled using Python's StandardScaler. The model took around two minutes to run, and the F1 score, precision score, and recall score all took around 0.978.

Based on the submission scores, it appears the original random forest classifier model and the final PCA random forest classifier model are the top models. These submission scores are reflective of the F1, precision, and recall scores obtained through Python's scoring metrics.

Submissions:

10 submissions for Nadeem Patel — Sort by: Select...

All | Successful | Selected

Submission and Description	Public Score
pca_rf_3_submission.csv just now by Nadeem Patel pca random forest/fixed flaw and scaled (step 8)	0.96675
pca_rf_2_submission.csv a minute ago by Nadeem Patel pca random forest/fixed flaw (step 8)	0.94535
km_submission.csv 2 minutes ago by Nadeem Patel k-means (step 8)	0.51939
pca_rf_1_submission.csv 4 minutes ago by Nadeem Patel pca random forest (step 6)	0.94450
rf_submission.csv 7 minutes ago by Nadeem Patel random forest (step 2)	0.96750

```
# pca with principal components that represent 95 percent of the variability in the explanatory variables
start=datetime.now()

pca = PCA(n_components=0.95, random_state=42)
pca_clf = pca.fit_transform(combined)

end=datetime.now()

print(end-start)
```

```
0:00:14.654709
```

Index:

```
import pandas as pd
import numpy as np
import time
import os.path
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import urllib.request as ur
from datetime import datetime
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score,\
                            classification_report, confusion_matrix
from sklearn.model_selection import cross_val_predict
from sklearn.cluster import KMeans
import itertools
import seaborn as sns
from matplotlib import pyplot as plt

import matplotlib
```

```
# read training and testing datasets
train = pd.read_csv('digit-recognizer/train.csv')
test = pd.read_csv('digit-recognizer/test.csv')
```

```
train.head()
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel781	pixe
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	

5 rows × 785 columns

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42000 entries, 0 to 41999
Columns: 785 entries, label to pixel783
dtypes: int64(785)
memory usage: 251.5 MB
```

```
train.describe()
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel77
count	42000.000000	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	42000.0	...	42000.000000	42000.000000	42000.000000	42000.0000
mean	4.456643	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.219286	0.117095	0.059024	0.0201
std	2.887730	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	6.312890	4.633819	3.274488	1.7598
min	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0.0000
25%	2.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0.0000
50%	4.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0.0000
75%	7.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.000000	0.000000	0.000000	0.0000
max	9.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	254.000000	254.000000	253.000000	253.0000

8 rows × 785 columns

```
train.isna().any().any()
```

```
False
```

```
print(train.shape)
print(test.shape)

df_X = pd.concat([train, test])
#df_Y = df_X['label']
df_X.drop('label', axis=1, inplace=True)
df_X = df_X.values.reshape(len(df_X), -1)

print(df_X.shape)
```

```
(42000, 785)
(28000, 784)
(70000, 784)
```

```
def plot_digit(data):
    image = data.reshape(28, 28)
    plt.imshow(image, cmap = matplotlib.cm.binary,
               interpolation="nearest")
    plt.axis("off")
```

```
# plot digits
def plot_digits(instances, images_per_row=10, **options):
    size = 28
    images_per_row = min(len(instances), images_per_row)
    images = [instance.reshape(size,size) for instance in instances]
    n_rows = (len(instances) - 1) // images_per_row + 1
    row_images = []
    n_empty = n_rows * images_per_row - len(instances)
    images.append(np.zeros((size, size * n_empty)))
    for row in range(n_rows):
        rimages = images[row * images_per_row : (row + 1) * images_per_row]
        row_images.append(np.concatenate(rimages, axis=1))
    image = np.concatenate(row_images, axis=0)
    plt.imshow(image, cmap = matplotlib.cm.binary, **options)
    plt.axis("off")
```

```
# get digits
plt.figure(figsize=(9,9))
example_images = np.r_[df_X[:12000:600], df_X[13000:30600:600], df_X[30600:60000:590]]
plot_digits(example_images, images_per_row=10)
#save_fig("more_digits_plot")
plt.show()
```

```
# features from training set
train_copy = train.copy()
train_copy.drop('label', axis=1, inplace=True)

X_train = train_copy.copy()
y_train = train['label']

# features shape
print(X_train.shape)
print(y_train.shape)
```

```
(42000, 784)
(42000,)
```

```
# random forest
start=datetime.now()

rf = RandomForestClassifier(n_estimators=784)
rf.fit(X_train, y_train)

end=datetime.now()

print(end-start)
```

```
0:04:10.991683
```

```
# rf cross-validation (X_train, y_train)
start=datetime.now()

rf_y_scores = cross_val_predict(rf, X_train, y_train)

end=datetime.now()

print(end-start)
print(rf_y_scores.shape)
```

```
0:21:14.157588
(42000,)
```

```
# scores from (y_train, rf_y_scores)
print("F1 score:", f1_score(y_train, rf_y_scores, average="macro"))
print("Precision score:", precision_score(y_train, rf_y_scores, average="macro"))
print("Recall score:", recall_score(y_train, rf_y_scores, average="macro"))
```

```
F1 score: 0.966233647278985
Precision score: 0.9662408480535699
Recall score: 0.9662462772130491
```

```
# run model on test dataset
rf_predictions = rf.predict(test)
rf_predictions.reshape(-1,1)
```

```
array([[2],
       [0],
       [9],
       ...,
       [3],
       [9],
       [2]])
```

```
print(rf_predictions.shape)
print(np.arange(1,28001).shape)
```

```
(28000,)
(28000,)
```

```
# rf cross-validation (test, rf_predictions)
start=datetime.now()

rf_test_scores = cross_val_predict(rf, test, rf_predictions)

end=datetime.now()

print(end-start)
print(rf_test_scores.shape)
```

```
0:09:38.626171
(28000,)
```

```
# scores from (rf_predictions, rf_test_scores)
print("F1 score:", f1_score(rf_predictions, rf_test_scores, average="macro"))
print("Precision score:", precision_score(rf_predictions, rf_test_scores, average="macro"))
print("Recall score:", recall_score(rf_predictions, rf_test_scores, average="macro"))
```

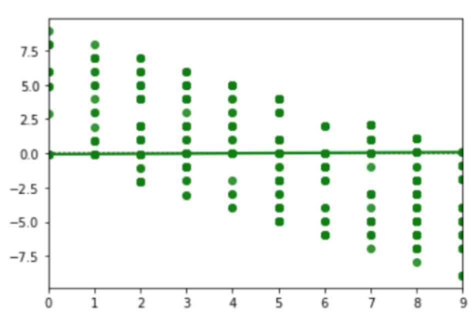
```
F1 score: 0.9813987684341976
Precision score: 0.9814595597978851
Recall score: 0.9813575962246658
```

```
# rf matrix
sns.residplot(rf_predictions, rf_test_scores, lowess=True, color="g")

conf_mx1 = confusion_matrix(rf_predictions, rf_test_scores)
conf_mx1
```

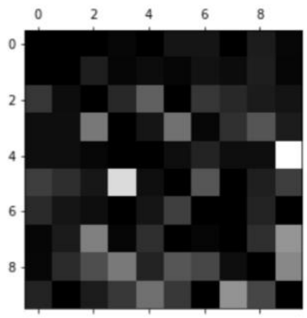
```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

```
array([[2769,    0,    0,    1,    0,    3,    3,    0,    4,    1],
       [   0, 3174,    5,    1,    2,    1,    3,    2,    5,    1],
       [   8,    2, 2794,    6,   14,    0,    8,    6,    4,    3],
       [   2,    2,   17, 2697,    3,   16,    1,    7,   12,    4],
       [   2,    2,    1,    0, 2702,    2,    5,    2,    2,   36],
       [   8,    6,    3,   28,    2, 2433,   11,    0,    4,    8],
       [   6,    3,    2,    0,    3,    9, 2734,    0,    5,    0],
       [   1,    4,   19,    1,    7,    0,    1, 2825,    7,   22],
       [   1,    6,   11,   17,    5,   12,   10,    2, 2644,   19],
       [   5,    0,    4,    8,   16,    8,    0,   21,   10, 2714]])
```



```
# get rf matrix
def plot_confusion_matrix(matrix):
    fig = plt.figure(figsize=(8,8))
    ax = fig.add_subplot(111)
    cax = ax.matshow(matrix)
    fig.colorbar(cax)

row_sums = conf_mx1.sum(axis=1, keepdims=True)
norm_conf_mx = conf_mx1 / row_sums
np.fill_diagonal(norm_conf_mx, 0)
plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
plt.show()
```




```
rf_d = {'ImageId': np.arange(1,28001), 'Label': rf_predictions}
rf_df = pd.DataFrame(data=rf_d)
rf_df.to_csv('rf_submission.csv', index=False)
```

```
rf_df.head()
```

	ImageId	Label
0	1	2
1	2	0
2	3	9
3	4	9
4	5	3

```
# combine features from training dataset (X_train) and testing dataset
combined = pd.concat([X_train, test])
```

```
# pca with principal components that represent 95 percent of the variability in the explanatory variables
start=datetime.now()

pca = PCA(n_components=0.95, random_state=42)
pca_clf = pca.fit_transform(combined)

end=datetime.now()

print(end-start)
```

```
0:00:14.654709
```

```
print(combined.shape)
print(pca.n_components_)
```

```
(70000, 784)
154
```

```
# get shapes of datasets before creating new datasets
print("Training dataset:", X_train.shape)
print("Testing dataset:", test.shape)
print("Combined dataset:", combined.shape)

# adjusted X_train
pca_X_train = pca_clf[:42000]
print("PCA training dataset:", pca_X_train.shape)
# adjusted X_test
pca_X_test = pca_clf[42000:]
print("PCA testing dataset:", pca_X_test.shape)
```

```
Training dataset: (42000, 784)
Testing dataset: (28000, 784)
Combined dataset: (70000, 784)
PCA training dataset: (42000, 154)
PCA testing dataset: (28000, 154)
```

```
# pca_rf on reduced explantory variables
start=datetime.now()

rf_2 = RandomForestClassifier(n_estimators=154)
rf_2.fit(pca_X_train, y_train)

end=datetime.now()
print(end-start)
```

```
0:01:50.922937
```

```
# rf_2 cross-validation (adjusted X_train, adjusted y_train)
start=datetime.now()

rf_2_y_scores = cross_val_predict(rf_2, pca_X_train, y_train)

end=datetime.now()

print(end-start)
print(rf_2_y_scores.shape)
```

```
0:07:26.959221
(42000,)
```

```
# scores from (rf_2_y_predictions, rf_test_scores)
print("F1 score:", f1_score(y_train, rf_2_y_scores, average="macro"))
print("Precision score:", precision_score(y_train, rf_2_y_scores, average="macro"))
print("Recall score:", recall_score(y_train, rf_2_y_scores, average="macro"))
```

```
F1 score: 0.9422069154496159
Precision score: 0.942299922711585
Recall score: 0.942210924638396
```

```
# run rf_2 model on adjusted test set
rf_predictions_2 = rf_2.predict(pca_X_test)
print(rf_predictions_2.shape)
print(np.arange(1,28001).shape)
```

```
(28000,)
(28000,)
```

```
# rf_2 cross-validation (test, rf_predictions_2)
start=datetime.now()

rf_2_test_scores = cross_val_predict(rf_2, test, rf_predictions_2)

end=datetime.now()

print(end-start)
print(rf_2_test_scores.shape)
```

```
0:02:17.822737
(28000,)
```

```
# scores from (rf_predictions_2, rf_2_test_scores)
print("F1 score:", f1_score(rf_predictions_2, rf_2_test_scores, average="macro"))
print("Precision score:", precision_score(rf_predictions_2, rf_2_test_scores, average="macro"))
print("Recall score:", recall_score(rf_predictions_2, rf_2_test_scores, average="macro"))
```

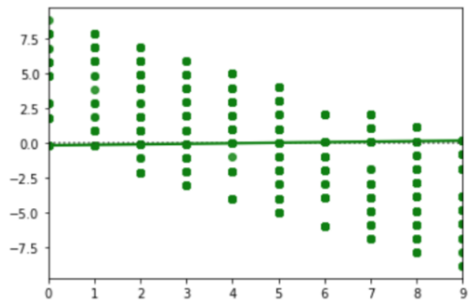
```
F1 score: 0.957433957080047
Precision score: 0.9574607334373001
Recall score: 0.9574516826095065
```

```
# rf_2 matrix
sns.residplot(rf_predictions_2, rf_2_test_scores, lowess=True, color="g")

conf_mx2 = confusion_matrix(rf_predictions_2, rf_2_test_scores)
conf_mx2
```

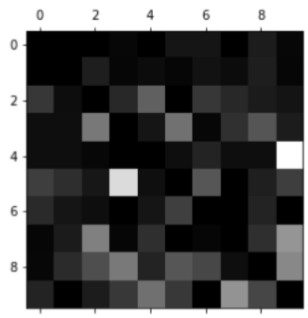
```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

```
array([[2730,    0,    8,    5,    0,    9,   10,    2,   12,    1],
       [   0, 3163,    8,    2,    4,    1,    2,    9,    8,    7],
       [  17,    3, 2695,   20,   22,    3,   22,   11,   23,    5],
       [   8,    8,   37, 2689,    8,   40,    5,   13,   56,    5],
       [   7,    0,   10,    1, 2663,   12,    6,   21,    6,   81],
       [  11,    9,    5,   49,    8, 2344,   22,    5,   32,   10],
       [  13,    0,   11,    4,   16,   13, 2702,    0,   10,    0],
       [   6,    7,   19,    9,   23,    3,    0, 2749,    7,   55],
       [   6,    8,   28,   41,   15,   21,   10,    5, 2509,   20],
       [   4,    2,    5,    9,   51,   15,    0,   34,   15, 2582]])
```



```
# get rf_2 matrix
def plot_confusion_matrix(matrix):
    fig = plt.figure(figsize=(8,8))
    ax = fig.add_subplot(111)
    cax = ax.matshow(matrix)
    fig.colorbar(cax)

row_sums = conf_mx1.sum(axis=1, keepdims=True)
norm_conf_mx = conf_mx1 / row_sums
np.fill_diagonal(norm_conf_mx, 0)
plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
plt.show()
```




```
rf_d_2 = {'ImageId': np.arange(1,28001), 'Label': rf_predictions_2}
rf_df_2 = pd.DataFrame(data=rf_d_2)
rf_df_2.to_csv('pca_rf_1_submission.csv', index=False)
rf_df_2.head()
```

	ImageId	Label
0	1	2
1	2	0
2	3	9
3	4	9
4	5	3

```
# kmeans

# pre-process images
X = X_train.values.reshape(len(X_train), -1)
Y = y_train

X = X.astype(float) / 255.

print(Y.shape)
print(X.shape)
print(X[0].shape)
```

```
(42000,)
(42000, 784)
(784,)
```

```
# group MNIST observations into 1 of 10 categories and assign labels
n_digits = len(np.unique(Y))
print(n_digits)

kmeans = KMeans(n_digits)
kmeans.fit(X)
kmeans.labels_
```

```
10

array([8, 3, 2, ..., 0, 1, 5], dtype=int32)
```

```python
# kmeans clustering
def infer_cluster_labels(kmeans, actual_labels):
    inferred_labels = {}

    for i in range(kmeans.n_clusters):
        # find index of points in cluster
        labels = []
        index = np.where(kmeans.labels_ == i)

        # append actual labels for each point in cluster
        labels.append(actual_labels[index])

        # determine most common label
        if len(labels[0]) == 1:
            counts = np.bincount(labels[0])
        else:
            counts = np.bincount(np.squeeze(labels))

        # assign the cluster to a value in the inferred_labels dictionary
        if np.argmax(counts) in inferred_labels:
            # append the new number to the existing array at this slot
            inferred_labels[np.argmax(counts)].append(i)
        else:
            # create a new array in this slot
            inferred_labels[np.argmax(counts)] = [i]

        #print(labels)
        #print('Cluster: {}, label: {}'.format(i, np.argmax(counts)))

    return inferred_labels

def infer_data_labels(X_labels, cluster_labels):
    # empty array of len(X)

    predicted_labels = np.zeros(len(X_labels)).astype(np.uint8)

    for i, cluster in enumerate(X_labels):
        for key, value in cluster_labels.items():
            if cluster in value:
                predicted_labels[i] = key

    return predicted_labels
```

```
# get cluster labels and predicted labels
cluster_labels=infer_cluster_labels(kmeans, Y.values)
X_clusters=kmeans.predict(test) #predict cluster labels/ can also use kmeans.labels_ as well
predicted_labels=infer_data_labels(X_clusters, cluster_labels)

print(X_clusters,X_clusters.shape)
print(cluster_labels)
print(predicted_labels[:20])
print(Y[:20])
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/sklearn/base.py:443: UserWarning: X has feature names, but KMeans was fitted without feature names
  warnings.warn(
```

```
[6 3 7 ... 9 4 4] (28000,)
{7: [0], 6: [1], 1: [2, 8], 0: [3, 4], 4: [5], 2: [6], 8: [7], 3: [9]}
[2 0 8 0 2 7 0 0 0 3 3 2 8 0 4 0 0 1 8 0]
0     1
1     0
2     1
3     4
4     0
5     0
6     7
7     3
8     5
9     3
10    8
11    9
12    1
```

```
#km_predictions = km.predict(test)
print(predicted_labels.shape)
print(np.arange(1,28001).shape)
```

```
(28000,)
(28000,)
```

```
# km cross-validation (test, predictions_labels)
start=datetime.now()

km_test_scores = cross_val_predict(kmeans, test, predicted_labels)

end=datetime.now()

print(end-start)
print(km_test_scores.shape)
```

```
0:01:25.071868
(28000,)
```

```
# scores from (rf_predictions, rf_test_scores)
print("F1 score:", f1_score(predicted_labels, km_test_scores, average="macro"))
print("Precision score:", precision_score(predicted_labels, km_test_scores, average="macro"))
print("Recall score:", recall_score(predicted_labels, km_test_scores, average="macro"))
```

```
F1 score: 0.0642343481674747
Precision score: 0.0801996585277825
Recall score: 0.06429223646638986
```

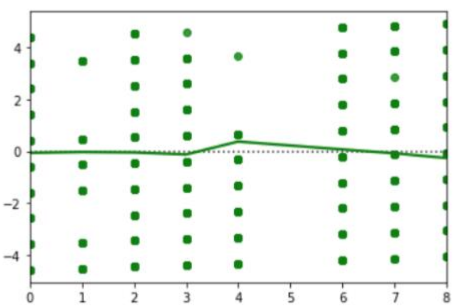
```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/sklearn/metrics/_classification.py:131
8: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_divis
ion` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
# km matrix
sns.residplot(predicted_labels, km_test_scores, lowess=True, color="g")

conf_mx3 = confusion_matrix(predicted_labels, km_test_scores)
conf_mx3
```

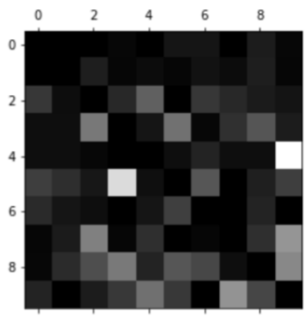
```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/seaborn/_decorators.py:36: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

```
array([[ 357,  568, 1112, 1495,  656, 1149,  593, 1442,  606,  635],
       [ 461,  525,    0,  289,  288,  242,    0,    0,  755,    0],
       [ 559,  134,   66,   63,  469,  111, 1095,   81,  177,   40],
       [  17,   63,   32,   49,  459,   14,  407, 1291,   33,    1],
       [ 199,  386,  368,  366,  356,  171,    0,    0,    1,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [  54,   62,   81,   39,  371,   79,    8,  123,   17, 1104],
       [   6, 1067,   32,   38,   44,  694,  499,    1,   16,  343],
       [ 587,  720, 1418,  279,  348,  468,  181,  254,  796,   90],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0]])
```



```python
# get km matrix
def plot_confusion_matrix(matrix):
    fig = plt.figure(figsize=(8,8))
    ax = fig.add_subplot(111)
    cax = ax.matshow(matrix)
    fig.colorbar(cax)

row_sums = conf_mx1.sum(axis=1, keepdims=True)
norm_conf_mx = conf_mx1 / row_sums
np.fill_diagonal(norm_conf_mx, 0)
plt.matshow(norm_conf_mx, cmap=plt.cm.gray)
plt.show()
```




```python
km_d = {'ImageId': np.arange(1,28001), 'Label': predicted_labels}
km_df = pd.DataFrame(data=km_d)
km_df.to_csv('km_submission.csv', index=False)
km_df.head()
```

	ImageId	Label
0	1	2
1	2	0
2	3	8
3	4	0
4	5	2

```
# PCA performed on training and testing data sets separately
pca_X_train = pca.fit_transform(X_train)
pca_X_test = pca.transform(test)

print(pca_X_train.shape)
print(pca_X_test.shape)
```

```
(42000, 154)
(28000, 154)
```

```
# pca_rf_2 on reduced explantory variables
start=datetime.now()

rf_3 = RandomForestClassifier(n_estimators=154)
rf_3.fit(pca_X_train, y_train)

end=datetime.now()

print(end-start)
```

```
0:02:10.602845
```

```
# run rf_3 model on adjusted test set
rf_predictions_3 = rf_3.predict(pca_X_test)
print(rf_predictions_3.shape)
print(np.arange(1,28001).shape)
```

```
(28000,)
(28000,)
```

```
# rf_3 cross-validation (test, rf_predictions_3)
start=datetime.now()

rf_3_test_scores = cross_val_predict(rf_3, test, rf_predictions_3)

end=datetime.now()

print(end-start)
print(rf_3_test_scores.shape)
```

```
0:02:13.346268
(28000,)
```

```
# scores from (rf_predictions_3, rf_3_test_scores)
print("F1 score:", f1_score(rf_predictions_3, rf_test_scores, average="macro"))
print("Precision score:", precision_score(rf_predictions_3, rf_test_scores, average="macro"))
print("Recall score:", recall_score(rf_predictions_3, rf_test_scores, average="macro"))
```

```
F1 score: 0.9552142256267839
Precision score: 0.9552821488299312
Recall score: 0.9552741551700075
```

```
rf_d_3 = {'ImageId': np.arange(1,28001), 'Label': rf_predictions_3}
rf_df_3 = pd.DataFrame(data=rf_d_3)
rf_df_3.to_csv('pca_rf_2_submission.csv', index=False)
rf_df_3.head()
```

	ImageId	Label
0	1	2
1	2	0
2	3	9
3	4	4
4	5	3

```
# PCA performed on training and testing data sets separately (scaled)
sc = StandardScaler()
pca_X_train = sc.fit_transform(X_train)
pca_X_test = sc.transform(test)

print(pca_X_train.shape)
print(pca_X_test.shape)
```

```
(42000, 784)
(28000, 784)
```

```
pca_X_train = pca.fit_transform(pca_X_train)
pca_X_test = pca.transform(pca_X_test)

print(pca_X_train.shape)
print(pca_X_test.shape)
```

```
(42000, 232)
(28000, 232)
```

```
# rf_4 on reduced explantory variables
start=datetime.now()

rf_4 = RandomForestClassifier(n_estimators=154)
rf_4.fit(pca_X_train, y_train)

end=datetime.now()

print(end-start)
```

```
0:00:55.457546
```

```
# run rf_4 model on adjusted test set
rf_predictions_4 = rf_4.predict(pca_X_test)
print(rf_predictions_4.shape)
print(np.arange(1,28001).shape)
```

```
(28000,)
(28000,)
```

```
# rf_4 cross-validation (test, rf_predictions_4)
start=datetime.now()

rf_4_test_scores = cross_val_predict(rf_4, test, rf_predictions_4)

end=datetime.now()

print(end-start)
print(rf_4_test_scores.shape)
```

```
0:02:02.822960
(28000,)
```

```
rf_d_4 = {'ImageId': np.arange(1,28001), 'Label': rf_predictions_4}
rf_df_4 = pd.DataFrame(data=rf_d_4)
rf_df_4.to_csv('pca_rf_3_submission.csv', index=False)
rf_df_4.head()
```

	ImageId	Label
0	1	2
1	2	0
2	3	9
3	4	9
4	5	3