Background on Data

The data provided for the purpose of this assignment came from the Kannada MNIST dataset, provide by Kaggle for competition purposes. It is a dataset comprised of images of number for which the goal is to be able to accurately classify them. This is a large dataset comprised of 784 features to be used, with each feature containing 6,000 instances.

Data Preparation

For the purpose of this assignment, there were specific data preparation steps that we were required to follow throughout the assignment. Initially, I was directed to use first use all 784 features as a means of prediction in creating a Random Forest Classifier (RFC). This RFC was then fitted to the test data provided, while recording the time it took for the algorithm to run. Following this step, I was tasked with executing Principal Component Analysis (PCA) on the combined training and test data sets together. This step would generate principal components that represented 95% of the variability in the explanatory variables. With these new principal components, another RFC was created. In this assignment, there was a major flaw in that the test data was used and manipulated in the training process only to then be used again for prediction after the RFC was fitted. This is a cardinal sin in data science, hence, I went back and conducted PCA on solely the training data. The newfound principal components were then used to compose a third and final RFC.

Results/Evaluation

The results pertaining to each individual RFC were interesting as there were 2 models that contained principal components and that did not. For the RFC without PCA data, Kaggle scored this model to be the highest (0.9206) while also being the fastest model to process (41.1s). PCA was then enacted on the combined training and test data. The number of principal components found that represented 95% of the variability in the features amount to 239 features. These were then fitted to a RFC, generating a Kaggle score of (0.89280) and doing so with a time of 2 minutes and 27 seconds. After revising this mistake in the data, I went back and conducted PCA analysis based on the training data alone, which resulted in 237 principal components. This was then fitted to another RFC, which garnered a Kaggle score of (0.89240) and was computed in a time of 2 minutes and 30 seconds.

Model Performance and Recommendation

From a performance standpoint, the classifier that performed best was the first RFC that was fitted on the training data alone without principal components. The second classifier cannot be used as the test data was used for training and was incorporated in the in Principal Component Analysis. This then leaves the third classifier that was created, in which the model took more time to run and was proven to be less accurate. From a management perspective, the first RFC would be optimal, as it took less time to run and was more accurate. In general, PCA will not hep to make a model perform better from an accuracy standpoint. This is because PCA is an algorithm that does not consider the response variable and prediction target in its analysis. PCA treats features with large variance favorably, but a feature with large variance might have very little to do with the desired predicted feature. As it pertains to the use of PCA, it should be used for datasets that are noisy and have and even larger number of features than what was worked with for this assignment. Instances such as image compression, rather than digit recognition, and speech recognition would be appropriate times to use. PCA.

Appendix

In [1]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

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

In [2]:
```python
import sklearn
import seaborn as sns
import time
import pandas as pd
import numpy as np
import os
from sklearn.manifold import LocallyLinearEmbedding
from sklearn.mixture import GaussianMixture
from scipy import stats
from sklearn.cluster import MiniBatchKMeans
from sklearn.metrics import silhouette_score
from sklearn.metrics import silhouette_samples
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from IPython.core.interactiveshell import InteractiveShell
from sklearn.linear_model import RidgeCV, ElasticNetCV, LassoCV
from sklearn.decomposition import PCA #Principal Component Analysis
from sklearn.decomposition import IncrementalPCA
from sklearn.decomposition import KernelPCA
from sklearn.preprocessing import LabelEncoder, StandardScaler #transforms categorical into
 numbers
from sklearn.model_selection import KFold
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import mean_squared_error, make_scorer
import xgboost
from sklearn.cluster import KMeans


%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.ticker import FixedLocator, FixedFormatter
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
```

In [3]:
```python
MNIST_train = pd.read_csv('train.csv')
MNIST_test = pd.read_csv('test.csv')
```

In [4]:
```python
MNIST_train.head()
```

Out[4]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pixel13	pixel14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

In [5]:
```python
print(MNIST_train.shape)
print(MNIST_test.shape)
```

```
(60000, 785)
(5000, 785)
```

In [6]:
```python
MNIST_test.head()
```

Out[6]:

	id	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pixel13	pixel14	p
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

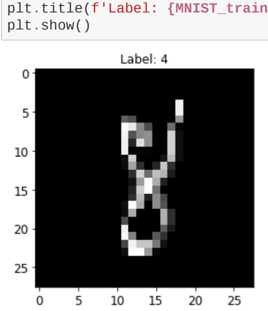
5 rows × 785 columns

In [7]:
```python
MNIST_train.label.value_counts()
```

Out[7]:
```
9    6000
8    6000
7    6000
6    6000
5    6000
4    6000
3    6000
2    6000
1    6000
0    6000
Name: label, dtype: int64
```

All labels have equal value

In [8]:
```python
num = 4
plot_num = MNIST_train.iloc[num, 1:]
plot_num = np.array(plot_num).reshape(28, -1)
plt.imshow(plot_num, cmap='gray')
plt.title(f'Label: {MNIST_train.iloc[num, 0]}')
plt.show()
```



In [9]:
```python
X_train, y_train = MNIST_train.drop('label', axis=1), MNIST_train['label']
```

RANdOM FOREST STEP 1

In [10]:
```python
rf= RandomForestClassifier()
%time rf.fit(X_train, y_train)
```

```
CPU times: user 41.5 s, sys: 108 ms, total: 41.6 s
Wall time: 41.5 s
```

Out[10]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [11]:
```python
X_test=MNIST_test.drop('id', axis=1)

y_pred= rf.predict(X_test)
```

In [12]:
```python
y_pred
```

Out[12]:
```
array([3, 0, 2, ..., 1, 6, 3])
```

In [13]:
```python
Identification = MNIST_test['id'].copy()
RF1 = pd.DataFrame(y_pred)
RF1.columns= ['label']
RandForest= pd.concat([Identification, RF1], axis= 1)
RandForest
```

Out[13]:

	id	label
0	0	3
1	1	0
2	2	2
3	3	6
4	4	7
...	...	...
4995	4995	1
4996	4996	1
4997	4997	1
4998	4998	6
4999	4999	3

5000 rows × 2 columns

PCA ANALYSIS

In [14]:
```python
X = np.concatenate((X_train, X_test), axis=0)
```

In [15]:
```python
X.shape
```

Out[15]:
```
(65000, 784)
```

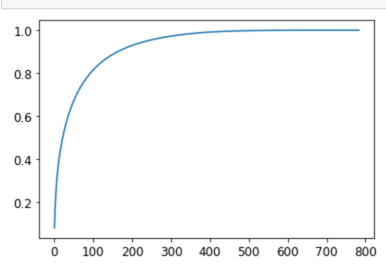
In [16]:
```python
pca= PCA()
%time pca.fit(X)
```

```
CPU times: user 23.5 s, sys: 1.5 s, total: 25 s
Wall time: 13.3 s
```

Out[16]:
```
PCA(copy=True, iterated_power='auto', n_components=None, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

In [17]:
```python
total_var= np.cumsum(pca.explained_variance_ratio_)
```

In [18]:
```python
plt.plot(total_var)
plt.show()
```



In [19]:
```python
idx_95 = np.where(total_var >= 0.95)[0][0]
print(idx_95)
print(total_var[idx_95])
```

```
238
0.9501355396380707
```

Exactly 239 dimensions are enough to contain 95% of the variance (Pythong starts indexing at 0, so it would be 239)

In [20]:
```python
X_pca = pca.transform(X_train)
```

In [21]:
```python
X_pca[:,:idx_95+1].shape
```

Out[21]:
```
(60000, 239)
```

In [22]:
```python
rf_pca= RandomForestClassifier()
%time rf_pca.fit(X_pca[:,:idx_95+1], y_train)
```

```
CPU times: user 2min 26s, sys: 224 ms, total: 2min 26s
Wall time: 2min 26s
```

Out[22]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [23]:
```python
X_test_pca = pca.transform(X_test)[:,:idx_95+1]
```

In [24]:
```python
y_pred_pca = rf_pca.predict(X_test_pca)
```

In [25]:
```python
y_pred_pca
```

Out[25]:
```
array([3, 0, 2, ..., 1, 6, 3])
```

In [26]:
```python
RF_PCA = pd.DataFrame(y_pred_pca)
RF_PCA.columns= ['label']
RandForest_PCA= pd.concat([Identification, RF_PCA], axis= 1)
RandForest_PCA
```

Out[26]:

	id	label
0	0	3
1	1	0
2	2	2
3	3	6
4	4	7
...	...	...
4995	4995	1
4996	4996	1
4997	4997	1
4998	4998	6
4999	4999	3

5000 rows × 2 columns

Observing Experiment Error

Flaw is that we were fitting pca to our test data

In [27]:
```python
MNIST_train.describe()
```

Out[27]:

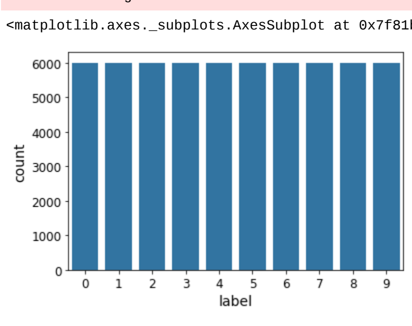
	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	
count	60000.000000	60000.0	60000.0	60000.0	60000.0	60000.0	60000.000000	60000.000000	60000.000000	60000.000000	60
mean	4.500000	0.0	0.0	0.0	0.0	0.0	0.008817	0.029467	0.037767	0.075933	
std	2.872305	0.0	0.0	0.0	0.0	0.0	1.474271	2.700491	2.726371	3.993023	
min	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	
25%	2.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	
50%	4.500000	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	
75%	7.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	
max	9.000000	0.0	0.0	0.0	0.0	0.0	255.000000	255.000000	255.000000	255.000000	

8 rows × 785 columns

In [28]:
```python
sns.countplot(y_train, color = sns.color_palette()[0])
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the fol
lowing variable as a keyword arg: x. From version 0.12, the only valid positional argument wi
ll be `data`, and passing other arguments without an explicit keyword will result in an error
or misinterpretation.
  FutureWarning
```

Out[28]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f81b9062410>
```



In [29]:
```python
pca_2 = PCA().fit(X_train)
```

In [30]:
```python
total_var2= np.cumsum(pca_2.explained_variance_ratio_)
idx_95_2 = np.where(total_var2 >= 0.95)[0][0]
idx_95_2
```

Out[30]:
```
236
```

In [31]:
```python
X_pca = pca_2.transform(X_train)
```

In [32]:
```python
rf_pca_2= RandomForestClassifier()
%time rf_pca_2.fit(X_pca[:,:idx_95_2+1], y_train)
```

```
CPU times: user 2min 25s, sys: 223 ms, total: 2min 25s
Wall time: 2min 25s
```

Out[32]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [33]:
```python
X_test_pca_2= pca_2.transform(X_test)[:,:idx_95_2+1]
```

In [34]:
```python
y_pred_pca_2 = rf_pca_2.predict(X_test_pca_2)
```

In [35]:
```python
RF_PCA_2 = pd.DataFrame(y_pred_pca_2)
RF_PCA_2.columns= ['label']
RandForest_PCA_2= pd.concat([Identification, RF_PCA_2], axis= 1)
RandForest_PCA_2
```

Out[35]:

	id	label
0	0	3
1	1	0
2	2	2
3	3	6
4	4	7
...	...	...
4995	4995	1
4996	4996	1
4997	4997	1
4998	4998	6
4999	4999	3

5000 rows × 2 columns

In [36]:
```python
pd.DataFrame(RandForest).to_csv('Flawed RandomForest.csv')
pd.DataFrame(RandForest_PCA).to_csv('Flawed RandomForest with PCA.csv')
pd.DataFrame(RandForest_PCA_2).to_csv('RandomForest Correct.csv')
```

K-MEANS

In [37]:
```python
kmeans = KMeans(n_clusters=10).fit(X_pca[:,:idx_95_2+1])
kmeans.labels_
kmeans.predict(X_test_pca_2)
kmeans.cluster_centers_
```

Out[37]:
```
array([[ 1.85863246e+01,  9.83115172e+02,  4.23168630e+01, ...,
        -1.82471233e+00, -5.29391308e-01,  1.07458844e+00],
       [ 3.68586848e+02,  1.11395032e+02, -2.73076998e+02, ...,
         2.13045086e-05, -6.75928707e-01,  1.50538872e+00],
       [-2.35414866e+02,  5.61218314e+02,  4.29137430e+02, ...,
         3.86847420e-01,  1.45035275e+00, -2.19873638e+00],
       ...,
       [-4.45641401e+02, -1.50601407e+02, -1.92509419e+02, ...,
         1.56244741e-01, -5.70661884e-01, -1.43337123e-01],
       [ 3.12571033e+02, -1.68120453e+02,  5.77007727e+01, ...,
         5.16126318e-01, -3.74553491e-01, -6.38474254e-01],
       [-2.85416199e+01, -7.83296669e+00, -4.13586962e+02, ...,
         4.04423006e-02,  4.06256676e-01,  1.58493550e-01]])
```

In [48]:
```python
y_kmeans_predict = kmeans.predict(X_test_pca_2)
print(y_kmeans_predict)
```

```
[7 6 0 ... 8 3 7]
```

In [55]:
```python
y_kmeans_match = kmeans.predict(X_pca[:,:idx_95_2+1])
match= []
for i in range(10):
  match.append(np.round(np.mean(y_kmeans_match[np.where(y_train==i)[0]])))

match
```

Out[55]:
```
[6.0, 8.0, 1.0, 7.0, 9.0, 4.0, 4.0, 6.0, 5.0, 2.0]
```

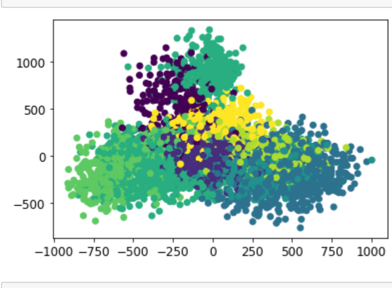
In [56]:
```python
for i in range(len(y_kmeans_predict)):
    y_kmeans_predict[i] =int(match[y_kmeans_predict[i]])

y_kmeans_predict
```

Out[56]:
```
array([6, 4, 6, ..., 5, 7, 6], dtype=int32)
```

In [49]:
```python
X_test_pca_2.shape
```

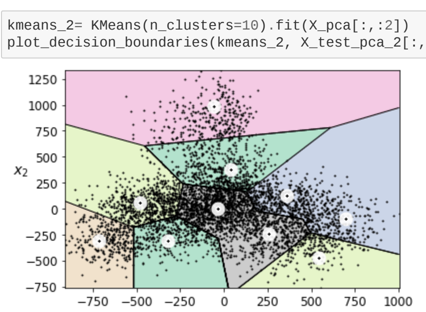
Out[49]:
```
(5000, 237)
```

In [57]:
```python
plt.scatter(X_test_pca_2[:,0], X_test_pca_2[:,1], c=y_kmeans_predict)
plt.show()
```



In [58]:
```python
def plot_data(X):
    plt.plot(X[:, 0], X[:, 1], 'k.', markersize=2)

def plot_centroids(centroids, weights=None, circle_color='w', cross_color='k'):
    if weights is not None:
        centroids = centroids[weights > weights.max() / 10]
    plt.scatter(centroids[:, 0], centroids[:, 1],
                marker='o', s=35, linewidths=8,
                color=circle_color, zorder=10, alpha=0.9)
    plt.scatter(centroids[:, 0], centroids[:, 1],
                marker='x', s=2, linewidths=12,
                color=cross_color, zorder=11, alpha=1)

def plot_decision_boundaries(clusterer, X, resolution=1000, show_centroids=True,
                             show_xlabels=True, show_ylabels=True):
    mins = X.min(axis=0) - 0.1
    maxs = X.max(axis=0) + 0.1
    xx, yy = np.meshgrid(np.linspace(mins[0], maxs[0], resolution),
                         np.linspace(mins[1], maxs[1], resolution))
    Z = clusterer.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(Z, extent=(mins[0], maxs[0], mins[1], maxs[1]),
                cmap="Pastel2")
    plt.contour(Z, extent=(mins[0], maxs[0], mins[1], maxs[1]),
                linewidths=1, colors='k')
    plot_data(X)
    if show_centroids:
        plot_centroids(clusterer.cluster_centers_)

    if show_xlabels:
        plt.xlabel("$x_1$", fontsize=14)
    else:
        plt.tick_params(labelbottom=False)
    if show_ylabels:
        plt.ylabel("$x_2$", fontsize=14, rotation=0)
    else:
        plt.tick_params(labelleft=False)
```

In [59]:
```python
kmeans_2= KMeans(n_clusters=10).fit(X_pca[:,:2])
plot_decision_boundaries(kmeans_2, X_test_pca_2[:,:2])
```



Account (User ID 5212331)

User Name

michaelvenit

Your username cannot be changed.

Submission and Description	Status	Private Score	Public Score	Use for Final Score
Kannada MNIST (version 8/8) a day ago by Michael Venit Revised PCA	Succeeded	0.90020	0.89240	☐
Kannada MNIST (version 7/8) a day ago by Michael Venit RF_PCA	Succeeded	0.89720	0.89280	☐
Kannada MNIST (version 6/8) a day ago by Michael Venit RF	Succeeded	0.92620	0.92060	☐