MSDS 422 Group 6
Aaron Chen, Nikhil Prabhu, Elvis Matos, Reese Mayer

Assignment 6: Digit Recognizer

Introduction:

In this paper, we explore the effectiveness of Random Forest and Principal Component Analysis (PCA) in predictive modeling. Initially, the dataset is divided into training and testing subsets to train and evaluate the models. A Random forest classifier is applied to the training data, achieving impressive accuracy and predicting labels for testing data. Subsequently, we employ PCA to reduce the dimensionality of the dataset while retaining its critical features. Using these reduced components, a second Random Forest classifier is constructed and evaluated, achieving a similar accuracy score with a longer processing time. However, a significant flaw is discovered: The mixing of the training and testing datasets, potentially leading to overfitting and biased model performance estimates. This highlights the importance of maintaining dataset integrity in predictive modeling, stressing the need for rigorous data separation to ensure unbiased model evaluation.

Random Forest:

The data was split into training and testing using the train_test_split function from scikit-learn, with 20% of the data reserved for testing. Then, a random forest classifier is created and trained on the training data. The goal is to use the model to predict the labels for the testing data, and the accuracy of the predictions is evaluated using accuracy_score. The output indicates that the model fitting took approximately 23.88 seconds, while the evaluation took only 0.33 seconds, with an accuracy of 96.29%. This suggests that the random forest classifier model achieved high accuracy on the presented dataset, and both the training and evaluation processes were efficiently conducted within a reasonable time frame.

PCA Component:

In order to create a PCA we first combined both the training and tests datasets. The goal was to generate a combined collection of principal components that reflected the data as a whole and capture the 95% of the variability in the explanatory variables while reducing dimensionality. The process took around 10 seconds to complete and the resulting dataset only contained 454 columns. The was a significantly smaller number of explanatory variables which indicated a success in reducing the dimensionality.

Random Forest PCA:

A second random forest classifier was built and evaluated using principal components identified. It splits the data into training and testing sets, trains the classifier, and evaluates its accuracy using the testing set. The output indicates an accuracy of approximately 94.01 and the time taken for fitting and evaluating the model is around 176.94 seconds. The resulting array represents the predicted labels for the testing data. This process demonstrates using principal components as features for training a classifier, with the Random Forest model yielding high accuracy in predicting the labels. However, The model takes considerable time due to the large number of principal components involved.

Design Flaw:

The experiment was found to combine both training and testing datasets into a single dataframe for model training and evaluation. This flaw likely leads to overfitting, as the model effectively learns from the testing data during training, compromising its ability to generalize to unseen data. Additionally, the new processing time of 165 seconds suggests that the original model's performance may have been overestimated due to its exposure to the testing data during training. To fix this issue and get an accurate assessment of the model's ability to perform well on new, unseen data, it's crucial to train the model using only training dataset and reserve the testing dataset solely for evaluating its performance.

Nikhilprabhu1999

Submission and Description	Public Score
k_means.csv Complete · 15s ago	0.16203
Random_Forrest.csv Complete · 3d ago	0.96264
Random_Forrest_2.csv Complete · 3d ago	0.95721

PCA_Random.csv Score: 0.96264
Complete · 2d ago

Appendix

Digit Recognizer

May 3, 2024

[1]:
```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import time
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import MiniBatchKMeans
from keras.datasets import mnist
```

1 1.

Fit a random forest classifier using the full set of explanatory variables and the model training set (csv).

[2]:
```python
start_load = time.time()

df = pd.read_csv('train.csv')
test = pd.read_csv("test.csv")

end_load = time.time()
load_time = end_load - start_load
print("Time taken to load data:", load_time, "seconds")


X = df.drop('label', axis=1)
y = df['label']
```

```
Time taken to load data: 1.5300960540771484 seconds
```

[3]:
```python
df
```

[3]:
```
       label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
0          1       0       0       0       0       0       0       0       0
1          0       0       0       0       0       0       0       0       0
```

```
2        1     0     0     0     0     0     0     0     0
3        4     0     0     0     0     0     0     0     0
4        0     0     0     0     0     0     0     0     0
…      …    …    …    …    …    …    …    …    …
41995    0     0     0     0     0     0     0     0     0
41996    1     0     0     0     0     0     0     0     0
41997    7     0     0     0     0     0     0     0     0
41998    6     0     0     0     0     0     0     0     0
41999    9     0     0     0     0     0     0     0     0

       pixel8  …  pixel774  pixel775  pixel776  pixel777  pixel778  \
0           0  …         0         0         0         0         0
1           0  …         0         0         0         0         0
2           0  …         0         0         0         0         0
3           0  …         0         0         0         0         0
4           0  …         0         0         0         0         0
…        …  …       …       …       …       …
41995       0  …         0         0         0         0         0
41996       0  …         0         0         0         0         0
41997       0  …         0         0         0         0         0
41998       0  …         0         0         0         0         0
41999       0  …         0         0         0         0         0

       pixel779  pixel780  pixel781  pixel782  pixel783
0             0         0         0         0         0
1             0         0         0         0         0
2             0         0         0         0         0
3             0         0         0         0         0
4             0         0         0         0         0
…          …       …       …       …       …
41995         0         0         0         0         0
41996         0         0         0         0         0
41997         0         0         0         0         0
41998         0         0         0         0         0
41999         0         0         0         0         0

[42000 rows x 785 columns]
```

2 2.

Record the time it takes to fit the model and then evaluate the model on the csvdata by submitting to Kaggle.com. Provide your Kaggle.com score and user ID.

[4]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

start_fit = time.time()
```

```python
clf = RandomForestClassifier(n_estimators=100, random_state=42)

clf.fit(X_train, y_train)
end_fit = time.time()
fit_time = end_fit - start_fit
print("Time taken to fit model:", fit_time, "seconds")
```

```
Time taken to fit model: 23.95928382873535 seconds
```

[5]:
```python
start_eval = time.time()
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
end_eval = time.time()
eval_time = end_eval - start_eval
print("Time taken to evaluate model:", eval_time, "seconds")
print("Accuracy:", accuracy)
```

```
Time taken to evaluate model: 0.19632601737976074 seconds
Accuracy: 0.9628571428571429
```

[6]:
```python
test_pred = clf.predict(test)
test_pred
```

[6]:
```
array([2, 0, 9, …, 3, 9, 2], dtype=int64)
```

[7]:
```python
y_test_df = pd.DataFrame(test_pred, columns=['label'])
y_test_df
```

[7]:

	label
0	2
1	0
2	9
3	9
4	3
…	…
27995	9
27996	7
27997	3
27998	9
27999	2

[28000 rows x 1 columns]

[8]:
```python
y_test_df['imageid'] = range(1, len(y_test_df) + 1)

print(y_test_df)
```

```
       label  imageid
0          2        1
```

```
1          0        2
2          9        3
3          9        4
4          3        5
…       …       …
27995      9    27996
27996      7    27997
27997      3    27998
27998      9    27999
27999      2    28000

[28000 rows x 2 columns]
```

[9]:
```python
# reverse imageid and label

reversed_df = y_test_df.iloc[:, ::-1]

print(reversed_df)
```

```
       imageid  label
0            1      2
1            2      0
2            3      9
3            4      9
4            5      3
…          …    …
27995    27996      9
27996    27997      7
27997    27998      3
27998    27999      9
27999    28000      2

[28000 rows x 2 columns]
```

[10]:
```python
# convert to csv
csv_file_path = 'Random_Forrest.csv'

reversed_df.to_csv(csv_file_path, index=False)

print("DataFrame has been exported to:", csv_file_path)
```

```
DataFrame has been exported to: Random_Forrest.csv
```

3 3.

Execute principal components analysis (PCA) on the combined training and test set data together, generating principal components that represent 95 percent of the variability in the explanatory variables. The number of principal components in the solution should be substantially fewer than the explanatory variables.

[11]:
```
# combine train and test
y_test = pd.concat([y_test_df,test], axis=1)
y_test_1 = y_test.drop('imageid', axis =1)
combined_df = pd.concat([df,y_test_1], axis=0)
combined_df
```

[11]:
```
       label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
0          1       0       0       0       0       0       0       0       0
1          0       0       0       0       0       0       0       0       0
2          1       0       0       0       0       0       0       0       0
3          4       0       0       0       0       0       0       0       0
4          0       0       0       0       0       0       0       0       0
…        …     …       …     …     …     …       …     …     …
27995      9       0       0       0       0       0       0       0       0
27996      7       0       0       0       0       0       0       0       0
27997      3       0       0       0       0       0       0       0       0
27998      9       0       0       0       0       0       0       0       0
27999      2       0       0       0       0       0       0       0       0

       pixel8  …  pixel774  pixel775  pixel776  pixel777  pixel778  \
0           0  …         0         0         0         0         0
1           0  …         0         0         0         0         0
2           0  …         0         0         0         0         0
3           0  …         0         0         0         0         0
4           0  …         0         0         0         0         0
…         …  …       …       …       …       …       …
27995       0  …         0         0         0         0         0
27996       0  …         0         0         0         0         0
27997       0  …         0         0         0         0         0
27998       0  …         0         0         0         0         0
27999       0  …         0         0         0         0         0

       pixel779  pixel780  pixel781  pixel782  pixel783
0             0         0         0         0         0
1             0         0         0         0         0
2             0         0         0         0         0
3             0         0         0         0         0
4             0         0         0         0         0
…          …       …       …       …       …
27995         0         0         0         0         0
27996         0         0         0         0         0
27997         0         0         0         0         0
27998         0         0         0         0         0
27999         0         0         0         0         0
```

```
[70000 rows x 785 columns]
```

4 4.

Record the time it takes to identify the principal components.

[12]:
```python
scaler = StandardScaler()
start_pca = time.time()
X_combined_std = scaler.fit_transform(combined_df)


pca = PCA()
X_pca = pca.fit_transform(X_combined_std)
total_variance = pca.explained_variance_ratio_.cumsum()
n_components = (total_variance >= 0.95).sum() + 1

print("Number of principal components to explain 95% of variance:",␣
 ↪n_components)

X_final = pd.DataFrame(X_pca[:, :n_components], index=combined_df.index)
X_final
end_pca = time.time()
eval_pca = end_pca - start_pca
print("Time taken to evaluate model:", eval_pca, "seconds")
X_final
```

```
Number of principal components to explain 95% of variance: 454
Time taken to evaluate model: 3.288274049758911 seconds
```

[12]:
```
               0          1         2         3         4         5         6  \
0      -5.103770 -5.193883  4.129428 -0.757342  4.967296  1.948331  4.707284
1      19.416553  5.911728  1.359399 -2.240566  3.224957 -1.838375 -3.864186
2      -7.568914 -1.773322  2.518321  2.386178  4.979415 -4.260778 -1.029372
3      -0.364229  5.877963  1.979055  4.309507  2.434167  2.121527  4.472888
4      26.674254  5.826827  1.058325 -2.684070  9.544750 -2.413195 -6.398265
…            …         …         …         …         …         …         …
27995  -1.249671  9.156697 -2.679299 -0.813136 -5.809634 -0.706761  2.374931
27996  -3.696867  9.278876 -5.445028  0.276755  2.364963 -3.198716  7.318582
27997  -2.920813  1.284515  5.742789 -9.442766 -0.146840 -2.517996 -1.225240
27998  -4.101044  3.140228 -3.830122 -1.470510 -6.860404 -2.845385  1.439611
27999   8.836960 -4.852251 -1.551691  3.665382  0.117476  7.829419  3.770698

              7         8         9  …       444       445       446  \
0     -4.837158  0.226110 -1.459938  … -0.138395 -0.013756 -0.013763
1      0.311089 -4.082777 -4.372776  … -0.168335 -0.287670  0.372016
2      1.806906  0.309212  0.023823  … -0.154042 -0.035840 -0.072291
3     -0.350765  0.760271  5.455303  … -0.545309  0.436237  0.517453
```

```
4     -1.579450 -4.035316 -5.861316  …  0.281558  0.347400  0.063029
…          …        …        …   …        …        …        …
27995 -4.785125 -2.073833  0.069072  …  0.223381  0.071370  0.110460
27996  3.521148 -3.802287 -1.322660  …  0.173659 -0.309458 -0.410873
27997  0.488151 -2.488263  1.319430  …  0.388225 -0.333170  0.003227
27998 -0.608666  2.025145 -0.150314  …  0.036194  0.118622  0.095963
27999 -1.277567  0.366598 -0.313848  … -0.139401  0.234405  0.210672

            447       448       449       450       451       452       453
0      0.028553 -0.386743 -0.222973 -0.051896  0.204799 -0.069173  0.137192
1     -0.588254  0.226903 -0.212130 -0.330771 -0.085448 -0.137573 -0.182385
2      0.073108 -0.139128  0.057570  0.003677 -0.135744  0.137551  0.159890
3      0.661639 -0.825990  0.017165 -0.102902  0.181021 -0.200326  0.306938
4      0.384936  0.155465  0.084413 -0.409809  0.038743  0.383303 -0.124780
…          …        …        …        …        …        …        …
27995 -0.430937  0.250653 -0.286696  0.278667 -0.032859  0.092102 -0.146900
27996  0.029322  0.071143 -0.107493 -0.081462 -0.448199  0.399206 -0.450660
27997 -0.324096 -0.186532 -0.095870  0.549077 -0.048783  0.259762 -0.064910
27998  0.364961 -0.033128 -0.550620 -0.034456 -0.112851  0.121364  0.166695
27999 -0.290557 -0.101256  0.070025 -0.118407 -0.125716  0.001136 -0.029115

[70000 rows x 454 columns]
```

5 5.

Using the identified principal components from step (2), use thecsvto build another random forest classifier.

[13]:
```
random_forrest2 = pd.DataFrame(combined_df, columns=['label'])
random_forrest2
```

[13]:
```
       label
0          1
1          0
2          1
3          4
4          0
…        …
27995      9
27996      7
27997      3
27998      9
27999      2

[70000 rows x 1 columns]
```

[14]:
```python
combined_df2 = pd.concat([random_forrest2, X_final], axis=1)
combined_df2
```

[14]:
```
       label          0         1         2         3         4         5  \
0          1  -5.103770 -5.193883  4.129428 -0.757342  4.967296  1.948331
1          0  19.416553  5.911728  1.359399 -2.240566  3.224957 -1.838375
2          1  -7.568914 -1.773322  2.518321  2.386178  4.979415 -4.260778
3          4  -0.364229  5.877963  1.979055  4.309507  2.434167  2.121527
4          0  26.674254  5.826827  1.058325 -2.684070  9.544750 -2.413195
…        …         …         …         …         …         …         …
27995      9  -1.249671  9.156697 -2.679299 -0.813136 -5.809634 -0.706761
27996      7  -3.696867  9.278876 -5.445028  0.276755  2.364963 -3.198716
27997      3  -2.920813  1.284515  5.742789 -9.442766 -0.146840 -2.517996
27998      9  -4.101044  3.140228 -3.830122 -1.470510 -6.860404 -2.845385
27999      2   8.836960 -4.852251 -1.551691  3.665382  0.117476  7.829419

              6         7         8  …       444       445       446  \
0      4.707284 -4.837158  0.226110  … -0.138395 -0.013756 -0.013763
1     -3.864186  0.311089 -4.082777  … -0.168335 -0.287670  0.372016
2     -1.029372  1.806906  0.309212  … -0.154042 -0.035840 -0.072291
3      4.472888 -0.350765  0.760271  … -0.545309  0.436237  0.517453
4     -6.398265 -1.579450 -4.035316  …  0.281558  0.347400  0.063029
…           …         …         … …         …         …         …
27995  2.374931 -4.785125 -2.073833  …  0.223381  0.071370  0.110460
27996  7.318582  3.521148 -3.802287  …  0.173659 -0.309458 -0.410873
27997 -1.225240  0.488151 -2.488263  …  0.388225 -0.333170  0.003227
27998  1.439611 -0.608666  2.025145  …  0.036194  0.118622  0.095963
27999  3.770698 -1.277567  0.366598  … -0.139401  0.234405  0.210672

            447       448       449       450       451       452       453
0      0.028553 -0.386743 -0.222973 -0.051896  0.204799 -0.069173  0.137192
1     -0.588254  0.226903 -0.212130 -0.330771 -0.085448 -0.137573 -0.182385
2      0.073108 -0.139128  0.057570  0.003677 -0.135744  0.137551  0.159890
3      0.661639 -0.825990  0.017165 -0.102902  0.181021 -0.200326  0.306938
4      0.384936  0.155465  0.084413 -0.409809  0.038743  0.383303 -0.124780
…           …         …         …         …         …         …         …
27995 -0.430937  0.250653 -0.286696  0.278667 -0.032859  0.092102 -0.146900
27996  0.029322  0.071143 -0.107493 -0.081462 -0.448199  0.399206 -0.450660
27997 -0.324096 -0.186532 -0.095870  0.549077 -0.048783  0.259762 -0.064910
27998  0.364961 -0.033128 -0.550620 -0.034456 -0.112851  0.121364  0.166695
27999 -0.290557 -0.101256  0.070025 -0.118407 -0.125716  0.001136 -0.029115

[70000 rows x 455 columns]
```

6 6.

Record the time it takes to fit the model and to evaluate the model on the csvdata by submitting to Kaggle.com. Provide your Kaggle.com score and user ID.

[15]:
```
X1= combined_df2.drop('label', axis=1)
y1 = combined_df2['label']


X_train1, X_test1, y_train1, y_test1 = train_test_split(X1, y1, test_size=0.2,␣
↪random_state=42)

# Step 2: Train the Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
start_random = time.time()
rf_classifier.fit(X_train1, y_train1)

# Step 3: Evaluate the model
y_pred1 = rf_classifier.predict(X_test1)
accuracy = accuracy_score(y_test1, y_pred1)
print("Accuracy:", accuracy)


end_random = time.time()
eval_random = end_random - start_random
print("Time taken to fit and evaluated the model:", eval_random, "seconds")
```

```
Accuracy: 0.9401428571428572
Time taken to fit and evaluated the model: 297.3497874736786 seconds
```

[16]:
```
test_pred2 = rf_classifier.predict(X_final)
test_pred2
```

[16]:
```
array([1, 0, 1, …, 3, 9, 2], dtype=int64)
```

[17]:
```
# formatting

y_test_df2 = pd.DataFrame(test_pred2, columns=['label'])
y_test_df2['imageid'] = range(1, len(y_test_df2) + 1)
y_test_df2
```

[17]:

	label	imageid
0	1	1
1	0	2
2	1	3
3	4	4
4	0	5
…	…	…

```
69995       9     69996
69996       7     69997
69997       3     69998
69998       9     69999
69999       2     70000

[70000 rows x 2 columns]
```

[18]:
```python
reversed_df2 = y_test_df2.iloc[:, ::-1]

print(reversed_df2)
```

```
       imageid  label
0            1      1
1            2      0
2            3      1
3            4      4
4            5      0
…          …    …
69995    69996      9
69996    69997      7
69997    69998      3
69998    69999      9
69999    70000      2

[70000 rows x 2 columns]
```

[19]:
```python
subset_df = reversed_df2[-28000:]
random_forrest2 = subset_df.drop('imageid', axis=1)
random_forrest2['imageid'] = range(1, len(random_forrest2) + 1)

random_forrest_2 = random_forrest2.iloc[:, ::-1]

random_forrest_2
```

[19]:

	imageid	label
42000	1	2
42001	2	0
42002	3	9
42003	4	4
42004	5	3
…	…	…
69995	27996	9
69996	27997	7
69997	27998	3
69998	27999	9
69999	28000	2

[28000 rows x 2 columns]

[20]:
```
random_forrest_2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28000 entries, 42000 to 69999
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   imageid  28000 non-null  int64
 1   label    28000 non-null  int64
dtypes: int64(2)
memory usage: 437.6 KB
```

[21]:
```
csv_file_path = 'Random_Forrest_2.csv'

random_forrest_2.to_csv(csv_file_path, index=False)

print("DataFrame has been exported to:", csv_file_path)
```

```
DataFrame has been exported to: Random_Forrest_2.csv
```

7 7.

Use k-means clustering to group MNIST observations into 1 of 10 categories and then assign labels.

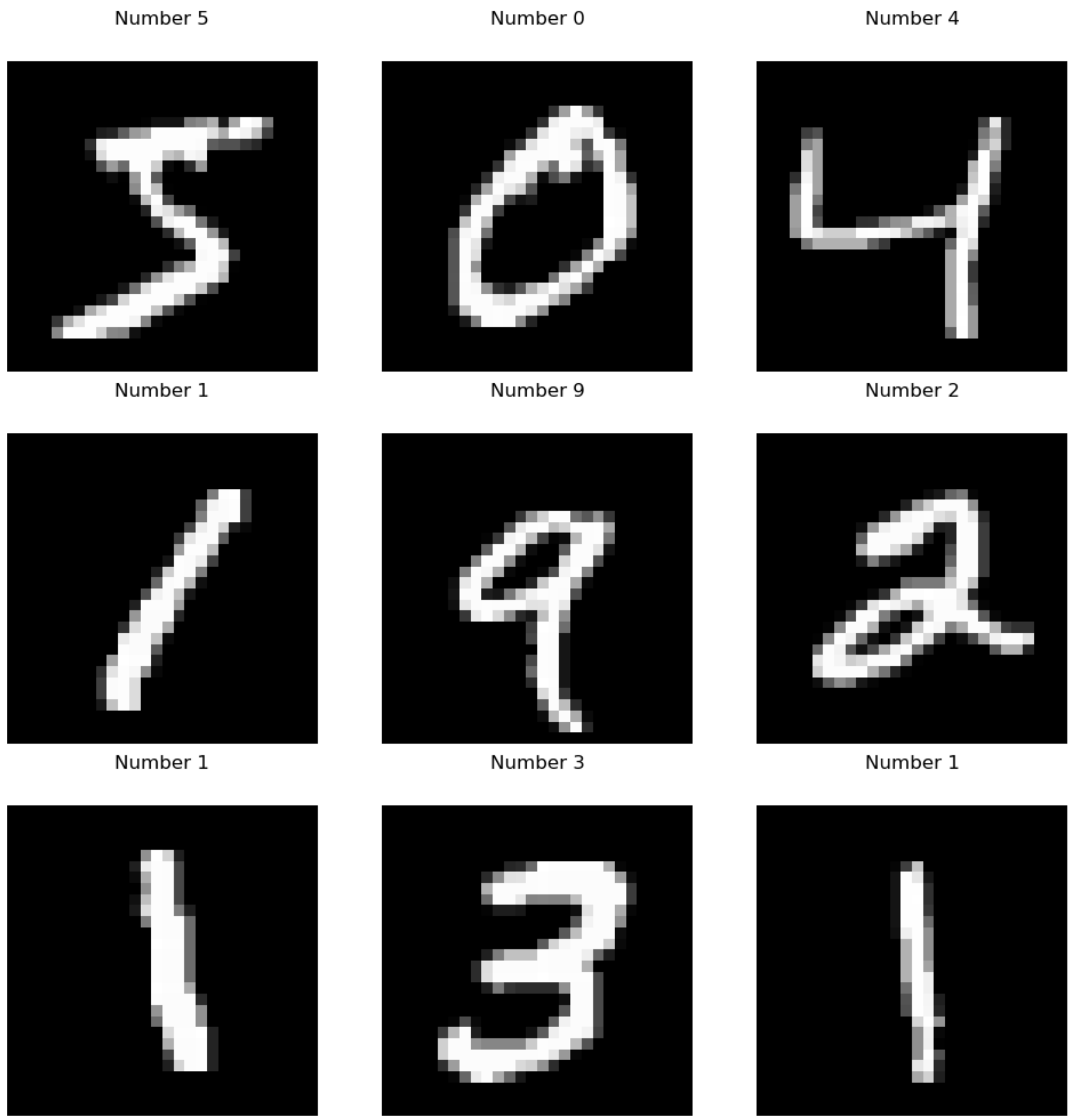
[22]:
```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

[23]:
```
print('Training Data: {}'.format(x_train.shape))
print('Training Labels: {}'.format(y_train.shape))
# Output should be:
# Training Data: (60000, 28, 28)
# Training Labels: (60000,)

print('Testing Data: {}'.format(x_test.shape))
print('Testing Labels: {}'.format(y_test.shape))
# Output should be:
# Testing Data: (10000, 28, 28)
# Testing Labels: (10000,)
```

```
Training Data: (60000, 28, 28)
Training Labels: (60000,)
Testing Data: (10000, 28, 28)
Testing Labels: (10000,)
```

[24]:
```
# EDA
fig, axs = plt.subplots(3, 3, figsize=(12, 12))
plt.gray()
```

```python
for i, ax in enumerate(axs.flat):
    ax.matshow(x_train[i])
    ax.axis('off')
    ax.set_title('Number {}'.format(y_train[i]))
plt.show()
```



[25]:
```python
# Convert each image to 1-dimensional array
X = x_train.reshape(len(x_train), -1)
Y = y_train

# Normalize the data to 0 - 1
```

```
X = X.astype(float) / 255.

# Printing the shape of the dataset to verify the changes
print(X.shape)  # This should print: (60000, 784)
print(X[0].shape)  # This should print: (784,)
```

```
(60000, 784)
(784,)
```

[26]:
```
n_digits = len(np.unique(y_test))
print("Number of unique digits:", n_digits)

# Initialize the KMeans model
kmeans = MiniBatchKMeans(n_clusters=n_digits)

# Fit the model to the training data
kmeans.fit(X)

# Retrieve the cluster labels for each data point in the training set
cluster_labels = kmeans.labels_
print("Cluster labels for the training data:", cluster_labels)
```

```
Number of unique digits: 10
```

```
C:\Users\Aaron\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1930:
FutureWarning: The default value of `n_init` will change from 3 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=3)
C:\Users\Aaron\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1962:
UserWarning: MiniBatchKMeans is known to have a memory leak on Windows with MKL,
when there are less chunks than available threads. You can prevent it by setting
batch_size >= 3072 or by setting the environment variable OMP_NUM_THREADS=4
  warnings.warn(
```

```
Cluster labels for the training data: [4 6 3 … 4 7 8]
```

[27]:
```
def infer_cluster_labels(kmeans, actual_labels):
    inferred_labels = {}
    for i in range(kmeans.n_clusters):
        # Find index of points in cluster
        index = np.where(kmeans.labels_ == i)[0]
        # Append actual labels for each point in cluster
        cluster_labels = actual_labels[index]
        # Determine most common label
        if len(cluster_labels) > 0:
            most_common = np.bincount(cluster_labels).argmax()
        else:
            most_common = -1  # Default or error value if no points in the␣
 ↪cluster
```

```
        # Assign the cluster to a value in the inferred_labels dictionary
        if most_common in inferred_labels:
            # Append the new number to the existing array at this slot
            inferred_labels[most_common].append(i)
        else:
            # Create a new array in this slot
            inferred_labels[most_common] = [i]
    return inferred_labels
```

[28]:
```
def infer_data_labels(X_clusters, cluster_labels):
    # Empty array of len(X)
    predicted_labels = np.zeros(len(X_clusters), dtype=np.uint8)
    for i, cluster in enumerate(X_clusters):
        for key, value in cluster_labels.items():
            if cluster in value:
                predicted_labels[i] = key
                break
    return predicted_labels
```

[29]:
```
# Assuming X and Y have been defined as your data and labels
kmeans = MiniBatchKMeans(n_clusters=36)
kmeans.fit(X)

# Infer labels for each cluster
cluster_labels = infer_cluster_labels(kmeans, Y)

# Predict labels for the training data
X_clusters = kmeans.predict(X)
predicted_labels = infer_data_labels(X_clusters, cluster_labels)

# Print the first 20 predicted and actual labels
print('Predicted labels:', predicted_labels[:20])
print('Actual labels:', Y[:20])

# Record centroid values
centroids = kmeans.cluster_centers_
```

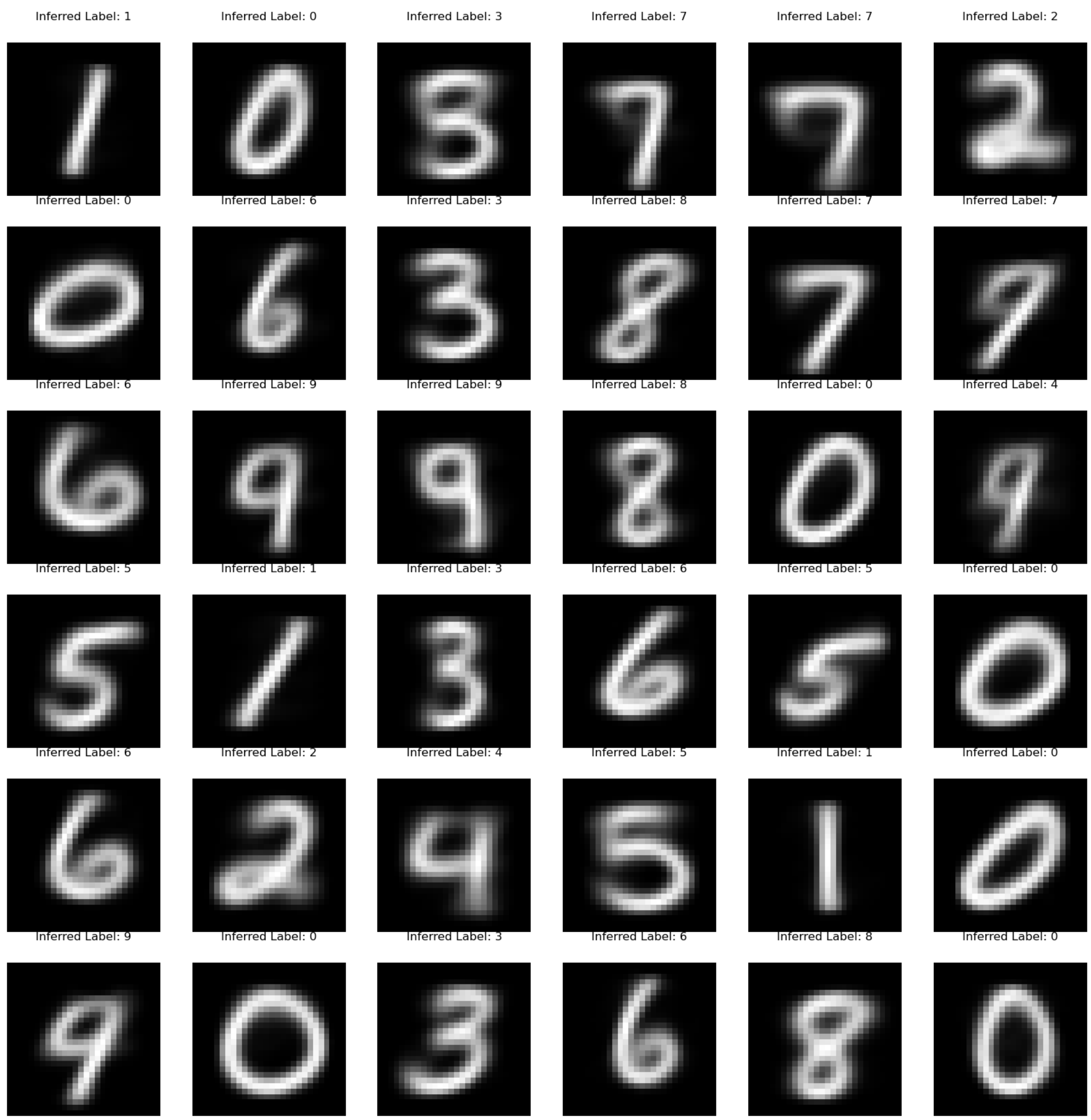
```
C:\Users\Aaron\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1930:
FutureWarning: The default value of `n_init` will change from 3 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=3)
C:\Users\Aaron\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1962:
UserWarning: MiniBatchKMeans is known to have a memory leak on Windows with MKL,
when there are less chunks than available threads. You can prevent it by setting
batch_size >= 3072 or by setting the environment variable OMP_NUM_THREADS=4
  warnings.warn(
```

```
Predicted labels: [3 0 2 1 9 2 1 3 1 4 3 5 3 6 1 7 2 8 6 9]
Actual labels: [5 0 4 1 9 2 1 3 1 4 3 5 3 6 1 7 2 8 6 9]
```

[30]:

```python
# Assuming X and Y have been defined as your data and labels
# Initialize and fit KMeans algorithm
kmeans = MiniBatchKMeans(n_clusters=36)
kmeans.fit(X)

# Record centroid values
centroids = kmeans.cluster_centers_

# Reshape centroids into images
images = centroids.reshape(36, 28, 28) * 255
images = images.astype(np.uint8)

# Determine cluster labels
cluster_labels = infer_cluster_labels(kmeans, Y)

# Create figure with subplots using matplotlib.pyplot
fig, axs = plt.subplots(6, 6, figsize=(20, 20))
plt.gray()

# Loop through subplots and add centroid images
for i, ax in enumerate(axs.flat):
    # Determine inferred label using cluster_labels dictionary
    inferred_label = [key for key, value in cluster_labels.items() if i in␣
 ↪value]
    if inferred_label:
        ax.set_title(f'Inferred Label: {inferred_label[0]}')
    else:
        ax.set_title('No Label')

    # Add image to subplot
    ax.matshow(images[i])
    ax.axis('off')

# Display the figure
plt.show()
```

```
C:\Users\Aaron\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1930:
FutureWarning: The default value of `n_init` will change from 3 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=3)
C:\Users\Aaron\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1962:
UserWarning: MiniBatchKMeans is known to have a memory leak on Windows with MKL,
when there are less chunks than available threads. You can prevent it by setting
batch_size >= 3072 or by setting the environment variable OMP_NUM_THREADS=4
  warnings.warn(
```



8 8.

Submit the RF Classifier, the PCA RF, and k-means estimations to Kaggle.com, and provide screen snapshots of your scores as well as your Kaggle.com user name.

[31]:
```
from sklearn import metrics
def calculate_metrics(estimator, data, labels):

    # Calculate and print metrics
    print('Number of Clusters: {}'.format(estimator.n_clusters))
    print('Inertia: {}'.format(estimator.inertia_))
```

```
    print('Homogeneity: {}'.format(metrics.homogeneity_score(labels, estimator.
↪labels_)))
```

[32]:
```
labels_flat = y.values.ravel()
labels_counts = np.unique(labels_flat).shape[0]
```

[35]:
```
start_time = time.time()

estimator = MiniBatchKMeans(n_clusters = 256)

X_test = np.multiply(test, 1.0 / 255.0)
y_test = estimator.fit(X_test)

estimator.fit(X_test)

# determine predicted labels
cluster_labels = infer_cluster_labels(estimator, labels_flat)
predicted_Y_test = infer_data_labels(estimator.labels_, cluster_labels)
end_time = time.time()
print('Duration: {}'.format(end_time - start_time))
print('Y_test:',predicted_Y_test)
```

```
C:\Users\Aaron\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1930:
FutureWarning: The default value of `n_init` will change from 3 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=3)
C:\Users\Aaron\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1962:
UserWarning: MiniBatchKMeans is known to have a memory leak on Windows with MKL,
when there are less chunks than available threads. You can prevent it by setting
batch_size >= 3072 or by setting the environment variable OMP_NUM_THREADS=4
  warnings.warn(
C:\Users\Aaron\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1930:
FutureWarning: The default value of `n_init` will change from 3 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=3)
C:\Users\Aaron\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1962:
UserWarning: MiniBatchKMeans is known to have a memory leak on Windows with MKL,
when there are less chunks than available threads. You can prevent it by setting
batch_size >= 3072 or by setting the environment variable OMP_NUM_THREADS=4
  warnings.warn(
```

```
Duration: 2.9397435188293457
Y_test: [1 0 7 … 3 2 7]
```

[36]:
```
start_time = time.time()
X_test = np.multiply(test, 1.0 / 255.0)

# Create an instance of MiniBatchKMeans
```

```
estimator = MiniBatchKMeans(n_clusters=10)

# Fit the model to your data
estimator.fit(X_test)

y_test = estimator.predict(X_test)

# determine predicted labels
cluster_labels = infer_cluster_labels(estimator, labels_flat)
predicted_Y_test = infer_data_labels(estimator.labels_, cluster_labels)
end_time = time.time()
print('Duration: {}'.format(end_time - start_time))
print('Y_test:',predicted_Y_test)
```

```
C:\Users\Aaron\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1930:
FutureWarning: The default value of `n_init` will change from 3 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=3)
C:\Users\Aaron\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1962:
UserWarning: MiniBatchKMeans is known to have a memory leak on Windows with MKL,
when there are less chunks than available threads. You can prevent it by setting
batch_size >= 3072 or by setting the environment variable OMP_NUM_THREADS=4
  warnings.warn(

Duration: 0.6765658855438232
Y_test: [1 2 3 … 1 9 1]
```

[37]:
```
y_test
```

[37]: array([1, 5, 3, …, 6, 0, 1])

[38]:
```
k_means_2 = pd.DataFrame(y_test, columns=['label'])
k_means_2
```

[38]:

	label
0	1
1	5
2	3
3	3
4	1
…	…
27995	0
27996	8
27997	6
27998	0
27999	1

[28000 rows x 1 columns]

[39]:
```python
k_means_2['label'] = k_means_2['label'].astype('int64')
```

[41]:
```python
k_means_2
```

[41]:
```
       label
0          1
1          5
2          3
3          3
4          1
…      …
27995      0
27996      8
27997      6
27998      0
27999      1

[28000 rows x 1 columns]
```

[42]:
```python
k_means_2['imageid'] = range(1, len(k_means_2) + 1)
k_means_2
```

[42]:
```
       label  imageid
0          1        1
1          5        2
2          3        3
3          3        4
4          1        5
…      …      …
27995      0    27996
27996      8    27997
27997      6    27998
27998      0    27999
27999      1    28000

[28000 rows x 2 columns]
```

[43]:
```python
reversed_k_means_2 = k_means_2.iloc[:, ::-1]

reversed_k_means_2
```

[43]:
```
       imageid  label
0            1      1
1            2      5
2            3      3
3            4      3
4            5      1
```

```
…         …     …
27995     27996     0
27996     27997     8
27997     27998     6
27998     27999     0
27999     28000     1

[28000 rows x 2 columns]
```

[44]:
```python
# convert to csv
csv_file_path = 'k_means.csv'

reversed_k_means_2.to_csv(csv_file_path, index=False)

print("DataFrame has been exported to:", csv_file_path)
```

```
DataFrame has been exported to: k_means.csv
```

9 9.

The experiment we have proposed has a major design flaw. Identify the flaw. Fix it. Rerun the experiment in a way that is consistent with a training-and-test regimen, and submit this to Kaggle.com.

[76]:
```python
df_2 = pd.read_csv('train.csv')
test_2 = pd.read_csv("test.csv")
test_2.head()
```

[76]:
```
   pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
0       0       0       0       0       0       0       0       0       0
1       0       0       0       0       0       0       0       0       0
2       0       0       0       0       0       0       0       0       0
3       0       0       0       0       0       0       0       0       0
4       0       0       0       0       0       0       0       0       0

   pixel9  …  pixel774  pixel775  pixel776  pixel777  pixel778  pixel779  \
0       0  …         0         0         0         0         0         0
1       0  …         0         0         0         0         0         0
2       0  …         0         0         0         0         0         0
3       0  …         0         0         0         0         0         0
4       0  …         0         0         0         0         0         0

   pixel780  pixel781  pixel782  pixel783
0         0         0         0         0
1         0         0         0         0
2         0         0         0         0
3         0         0         0         0
```

```
4          0          0          0          0

[5 rows x 784 columns]
```

[77]:
```python
scaler = StandardScaler()
start_pca = time.time()
X_df = scaler.fit_transform(df_2)


pca = PCA()
X_pca2 = pca.fit_transform(X_df)
total_variance2 = pca.explained_variance_ratio_.cumsum()
n_components2 = (total_variance2 >= .95).sum() + 1

print("Number of principal components to explain 95% of variance:",␣
 ↪n_components2)

X_final_2 = pd.DataFrame(X_pca2[:, :n_components], index=df_2.index)
X_final_2
end_pca = time.time()
eval_pca = end_pca - start_pca
print("Time taken to evaluate model:", eval_pca, "seconds")
X_final_2
```

```
Number of principal components to explain 95% of variance: 466
Time taken to evaluate model: 1.8500235080718994 seconds
```

[77]:
```
               0          1          2          3          4          5    \
0      -5.012867  -5.498016   3.834079  -0.911208   4.901249   2.109432
1      19.331395   6.018994   1.533175  -2.393805   3.160668  -1.732271
2      -7.535735  -1.959875   2.425707   2.222502   5.263602  -4.077669
3      -0.477535   5.716447   2.284497   4.262101   2.430751   2.172512
4      26.602728   6.049490   1.153826  -3.034488   9.458046  -2.078839
…           …         …          …          …          …          …
41995  13.757582  -1.252759  -3.884442  -5.412562  10.998729  -4.880634
41996  -8.761718  -1.455617   2.494011   1.508431   5.950198  -2.738448
41997   0.349180   7.552364 -11.811577  -3.241547   0.814273   0.277451
41998   2.294988  -4.282627   0.415952  10.028829  -5.546775  -5.579433
41999  -4.900233   1.695605  -2.494849   2.221236  -1.058251   0.154667

              6          7          8          9    …       456        457   \
0       4.676996  -4.783619   0.248476  -1.464685  …  -0.219396   0.085102
1      -3.775959   0.170572  -4.131370  -4.297755  …   1.034829  -0.635973
2      -1.047423   1.729455   0.434058  -0.067500  …   0.312861  -0.144592
3       4.414617  -0.358598   0.965020   5.513745  …   0.253014   0.217565
4      -6.239619  -1.785187  -4.079610  -5.778000  …  -0.370614  -0.057745
…           …         …          …          …   …       …
41995  -0.046655  -5.104049  -4.197285  -0.703786  …   0.139179  -0.252397
```

```
41996 -0.385049   0.525329  0.987921 -1.116523  …  0.039814 -0.271099
41997 -2.062746  10.554643 -2.236491 -1.878677  …  0.187921 -0.323872
41998  0.173605   5.439562  2.213281 -1.766359  … -0.095235 -0.112722
41999  1.183296  -3.288391 -1.616214 -1.439040  …  0.284390 -0.250979

            458       459       460       461       462       463       464  \
0      0.100703  0.182718 -0.257493  0.207047  0.162810  0.267236  0.124452
1     -0.292971  0.986034  0.207907 -0.031758 -0.450395 -0.320007 -0.377114
2     -0.067551  0.147673 -0.104898 -0.214094  0.125024 -0.167062  0.105325
3      0.533490  0.223535  0.832896  0.736629 -0.093954 -0.923588 -0.518855
4     -0.046481  0.183534  0.390126  0.331432 -0.059655  0.201392  0.156206
…           …         …         …         …         …         …         …
41995 -0.150053  0.180256  0.218848  0.039020  0.374936  0.289252 -0.277490
41996  0.372753  0.061815 -0.254282 -0.092567 -0.106489  0.069423 -0.079046
41997  0.223313 -0.239063  0.245395  0.013941 -0.480762  0.060393  0.354993
41998 -0.394967 -0.203291  0.164720 -0.225375 -0.182538 -0.282746 -0.417783
41999 -0.558102  0.187115  0.179280 -0.099341  0.167251 -0.012786 -0.167557

            465
0     -0.046525
1     -0.604506
2     -0.202823
3     -0.238233
4      0.770097
…           …
41995 -0.079300
41996  0.350960
41997 -0.129146
41998 -0.043643
41999  0.213934

[42000 rows x 466 columns]
```

[78]:
```python
X_final_2['label'] = df_2['label']
X_final_2
```

[78]:
```
               0         1          2          3          4         5  \
0      -5.012867 -5.498016   3.834079  -0.911208   4.901249  2.109432
1      19.331395  6.018994   1.533175  -2.393805   3.160668 -1.732271
2      -7.535735 -1.959875   2.425707   2.222502   5.263602 -4.077669
3      -0.477535  5.716447   2.284497   4.262101   2.430751  2.172512
4      26.602728  6.049490   1.153826  -3.034488   9.458046 -2.078839
…            …         …          …          …          …         …
41995  13.757582 -1.252759  -3.884442  -5.412562  10.998729 -4.880634
41996  -8.761718 -1.455617   2.494011   1.508431   5.950198 -2.738448
41997   0.349180  7.552364 -11.811577  -3.241547   0.814273  0.277451
41998   2.294988 -4.282627   0.415952  10.028829  -5.546775 -5.579433
```

```
41999  -4.900233  1.695605  -2.494849   2.221236  -1.058251  0.154667

              6          7         8         9  …        457       458  \
0      4.676996  -4.783619  0.248476 -1.464685  …  0.085102  0.100703
1     -3.775959   0.170572 -4.131370 -4.297755  … -0.635973 -0.292971
2     -1.047423   1.729455  0.434058 -0.067500  … -0.144592 -0.067551
3      4.414617  -0.358598  0.965020  5.513745  …  0.217565  0.533490
4     -6.239619  -1.785187 -4.079610 -5.778000  … -0.057745 -0.046481
…           …          …         …         …  …        …         …
41995 -0.046655  -5.104049 -4.197285 -0.703786  … -0.252397 -0.150053
41996 -0.385049   0.525329  0.987921 -1.116523  … -0.271099  0.372753
41997 -2.062746  10.554643 -2.236491 -1.878677  … -0.323872  0.223313
41998  0.173605   5.439562  2.213281 -1.766359  … -0.112722 -0.394967
41999  1.183296  -3.288391 -1.616214 -1.439040  … -0.250979 -0.558102

            459       460       461       462       463       464       465  \
0      0.182718 -0.257493  0.207047  0.162810  0.267236  0.124452 -0.046525
1      0.986034  0.207907 -0.031758 -0.450395 -0.320007 -0.377114 -0.604506
2      0.147673 -0.104898 -0.214094  0.125024 -0.167062  0.105325 -0.202823
3      0.223535  0.832896  0.736629 -0.093954 -0.923588 -0.518855 -0.238233
4      0.183534  0.390126  0.331432 -0.059655  0.201392  0.156206  0.770097
…           …         …         …         …         …         …         …
41995  0.180256  0.218848  0.039020  0.374936  0.289252 -0.277490 -0.079300
41996  0.061815 -0.254282 -0.092567 -0.106489  0.069423 -0.079046  0.350960
41997 -0.239063  0.245395  0.013941 -0.480762  0.060393  0.354993 -0.129146
41998 -0.203291  0.164720 -0.225375 -0.182538 -0.282746 -0.417783 -0.043643
41999  0.187115  0.179280 -0.099341  0.167251 -0.012786 -0.167557  0.213934

       label
0          1
1          0
2          1
3          4
4          0
…        …
41995      0
41996      1
41997      7
41998      6
41999      9

[42000 rows x 467 columns]
```

[79]:

```
X3 = df_2.drop('label', axis=1)
y3 = df_2['label']
```

```
X_train3, X_test3, y_train3, y_test3 = train_test_split(X3, y3, test_size=0.2, 
 ↪random_state=42)

# Step 2: Train the Random Forest classifier
rf_classifier3 = RandomForestClassifier(n_estimators=100, random_state=42)
start_random = time.time()
rf_classifier3.fit(X_train3, y_train3)

# Step 3: Evaluate the model
y_pred3 = rf_classifier3.predict(X_test3)
accuracy3 = accuracy_score(y_test3, y_pred3)
print("Accuracy:", accuracy3)


end_random = time.time()
eval_random = end_random - start_random
print("Time taken to fit and evaluated the model:", eval_random, "seconds")
```

```
Accuracy: 0.9628571428571429
Time taken to fit and evaluated the model: 24.08950138092041 seconds
```

[80]:
```
test_pred4 = rf_classifier3.predict(test_2)
test_pred4
```

[80]:
```
array([2, 0, 9, …, 3, 9, 2], dtype=int64)
```

[81]:
```
y_test_df_2 = pd.DataFrame(test_pred4, columns=['label'])
y_test_df_2
```

[81]:

	label
0	2
1	0
2	9
3	9
4	3
…	…
27995	9
27996	7
27997	3
27998	9
27999	2

[28000 rows x 1 columns]

[82]:
```
y_test_df_2['imageid'] = range(1, len(y_test_df_2) + 1)
```

[83]:
```
y_test_df_2
```

[83]:
```
       label  imageid
0          2        1
1          0        2
2          9        3
3          9        4
4          3        5
…        …      …
27995      9    27996
27996      7    27997
27997      3    27998
27998      9    27999
27999      2    28000

[28000 rows x 2 columns]
```

[84]:
```python
reversed_df4 = y_test_df_2.iloc[:, ::-1]

print(reversed_df4)
```

```
       imageid  label
0            1      2
1            2      0
2            3      9
3            4      9
4            5      3
…          …    …
27995    27996      9
27996    27997      7
27997    27998      3
27998    27999      9
27999    28000      2

[28000 rows x 2 columns]
```

[85]:
```python
csv2 = 'PCA_Random.csv'

reversed_df4.to_csv(csv2, index=False)

print("DataFrame has been exported to:", csv2)
```

```
DataFrame has been exported to: PCA_Random.csv
```