MNIST Digit Recognizer: PCA, Random Forest, K-Means

According to the Kaggle overview, “MNIST ("Modified National Institute of Standards and Technology") is the de facto “hello world” dataset of computer vision.” It is a classic dataset to use for measuring and benchmarking classification algorithms. In this research, we will identify correct digits from a dataset of tens of thousands of handwritten images. The dataset contains 700+ columns each indicating a pixel of the chart. We will apply random forest classifier, Principal Component Analysis, and K-means clustering to analyze this dataset, making predictions and identifying principal components that best explain variability. First, we performed basic exploratory data analysis to better understand the dataset. Figures 1-5 shows 785 columns and int values that range from 0 (black pixel) to 255 (white). Figure 6 shows that there is a fairly even distribution of labels from 0-9. There are no null values. Figure 7 shows that 76 columns have constant values (all 255 or all 0). These will not contribute to any of the classification models we create so we drop these columns. The dataset now has 709 columns and is ready to be worked with.

After splitting the dataset into an 80/20 training/testing split we create our first model - a random forest classifier using grid search to tune hyperparameters. After finding the best hyperparameters, we train the classifier on the training set. Figures 8-11 shows this process, and figure 10 shows that it took 21.68 seconds to train. Figures 12-14 shows the confusion matrices in both table and heatmap form for training and testing data. Figure 37 shows a Kaggle score of 0.91 - a good start.

To try and improve upon this model we used Principal Components Analysis (PCA) to try and perform dimensionality reduction to better train a random forest classifier model. Performing PCA took 12.68 seconds as shown in Figure 16. Figures 17-19 shows the same process as the first random forest - using Grid Search to tune hyperparameters after PCA reduced the features to 153 columns. After finding the best parameters it took 1 minute and 20 seconds to fit (Figure

20). Figures 21-25 shows the resulting confusion matrices which looked quite similar to the original random forest classifier. Our Kaggle score using this classification model was about 0.92 (Figure 38), which was a slight improvement.

K-Means Clustering is a popular unsupervised machine learning algorithm used for partitioning a dataset into K distinct, non-overlapping clusters. It iteratively assigns each data point to the nearest cluster centroid based on a distance metric, typically Euclidean distance. Each data point is assigned to the cluster with the nearest centroid and then repeats the process and updates the centroids until the centroids no longer change significantly. In this research we will first decide how many clusters we want K-means to produce, then map the K-means' cluster label with the correct trained dataset's label. This way we can apply the force of unsupervised learning in classification problems.

Since we have 10 labels, our immediate guess is to have the K-means algorithm produce 10 clusters. After training this, the model assigned a label to each row. However, since K-means is an unsupervised method, it labels each cluster randomly and we need to assign the true label to each cluster based on the trained dataset's label. We used an algorithm that maps each cluster based on the most common (majority class) true label in this cluster. For example, if a cluster with a random label of "1" has 30% true label of "2" and 70% true label of "3", it will be assigned a label of 2. After applying this algorithm, we got a decent prediction based on the confusion matrix (Figure 31). However, a problem occurred: Our model prediction has nothing for label "5" because no cluster has this label as the majority. This model is therefore not useful.

We quickly realized 10 clusters are too few as the MNIST graph can have charts plotted on different pixels in different shapes but have the same label. We may need many more clusters to generalize different kinds of charts. We then used the Silhouette score to analyze the best number of clusters, but the result shows a decreasing trend of Silhouette score when the cluster goes up (Figures 26-27). However, when we try different cluster numbers and see how the model performs, we find the higher the better. The Silhouette score is not useful in this scenario and we have not figured out the reason for this conflict. In the end, we decided to use a cluster of 2000 to train k-means (unlikely to underfit, nor overfit) and then map these clusters based on the majority vote. The result is an amazing categorization accuracy of 0.94 (Figure 39).

Appendix

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

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
%cd /content/drive/My Drive/
df = pd.read_csv('MNIST_Digit/train.csv')
df.head(5)
```

```
df = pd.read_csv('MNIST_Digit/train.csv')
df.head(5)
```

	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

Figure 1

```
# Check datatypes, all numeric data
data_type_counts = df.dtypes.value_counts()
print(data_type_counts)

int64 785 Name: count, dtype: int64
```

Figure 2

```
len(df.columns)

785
```

Figure 3

```
#Null Value Columns
nullseries= df.isna().sum()
```

```
print(nullseries[nullseries > 0])
```

```
Series([], dtype: int64)
```

Figure 4

```
# Describe the data
df.describe()
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779
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	42000.00000	42000.000000	42000.000000
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	0.02019	0.017238	0.002857
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	1.75987	1.894498	0.414264
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	0.00000	0.000000	0.000000
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	0.00000	0.000000	0.000000
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	0.00000	0.000000	0.000000
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	0.00000	0.000000	0.000000
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	253.00000	254.000000	62.000000

8 rows × 785 columns

Figure 5

```
#Create Histogram
import matplotlib.pyplot as plt

# Assuming 'df' is your DataFrame and 'Bankrupt?' is the column name
plt.hist(df['label'], bins=10, edgecolor='black') # Assuming binary data, adjust 'bins' as needed
plt.xlabel('label')
plt.ylabel('Frequency')
plt.title('Histogram of Digits')
plt.show()
```

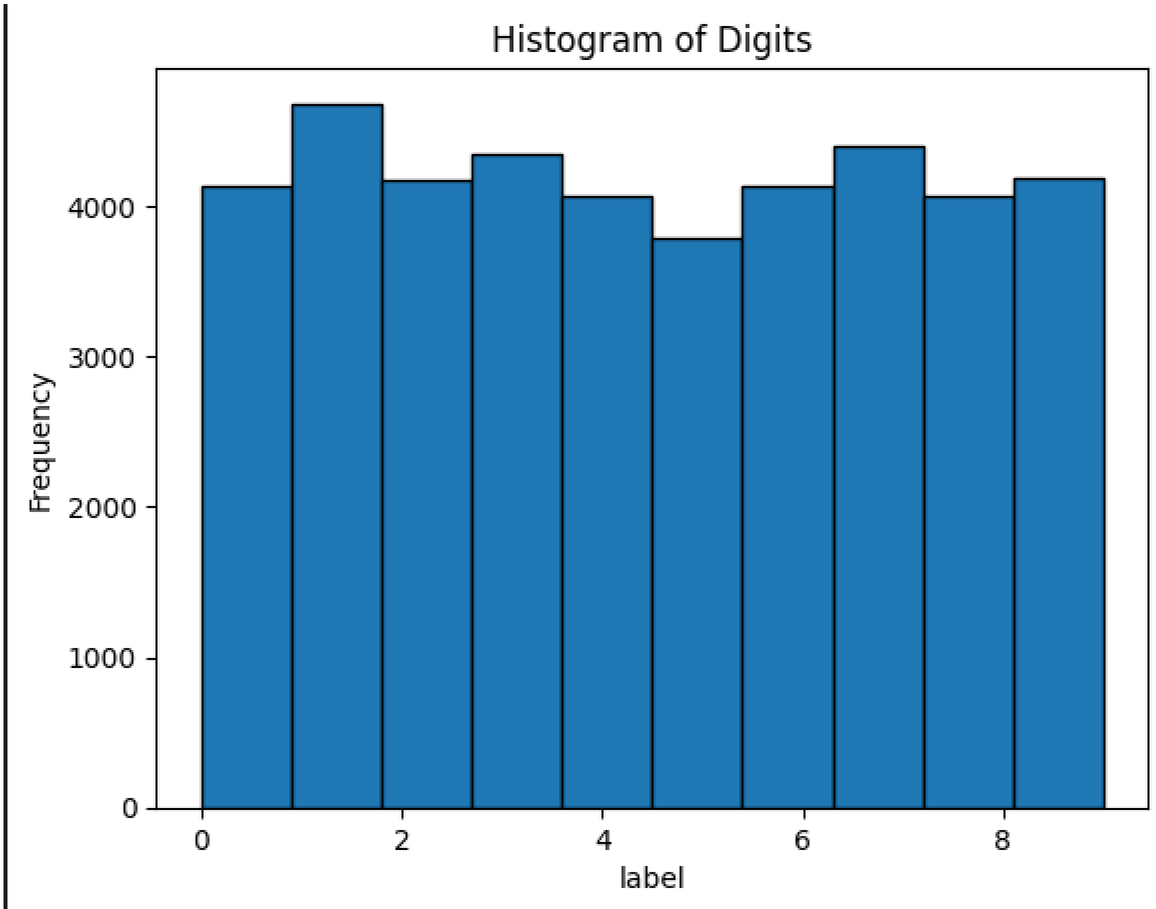

Figure 6

```
# Remove any columns with constant values. They won't contribute to the
classifiers
clean_df = df.copy()
black_pixels = []
white_pixels = []
print(len(df.columns))
for pixel in df.columns:
if max(df[pixel]) == 0:
black_pixels.append(pixel)
if min(df[pixel]) == 255:
white_pixels.append(pixel)

clean_df = clean_df.drop(black_pixels, axis=1)
clean_df = clean_df.drop(white_pixels, axis=1)
print(len(black_pixels))
```

```
print(len(white_pixels))
print(len(clean_df.columns))
```

```
785 76 0 709
```

Figure 7

Split Training and Testing

```
x = clean_df.drop(columns=['label'])
y = clean_df['label']

# Use K-Fold Later
from sklearn.model_selection import train_test_split
# Split the dataset into 80% train and 20% test
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)
```

Random Forest Classifier Training

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report

# Standard Scale
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)


rand_forest = RandomForestClassifier()

# Parameter Grid
param_grid = {
'n_estimators': [10, 100],
```

```
'max_features': ['sqrt', 'log2'],
'max_depth': [2, 4, 6, 8, 10],
'criterion': ['gini', 'entropy', 'log_loss']
}
```

Figure 8

```
# Grid Search
kf = KFold(n_splits=5, shuffle=True, random_state=42)
grid_search = GridSearchCV(rand_forest, param_grid, cv=kf, n_jobs=-1)
grid_search.fit(x_train_scaled, y_train)
```

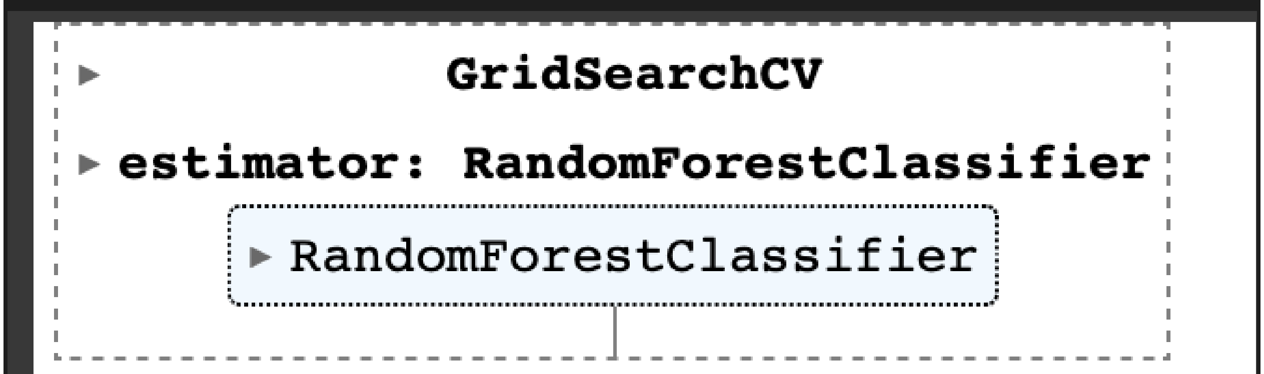

Figure 9

```
from datetime import datetime

# Train on tuned logistic regression
best_params = grid_search.best_params_
tuned_rand_forest = RandomForestClassifier(**best_params)
start = datetime.now()
tuned_rand_forest.fit(x_train_scaled, y_train)
end = datetime.now()
print(end-start)
```

```
# hyperparameter tuning
# 1. n_estimators (number of trees)
# 2. max_features (maximum features considered for splitting a node)
# 3. max_depth (maximum number of levels in each tree)
# 4. splitting criteria (entropy or gini)

0:00:21.677312
```

Figure 10

```
# Random Forest Best Params
best_params

{'criterion': 'entropy',
 'max_depth': 10,
 'max_features': 'sqrt',
 'n_estimators': 100}
```

Figure 11

```
# Predict
rand_forest_y_train_pred = tuned_rand_forest.predict(x_train_scaled)
train_conf_matrix = confusion_matrix(y_train, rand_forest_y_train_pred)
print("Confusion Matrix (Training Data):\n", train_conf_matrix)
print("\nClassification Report (Training Data):\n",
classification_report(y_train, rand_forest_y_train_pred))

# On testing data
rand_forest_y_test_pred = tuned_rand_forest.predict(x_test_scaled)
test_conf_matrix = confusion_matrix(y_test, rand_forest_y_test_pred)
print("\nConfusion Matrix (Testing Data):\n", test_conf_matrix)
print("\nClassification Report (Testing Data):\n",
classification_report(y_test, rand_forest_y_test_pred))

Confusion Matrix (Training Data): [[2918 0 0 0 2 0 3 0 9 0] [ 0 3271 7 4 3
1 1 4 3 1] [ 0 5 2820 2 11 0 5 28 4 8] [ 0 3 20 2909 2 16 4 18 15 9] [ 0 4
0 0 2754 0 4 4 2 82] [ 4 6 0 5 2 2675 3 0 8 7] [ 4 4 1 0 3 4 2865 0 0 0]
```

```
[ 0 16 25 1 13 0 0 2945 3 39] [ 1 18 5 7 5 0 1 1 2784 32] [ 3 10 0 32 25 4
0 30 8 2845]] Classification Report (Training Data): precision recall f1-
score support 0 1.00 1.00 1.00 2932 1 0.98 0.99 0.99 3295 2 0.98 0.98 0.98
2883 3 0.98 0.97 0.98 2996 4 0.98 0.97 0.97 2850 5 0.99 0.99 0.99 2710 6
0.99 0.99 0.99 2881 7 0.97 0.97 0.97 3042 8 0.98 0.98 0.98 2854 9 0.94
0.96 0.95 2957 accuracy 0.98 29400 macro avg 0.98 0.98 0.98 29400 weighted
avg 0.98 0.98 0.98 29400 Confusion Matrix (Testing Data): [[1184 0 1 1 1 0
5 0 7 1] [ 0 1367 3 7 1 1 6 2 1 1] [ 6 7 1214 6 13 3 12 21 9 3] [ 6 4 21
1226 2 36 4 19 21 16] [ 2 0 1 0 1153 0 8 3 6 49] [ 2 5 3 26 2 1006 15 2 11
13] [ 10 4 2 0 6 5 1219 1 9 0] [ 1 11 21 1 12 1 0 1264 3 45] [ 2 12 7 12 6
8 7 2 1133 20] [ 9 5 4 23 21 4 2 12 11 1140]] Classification Report
(Testing Data): precision recall f1-score support 0 0.97 0.99 0.98 1200 1
0.97 0.98 0.98 1389 2 0.95 0.94 0.94 1294 3 0.94 0.90 0.92 1355 4 0.95
0.94 0.95 1222 5 0.95 0.93 0.94 1085 6 0.95 0.97 0.96 1256 7 0.95 0.93
0.94 1359 8 0.94 0.94 0.94 1209 9 0.89 0.93 0.91 1231 accuracy 0.94 12600
macro avg 0.94 0.94 0.94 12600 weighted avg 0.95 0.94 0.94 12600
```

Figure 12

```
# import required modules for performance evaluation
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score

# Training Data confusion matrix for random forest
class_names=[0,1] # name of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(train_conf_matrix), annot=True,
cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
```

```
plt.tight_layout()
plt.title('Training data confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

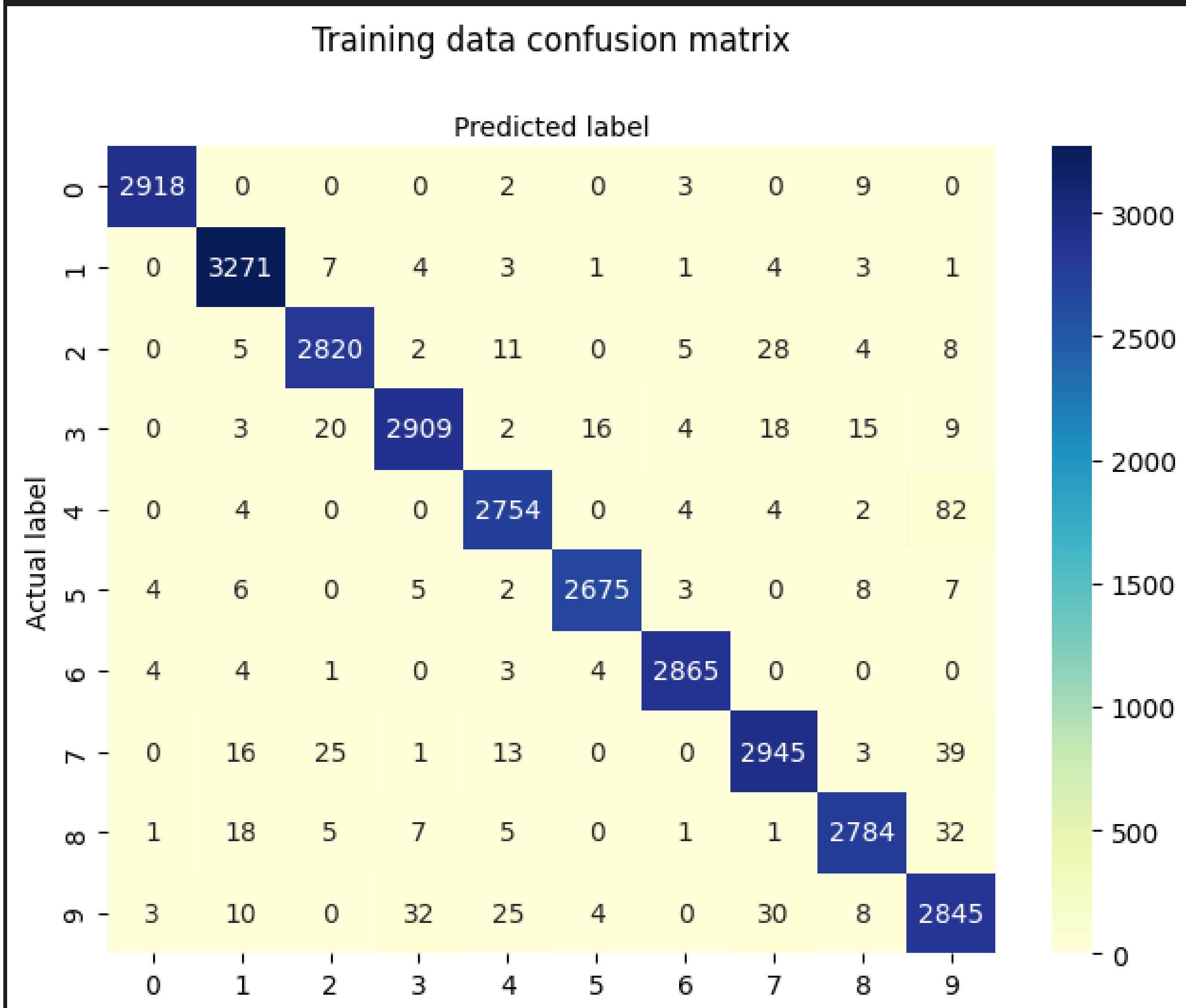

Figure 13

```
# Testing Data confusion matrix for random forest
class_names=[0,1] # name of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
```

```
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(test_conf_matrix), annot=True,
cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Test data confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

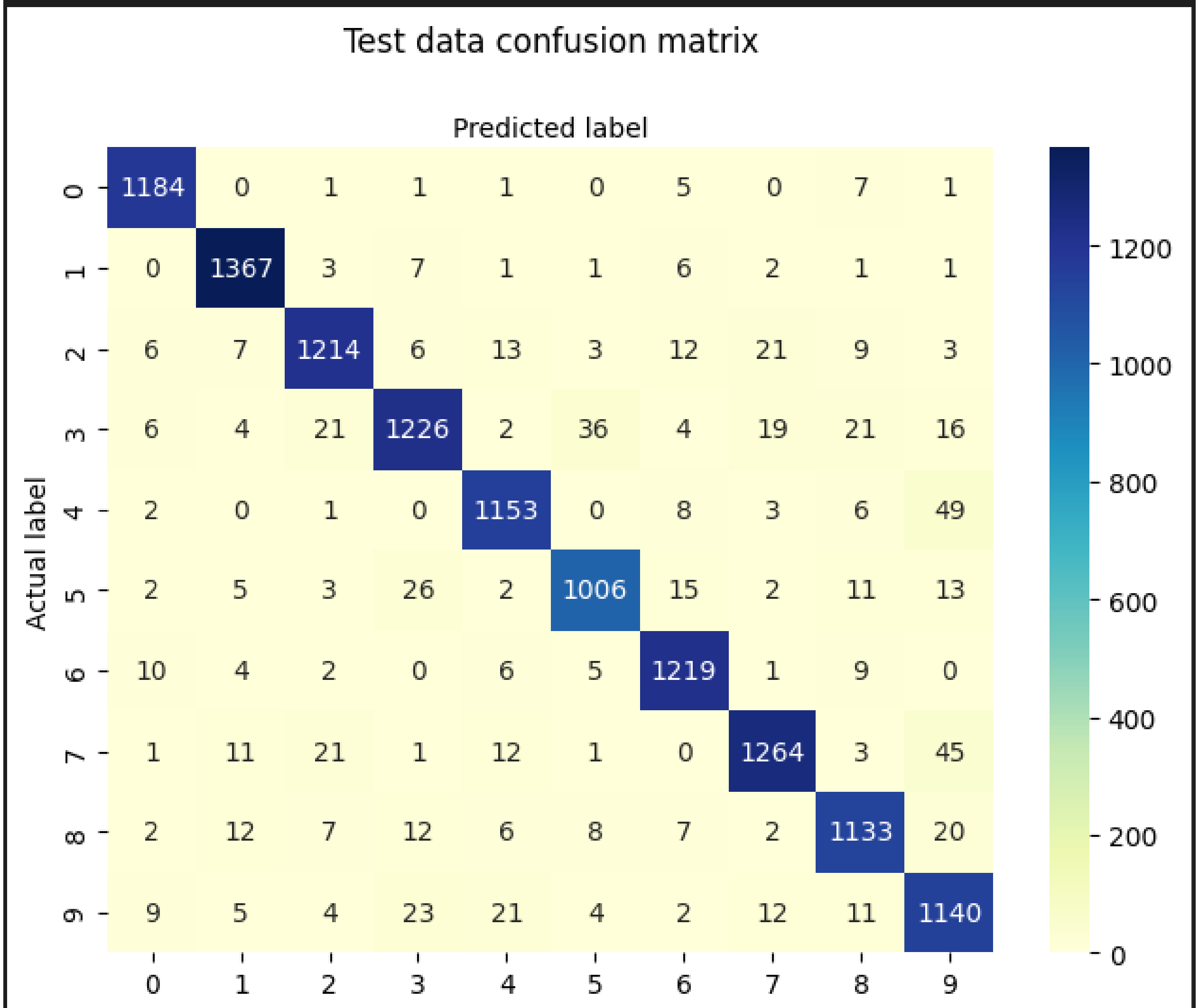

Figure 14

Random Forest Classifier Prediction

```
test_df = pd.read_csv('MNIST_Digit/test.csv')
# Remove any columns with constant values. They won't contribute to the
classifiers
clean_test_df = test_df.copy()
black_pixels = []
white_pixels = []
for pixel in df.columns:
if max(df[pixel]) == 0:
black_pixels.append(pixel)
if min(df[pixel]) == 255:
white_pixels.append(pixel)

clean_test_df = clean_test_df.drop(black_pixels, axis=1)
clean_test_df = clean_test_df.drop(white_pixels, axis=1)
print(len(clean_df.columns))
print(len(clean_test_df.columns)) # should be 1 less than clean_df because
there is no label column
```

```
709 708
```

Figure 15

```
# Standard Scale
scaler = StandardScaler()
predict_scaled = scaler.fit_transform(clean_test_df)

predictions = tuned_rand_forest.predict(predict_scaled)
imageId = pd.Series(range(1, len(predictions)+1)).astype(int)
result = {'ImageId': imageId, 'Label': predictions}
result_df = pd.DataFrame(result)
result_df.to_csv('MNIST_Digit/rand_forest_prediction.csv',index=False)
```

PCA + Random Forest Classifier

```
from sklearn.decomposition import PCA
from datetime import datetime
```

```
pca = PCA(n_components=0.95)
start = datetime.now()
pca.fit(x_train)
end = datetime.now()
print(end-start)

0:00:12.680576
```

Figure 16

```
x_train_pca_transform = pca.transform(x_train)
x_test_pca_transform = pca.transform(x_test)

print(x_train_pca_transform.shape)
print(x_test_pca_transform.shape)

(29400, 153) (12600, 153)
```

Figure 17

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler

# Standard Scale
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train_pca_transform)
x_test_scaled = scaler.fit_transform(x_test_pca_transform)

rand_forest = RandomForestClassifier()

# Parameter Grid
param_grid = {
'n_estimators': [10, 100],
'max_features': ['sqrt', 'log2'],
'max_depth': [4, 6, 8, 10],
```

```
'criterion': ['gini', 'entropy']
}
```

Figure 18

```
# Grid Search
kf = KFold(n_splits=5, shuffle=True, random_state=42)
grid_search = GridSearchCV(rand_forest, param_grid, cv=kf, n_jobs=-1)
grid_search.fit(x_train_scaled, y_train)
```

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
▸ RandomForestClassifier
```

Figure 19

```
from datetime import datetime

# Train on tuned logistic regression
best_params = grid_search.best_params_
tuned_rand_forest_pca = RandomForestClassifier(**best_params)
start = datetime.now()
tuned_rand_forest_pca.fit(x_train_scaled, y_train)
end = datetime.now()
print(end-start)

# hyperparameter tuning
# 1. n_estimators (number of trees)
# 2. max_features (maximum features considered for splitting a node)
# 3. max_depth (maximum number of levels in each tree)
```

```
# 4. splitting criteria (entropy or gini)

0:01:34.620560
```

Figure 20

```
# Random Forest w/PCA Best Params
best_params

{'criterion': 'entropy',
 'max_depth': 10,
 'max_features': 'sqrt',
 'n_estimators': 100}
```

Figure 21

```
# Predict
rand_forest_pca_y_train_pred =
tuned_rand_forest_pca.predict(x_train_scaled)
train_conf_matrix = confusion_matrix(y_train,
rand_forest_pca_y_train_pred)
print("Confusion Matrix (Training Data):\n", train_conf_matrix)
print("\nClassification Report (Training Data):\n",
classification_report(y_train, rand_forest_pca_y_train_pred))

# On testing data
rand_forest_pca_y_test_pred = tuned_rand_forest_pca.predict(x_test_scaled)
test_conf_matrix = confusion_matrix(y_test, rand_forest_pca_y_test_pred)
print("\nConfusion Matrix (Testing Data):\n", test_conf_matrix)
print("\nClassification Report (Testing Data):\n",
classification_report(y_test, rand_forest_pca_y_test_pred))

Confusion Matrix (Training Data): [[2898 0 2 6 1 1 13 1 9 1] [ 1 3256 10
11 0 2 6 4 5 0] [ 7 6 2788 9 7 0 1 16 45 4] [ 0 1 27 2896 1 11 5 12 32 11]
[ 1 4 7 0 2793 0 6 4 5 30] [ 8 0 1 18 2 2656 11 1 9 4] [ 6 0 4 1 4 9 2855
0 1 1] [ 0 12 32 2 3 0 0 2963 10 20] [ 4 14 11 39 6 23 5 6 2733 13] [ 1 1
2 31 23 4 0 26 17 2852]] Classification Report (Training Data): precision
```

```
recall f1-score support 0 0.99 0.99 0.99 2932 1 0.99 0.99 0.99 3295 2 0.97
0.97 0.97 2883 3 0.96 0.97 0.96 2996 4 0.98 0.98 0.98 2850 5 0.98 0.98
0.98 2710 6 0.98 0.99 0.99 2881 7 0.98 0.97 0.98 3042 8 0.95 0.96 0.96
2854 9 0.97 0.96 0.97 2957 accuracy 0.98 29400 macro avg 0.98 0.98 0.98
29400 weighted avg 0.98 0.98 0.98 29400 Confusion Matrix (Testing Data):
[[1161 0 2 6 4 2 14 1 9 1] [ 0 1361 4 2 1 7 8 2 4 0] [ 13 10 1146 31 21 4
9 16 42 2] [ 8 4 23 1203 1 26 8 18 46 18] [ 1 8 9 1 1116 2 14 6 8 57] [ 7
2 5 39 13 982 22 4 6 5] [ 24 2 6 1 6 17 1197 0 3 0] [ 2 20 23 2 16 1 0
1254 5 36] [ 4 9 13 52 7 28 13 4 1056 23] [ 5 3 6 24 34 10 0 49 10 1090]]
Classification Report (Testing Data): precision recall f1-score support 0
0.95 0.97 0.96 1200 1 0.96 0.98 0.97 1389 2 0.93 0.89 0.91 1294 3 0.88
0.89 0.89 1355 4 0.92 0.91 0.91 1222 5 0.91 0.91 0.91 1085 6 0.93 0.95
0.94 1256 7 0.93 0.92 0.92 1359 8 0.89 0.87 0.88 1209 9 0.88 0.89 0.89
1231 accuracy 0.92 12600 macro avg 0.92 0.92 0.92 12600 weighted avg 0.92
0.92 0.92 12600
```

Figure 22

```
# import required modules for performance evaluation
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score

# Training Data confusion matrix for random forest
class_names=[0,1] # name of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(train_conf_matrix), annot=True,
cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
```

```
plt.title('Training data confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

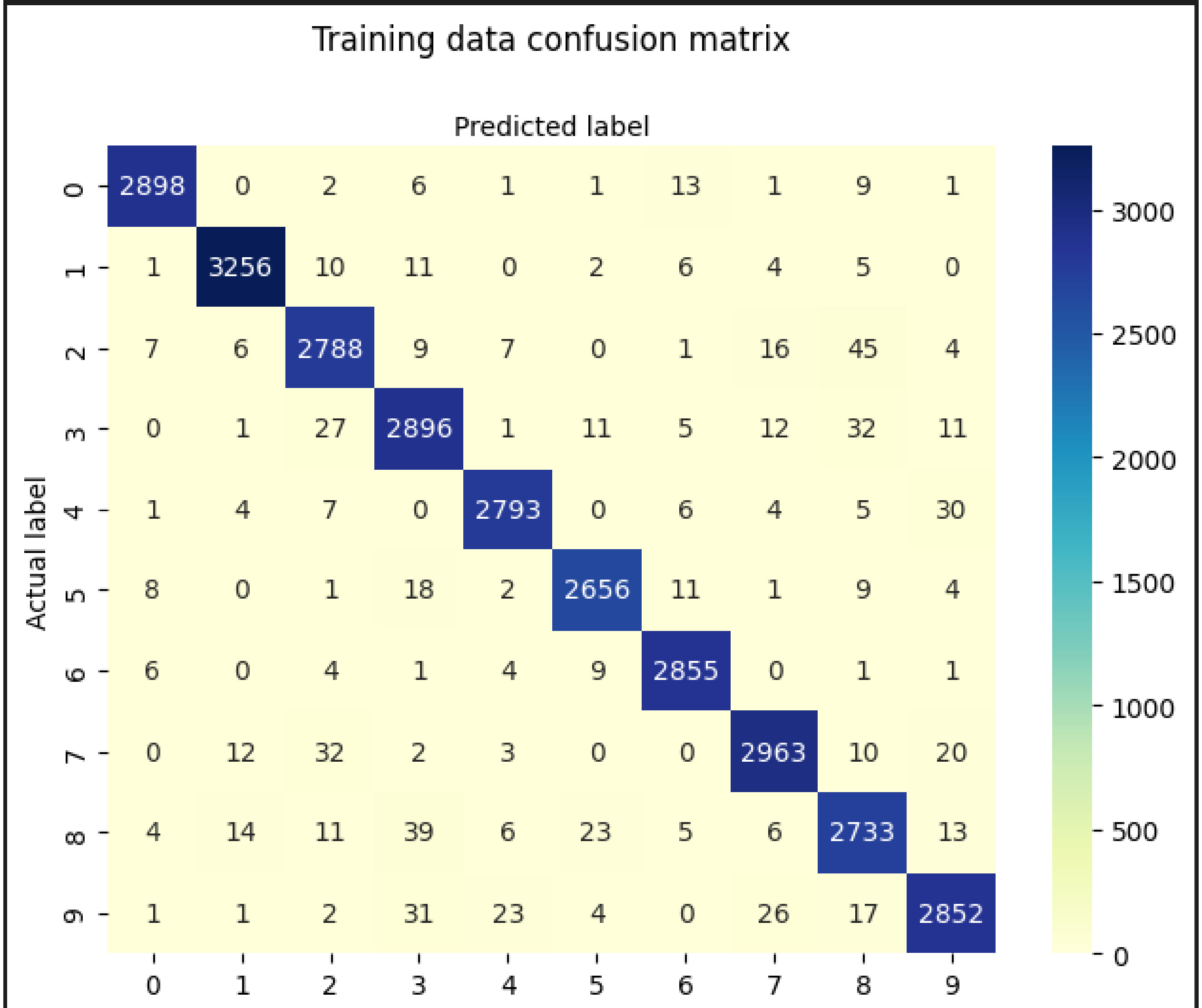

Figure 23

```
# Testing Data confusion matrix for random forest
class_names=[0,1] # name of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
```

```
sns.heatmap(pd.DataFrame(test_conf_matrix), annot=True,
cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Test data confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

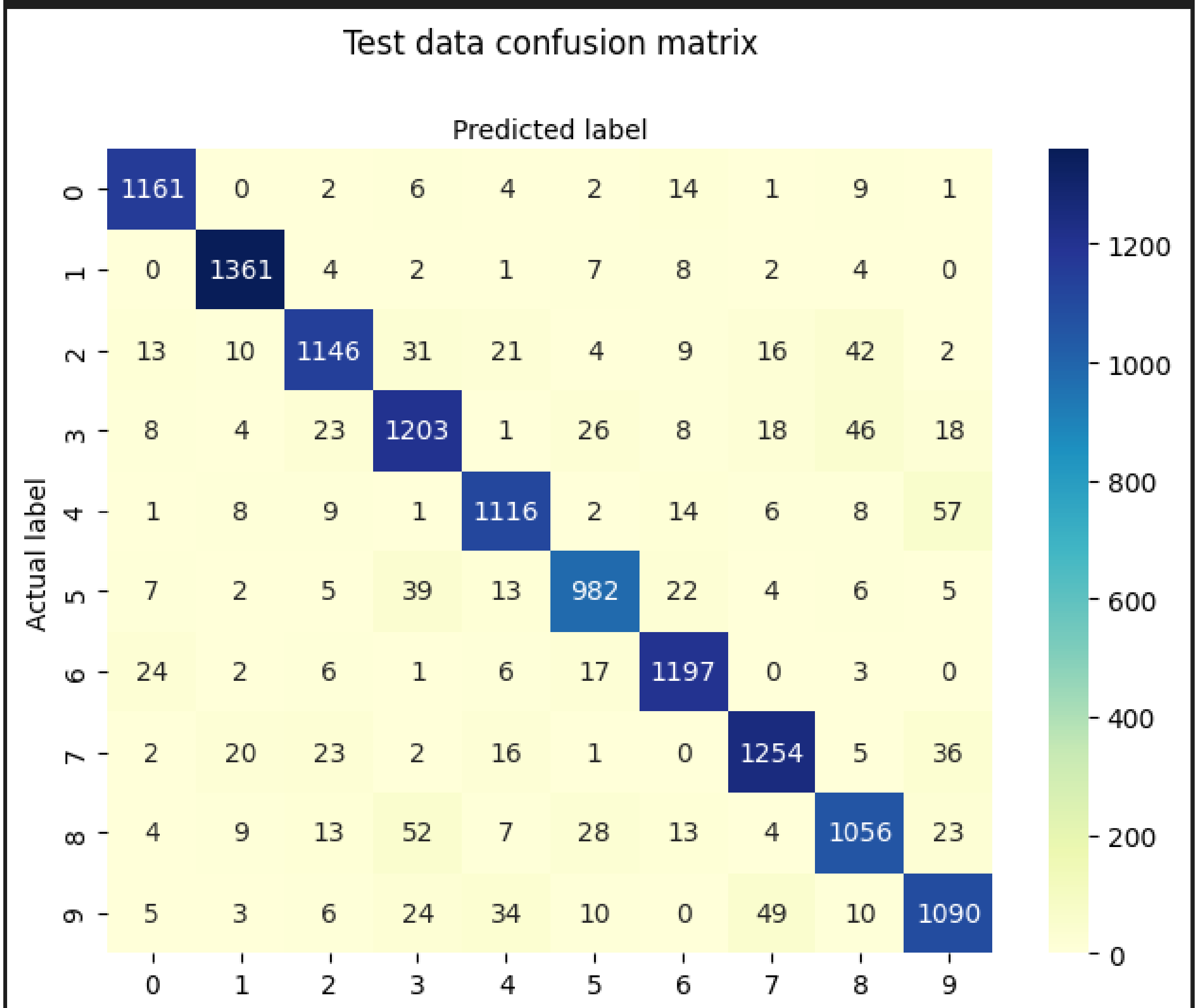

Figure 24

PCA + Random Forest Classifier Prediction

```
test_df = pd.read_csv('MNIST_Digit/test.csv')
```

```
# Remove any columns with constant values. They won't contribute to the
classifiers
clean_test_df = test_df.copy()
black_pixels = []
white_pixels = []
for pixel in df.columns:
if max(df[pixel]) == 0:
black_pixels.append(pixel)
if min(df[pixel]) == 255:
white_pixels.append(pixel)

clean_test_df = clean_test_df.drop(black_pixels, axis=1)
clean_test_df = clean_test_df.drop(white_pixels, axis=1)
print(len(clean_df.columns))
print(len(clean_test_df.columns)) # should be 1 less than clean_df because
there is no label column
```

```
709 708
```

Figure 25

```
# Standard Scale
pca_clean_test_df = pca.transform(clean_test_df)
scaler = StandardScaler()
predict_scaled = scaler.fit_transform(pca_clean_test_df)

predictions = tuned_rand_forest_pca.predict(predict_scaled)
imageId = pd.Series(range(1, len(predictions)+1)).astype(int)
result = {'ImageId': imageId, 'Label': predictions}
result_df = pd.DataFrame(result)
result_df.to_csv('MNIST_Digit/rand_forest_pca_prediction.csv',index=False)
```

K-Mean Clustering

```
from sklearn.cluster import MiniBatchKMeans
import numpy as np
from sklearn.metrics import silhouette_score
range_n_clusters = np.arange(100,500, 100)
```

```
silhouette_avg = []
for num_clusters in range_n_clusters:

# initialise kmeans
kmeans = MiniBatchKMeans(n_clusters=num_clusters, n_init='auto')
kmeans.fit(x_train)
cluster_labels = kmeans.labels_

# silhouette score
silhouette_avg.append(silhouette_score(x_train, cluster_labels))

plt.plot(range_n_clusters,silhouette_avg,'bx-')
plt.xlabel('Values of K')
plt.ylabel('Silhouette score')
plt.title('Silhouette analysis For Optimal k')
plt.show()
```

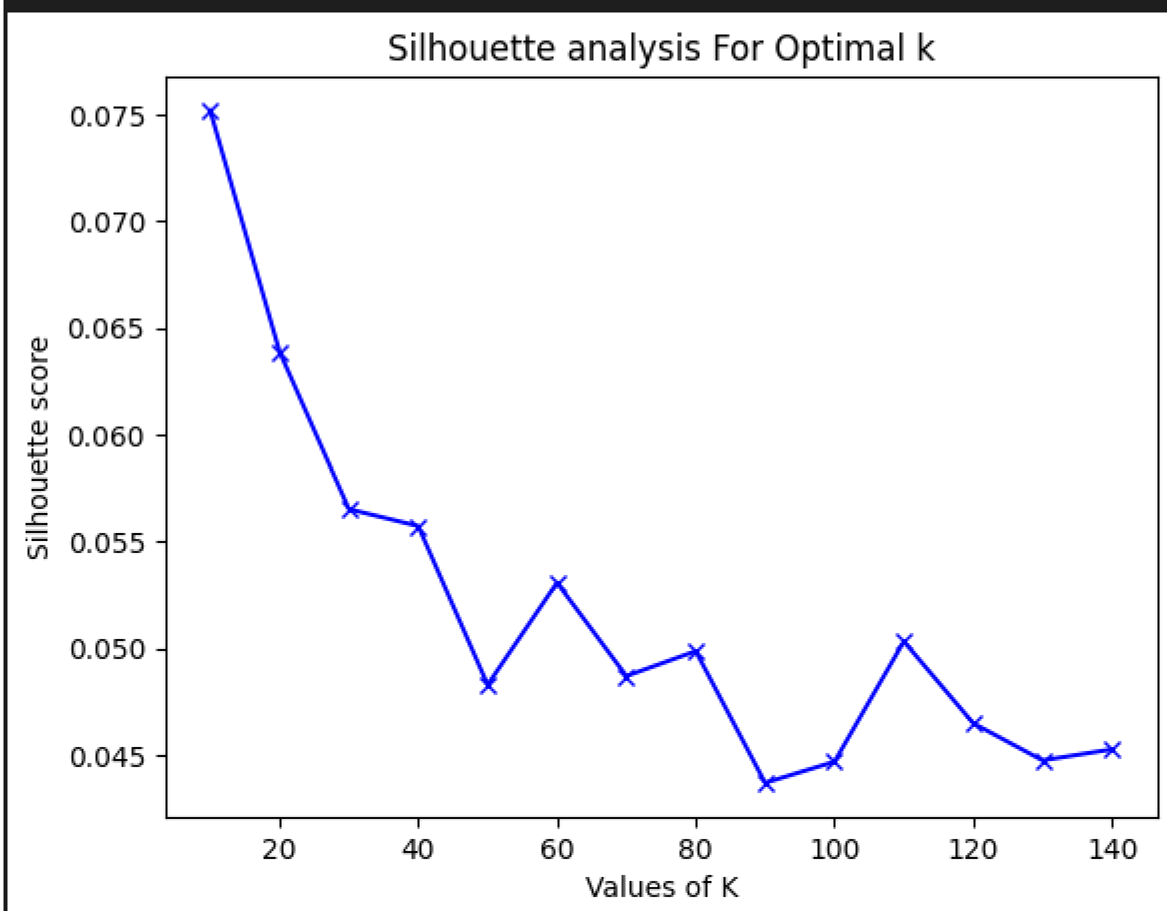

Figure 26

```
range_n_clusters = np.arange(100,500, 100)
silhouette_avg = []
for num_clusters in range_n_clusters:

# initialise kmeans
kmeans = MiniBatchKMeans(n_clusters=num_clusters, n_init='auto')
kmeans.fit(x_train)
cluster_labels = kmeans.labels_

# silhouette score
silhouette_avg.append(silhouette_score(x_train, cluster_labels))

plt.plot(range_n_clusters,silhouette_avg,'bx-')
plt.xlabel('Values of K')
plt.ylabel('Silhouette score')
plt.title('Silhouette analysis For Optimal k')
plt.show()
```

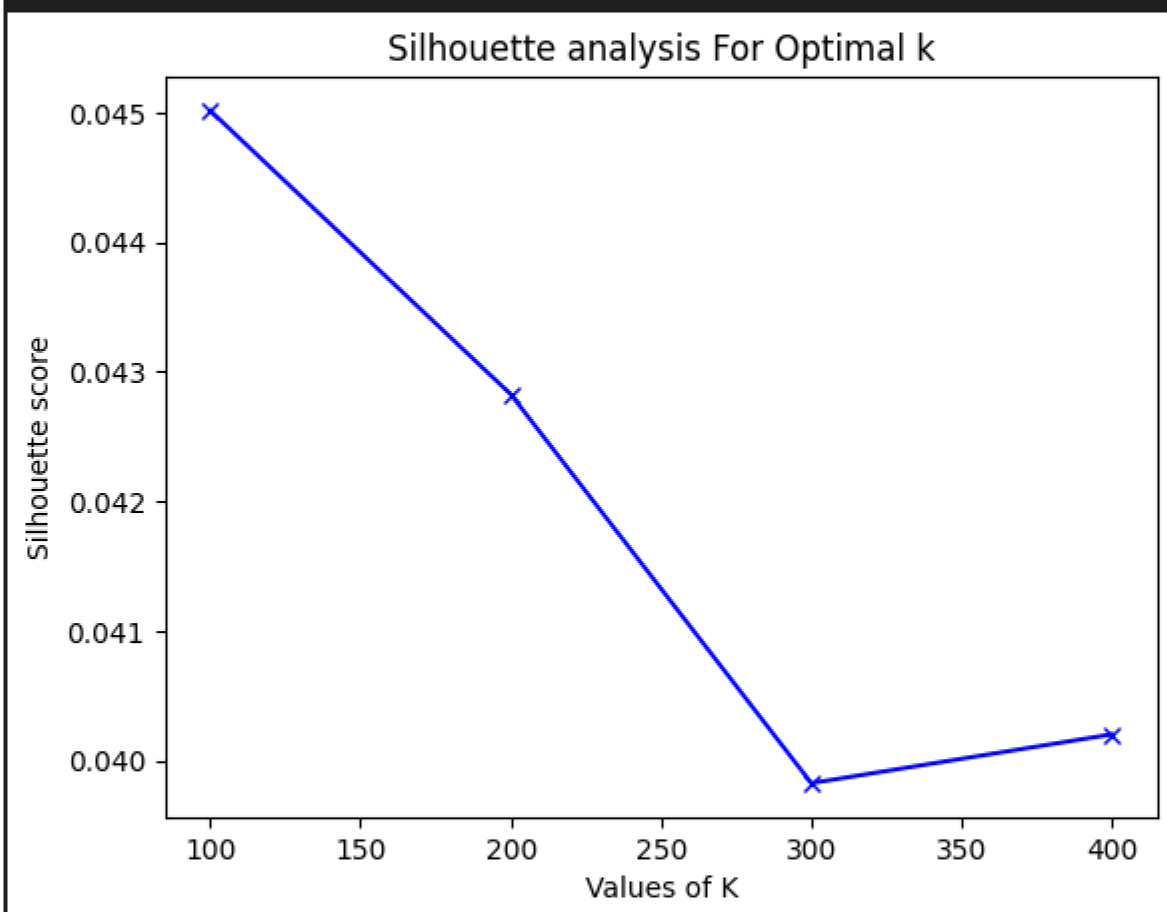

Figure 27

```
kmeans = MiniBatchKMeans(n_clusters = 2000, random_state=42,n_init='auto')
# Fit the model to the training data
kmeans.fit(x_train)
kmeans.labels_

array([1543,  358,  710, ..., 1037,   97,  980], dtype=int32)
```

Figure 28

```
# Majority Vote method to map k-mean label to actual y label
from collections import defaultdict

cluster_labels_train = kmeans.labels_

cluster_to_label = defaultdict(lambda: defaultdict(int))
for cluster_label, true_label in zip(cluster_labels_train, y_train):
cluster_to_label[cluster_label][true_label] += 1

# Step 4: Assign cluster labels based on the best match
cluster_majority_label = {}
for cluster_label, label_counts in cluster_to_label.items():
majority_label = max(label_counts, key=label_counts.get)
cluster_majority_label[cluster_label] = majority_label

y_pred = np.array([cluster_majority_label[cluster_label] for cluster_label
in cluster_labels_train])

print(y_pred[:20])
print(y_train[:20])

[6 5 3 4 7 8 6 7 0 9 9 7 6 9 9 3 1 6 3 0] 34941 6 24433 5 24432 3 8832 4
30291 7 28009 8 27876 6 120 7 30457 0 4634 9 13579 9 16089 7 7438 6 6879 9
9480 9 11189 3 30759 1 18444 6 11788 3 17052 0 Name: label, dtype: int64
```

Figure 29

```
# import required modules for performance evaluation
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import confusion_matrix, classification_report
# Predict
train_conf_matrix = confusion_matrix(y_train, y_pred)
print("Confusion Matrix (Training Data):\n", train_conf_matrix)
```

```
Confusion Matrix (Training Data): [[3269 0 7 2 1 10 21 0 4 2] [ 1 3721 14
4 5 4 6 7 3 10] [ 26 9 3179 20 4 2 14 42 27 8] [ 4 6 36 3185 2 72 8 14 64
23] [ 1 21 5 0 3015 0 27 24 2 138] [ 9 4 12 114 11 2849 49 1 29 15] [ 14 5
7 0 5 16 3298 0 7 0] [ 2 29 28 2 31 0 0 3273 8 135] [ 12 17 22 67 34 115
22 8 2891 40] [ 6 5 11 17 143 25 6 68 19 3050]]
```

Figure 30

```
# Training Data confusion matrix for random forest
class_names=[0,1] # name of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(train_conf_matrix), annot=True,
cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title(f'Training data confusion matrix of {len(set(kmeans.labels_))}
clusters', y=1.1)
```

```
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

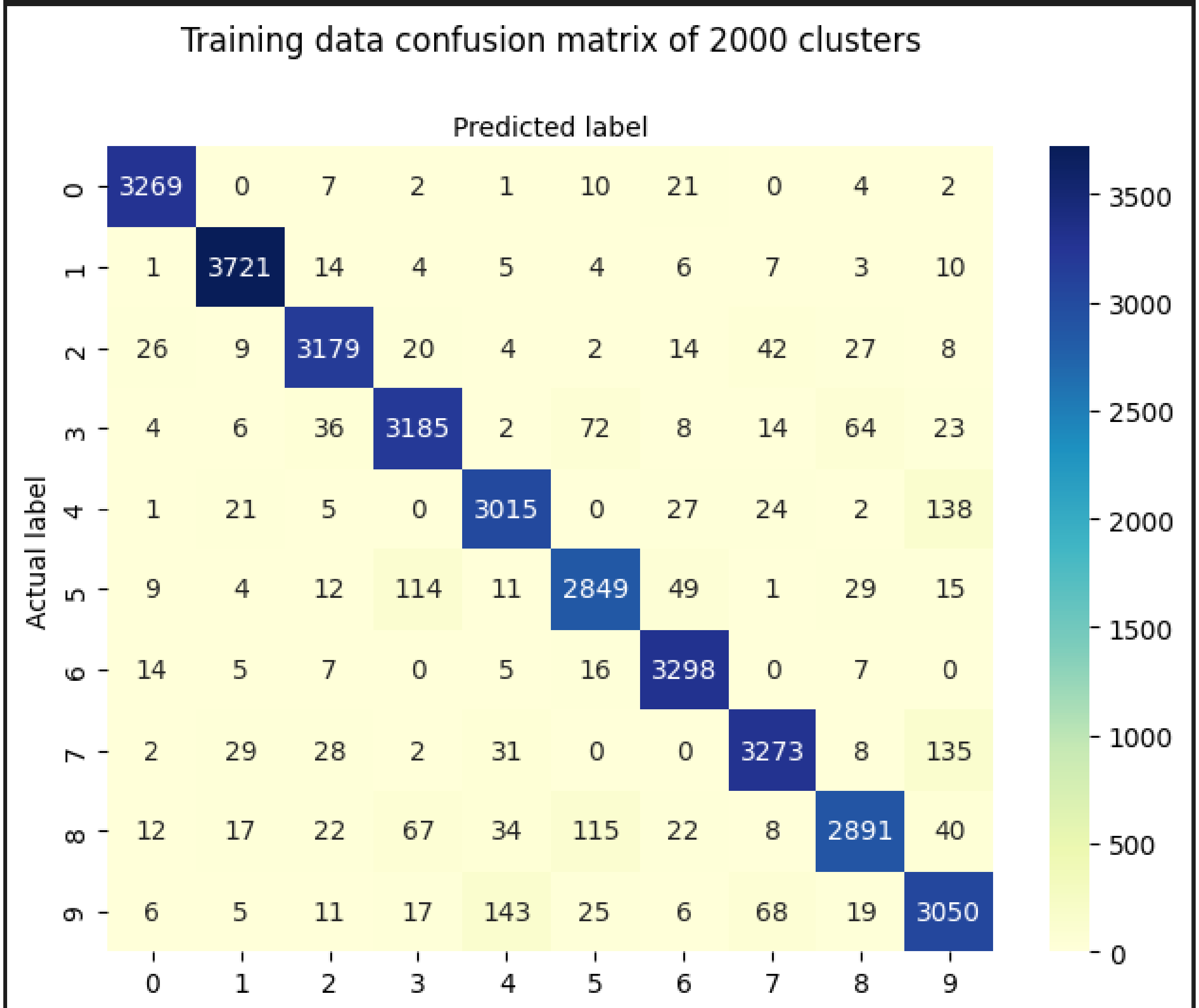

Figure 31

```
from sklearn.metrics import adjusted_rand_score,
normalized_mutual_info_score, silhouette_score

# Assuming you already have y_train and y_pred

# Calculate Adjusted Rand Index
ari = adjusted_rand_score(y_train, y_pred)
```

```
# Calculate Normalized Mutual Information
nmi = normalized_mutual_info_score(y_train, y_pred)

print("Adjusted Rand Index:", ari)
print("Normalized Mutual Information:", nmi)
```

```
Adjusted Rand Index: 0.8832843829515762 Normalized Mutual Information:
0.8719751665793185
```

Figure 32

```
# Check on testing data
cluster_labels_test = kmeans.predict(x_test)
y_pred_test = np.array([cluster_majority_label[cluster_label] for
cluster_label in cluster_labels_test])

test_conf_matrix = confusion_matrix(y_test, y_pred_test)
print("Confusion Matrix (Training Data):\n", test_conf_matrix)
```

```
Confusion Matrix (Training Data): [[806 0 1 0 1 1 7 0 0 0] [ 0 900 1 0 3 1
2 1 0 1] [ 10 8 784 5 3 3 6 15 8 4] [ 0 3 6 867 1 25 1 5 20 9] [ 3 2 1 0
770 1 10 2 1 49] [ 0 1 1 18 0 661 9 1 8 3] [ 5 1 0 0 1 2 774 0 2 0] [ 0 14
10 0 3 0 0 817 2 47] [ 1 4 5 22 8 29 4 2 749 11] [ 4 4 3 5 26 8 1 23 6
758]]
```

Figure 33

```
# Training Data confusion matrix for random forest
class_names=[0,1] # name of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(test_conf_matrix), annot=True,
cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
```

```
plt.tight_layout()
plt.title(f'Testing data confusion matrix of {len(set(kmeans.labels_))}', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
```

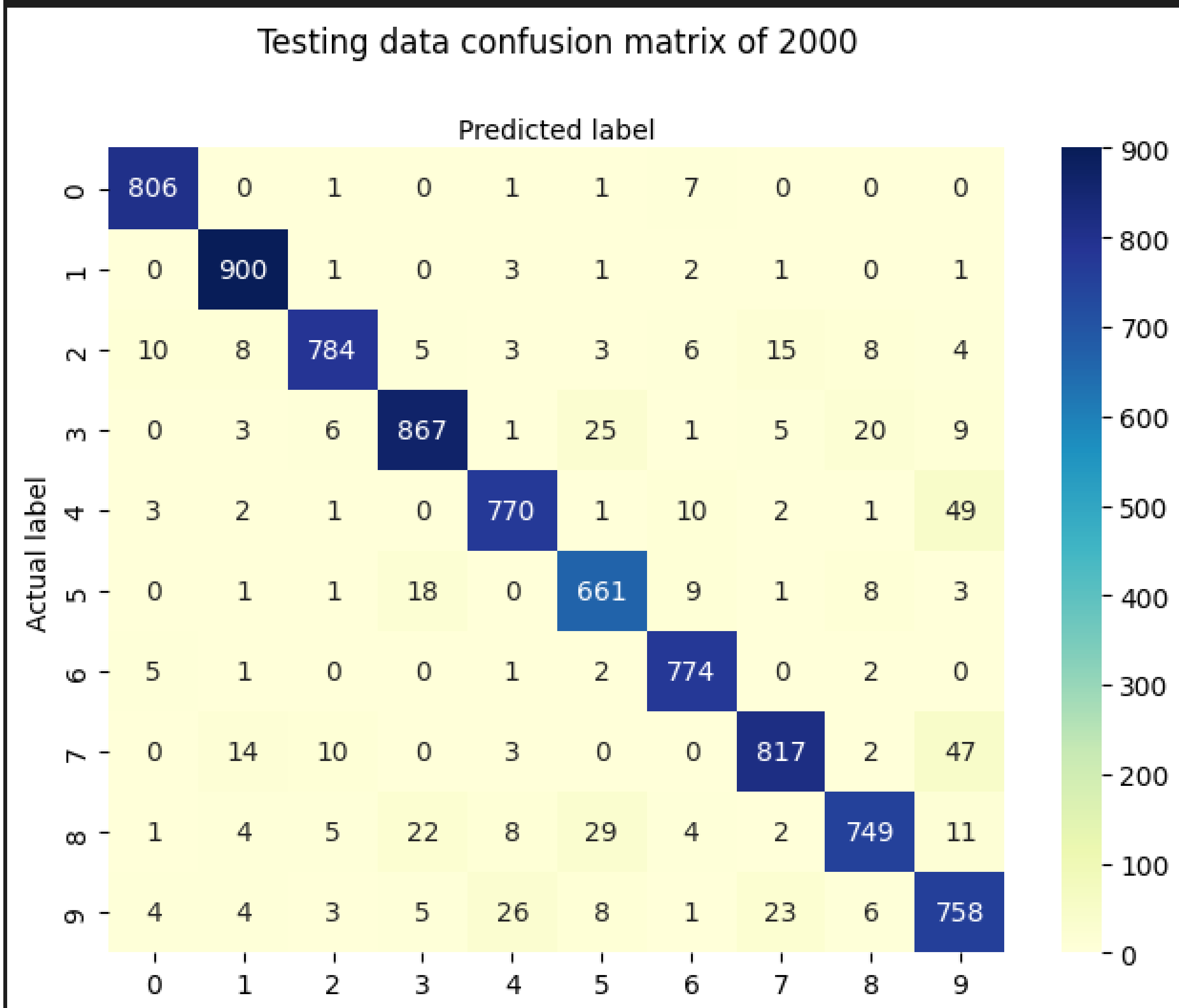

Figure 34

K-Means Prediction on Test Dataset

```
# Make prediction on test dataset
test_df = pd.read_csv('MNIST_Digit/test.csv')
```

```
# Remove any columns with constant values. They won't contribute to the
classifiers
clean_test_df = test_df.copy()
black_pixels = []
white_pixels = []
for pixel in df.columns:
if max(df[pixel]) == 0:
black_pixels.append(pixel)
if min(df[pixel]) == 255:
white_pixels.append(pixel)

clean_test_df = clean_test_df.drop(black_pixels, axis=1)
clean_test_df = clean_test_df.drop(white_pixels, axis=1)
print(len(clean_df.columns))
print(len(clean_test_df.columns)) # should be 1 less than clean_df because
there is no label column

709 708
```

Figure 35

```
# predict test data label
cluster_labels_final = kmeans.predict(clean_test_df)
y_pred_final = np.array([cluster_majority_label[cluster_label] for
cluster_label in cluster_labels_final])
imageId = pd.Series(range(1, len(y_pred_final)+1)).astype(int)
result = {'ImageId': imageId, 'Label': y_pred_final}
result_df = pd.DataFrame(result)
result_df.head()
```

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

Figure 36

```
result_df.to_csv('MNIST_Digit/kmeans_prediction_2000.csv',index=False)
```

Figure 37 - Random Forest Kaggle Score

Username - zacharycmiel

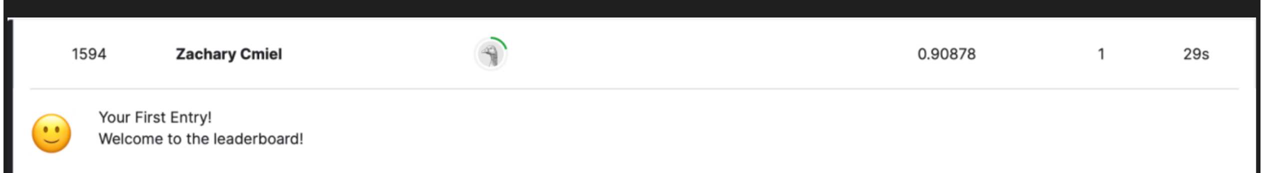

Figure 38 - Random Forest w/ PCA Kaggle

Score

Username - zacharycmiel

1568	Zachary Cmiel	0.91742	3	29s

Your Best Entry!
Your submission scored 0.91585, which is not an improvement of your previous score. Keep trying!

Figure 39 - K Means Kaggle Score

Username - JZHAO8

1539 **JZHAO8** 0.93785

Your Best Entry!
Your most recent submission scored 0.93785, which is an improvement of your previous score of 0.91550. Great job!

Tweet this