Victory_PCA_Core_Assignment

April 21, 2025

1 Task

Your task is to perform PCA to speed up a classification algorithm on a high-dimensional dataset. You will fit a model on the original scaled data, and a different one on data after transformation using a PCA model. You will compare the computation time and the evaluation scores.

We will use the MNIST digits dataset, which comes pre-installed in sklearn. This dataset has 28x28 pixel images of handwritten digits 0-9. Your task is to classify these to determine which digits they are.

Use PCA to lower the dimensions in this dataset while retaining 95% of the variance. You can do this when instantiating the PCA by giving the `n_components=` argument a float between 0 and 1.

[24]:
```
# Importing Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report
```

[3]:
```
# Initializing pca
pca = PCA(n_components=.95)
```

1. Load the Data

You can load the dataset using this code:

[4]:
```
# load the dataset
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784')
```

```
# view the shape of the dataset
mnist.data.shape
```

[4]:
```
(70000, 784)
```

The dataset has shape (70000, 784), meaning we are working with 70,000 images with 784 dimensions!

Note

- You can access the X features data using mnist.data.
- And, you can access the y target using mnist.target.

[10]:
```
# Displaying the X features
x = mnist.data
x.head()
```

[10]:
```
   pixel1  pixel2  pixel3  pixel4  …  pixel781  pixel782  pixel783  pixel784
0       0       0       0       0  …         0         0         0         0
1       0       0       0       0  …         0         0         0         0
2       0       0       0       0  …         0         0         0         0
3       0       0       0       0  …         0         0         0         0
4       0       0       0       0  …         0         0         0         0

[5 rows x 784 columns]
```

[20]:
```
# Displaying Y features
y = mnist.target
y.head()
```

[20]:
```
0    5
1    0
2    4
3    1
4    9
Name: class, dtype: category
Categories (10, object): ['0', '1', '2', '3', …, '6', '7', '8', '9']
```

[21]:
```
y.unique()
```

[21]:
```
['5', '0', '4', '1', '9', '2', '3', '6', '7', '8']
Categories (10, object): ['0', '1', '2', '3', …, '6', '7', '8', '9']
```

2. Prepare the Data

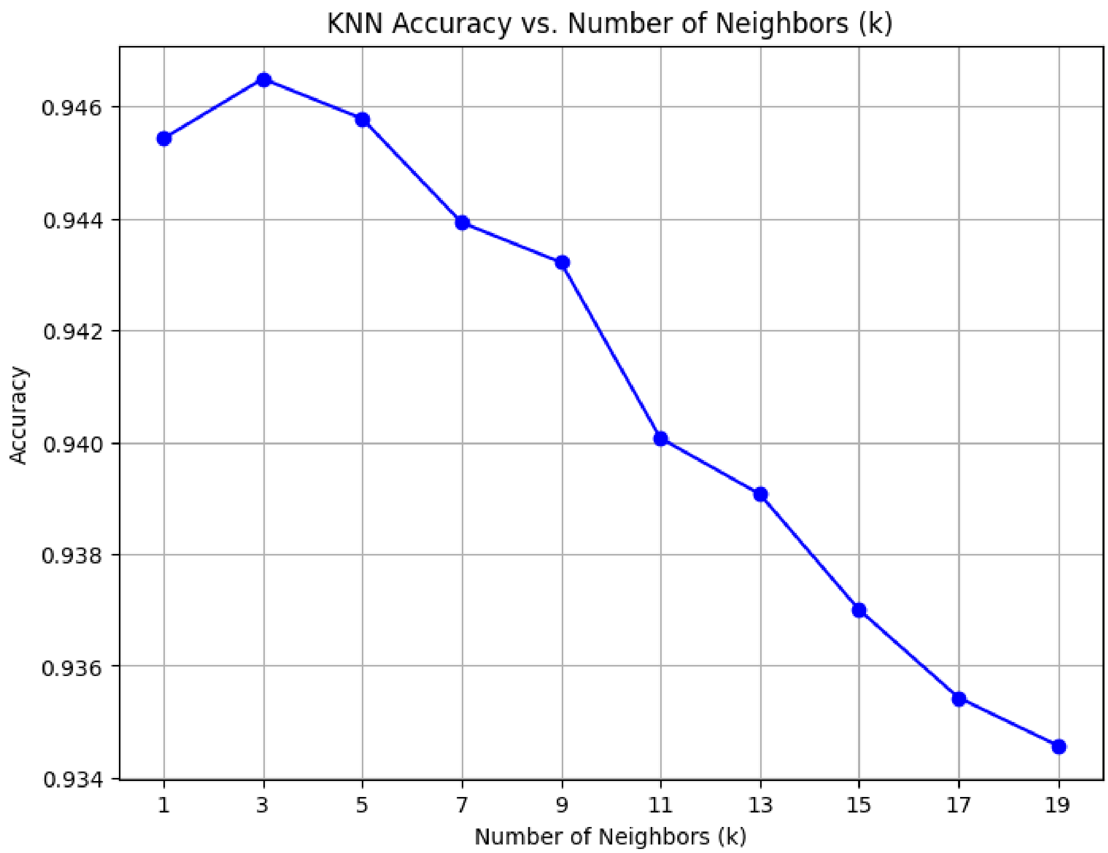
Prepare the data for modeling. Scale and apply PCA to your data, while retaining 95% of the variance. Be sure not to leak information.

[22]:
```python
# Spliting the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
↪random_state=42)

# Scaling x_train and x_test
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

# Applying PCA
x_train_pca = pca.fit_transform(x_train_scaled)
x_test_pca = pca.transform(x_test_scaled)
```

3. Create 2 KNN models.

- One that uses the PCA transformed data to predict which number each image shows.
- One that uses the original data, without the PCA transformation (but, remember you still need to scale the data!)

[26]:
```python
# Finding the best K value
k_values = range(1, 21, 2)

# Store accuracies for each k value
accuracies = []

# Train and test KNN with different k values
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train_scaled, y_train)
    y_pred = knn.predict(x_test_scaled)
    accuracies.append(accuracy_score(y_test, y_pred))

# Plotting the graph
plt.figure(figsize=(8, 6))
plt.plot(k_values, accuracies, marker='o', linestyle='-', color='b')
plt.title('KNN Accuracy vs. Number of Neighbors (k)')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.xticks(k_values)
plt.grid(True)
plt.show()


"""
From the graph, it appears that the highest accuracy is achieved when the␣
↪number of neighbors (k) is 3.
"""
```



[27]:
```
# KNN model for One that uses the PCA transformed data to predict which number␣
↪each image shows.
knn_pca = KNeighborsClassifier(n_neighbors=3)
knn_pca.fit(x_train_pca, y_train)
```

[27]: KNeighborsClassifier(n_neighbors=3)

[28]:
```
# KNN model for One that uses the original data, without the PCA transformation
knn_original = KNeighborsClassifier(n_neighbors=3)
knn_original.fit(x_train_scaled, y_train)
```

[28]: KNeighborsClassifier(n_neighbors=3)

4. Evaluate and compare the models.

Use separate cells to make predictions using each model. Include the cell magic command: `%%time` at the top of your cells when making predictions to see which model can create predictions faster, the one trained on PCA data or the one trained on non-PCA data. Evaluate both models using multiple appropriate metrics.

'%%time' will output under the cell a count of how long it takes the code in that cell to run.

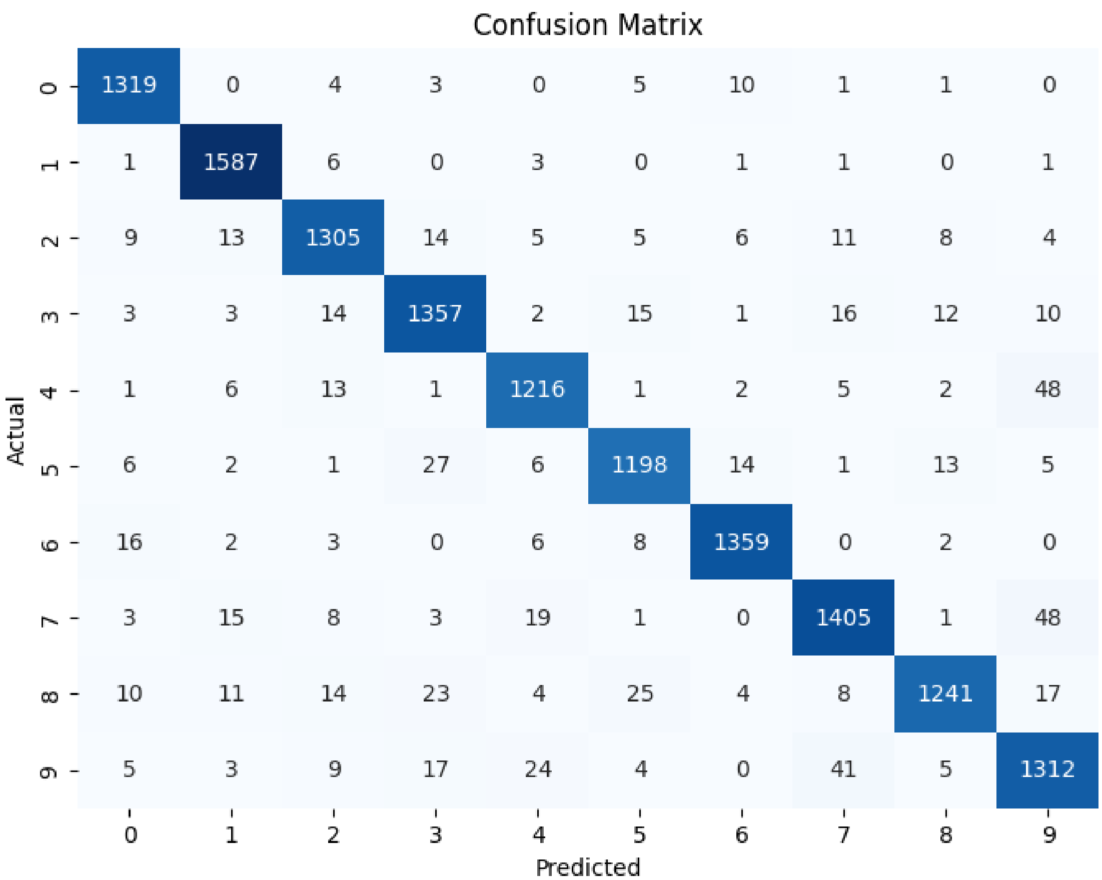
[29]:
```
# Trained model with pca
%%time
y_pred_pca = knn_pca.predict(x_test_pca)
```

```
CPU times: user 24 s, sys: 29.6 ms, total: 24 s
Wall time: 24.1 s
```

[33]:
```
# Evaluating the model with pca
cm = confusion_matrix(y_test, y_pred_pca)

# Classification Report
cr = classification_report(y_test, y_pred_pca)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97      1343
           1       0.97      0.99      0.98      1600
           2       0.95      0.95      0.95      1380
           3       0.94      0.95      0.94      1433
           4       0.95      0.94      0.94      1295
           5       0.95      0.94      0.95      1273
           6       0.97      0.97      0.97      1396
           7       0.94      0.93      0.94      1503
           8       0.97      0.91      0.94      1357
           9       0.91      0.92      0.92      1420

    accuracy                           0.95     14000
   macro avg       0.95      0.95      0.95     14000
weighted avg       0.95      0.95      0.95     14000
```

[36]:
```
# Heatmap showing the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



[30]:
```
# Original data
%%time
y_pred_original = knn_original.predict(x_test_scaled)
```

```
CPU times: user 51.3 s, sys: 149 ms, total: 51.4 s
Wall time: 52.6 s
```

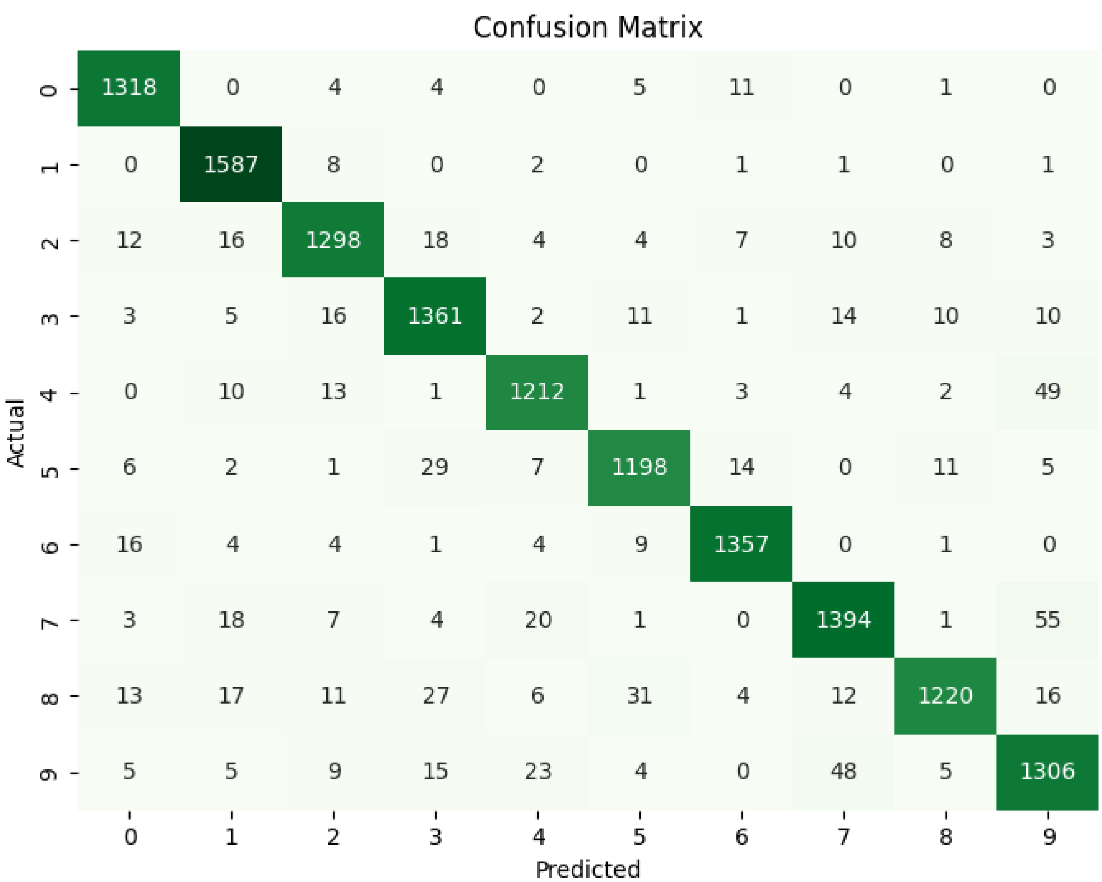
[37]:
```
# Evalutaing model with original data
cm = confusion_matrix(y_test, y_pred_original)

# Classification Report
cr = classification_report(y_test, y_pred_original)
print(cr)
```

```
              precision    recall  f1-score   support

           0       0.96      0.98      0.97      1343
           1       0.95      0.99      0.97      1600
           2       0.95      0.94      0.94      1380
           3       0.93      0.95      0.94      1433
```

```
           4       0.95      0.94      0.94      1295
           5       0.95      0.94      0.94      1273
           6       0.97      0.97      0.97      1396
           7       0.94      0.93      0.93      1503
           8       0.97      0.90      0.93      1357
           9       0.90      0.92      0.91      1420

    accuracy                           0.95     14000
   macro avg       0.95      0.95      0.95     14000
weighted avg       0.95      0.95      0.95     14000
```

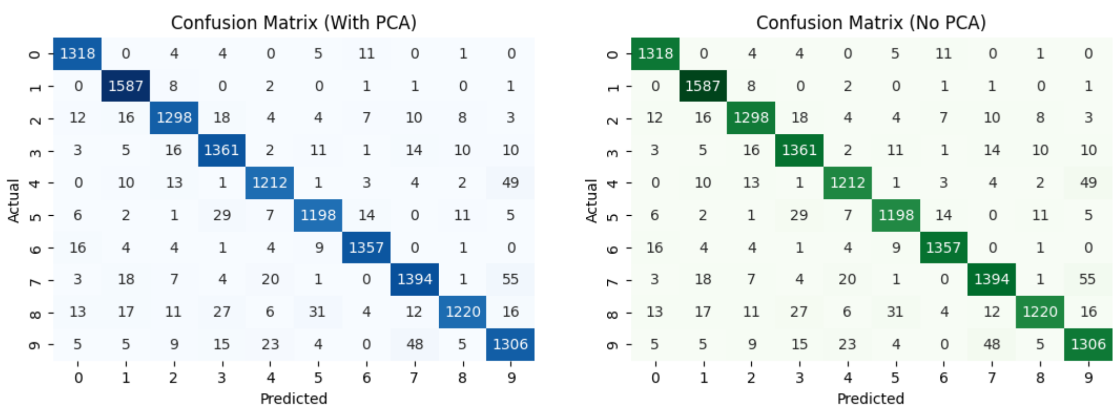
[38]:

```python
# Heatmap showing the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



[42]:

```python
# Creating a subplot
fig, axes = plt.subplots(1, 2, figsize=(13, 4))

# model with pca
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, ax=axes[0])
axes[0].set_xlabel('Predicted')
axes[0].set_ylabel('Actual')
axes[0].set_title('Confusion Matrix (With PCA)')

# model with no pca
sns.heatmap(cm, annot=True, fmt='d', cmap='Greens', cbar=False, ax=axes[1])
axes[1].set_xlabel('Predicted')
axes[1].set_ylabel('Actual')
axes[1].set_title('Confusion Matrix (No PCA)')
plt.show()
```



5. Answer the Following Questions in the Text:

- Which model performed the best on the test set?
- Both models had the same performance ie achieving the same accuracy and producing an equal number of correct predictions across all classes, as indicated by the matching values along the main diagonal of their respective confusion matrices.
- Which model was the fastest at making predictions?
- The model with PCA

[ ]: