Week 5

PCA & KNN

1.1. Objectives

In this laboratory session you will use Principal Component Analysis (PCA) and the k-nearest neighbour (KNN) algorithms to perform automatic recognition of human handwriting. Specifically, you will use PCA to automatically extract a set of features, and KNN to classify digits from the MNIST dataset using the features. Your objectives are:

1. Load the MNIST images, and learn about the CI tools within the scikit-learn package
2. Perform PCA on the training set and evaluate the variance distribution
3. Use the principal components from objective 2 to create a KNN classifier
4. Extract the same principal components from the test data set, and classify them using the KNN classifier
5. Evaluate the overall performance and speed, and experiment with the different parameters

At the end of this laboratory session you should have a working handwriting recognition system that you can directly compare to the neural network system in the previous labs. You will also have a working demonstration of both PCA and KNN. You are free to modify or extend this in any way you want for your future coursework.

1.2. Introduction

We will be the same tools as the previous laboratory (i.e Visual Studio Code & the Anaconda Python distribution). Follow the instructions from Laboratory 1 to create a new project, be sure to select Anaconda as the Python environment to run.

MNIST

We will be using the same MNIST database as Week 4, as a reminder MNIST is a large database of handwritten digits that is commonly used for training various image processing systems. The MNIST database contains 60,000 training images and 10,000 testing images. There have been many scientific papers on attempts to achieve the lowest error rate; one paper, using a hierarchical system of convolutional neural networks, manages to get an error rate on the MNIST database of 0.23%.

MNIST Files

The MNIST database in CSV format is available to download from Moodle. There are four files, the full test and training sets, and the smaller subsets of 100 training and 10 test sets.

Scikit-learn

In the laboratories so far, you have created most of the functional code yourselves, making use of libraries such as numpy and matplotlib to perform useful functions. This has been a worthwhile exercise because it enables a far greater level of understanding than simply using someone else's code. However, Computational Intelligence and machine learning are very popular areas of research and there are many Python libraries that provide you with working examples of structures such as neural networks. In this laboratory, we will be using the PCA algorithm and the KNN classifier. We could write the code for these from scratch, but it would become very long and quite tedious. Instead we will make use of a very popular Python library called scikit-learn. Scikit-learn is a machine learning library that includes various classification, regression and clustering algorithms including PCA and KNN. There is plenty of documentation available on the scikit-learn website if you wish to explore the algorithms further.

1.3. Getting Started

If you have not already done so, create a new Python project using the instructions given in the lab from week 1.

Principal Component Analysis

Our first step is to look at the code that reads in the MNIST data and performs PCA. In code listing 6.1 the two CSV files are loaded, making use of a numpy file read command. Next the data and labels are identified and a PCA object is initiated with a parameter that sets the number of components to extract. The training data is then fed into the PCA object, once the principal components have been extracted it is possible to plot the variance that is represented by the components (if the components represented all the variance, then the sum of the variances would be 1).

PCA

```
# Numpy for useful maths
import numpy
# Sklearn contains some useful CI tools
# PCA
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
# k Nearest Neighbour
from sklearn.neighbors import KNeighborsClassifier
# Matplotlib for plotting
import matplotlib.pyplot as plt

# Load the train and test MNIST data
train = numpy.loadtxt('mnist_train_100.csv', delimiter=',')
test = numpy.loadtxt('mnist_test_10.csv', delimiter=',')

# Separate labels from training data
train_data = train[:, 1:]
train_labels = train[:, 0]
test_data = test[:, 1:]
test_labels = test[:, 0]

# Select number of components to extract
pca = PCA(n_components = 10)
# Fit to the training data
pca.fit(train_data)

# Determine amount of variance explained by components
print("Total Variance Explained: ", numpy.sum(pca.explained_variance_ratio_))

# Plot the explained variance
plt.plot(pca.explained_variance_ratio_)
plt.title('Variance Explained by Extracted Componenents')
plt.ylabel('Variance')
plt.xlabel('Principal Components')
plt.show()
```

Code 6.1: PCA.py

k-Nearest Neighbour

Once we have setup our PCA, we can move on the classification stage. Code listing 6.2 shows the next steps, the training and test data are both transformed using PCA into a list of principal component scores, these scores are then normalised. Next, we create a KNN classifier with the training principal component scores and labels as the training data, note that the KNN is initiated with $k = 5$ and the $p = 2$ (Euclidean) norm. Finally, we can now use the KNN object to predict the classification for the test data. As with the previous laboratories a scorecard is created, and the percentage success rate is calculated.

KNN

```
# Extract the principal components from the training data
train_ext = pca.fit_transform(train_data)
# Transform the test data using the same components
test_ext = pca.transform(test_data)

# Normalise the data sets
min_max_scaler = MinMaxScaler()
train_norm = min_max_scaler.fit_transform(train_ext)
test_norm = min_max_scaler.fit_transform(test_ext)

# Create a KNN classification system with k = 5
# Uses the p2 (Euclidean) norm
knn = KNeighborsClassifier(n_neighbors=5, p=2)
knn.fit(train_norm, train_labels)

# Feed the test data in the classifier to get the predictions
pred = knn.predict(test_norm)

# Check how many were correct
scorecard = []

for i, sample in enumerate(test_data):
    # Check if the KNN classification was correct
    if round(pred[i]) == test_labels[i]:
        scorecard.append(1)
    else:
        scorecard.append(0)
    pass

# Calculate the performance score, the fraction of correct answers
scorecard_array = numpy.asarray(scorecard)
print(”Performance = ”, (scorecard_array.sum() / scorecard_array.size) * 100, ’ % ’)
```

Code 6.2: KNN.py

1.4. Exercises

Spend some time familiarising yourself with the code for extracting the principal component scores and the KNN classifier. You do not need to fully understand the internal workings, but you should understand the basic flow of the code. Here are some exercises to try:

1. Try increasing the number of principal components, how many are needed to explain all the variance within the training data?
2. How many principal components do you think is enough to perform effective classification?
3. What is the best performance you can get on the MNIST100/10 data set? What happens if you change the parameters *k* and *p*?
4. How does your best performance compare to the neural network from laboratory four?
5. What can you say about the time taken to run this type of classifier?
6. Using the code you have been provided, use PCA to reduce the number of features and then use a neural network to classify the MNIST characters. Does this give you the best result?