Optimisation and Data Analytics Project Report

Omar Ali Sheikh-Omar
Department of Engineering
Aarhus University

Abstract—In this report, I analyse the performance characteristics of five classification algorithms on the ORL database of face images and the MNIST database of images of handwritten digits. Using different types of visualisations, I also explain why classification accuracy drops significantly when Principal Component Analysis is used to project the original data set into a two-dimensional feature space. Out of the five classifiers, I found that the nearest neighbour classifier was the most versatile as it performed very well on all data sets.

Index Terms—Machine Learning, Image Classification, Visualisation

1 Introduction

Classifying images into different categories is easy for humans because our visual system is highly efficient at recognising patterns in images. Even when we look at cluttered scenes where the salient object is occluded, deformed, or otherwise varied in terms of size and illumination, we are able to quickly identify it. However, computers struggle with this seemingly mundane task.

The image classification problem is the task of assigning a label to an image from a set of predefined labels based on the content of the image. A classical computer system that can solve this problem typically comprise three steps; preprocessing, feature extraction and classification [1].

In order to make the subsequent steps easier and reduce the variations in the images, it is important to preprocess an image before forwarding it to the next stage in the pipeline. For instance, we can scale an image so all images have the same dimensions. We may also want to crop the image and centre the important object in the image.

Next step involves preserving distinguishing aspects of the image and ignoring irrelevant ones. The aim of feature extraction is to represent an input image in a way that is similar to the other images that belong in the same category. Feature extraction is dependent on the problem domain. For instance, in face recognition applications certain characteristics such as the scale and the orientation of the face in the image should be disregarded when extracting features. In the domain of handwriting recognition, the representation of an input image should ignore the stroke size of the pen.

Once an image representation is generated in the feature extraction step, it passed onto to a classifier which is essentially a function. This function is trained on a data set that contains a label for each sample in this set. Depending on the type of the classifier and the quality of the training set, the trained classifier can become good at discerning patterns that are inherent in the images in each category.

Modern state-of-the-art image classifiers are based on a special type of neural networks called Convolutional Neural Network (CNN). A CNN is a multi-layer neural network that consists of a number of alternating convolutional layers and subsampling layers followed by one or more fully-connected layers. The convolutional and subsampling layers provide automatic feature extraction and the fully connected layers perform the classification. These networks have repeatably produced exceptional results achieving near human-level classification accuracy. Figure 1 depicts a typical CNN architecture that achieves 99.75% accuracy on the MNIST data set.

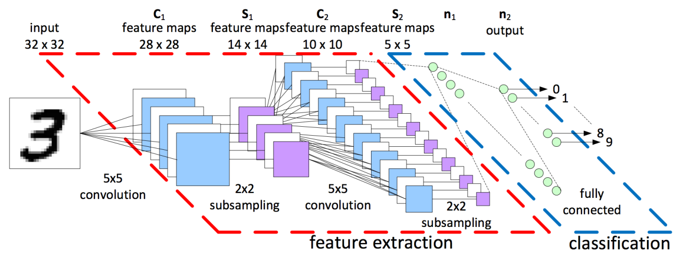

Fig. 1. A typical CNN for the MNIST image classification problem. Source: https://bit.ly/2JD8fyi.

2 Assignment and Data Sets

The project assignment was to implement and compare the performance of five classification algorithms on two image databases; the ORL database of faces and the MNIST database of handwritten digits.

The ORL database consists of 400 face images of 40 different people that were taken between April 1992 and April 1994 at the Olivetti Research Laboratory in Cambridge [2]. Each image is a grey scale image of dimensions 30×40. As can be observed from Figure 2, there are variations in each image. Some of the face images were taken while the subject was smiling, wearing glasses or have their eyes closed. There are also slight variations in face orientation and the overall illumination of the image.

The other image database is the popular MNIST that is often used to benchmark and compare different classification methods [3]. This database consists of 70,000 images of handwritten digits. The samples are randomly split, so 60,000 of the images are in the training set and the remaining are in the test set. Each image is 28 by 28 and the digits

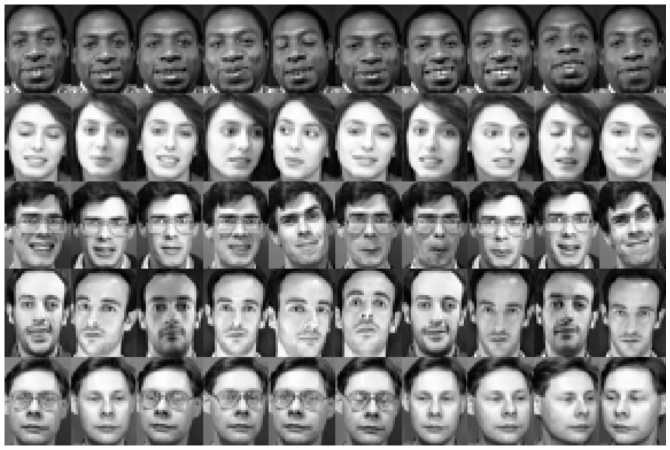

Fig. 2. Ten facial images of five subjects

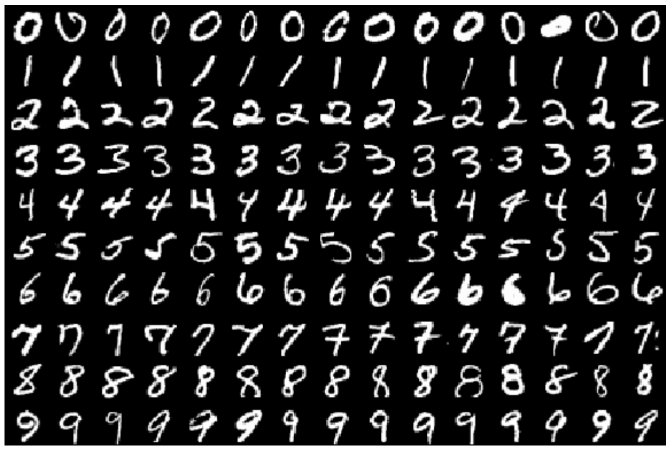

Fig. 3. A selection of MNIST digits

are centred in the image. Figure 3 shows a selection of the digits in the MNIST database. There are some variations within each class. For examples, the shape of the digits and thickness of the pen varies.

3 Classifiers

Before outlining the experiments, this section briefly describes the different classifiers that are used in this project. In the following subsections, following notation are used:

- N is the number of samples in the training set
- K is the number of classes in the training set
- N_k is the number of samples in class k
- $T = \{(\boldsymbol{x}_1, l_1), \cdots, (\boldsymbol{x}_N, l_N)\}$ is the training set where $\boldsymbol{x}_i \in \mathbb{R}^D$ is ith sample and $l_i \in \{c_1, \cdots, c_K\}$ is the label of $\boldsymbol{x}_i$.
- $\boldsymbol{x}'$ is an unseen sample from the test set

3.1 Nearest Class Centroid

The nearest class centroid (NCC) classifier assigns a label to a new sample based on the label of the closest centroid. The method consists of two steps. First, it represents the samples in each class by a single vector $\boldsymbol{\mu}_k$ called the centroid. The computation of this vector is relatively straightforward. Suppose $\mathbf{C}_k = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_{N_k}\}$ is a set of N_k samples in class k. The centroid for class k can be computed as follows:

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i=1}^{N_k} \boldsymbol{x}_i \tag{1}$$

A centroid can be seen as a summary of the samples in a class. Once it is computed, NCC disregards the shape and the distribution of the training samples. In the probabilistic point of view, NCC can be viewed as a special case of the probability-based classifier if we make two assumptions. First, the samples in each class k follow a unimodal normal distribution centred at $\boldsymbol{\mu}_k$ with the identity as the covariance matrix:

$$p(\boldsymbol{x}_i \mid k) \sim N(\boldsymbol{\mu}_k, \mathbf{I}) \tag{2}$$

where $\boldsymbol{x}_i \in \mathbf{C}_k$. Second, all classes have equal probability i.e., $P(k) = 1/K$.

Classification of a new sample $\boldsymbol{x}'$ is performed by computing the distances between the new sample and all the centroids, and then assigning $\boldsymbol{x}'$ the label of the closest centroid. Various distance functions can be used depending on the data set. Typically, we use the Manhattan distance or the Euclidean distance. Both of these distance functions can be seen as a generalisation of the Minkowski distance:

$$d(\boldsymbol{x}', \boldsymbol{\mu}_k) = \left(\sum_{d=1}^{D} |\boldsymbol{x}'_d - \boldsymbol{\mu}_{kd}|^p \right)^{1/p} \tag{3}$$

When p is set to 1 or 2 then Equation 3 becomes the Manhattan distance or the Euclidean distance respectively.

3.2 Nearest Subclass Centroid

Some data sets may contain several clusters of samples within a given class. To accommodate such cases, Nearest Subclass Centroid (NSC) is able to represent each class by m centroids where m is a hyperparameter of the algorithm specified by the user. Typically, the user does not know beforehand the number of clusters within each class. Grid search can be used to find a good value of m for a given data set.

Calculating the centroids is similar to NCC. Suppose $\mathbf{C}_{km} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_{N_{km}}\}$ is the set of samples in the class k and subclass m. The centroid or the mean vector of the samples in each subclass can be computed as follows:

$$\boldsymbol{\mu}_{km} = \frac{1}{N_{km}} \sum_{i=1}^{N_{km}} \boldsymbol{x}_i \tag{4}$$

where $\boldsymbol{x}_i \in \mathbf{C}_{km}$. Since $\mathbf{C}_{km}$ is unknown, we can use a clustering algorithm like k-Means on the samples in each class k to find the subclasses.

The classification method is similar to the NCC classifier. First, the distances between the new sample $\boldsymbol{x}'$ and all the centroids are calculated. Then, we find the centroid with the smallest distance and give $\boldsymbol{x}'$ the same label as this centroid. The NSC classifier is better at capturing groups of samples within each class than the NCC classifier. This means that the NSC will perform better on data sets where samples in each category do not form a single cluster. However, the price is an increase in processing time during the training and the classification.

3.3 Nearest Neighbour

One of the simplest classification algorithms is the nearest neighbour (NN) classifier. Given a new sample $\boldsymbol{x}'$, it computes the distance between the new sample and each sample in the training set. The new sample $\boldsymbol{x}'$ is assigned the same class as the closest sample in the training set. The NN classifier can be seen as an NSC classifier where m (the number of subclasses) is equal to the number of samples in the training set N [4, p.37].

NN does not make any assumptions about the samples in each class, which allows it to create a rather complicated decision boundary [1, p.178]. This comes with one major drawback. When the sample size is small, the classifier has a tendency to overfit i.e., it learns the training set too well and as a consequence makes poor predictions on new unseen samples. The generalised algorithm called k-nearest neighbour (kNN) combat this issue by comparing a new sample to a number $k > 1$ of nearest neighbours. Classification is performed by taking the k closest samples in the training set and assigning $\boldsymbol{x}'$ the label of the majority class. The value of k is a hyperparameter of the algorithm. Usually, we pick k to be an odd value because it works as a tiebreaker.

3.4 Perceptron

The Perceptron classifier is based on a linear discriminant function of the form:

$$g(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0 \tag{5}$$

where $\boldsymbol{x} \in \mathbb{R}^D$ is a sample in the training set, $\boldsymbol{w} \in \mathbb{R}^D$ is called the weight vector and w_0 is called the bias [4, p.41]. The vector $\boldsymbol{w}$ represents the normal of the hyperplane defined by $g(\boldsymbol{x}) = 0$ and controls the orientation of the hyperplane. The bias w_0 is the displacement of the hyperplane from the origin.

Since the underlying discriminant function classifies any sample $\boldsymbol{x}$ into one of two classes, the Perceptron is a binary classifier. If $g(\boldsymbol{x})$ is positive then $\boldsymbol{x}$ is assigned to the positive class because $\boldsymbol{x}$ has the same "general direction" as the weight vector $\boldsymbol{w}$. If $g(\boldsymbol{x})$ is negative then $\boldsymbol{x}$ assigned to the negative class because sample $\boldsymbol{x}$ is on the other side of the hyperplane. In the case where $g(\boldsymbol{x})$ is zero i.e., the sample $\boldsymbol{x}$ is on the hyperplane, the classification is unknown. To have a deterministic algorithm, we can choose to always classify samples on the hyperplane to belong to the positive class.

3.4.1 Linearly Separable Case

Initially, we do not know the weight vector $\mathbf{w}$ defining a hyperplane that perfectly discriminates the samples into two classes. We can estimate $\mathbf{w}$ by formulating and optimising an objective function. If we assume that the samples in the training set are linearly separable then we know there exists a hyperplane that separates all training samples into two classes. Suppose $l_i \in \{-1, 1\}$ is the binary label associated with the ith sample $\mathbf{x}_i$ in the training set. Once we have found an optimal weight vector $\mathbf{w}^*$, the following inequality will hold for all training samples:

$$l_i\tilde{\mathbf{w}}^{*T}\tilde{\mathbf{x}}_i \geq 0 \tag{6}$$

where $\tilde{\mathbf{x}} = [1, \mathbf{x}^T]$ and $\tilde{\mathbf{w}} = [w_0, \mathbf{w}^T]$ are the augmented vectors which are used for compact math expressions.

We can use this knowledge to define an objective function for finding the optimal weight vector. Suppose χ is the set of training samples that are misclassified by some $\tilde{\mathbf{w}}$ i.e., where $l_i\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}_i$ is negative. We can derive an objective function based on the the misclassified samples:

$$\mathcal{J}_p(\tilde{\mathbf{w}}) = \sum_{\tilde{\mathbf{x}}_i \in \chi} -l_i\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}_i \tag{7}$$

Essentially, the objective function $\mathcal{J}_p(\tilde{\mathbf{w}})$ is finding the sum of the distances between each of the misclassified samples and the discriminating hyperplane. When there are no misclassified samples i.e., χ is the empty set, then the optimal weight vector $\tilde{\mathbf{w}}^*$ is found, which makes $\mathcal{J}_p(\tilde{\mathbf{w}}^*)$ equal zero.

Optimising $\tilde{\mathbf{w}}$ can be done with a gradient-based iterative algorithm using the following update rule:

$$\begin{aligned}\tilde{\mathbf{w}}^{(t)} &= \tilde{\mathbf{w}}^{(t-1)} + \eta(t)\nabla\mathcal{J}_p & (8)\\ &= \tilde{\mathbf{w}}^{(t-1)} + \eta(t)\sum_{\tilde{\mathbf{x}}_i \in \chi} -l_i\tilde{\mathbf{w}}^T & (9)\end{aligned}$$

where $t > 0$ is the current epoch and $\eta(t) > 0$ is the learning rate which determines how much $\tilde{\mathbf{w}}$ is updated in each epoch.

The learning rate can be constant. Picking the right learning rate requires careful consideration. When the learning is too small, the algorithm needs more epochs to converge. On the other hand, if $\eta(t)$ is too large, the algorithm may overshoot and diverge [1, p.237]. One solution is to let the learning rate change during the learning process [1, p.313]. A simple example is a decaying learning rate where the learning rate is decreased with each epoch. The idea is to take larger steps towards the optimum at the beginning of the algorithm and reduce the step size when the algorithm is approaching convergence:

$$\eta(t) = \eta_0\frac{1}{1 + dt} \tag{10}$$

where η_0 is the initial learning rate and d is the decay rate.

3.4.2 Minimum Squared Error

Samples in some data sets may not be linearly separable. Given a non-linearly separable training set, the gradient-based algorithm with the Perceptron objective function $\mathcal{J}_p$ will never converge. The number of misclassified samples will never be zero and the hyperplane defined by $\mathbf{w}$ will keep shifting in each epoch, resulting in an algorithm that never converges.

Another approach is to define an objective function based on the least-squares problem [4, p.47]:

$$\mathcal{J}_{mse}(\tilde{\mathbf{w}}) = \|\tilde{\mathbf{X}}^T\tilde{\mathbf{w}} - \mathbf{b}\|_2^2 \tag{11}$$

where $\tilde{\mathbf{X}} = [\tilde{\mathbf{x}}_1, \cdots, \tilde{\mathbf{x}}_N]$ is a matrix of the augmented training samples and $\mathbf{b} \in \mathbb{R}^N$ is the target vector that contains the labels of each sample i.e., $b_i \in \{-1, 1\}$ expresses the label of the ith sample $\mathbf{x}_i$.

The new objective function $\mathcal{J}_{mse}(\tilde{\mathbf{w}})$ computes the squared error of using a given $\tilde{\mathbf{w}}$ on all the samples in the training set. When $\tilde{\mathbf{X}}$ is a full rank matrix, then the least-squares solution is given by [5, p.218]:

$$\tilde{\mathbf{w}}^* = (\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T)^{-1}\tilde{\mathbf{X}}\mathbf{b} \tag{12}$$

The least-squares solution in Equation 12 can be derived by taking the derivative of the objective function and setting it to zero.

For tall and thin matrices i.e., $D + 1 > N$, the product $\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T$ does not have an inverse. The ORL data set is an example of a tall and thin matrix. The number of dimensions in the ORL data set $D = 1200$ is larger than the number of samples $N = 400$ so we have $\mathbf{X}_{ORL} \in \mathbb{R}^{1200\times 400}$. Therefore, in the general case, we use a pseudo-inverse to solve the least-squares problem.

Let us take one step back and look at why we can use the pseudo-inverse. If we assume that $\tilde{\mathbf{X}}$ is an $N \times N$ matrix and its rank is N i.e., a square nonsingular matrix, then the least square solution would be:

$$\tilde{\mathbf{w}}^* = \tilde{\mathbf{X}}^{-1}\mathbf{b} \tag{13}$$

This means that in order to solve the least-squares problem, it is sufficient to know the inverse of $\tilde{\mathbf{X}}$. When the inverse does not exist, we can use pseudo-inverse $\tilde{\mathbf{X}}^\dagger$ in its stead:

$$\tilde{\mathbf{w}}^* = \tilde{\mathbf{X}}^\dagger\mathbf{b} \tag{14}$$

When $\tilde{\mathbf{X}}$ has full rank then it follows from Equation 12 that $\tilde{\mathbf{X}}^\dagger = (\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T)^{-1}\tilde{\mathbf{X}}$. In the case where $\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T$ is singular, we can employ a regularlised pseudo-inverse which makes the matrix invertable by adding a scaled version of the identity matrix:

$$\tilde{\mathbf{X}}^\dagger = \lim_{\epsilon\to 0}(\tilde{\mathbf{X}}\tilde{\mathbf{X}}^T + \epsilon\mathbf{I})^{-1}\tilde{\mathbf{X}} \tag{15}$$

where ϵ is a small value and $\mathbf{I}$ is the identity matrix. The value of ϵ depends on the training sample.

3.4.3 Multi-class Classification

Although the Perceptron is an inherently binary classifier, it can be used for classification tasks with more than two classes. Suppose $\mathbf{T} = \{(\mathbf{x}_1, l_1), \cdots, (\mathbf{x}_N, l_N)\}$ is the training set and $l_i \in \{c_1, \cdots, c_K\}$ is the label for $\mathbf{x}_i$. There are two strategies to combine binary classifiers to perform K-class classification task; One-versus-Rest and One-versus-One [4].

The One-versus-Rest strategy trains K binary classifiers. Each classifier g_k is trained using the entire training set to discriminate between samples belonging to positive class c_k and the samples belonging to the other classes c_m where $m \neq k$. To classify a new sample $\mathbf{x}'$, the output of each decision function $g_k(\mathbf{x}')$ are considered. Since the response of a decision function is actually the distance between to the input $\mathbf{x}'$ and the separating hyperplane, it is reasonable to classify $\mathbf{x}'$ to the class for which the decision function yields the highest output.

One-versus-One trains $K(K-1)/2$ binary classifiers where each classifier g_{km} is trained using a subset of the samples that belong to either class c_k or class c_m. It is faster to train a classifier since it only uses a subset of the data. On the downside, more classifiers must be trained. Classifying a new sample $\boldsymbol{x}'$ is based on majority voting. The new sample is tested on all classifiers and each classifier casts one vote on the class that it believes $\boldsymbol{x}'$ belongs to.

4 Principal Component Analysis

In this project, we are asked to evaluate the performance of the classifiers using the two-dimensional representation of the original dataset after applying Principal Component Analysis (PCA). This section gives a brief overview of PCA and how it works.

Given a set of N samples $\mathbf{X} = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ in $\mathbb{R}^D$, the aim of PCA is to find a lower-dimensional representation in $\mathbb{R}^d, d < D$ of the data samples that maximises the spread of each dimension [4]. Roughly speaking, maximising the variance reduces the likelihood of loosing important information when the samples are projected to a lower subspace. However, we are bound to lose information, when the dimensionality reduction is extreme e.g. going from 1200 dimensions to two dimensions.

PCA performs a linear projection to transform the original samples into a lower-dimensional subspace which is given by:

$$\mathbf{Y} = \mathbf{W}^T\mathbf{X} \tag{16}$$

where $\mathbf{Y} \in \mathbb{R}^{d\times N}$ is the transformed samples, $\mathbf{W} \in \mathbb{R}^{D\times d}$ is the projection matrix and $\mathbf{X} \in \mathbb{R}^{D\times N}$ is the original samples. An optimal projection matrix $\mathbf{W}^*$ that maximises the variance in each direction can be found by optimising the following objective function:

$$\mathbf{W}^* = \arg\max Tr(\mathbf{W}^T\mathbf{S}_T\mathbf{W}) \tag{17}$$

$$\text{subject to}: \mathbf{W}^T\mathbf{W} = \mathbf{I} \tag{18}$$

where $Tr(\cdot)$ is the trace operator that sums the diagonal entries of a matrix and $\mathbf{S}_T$ is the total scatter matrix of the centred samples given by:

$$\mathbf{S}_T = \bar{\mathbf{X}}\bar{\mathbf{X}}^T = (\mathbf{X} - \boldsymbol{\mu})(\mathbf{X} - \boldsymbol{\mu})^T \tag{19}$$

Maximising the the sum of the diagonal entries of the matrix $\mathbf{W}^T\mathbf{S}_T\mathbf{W}$ achieves the goal of maximising the variance of the transformed samples. The constraint in Equation 18 expresses the idea of having a subspace where the basis vectors are orthonormal. This ensures that the each dimension is uncorrelated to the other dimensions.

The solution to the optimisation problem can be found by performing eigenanalysis on the total scatter matrix $\mathbf{S}_T$. The optimal projection matrix $\mathbf{W}^*$ is formed by the eigenvectors sorted in descending order of the corresponding eigenvalues.

5 Method Description

In this section, I will detail the method that I used to conduct the experiments.

I opted to use Python with the libraries Scikit-Learn and NumPy to implement the seven classifiers. To save time, I chose to use the Scikit-Learn's default implementations of NCC and NN; the classes `NearestCentroid` and `KNeighborsClassifier` respectively. I implemented the NSC classifier using the `kMeans` algorithm in Scikit-Learn. Also, I implemented the two Perceptron classifiers using the One-versus-Rest classification strategy. The figures in this report are generated using Matplotlib and Seaborn.

For training the classifiers, I used the original data representations and two-dimensional outputs from

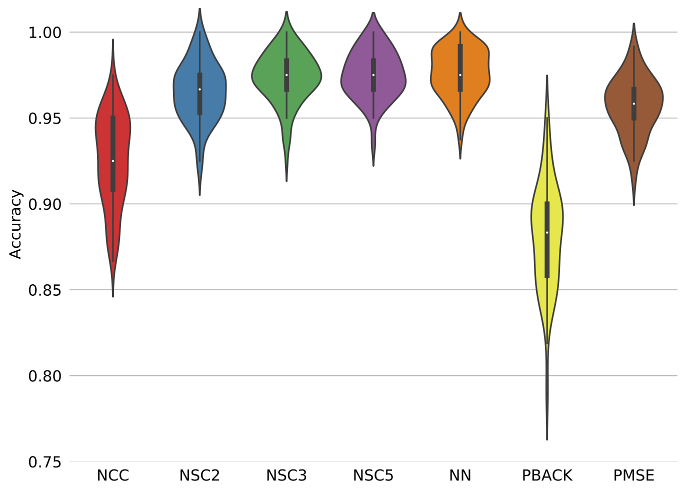

Fig. 4. Performance characteristics of the seven classifiers on the original ORL dataset. NCC is the Nearest Class Centroid classifier. NSC2, NSC3 and NSC5 are the Nearest Subclass Centroid classifiers with subclasses 2, 3 and 5 respectively. PBACK and PMSE are the Perceptron classifiers (PMSE uses the least-square solution).

PCA. To find good hyperparameters, I wrote a Python script (`hypertune.py`) that performs a grid search with 5-fold stratified cross-validation using Scikit-Learn's `GridSearchCV`. The result of each run was stored in a JSON file in the folder `params`. The best hyperparameters were picked based on the mean of the test scores.

To compare the performance of the seven classifiers on the original data sets and the PCA transformed 2D representations, each classifier is evaluated in the same manner to ensure a fair comparison. The `benchmark.py` script is responsible for performing the benchmark. It stores its results in the `benchmark_results` directory.

For convenience, the benchmark is performed using a stratified 3-fold cross-validation repeated 33 times. This gives me 99 performance measurements on each combination of seven classifiers and four data sets. The number of measurements may not be enough but I think it strikes a good balance. Although this approach takes a very long time to run, the measurements give me a good way to analyse the performance characteristics of the different classifiers. It is easier to see whether a classifier is able to learn the underlying mapping between the samples in one class and the corresponding class label even though it is trained with different training sets. Also, this approach allows me to create nice visualisations of the results for analysis and comparisons.

6 Results

This section describes the results obtained by running the benchmark script.

6.1 Original ORL

Figure 4 provides a graphical summary of the benchmark results collected by running the seven classifiers on the original ORL data set. In general, we see that the accuracy at different runs vary somewhat where some classifiers have a larger variance than others. The spread of the accuracy scores can be explained by the small sample size of the dataset. In the ORL database, each class has only ten samples where seven of them are used for training and the remaining three are used for testing. The seven randomly selected samples might not be representative of the whole population. To make this point more concrete, consider for example subject 7 in the ORL dataset. The person is wearing glasses, in seven of the face images while in the remaining three, he is without glasses. Suppose that in a given split, the training set consists only of the seven images where subject 7 is wearing glasses. In this scenario, a classifier may produce a suboptimal prediction because the test images of subject 7 might be more similar to other subjects than subject 7. If the sample size was much larger e.g. 10,000 then the probability of having a split where the training set only consists of samples that exhibit a certain characteristic — which is not present in the test set — is much smaller. This means that classifiers which are trained on a relatively small number of samples are more likely to produce a wider range of classification scores. This may explain the relatively large variances we see in the violin plots.

NCC assumes that the samples in each class come from a unimodal normal distribution and each class can, therefore, be represented by the mean of the samples in that class. This assumption may not work well for the ORL dataset considering its small sample size. As we discussed in the example with subject 7, representing a class with a single point may lead to an erroneous classification because the mean of another class may be closer to the samples in the test set.

Having subclasses improves the performance slightly and reduces the variability of the classifier results. One possible explanation is that each subclass describes the average of some prominent characteristics for a subject. During testing, a new sample is assigned to the label of the subclass with matching characteristics. NSC2, NSC3, NSC5 and NN seem to perform well. The violin plots for NSC3, NSC5 and NN are comparatively short, which suggest that these classifiers tend to generalise better on new samples.

The two Perceptron classifiers exhibit different performance characteristics. While the PMSE classifier performs only slightly worse than the best classifier, the accuracy measurements for the PBACK classifier varies from 0.78 to 0.95.

Due to the small sample size, it seems that the PBACK classifier is learning the training set too well and has difficulty generalising to the validation set. One explanation is that large variance is caused by the fact that the discriminating hyperplane found by the PBACK algorithm is arbitrary. Assuming samples in the training set are linearly separable, there are an infinite number of hyperplanes that separates the samples into two groups. Since the implemented PBACK algorithm does not have a built-in mechanism to maximise the margin like SVM, in some cases PBACK algorithm may find a hyperplane that is very close to the samples in either one of the classes when the iterative algorithm converges. In such cases, the algorithm may misclassify all the test samples for some subjects.

PMSE is not susceptible to these variations since the optimal weight vector $\tilde{\boldsymbol{w}}^*$ is found analytically. This explains

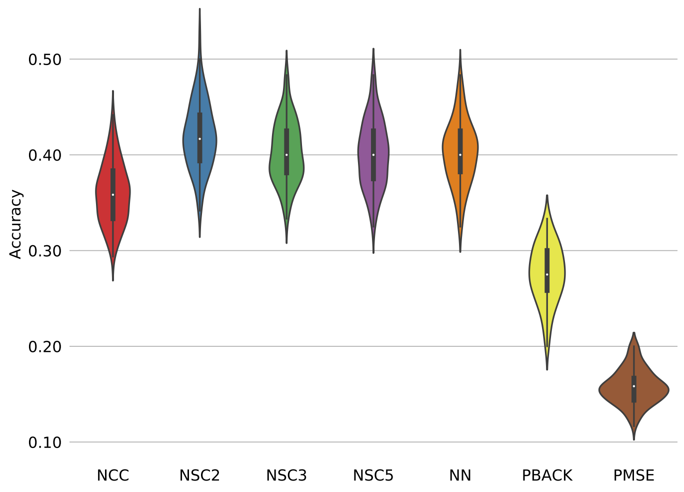

Fig. 5. Scatter plot of the 2D ORL samples after applying PCA.

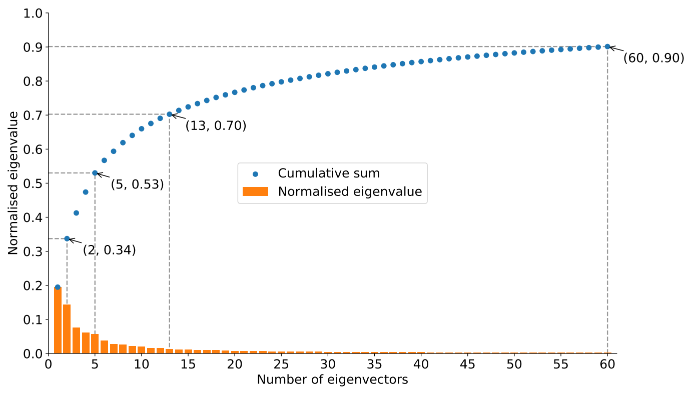

Fig. 6. The first 60 eigenvectors of the total scatter matrix $\mathbf{S}_T$ of the original ORL dataset. The eigenvectors are sorted in decreasing order by their corresponding normalised eigenvalues. The dashed lines highlight cumulative sum of the normalised eigenvalues of the first 2, 5, 13 and 60 eigenvectors.

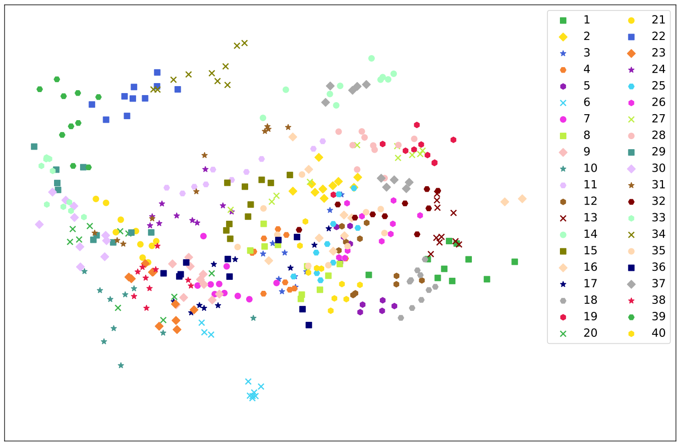

Fig. 7. Scatter plot of the 2D ORL samples after applying PCA.

why the measurements of the PMSE classifier are closely grouped and have less spread than PBACK.

6.2 PCA-transformed ORL

There is a significant impact on classifiers' performance when PCA is used to reduce the number of dimensions from 1200 to only two. The results of the benchmark using the 2D version of the ORL dataset is depicted in Figure 5. The accuracy scores do not exceed 0.52. The lowest score measured is 0.11 which is slightly better than a classifier that predicts one class for any input.

To understand why this happens, let us first try to understand how much information we lose by reducing 1200 dimensions to two. The projection matrix $\mathbf{W}$ used by PCA is composed of the eigenvectors of the total scatter matrix $\mathbf{S}_T$ sorted in descending order of their corresponding eigenvalues. This means that the first eigenvector corresponds to the first principal component, the second eigenvector corresponds to the second principal component and so on.

If we normalise the eigenvalues and study the cumulative sum of the normalised eigenvalues, it is easy to see how much variance remains in the data set for each additional principal component used in the projection matrix $\mathbf{W}$. Figure 6 shows the normalised eigenvalues and cumulative sum of these values. We can see that the two largest eigenvectors only capture 34 per cent of the variance in the data set. To retain 50%, 70% or 90% of the variance in the data set, we need use the 5, 13 or 60 principal components respectively.

Now, let us analyse the results of the classifiers. We can visualise the data since each sample only has two dimensions. Figure 7 shows ORL samples in the reduced feature space. Clearly, it is difficult to classify the samples without the colour and shapes of the samples to aid us because the samples belonging to the same class are intermixed with samples from other classes.

NCC creates a single centroid for each class. In the ORL data set, the centroid corresponds to the mean of the 7 samples (because of the 70% training and 30% test split). If the samples in one class are intermixed with samples from another class, then the centroids of different classes may be very close to each other or even on top of each other. In this situation, the NCC classifier will assign a new sample the label of the closest centroid, which is likely to be incorrect.

The NSC classifiers perform slightly better than NCC because they utilise more than one centroid to represent each class. In the case where multiple samples from the same class are not very close to samples from another class, the prediction of the NSC classifiers becomes more accurate. If the samples from different classes are close to each, then the centroids representing the different classes may be indistinguishable from one another. Similar behaviour can be observed for the NN classifier.

Based on the visualisation in Figure 7, it is also easy to see why the Perceptron classifiers have difficulty discriminating between samples of different classes using the one-versus-rest classification strategy. Drawing a line that perfectly separates samples belonging to one class from the samples belonging to the remaining classes is impossible which implies that PBACK never converges. In some cases, this classifier will assign new samples of the negative class to the positive class because of the limited number of samples for training.

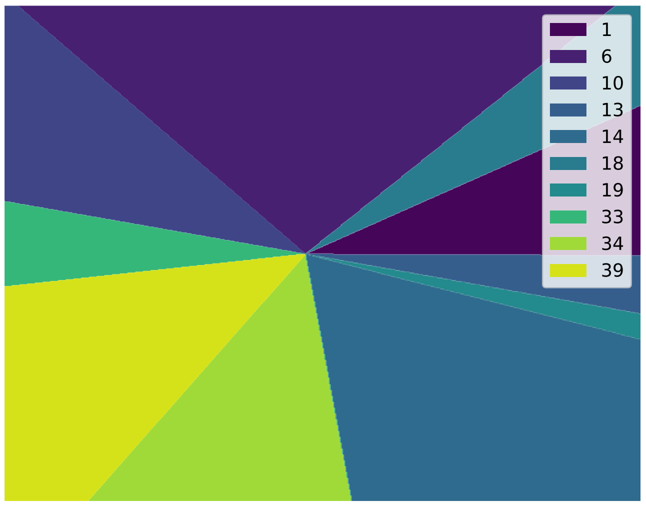

Fig. 8. The decision boundary of the PMSE classifier trained on 70% (randomly selected) of the 2D ORL samples.

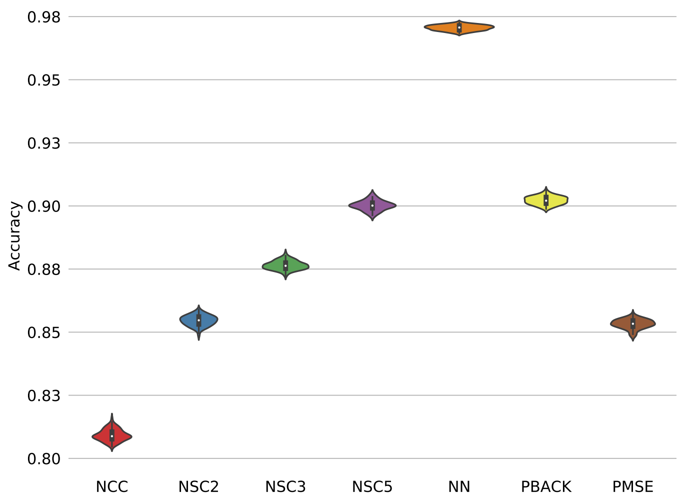

Fig. 9. Performance characteristics of the seven classifiers on the original MNIST dataset

The PMSE is the worst performing classifier on the 2D representation of the ORL dataset. Visualising its decision boundaries (see Figure 8) we observe that the classifier can only distinguish between ten of 40 classes in the ORL dataset. In comparison, PBACK discriminates between 26 classes.

6.3 Original MNIST

The first thing that we notice when looking at the performance characteristics of the different classifiers on the original MNIST dataset (see Figure 9) is that the accuracy measurements for each classifier are grouped close together. This indicates that the trained classifiers are better at generalising to unseen samples. This can be explained by the large sample size.

Benchmark results for the NCC classifier indicate that the samples in each class are not clustered in a hypersphere that allows them to be described by a single point $\boldsymbol{\mu}_k$. Instead, it seems that the samples in each class are grouped in a number of hyperspheres. This interpretation is based on the results of the NSC classifiers. As the number of points used to represent a class increase, we observe that the accuracy of the classifier increases as well. The classifiers NSC2, NSC3 and NSC5 use two, three and five points, respectively, to describe a class. The NN classifier corresponds to the extreme case, where each class is represented by all the samples in that class.

We observe the PBACK performs slightly better than the PMSE classifier on the original MNIST dataset. This suggests that the objective function and the iterative approach of the PBACK classifier seem to find a better discriminating hyperplane than the objective function based on minimising the squared error used by PMSE.

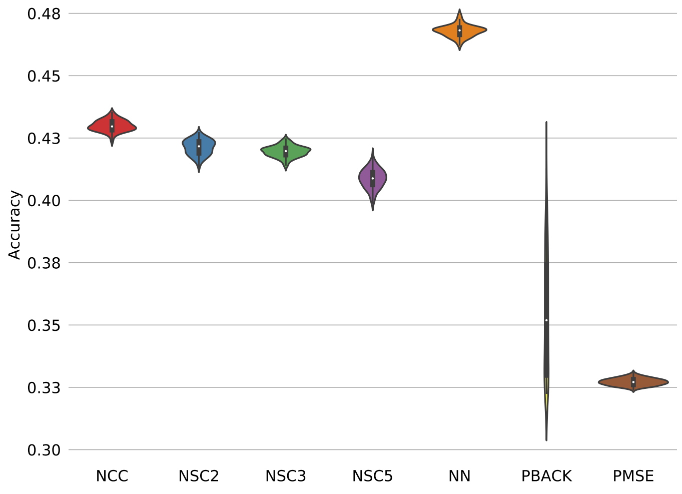

Fig. 10. Performance characteristics of the seven classifiers on the 2D version of MNIST dataset

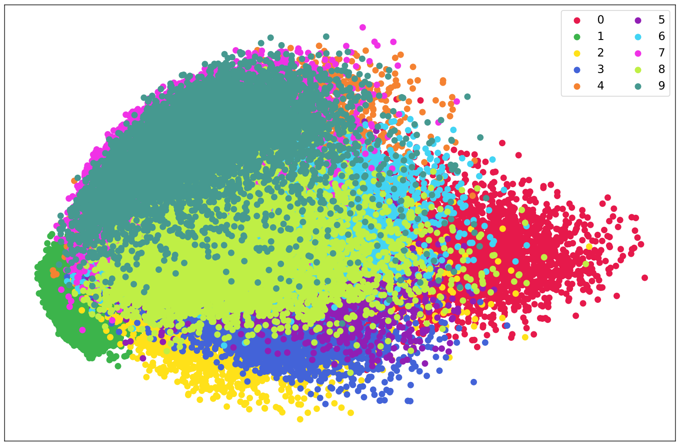

Fig. 11. Scatter plot of the PCA-transformed MNIST data set.

6.4 PCA-transformed MNIST

Again, the performance of classifier suffers by reducing the number of dimensions using PCA. Figure 10 illustrates the performance of the seven classifiers on the 2D representation of the MNIST data set. To explain the behaviour of these classifiers, it is useful to look a scatter plot of the two-dimensional samples visualised in Figure 11.

As NCC is using the mean vector of the samples in each class to represent that class, it is likely to assign a wrong label to a new sample $\boldsymbol{x}'$ that is located in the centre of the blob. However, it will make better classification if $\boldsymbol{x}'$ is on the outskirts of the cloud of data — especially if the new sample belongs to class 0, 1 or 2. A similar explanation applies to the classifiers NSC2, NSC3 and NSC5. The NN classifier performs a bit better because it assigns a label to a new sample based on the closest sample and can, therefore, construct a more complex decision boundary between the ten classes.

In comparison, the two Perceptron classifiers do not perform well on this data set. This can be explained by the fact it is impossible to find a good separating hyperplane using the one-versus-rest classification strategy. The variance of the measurements for the PBACK classifier is very large. This probably happens because the number of misclassified samples is never zero so when the iterative algorithm is stopped (number of epochs has reached 1,000), the separating hyperplane found at that time is rather arbitrary. This can easily be fixed by tweaking the algorithm so it returns the weight vector that has the lowest number of misclassified samples during the entire training phase.

7 Conclusion

When attempting to tackle a classification problem, it is important to evaluate different classifiers on the given data sets as each classifier may respond differently. Also, an understanding of how each classifier works and which assumptions it makes on the underlying distribution of the data set is useful knowledge.

In this project, I benchmarked seven different classifiers on four data sets by running each combination of classifiers and data sets 99 times. At each run, the benchmark script used two-thirds of the data set for training and the remaining for testing. The classifiers include:

- Nearest Class Centroid (NCC)
- Nearest Subclass Centroid with 2 subclasses (NSC2)
- Nearest Subclass Centroid with 3 subclasses (NSC3)
- Nearest Subclass Centroid with 5 subclasses (NSC5)
- Nearest Neighbour (NN)
- Perceptron (PBACK)
- Perceptron using mean squares solution (PMSE)

The four data sets include:

- Original ORL data set
- Original MNIST data set
- 2D representation of the ORL data set after PCA
- 2D representation of the MNIST data set after PCA

In general, the nearest neighbour classifier performed very well on all four data sets, while the other classifiers had different performance characteristics depending on the data set. I analysed the benchmark results and used different visualisations to help explain my analysis. I explained how the performance of the classifiers suffered when the number of dimensions in each sample is reduced to two. Plotting the cumulative sum of the normalised eigenvalues explained how much of the variance in the data is lost when Principal Component Analysis (PCA) is used to reduce the number of dimensions.

In conclusion, the neighbour classifier seems to be a robust classifier that can be used as a baseline classifier when comparing different classification algorithms.

References

[1] Duda, Richard O., Peter E. Hart, and David G. Stork. ”Pattern classification” 2nd Edition, John Wiley & Sons, 2012.
[2] AT&T Laboratories Cambridge, ”The Database of Faces”, Online at https://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html (1994).
[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. ”Gradient-based learning applied to document recognition.” Proceedings of the IEEE, 86(11):2278-2324, November 1998.
[4] Alexandros Iosifidis. ”Introduction to Machine Learning”, November 2017.
[5] Chong, Edwin KP and Zak, Stanislaw H. ”An introduction to optimization” 4th Edition, John Wiley & Sons, 2013