Image Classification problem

*Five classifiers used in image classfication problem

Søren H. Hermansen
Department of Engineering
Aarhus University
Aarhus, Denmark
au544168@uni.au.dk

***Abstract*—This paper introduces and works on the image classification problem, by using five different classifiers for classification and predicting data. The paper will explain and present the different classifiers with their respective predictive accuracy for the classification problem. The databases used are MNIST digit database and the ORL facial images database. PCA will be applied on the data, reducing dimensionality. The goal is to use classifiers on the datasets and comparing their performance, revealing solutions for the image classification problem.**

Keywords*—Machine learning, image classification, nearest centroid, nearest sub-class centroid, nearest neighbor, perceptron, stochastic gradient descent, backpropagation, minimum-squared error, principal component analysis, supervised learning, MNIST, ORL

I. Introduction

For a human, identifying and classifying specific objects can be quite easy. Humans learn from the day they were born, about the world and things in it. They learn from they predecessors and other humans. For a computer, this task can be very hard. Even though the algorithm is fed with a lot of training data, it does not indicate that the algorithm will be better at classifying the data. The data can vary a lot in size, shape and form. This is especially the case with images. For the computer to predict the right label the data needs to have a lot of similarities.

Thankfully, humans can help the algorithm in classifying difference in variance of the data. This method is most commonly called training the algorithm. Training the algorithm is part of supervised learning, in that the algorithm is supervised in classifying correctly. For the most part the classifiers based on the idea that data clusters. This means that classes of the same labelled data, will most likely be clustered together. This also ties in with trying to discover similarities between the data, which ultimately becomes a classification.

This paper introduces multiple ways of analysing and classifying data. For the classification the paper explores five different classifiers, and for analysing the data the paper explores one method. The methods explored in this paper will be explained, scored based on prediction performance and discussed, concluding in different ways to approach the image classification problem.

II. Database

In this paper two databases are analyzed and used in the images classification problem. The first one being the MNIST database, which consists of images of written digits from zero to nine and are labelled by humans. The other database is ORL which features images of forty different faces, that has also been labelled by humans.

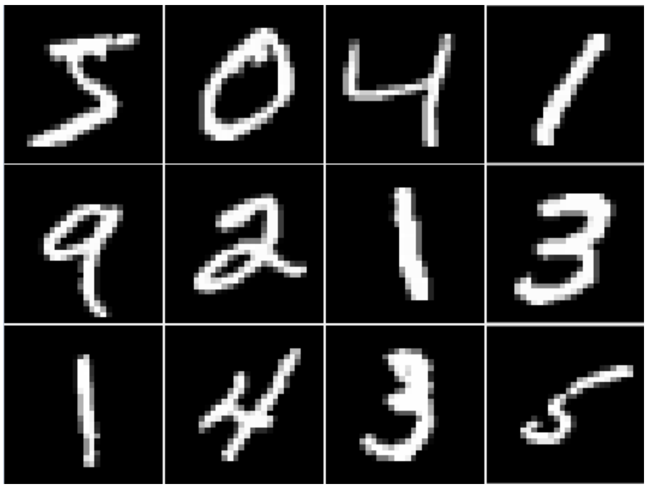
Figure 1: 12 images of the MNIST digit database

A. MNIST

The MNIST database consists of seventy-thousand images of digits, ranging from zero to nine. The images format is originally a 28x28 pixel image. Before using the images, they are vectorized, meaning that the 28x28 pixels becomes the dimensions of the vector, in $\mathbb{R}^D$, becoming 784 features. The database is split into sixty-thousand training images and labels, and ten-thousand testing images and labels. Figure 1 displays 12 of the seventy-thousand images in the MNIST database. The figure also shows the difference in variance between some of the digits, specifically in this case the number “4” is drawn quite differently.

Table 1 displays the distribution of the training data from the MNIST set. An abundance of training data does not explicitly result in better performance, but it is overall better to have a larger dataset, than smaller.

Table 1: MNIST Digits class distribution

0	**1**	**2**	**3**	**4**
5923	6742	5958	6131	5842
5	**6**	**7**	**8**	**9**
5421	5918	6265	5851	5949

B. *ORL*

The ORL database consists of 400 images of faces from forty different people, where there are ten images of each person. The images format is originally a 40x30 pixel image. Before using the images, they are vectorized, creating 1200 features. The ORL database does not have a specific split, and for later in this paper, a method for randomly splitting the ORL database into a testing and training set will be discussed. 12 of the images from the ORL data base are displayed in figure 2.

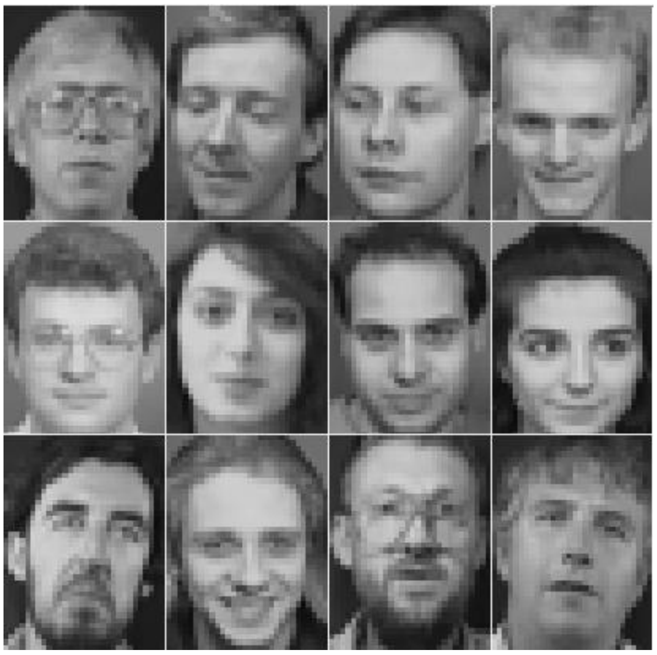

Figure 2: 12 images from ORL database

III. Methods

For analyzing and classifying the data six different methods are used. The ideas behind the methods will be explained in this section.

A. *Principal Component Analysis(PCA)*

The data in the databases, exhibits very high dimensions when vectorized and can be difficult to imagine or to plot the data for analysis. A method for decreasing the dimensions can be used. The method used in this paper is called the principal component analysis (PCA) and is a powerful tool to visualize the clustering of the data, through scatter plots.

The PCA creates components which is fitting a line on the data so that the line covers the majority of variance in the data. This line is called the first principal component and for the second principal component it creates a line that is perpendicular to the first. This is done to cover the second most majority of the variance. As for theoretical view there could be as many principal components as there are features. But since the two first principal components covers most of the variance it enables the plotting of a 2-D graph.

B. *Nearest Centroid Classifier(NCC)*

The nearest centroid classifier classifies data based on proximity to a class centroid. The Euclidean distance between each vector and centroid is calculated, and as the name suggest, the nearest centroid will be the class that vector belongs to. The centroids for the classes are calculated as the mean of all the vectors in the class. The nearest centroid classifier is excellent for classifying datapoints which have low overlap or a high margin between data classes but is severely limited when the data have outliers which are close to other class centroids. The calculation of the mean vector can be seen in equation 1 [1]. Whereas k is the class of the datapoint, and *l* is the label.

$$\mu_k = \frac{1}{N_k} \sum_{i, l_i = k} x_i \, , k = 1, \ldots, K \tag{1}$$

To downscale the issues of having only one centroid, as the mean vector of the data cluster, it is possible to create smaller data clusters within the same class to create multiple centroids. This will be discussed in the next method.

C. *Nearest Sub-class centroid classifier(NSC)*

The nearest sub-class centroid classifier is like the nearest centroid classifier. As the name describes it uses sub-classes for creating the centroids. Sub-classes mean creating smaller parts of the data and creating then creating multiple centroids depicting the mean vector of the smaller classes.

For creating smaller classes or clusters, a cluster algorithm can be used. In this paper the k-means clustering algorithm is used. The k-means algorithm is an unsupervised learning technique which places X number of clusters, randomly assigned to a starting position. The starting position is most likely one of the vectors in the dataset, upon which the k-means is applied. The distance is then calculated to the clusters, and which ever is closest would be the cluster it belongs to. The k-means then calculates the new mean vector for all the datapoints belonging to the class. Then it moves its center to that mean. This is an iterative process until there is no movement in the clusters or the datapoint does not shift class. Equation 2 [1] shows the calculation of the sub class mean vectors, whereas q_i is the label of the sub class and m is the subclass.

$$\mu_{\boldsymbol{km}} = \frac{1}{N_{km}} \sum_{i, l_i = k, q_i = m} x_i \tag{2}$$

For the nearest sub-class classifier, the k-means is applied to parts of the data that is labelled to only one class. This means that it already knows which class the centroid is going to belong to. This allows for creating sub-class mean vectors. The Euclidean distance between a given test datapoint and a class centroid is now smaller and some of the variances in the Nearest centroid classifier is eliminated. It is important to not that there are other ways of calculating the distance than Euclidean. Although it is the most used as distance measure.

D. *Nearest neighbor classifier(NN)*

The nearest neighbor classifier, classifies based on proximity of the nearest vector from a given datapoint, also known as its neighbor data point. The nearest neighbor classifier stems from the idea that labelled data from a certain class will be in a cluster. Therefore, the prediction of the datapoint would most likely be of the same class as the nearest neighbor. As this classifier utilizes proximity it measures the Euclidean distance between the prediction and the nearest vector.

To further solidify the classification the nearest neighbor can utilize the fact that there can be multiple neighbors, and from that standpoint it can produce a better prediction. This can be good, since in some clusters of data there could be outliers from other clusters. If the outliers are closer than the inherent class, the classification would be false.

Utilizing additional data from, a vast number of neighbors the classification could become more precise, although calculating a huge number of neighbors is both time consuming and can be its downfall. This would then produce the idea that, in some cases of databases, there would be a correct number of neighbors to calculate for it to become more precise in its classification.

This classifier is a supervised learning algorithm because it utilizes labelled training data and calculates the distance from a test set datapoint to the training datapoints.

E. *Perceptron with Backpropagation (PCPBP)*

The perceptron is a binary classifier that uses a decision function to either classify the data into one class or the other. This is also called a hyperplane between the two classifications. The optimal hyperplane is very hard to find on datasets that have misplaced samples.

A linear decision function applied upon vector $\boldsymbol{x}$ is written as:

$$g(x) = \sum_{d=1}^{D} w_d x_d + w_0 \tag{3}$$

Where D is the dimensions in $\mathbb{R}^D$, w is the weight vector and expresses the orientation of the hyperplane. w_0 is the bias and is the displacement of the hyperplane from the origin. Since $g(x)$ is a decision function it can be said that if $g(x) < 0$ it belongs to one class and if $g(x) > 0$ it belongs to the other, creating a classification line. If the sample is on the line, it cannot be classified, in which a margin can be introduced. The margin makes sure that the decision function is not placed on the samples.

Backpropagation is when the perceptron is trained, a sample is given, where the expected output is known, and the output is measured. If the output does not meet the expected outcome, weights in a network can be altered to produce the correct output. This is done for each sample to minimize the cost function, which measures the performance of the system. If the cost function is high, it means that there is a lot of errors in the prediction, and vice versa. Although, the classifier can be overtrained if the cost function is very low, since test data can have some differences from the training. For backpropagation it can be seen as moving backwards in the system to optimize the weights of the system to get the desired output or minimizing the cost function.

In backpropagation the error, which needs to be corrected, is based on gradient descent. The weights are updated based on the calculation of the error. How much the weights are changed is based upon a learning rate also denoted η. Pick the right learning rate can lead to better performance and outcome. But for the most part a sufficient small value of the learning rate is a good start. If the learning rate is two high there can be fluctuations and if it is too small, the training process will be long.

F. *Perceptron with MSE (PCPMSE)*

The minimum square error solution is based on a non-linearly separable case, where it maps the training vectors to random target values to find the optimal weights. This contrary to the backpropagation, where it only focuses on the misclassified data. MSE seeks the weight vector that can satisfy equation 4 [1].

$$\boldsymbol{w}^T \boldsymbol{x_i} = \boldsymbol{b_i}, \qquad i = 1, \dots, N \tag{4}$$

Where $\boldsymbol{w}$ is the weight vector, $\boldsymbol{x}$ is the training vectors and $\boldsymbol{b}$ is the expected outputs.

IV. Experiments and Results

In this section the experiments and result of applying the different classifiers on the two data sets will be shown. From the results of the experiments the different strengths and weaknesses for the classifiers will be shown. For the most part the performance of the classifiers will be brought to light in the way of accuracy which describes the difference in the classifiers performance to predict a certain outcome, based on a certain input of data. As part of the result there will be some visualization which, is a great tool to analyze the data without it being in an abstract format. Here the PCA will come in handy since it is hard to visualize something in higher dimensions. The PCA scatter plot shows the clustering of data. The scatter plot for the 2 PCA components of the MNIST data set is shown in figure 3. It can be clearly seen that the data is clustered on top of each other. As for the scatter plot of the 2 PCA components of the ORL dataset, the data is shown to be more spread out. As for how much the components cover in terms of variance, MNIST first component covers about 10% of the variance and the second covers 7%. This means that it loses a lot of the original data when PCA is applied. For ORL the first component covers about 20% and the second, about 14%. It is higher than the MNIST but it still does not preserve a large amount of data.

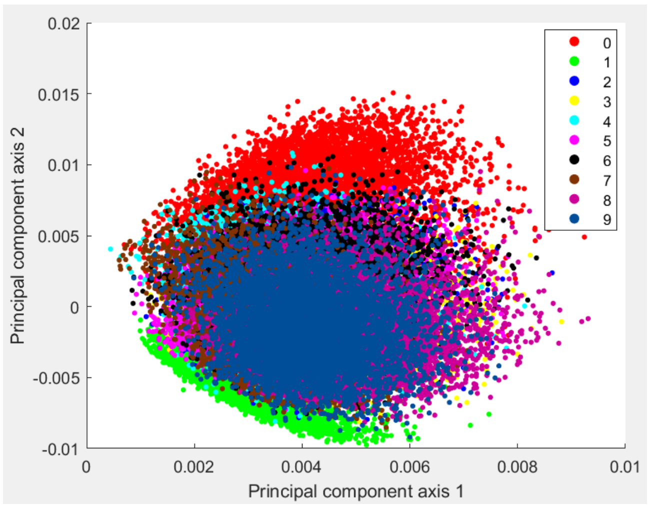

Figure 3: Scatter plot for MNIST 2-components PCA

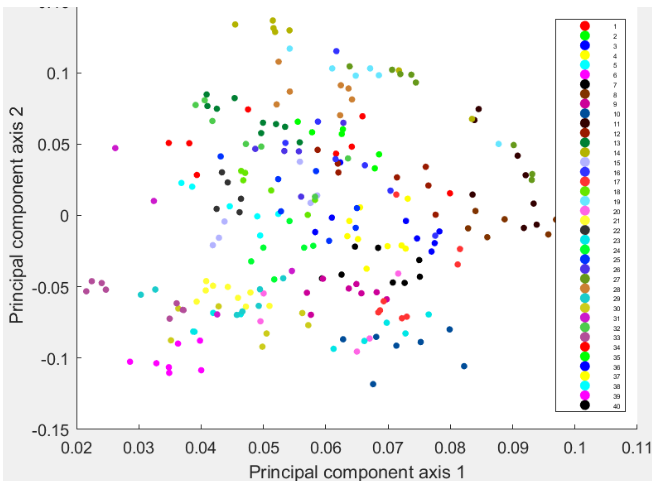

Figure 4: Scatter plot for ORL training 2-components PCA

As described earlier the MNIST data is already split into a 60 thousand labelled training set and 10 thousand as labelled testing set. The ORL is not split from the start so, firstly a 70/30 split is created, leaving 280 samples for the training set and 120 samples for the testing set. This is done using Matlab's built in divide function which randomly divides all the data into the desired split, this case being 70/30. This could later become a hindrance since the data split could end up creating less training data for some classes.

There will be four different sets that will be used for training and testing the classifiers. The two first are MNIST and ORL data, and the last two will be the PCA versions of the sets where only two components will be used. The overview of the datasets can be seen in table 2.

Table 2: Overview of datasets

Dataset	Train	Test
MNIST	784x60000	784x10000
ORL	1200x280	1200x120
PCA MNIST	2x60000	2x10000
PCA ORL	2x280	2x120

A. Nearset centroid classifier

The first classifier is the nearest centroid classifier and the performance on the different sets can be seen in table 2.

Table 3: NCC performance

	MNIST	ORL	MNIST (PCA)	ORL (PCA)
NCC (%)	82.03	87.5	27.8	15

B. Nearest sub-class centroid classifier

For the next classifier the nearest sub-class will be tested and score. For number of sub-classes there will be 2,3 and 5. The performance can be seen in table 3.

Table 4: NSC Performance

	MNIST	ORL	MNIST (PCA)	ORL (PCA)
NSC (2) (%)	82.17	87.5	28.44	15
NSC (3) (%)	85.98	95	27.53	10
NSC (5) (%)	89.52	95.83	28.24	10.83

C. Nearest neighbor classifier

Next is the nearest neighbor classifier, where several amounts of neighbors can be chosen. For this paper the number of chosen neighbors will be 1,2,5,10 and 15. The performance can be seen in table 4.

Table 5: KNN Performance

	MNIST	ORL	MNIST (PCA)	ORL (PCA)
KNN(1) (%)	96.91	95.83	10.83	26.9
KNN(2) (%)	96.27	83.33	12.5	27.17
KNN(5) (%)	93.92	55	10.83	27.14
KNN(10) (%)	90.24	29.16	11.66	26.33
KNN(15) (%)	87.44	12.5	8.33	25.78

D. Perceptron with backpropagation

For the fourth classifier, namely the perceptron with Backpropagation, will be tested. For the implementation, Stochastic Gradient Descent (SGD) from scikit learn [] has been used. For the loss function, "hinge" is used. The parameters being tuned is the learning rate. The learning rate η will be 0.1, 0.01, 0.001 and 0.0001. The performance of the perceptron with backpropagation can be seen in table 5.

Table 6: Performance of Perceptron with Backpropagation

	MNIST	ORL	MNIST (PCA)	ORL (PCA)
η (0.1) (%)	89.31	90.83	12.16	0
η (0.01) (%)	91.34	94.16	17.79	4.16
η (0.001) (%)	91.85	75.83	21.57	0.83
η (0.0001) (%)	91.14	13.33	22.75	0.83

E. Perceptron with MSE

For the performance of the last classifier, the perceptron with minimum-square error, will be tested. As the same as the last on the SGD has been used with squared loss as its loss function. The learning rate will be 0.01, 0.001 and 0.0001. The performance can be seen in table 6.

Table 7: Performance of Perceptron with MSE

	MNIST	ORL	MNIST (PCA)	ORL (PCA)
η (0.01) (%)	75.29	2.5	18.59	0
η (0.001) (%)	85.61	80	13.63	0
η (0.0001) (%)	85.56	35.83	14.29	0

F. Summary of results

Before diving into the discussion, figure 5 shows the summary of the performance score for the classifiers predicting on the four different datasets.

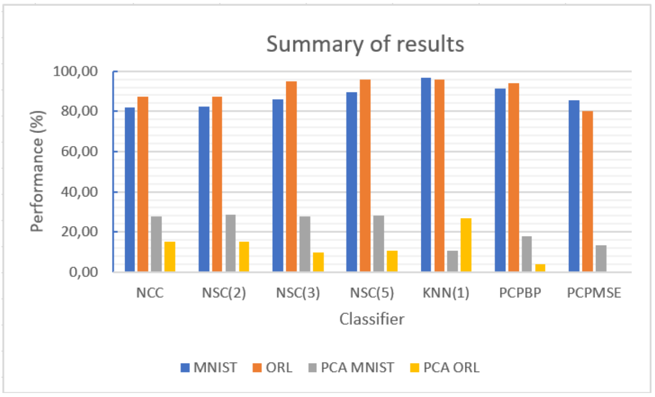

Figure 5: Summary of classifier performance

V. Discussion

Now the results have been found and shown, the performance of the different classifiers can be compared, and there might be an optimal classifier for each of the datasets. As a precaution for the results the split data from the ORL is random, therefore running the split multiple times could be beneficial, to see if the split is either favorable or not.

For the NCC the results are reasonable for the two datasets, being MNIST and ORL. Although for PCA part of the classification it seems to fall short quite extensively. As the NCC performs better on the ORL than MNIST. The ORL PCA version seems to be a lot lower than the MNIST PCA version. As for the PCA result themselves, it can be explained that for both having a high dimensionality, and then crushing it into only two dimensions, would mean that a lot of the classifying data would be lost. Looking at the images of both datasets its also quite clear that for MNIST a lot of the data is mostly centered on a small part of the images.

For NSC it seems that the greater amount of sub-class has increased the classification. Comparing NSC to NCC the introduction of extra sub-classes, produce better predictability, for nearly all the datasets. This however comes with the cost of performance, of which the introduction of multiple new sub-classes, increases the time of the k-means clustering algorithm. But if the target is high predictability with low regards to performance, it outperforms NCC. This can also be explained as the sub-class centroid will most likely be more towards the edge's clusters. For the PCA versions, there is not much difference.

The KNN is quite good at predicting with only having one neighbor to classify from. The performance drops with the introduction of extra neighbors, which describes that the data clusters are very close to each other. It drops more in ORL than in the MNIST which means that ORL have a large spread of class data, and there could be many outliers. The KNN is the slowest of the algorithms presented in this paper, meaning that it beats NSC as being the best predicter with disregard to time performance.

For the perceptron trained with backpropagation it seems that it becomes more precise for MNIST when the learning rate is lower, but almost the other way round for ORL. This shows that it is important to pick the right learning rate, as the prediction for ORL becomes better the first time the learning rate dropped, but the performance quickly deteriorate. The classification is quite fast and easily outperforms the KNN, with multitudes of neighbors, in terms of speed and rivals in terms of predictions.

The perceptron trained with MSE seems to have high variance in performance for the ORL dataset, and it seems quite important to pick the right learning rate. This follows the pattern for the backpropagation that the data weights are quite fragile in the change of the learning rate. For the MNIST it also performs lower than backpropagation.

For the perceptron's it seems that ORL has a very high variance in terms of performance when tuning the hyperparameter. One of the reason could be that there is not enough data to train the algorithms or optimize the weights. It can also be stated that the ratio between data classes and labels are quite different in the two dataset, which could explain why the classifiers almost always performs better on MNIST.

VI. Conclusion and Summary

This paper explored the image classification problem in which two image databases were used to clarify the performance of five different classifiers. The paper also explored the idea that there is not a classifier for all databases. There can be a high variety in performance for different classifiers for different databases. In this classification problem the use of dimension reduction algorithm was unsuccessful at bringing any competitive performance. The paper also explored that the image classification problem could be solved or bring a high performance by using the right classifiers. But it can be concluded that there is not a perfect classifier for all data.

References

[1]. A Iosifidis, Introduction to machine learning, Aarhus university, Department of Engineering, Electrical & Computer engineering, 2018

[2]. Mathworks, Matlab 2021b, https://se.mathworks.com/, 2021

[3]. Matlab Deep Learning Toolbox, Version 14.3, https://se.mathworks.com/products/deep-learning.html

[4]. Matlab Bioinformatics Toolbox, Version 4.15.12, https://se.mathworks.com/products/bioinfo.html

[5]. Matlab statistics and machine learning toolbox, Version 12.2, https://se.mathworks.com/products/statistics.html

[6]. Python, Sci-kit learn, https://scikit-learn.org/stable/