Image Classification For Beginners

*image classification presented in a understandable way

Kelvin Østergaard Pagels
Department of Engineering, Aarhus University
Finlandsgade 22, DK-8200 Århus N, Denmark
Aarhus, Denmark
pagels@gmail.com

***Abstract*—In this paper 5 different classifiers will be presented by the use of the well known MNIST and ORL data set. The paper presents a comparison between the classifiers where it is shown that if they are in their right field they will perform satisfiable.**

***Index Terms*—machine learning, classifiers, visualization,nearest neigbor, perceptron, sub-class, MNIST, ORL**

I. Introduction

The task of recognizing a visual concept is normally relatively easy for a human to perform, but for a computer algorithm it can be rather complex. The reason for this can be many; If the object is dark or too light it can be a challenge. If the object is oriented in a way that makes it hard for a camera to detect it. If the object is blending into their environment. These are just some of the reasons why it's a complex process. The classes of interest are many such as animals, tables, lamps and computers.

It's easy to imagine how to write an algorithm that reads from a command line or an algorithm that sort an array with numbers that are unsorted. But to write an algorithm that takes all the things above into consideration seems complex but the solution is not. The idea of how to write an algorithm like this is just like with children, we will give them a lot of examples of vegetables and as time goes they will learn to distinguish between them. Take this example and now provide the computer with a lot of examples of each class (training datasets) and let it develop a learning algorithm that is able to distinguish between the classes. When the learning algorithm is developed it's time to take testing datasets that haven't been part of the training. We provide the learning algorithm with the testing datasets and it gives us a prediction of which class it thinks it belongs to.

The image classification is characterized as input, learning and evaluation. The input consist of a training set with a number of images each labeled with one of the classes in the dataset. The task of the learning is to take the training set and develop the learning algorithm so it's able to distinguish between the different classes. When the learning is done it's time to evaluate which consist of evaluating how well the learning algorithm performed. Throughout this paper different learning algorithms will be introduced, tested and validated.

II. Data

To be able to train, test and validate the different classifiers data sets are needed. The data sets that are gonna be used throughout this paper is the well known MNIST dataset and ORL data set [2]. These data sets are widely used around the internet and is depicting faces and digits. The MNIST dataset is a set of 70k (vectorized) 28x28 pixel images depicting hand-written numbers. An example of 60 digits are shown in Figure 1.

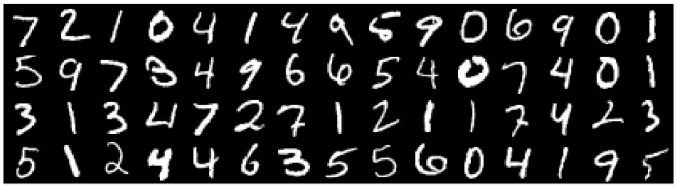

Fig. 1. 60 random digits are shown

Where the 70k hand-written numbers are randomly divided over the different classes as follows:

TABLE I
NUMBER OF DIGITS DISTRIBUTED OVER CLASSES.

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

The ORL data set consist of 400 (vectorized) 40x30 facial images depicting 40 persons. The data set randomly split each of the 40 classes in 70% training and 30% testing images. An example of 10 of the 40 faces are shown in Figure 2.

III. Methods

To apply image classification on data classifiers are needed. In this section the different classifier used in this paper will be introduced. different classfications will be described briefly.

A. Nearest class centroid classifier

To apply image classification on the described data sets different classifiers are needed. The job of these classifiers are to use the training data to determine the best classifiers

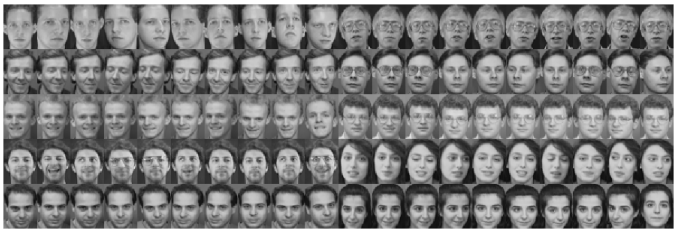

Fig. 2. 10 of the 40 faces are shown

to classify samples correctly. To accomplish this 5 different classifiers will be tested and validated in this report.

The nearest class centroid classifier(NCC) is a classification where each class is represented by its centroid. The prediction consists of computing a centroid for each class. Then measure the distance (generally Euclidean) from the data point X to the centroid of each class. The data point X will get assigned to the class it is closest to. This process will continue until all the data points are assigned to a class. When the training and testing is completed the quality of the classifier is evaluated.

B. Nearest sub-class centroid classifier

The nearest sub-class centroid classifier(NSCC) is a classification where each class forms several subclasses. Instead of computing one centroid for each class as in the **NCC** classifier this algorithm computes a centroid for every subclass. To define the subclasses of each class a clustering algorithm K-Means is applied using the samples of the corresponding class. The K-Means algorithm then finds K subclasses per class. Then the training is done like in the **NCC** classifier.

C. Nearest Neighbor classifier

The nearest neighbor classifier(NNC) is a classification where the subclasses in the **NSCC** classifier is equal to the number of its samples. So instead of computing only K subclasses for every class it computes the number of its samples subclasses. Then the training is done like in the **NCC** classifier.

D. Perceptron using Backpropagation

The perceptron using Backpropagation(PBP) is a bit different from the described classifiers. It trains, test and predict like them but it does it in a different way. The main idea is to find the optimal decision function (hyperplane) between two classes. This decision function corresponds to a hyperplane in $\mathbb{R}^D$ that will discriminate the samples that belongs to different classes as shown in Figure 3. Here the optimal hyperplane is shown which most likely never can be found if some samples are misplaced. Linear discriminant functions can be categorized in different ways such as the two-category case, multi-category case and the general case. In this section only the two-category case will be considered. A discriminant function that is a linear combination of the components of x

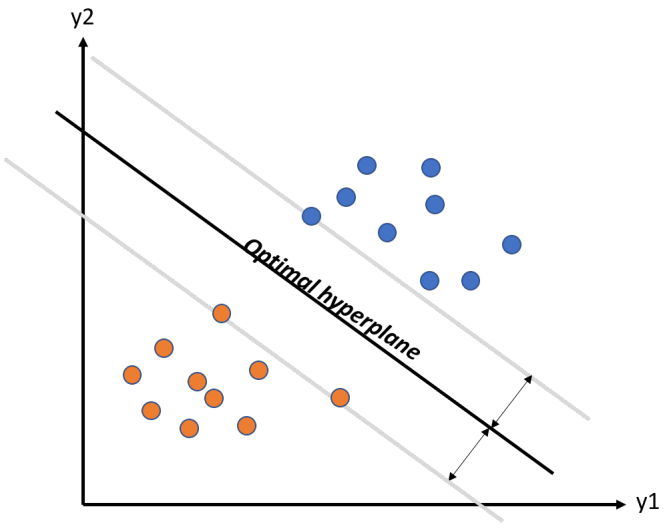

Fig. 3. Optimal hyperplane

can be written as in (1) [1] which gives a measurement of the distance of x from the hyperplane:

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

Where $w \in \mathbb{R}^D$ and w_0 are the *weight vector* and *bias*. w expresses the orientation of the discriminant hyperplane, while w_0 expresses the displacement of the hyperplane from the origin. In the two-category classifier the following decision run is implemented: Decide class 1 if g(x) $>$ 0 and class 2 if g(x) $<$ 0. The data is classified as a set of vector $x_i, i = 1, ..., N$ where each has a binary label $l_i \in 1, -1$. This is also called the *binary classification problems*. A weight vector that optimally discriminates the two classes must be found. Of course if the images are badly-separated it's not possible to find a weight that classifies all the vectors correctly. An example is given: Four images (2x2pixels) are represented as in (2). Where the first two images are represented as class 1, image 3 as class 2 and image 4 as class 3.

$$\{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \mathbf{x_4}\} = \begin{matrix} 1 & 1 & 2 & 3 \end{matrix} \begin{bmatrix} 0.18 & 0.18 & 0.34 & 0.23 \\ 0.12 & 0.12 & 0.56 & 0.45 \\ 0.18 & 0.18 & 0.67 & 0.21 \\ 0.10 & 0.10 & 0.32 & 0.22 \end{bmatrix} \tag{2}$$

Each class will have a binary vector consisting of (row x columns) and in this case it will be a 1x4 vector. Where class 1 will have the first two elements as 1 and the last two as -1. The binary matrix would be presented as:

$$\mathbf{b} = \begin{matrix} 1 & 2 & 3 \end{matrix} \begin{bmatrix} 1 & -1 & -1 \\ 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix} \tag{3}$$

Given this set of vectors its possible to determine the optimal hyperplane for our classification problem. Using the binary classification [1] a new decision function can be derived:

$$f(\mathrm{w}^*, x_i) = l_i g(\mathrm{w}^*, x_i) = l_i \mathrm{w}^{*T} x_i \geqq 0 \tag{4}$$

Where $i = 1, ..., N$. When $f(\mathrm{w}^*, x_i) \geq 0$ the samples are classified correctly and when $f(\mathrm{w}^*, x_i) \leq 0$ the samples are missclassified. The weight w will be initialized. In this algorithm only the misclassified samples are considered and they are put in $\chi(\mathrm{w})$. The algorithm starts by taking one binary vector for a class and looks through all the samples. This is done by taking the element that corresponds to that given sample and multiply them together. Here an example are shown with the first image as first iteration and element 1 in class 1. Note that the weight and the sample has been augmented.

$$f(\mathrm{w}, x_i) = 1 \cdot \begin{bmatrix} 0.1 \\ 0.2 \\ 0.8 \\ 0.1 \\ 1 \end{bmatrix}^T \cdot \begin{bmatrix} 0.18 \\ 0.12 \\ 0.18 \\ 0.10 \\ 0.2 \end{bmatrix} = 0.396 \tag{5}$$

When all the images are run through the weight will be updated by only looking at the misclassified samples. To optimize w the criterion function must be defined. After applying $f(\mathrm{w}, x_i)$ all the misclassified vectors will be collected in a vector set $\chi(\mathrm{w})$. The above calculation would be classified correctly and therefore not put in χ. The optimal solution is when a given w gives a empty χ. When the algorithm is initialized χ will be empty. To update the weight w the overall error is added to the previous weight. The overall error is given by [1]:

$$\mathcal{J}_p(\mathrm{w}) = \sum_{x_i \in \chi} -f(\mathrm{w}, x_i) = \sum_{x_i \in \chi} -l_i \mathrm{w}^T x_i \tag{6}$$

All the classes are run through and for each class the algorithm will calculate the discriminant function which will divide the feature space into two regions, the correct classified and the misclassified. The hyperplane will be placed at a random place at start and most likely many images will be misplaces. In the batch perceptron algorithm the weight w will only be updated when all the images are classified which means that all the misclassified samples are put in χ. When all the samples are run through, which is called a epoch, w will be updated with all the misclassified samples as follow which is called the Perceptron criterion function:

$$w(t+1) = w(t) + \eta(t) \sum_{x_i \in \chi} l_i x_i \tag{7}$$

Where η is set to a constant value, which is the rate of changed and is called the *learning rate*. The algorithm will perform multiple epochs until all samples are correctly classified which will happen when χ is empty. To not end in an infinity loop with epochs its a good idea to set a max limit of iterations. The reasoning for this is that not all samples will be classified correctly. The learning rate η must be chosen with caution since it changes the accuracy a lot just by minor changes. When the weight w has been calculated for one class the algorithm will continue until all classes got a weight. When all the weights are calculated its time to train the training samples. This is done by multiplying the weight with the testing samples:

$$q = W \cdot X_{TEST} \tag{8}$$

This calculation will give a matrix where each row represent each class. The max index of each column represent which class it is classified to. To follow the example previous the correct classification would give the matrix where the maximum elements are highlighted:

$$P = \begin{pmatrix} \boxed{0.8} & \boxed{1.2} & 1.1 & -1 \\ 0.2 & -0.2 & \boxed{1.8} & 0.4 \\ 0.3 & 0.2 & 0.1 & \boxed{1.0} \end{pmatrix}$$

This result classifies image 1 and 2 to class 1, image 3 to class 2 and image 4 to class 3.

IV. Perceptron using MSE

Perceptron using MSE is a minor version of Perceptron using Backpropagation. In this classification the idea is to instead of only focusing on the missclassified samples, the Minimum Square Error (MSE) solution tries to map all the training vectors to some arbitrary vectors to some arbitrary target values. That means that it seeks the weight vector w that can satisfy the following:

$$\mathrm{w}^T x_i = b_i, i = 1, ..., N, \tag{9}$$

In the above, x_i is the training vectors and b_i is the binary vector. In order to determine the optimal w^*, we form the following optimization problem

$$\mathcal{J}_s = \sum_{i=1}^{N} (\mathrm{w}^T x_i - b_i)^2 = ||X^T \mathrm{w} - b||_2^2 \tag{10}$$

And by minimizing $\mathcal{J}_s$ the optimal w can be determined:

$$\mathrm{w} = (XX^T)^- 1Xb = X^\dagger b \tag{11}$$

The weight can now be calculated and the next step is to follow the steps in the Perceptron using BackPropagation.

V. Visualization

In Machine Learning data visualisation is an important part of analysing data. One of the benefits of visualization is that it allows visual access of huge amounts of data in a easily digestible visuals. Data visualisation can be presented in a pictorial or graphical format. Throughout this paper the different classification algorithms will be presented in both pictorial and graphical format. To visualize data that has many variables like the MNIST and ORL the TSNE algorithm [3] has been applied. All the visualisation of data has been performed in MATLAB. Some illustrations will be using scatterplot where others will use the TSNE algorithm which

helps us to visualize data into the 2-dimensional space. On Figure 4 the TSNE algorithm is used to represent the nearest sub-class centroid classifier with k = 3 by using the MNIST data set. This plot illustrate how the algorithm distinguish

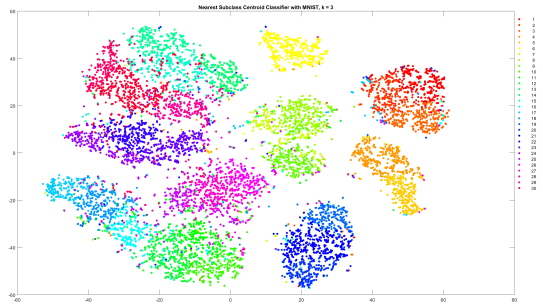

Fig. 4. Nearest sub-class centroid classifier with k = 3

between the classes in the MNIST dataset. The training data is plotted into a 2-dimensional space to illustrate how close the images are together where each dot in the specific color is a number. The color of the specific dot specifies which class that dot represent. The dots that are placed inside another color are misclassified samples. Figure 5 shows how the 10 digits have been discriminated into different classes.

VI. Result

In this section all the classification techniques will be shown and the advantage and disadvantage discovered will be listed. Since the ORL dataset has only 400 samples it will be explained through scatterplots and tables. The MNIST dataset will be explained with tables and by applying the TSNE algorithm. As explained in previous chapters the distribution of the data is 70%training and 30% testing. Instead of sticking to that distribution all the datas will be merged together and divided randomly in two groups again; training and testing. The reasoning for this is to get different result and see how the classifiers works with different inputs.

The **NCC** is the first described algorithm where each class is represented by its centroid. Figure 5 shows how it discriminate the samples into classes where each class is represented by a color. By looking at the figure it is clear that the algorithm does a satisfying job where the accuracy of classifying samples are 91.67%. To have an understanding of how the algorithms performs the algorithm is executed multiple times depending on the deviation of the accuracy. If the accuracy changes significantly more tests are performed and if it has minor changes only few tests will be performed. While running the algorithm it was observed that the accuracy of the classification did not change significantly and for that reason only 5 runs have been performed. The accuracy for each run is shown in Table II. The **NCC** with MNIST data does perform to a accuracy for about 80 %. To see how the algorithm works on the ORL data set the same tests are performed. While doing the tests it was observed that the accuracy changed significantly

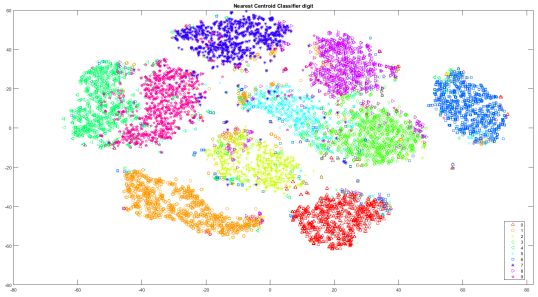

Fig. 5. NCC with MNIST. The training and testing set consist of 60k training and 10k testing samples respectively. This plot illustrates the 10k samples.

TABLE II
THE TABLE SHOWS THE RATIO (TRAINING/TEST) AND THE NUMBER OF RUNS PERFORMED. NCC WITH MNIST DATA SET.

Ratio/Run	1	2	3	4	5
50/50	81,14	81,14	80,66	81,04	80,94
70/30	80,62	81,07	81,19	81,18	80,91
80/10	81,33	80,67	81,48	81,12	81,74

and therefore 1000 runs have been executed. The distribution of testing and training can be seen in Table III. The table shows that the accuracy (about 92%) for the ORL data set is better then for the MNIST data set(about 80%).

TABLE III
THE TABLE SHOWS THE RATIO (TRAINING/TEST) AND THE NUMBER OF RUNS PERFORMED. NCC WITH MNIST DATA SET.

Ratio/Run	1000
50/50	91.92%
70/30	91.76%
90/10	91.92%

The **NSCC** is the second explained classifier where each class is represented by K sub-classes which should optimize the classification process. This algorithm is more tricky to implement and takes some time to get around it to work. On Figure 6 a plot is shown of how it predicts the testing data for the MNIST with K = 2. The labels on the right side represents the subclasses and in this case subclass 1 and 2 belongs to class 1. If we take a look at the plot its possible to see where the different subclasses are located. The shuffling of data is really important in this algorithm since k-means always find the best sub-classes that represent the class. So if the training data is the same and the data shuffled k-means will still find the same sub-classes. In Table IV the accuracy for different train/ test ratio is shown. The Table shows the accuracy when the algorithm executed 1000 times. It can be seen that the accuracy have minor changes even if the train / test ratio changes. Here the accuracy is about 86% and performs better then the **NCC** algorithm which had a accuracy of 80%.

The algorithm is performed for the ORL data set and by looking at the Table V it is shown that it gives a higher accuracy at around 94% compared to the 91% in the **NCC**

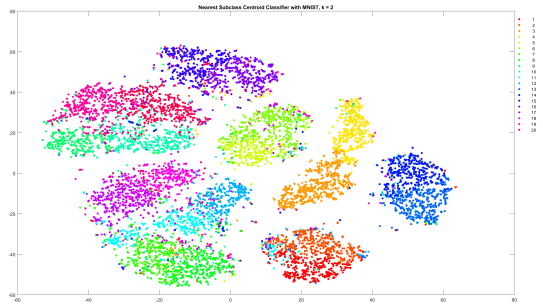

Fig. 6. NSCC with MNIST and K = 2. The training and testing set consist of 60k and 10k samples respectively. This plot illustrates the 10k samples.

TABLE IV
NSCC WITH MNIST, K = 2.

Train/ run	1000
50/50	85,84
30/70	86,46
15/85	86,16
10/90	86,12

algorithm. Again the accuracy does not change by changing the ratio of train and testing data.

TABLE V
NCC WITH ORL, K = 2 THAT ILLUSTRATES 5 AND 1000 RUNS.

Train/ Test	1	2	3	4	5	1000
50/50	92,50	95	95,83	99,16	90,83	94.25
70/30	95,83	92,50	95	93,33	93,33	94.01
90/10	94,16	96,66	93,33	87,50	95	93.85

From Table IV and V it was seen that the accuracy had minor changes when the train / testing ratio changed. For this reason it was seen as redundant to check the different ratios for K = 3 and 5. Therefore on Table VI the accuracy for K = 2,3 and 5 running 1000 times with 70/30 ratio, have been gathered for both ORL and MNIST data set. Note that the accuracy for ORL data set does not change from 2 to 5. The reason why it does not change is probably because the dataset is so little and it already classified the samples that could be classified correctly. But on the other hand the MNIST dataset changes significantly. This is because its a big dataset and there is about 6000 digits per class. Its easy to imagine that by increasing K in the MNIST dataset the accuracy would increase for the MNIST dataset.

TABLE VI
NSC WITH ORL AND MNIST REPRESENTING K = 2, 3, 5

	K = 2	K = 3	K = 5
ORL	94.25	94.82	95.21
MNIST	85.46	88.18	90.53

The last plot for **NSCC** is with K = 5 and it is shown on Figure 7. This plot is abit more tricky to distinguish between which subclasses belongs to what. But by looking carefully it becomes clear where the subclasses are placed.

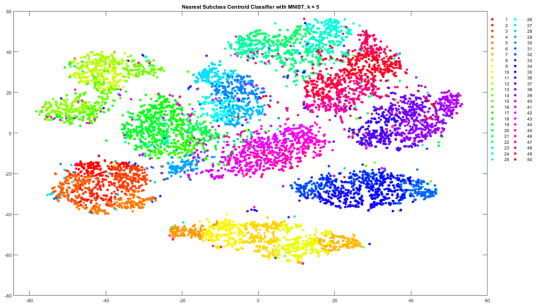

Fig. 7. NSCC with MNIST and K = 5. The training and testing set consist of 60k and 10k samples respectively. This plot illustrates the 10k samples.

The **NNC** is one of easiest algorithms to implement since its just checking all the training data against all the samples in the testing data. But it is also one of the most time consuming classifiers explained in this paper. For the MNIST data set it had a accuracy of 90.5% but this was very time exhaustive. The **NNC** for the ORL data set is presented in a scatterplot which shows the correct vs predicted labels. This can be seen on Figure 8. This plot shows that there is only 3 faces that are misclassified which gives an accuracy of 97.5%. The correct labels can be seen on the y-axis where the predicted labels can be seen on the x-axis.

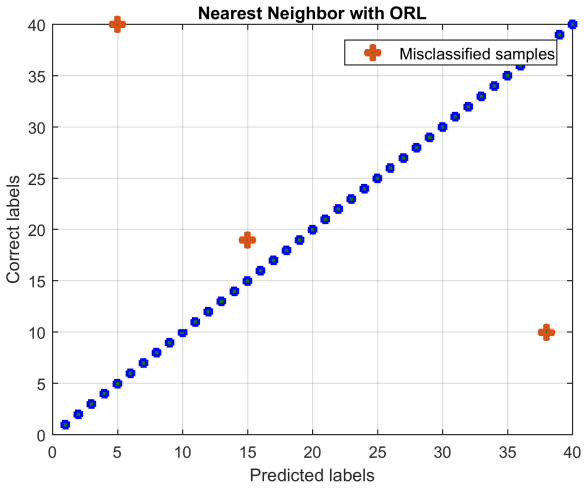

Fig. 8. **NNC** with ORL. The figure illustrates the correct vs predicted labels. The testing set consist of 220 samples and the testing set consist of 180 samples. This figure illustrates the 180 samples with a 97.5% accuracy.

The **PBP** is the most advanced algorithm to implement. The advantage of this algorithm is that it adds some parameters that can be tweaked. On Table VII 3 different etas are shown for the MNIST data set and the accuracy is almost the same with all the tests. By this it can be concluded that it does not add any increase in performance compared to the previous algorithms. Since eta is a changable parameter the accuracy

could possible be improved. But in tests performed in this paper that eta havent been found.

TABLE VII
PBP WITH MNIST, SHOWING THE MEAN OF 3 ETAS WITH 100 RUNS

ETA	0.5	0.1	0.9
Accuracy	79	82	80.5

To visualize the Perceptron using Backprogation with the ORL data set a bar plot is showed in Figure 9. This plot illustrate how many faces are misclassified and the total number of misclassified faces are 25. Again by changing eta did not add an increase in accuracy. But it should be noted that this could be a mistake by the writer.

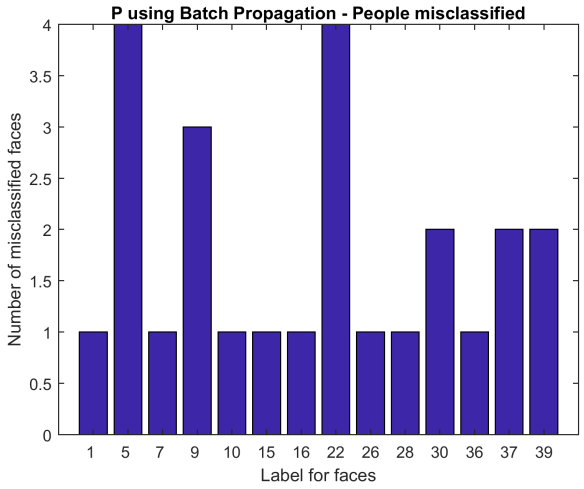

Fig. 9. PBP with ORL representing People Misclassified

The last algorithm Perceptron using MSE is represented with ORL data and is shown on Figure 10. Here it is visualized that 9 faces have been misclassified. For the MNIST data set the accuracy is around 91%.

The **NCC** performed great for small data sets with a accuracy on 91.6% where it performed worse with a big data set of 80.94%. The **NSCC** is an extention of the **NCC** and by increasing K the classifier did a good job with the consequence of increased computational time. The **NSCC** added considerable additions when it came to increasing performance of the algorithms. By increasing K the accuracy for the algorithm was improved. For the big dataset the accuracy came to 97.1% with K = 1000 but with a computational time on 23minutes. For the small data set the accuracy was 95% with a small computation time. The **NNC** is the most exhaustive classifier of all of them. It performed with a accuracy of 90.5% for big data sets and 97.5% for the small data set.

With the PBPC the accuracy was around 80% for big data sets and is the worst performed of them all. For the small data set it performed around 90%. But this algorithm adds a parameter η that can be changed. By choosing η to the correct value the accuracy could probably be better then the one found. The last algorithm Perceptron using MSE performed 90.1% for

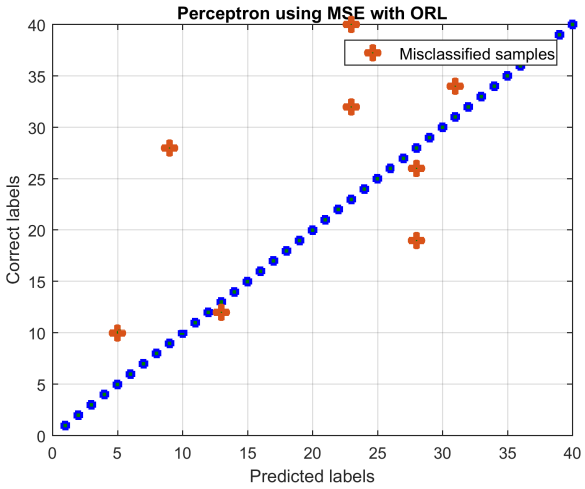

Fig. 10. Perceptron using MSE with ORL. The testing set consist of 220 samples and the testing set consist of 180 samples. This figure illustrates the 180 samples with a 90.1% accuracy.

the big data and small data. On the other hand the classifier was the fastest of all of them.

To choose a classifier really depend on what kinda job that need to be done and from the writers perspective its all about if the data set is big or small. If the dataset is big and time is not a factor the **NSCC** would be choosen with a big K. This gives a great performance but with a long computation time. If the dataset is small the **NSCC** could be choosen with a 95% accuracy but again long computation time. If the algorithms need to be implemented on a critical system the algorithms described in the paper would probably not be sufficient. If computation time is a factor and a accuracy of 90% is acceptable then the **PMSE** would be the best choice for the big and small datas.

All the different classifiers have different roles and its not possible to give one algorithm that would be best for every cases. The different classifiers have different roles and each of them are there to stay.

VII. CONCLUSION

In this paper 5 different classifiers have been presented. They are all based on input, learning and evaluating. They have all been compared to the data sets MNIST and ORL. The advantage and disadvantage of the classiers have been presented throughout the report. The classifiers have different objectives and to achieve the best performance it is needed to use them the way they have been designed.

REFERENCES

[1] A Iosifidis, Introduction to Machine Learning. Aarhus University, Department of Engineering, Electrical & Computer Engineering, 2017.
[2] MNIST AND ORL DATA, https://github.com/CUFCTL/face-recognition/wiki/Datasets
[3] MATLAB, https://se.mathworks.com/help/stats/t-sne.html