Optimization and Data Analytics Project

Peter Marcus Hoveling
Institute for computer technology
Aarhus, Denmark
au536878@uni.au.dk

ABSTRACT

In this Project we will implement and compare the performance of five classification schemes. Implementation will be in MATLAB, C/C++ or Python. Finally, we will write a report following the standard scientific writing style. For the submission of the project, we will include code source files and the report in a .zip file. Submission will be through black board.

Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/201508876PMH/Classification_schemes.git. All of the used python libraries will be referenced here[5][6][1][3][4].

1 INTRODUCTION

Given for this project is five classification schemes; Nearest class centroid, Nearest sub-class centroid using the number of subclasses in the set {2,3,5}, Nearest Neighbor, Perceptron trained using Backpropagation and Perceptron trained using MSE (least squares solution). Of these five, only the first three will be implemented and evaluated.
For the classification schemes, two data sets; MNIST (60k/10k) train/test splits and ORL (40 classes in 70% training and 30% test images) are given. In both data sets, the training data will be used to determine the best values for the hyper-parameters of each method. Then using the best hyper-parameters values, the methods will be trained on the entire training set and evaluated by performance on the test set.

2 IMAGE CLASSIFICATION PROBLEM

The image classification problems stems from the hardships of having a computer successfully recognising an image. Given a data set with multiple images, how and at what success rate can the system tell the images apart?
Data analytics, which is the discovery, interpretation and communication of meaningful patterns in data, and optimization which is the process for selecting the best element from set of available alternatives, are the building blocks for machine learning, which will help us solve this problem.

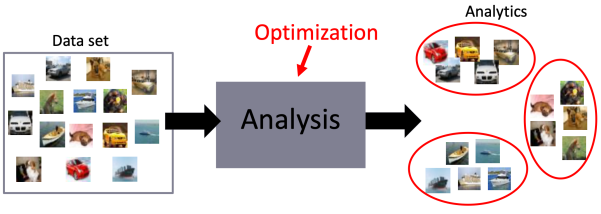

Figure 1: Image classification problem visualised

3 CLASSIFICATION SCHEMES

As mentioned, we will be looking at three out of the five classification schemes; Nearest class centroid classifier, Nearest sub-class centroid classifier and Nearest Neighbor classifier.

3.1 Nearest class centroid classifier

The nearest class centroid classifier, shortened NCC, takes a given set of n samples, each represented by a vector x_i with a corresponding label l_i. These vectors could anything, fx. a binary representation of images with matching labels being fx. "car01".

The NCC is fed all vectors with matching labels and calculates the means, resulting in a number of centroids from the following equation.

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

As an example, the ORL data set which is comprised of 40 classes (10 images per class), a total of 400 images is split into training and testing. Of these images, 70% are used for training and 30% for testing. The NCC takes 7 vectors from each of the 40 classes, calculates the means and creates 40 centroids. From these 40 centroids the test data, which would be the remaining 3 images from each class have their means calculated and matched to the closest resembling centriod.
Looking at equation 2 we see the formula for this. Here x_* is our test image.x

$$d(x_*, \mu_k) = \|x_* - \mu_k\|_2^2 \tag{2}$$

When the test image has been matched to a centroid, the NCC has then tried its luck predicting which image matches which image class.

3.2 Nearest sub-class centroid classifier

For cases where each class forms several subclasses, a variant of NCC called Nearest sub-class centroid classifier, shortened NSC can be used. An example of this could be a `Cars` superclass, with its subclasses being types of sports cars `Fast`, `Slow` and so on...
If the number of subclasses is given, we do as with NCC, calculate the means for the given n subclass samples and find our centroid. The formula for finding this is given by following

$$\mu_{km} = \frac{1}{N_{km}} \sum_{i,l_i=k,q_i=m} x_i \tag{3}$$

When the different centroids are found, the NSC tries to match the given test data with the closest matching centroid. This is done with the following equation

$$d(x_*, \mu_{km}) = \|x_* - \mu_{km}\|_2^2 \tag{4}$$

An important note is that to define subclasses of each class, one would typically apply a clustering algorithm fx. k-means. The number of subclasses per class is a parameter for NSC and needs to be decided beforehand[2].

3.3 Nearest Neighbour classifier

The last classifier we will be looking at is the Nearest neighbor classifier, shortened NNC. The NCC is used in thoes cases, where the number of subclasses per class is equal to the number of total samples. In the case of our test set ORL with a total of 400 images, all of the training data (70% ~ 280 images) would be its own centroid.

4 EXPERIMENTS

In the following sections we will be looking at the three mentioned classification schemes; NCC, NSC and NNC. We will be looking at what results the different schemes yielded for both the MNIST and ORL data set and try to analyse some of the visualised data.

For the MNIST data set, four files are given. These files are one set of 10.000 test images with labels and 60.000 training images with labels

- `t10k-images-idx3-ubyte`
- `t10k-labels-idx1-ubyte`
- `train-images-idx3-ubyte`
- `train-labels-idx1-ubyte`

For the ORL data set, two files are given, one raw data file and one label file. The data file is a $400x1200$ matrix with binary values for image representation and the `lbls` file is a list of 40 classes, where each class has 10 possible images to match.

- `orl_data.txt`
- `orl_lbls.txt`

5 NCC RESULTS

5.1 MNIST

The first thing we do is load the MNIST data set into the correct context. As the training data and test data are seperated into their own files, we call upon the pythong library `python-mnist` to load the files. When the files have been loaded, a simple function `display_image(mndata, imagetraining[1])` is called to visualise the data. Looking at figure 2, we can see the first image loaded from the MNIST data set.

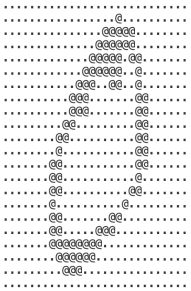

Figure 2: One of many images from the MNIST data set

When the MNIST data set has been loaded and visually verified, we begin to train our model. This is done as shown in figure 3. We firstly prepare our training and test images, by converting their previous vector values to x, y coordinates with pca. We then create our model and train it by calling `.fit(training_images, labels_training)` where the training data is fed to our model. Lastly we call upon the `.predict()` function and return our data.

```python
def nearest_class_centroid(images_training, labels_training, images_testing, labels_testing):

    pca = PCA(n_components=(2))
    training_images_pca = pca.fit_transform(images_training)
    test_images_pca = pca.fit_transform(images_testing)

    clf = NearestCentroid()
    clf.fit(training_images_pca, labels_training)
    print("Centoids: \n", clf.centroids_)

    return (clf.predict(test_images_pca),
            clf.centroids_,
            training_images_pca,
            test_images_pca)
```

Figure 3: Training and preparing data for analysis

An overview of what is called and in what order can be seen in figure 4. As mentioned, we first load, then show image, train model and prepare data, calculate success rate and lastly plot the results.

```python
    # Load data
    images_training, labels_training, images_testing, labels_testing, mndata = load_t10k_images()

    # Show one of images loaded
    diplay_image(mndata, images_training[1])

    # Train algorithm and apply test data
    data = nearest_class_centroid(images_training, labels_training,images_testing, labels_testing)

    # Calculate success rate
    calculate_success_rate(images_training, labels_training, images_testing, labels_testing)

    # Plot results
    plot_data(labels_training, data[0], data[1], data[2], data[3])
    plt.show()
```

Figure 4: `./main.py` call order

When looking at the success rate of the NCC with the MNIST data set of 10.000 test images and 60.000 training images, we end up with a correct prediction rate of 15.42%. The way the success rate is measured is by comparing all the test image labels with the predicted model labels.

> Total image labels: `10.000`
> Successfully matched image labels: `1542`
> Percentage: `15,42%`

For plotting the data, the python library `matplotlib.pyplot` has been used. Using random colors to visually split the different clusterings of test images with their assigned centroids can be seen in figure 5. The larger circles with a black outline indicates the centroids, where each color represents the different clusterings. An important note is that the scatter-plot does not show which test images have been wrongly placed, only how the NCC tries do match the image patterns.

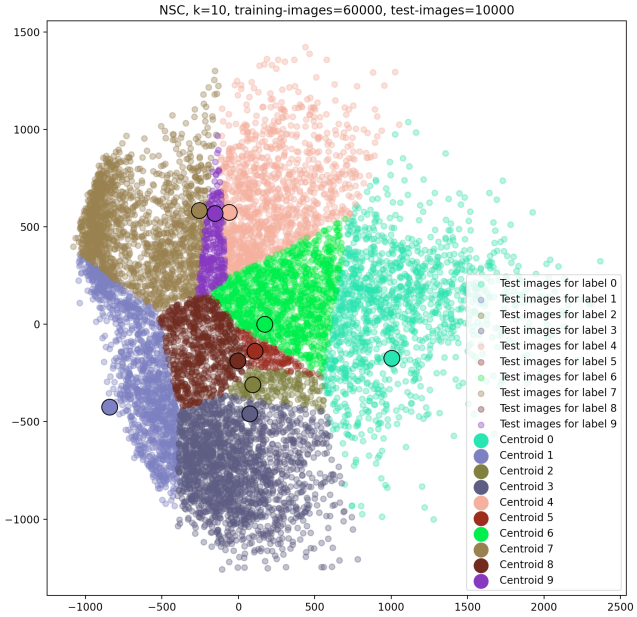

Figure 5: NCC MNIST plottet with scatterplot

5.2 ORL

For the ORL data set we load it from the two given files. These files are not split with test and training images as MNIST and two functions are therefor made to load the faces into context. These load functions are then tested with a `display_image(imageNumber, loaded_images)` function to visually determine if the data has been loaded correctly, see figure 6.

Figure 6: One of 400 faces from the ORL data set

The ORL data set has now been loaded and is ready to be trained via the NCC. We do as with MNIST, by firstly converting our (1x1200) dimension vectors to pca values and then training our NCC model with the pca training image values. After our NCC model has been trained by the feeding of data, we return the prediction of the model to be analysed by our success-rate function and plot.

The way the success-rate is calculated is by comparing the original test-image labels with our models predicted test-image placement (by label). Figure 7 is the visual representation of what is compared. Every index i compared, if they match, our model has predicted a test-image to be in the correct label group. So fx. index 0 from the `predicted labels`-array is compared with index [0] from `Test labels`-array. This function is the same as we saw for MNIST, but showed herd reasoned the shorter output.

```
Predicted labes:
 ['20' '30' '30' '7' '36' '7' '24' '24' '24' '15' '15' '31' '29' '33' '29'
 '34' '34' '34' '11' '11' '11' '31' '24' '24' '24' '11' '37' '11' '37' '28'
 '31' '40' '40' '12' '21' '21' '21' '20' '20' '20' '6' '40' '6' '25' '3'
 '35' '9' '9' '15' '11' '11' '11' '30' '29' '33' '10' '7' '6' '37' '37'
 '19' '27' '27' '27' '1' '18' '18' '34' '37' '11' '27' '16' '16' '9' '9'
 '24' '21' '21' '24' '4' '36' '6' '23' '7' '6' '19' '13' '13' '19' '37'
 '27' '16' '27' '40' '23' '9' '7' '19' '19' '13' '18' '18' '18' '24' '38'
 '24' '2' '11' '24' '6' '23' '23' '37' '2' '28' '13' '1' '1' '24' '21'
 '24']
Test labels:
 ['1', '1', '1', '2', '2', '2', '3', '3', '3', '4', '4', '4', '5', '5', '5', '
6', '6', '6', '7', '7', '7', '8', '8', '8', '9', '9', '9', '10', '10', '10', '
11', '11', '11', '12', '12', '12', '13', '13', '13', '14', '14', '14', '15', '
15', '15', '16', '16', '16', '17', '17', '17', '18', '18', '18', '19', '19', '
19', '20', '20', '20', '21', '21', '21', '22', '22', '22', '23', '23', '23', '
24', '24', '24', '25', '25', '25', '26', '26', '26', '27', '27', '27', '28', '
28', '28', '29', '29', '29', '30', '30', '30', '31', '31', '31', '32', '32', '
32', '33', '33', '33', '34', '34', '34', '35', '35', '35', '36', '36', '36', '
37', '37', '37', '38', '38', '38', '39', '39', '39', '40', '40', '40']
```

Figure 7: Comparison between predicted and test-image labels

Unfortunately the success rate for the ORL data set resulted in 0 out of the 120 labels compared matching. The reason for this could be argued to be the small training set of only 280 images. The larger the training set, the better and more accurate the predictions will be.

Total image labels: `120`
Successfully matched image labels: `0`
Percentage: `0,00%`

6 NSC RESULTS

6.1 MNIST

For the NSC classifier we calculate, train and predict our model differently from the previous NCC classifier. As mentioned in section 2 an important note is to define subclasses of each class, as the number of subclasses per class is a parameter. K-means which is a clustering algorithm would typically be applied if the given number of cluster k wasn't given.

Different tools can be used to find the optimal k-value, where the elbow graph is one of them. The elbow graph shows a line which at a breaking point visually shows the optimal k...hence the name. Looking at figure 8 we see the drawn elbow graph with the optimal k being three for set 2, since the breaking point is most evident there. One could argue though, that a k-value of two also would be reasonable. For scenarios like this, where an albow graph can be hard to read, a scatterplot can visually help tell the optimal k.

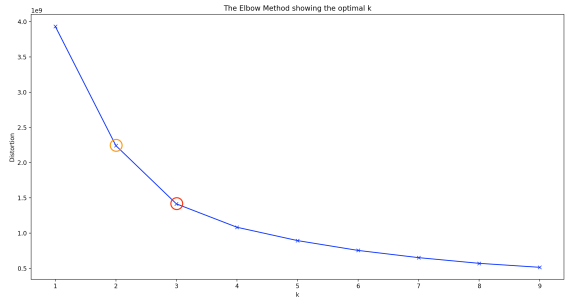

Figure 8: Elbow graph for NSC sub-class set 2

Looking at the scatter plot for the NSC classifier scheme, after our optimal k value was determined to me three, can be seen in figure 9.

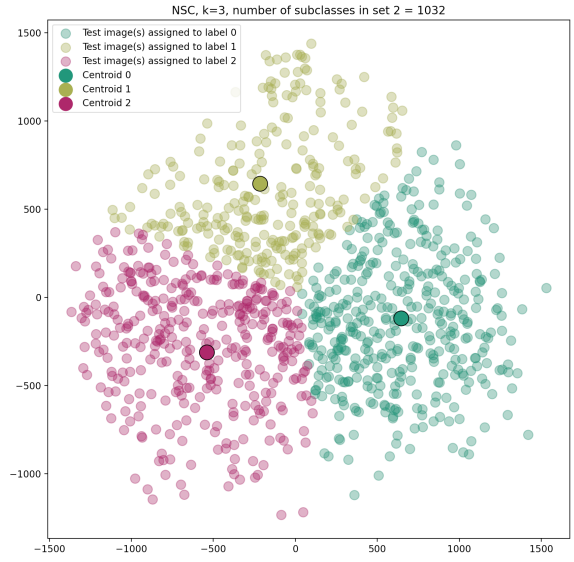

Figure 9: NSC scatterplot for subset with labels 2, where $k = 3$

The same scatterplot has been attempted again, but with $k = 2$, see figure 10. The clustering of either two or three could arguably be considered reasonable.

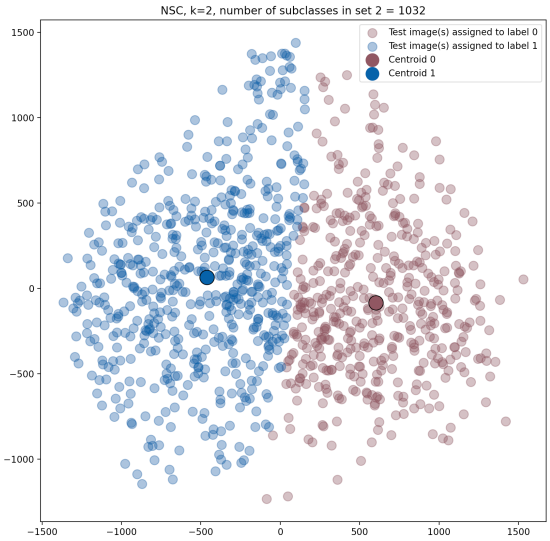

Figure 10: NSC scatterplot for subset with labels 2, where $k = 2$

Looking at the probability rates for all of the subclasses in the set $\{2,3,5\}$ with $k = 3$, we see that the distribution of the subclasses seem to be evenly distributed. This fits our perception of how the scatterplot plots the test images as seen in figure 9.

Subclass 2

Total image labels: `1032`
Probability for class 0: `39.437%`
Probability for class 1: `25.000%`
Probability for class 2: `35.562%`

Subclass 3

Total image labels: `1010`
Probability for class 0: `34.356%`
Probability for class 1: `33.960%`
Probability for class 2: `31.683%`

Subclass 5

Total image labels: `892`
Probability for class 0: `36.322%`
Probability for class 1: `30.381%`
Probability for class 2: `33.295%`

6.2 ORL

For the ORL data set, we try to mimic the steps for the MNIST data set, by firstly looking at the elbow graph, see figure 11. We can clearly see the breaking point here being $k = 3$. An important note is also that the elbow graph will show a different break for each sub-class tested for. However for all of the three sub-classes in set $\{2,3,5\}$, they all share the same breaking point at $k = 3$.

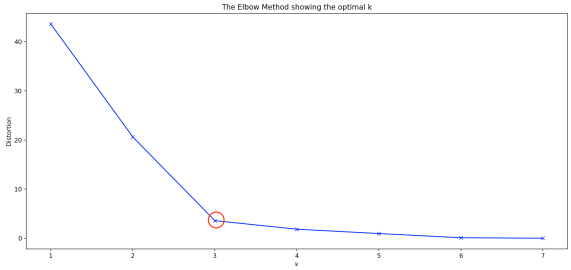

Figure 11: Elbow graph for NSC sub-class set 2

Looking at figure 12, we see how each of the 3 test images are matched to the nearest centroid. In total our NSC model was only trained with 7 images and given 3 images to predict. The scatterplot therefor looks a bit empty.

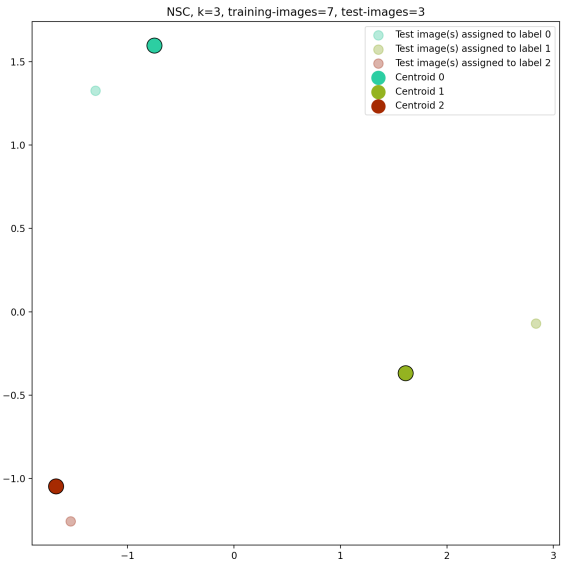

Figure 12: NSC Scatterplot for sub-class set 2, with $k = 3$

Looking at how the three subclass in set `{2,3,5}` have been categorised, we see that they resemble the depicted scatterplot.

Subclass 2

Total image labels: 3
Probability for class 0: `33.333%`
Probability for class 1: `33.333%`
Probability for class 2: `33.333%`

Subclass 3

Total image labels: 3
Probability for class 0: `33.333%`
Probability for class 1: `33.333%`
Probability for class 2: `33.333%`

Subclass 5

Total image labels: 3
Probability for class 0: `33.333%`
Probability for class 1: `33.333%`
Probability for class 2: `33.333%`

7 NNC RESULTS

7.1 ORL

For the NNC classification scheme, the number of subclasses per class is equal to the number samples. Looking at figure 13, we see total of 280 training images, which one could argue all are their own centroid. The 120 test image have then been placed to the closest matching centroid.

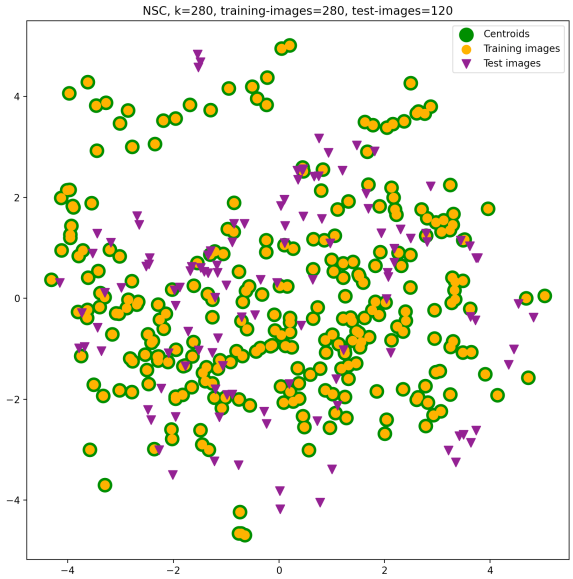

Figure 13: NNC scatter for ORL data set

When looking at the success rate for correctly matching test images, we see the following.

Total image labels: `120`
Successfully matched image labels: `3`
Percentage: `2.5%`

The success rate can also be calculated from the python library `sklearn`, where the function `neighbors.KNeighborsClassifier()` is used. Calculating the success rate with the same data yields close but not similar result.

Total image labels: `120`
Successfully matched image labels: `0`
Percentage: `0%`

One could argue, that the python library function has some optimisations in place to handle large data sets. Giving the function a measly 120 images, might be the reasoning behind the 0% match-rate.

7.2 MNIST

For the MNIST data set; 10.000 test images and 60.000 training images, we see the following success rate matches.

Total image labels: 10000
Successfully matched image labels: 1115
Percentage: 11.15%

The reasoning for a much higher match rate than the ORL data set, would arguably be because of the larger training set. The better the model is trained, the better it gets at predicting image labels.

8 DISCUSSION

Looking at the bigger picture, all of the classification schemes each have their own place when analysing data. For this project two data sets where given. ORL which was small, but with more fine grained subclass sets, and the MNIST data set which was large, but with more spread labels.

When comparing the different schemes to one another, we would see that if the data is spread out/imbalanced the NNC classifier scheme could cause some matching problems, as the voting system of the k-neighbour will be effected. A better scheme would be the NCC, as this would create centroids and therefor group the spread. Would one like to have a more detailed grouping and matching of information, the NSC would be the better choice, as this scheme specialises in subclasses.

Looking back at some of the results, from both the scatterplots and success match results, we see that the scatterplots don't always fairly represent the success match rate. The scatterplots are great for visually showing where the test data has been placed based on the trained model predictions and not error proving. These plots do not show the wrongly placed test data and one should therefor be wary of what they understand and read by the plots.
The success matching in some of the classifications schemes is arguably very low. The reasoning for this could be, that more data was needed. The more data which is fed the model, the better the model is at guessing the test data.

9 CONCLUSION

To round it all off, we can conclude that no classification scheme is per say, better than one another. Each classification scheme should be chosen on what data is to be analysed. For future use of classification schemes, a good idea is to try to plot the data first, to visually see what the predicted data might look like, to better decide on what scheme to choose. Furthermore, the elbow method, which was used to determine the correct number of clusters k, can be used instead of guessing, if the plotted data cant properly be read. Furthermore, some libraries can yield different results versus implementing the algorithms yourself. However with large data sets, the library functions could be optimised for this and return a faster result with a trade off of some minor erros.

REFERENCES

[1] Python documentation. [n.d.]. https://docs.python.org/3/library/random.html
[2] Alexandros Iosifidis. 2018. In *Introduction-to-machine-learning ((chap 4) 'Linear Methods')*, Theerasak Thanasankit (Ed.). 37–38.
[3] matplot documentation. [n.d.]. https://matplotlib.org/api/pyplot_api.html
[4] sklearn. [n.d.]. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
[5] sklearn.cluster. [n.d.]. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
[6] sklearn.cluster. [n.d.]. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html