NTNU | Norwegian University of Science and Technology

TTT4275 Estimation, Detection and Classification (Spring 2020)

Classification of Iris Variants and Hand-Written Numbers

Team 5 - Process Report

Jon Petter

Yamen Zaza

Abstract

It has become evident that data is a major aspect of our lives right now. Whether on research frontiers or casually in homes, data is generated and most importantly it becomes available for analysis. Numerous artificial intelligence-based analysis is present and has influenced many fields of science with incredibly accurate results.

The project only focuses on one aspect of the EDC course which is classification. In this project, less sophisticated algorithms were developed in Python to perform classification tasks on both an Iris and hand-written datasets. Furthermore, the results attained match theoretical concepts to a high degree and are discussed in detail. Major results reiterate on the importance of training set size, feature analysis and linear separability, classifier choice, and performance. Increasing the training set size and its variance improves classifier performance when it comes to generalizing on real data. In addition, having linearly separable features in the data set is crucial to achieve satisfactory convergence of linear classifiers. Results from the handwritten numbers dataset confirm that reference-based classifiers can lead to acceptable models when resources are scarce.

It was apparent at the end that machine learning holds many prominent tools within that can be beneficial in many aspects of sciences and delivers astounding results.

Contents

Figures and Tables

Introduction

It has become very important to make use of captured data to see underlying patterns. Consequently, this would help not only automize certain processes but also speed them up and drastically increase accuracy. By understanding and cleaning available datasets, the data became useful. Wouldn't it be nice if we can sort huge factory inventories by size or recognize faces? Such application has proven to work with sharp results and much more potential can be used to our advantage if we use our data.

The course Estimation, Detection, and Classification (EDC) which dove into the theory of each topic is an integral part of Signal Processing and Communications studies. This project however is devised to increase exposure to hands-on experience that otherwise cannot be achieved in class. The main goal of this project is to develop pieces of written software that can carry out classification tasks while mainly using the theory studied in class as a basis for development.

The report will answer the questions presented in the task description and will have a general layout divided into 4 sections as follows:

- Section I: classification theory relevant to project tasks.
- Section II: short summary on motivation and tasks.
- Section III: Discuss of implementation and results.
- Section IV: Conclusion and closing remarks

The general approach to the project was to first cover technical details from the provided course material. This was followed by studying every part of the tasks and coming up with a starting point for pseudo code. Finally, rough scripts are commented and organized with a focus of writing an easy to run code.

Classification Theory

Supervised learning is a branch of machine learning that works by labeled examples. The classification problems in this project used a supervised learning approach. Generally speaking, classification requires training on datasets before performance can be tested. This section of the report shall serve as a gentle introduction to the theory used.

Linear Classifier (Perceptron)

In order to classify the different Iris variants, we need to develop a classifier. In essence, a classifier is capable of predicting under which category a certain set of features fall under. The following figure is presented to further explain linear classifiers:

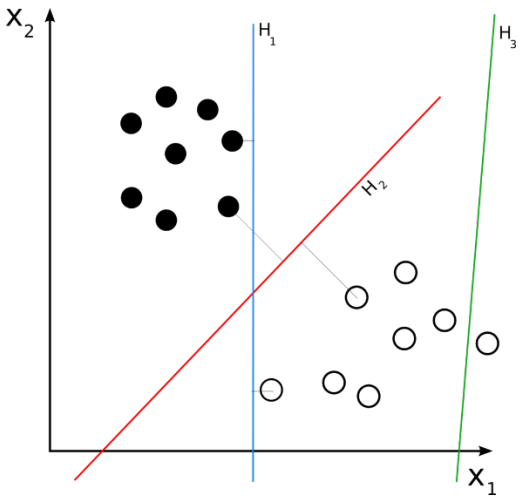

Figure 1: Three linear classifier. Source [1]

The figure shows three linear classifiers where the blue and red ones correctly classify data while the green one is wrong. It is important to note that the data is also clearly separable meaning that there is no overlap which prevents the separation of two different kinds of data using a single decision surface (1D: point, 2D: line, …). This is crucial as convergence can be only reached, regardless of time taken, if data is linearly separable.

For the Iris task, a discriminative training method called the Perceptron was used to optimize the weights of Iris features by minimizing the Mean Squared Error (MSE). A perceptron attempts to iteratively minimize the error in the system through the use of cost functions which quantify the wrongness of the predictions. This is done through a forward, backward iterative propagation process called gradient descent which can basically find the global minimum of a function with the right hyperparameters. The following figure shows a perceptron's structure:

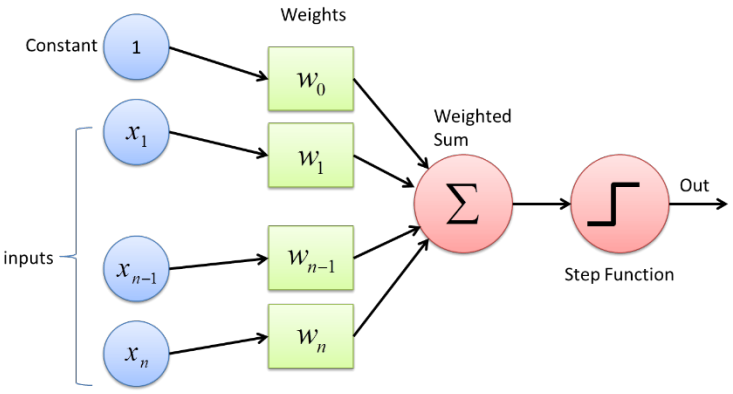

Figure 2: Perceptron structure. Source [2]

One hot encoding is used to describe multiple classes in a binary manner without introducing unwanted relative distance. Furthermore, a nonlinear activation function is used to introduce nonlinearity into the model and thus make it more robust. Sigmoid functions are one example of activation functions used with probability-realted values (ex: max probability decision rule) due to their nature of mapping outputs between 0 and 1.

The model is tested using part of the original dataset and further analysis is seen through:

- Confusion matrix: a table showing predicted labels vs actual labels. A lot of descriptive statistics can be derived like error rate.
- Histograms: shows distribution of features and helps in linear separability analysis.

K – Nearest Neighbors Classifier

Lazy learning characterizes template-based classification for the simple reason that the whole training set needs to be queried for every classification. It takes significantly more time to train and test when a huge set of features are present but can be run in parallel to solve different problems where data is constantly changing [3].

Nearest neighbor classifier is a special case of KNN classifiers where k=1. In principle, the classifier finds the Euclidean distance between the test point and all other points. The k nearest points will be used to classify new data. The figure below clarifies the approach:

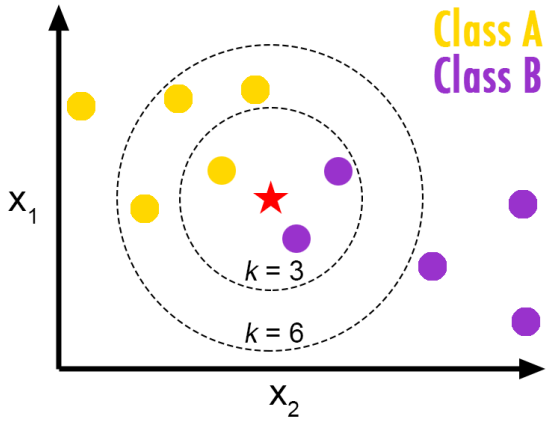

Figure 3: KNN representation. Source [4]

K-means Clustering

Unsupervised learning works on unlabeled data and utilizes a distance metric of preference to uniquely group data into k cluster. K-means algorithm uses Expectation-Maximization (EM) where k points in space are randomly picked as centroids and iteratively moved until convergence. The two iterative calculations are:

1- Compute distance for every point to every centroid and pick the closest centroid for every point.
2- Compute the average spatial location of the centroid with respect to all points belonging to it thus moving the centroid.

Convergence is achieved when the squared distance of all points belonging to a cluster is less than that to other centroids.

K-means clustering can be used to generate reference-based classifiers. In this approach, a smaller set of training templates are picked by being the centroids of k-means clustering of a

class. We end up with a representative reference to every class in a training set which can lead to a much smaller training set for template-based classifiers.

Classifier Performance

For our purposes, the measurement of classifier performance will be the error rate. The Minimum Error Rate (MER) can be found when the parameters (mean and variance) of the source of data are known. Unfortunately, this is not the case for real world problems where the data is presented without prior information of the source. Nevertheless, an estimated error rate can give an understanding of how well a classifier does.

$$Estimated\ Error\ Rate = \frac{Total\ Number\ of\ Errors}{Total\ Number\ of\ Training\ Data}$$

The above equation shall serve is the estimated error rate for realistic problem. The true MER cannot be determined in real applications since the same source can produce many different training sets in which their classifiers will give different estimated error rates. The true, unknown MER depends on the training set while the estimated error rate depends on the testing set [5].

Testing of trained models happens on two stages:

1- Test with the training set to find how much the trained model fits the dataset.
2- Test with the validation set to measure the performance.

A major consequence of data size can be seen in the following statement. Assuming the right choice of classifier, a bigger training set can potentially improve the estimated error rate while a bigger test will definitely give more accurate error rate estimates. So, basically a bigger training set means a lower error rate in principle but to accurately capture that we need an adequately large testing set. However, the increase in performance when using bigger training sets is bounded by learning curves that are application specific. The figure below illustrative the aforementioned points:

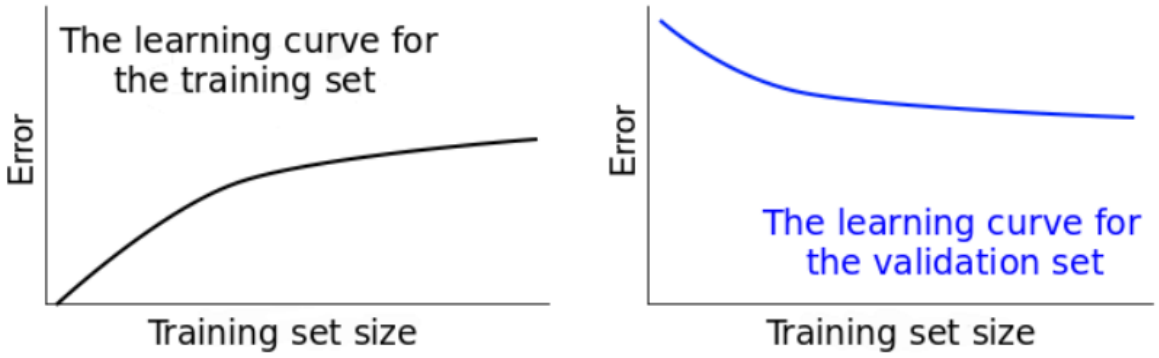

Figure 4: Learning curves for training set [left] and validation set [right]. Read 3.5 of [5] for theory.

When the difference between the training set estimated error rate (EER_D) and the test set estimated error rate (EER_T) is smaller than 2% - 4% then we can say that the classifier

generalizes well to real world data. Usually $EER_D < EER_T$ and a well representative dataset is required for robust classification on real data [5].

Project Tasks

With the interests of both team members aligning with the classification project, it was thus concluded that this will be the group's choice. Wanting to gain experience beyond theory was the main motivation for picking the classification project. Three tasks were presented in which a team needs to pick only two, with one being mandatory. Therefore, teams had to choose between working on handwritten numbers recognition or natural language processing of vowels. This section of the report will describe the team tasks:

The Iris Task

In this task, a dataset with 150 examples equally distributed among three Iris classes was provided. The Iris flower classes are Setosa, Versicolor, and Virginica. Each example has a 4-dimensional feature vector for sepal length, sepal width, petal length, and petal width. An example of how the features are extracted is below:

Figure 5: Versicolor Iris Features

The first part of this task asks for a linear classifier to be designed and trained on the dataset. The dataset set needs to be split into two sets for purposes of training and testing respectively. The following is to be taken into consideration:

- A perceptron using the gradient descent on the MSE cost function will train the linear classifier.
- For our purposes we will only have a training set and a test set.
- Testing performance: training and testing sets both used

As for the second part of the Iris task, linear separability of the features in the dataset is tested. Histograms will be used to find overlapping features between classes. Following that, multiple experiments that aims to create linear classifiers with different numbers of omitted feature need to be performed and analyzed.

Classification of Handwritten Numbers 0-9

A huge training set with 60000 carefully engineered and preprocessed examples is available along with 10000 test examples. Both sets are labeled and ready for the training and testing of different classifiers. The following is to be taken into consideration:

- The training set is not equally divided among the 10 classes of handwritten numbers.
- Each example is a 784-dimensional feature vector (i.e. 784 pixels) that when reshaped into a 28x28 matrix will reflect an image of a number.
- 8-bit encoding of number images is used with values ranging between 0-255 for every pixel.
- A wide variety of examples exist in the training set which reflects easier and more difficult handwritten numbers scenarios.
- Binary files need to be appropriately read to get the correct datasets.

The first part of this tasks requires an NN classifier to be designed and tested. Testing will have two attributes: confusion matrix and error rate. For memory related issues, dataset chunking will be used.

As for the second part, K-means will be used to reduce the size of the original training set from 10000 to 640 only. This needs to be done using K=64 clusters for every class where every cluster centroid will be a reference template. Following that, the NN classifier will be trained and tested on the new training dataset. Furthermore, a KNN classifier with K=7 need to be designed, tested, and compared against:

1- NN classifier.
2- NN classifier with k-means.

Implementation and Results

In this section of the report, the implementation decisions and approaches will be explained in detail. The task questions will be answered in a sequentially fashion with results extracted from the written scripts. A couple of general points on implementation will be addressed first before going into detail for every task and its parts.

Python was selected to be the programming language of choice to write all scrips for this project. This is because Python has a great support for machine learning applications through tons of packages that are ready to be used right out of the box. Moreover, Jupyter Notebook was used to develop and debug the code. Despite not having essential Integrated Development Environment tools, a lot of handy extensions can be added through extensions. Unfortunately, this would mean that whoever needs to correct this report will need to get a set of specific tools which is why the report is written and formatted on Microsoft Word instead. Although MATLAB specializes in matrix operations, Python does just fine as well. The following packages/libraries are used:

- Pandas: great for data manipulations and analysis.
- NumPy: useful for mathematical operations on huge matrices.
- Matplotlib: needed for some part of visualization.
- Timeit: measures execution time for selected parts of code.

Note: Complete code scripts will be provided separately.

The Iris Task

A linear classifier was designed based on relevant theory. The learning rate was fixed at $\eta = 0.03$ which balanced between speed and accuracy. An extra feature that was added when implementing gradient is training precision. Training precision is the percent difference between two consecutive iterations. This was used along with a maximum iterations parameter to allow for automatic detection of convergence based on user preference. In other words, the learning algorithm will stop when either maximum iterations or training precision is achieved. In addition, the splitting of datasets was made to support any arbitrary size setting.

The confusion matrix and error rate for both the training and test sets when training using the first 30 samples of every class and with a training precision of 3.5% can be seen below:

```
Precision reached. Ending training!
Training finished after 484 iterations in 0.948896
The found weights are:
 [[ 0.47654757  1.6091718  -2.41308496 -1.11580737]
 [ 2.24061604 -3.83246281  0.17835303 -2.34073176]
 [-3.87797483 -3.17632317  5.443761    4.16257452]]
The error rate is 5.00% using the test set with a confusion matrix of:
```

	ClassOne	ClassTwo	ClassThree
Class One	20	0	0
Class Two	0	18	1
Class Three	0	2	19

```
The error rate is 3.33% using the training set with a confusion matrix of:
```

	ClassOne	ClassTwo	ClassThree
Class One	30	0	0
Class Two	0	28	1
Class Three	0	2	29

Figure 6: Results for part 1 - section c. Training is done with first 30 samples.

The training and testing phases were repeated but this time with the last 20 samples used for training. Below are the results using the same training precision:

```
Precision reached. Ending training!
Training finished after 625 iterations in 1.281020
The found weights are:
 [[ 0.54984084  1.68609397 -2.6362364  -1.24018381]
 [ 1.62300826 -3.56584033  1.31826947 -3.88912396]
 [-3.55261963 -3.72592766  4.94426835  5.0038154 ]]
The error rate is 1.67% using the test set with a confusion matrix of:
```

	ClassOne	ClassTwo	ClassThree
Class One	20	0	0
Class Two	0	19	0
Class Three	0	1	20

```
The error rate is 7.78% using the training set with a confusion matrix of:
```

	ClassOne	ClassTwo	ClassThree
Class One	30	0	0
Class Two	0	27	4
Class Three	0	3	26

Figure 7: Results for part 1 - section d. Training is done with last 30 samples.

The sizes of both the training and testing sets are the same in the previous results thus it is irrelevant for the performance of the classifier. However, we notice that the error rate of the training set in second case is lower which means that the model learned the training set better. This is not necessarily good as the training set in the second case probably has lower variance which and therefore an oversimplified (overfitted) model. The negative effects are seen on the estimated error rate where it is better for the first case.

This is kind of testing, nonetheless, can help us learn a lot more about the data we are dealing with. In principle, an additional validation set will be used to iterative design, test, and select better classifiers before using a test set to maximize performance. Another important aspect is the training set size which is problematic for performance in the Iris dataset. The algorithm has access to a limited amount of data that will be further split into two. One method that could counteract this phenomenon is the leave-on-out strategy - many variations exist for this strategy in the form of $Dataset\ Size - Leave\ \#$. This strategy would train with 49 samples and test with 1 for 50 times with the EER being the mean of the error rate for each round.

Based on the theoretical criterion for classifier generality, we can say that the first classifier generalizes well while the second classifier is not terrible but will struggle in correctly classifying unseen data more often.

In testing linear separability, multiple trials have been carried out to understand the effect of removing certain features. It was not very clear which feature overlapped the most in certain class, so some trial an error had to be made. The following table shows the different experiments done in sequence with the results acquired:

Table 1: Experiment results. Each trial to remove most overlapping feature is separated by [-]. See notes below.

Feat.$_{removv.}$	EER_D [%]	Class Mostly Wrong	Bias	EER_T [%]	Learning Rate
0-1-2	5-16.67-66.67	2-2-ALL	3-3-3	4.44-30-66.67	0.03-0.038-0.03
0-1-3	5-16.67-66.67	2-2-ALL	3-3-3	4.44-30-66.67	0.03-0.038-0.03
0-3-1	5-8.33-66.67	2-2-ALL	3-3-1	4.44-12.22-66.67	0.03-0.03-0.06
0-3-2	5-8.33-66.67	2-2-ALL	3-3-3	4.44-12.22-66.67	0.03-0.03-0.2
1-0	3.33-16.67	2-3	3-3	3.33-30	0.02-0.04
2-1	10-8.33	3-2	2-3	8.89-8.89	0.026-0.025
2-0	10-3.33	3-2	2-3	8.89-3.33	0.026-0.0001
3-0	8.33-8.33	2-2	3-3	5.56-12.22	0.02-0.03

Note 1: More experiments were carried out but there carry the most significance.

Note 2: Experiment stops when features do not overlap or when the error gets too high.

Note 3: The scripts provided can be used to easily generate results for all question variations.

Note 4: Bias in the table refers to the tendency of the classifier to incorrectly classify an image under one class of the others.

Note 5: The first column represents the process of removing features. Ex: 0-1 means that the first feature was removed then the second.

When the experiments are done by eye, the first feature (0) was removed followed by the second (1). After that, no features overlapped. This is best described by either row 1 or 5 of Table 1 above. It was noticed that less features required more time which means it was harder to learn the dataset unless the learning rate is increased. Moreover, with less features, the error rate increases on average which is due to having less model complexity. This can be an indirect complication of high correlation between some of the Iris features in the datasets. It has been proven that high correlated features can hinder a classifier from attaining a good classification accuracy [6].

Sudden spikes in estimated error rates could signal that either the data set contains very little information to make a good model or that an essential, relevant feature for classification was removed.

Generally speaking, the first and second features (0 and 1) cause the most trouble for the classifier and result in the lowest estimated error rates when removed. Also, both are mainly linearly inseparable from class 3.

Handwritten Number 0-9

This task required more specialized Python packages since writing up the algorithms for NN, KNN, and k-means was not efficient enough to stop memory errors from popping up constantly. That being said, the *sklearn* library was used along the way to do a lot of the heavy lifting as it is an excellent tool for machine learning applications in general.

Mini-batch was used for the k-means algorithm due to the big number of training examples and memory restrictions. It is notable to mention that the Python approach was surprisingly much faster that estimated MATLAB processing times. When researched for the reason, the results mainly talked about how well optimized the packages are and well suited for huge data.

Using Euclidean distance as the distance metric, an NN classifier was first designed, check for performance, and tested. The following includes the performance results of this classifier:

```
Training of the NN algorithm has started.
It took  35.05042243003845  seconds!
The model is saved at ./temp/final_result_sk_pdam.out' using numpy module.
Overall error rate for the NN classifier is  3.0900000000000034 %
```

	Zero	One	Two	Three	Four	Five	Six	Seven	Eight	Nine
Zero	973	1	1	0	0	1	3	1	0	0
One	0	1129	3	0	1	1	1	0	0	0
Two	7	6	992	5	1	0	2	16	3	0
Three	0	1	2	970	1	19	0	7	7	3
Four	0	7	0	0	944	0	3	5	1	22
Five	1	1	0	12	2	860	5	1	6	4
Six	4	2	0	0	3	5	944	0	0	0
Seven	0	14	6	2	4	0	0	992	0	10
Eight	6	1	3	14	5	13	3	4	920	5
Nine	2	5	1	6	10	5	1	11	1	967

Figure 8: NN classifier confusion matrix and error rate.

Note: All the results presented in this section can be seen inside the attached Jupyter Notebook for more clarity in viewing the results and testing the code.

It was quiet challenging to find results that I disagreed with as a human, but a minute thing was noticed. It was interesting to see how some of the wrongly classified images captured the rotation angle of some digits. It can be speculated that such misclassifications are due to the more difficult examples in the training dataset.

The following figure shows some of plotted outputs from the results of the NN classifier:

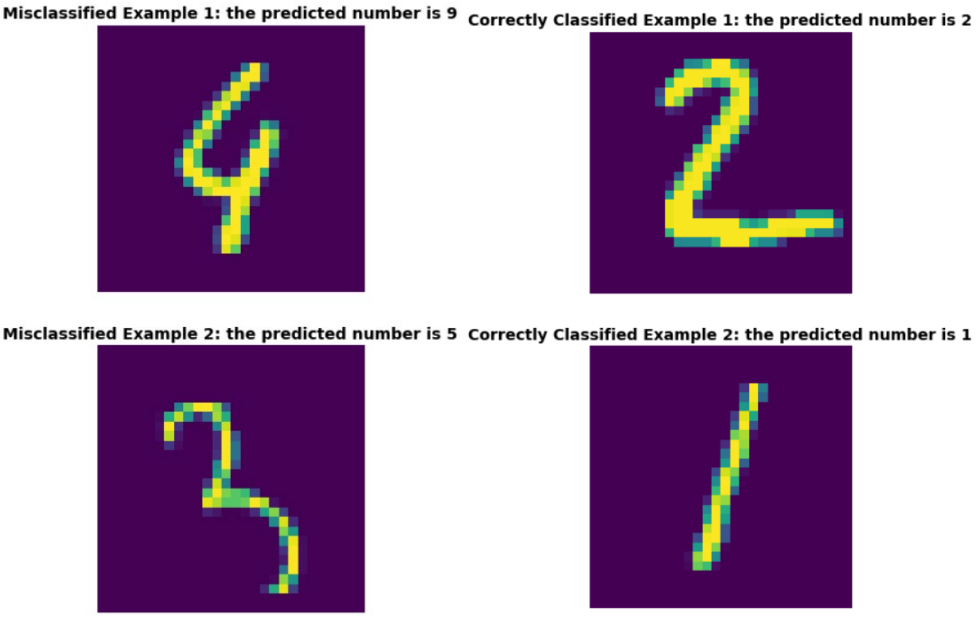

Figure 9: NN classifier output examples.

For the second part, mini-batch k-means was used to create a reference-based classifier. The following are the performance stats of the new NN classifier:

```
Preparing datasets using kmeans with mini-batch and 64 clusters.
It took  8.203887701034546  seconds!
The new kmeans dataset is saved at './temp/sk_kmeans_mini_dataset.joblib' using joblib module.
Training of the NN algorithm has started.
It took  46.61685299873352  seconds!
The model is saved at ./temp/final_result_sk_pdam.out' using numpy module.
Overall error rate for the NN classifier is  5.300000000000011 %
```

	Zero	One	Two	Three	Four	Five	Six	Seven	Eight	Nine
Zero	964	1	5	0	0	3	5	1	1	0
One	0	1128	0	2	1	0	3	0	0	1
Two	8	5	986	4	1	0	3	11	13	1
Three	0	3	11	914	1	31	0	11	29	10
Four	1	9	2	0	903	0	8	3	3	53
Five	3	0	1	10	2	851	8	1	7	9
Six	6	4	0	1	4	7	933	0	2	1
Seven	1	19	11	1	8	0	0	960	0	28
Eight	3	1	4	20	3	25	3	4	905	6
Nine	3	5	5	4	33	3	1	25	4	926

Figure 10: Results for the NN classifier with k=64 k-means clustering WITH mini-batch. The figure shows the confusion matrix and error rate.

In comparing the two different approaching in designing an NN filter, the processing time that includes the k-means algorithm is slightly higher. The extra time was used for the k-means algorithm to finish. Note here that the processing time without the use of mini-batch is significantly higher but with a lower error rate after passing the templates to the NN classifier. The result without mini-batch can be viewed in the following figure:

```
Preparing datasets using kmeans with 64 clusters.
It took  287.1633059978485  seconds!
Training of the NN algorithm has started.
It took  43.55336308479309  seconds!
The model is saved at ./temp/final_result_sk_pdam.out' using numpy module.
Overall error rate for the NN classifier is  4.790000000000006 %
```

	Zero	One	Two	Three	Four	Five	Six	Seven	Eight	Nine
Zero	960	1	5	0	1	4	5	1	2	1
One	0	1132	1	0	0	0	1	0	0	1
Two	7	7	977	10	2	0	3	11	15	0
Three	0	0	2	956	1	18	0	9	15	9
Four	0	7	2	0	919	0	10	4	2	38
Five	5	0	1	19	2	845	9	2	5	4
Six	6	4	1	0	4	0	940	0	3	0
Seven	1	18	9	0	9	1	0	964	3	23
Eight	6	1	5	18	3	21	3	7	904	6
Nine	4	4	5	7	30	3	1	26	5	924

Figure 11: Results for the NN classifier with k=64 k-means clustering WITHOUT mini-batch. The figure shows the confusion matrix and error rate.

On the other hand, the error rate is slightly higher when using k-means due to a reduction in the size of the training set. Nevertheless, a compression from 10000 to 640 is significant and allows for relatively acceptable results while using less memory and storage space on the computing device.

TEST with train and test error and make a re-run of all to test code with minor change in conf. function parameters

In the last section of task 2, a KNN classifier with K=7 was designed and tested. The following shows the results:

```
Training of the kNN algorithm has started.
It took 46.62478232383728  seconds to train!
The model is saved at './temp/sk_KNN_model.joblib' using joblib module.
Prediciton of the test set using the kNN algorithm has started.
It took 876.6635076999664  seconds!
The predicted results are saved at './temp/sk_knn_pred.out' using numpy module.
The total error rate for the KNN algrothm that used the sklearn package was  3.0600000000000023 %
```

	0	1	2	3	4	5	6	7	8	9
0	974	1	1	0	0	1	2	1	0	0
1	0	1133	2	0	0	0	0	0	0	0
2	11	8	988	2	1	0	2	16	4	0
3	0	3	2	976	1	12	1	7	4	4
4	1	8	0	0	945	0	5	1	1	21
5	5	0	0	8	2	866	4	1	2	4
6	6	3	0	0	3	2	944	0	0	0
7	0	25	3	0	1	0	0	989	0	10
8	6	4	6	11	7	12	1	6	916	5
9	5	6	3	6	8	4	1	11	2	963

Figure 12: Results for the KNN classifier. The figure shows the error rate and confusion matrix. Each number refers to a class.

The time taken for the KNN algorithm to classify the test images is dramatically higher that the previous two classifiers; however, it achieved the lowest error rate. One can argue that the dataset is closely packed, thus a higher K value would be able to classify data better.

In real life application where speed is a limiting factor, going with a reference-based classifier would be the optimal approach. On the contrary, if performance is what matter like in medical research then using big datasets with KNN would be best.

Conclusion

At the end of this project, it is paramount to indicate that we only touched on a small fraction of machine learning through the use of supervised learning in classification problems. One can only think of the vast applications that other branches of machine learning are capable of solving. The key idea here is that understanding the data at hand and developing the right classifier might be a tedious job but is surely one that pays off. Countless manual work could be replaced by a good model.

References

[1]: https://en.wikipedia.org/wiki/Linear_classifier.

[2]: https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53.

[3]: https://en.wikipedia.org/wiki/Lazy_learning.

[4]: https://medium.com/@bengikoseoglu/why-log-loss-metric-shouldnt-be-used-to-evaluate-nearest-neighbour-classification-1fe314f460a2.

[5]: Evaluation of Classifiers: TTT4275 - Estimation, Detection, and Classification. Section 3.5 of the classification compendium.

[6]: https://www.researchgate.net/publication/3201643_Texture_fusion_and_feature_selection_applied_to_SAR_imagery.

Appendix

The scripts for both task one and two can be in the form of two Jupyter notebooks attached with the submission.