2DI70 - Nearest Neighbor Classification

Pieter Derks - 0679959 Adam Eljasiak - 0965449
Casper Siksma - 0902066 Stepan Veretennikov - 1463586

March 2020

1 Introduction

In this report we will consider the k-Nearest Neighbor learning rule (k-NN) and its implementation in an attempt to answer a set of given questions. The k-Nearest Neighbor learning rule is used to classify data points based on the labels of the points nearest to it. Mathematically the k-Nearest Neighbor learning rule is defined as equation 1.

$$\hat{f}_n(x) = \arg\max_{y \in \mathcal{Y}} \left\{ \sum_{i=1}^{k} 1\{Y_{(i)}(x) = y\} \right\} \tag{1}$$

We have created an implementation of the k-NN learning rule as an algorithm for classification in Python. Classification is performed on the MNIST dataset, consisting of handwritten digits in the form of images of 28 by 28 pixels with pixel intensities taking values in $\{0, 1, ..., 255\}$. The images are accompanied by a "true" label, in $\mathcal{Y} = \{0, 1, ..., 9\}$. The data set is partitioned in training and testing subsets and for the first couple of questions we will use a smaller sample of the training and testing subsets to limit computation time. The algorithm implementing the k-NN learning rule is then analyzed based on accuracy for different values of k using various distance metrics and the 0/1 loss function defined as equation 2.

$$\ell(\hat{y}, y) = \begin{cases} 1 & \text{if } \hat{y} \neq y \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The empirical risk is then simply given as the average number of errors we make with the classification of digits, shown in equation 3.

$$\frac{1}{m} \sum_{i=1}^{m} 1\left\{ \hat{Y}_i' \neq Y_i' \right\} \tag{3}$$

To help in determining the optimal value of k without looking at the testing data set, we also estimate the risk of the k-NN learning rule using Leave-One-Out Cross-Validation (LOOCV) for different values of k.

The report is structured such that each subsection of section 2 addresses a respective question. Finally, we will look at all our findings and comment on our choices and results in section 3.

2 Questions

2.1 Question (a)

The k-NN rule was implemented in a brute-force manner (Appendix 4.1), where the algorithm iterates over the whole training set to determine the nearest neighbors of an arbitrary new point. Specifically, a list of candidate neighbors of length k is maintained during the consequential iteration process and is updated on the occurrence of a closer neighbor. Finally, the most frequent label in the resulting list is declared as the predicted label. The distance between points is measured using the Euclidean distance defined in equation 4.

$$\sqrt{\sum_{i=1}^{l}(x_i - y_i)^2} \tag{4}$$

The algorithm is also designed to ensure that the problem of ties (in equation 1) is addressed in a stable way. Particularly, two types of ties are considered: equidistant points and conflicting votes. Concerning the ties of equidistant points, where one or more points have the same distance to a new arbitrary point x, equidistant points will be given the same order in the ordered set of distances from the arbitrary new point as the order in which they appear in the training set. This means that the list of nearest neighbors is updated only in case there exists one or more points that prove to be strictly closer than those currently stored in the list of neighbors.

Regarding the ties of conflicting votes, when several labels occur the same number of times among the k nearest neighbors while all being the most frequent at the same time, the priority for prediction is given to the neighbor which is the closest to the target point based on the distance metric.

The performance of the algorithm was evaluated using the empirical loss based on the 0/1 loss function, as defined above in equation 2, for $k \in \{1, 2, ..., 20\}$. The resulting values of the empirical loss for both the test and the training sets, as well as the accuracy values are provided in table 1. Figure 1 plots the empirical train and test loss as a function of k.

Table 1: Empirical training and test loss, accuracy for $k \in \{1, ..., 20\}$

k	Train accuracy	Train loss	Test accuracy	Test loss
1	1.00000	0.00000	0.91500	0.08500
2	1.00000	0.00000	0.91500	0.08500
3	0.97267	0.02733	0.92300	0.07700
4	0.97300	0.02700	0.92400	0.07600
5	0.95933	0.04067	0.92800	0.07200
6	0.95733	0.04267	0.92400	0.07600
7	0.94867	0.05133	0.92000	0.08000
8	0.94667	0.05333	0.91900	0.08100
9	0.93767	0.06233	0.91700	0.08300
10	0.93733	0.06267	0.91000	0.09000
11	0.92933	0.07067	0.90600	0.09400
12	0.92967	0.07033	0.90300	0.09700
13	0.92467	0.07533	0.90200	0.09800
14	0.92667	0.07333	0.89700	0.10300
15	0.91967	0.08033	0.89700	0.10300
16	0.91933	0.08067	0.89500	0.10500
17	0.91567	0.08433	0.89600	0.10400
18	0.91700	0.08300	0.88900	0.11100
19	0.91533	0.08467	0.88800	0.11200
20	0.91433	0.08567	0.88700	0.11300

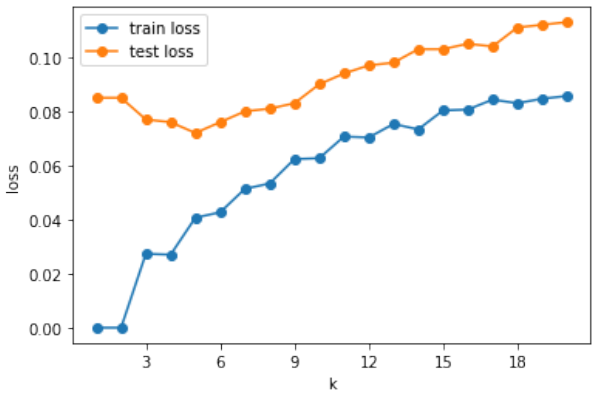

Figure 1: Empirical loss measured for the train and test sets.

2.2 Question (b)

In order to help determine the optimal value of k without looking at the testing set, cross-validation can be used. Leave-One-Out Cross-Validation (LOOCV)

was implemented and performed on the train set to determine the number k of nearest neighbors such that the empirical loss is minimized. The resulting values are provided in the table 2. Figure 2 displays the empirical loss values, plotted as a function of $k \in \{0, 1, ..., 20\}$ for both the train set during LOOCV and the test set.

The results of the cross-validation procedure suggest that the optimal value k that minimizes the empirical loss is 3 (though 4 is equally optimal, but simplicity is preferred). Despite the fact that $k = 5$ would minimize the empirical loss on the test, we do not take this into account as the choice of k should be made without looking at the test set.

Table 2: LOOCV Risk estimates (empirical loss) and accuracy for $k \in \{1, ..., 20\}$

k	LOOCV Train accuracy	LOOCV Train loss
1	0.91933	0.08067
2	0.91933	0.08067
3	0.92700	0.07300
4	0.92700	0.07300
5	0.92167	0.07833
6	0.92100	0.07900
7	0.91833	0.08167
8	0.91767	0.08233
9	0.91667	0.08333
10	0.91133	0.08867
11	0.91200	0.08800
12	0.91100	0.08900
13	0.90767	0.09233
14	0.90700	0.09300
15	0.90567	0.09433
16	0.90500	0.09500
17	0.90433	0.09567
18	0.90433	0.09567
19	0.90233	0.09767
20	0.90167	0.09833

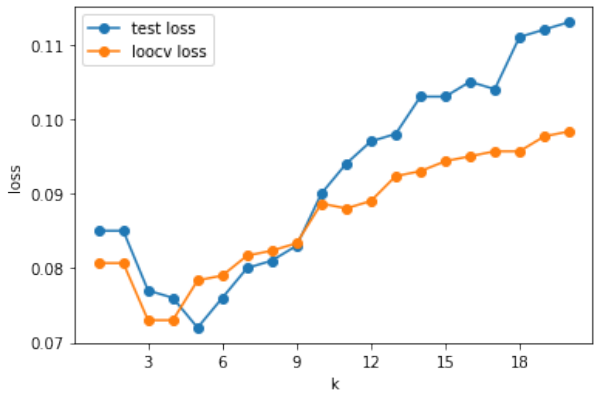

Figure 2: Empirical loss on LOOCV on the train set with respect to parameter k.

2.3 Question (c)

As the distance measure plays an important role and so far we have only used the Euclidean distance, we will also consider a generalization of the Euclidian distance, known as ℓ_p distances or Minkowski distances. This generalization is given in equation 5 and takes a parameter $p \geq 1$.

$$d_p(x, y) = \left(\sum_{i=1}^{l} |x_i - y_i|^p \right)^{1/p} \quad \text{for } x, y \in \mathbb{R} \tag{5}$$

For the purpose of simplifying computation, only integer values of $p \in \{1, 2, ..., 15\}$ were tested. For each value of p, we ran the algorithm using LOOCV as explained previously in subsection 2.3 for $k \in \{1, ..., 20\}$.

The dependence of the loss function on both parameters k and p can be visualized with the level-sets contour graph, as can be seen in the figure 3. It shows that the area of the smallest loss values is centered around small values of k, approximately between 2 and 6, while at the same time the parameter p reaches its optimal value in the area of 5 or higher. Figure 4 shows more explicitly that the optimal value of parameter k is reached independently of p, and is equal to 3. Figure 5 displays the loss function graph for fixed $k = 3$ with respect to p, so it could be seen that the loss is minimized when $p = 8$. The conclusion based on visual inspection is well supported by formal calculations, thus we can say that the optimal combination of parameters is $p = 8,\ k = 3$ with corresponding loss function value of 0.05767 with an accuracy of 94.233%.

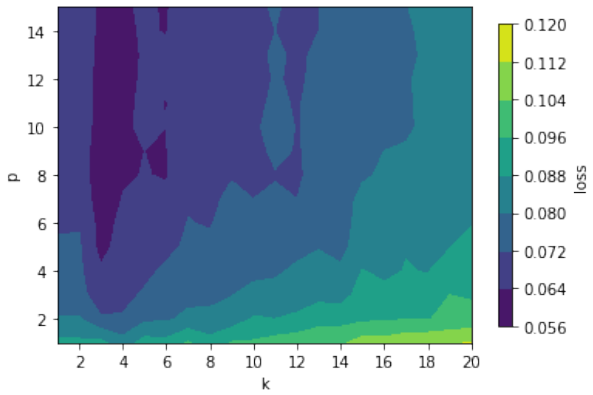

Figure 3: Empirical training loss using LOOCV with respect to parameters p, k.

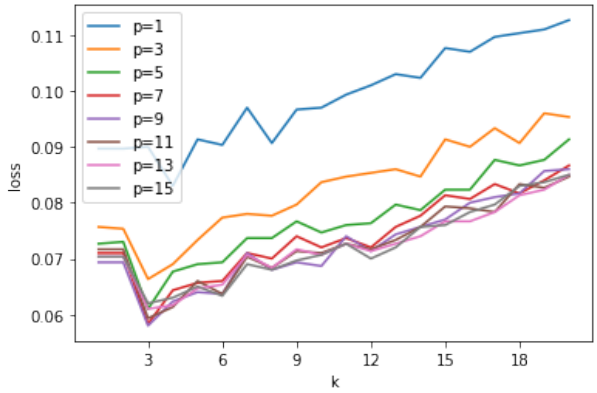

Figure 4: Empirical training loss for particular values of p with respect to k.

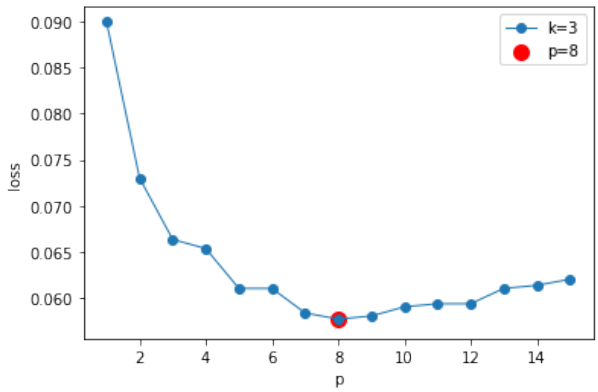

Figure 5: Empirical training loss using LOOCV with respect to p, for $k = 3$.

2.4 Question (d)

We came up with several improvement ideas which will be discussed briefly below.

2.4.1 Smart trade-off

In the previous question (c) it could be seen that the most optimal parameter is $p = 8$ for the Minkowski distance. Computing this Minkowski distance with high p however is computationally intensive. Depending on the goal at hand it might be a good idea to lose a little bit of accuracy to instead have a much faster running time. For this the Euclidean distance would be much better, which is equivalent to Minkowski distance with $p = 2$ and can be computed much faster.

2.4.2 Standardize input values

Currently the input values are pixel values with a range of 0-255. This input could be standardized to a black and white scale rather then a grey scale value. This can be done by making all values of a treshold t or higher white and all values of t or lower black. So each pixel would then either be of value 0 for black or 1 for white. Doing this transformation of the input will cause some information loss, as no distinction between various grey scale values can be made. However we primarily look at the shape to identify the numbers which will still remain, though we could get some distortion around the edges. Having only binary values could make computation significantly easier.

2.4.3 Use distance metric as weight

The idea is to take distance into account when scoring the nearest k neighbors. Intuitively the k-NN algorithm works because a point is likely close other points with the same label. Therefore, the k nearest neighbors are used. However not all these neighbors are equal, some are more distant to the point we want to classify then others therefore the distance can be used as a metric to weigh closer points more heavily than more distant points. The main question is then on how to derive a good scoring system for this. A simple way to do this is to divide each point of the k nearest points by its distance. Then sum these inverse distances for each label. For example, $k = 3$ and the distances of the points are 1, 2 and 3 respectively. This means that the first point has a weight of 1, the second point of $\frac{1}{2}$ and the third point of $\frac{1}{3}$. This in contrast to the original version where each point would have a weight of 1. Since we don't know how to exactly penalize this distance an exploration could be done. Take the function $W = (\frac{1}{D})^L$ where W is the weight, D the distance and L the parameter to be defined by exploration similar to k and p in the previous questions. A higher value of L will more heavily penalize distances then lower values of L. Having $L = 0$ is the same as the original k-NN algorithm leading to all weights to be equal to 1 and $L = 1$ would be a weight inverse to the distance as given in the example before. Interesting would probably be exploring L in the range of $[0, 5]$.

2.5 Question (e)

As the Euclidean distance (same as Minkowski distance with $p = 2$) is much quicker to calculate than $p = 8$, we will use $p = 2$ for the full training set. The brute-force approach of our implementation proved to be too inefficient to use with the full training set. Running the algorithm for all $k \in \{1, ..., 20\}$ took almost half an hour to complete for the smaller sample subsets. Considering that the full data set is an order of magnitude larger than the samples we have been using, it is not practical to use. This lead us to come up with a different implementation that uses k-d trees (Appendix 4.2). By using a k-d tree we prune a lot of the neighbor search space as a lookup becomes $O(\log_2 n)$ rather than $O(n)$, speeding up the running time tremendously. Compared to the brute force implementation, the k-d tree implementation takes only three minutes to complete for all values of k. Still, using the k-d tree implementation and half the full training set (30000 images) we were only able to test $k \in \{1, ..., 5\}$ within the available time as it took almost two hours to complete.

After running the algorithm on half the full training set we obtain the results in table 3 and figures 6 and 7. After inspection of the results we can say that the optimal value of k equals 4, with corresponding loss function value of 0.03050 and an accuracy of 96.950%.

Table 3: LOOCV Risk estimates (empirical loss) and accuracy for $k \in \{1, ..., 5\}$ using the full training set

k	LOOCV Train accuracy	LOOCV Train loss
1	0.96823	0.03177
2	0.96823	0.03177
3	0.96797	0.03203
4	0.96950	0.03050
5	0.96707	0.03293

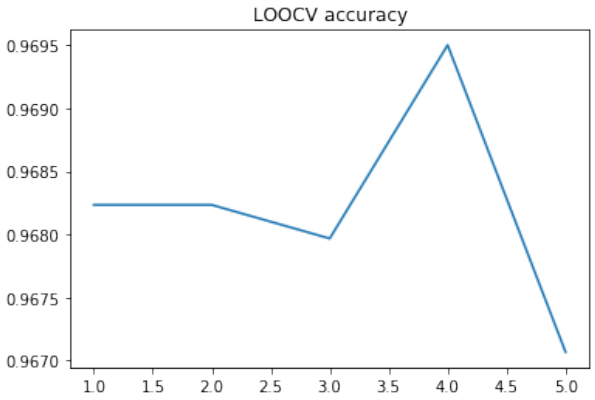

Figure 6: Empirical training accuracy using LOOCV with respect to k.

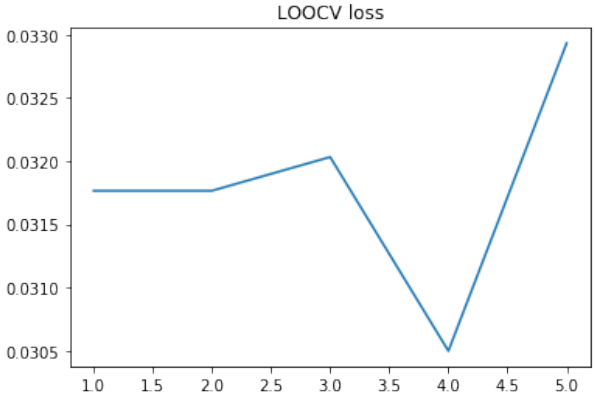

Figure 7: Empirical training loss using LOOCV with respect to k.

2.6 Question (f)

Using half of the full testing set (5000 images) we were able to compute the empirical test loss for different values of k.

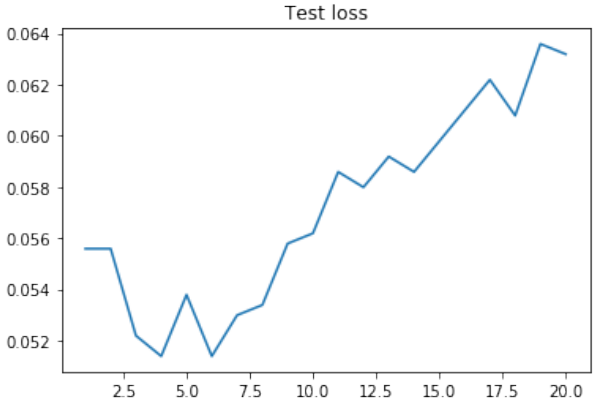

Figure 8: Empirical test loss on half the full test set with respect to k

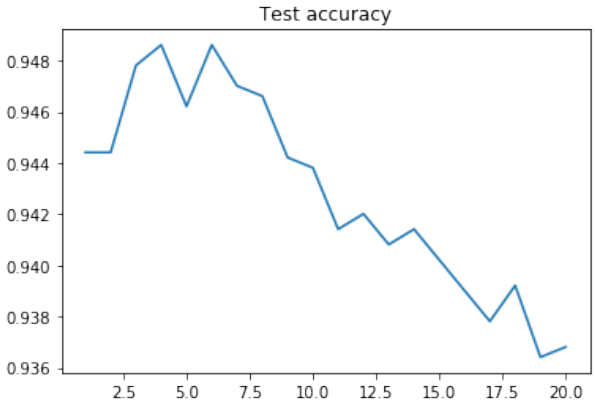

Figure 9: Empirical test accuracy on half the full test set with respect to k

From the results we have that with $k = 4$, the empirical test accuracy is given by 94.861% and the empirical loss is given by 0.05139 which is higher than the computed loss estimate. If we were able to look at the test dataset earlier we would still have chosen $k = 4$ as this is ideally only based on the training dataset. Moreover, the plot of the test loss in figure 8 shows that despite the loss being higher, $k = 4$ is still the optimal choice.

2.7 Question (g)

Only 19% of the pixel values in the MNIST dataset are non-zero, which creates an opportunity to process the training data in a way that preserves most of the information, but drastically reduces the size of the dataset. Taking as example the brute-force implementation, it takes $O(n + n * \log_2 n)$ operations to classify

an arbitrary input, where n is the size of the training set. It quickly becomes clear that processing large datasets might be unfeasible. We will examine if applying Principal Component Analysis (PCA), a method for dimensionality reduction, can be beneficial in the case of MNIST. To check the impact of PCA on MNIST, it needs to be decided which number of principal components (PC) should be preserved. The testing is done using naive k-NN classifier using the best performing set of parameters established in Question (c) and PCA implementation originating from package Scikit-Learn. The code for the experiments can be found in Appendix 4.3 The sample training set was transformed to inspect the relation between number of PCs and cumulative ratio of variance explained. The first intuition was to aim for 95% of explained variance, and figure 10 shows that this amount of explained variance ratio is achieved when using 144 or more PCs. To verify how many are actually needed, performance of the naive KNN classifier was measured on the smaller training set using LOOCV. The results are promising, because as shown in Figure 11, it is only necessary to maintain around 45 PCs to reach the peak performance for all the values of $k \in \{2, 3, 4, 5\}$ close to the optimal established in Question (c). Based on this result, the training data was transformed using 45 PCs, and this time verified on the first half of the full testing dataset - the classifier achieved 96.097%, 96.820%, 96.753% and 96.740% accuracy, respectively for consecutive values of k. We can observe that reported accuracy is almost identical to the one from Question (c). It did not decrease because the dataset is sparse, which made information loss marginal. The testing accuracy is more conservative, achieving 94.4%, 95.18%, 95.2% and 95.31% test accuracy for the same values of k as before. It seems that compressed dataset allows for some degree of statistical generalization, which provides additional robustness. Obtained results are slightly better than what were observed in Question (f). Therefore it can be concluded that applying PCA is beneficial in this scenario - in terms of speed and accuracy.

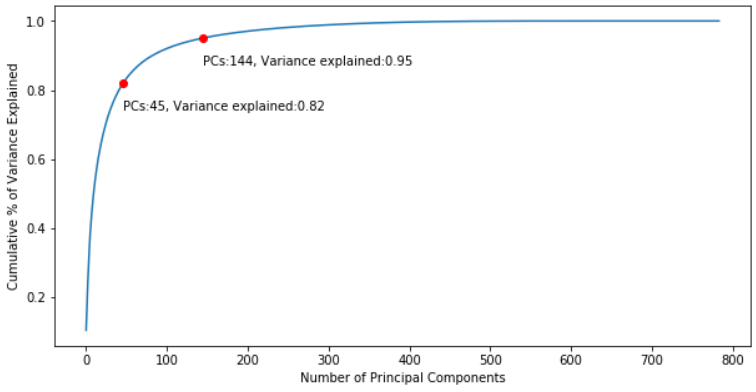

Figure 10: Cumulative percentage of variance explained by sorted principal components using sample training set.

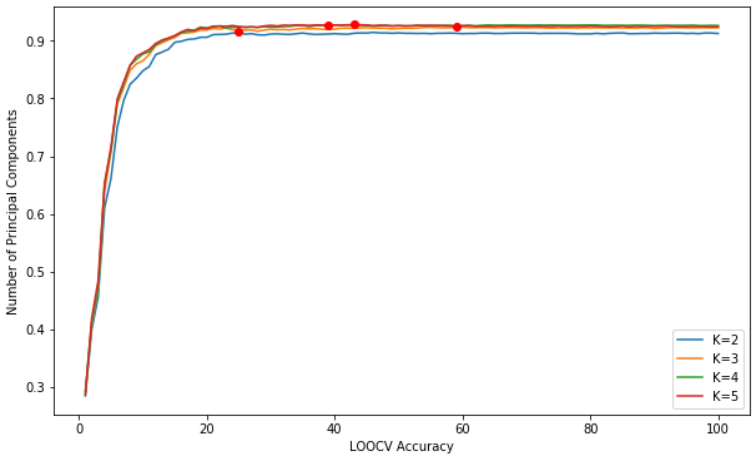

Figure 11: Accuracy of the k-NN classifier using sample testing set projected on 45 most important principal components of the sample training set for different values of k. Best accuracy for each k is marked with a red dot (91.500%, 92.367%, 92.8%, 92.767% respectively)

3 Conclusion

Overall we find that the k-Nearest Neighbor learning rule is very powerful and easy to tune with only two parameters, k and p. With accuracy of over 90% this very simple rule is able to tackle difficult problems such as classifying handwritten digits with relative ease. While the advantages are clear, there are also some downsides to the k-Nearest Neighbors that became apparent in section 2. The implementation really does matter, as the dataset grows larger, the efficiency of the algorithm becomes important. With our brute force implementation using the relatively small sampled datasets still taking almost half an hour to complete, it became unusable for larger datasets. The k-d tree implementation was able to speed up the algorithm significantly but is still quite slow and unpractical to use for datasets with large n. Dimensionality has the same effect, highly dimensional data is simply not well suited for k-Nearest Neighbors as the cost of calculating the distances between points becomes unmanageable. Finally, we found that using Principal Component Analysis (PCA) can prove to give better performance.

4 Appendices

4.1 Appendix A

For Appendix A please refer to the included '*source.pdf*' and the 'knn.py' and 'SLT2.ipynb' filename headers.

4.2 Appendix B

For Appendix B please refer to the included '*source.pdf*' and the 'knn.py' and 'SLT2.ipynb' filename headers.

4.3 Appendix C

For Appendix C please refer to the included '*source.pdf*' and the 'knn.py' and 'naive_knn.ipynb' filename headers.