ROB313 Assignment 1: Investigating the k-NN Algorithm and Linear Regression

Andrew Jairam

1006941972

andrew.jairam@mail.utoronto.ca

Introduction and Objectives

Machine learning is a complex field with many different ways to train data, and as such, an introduction to the field warrants the investigation of simpler methods to understand the workings of how machines learn from data. Two such methods are the k-NN and linear regression methods, which are the main focus of this report.

The first objective of this exercise is to investigate both the k-NN classification and regression methods by examining the effects of choosing different hyperparameters, and good practices of how to find optimal hyperparameters that improve fitting accuracy. Namely, the cross-validation method will be used to practice the importance of splitting the data into testing, training, and validation. Additionally, the k-NN algorithm will be optimized both by choosing optimal hyperparameters to improve accuracy and by experimenting with a k-d tree to improve runtime.

Secondly, the k-NN performance will be compared to linear regression by SVD to understand the advantages and drawbacks of each method, and when it is appropriate to choose a particular method.

k-NN Regression Analysis

To begin studying the k-NN algorithm, a brute force approach will be implemented to understand the functionality of k-NN regression. To choose the optimal hyperparameters *k,* the number of neighbours used for comparison, and *l*, the distance metric used for comparisons, 5-fold cross-validation was used across the training and validation sets stacked for each data set. For a given *k* and *l* value, the implemented cross-validation function splits the stacked training set into five parts, trains each of the parts on the other four parts, and returns the cross-validation RMSE averaged over the five sets. The hyperparameter values were chosen using this function by choosing the *k* value (in the range of 1 to the square root of the dataset size as per convention), and the *l* value that minimize the returned RMSE error.

The test RMSE error between predictions on the test set and the actual recorded values were calculated using the optimal hyperparameters calculated from cross-validation. The results are tabulated in Figure 1 below.

Parameter	**Mauna Loa**	**Rosenbrock**	**Pumadyn32nm**
Cross Validation RMSE	0.04141170123333	0.11665099096796	0.71308404360841
k	2	1	27
Preferred l	2	2	2
Test RMSE	0.40466508623304	0.06810446933737	0.67319636272489

Figure 1: Results of Hyperparameter Selection using 5-Fold Cross-Validation

The lower optimal k values found for the Mauna Loa and Rosenbrock datasets can be justified by the smaller size of their training sets. Interestingly, however, although the Rosenbrock training set is larger than the Mauna Loa dataset, a smaller k value is optimal. This could suggest that the dataset is more spread out, making it harder to fit a trend. Similarly, since the Pumadyn32nm dataset is much larger than the other two regression datasets, a larger k value is optimal. Thus, in general, it can be concluded that more data suggests that larger values of k are more viable. Furthermore, the euclidean distance metric was found to be superior to the manhattan distance metric for all cases, so the l_2 metric can be claimed as better. This would make physical sense as euclidean distance between two points is a better comparison to judge how “close” they are in space.

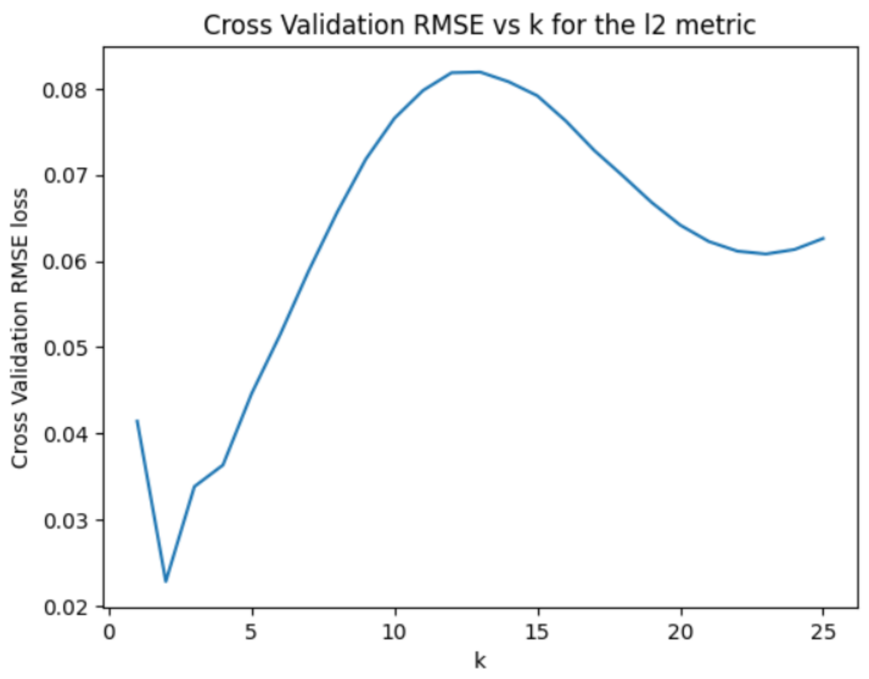

Figure 2: Cross Validation RMSE vs *k* for the Mauna Loa Dataset

To further examine the effect of the choice of hyperparameter *k,* the recorded cross-validation RMSE for the Mauna Loa dataset across *k* in the range of 1 to $\sqrt{n} \sim 25$ was plotted as shown in Figure 2 above. The RMSE value peaks at about *k* = 12 and then decays to a local minimum at around *k* = 22. This shows that although one wants to balance the effects of overfitting and underfitting in the k-NN algorithm, this effect is not quite achieved at a *k* value in the middle of the conventional region of $[1, \sqrt{n}]$. For all three datasets, the optimal *k* value was towards the lower end of this region, which is an interesting conclusion.

Regardless, the cross-validation RMSE value is still quite small, so the effect on the dataset is barely noticeable. This is made clear when plotting the cross-validation sets for different *k* on the training set as shown in Figure 3: as it can be seen, all values of *k* fit the actual training set quite nicely.

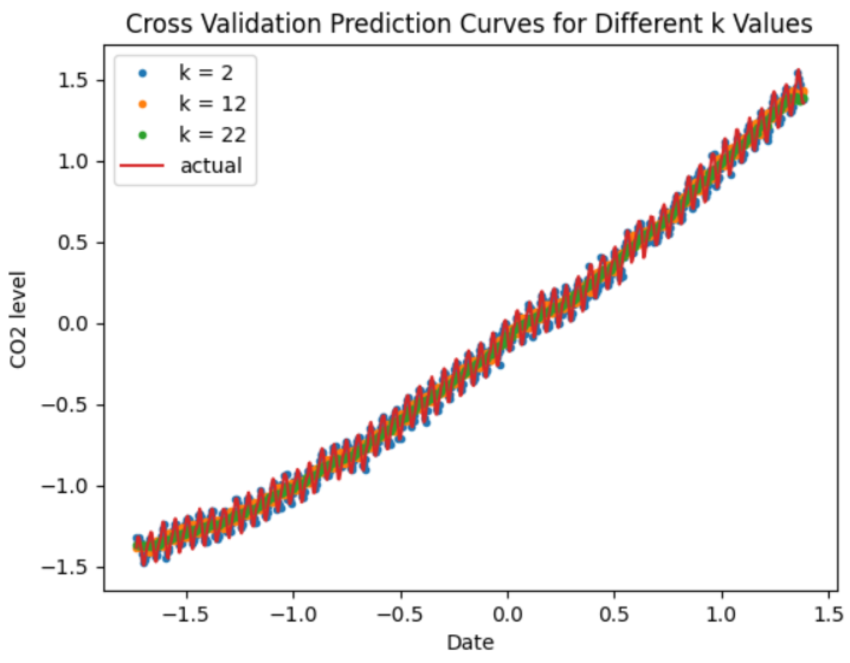

Figure 3: Cross-Validation Prediction Curves for various *k* Values using the Mauna Loa Training Set

To complete the analysis of the brute force k-NN algorithm, the prediction on the training set was plotted on the full dataset as shown in Figure 4 below. The prediction on the training set is completely off: the k-NN algorithm predicts a constant value across the test set. This exposes a key limitation of k-NN regression: predicting future data is not possible since the *k* nearest neighbours are always the same final points. Thus, it is not appropriate to use k-NN methods when trying to predict data.

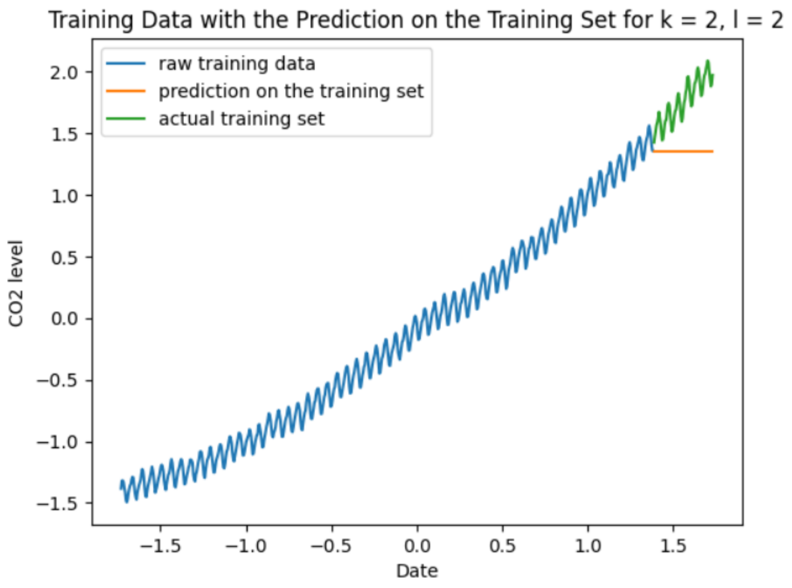

Figure 4: Prediction on the Test Set

Analysis of Time Improvement using K-D Trees

One of the main disadvantages of the previous implementation of the k-NN algorithm is its runtime complexity: checking one feature at a time in a large dataset is a highly inefficient process, especially if each feature is multivariate. As such, it is important to consider more optimized implementations of the k-NN algorithm, such as the k-d tree approach: an algorithm that checks multiple feature points at once.

To view this improvement, the times to run the brute force algorithm compared to the sklearn K-D tree implementation were plotted against the varying length of the Rosenbrock dataset feature vector, *d,* with the other hyperparameters fixed at k = 5 and l = l_2 . The resulting plot taking *d* in the range of 2 to 300 is shown in Figure 5.

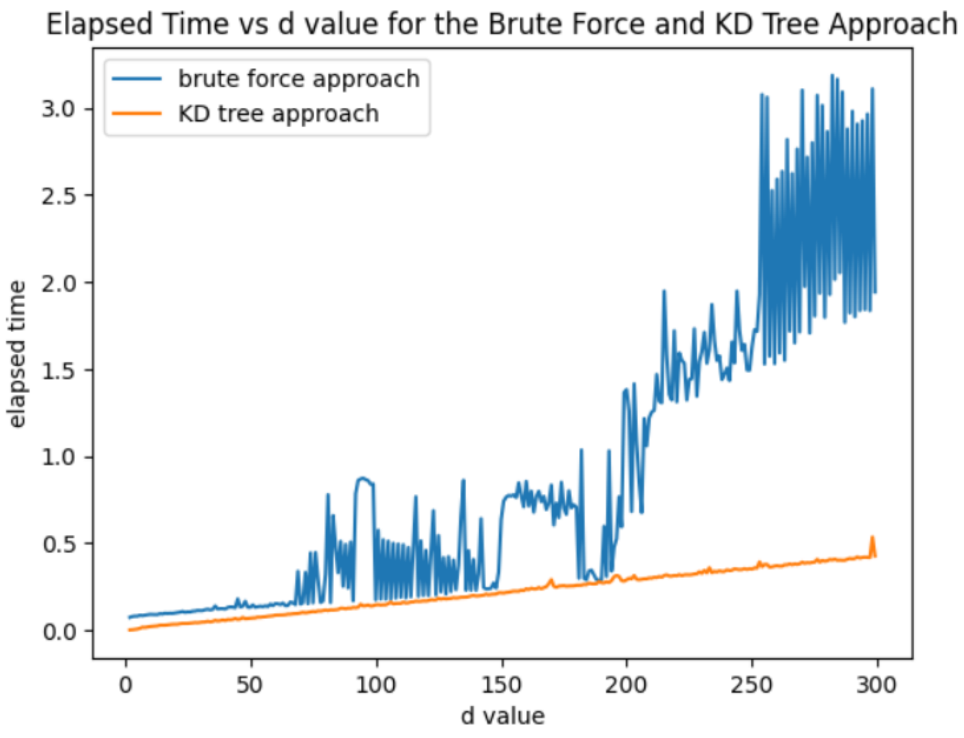

Figure 5: Resulting Timing Analysis of the Two k-NN Methods

As clearly displayed, the k-d tree outperforms the brute force method for any *d* value. The runtime of the k-d tree implementation of k-NN maintains a linear trend against increasing *d,* while the brute force approach appears to grow exponentially. Thus, it can be concluded that based on the results of this analysis, the k-d tree implementation should always be used in k-NN analysis.

k-NN Classification Analysis

The k-NN classification algorithm is perhaps the more well-known usage of k-NN, and in this section, a k-d implementation of the algorithm will be analyzed. The optimal hyperparameters *k* and *l* were determined similarly as they were found in the regression implementation, but instead of returning the smallest cross-validation RMSE, the hyperparameters yielding the greatest accuracy were returned for each dataset. To do this, the prediction vector using the validation test points returned by the algorithm was compared to the actual validation test point values, and a count was maintained for each correct matching prediction, which was divided by the total number of test points in the validation set. As before, the conventional rule of *k* in the range $[1, \sqrt{n}]$ was used, and both the euclidean l_2 and manhattan l_1 were tested. The results of this procedure are shown in Figure 6 below.

Parameter	**iris**	**mnist_small**
k	7	1
Preferred *l*	2	2
Accuracy	87.1%	95.0%

Figure 6: Results of the k-NN Classification Analysis

As before, the euclidean l_2 metric is seen to outperform the manhattan metric and can be claimed superior. However, unlike regression, larger *k* values are optimal for smaller datasets like the Iris dataset, while smaller *k* values appear to be optimal for larger datasets, like the mnist dataset. This logically seems correct: if there were infinite training data points, it would be the most accurate to take the singular training data point that closely matches the input, so a $k = 1$ approximation makes sense for larger training sets. On the other hand, if there were less training data, the algorithm would be prone to outliers, which warrants a higher *k* value. Regardless, the recorded *k* values for both datasets are still small, so a conclusion would be that in general, choosing *k* on the lower end is better.

Another comment to make is that the larger mnist dataset took 40 minutes of runtime to complete, which uncovers another disadvantage of the algorithm. This conclusion is significant only when comparing the k-NN method to the linear regression method, which was much faster to complete.

Linear Regression Analysis

Finally, the linear regression method was analyzed as another method to guess data. The singular value decomposition implementation was used as it is quite efficient to execute in Python.

To test performance, the test set RMSE for regression datasets and the accuracy of predictions on the test set measured of the classification sets using the linear regression method were found and tabulated in Figure 7 below. The metrics were calculated similarly to before: for RMSE, the RMSE formula was applied to every output prediction and its respective expected output, and for accuracy, the number of predictions correct in comparison to the expected output was divided by the total number of data points. These calculations were done on the test set: the training set and validation sets were combined for this analysis.

Parameter	**Mauna Loa**	**Rosenbrock**	**Pumadyn32**	**Iris**	**mnist_small**
RMSE Value	0.34938831	0.98408720	0.86224124	N/A	N/A
Accuracy	N/A	N/A	N/A	86.7%	85.8%

Figure 7: Results of the Linear Regression Implementation on all Datasets

For all datasets, linear regression is much less accurate than the k-NN algorithm, demonstrating more test RMSE for regression sets and less accuracy for classification datasets. This does agree with theory: linear regression is sensitive to underfitting and outliers, so for the more distributed datasets, it is expected to perform worse. Its major advantage is its speed due to its simplicity: all kernels ran in under two minutes in comparison to the k-NN methods which took times in the range of 15 to 40 minutes for the larger datasets. Thus, the conclusion is that linear regression sacrifices accuracy for faster runtime, which can be important in certain applications.

Conclusion

The outcome of the study of the k-NN and linear regression methods in this report has outlined a few key findings for each algorithm. For the k-NN algorithm, it was concluded that the euclidean l_2 distance metric is superior to the manhattan l_1 distance metric to yield a more optimal fit. For regression, it was found that a large *k* for large datasets and a small *k* for small datasets is optimal, and vice versa for classification: small *k* for large datasets and large *k* for small datasets is optimal. However, this conclusion should be taken loosely because there are likely other factors causing this to happen, and instead, the choice of *k* should be determined by cross-validation. Furthermore, the k-d tree implementation is vastly superior to the brute force method in terms of runtime costs.

When comparing k-NN to regression, it was found that regression is simpler to implement and faster to run at the cost of less accuracy in comparison to k-NN methods. Additionally, it was found that the k-NN algorithm cannot handle predicting future data outcomes, which linear regression can handle. As such, linear regression is best used for time-sensitive applications and fitting data trends to estimate future outcomes, while k-NN is better when more accuracy is needed, and when the output of the algorithm is being 'sorted' into data that is already known. This makes k-NN much better suited for classification.