Samuel Looper 2019-02-01
1003044181

ROB313 Assignment #1

1. Assignment Objectives

The purpose of this assignment is to evaluate the effectiveness of various permutations of a k Nearest Neighbors (kNN) algorithm for regression and classification. A modular kNN was developed to estimate a system output (y), from a system state (x) and a set of training data with both the state and the output. This algorithm could be configured for various values of k and various distance metrics. The algorithm was tested against 5 separate given data sets to test the different permutations of the algorithm prescribed by the assignment. First, a 5-fold cross-validation methodology was used to train the kNN model to find optimal values of k and optimal distance metrics for the different regression training sets. Next the same methodology was used for classification. Next, several variations of the kNN algorithm, including brute force, vectorization, and the k-d tree were compared in efficiency and performance. Finally, the kNN algorithm was compared to a linear regression on efficiency and performance.

2. Algorithm Overview

With the aforementioned objectives in mind, a modular kNN algorithm was developed with enough configurable inputs for the demands of the assignment. Distance and RMSE functions were created, as well as helper functions such as get_neighbors(), which returned the values considered in the kNN estimation. The final kNN function had a an x & y training set input, a testing set input, and inputs for k and the distance metric. This formed the cornerstone for the rest of the assignment. For the first part a 5-fold cross-validation function was created, which compared RMSE values for the kNN estimation to find an optimal k value from 1 to 10 and an optimal distance metric from the 3 given (L1, L2, and L_Infiniti).

3. Results

3.1 k-NN Regression

Firstly, the base kNN algorithm was applied to regression, with the objective of finding an optimal k value and optimal distance metric. Each regression set had the RMSE value at each k and distance metric recorded and plotted to find a minima, and the final results recorded

Mauna Loa Data Set

Optimal K Value:	2
Optimal Distance Metric:	L1
Cross-Validation RMSE:	0.034846946
Testing RMSE:	0.0257505016

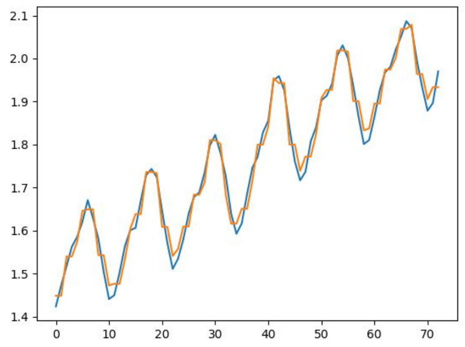

Fig 1: Result of the Mauna Loa kNN estimation with optimal configuration on training set

Rosenbrock Data Set

Optimal K Value:	2
Optimal Distance Metric:	L1
Cross-Validation RMSE:	0.2146630871
Testing RMSE:	0.2543624689

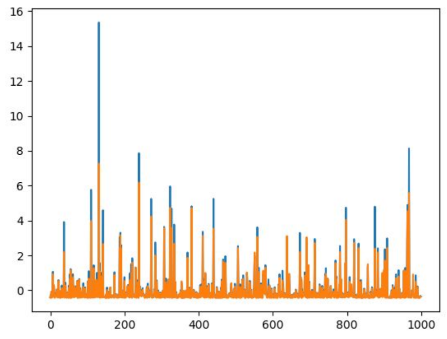

Fig 2: Result of the Rosenbrock kNN estimation with optimal configuration on testing set

Puma560 Data Set

Optimal K Value:	9
Optimal Distance Metric:	L1
Cross-Validation RMSE:	0.8609740118
Testing RMSE:	0.8597578525

Note: The optimal k-value 9 represents the highest tested k value due to time constraints

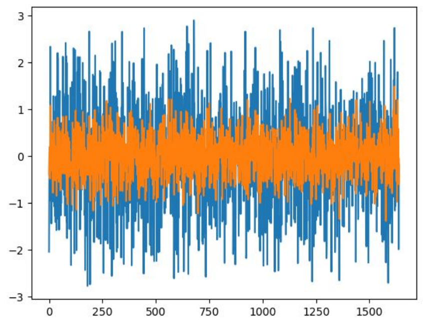

Fig 3: Result of the Puma560 kNN estimation with optimal configuration on testing set

Mauna Loa K Value Analysis

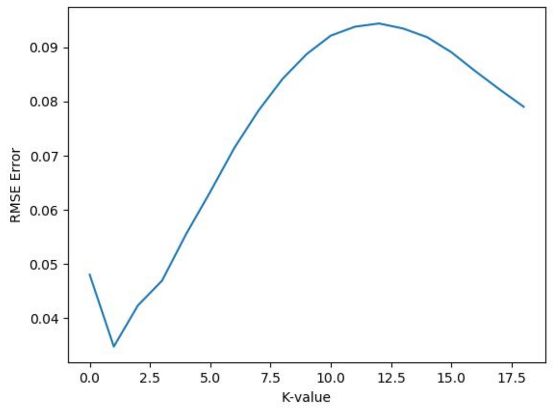

Fig 4: Plot of averaged RMSE Error in Cross Validation with respect to K-value

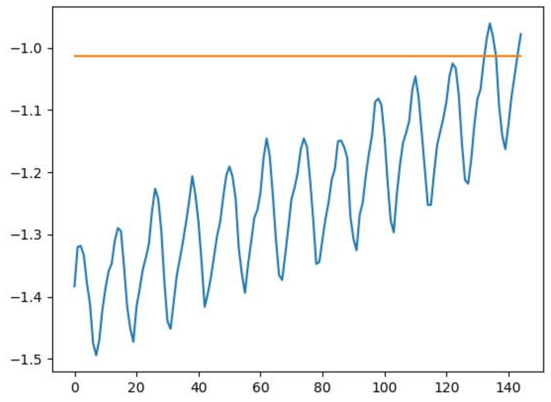

Fig 5: Result of optimal k value on testing set

The Mauna Loa data set created poor results regardless of the kNN configuration. This was because the training, validation, and testing data sets took on values in completely distinct intervals. In figure 5, you can see that the validation outputs have most values below -1, as low as -1.5, while the training and testing sets only had values higher than -1.1. As a result, the best result for the kNN would be -1.1 for all values below -1.1, resulting in a poor fit.

3.2 k-NN Classification

	Iris Data Set	MNIST Data Set
Optimal K Value:	5	4
Optimal Distance Metric:	L1	L2
Cross-Validation Accuracy:	0.807692307	0.944614
Testing Accuracy:	1.0	0.38

Fig 6: Table summarizing results from the Classification portion of the assignment

Next, the kNN algorithm was reconfigured for classification. In this case the k closest neighbors' class was polled, and the most common recurring class was assigned to the testing point. Just like in part 1, a 5 fold cross-validation with 3 distance metrics and 10 k values was tested for an optimal configuration. The error in both the cross-validation and testing processes is noted in the table above

3.3 Modified k-NN

Next, the kNN algorithm was tested for performance in terms of root-mean squared error and time complexity in different variations. Firstly, the brute force kNN algorithm, which has been used thus far in our analysis, was tested, then a partially vectorized algorithm, a fully vectorized algorithm, and finally one which was both fully vectorized and utilized the KD-Tree data structure. The tests were performed using the L2 distance matrix and a constant k=5. Below are plots of the error and runtime for each variation with respect to the dimensionality of the Rosenbrock set (with n=5000) used to test the algorithms.

3.3.1 Brute Force

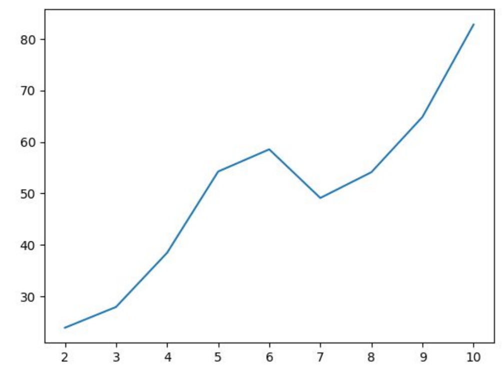

Fig 7: Brute Force runtime (seconds, y-axis) with respect to dimensionality (x-axis)

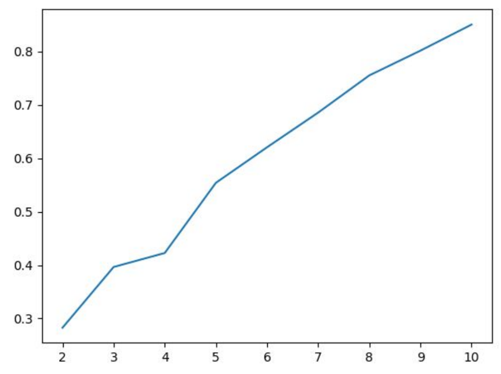

Fig 8: Brute Force RMSE (y-axis), with respect to dimensionality (x-axis)

3.3.2 Partial Vectorization

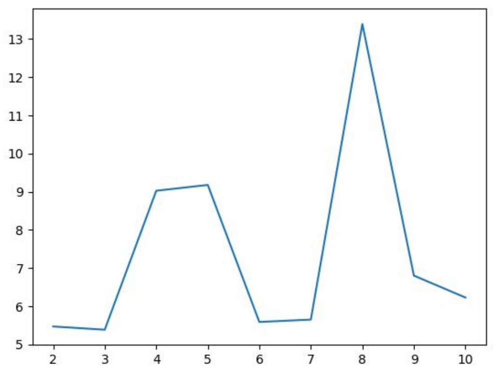

Fig 9: Partial Vectorized runtime (seconds, y-axis) with respect to dimensionality (x-axis)

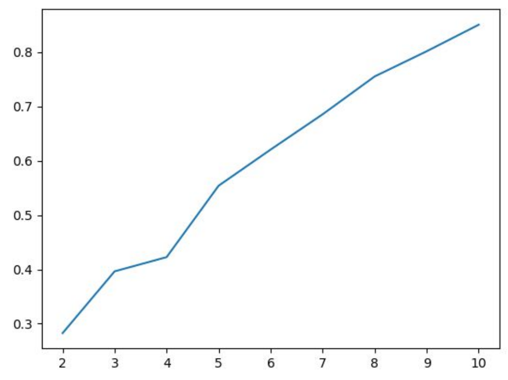

Fig 10: Partial Vectorized RMSE (y-axis), with respect to dimensionality (x-axis)

3.3.3 Full Vectorization

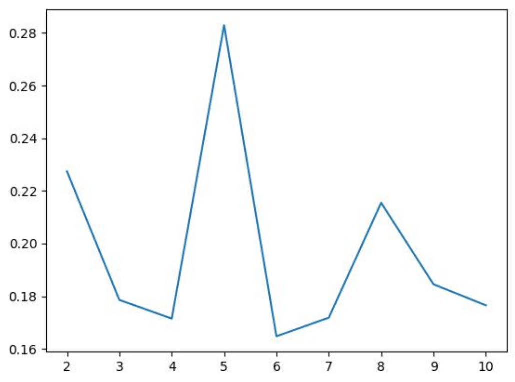

Fig 11: Full Vectorized runtime (seconds, y-axis) with respect to dimensionality (x-axis)

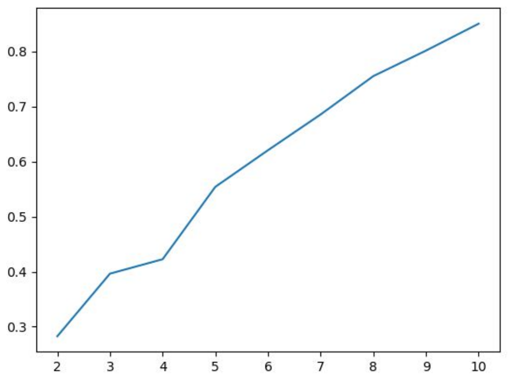

Fig 12: Full Vectorized RMSE (y-axis), with respect to dimensionality (x-axis)

3.3.4 KD-Tree & Vectorization

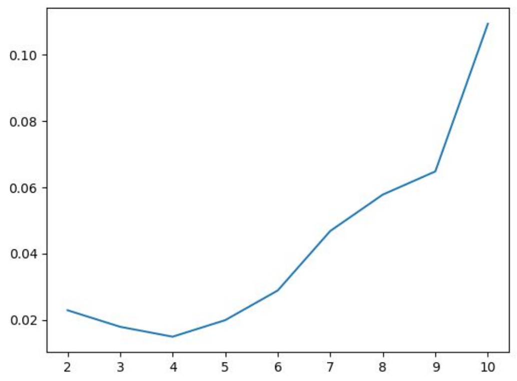

Fig 13: KD Tree runtime (seconds, y-axis) with respect to dimensionality (x-axis)

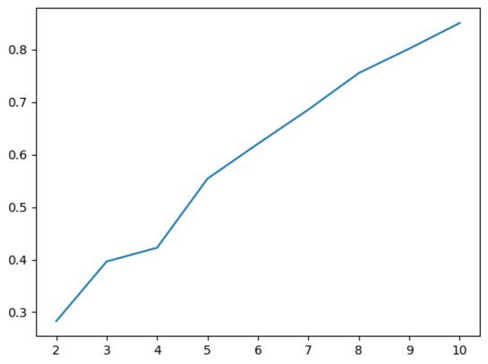

Fig 14: KD Tree RMSE (y-axis) with respect to dimensionality (x-axis)

In conclusion, the type of algorithm used does not seem to impact the RMSE at any dimensionality, as the error curves are exactly the same. On the other hand, the RMSE is affected by the dimensionality, as it increases approximately linearly with respect to the number of dimensions in the data set.

The type of algorithm does have a major impact on the runtime. The brute force algorithm has a long runtime that increases approximately quadratically with respect to dimensionality.

Partial Vectorization reduces the runtime on average by an order of magnitude, and Full Vectorization by another order order of magnitude. Both of these algorithms are affected rather sporadically with respect to dimensionality, as the runtime peaks at specific d values, with no discernable patterns. The KD Tree reduces the algorithm by yet another order of magnitude, but seems to also increase quadratically or exponentially with respect to dimensionality.

3.4 Linear Regression

In this section, singular value decomposition was used to train a linear model by finding the optimal weights in the linear regression. The algorithm is compared to kNN with respect to its RMSE for regression sets and success rate for classification. The statistics are summarized in the table below.

Data Set	**kNN Metric (RMSE or Success Rate)**	**Linear Model Metric (RMSE or Success Rate)**
Mauna Loa	0.0257505016	0.34938831049
Rosenbrock (n=5000, d=2)	0.2543624689	0.98408720306
Puma560	0.8597578525	0.86225124365
Iris	1.0000000000	0.66666666666
MNIST	0.3800000000	0.7920000000

Fig 15: Table summarizing the success metrics for each data set for kNN vs regression

In conclusion, it is clear that the the kNN algorithm is far superior in terms of accuracy with respect to RMSE for regression data sets, and one of the classification sets with respect to success rate. On the other hand, linear regression was far more effective in for the MNIST data set. It appears that kNN's relative advantage is prevalent for low dimension data sets (as opposed to the Puma560 and MNIST data sets)