ROB313 Assignment 1

Objective

The objective of this assignment is to familiarize ourselves with the first learning algorithm taught in lecture – k-Nearest Neighbors. This algorithm is commonly used in information retrieval applications and can be highly optimized for any purpose using matrix operations and other data structures. In this assignment, we also are introduced to using numpy to perform matrix operations and using data structures such as k-d trees to store information and retrieve it quickly.

Code Structure and Strategy

The strategy for the code was to create 3 files, one for the simple k-NN code for regression and classification sets, main_knn.py, that can be run to solve Q1 and Q2, a file for the process of vectorizing k-NN, vectorized_knn.py (Q3), and a file for the implementation of SVD for regression and classification, svd.py, that solves Q4. This allows each part of the assignment to be run independently. There is a small amount of duplicated code between the regression function for the test data in main_knn.py and the function that executes the regression on test data in the double loop (part a) of vectorized_knn.py. More info on running the code can be found in the README.md file.

In the main_knn.py file, we see functions for five-fold regression, and one-fold classification. These functions were not yet vectorized, so the order of the loops in both functions are such that the distances between a validation point and all training points is only computed once per fold, then the k closest values are extracted directly. This improved the runtime of the code extensively from the initial method, which computed the distances for the validation point $k\ x\ num\ fold$ times.

Discussion

Q1: kNN Regression

Table 1, regression dataset results

Dataset	Estimated k	Preferred distance metric	Cross validation RMSE	Test validation RMSE
Mauna Loa	2	L2	0.031804	0.440705
Rosenbrock	2	L2	0.330742	0.247598
Pumadyn32nm	20	L-∞	0.873513	0.830699

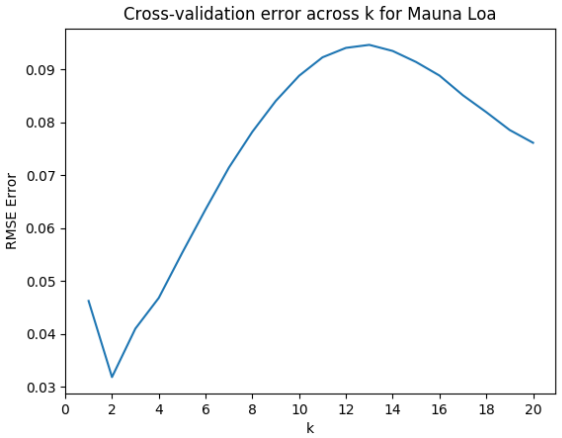

Figure 1, Mauna Loa dataset RMSE error with L-1/L-2/L-∞ distance functions

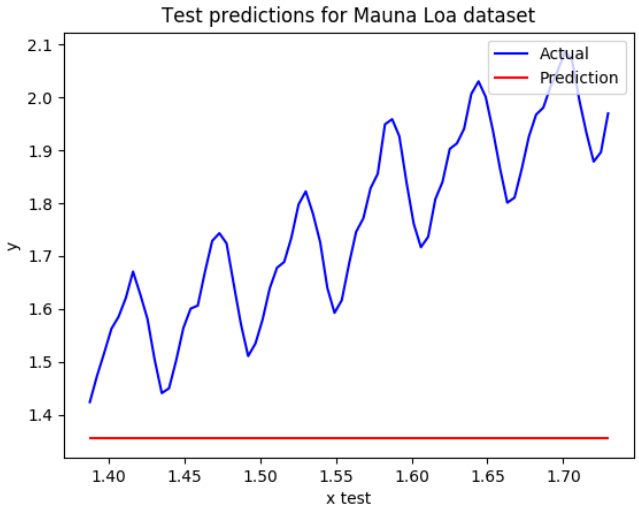

Figure 2, Test prediction for Mauna Loa dataset

It is worth noting that the Mauna Loa dataset is time dependant. Since it is the data of the global amounts of greenhouse gas, the data that is used to create and train the k-NN model will be lower than test data since the greenhouse gas content in our atmosphere is continuously increasing. The output of the k-NN algorithm is, as expected, a straight line. Many other models (including linear regression – see Q4 results) would be better suited to predict the trends of time sensitive data.

Q2 Classification

For multiclass classification problems, we continue to look for the most optimal combination of k and a distance function, however now we are looking for the highest percentage accuracy (rather than the lowest RMSE). We must also select a policy for the selection of k, since sometimes there can be ambiguity in which prediction value is the closest to our training point – for example, in binary classification with an even k value, if there is an equal number of occurrences for both of the classes in the k closest points that will determine the prediction.

To solve this problem, we simply implemented the following policy: the algorithm will select the first of the training points that it sees out of the prediction points with the same number of occurrences.

Table 2, Classification dataset results

Dataset	Estimated k	Preferred distance metric	Validation accuracy	Test accuracy
Iris	15	L1	90.32%	100%
Mnist Small	1	L2	95.00%	95.90%

Q3 Code Vectorization

The 4 components of fully vectorizing the code were as follows (labelled as such on the following result plots):

Part a: brute-force, nested loop to measure the distance between all points once
Part b: partial vectorization, an outer loop with some vector/matrix operations using numpy.
Part c: full vectorization, no loops in the code, including in the RMSE function.
Part d: full vectorization using a k-d tree data structure with the sklearn Python library.

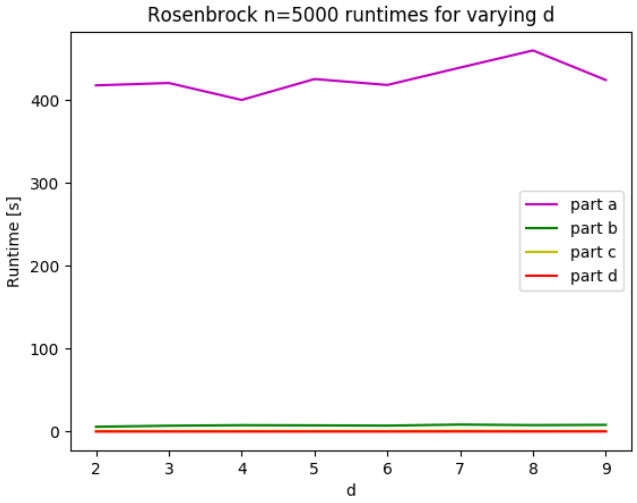

Figure 3, Runtimes for vectorization parts a, b, c, and d

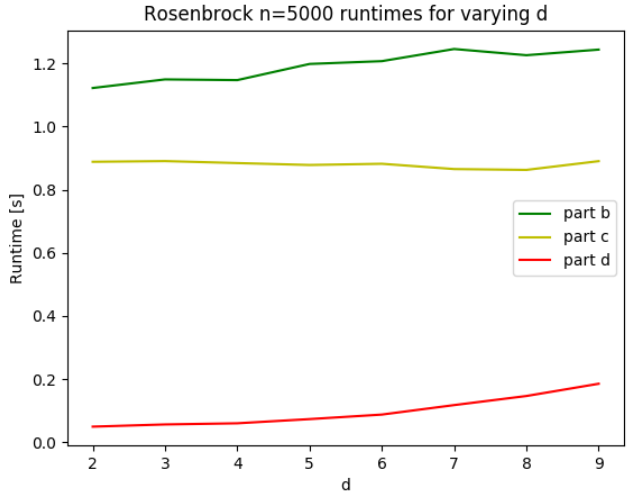

Figure 4, Zoom in on the runtimes of parts b, c, and d for vectorization

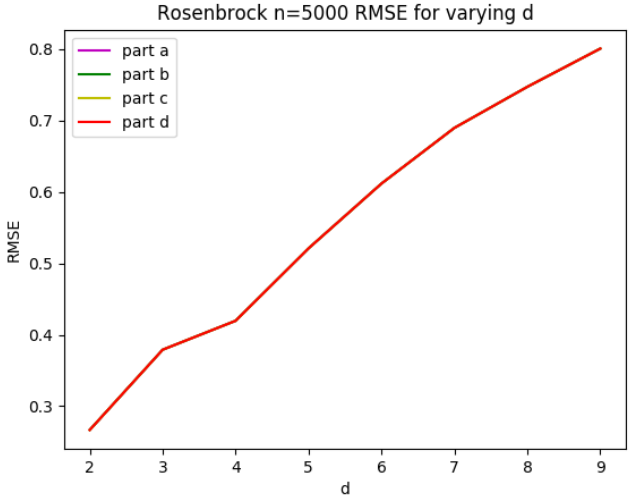

Figure 5, RMSE errors for the 4 incremental methods of vectorization

The runtimes decrease by at least an order of magnitude for each of the methods of vectorizing the code. For part a, we see runtimes averaging around 420s, for part b we see around 8s, part c 0.8s, and 0.08s for part d. These results show that performing matrix operations in Python can be much faster while still providing the same results and data accuracy.

The k-d tree data structure is used as an efficient way to represent the k-NN data as when we store the neighbor data in tree nodes, we are able to search for a specific nodes neighbours in O(logn) time, only comparing the node in the tree with a few other nodes and pruning the rest based on proximity conditions of groups of points.

Q4 Linear Regression with SVD

Table 3, SVD results for regression and classification sets

Dataset	Validation RMSE/Accuracy	Runtime [s]
Mauna Loa	0.3494	0.0308
Rosenbrock (n=1000)	0.9841	0.3096
Pumadyn32nm	0.8623	6.651
Iris	86.66%	0.0014
Minst Small	85.70%	40.54

Note: for Table 3, in the Validation RMSE/Accuracy column, the first 3 sets are regression, and show RMSE values, and the following 2 datasets are classification, thus showing Accuracy values there.

As discussed in the Q1 discussion, a linear regression model could be very beneficial for applications such as predicting time dependant data. Figure 6 shows the application of the SVD linear regression algorithm on the Mauna Loa dataset, resulting in a lower RMSE value than the k-NN regression.

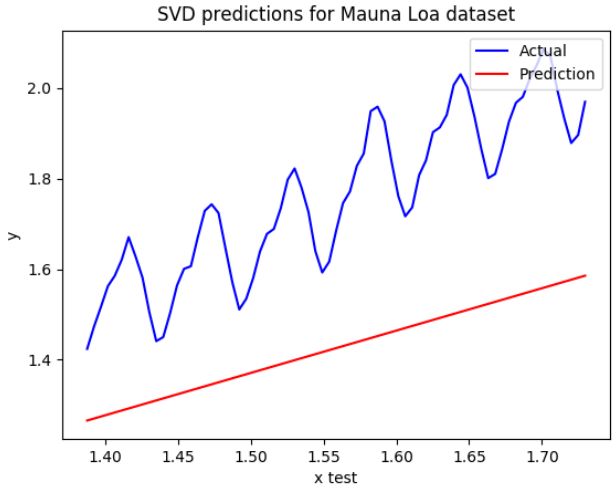

Figure 6, Mauna Loa linear regression predictions

The SVD linear regression performs quite well on classification problems as well, however not as well as the k-NN algorithm. This is because the SVD generates probabilities that a test point is part of a certain class, so if there is ambiguity, it may not choose the correct one. We do not get to determine the policy for the one-of-k binary classification algorithm (SVD for classification).