Samuel Atkins
February 12, 2019
1002951754

# ROB313 Assignment 1

## Assignment Objectives:

### Learning Objectives:

1. Understand the differences in performance for the various implementations of the k-NN algorithm.
2. Understand the performance difference between k-NN and SVD and the importance of algorithm selection.
3. Understand how to manually implement the k-NN algorithm and the SVD algorithm for classification and regression.

### Functional Objectives:

1. Using the $L_1$, $L_2$, and $L_{INF}$ distance metrics, implement the k-NN algorithm for regression and classification using:
    a. A double for-loop.
    b. A partially vectorized approach to eliminate the need for two for-loops.
    c. A completely vectorized approach with no loops.
    d. A k-d tree structure to compute the nearest neighbours.
2. Minimize the least squares loss function using SVD.
3. Record all relevant plots, figures, and data in order to understand the significance of each implementation.

## Code Structure Explanation:

### Helper Functions:

All of the implemented functions required for this assignment were implemented in the same file. At the top of the file are the following helper functions:

- RMSEs(measurements, actuals)
    - This function takes in a (1, n) list of measurements and a (1, n) list of actuals. This function computes the root mean squared error between the measurements and actuals.
- numberCorrect(measurements, actuals)
    - This function is used for classification. It computes the number of measurements that match their corresponding actual values.
- l1(a, b), l2(a, b), lInf(a, b)
    - These functions compute the l1, l2, and lInf distance metrics, respectively.

- ❖ generateFolds(data)
    - This function takes in data and returns 5 data folds. Each fold contains a validation set that is one fifth of the original data set and a training set which is the other four fifths of the data set.

### Question Implementations:

Immediately following the helper functions are the required algorithm implementations for each question. The file is clearly segmented into sections for each question. Plotting and figure generation occurs near the end of the file, just before the main function.

To plot the required plots for question 1, along with the RMSE errors for each optimal regression dataset setting, call the function **question1()**.

The first part of the **question2()** function prints the test accuracy of the k-NN algorithm for the optimal settings of the iris and mnist_small datasets. Afterwards, the percentage of correct classifications for both the iris and mnist_small datasets for k values ranging from 1 to 35 will be printed.

The function **question3()** will produce the time that it takes to run each of the k-NN variations for a range of d values. This function will also produce a plot of the time that it took to run each k-NN variation as a function of d.

The **question4()** function will print the RMSE of the first three datasets returned by the function that minimizes the least squares using SVD. It will then print the test accuracy of this algorithm for the final two datasets. The question4() function also prints the time that it takes to run this algorithm on the rosenbrock dataset for comparison with the other k-NN implementations.

## *Question 1:* k-NN Algorithm for Regression

The k-NN algorithm for regression was implemented in question 1. Using 5-fold cross-validation, the optimal k values and optimal distance metrics for each dataset were found. These results, along with the cross-validation RMSE and test RMSE are displayed in Table 1 below:

*Table 1: k-NN Regression Results (Question 1)*

| Dataset: | Optimal K Value: | Optimal Distance Metric: | Cross-Validation RMSE: | Test RMSE: |
|---|---|---|---|---|
| mauna_loa | 2 | $L_1$ | 0.0372 | 0.441 |
| rosenbrock | 1 | $L_{INF}$ | 0.245 | 0.288 |
| pumadyn32nm | 16 | $L_1$ | 1.05 | 0.834 |

Notice that in the case of mauna_loa the test RMSE increases by a factor of 10. This dramatic increase in error is a sign that our algorithm poorly predicted the output of the testing data. The other datasets had test RMSEs that were on par or lower than their respective cross-validation RMSEs.

To further analyze the relationship between testing and training data, we consider only the first dataset, mauna_loa. Figure 1 below illustrates the cross-validation loss as a function of k:

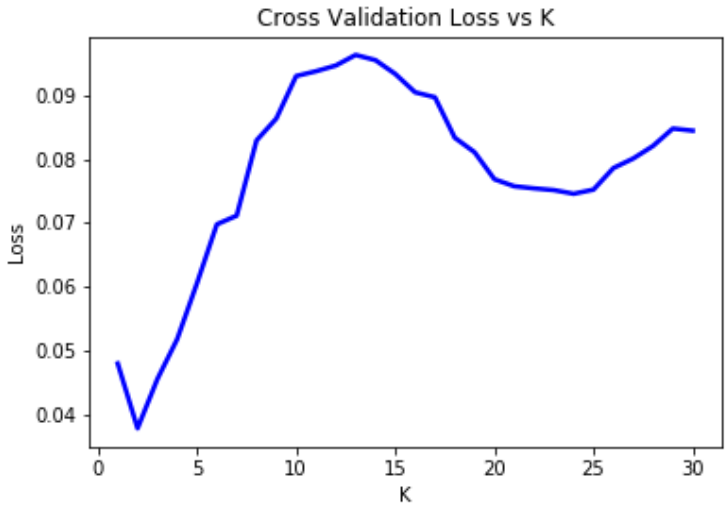


*Figure 1: Cross Validation Loss vs K (Question 1)*

As delineated by the above figure, the optimal k value for the mauna_loa dataset is k = 2. This is consistent with the results detailed in Table 1. Figure 2 below describes the cross-validation prediction curve for mauna_loa at several different k values. As one can see from the figure, a linear trend closely following the diagonal is evident:

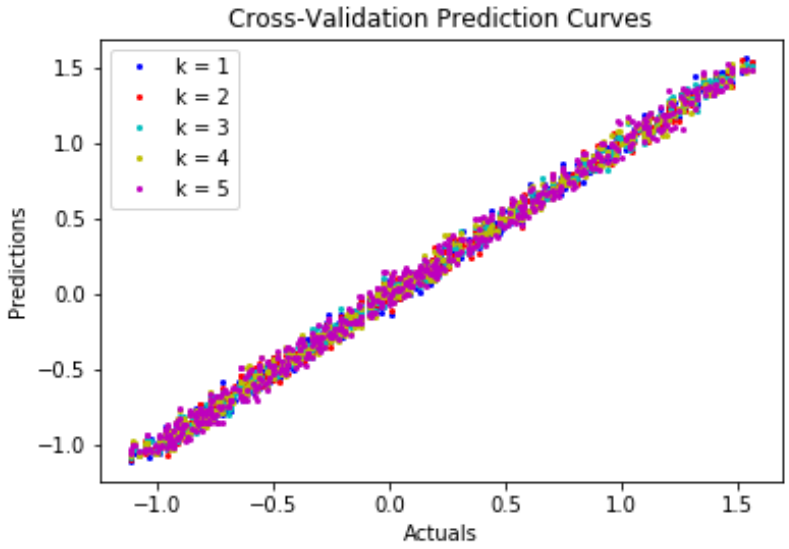


*Figure 2: Predictions vs Actuals for mauna_loa Cross-Validation (Question 1)*

A linear trend is desired because it confirms that during cross-validation our k-NN algorithm is accurately predicting the testing set. Points that occur on the diagonal are points that accurately predict their actuals. Figure 3 below is a plot of the predicted test values as a function of the actual values:

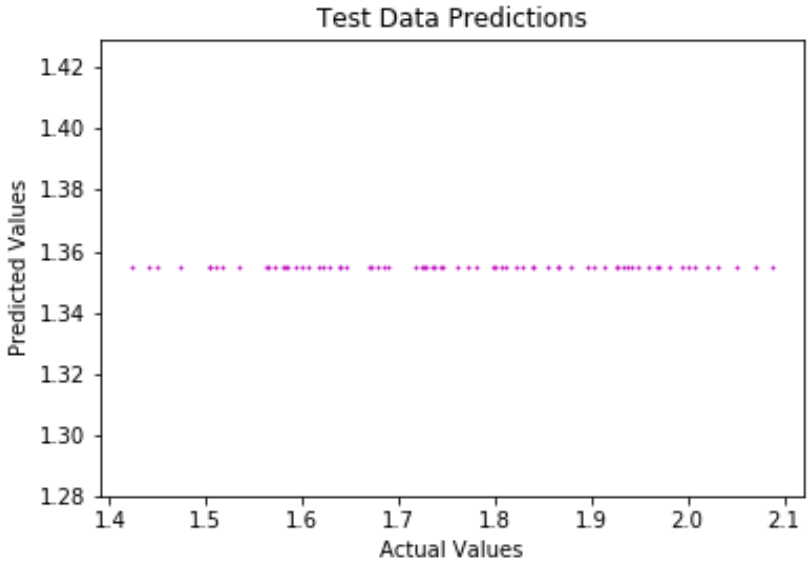


*Figure 3: Prediction vs Actuals for mauna_loa Testing Data (Question 1)*

The points in this figure tell us that the k-NN algorithm did not fare well. The predicted values are horizontal and do not follow the desired diagonal. Figures 4 and 5 below illustrate the smallest x_test inputs and the largest x_train inputs, respectively, for the mauna_loa dataset:

```
In [63]: x_test[0][:10]
Out[63]:
[array([1.38754265]),
 array([1.3922755]),
 array([1.39706538]),
 array([1.40179823]),
 array([1.40653108]),
 array([1.41132095]),
 array([1.4160538]),
 array([1.42078665]),
 array([1.42557653]),
 array([1.43030938])]
```

*Figure 4: Smallest x_test Values for moana_loa*

```
In [38]: x_train[0][:10]
Out[38]:
[array([1.3828098]),
 array([1.37801993]),
 array([1.37328708]),
 array([1.36855423]),
 array([1.36376435]),
 array([1.3590315]),
 array([1.35429865]),
 array([1.34950878]),
 array([1.34477593]),
 array([1.34004307])]
```

*Figure 5: Largest x_train Values for moana_loa*

Figure 3 boasts a horizontal line because the test data inputs in the mauna_loa dataset are all closest to the same point. The lowest test data input is 1.388 and the largest training data input is 1.383, as illustrated in Figures 4 and 5 above. This means that all of the test data inputs will be closest to the same k points which is why the predicted values are all the same.

## Question 2: k-NN Algorithm for Classification

The k-NN algorithm for classification was implemented in question 2. Using 5-fold cross-validation the optimal k values and the preferred distance metrics for these datasets were found. These results are shown in Table 2 below:

*Table 2: k-NN Algorithm for Classification Results*

| Datasets: | Optimal K Value: | Preferred Distance Metric: | Cross-Validation Accuracy: | Test Accuracy: |
|---|---|---|---|---|
| iris | 7 | $L_1$ | 100% | 100% |
| mnist_small | 3 | $L_2$ | 92.1% | 93.4% |

The k-NN algorithm was very successful on both the iris dataset and the mnist_small dataset. With respect to the iris dataset, the algorithm achieved perfect cross-validation and test accuracy. With respect to the mnist_small dataset, a strong cross-validation accuracy of 92.1% and a test accuracy of 93.4% was reported. These values confirm the effectiveness of the k-NN algorithm for classification.

## Question 3: Various Implementations of k-NN

The following k-NN algorithm types were implemented in question 3:

- A double for-loop implementation.
- A single for-loop implementation using broadcasting.
- A no-loop implementation taking advantage of vector operations and broadcasting.
- A k-d tree implementation.

These implementations were tested using the rosenbrock dataset. The run-time of each algorithm was recorded as the d value of the rosenbrock dataset ranged from d = 2 to d = 10. The performance results of the implementations listed above are illustrated in Figure 7 below:

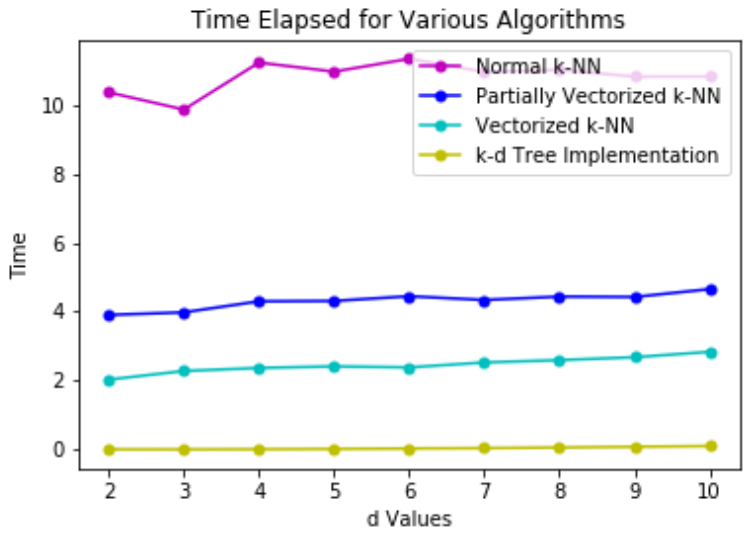


*Figure 7: Run-time of k-NN Implementations for Various d Values*

From this figure it is apparent that the double for-loop approach takes significantly longer for all d values. Furthermore, it is evident that the more vectorized the approach becomes, the faster the implementation becomes. This is because the array operations that the vectorized algorithms take advantage of run much faster. These array operations have been internally optimized and operate in C as opposed to Python, which dramatically reduces the iteration run-times. From Figure 7 it is also clear that the time elapsed of all of the implementations slowly increases with d. This makes sense because as the dimension of the dataset increases, the time that it takes to regress the data should also increase.

## Question 4: Minimizing Least-Squares Loss Function Using SVD

Using SVD, another method for regression and classification was implemented. This method was applied to all 5 datasets. The results of this algorithm are included in Tables 3 and 4 below:

*Table 3: Results of Least-Squares Loss for Regression*

| Datasets: | Test RMSE: |
|---|---|
| mauna_loa | 0.349 |
| rosenbrock | 0.983 |
| pumadyn32nm | 0.862 |

*Table 4: Results of Least-Squares Loss for Classification*

| Datasets: | Accuracy: |
|---|---|
| iris | 86.7% |
| mnist_small | 85.6% |

This algorithm produces a test RMSE that is on par with the k-NN algorithm. The least-squares loss regression algorithm using SVD returned test RMSEs of 0.349 for the mauna_loa dataset and 0.862 for the pumadyn32nm dataset. These RMSEs are very similar to the 0.441 for the mauna_loa dataset and 0.834 for the pumadyn32nm dataset returned by the k-NN algorithm. This algorithm did, however, return a lackluster test RMSE of 0.983 for the rosenbrock dataset. This is much greater than the test RMSE of 0.288 previously returned by the k-NN algorithm. All in all, however, the least-squares method for regression returns strong results.

With respect to classification, however, we can see that this algorithm is not as accurate as k-NN. This algorithm was 86.7% accurate for the iris dataset whereas k-NN was completely accurate. Moreover, this algorithm was only 85.6% accurate for the mnist_small, which pales in comparison to the 93.4% test accuracy returned by the k-NN implementation.

Run-time-wise, this algorithm is quite fast. This implementation was tested on the rosenbrock dataset using a d-value of 2. With this setting, this algorithm takes 0.004 seconds to run. This is faster than all k-NN implementations including the k-d tree implementation. To conclude, using SVD to compute the least squares is very effective and fast for regression.