Lucas Summers

lsumme01@calpoly.edu

CSC 466 Lab 0

Runtime Results:

Version	***Dataset***	***Metric***	***Runtime (s)***
1	Iris	Manhattan Distance	1.19
1	Iris	Cosine Similarity	1.25
1	Cal Housing	Manhattan Distance	DNF
1	Cal Housing	Cosine Similarity	DNF
2	Iris	Manhattan Distance	0.0015
2	Iris	Cosine Similarity	0.013
2	Cal Housing	Manhattan Distance	9.63
2	Cal Housing	Cosine Similarity	4.45
3	Iris	Manhattan Distance	0.0012
3	Iris	Cosine Similarity	0.055
3	Cal Housing	Manhattan Distance	3.96
3	Cal Housing	Cosine Similarity	3.89

Reflection:
Loading the datasets was fairly easy to understand, given the sklearn documentation. Both had different ways of converting the data into a pandas dataframe, but it was simple enough to figure out. I used the **run_metric** function to run each version, as well as handle timing and reporting the results in a uniform way.

Version 1 was also fairly easy to write. I made each pairwise metric a function that compares the two points provided. Than the **compute_pairwise** function computes the given metric on every pair of data points by using **iloc** to access the dataframe, storing the results of the metric in a python list. Using a NumPy array probably would've been much faster for working with a large number of pairs, or even not storing all the pairs would be more space and time efficient. For both metrics, I had to use **for a, b in zip(x, y)** in order to calculate the sums across all features. I also had to factor in that the most similar for a distance metric is the min of the results while the most similar for similarity is the max of the results, and the opposite for least similar pairs. Overall, the Iris dataset was fairly quick given the number of data points, while the California Housing dataset was very slow and did not finish in a reasonable time on the server, revealing the extremely inefficient processing methods and use of nested loops.

Version 2 was the hardest to write in terms of finding the needed information to fully utilize NumPy and pandas. For the metrics, both utilized converting the dataframe into a NumPy array. I then had to learn about broadcasting and all the special functions that interact with NumPy arrays. For example in

manhattan_distance_np function, using **np.abs(df[:, np.newaxis, :] - df[np.newaxis, :, :])** to calculate the differences between every pair of rows by reshaping the array was challenging to figure out initially. I realize that storing all these differences could have memory issues for a significantly large dataset. The **cosine_similarity_np** function was much easier in terms of finding the right NumPy functions for the job. I had to be careful to handle divide by zero errors, just replacing them with zeros in the results. The hardest to write was **find_minmax_np,** mostly because I had to deal with not factoring in the diagonal of the matrix and then using **unravel_index** as well as returning the results in the right form. Overall, due to the lack of nested loops and optimized NumPy array functions, this version ran exponentially faster than Version 1, which was especially noticeable for the California Housing dataset.

Version 3 was by far the easiest to write, because sklearn makes it very easy to compute pairwise metrics with just one function call. I wrapped these in my own functions so I could also use **run_metric** to run on the data and create a report. It also ran the fastest out of the other versions, with only small improvements over Version 2, presumably because its even further optimized for speed.