Homework #2 KNN CS460G

Alex Polus

2.17.18

**In order to execute, all relevant data files must be present in the same folder as each of my programs**

**Press Enter to advance through the KNN3 program**

1. K = 3 implementation:

Usage: Python3 KNN3.py

For my implementation, I decided to use Cosine Similarity to determine similarity between users. This was calculated with two vectors at a time, and the result was the dot product of the two vectors divided by the product of the magnitude of each vector.

After that, I converted the base and test files into 2D arrays of the form: array[row][column], and then made all the entries into floats. I also removed all of the last column, which was data irrelevant to this assignment.

Next, I established the number of users and the number of possible movies, which was crucial to my approach, because I used this to build a 2D array in the form: array[user][all the movies], where each index of [all them movies] was the rating of each movie corresponding to that index plus one, and the [user] was all of the system users where each user was that index plus one.

At this point, I had a generic multidimensional vector space to fill in with my actual data. Therefore, the “training” for this project was simply populating this 2D array with each user’s rating for each movie.

On line 140 I defined PredictAndFindError, which takes in a row from the test data, guesses what that user will rate that movie, and then compares it to the real value, outputting the error for that row. The rating is guessed by finding the 3 most similar users, excluding the user at hand, who has rated the movie of interest. Those three ratings are then averaged using a weighted average based on similarities, and that average is the hypothesis value used in the computation of mean squared error.

Using this function, I computed the error for all instances in the test set, and computed the average error.

**Average error across test for K=3 approach: 1.2527390933110114**

**Runtime: about 5 minutes**

1. Cross-Validation:

Usage: python3 CrossValidation.py

I found this portion of the program to be quite challenging. The structure of my code used for K=3 was not all applicable to this portion, as in part 1, I made certain assumptions about the size of certain arrays in my error function.

To properly cross-validate, I decided to make 5 folds out of my base data. To make these folds, I shuffled the rows into five piles one row at a time as if I was dealing a deck of cards.

Then, I created a “compliment” of each fold, which includes all of the rows that did not go into an individual fold. These compliments were then converted into the training data for each fold – a 2D array in the form: array[user][all the movies], and I populated each movie vector with that users ratings.

Finally, with some minor tweaks to the PredictAndFindError functions from part 1 to allow a mutable K value (number of neighbors) and to take the training set as a parameter, I intended to compute the average error across each fold for each K value, and consequently find the ideal K value for the model, however I was only successful with calculating the error across all folds and the average error for K=1. There must be some error with my iterator through k values on line 186 or something around there. That average is listed below. I believe with a minor tweak I would be able to get the error for all K values and choose the optimal one.

**Keep in mind, this program took like 3 hours to run, so you may not want to test it for running until completion – although you are certainly welcome to.**

Error values for….

K=1: 1.7938031249999997