John Geddes

CS 460G

Homework 2 Report

February 19, 2018

**Implementation Decisions**

I decided to use cosine similarity as the distance calculation because I could imagine the users as vectors but I couldn’t imagine them spatially because they are larger than 3 dimensions. And I’ve always thought of Euclidian distance as spatial distance. There wasn’t really a reason to pick one over the other so I just picked one.

When calculating similarity, I passed in the movieID of the movie I wanted to predict in the test file. Using the movieID, I only found neighbors to the current userID that have rated the movieID I wanted to predict. Using this pruning method, I believe the program is much faster than without pruning. I also did this because I can’t predict a rating on a movie that a neighbor hasn’t rated.

When predicting the rating based on neighbors, I used weights rather than majority rating. I used the similarity of each neighbor as the weight for the rating the neighbor gave. If multiple neighbors gave the same rating, I would sum all similarities for that rating. This method almost entirely prevents any sort of “tie” between ratings and I don’t have to guess. I’m not sure if this improved or hindered the accuracy of the prediction.

In the first part of this homework, I kept the number of neighbors k=3. Using the KNN algorithm that I wrote, the resulting mean squared error for the predictions on the provided test set was **1.7559912854**.

For cross-validation I decided to use the k-fold method. After reading up on what leave-one-out and k-fold are, I found it obvious that the k-fold method was significantly faster than the leave-one-out method. So, to save time during testing, I used the k-fold cross-validation method. I decided to loop through 5 values of k-folds along with the 5 values of k-neighbors. This produced 25 data points for average mean squared errors.

I implemented the k-fold cross-validation method by randomly dividing the original training data set into *k* folds, with each entry being a single line from the original training file. After dividing the data set into *k* folds, I iterated over all folds, choosing one to be the validation set and the rest of the folds to be the training set. I then trained the KNN algorithm on the training set and tested the predictions on the validation set. For example, if k=2, there would be 2 folds. I would make the first half of the data set be the validation set and the second be the training set. After training and predictions, I would make the first half be the training set and the second be the validation set. For larger numbers of folds, I would need to train and test *k* number of times because each fold has to be the validation set exactly once.

I also iterated the number of *k*-neighbors between 1 and 10 along with iterating the number of folds between 1 and 10. I chose 5 values of neighbors and 5 values for folds to produce 25 data points for average mean squared error. I used all odd numbers between 1 and 10 for the number of neighbors to test and all even numbers between 2 and 10 for the number of folds to test.

Based on the data on the next page, the lowest average mean squared error is 1.722740936 for when k-neighbors = 9. I can conclude that when the number of neighbors is higher, the prediction will be more accurate. Although, the testing shows that when the number of folds changed, the mean squared error changed erratically, regardless of the k value.

**Average Mean Squared Errors**

Before cross-validation, the mean square error for k=3 on the given test set was **1.7559912854**.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | Number of Folds | | | | |
| 2 | 4 | 6 | 8 | 10 |
| Number of Neighbors | 1 | 1.69034994698 | 1.70957446809 | 1.75346851654 | 1.76310043668 | 1.75808249721 |
| 3 | 1.75185577943 | 1.81382978723 | 1.66129032258 | 1.76939890710 | 1.66146993318 |
| 5 | 1.51961823966 | 1.61293743372 | 1.58369098712 | 1.50598476605 | 1.51444444444 |
| 7 | 1.35737009544 | 1.53411513859 | 1.62847965739 | 1.74890829694 | 1.71252796421 |
| 9 | 1.47664543524 | 1.50478214665 | 1.55328310011 | 1.59316427784 | 1.63395810364 |

Chart 1: The labelled data points are the average error for each value of k-neighbors.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | Number of Folds | | | | |
| 2 | 4 | 6 | 8 | 10 |
| Number of Neighbors | 1 | 1.68398727466 | 1.90191897655 | 2.01510248112 | 1.92519251925 | 2.00223713647 |
| 3 | 1.80275715801 | 2.03723404255 | 1.94174757282 | 1.98358862144 | 1.99552572707 |
| 5 | 1.61187698834 | 1.89266737513 | 1.92274678112 | 1.77573131094 | 2.00224719101 |
| 7 | 1.71792152704 | 1.84328358209 | 1.67274678112 | 1.80926130099 | 1.98444444444 |
| 9 | 1.47452229299 | 1.57704569607 | 1.59505907626 | 1.64466446645 | 1.90224719101 |

Chart 2: Trial 2 cross validation results with averages labelled

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | Number of Folds | | | | |
| 2 | 4 | 6 | 8 | 10 |
| Number of Neighbors | 1 | 1.68398727466 | 1.90191897655 | 2.01510248112 | 1.92519251925 | 2.00223713647 |
| 3 | 1.80275715801 | 2.03723404255 | 1.94174757282 | 1.98358862144 | 1.99552572707 |
| 5 | 1.61187698834 | 1.89266737513 | 1.92274678112 | 1.77573131094 | 2.00224719101 |
| 7 | 1.71792152704 | 1.84328358209 | 1.67274678112 | 1.80926130099 | 1.98444444444 |
| 9 | 1.47452229299 | 1.57704569607 | 1.59505907626 | 1.64466446645 | 1.90224719101 |

Chart 3: Trial 3 cross validation results with averages labelled