Tristan Basil

CS460G-001 Machine Learning – Dr. Brent Harrison

2/19/2018

Problem Set 2 - Report

This assignment implements a K-Nearest Neighbor algorithm that is used to predict movie ratings for a given user. There is one included program: Program2.py. It is implemented in Python 2. This report will provide a brief overview of the K-Nearest Neighbor implementation for K=3, and then with cross validation.

**Part 1: User-Based Collaborative Filtering**

Program2.py implements K-Nearest Neighbor using cosine similarity. It is implemented in Python 2 and uses the following packages:

1. **sys** to get command line arguments (the filename),
2. **random** to choose elements from the dataset to create folds in cross validation, and
3. **copy** to deep copy sets/lists.

For this part, it runs with K=3 by default and is run with the following command:

$ python Program2.py [file1] [file2]

Where [file1] should be replaced with the base training set (data/u1-base.base) and [file2] should be replaced with the test data set (data/u1-test.test). This will initialize the K-Nearest Neighbor algorithm, then display a progress indicator to show how many users’ ratings have been estimated by the algorithm. After running through each user, the final mean squared error will be displayed. Note that is can take around 6 minutes to run for this part. There were a few implementation decisions worth mentioning:

1. This implementation uses cosine similarity to determine the best neighbors.
2. If a user was missing a rating for a movie, it was defaulted to the value of 2.5. This was to simplify comparing the vectors of movie ratings.
3. If a user was missing in the training set completely, 2.5 was substituted for every movie rating for that user.

The final mean squared error results for the test set with K=3 was **1.42432333999**.

**Part 2: Determine the best K using cross validation**

Cross validation can be run to determine the best value K for the training dataset, and then to use that value with the test set. It is implemented using the fold-based method with 4 folds. It can be run in the same way as part 1, but with the ‘-cv’ flag:

$ python Program2.py [file1] [file2] -cv

Where [file1] should be replaced with the base training set (data/u1-base.base) and [file2] should be replaced with the test data set (data/u1-test.test). This will initialize the K-Nearest Neighbor algorithm, initialize 4 folds to cross validate, then display a progress indicator to show the progress of each fold for each value K (1, 3, 5, 7, or 9). After running through the 4 folds for a value K, the average mean squared error for that K will be displayed. The K yielding the minimum error will be chosen at the end, and run against the test set. Note that the program will take around three hours to run for this part. There were a few implementation decisions worth mentioning for the cross validation piece (the three aforementioned implementation decisions still apply since this is the same program):

1. As permitted by Dr. Harrison, only 5 K values in the range 1-10 were tested (1, 3, 5, 7, and 9). This seemed like a good spread to test different K values.
2. I chose the number of folds to be 4 to ensure the same number of examples across each fold, and to prevent longer than necessary runtime.

**Description of cross validation algorithm:** The algorithm I used to cross validate the best K value for this program is the fold-based method. Here is a basic outline of the steps I used to cross validate, which could be used to re-implement my method:

1. The training set is partitioned into 4 evenly sized folds, randomly selecting unique examples from the training set. 4 folds seemed to be a good balance between having enough examples in each fold and ensured that each fold would receive the same number of examples. Elements are randomly chosen from the training set to create each fold and each element can only pertain to 1 fold. For this training set, it results in 4 folds with 20,000 unique randomly selected examples from the training set each.
2. The next value for K (1, 3, 5, 7, 9) is chosen to cross validate each fold.
3. The next fold (0, 1, 2, 3) is chosen for the respective K value.
4. Every other fold is combined, and used to train the K-Nearest Neighbor algorithm. The chosen fold is then tested against this combination of the folds, yielding a mean squared error for that fold, which is added to the running total for the current value K.
5. If not out of folds, go back to step three, choosing the next fold.
6. If we have tested and added together the mean squared error for each fold against the others, calculate the average of the running total of mean squared errors for that value K to get the average mean squared error and store it for that K.
7. If not out of K values, go to step 2 and choose the next value K.
8. Find the lowest average mean squared error value created from the K’s. This is the K value that should be used for the test set.

**Observations:** When running cross validation for my implementation, the average mean squared error values noticeably decreased as K increased. This is likely due to following a more general trend as more users are chosen, which resulted in better performance on average. These general trends are much more defined in a data set with so many examples (like the training data provided), and yield more accurate results over the thousands of different examples tested. If I were to continue increasing K, I would expect the average mean squared error to stop increasing and actually decrease at some point, however. This is because eventually, the prediction would just approach the average for all users, instead of the most similar ones.

I also noticed that validating a fold took substantially longer performance-wise than just testing the test set with K=3. (~10 minutes per fold, vs 6 for the one test set). I think this is due to the different accesses into each dictionary adding overhead with the folds.

The average mean squared error for each K value (1, 3, 5, 7, 9) follows:

* K=1: **2.30442856943**
* K=3: **1.66527123692**
* K=5: **1.56653052192**
* K=7: **1.53358492419**
* K=9: **1.51243688182**

It’s also worth noting that the mean squared error on the test set for K=9, the best K value according to cross validation, was: **1.3312959859** (which was lower than the error for K=3).

Resources Used:

* <https://stackoverflow.com/questions/3282823/get-the-key-corresponding-to-the-minimum-value-within-a-dictionary> (used to find corresponding key to a minimum value in a dictionary)
* <https://stackoverflow.com/questions/26584003/output-to-the-same-line-overwriting-previous> (used to create the progress indicators for each fold/testing the test set)

Mean squared error for data/synthetic-1.csv Degree 9 3.71807083629

Weights [1.8593309374557587, 0.469029295044813, 1.0009040266195355, 0.02823990158445133, 0.3483149945881487, -0.254162867611155, 0.0025154290154203013, 0.1387823738059825, -0.014979951454633354, -0.02102694450334377]

Mean squared error for data/synthetic-3.csv Degree 9 0.353643751764

Weights [0.28532819186358716, 0.07814955691534739, 0.2712829944326238, 0.1072025390680838, -0.015972541046617345, -0.026648880978534153, -0.2947663830572262, -0.03149825327229785, 0.0791039847402398, 0.007755762483492536]

Mean squared error for data/synthetic-2.csv Degree 9 12.8524562324

Weights [-4.1919341790314135, 3.1626238520441317, 0.48418632994009536, 3.6504176341522476, 0.9890263470101822, 1.597811277741065, -0.6978045698203058, -2.226252006643883, 0.13613283979548946, 0.37993946918413085]