CS 294-34 Homework 4

Due: Thursday, November 12, 2009

Part 1: Collaborative Filtering

Preliminaries The collaborative filtering section of the assignment will make use of the recently released machine learning benchmarking site, MLcomp. Before tackling the problems, please visit `http://mlcomp.org` and create a user account. Ultimately, you will upload the algorithms you've implemented to MLcomp for evaluation. We strongly recommend the use of R for this section, as we are providing you with template files and a wrapper script that implement the MLcomp interface. The additional files for this assignment include a folder of template files (`template`) and an example collaborative filtering algorithm implementation (`example`) that implements the MLcomp interface. The `learn.R` and `predict.R` files in `template` contain partial implementations and comments indicating where additional code is needed.

How to Use MLcomp

1. Go to `http://mlcomp.org`.
2. Click "Start." Then, create an account by clicking "sign up" on right sidebar.
3. To view only results within the Collaborative Filtering domain, choose CollaborativeFiltering under "Select domain of interest:" in the right sidebar.
4. Explore the various programs and datasets. To see an example program, navigate to the program "lester-shrunk_item_mean-r." The code for this program can be downloaded by clicking the download link (the same code is provided in the `example` folder for the homework assignment).
5. The MLcomp API is relatively simple, and there is plenty of information on the Help page. It will be easiest to use the template files provided with the assignment, but in case you want to go further, here's a quick summary of the API.

   Your code will require a `run` script in the root directory. This script will be used in two ways, the first for training:

   ```
   run learn input_file
   ```

   Your code will do some training and will have to write its state to disk. The `input_file` contains a line for each rating, where each line is "userid movieid rating", separated by spaces, with the userid and movieid both integers and the rating a real number.

   At prediction time, this command will be called:

   ```
   run predict input_file output_file
   ```

   This code will have to read its learned state from disk and read data from `input_file`, which has the same format as above (the ratings in this file are dummy values and should be ignored). It will then write to `output_file` a single real-valued rating prediction on each line, corresponding to the same lines as `input_file`. These output values will be measured against the true ratings.
6. You can upload any program as a zip file, with the `run` script in the root directory. It's also convenient to include a `metadata` file in the root directory as well, which has the following format:

```
name: your_user_name-alg_name-r
description: Your description of the algorithm
task: CollaborativeFiltering
restricted_access: true
```

7. The MLcomp team (Jake Abernethy, Percy Liang, Alex Simma) would love to hear your feedback about the site. Feel free to make suggestions or bug reports in the comment box!

Problem 1: Matrix Factorization for Collaborative Filtering In this problem you will implement one of the matrix factorization techniques discussed in lecture. Begin by downloading and unzipping the movielens100k dataset from MLcomp. This dataset of 100,000 movie ratings on the scale of 1-5 stars was collected by the GroupLens Research Project at the University of Minnesota and is often used to compare collaborative filtering algorithms. The files `train` and `test` are the training and test sets for this dataset. Each line of train or test contains three numbers separated by spaces and corresponding to user id, movie id, and rating, respectively.

(a) In the `template` folder, you will find files `learn.R`, `predict.R`, and `run`. Select one of the matrix factorization methods discussed in lecture, and implement the training procedure (i.e. the learning of the model parameters) in `learn.R`. Your `learn.R` function will receive the name of a training set file as a command line argument. The `learn.R` function should read in the training set from the given file, learn the model parameters using the training set data, and then save the model parameters to disk (see an example of this procedure in the `example` folder).

In `predict.R`, you will implement the prediction procedure for your model. The `predict.R` function will receive two command line arguments: the first is the test set file name, and the second is the name of the file to which your predictions should be written. Your `predict.R` function should read in a test set from a given file, load the learned model parameters saved by `learn.R`, form real-valued predictions for each datapoint in the test set, and then write the predictions to the given prediction file (see an example of this procedure in the `example` folder). You should output your predictions in the order of the test set examples with one prediction per line.

Use the wrapper script `run` to train your algorithm on the movielens100k training set and to form predictions on the movielens100k test set. The relevant commands are `run learn train` and `run predict test predictionFileName`, where `train` and `test` are the dataset files in the downloaded movielens100k folder.

(b) Evaluate the performance of your method on the movielens100k `test` file, using the root mean squared error and mean absolute error metrics. Experiment with different numbers of dimensions (K) and describe how performance is affected. Are there any drawbacks to using a larger number of dimensions?

(c) Can you find anything interpretable in your model? Try clustering the learned movie vectors. Do any intuitive clusters result? Note that movie titles and genre information are provided along with the movielens100k dataset in (see `u.item` and `u.genre`). Investigate whether there is any relationship between the user vectors and user demographic features (also provided with the dataset in `u.user`). This kind of analysis can be done for any matrix-factorization based method.

Problem 2: The Practical Machine Learning Prize In this problem, you will compete against your classmates to win the coveted Practical Machine Learning Prize. Each student will upload two collaborative filtering algorithms to MLcomp. After all homeworks have been submitted, we will evaluate each of your submitted algorithms on a secret collaborative filtering dataset. The student whose submitted algorithm achieves the lowest RMSE on the hidden dataset will win the prize (yes, there is a real prize).

(a) Implement an alternative method from the collaborative filtering lecture (e.g. KNN, Naive Bayes, or even an integrated KNN-SVD model), using `learn.R` and `predict.R` template functions as in the

previous question. You may use the downloaded movielens100k dataset to evaluate and tune your model.

(b) Place your `learn.R`, `predict.R`, and `run` files in a single directory along with a `metadata` file (described in the "How to Use MLcomp" section). When specifying the names of your uploaded programs in `metadata`, use the convention

your_user_name`-`*algorithm_name*`-r`

(e.g. `lester-shrunk_item_mean-r`). Zip the directory containing your program files, and upload the zip file to MLcomp. Do the same for your matrix factorization program from the previous problem. Create runs for both of your programs on the movielens1m dataset, which contains 1 million ratings on the scale of 1-5 stars. Compare the performance of your algorithm with the performance of previously submitted benchmark algorithms (e.g. `lester-shrunk_item_mean-r` and `lester-mf_rsgd-r`). Write down the submitted names of your programs along with the RMSE achieved by each on the movielens1m dataset.

Part 2: Active Learning

Problem 3: SVM with Active Learning The active learning concepts discussed in class are pretty high-level. We will implement the "uncertainty sampling" concept here. Let's take the spam dataset from Assignment 1 (or some dataset of your interest), and perform classication using an SVM. Complete the following:

(a) Randomly set aside a subset of the data for testing; use the rest for training.

(b) Select increasing sizes of the training set and train a SVM from each training set. Plot the test error of each SVM versus the training size. For simplicity, you may use a single set of good parameters that were selected in homework 1.

(c) Start off with a small training set of 100 data points, and train the corresponding SVM. Using the current SVM, pick the next training data that is closest to the separating hyperplane (i.e. the data point with the highest uncertainty). To speed up the training process, you can select a batch of data points at a time. Add the selected data to the current training set, and retrain a SVM. Repeat the process until the entire training set is used. Plot the test error of each SVM versus the training size. Be sure to use the same test set for both parts 2 and 3.

(d) Compare the plots. Do they make sense?

(e) Bonus: Instead of Uncertainty Sampling, we can also use "Query by Committee". Pick a set of classication techniques (e.g. nearest-neighbor, decision tree, naive bayes) and repeat the previous experiment with the following criterion for choosing the next data point: select the point that has the most disagreements among the classiers. Did this approach perform better than Uncertainty Sampling?