Capstone Proposal

December 27, 2017

1 Machine Learning Engineer Nanodegree

1.1 Capstone Proposal

John Snyder December 27th, 2017

1.2 Amazon Book Recommendation System

1.2.1 Domain Background

Recommender systems are essential to a number of online services. Netflix uses an ensemble of algorithms to estimate which films users will enjoy based on their past ratings. Spotify and Pandora determine what songs to play based on past songs users have liked or disliked. Online companies are well aware that leveraging their users' data can result in greater usage and sales. In fact, companies are willing to invest large resources for even modest improvements to their recommendation algorithms. For example, in 2009 Netflix awarded $1,000,000 for the development of a new ensemble algorithm for their movie recommender.

In the mid 1990s, online newsgroups began using an early form of collaborative filtering known as Tapestry. Tapestry was a relatively simple filtering system, where users could define tags and fellow users of interest. For example, a user could say they want articles tagged as "funny" by users "Jason, Katy, and Kim." Soon after, Amazon followed Tapestry's model in their early recommendation system. Because of the size of Amazon's user and item base (even early in the company's history), a simple filtering system wouldn't work. Instead, Amazon developed a method for measuring user to user similarity. This system looks for two users with similar purchase histories and ratings, and recommends new items based on what the other user has purchased. Since then, this collaborative filtering mechanism has been at the heart of recommender systems. More recently, several new recommendation methods have appeared. Singular value decomposition and deep learning techniques such as restricted Boltzmann machines and auto-encoders are used to generated some set of latent features describing the underlying data generation process. This project will explore and compare the efficacy of these new techniques.

1.2.2 Problem Statement

While collaborative filtering remains a popular method for recommendation systems, new techniques have grown out of recent developments in machines learning. Singular value decomposition, restricted Boltzmann machines, and autoencoders have all shown great potential as recommendation algorithms. But do they actually outperform more traditional techniques? This project tests these algorithms against each other using a large, publicly available, dataset.

1.2.3 Datasets and Inputs

This project uses publicly available Amazon book reviews. The data was collected using a webcrawler to on all Amazon reviews posted between May 1996 - July 2014. In total, 142.8 million reviews were collected. This project uses a subset of the total dataset, only book reviews for users who have reviewed at least 5 books, giving 8898041 reviews covering 603668 users and 367982 items.

The inputs and outputs for this project are relatively simple. The data will be split between a training and test sample. User ratings in the training sample will be used to predict ratings in the test sample. While it is possible to use the text of a review itself using more complex machine learning techniques, such as recurrent neural networks, this project focuses only on the rating, which can vary from 1-5. I restrict the number of inputs in order to ensure a reasonable comparison across the different algorithms. Additionally, the data are restricted to users who have posted at minimum 5 reviews, in order to ensure the user-item matrix has sufficient density for both training and testing.

The dataset is available from the University of California San Diego.

1.2.4 Solution Statement

Given that the purpose of this project is to compare the performance of a range of machine learning algorithm, the solution is a straightforward comparison of the respective performance of each algorithm on a series of holdout samples. Performance is measured by the root mean squared error (RMSE) of the predictions on the test sample. In order to test both overall peroformance and reliability, the performance is measured across samples using K fold cross validation, and comparing both the mean and variance in performance.

1.2.5 Benchmark Model

A simple benchmark for this project is to take the overall mean score, as well as the mean of either each user or each item and use the mean to guess values in some holdout data. Below are baseline metrics using RMSE on 10 fold cross validation.

10 Fold Cross Validation Performance

	Mean RMSE	RMSE Variance
User	0.966	9.699e-7
Item	1.009	6.265e-7
Overall	1.058	5.161e-7

These results show that overall ratings tend to cluster in a relatively small range. In fact, just looking at the mean and standard deviation for all ratings, we get a mean rating of 4.250, and a standard deviation of 1.058. As expected, this matches with the overall mean the in the cross validation. This also indicates that we can generally get within about 1 star of a given rating by just guessing 4.25. However, the results also show that we can do slightly better by guessing the individual mean for each user. This seems to indicate some users are simply tougher reviewers than others.

The algorithms employed in this project take into account the correlation between users and items, which can, hopefully, result in a lower RMSE versus the baseline.

1.2.6 Evaluation Metrics

Given that ratings could be considered either ordinal or interval measures, there is some debate as to the best evaluation metric for such recommender systems. If ratings are taken as ordinal, the recommender system could be formulated as a multiclass classification problem. In which case ROC-AUC, F1, or cumulative gain could be used to measure performance. However, for this project, I intend to follow the standard laid out in the Netflix prize evaluation, which treats ratings as a continuous measure, using root mean squared error to measure performance (RMSE).

RMSE, as the name suggests, takes the square root of the sum of all squared errors, divided by the sample size. Essentially, this is giving us, on average, how far our estimate is from the actual value.

$RMSE = \sqrt{\frac{\Sigma_i (y_i - \hat{y})^2}{N}}$

1.2.7 Project Design

The overall goal of this project is to compare the baseline performance values to those derived from collaborative filtering, singular value decomposition, linear regression, restricted Boltzmann machines, and auto-encoders.

The workflow involves two major components. First, for each algorithm, construct a model that takes as input a user-item rating matrix. The model should train on one such matrix, and score performance on a second holdout matrix. Second, I will construct a pipeline that will determine training and test sets for a 10 fold cross validation.

While these components sound simple at first, there are significant challenges that will need to be addressed. First, in some cases there will be users or items that appear only in the training or test set. Meaning there will be no training for a given user-item pairing. For example, we may have a user that appears several times in the test data, but never appears in the training data. For such a user, we might simply guess the item's mean rating. While it might be tempting to use the user's mean from the test data, this is a form of information leak from the testing data, and therefore should not be used.

Second, the size of the data can be problematic. The data listed in a long edgelist type format can be represented with 3 numeric values for just under 9 million reviews, giving a total size of about 216 MB. However, if constucted as a user-item matrix, the resulting matrix has roughly 222 billion cells, which wont fit in memory. Therefore, the data will need to be stored as a sparse matrix representation, which will add some complexity. This also mean we can't generate predictions for all user-item pairs, because it resulting matrix will be too large. This is reasonable, however, because most user item pairs do not need predictions to measure performance. Additionally, most pairs will not have sufficient data to outperform the simple mean baseline. Therefore, the implementation of the algorithm should only store recommendation scores for which we have a target value in the testing data.

Therefore, the most difficult portion of this project will likely be implementing a pipeline for each algorithm that is capable of handling either a sparse matrix, or that can convert only portions that fit in memory to a dense matrix as needed. This will involve implementing several of my own methods to read the SciPy sparse matrix form into algorithms implemented using a combination of SciPy, NumPy, SKLearn, Statsmodels, and Tensorflow. In the Tensorflow case, matrix operations will be accelerted using an Nvidia GTX 1070 GPU.

Overall, this project will combine elements of both supervised and unsupervised learning, as well as neural networks. It will also involve a significant amount of performance optimizations. While the implementation will be complex, the results will appear in a simple, straightforward

table, that can informatively compare different recommender systems.

In [ ]: