1 Problem

"What should I do when I'm bored?" is a question that almost everyone has asked at one time. A brief look at common Google searches highlights the demand for activity ideas. We started this project to be able to answer this queston for our users. The only information that should be required from a user is a number of ratings of other businesses. Using only that information we want to be able to give a predicted rating for any other business. We should also be able to give a user a list of their highest predicted ratings as a good answer to "what should I do?". Additionally we would like to give a user some insight into why we are recommending a business. This will help to give confidence in our recommendations, and will also give the user some expectation of what to expect when they arrive at the business.

2 Data

We acquired the academic dataset from yelp. The rating consists of ?? users and ?? businesses. The businesses are all located in the vicinity of 30 different universities in the United States. There are a total of ?? ratings. We initially misunderstood the datasets. We thought we were obtaining two different datasets, one from Michigan and one from Princeton, but we actually recieved the entire dataset twice. Our initial plan was to get two different datasets, training on the first then finally testing on the second. Unfortunately we unknowingly trained on the entire body of data, so we weren't able to test on any fresh data.

For our application, this isn't as big of a problem as others in practice. This is because in practice, we will be able to tune the model on the entire body of data fairly often. Because of our optimizations, tests run fairly quickly making it possible to adjust the model fairly often. Keeping a set of fresh data to test on is important for situations where you wish to have a solution that will work on all datasets without any need for tuning and that really isn't our goal.

3 Methods

3.1 Collaborative Filtering

To give predicted ratings we decided to use a form of *Collaborative Filtering* (CF). CF techniques use the tastes of a large collection of users to predict the taste of a single user.

A very simple example of collaborative filtering would be to use a method such as difference squared to measure the distance between the ratings of any two users. Once we have defined our distance function, if we want to predict a rating of user b on business b we can do the following: Find the distance between u and each user with a rating for b, choosing the user v with the shortest distance to u. Now we say that since u and v usually have very similar ratings, u will probably feel the same about b as v does. This is a very naive method, that can be improved by things like taking more related users into account and averaging their rating for b.

Even with improvements to the previous method, we don't really take the way people make decisions about businesses into account. It won't be terribly accurate, and there are a number of situations in which you won't be able to get a rating at all. If none of the users related to a user have a rating for a business we wish to predict for, we will be unable to make any prediction for that business at all.

3.2 Non-negative Matrix Factorization

To correct for these problems we decided to use non-negative matrix factorization (NMF). NMF fixes both of the problems that are mentioned above. It decomposes relationships between users and businesses into some number of factors that influence the relationship. Because of this it ends up being a much better model for why people actually like businesses that they like. It can use less direct relationships between users, so even if closely related users don't have a rating for a business, we can still give a prediction for it.

NMF is one of many methods that solve the matrix factorization problem. The goal of matrix factorization is that given matrix M should be decomposed into matrices P and Q such that $P \times Q \approx M$. This means that if matrix M has dimensions $u \times b$, P should be $u \times k$ and Q should be $k \times b$. We know that b and u should be the number and businesses and users respectively, but what is k? In NMF k corresponds to the number of *latent factors* we believe determine the relationship between users and businesses.

The only input to the NMF algorithm is a matrix M of size $b \times u$ and the integer k. The output of NMF is then matrices P and Q. P contains a relationship between every user and every latent factor. Similarly Q describes relationships between businesses and latent factors. These relationships aren't on the same scale as the input ratings of M, if everything worked correctly they should be smaller. They are relative however. A value of .9 between a user and a factor suggests that that user has a stronger relationship to the factor than a value of .4. Because $P \times Q$ is an approximation for M, $P_u \cdot Q_b$ approximates a rating for user u on business b

The equation for calculating NMF unfortunately doesn't have a closed form, so we need to use an iterative gradient descent approach to discover good values for P and Q. This means that we initialize P and Q to some values, and then calculate the distance from $P \times Q$ to M. We then move the values in P and Q in the correct direction to decrease the distance between $P \times Q$ and M. This means that the value of $P \times Q$ should move closer to M with each iteration. At the end of each iteration we measure the difference between the previous distance and the current one. When we make a small enough change the algorithm terminates.

Each iteration is controlled by two additional parameters. α scales the amount that we increase or decrease the values in P and Q. A larger α can lead to faster convergence, but if α gets too big, it is possible that we will greatly overshoot the answer and simply oscillate around it without ever reaching convergence, or just reach convergence much slower than we would with a more appropriate value for α. We also use β to control the effect of our normalization term. A larger β will decrease the chance of overfitting but make convergence slower.

3.3 Framework

python site, mysql db so on...

3.4 Challenges

Our initial implementation had two problems: it used a dense matrix representation, and it was implemented in Python. Because the gradient descent algorithm only operates on non-zero ratings,