6701 HW2

Viren Bajaj

Question 1

I decided to implement a gaussian matrix factorization model and used MAP estimation on the movie lens dataset to predict the rating a user would give to an unseen movie.

The main reason behind the choice of MAP estimation over variational inference because MAP estimation has been known to perform well with a gaussian matrix factorization model.

The Data

The main data set in the file 'ratings.csv' consists of 610 users who have cumulatively rated 9724 movies for a total of 403344 ratings. The value of ratings range from 0.5 to 5.0 in 0.5 increments. Through the data in the file 'movies.csv', we also have access to the names of the movies and tags that describe their genre such as 'drama', 'comedy', etc. from a set of 19 tags including 'no tag listed'.

The Model

The generative process for the data was assumed to be as follows:

1. For each user i:
   Draw preferences vector $\theta_i \sim \mathcal{N}_K(0, \eta_\theta^2)$
2. For each movie j:
   Draw attributes vector $\beta_j \sim \mathcal{N}_K(0, \eta_\beta^2)$
3. For each rating x_{ij}:
   Draw predicted rating $x_{ij}|\theta_i, \beta_j \sim \mathcal{N}_K(\theta_i \cdot \beta_j, 1)$

Where $\mathcal{N}_K(0, \eta_\theta^2)$ represents a bank of K normal distributions with mean 0 and variance η^2. K is dimension of the latent space, η_θ, and η_β are the the standard deviations. These are hyper-parameters in our model.[1]

[1] In practice, instead of drawing the predicted rating from a normal distribution, I calculated it as: $x_{ij}|\theta_i, \beta_j = \theta_i \cdot \beta_j$. I did so to remove any uncertainty during the evaluation of the model, i.e., get consistent values for the root mean squared error of the predicted ratings.

Data Preparation

To prepare the data for model fitting, I **centered and scaled the ratings** values such that their median is 0 and the range is from -1 to 1. I did this because the rating and predicted rating need to have the same median and scale so they can be sensibly compared. The latent vectors have a range of -1 to 1 because they are normalized in my algorithm (more on this later). I realized this after quite some time. Initially, I was only dividing the rating by 5 while calculating the root mean squared error, which was incorrect.

I also split the data into a **train, dev,** and **test** set as follows: for each user, create a train, dev, and test of the movies rated by that user using the percentages $80\%, 10\%, 10\%$ respectively. This resulted in 402095, 50295 and 51790 data points in the sets respectively. An important feature of the split is to randomly shuffle the movies rated by every user before assigning them to the sets to remove any systematic bias in the data (e.g. ratings may vary with time and the ratings may be sorted based on timestamp). Further, to make the analysis reproducible, the seed used to shuffle the movies was fixed.

MAP Estimation

A **gradient coordinate ascent** algorithm was used to climb the gradient of the log joint, which served as a proxy for the log posterior since they are same within a constant additive factor (log of the evidence). The algorithm is given in the appendix in Figs 4, 5.

An important aspect in ensuring correct calculations was **normalizing the latent vectors** at initialization and then after every update. This was necessary to avoid underflows and nan's. Notice that while updating the preference vectors we sum over all movies rated by a user scaled by the errors in the ratings (and similarly all the ratings received by a movie while updating the attribute vector). This update when multiplied by the learning rate can snowball into extreme values resulting is underflows and nans within the course of a few epochs. Normalizing the values to lie between -1 and 1 makes sense because that is the normalized range of the dot product (the cosine function) which captures the similarity between preferences and attributes that we intend to capture.

The inference algorithm was chosen because it is massively **parallelizable**. This was an important feature of my code that made it efficient. The run time of one epoch went down by 4 times (from 20s to 5s) after splitting the work up into the number of CPUs on my machine(8). In fact the parallel computation saw a speed up of about 145 times (from 20s to .137s), the rest of the time in the update was taken to reconstruct the dictionary object that stored the latent vectors.

Another piece of code that made my code faster was the function I used to iterate over the pandas dataframe. Earlier I was iterating over the rows using *pandas.iterrow*s, which returned a Pandas Series of column values. Later, I found out that the function *pandas.itertuples*, returns a tuple of column values, which is significantly faster. This gave me 10x speed up in calculating the log likelihood of the data (from 5s to 0.5s).

Initially, while testing **convergence**, I had set the learning rate to 10 or 100 to view the overall behavior of the algorithm. And in fact I saw convergence when the ground truth rating wasn't scaled. It turned out that all the vectors were being set to their max normalized value of 1 in an attempt to be closer to all the ratings, most of which were greater than 1. This behavior vanished when the ratings were scaled and the log likelihood decreasing after the first step. Once the learning rate was $\sim .01$, the convergence was reasonably fast and converged to the same regions as with smaller learning rates ~ 0.001.

I also created a function to **search the hyper parameter space** for a set of parameters that minimized the root mean squared error on the dev set. The search was uniform over some reasonable ranges for the hyper parameters. In the future I would like to learn the standard deviations ηs through cross-validation. I tested K for values from 1 to a 1000. The value of K that produced the least RMSE on the dev set was 117, equal to 0.203. I specifically tested K=19 because that is the number of genre tags in the data set, however the dev RMSE was higher (0.207), even though the train RMSE was lower (0.136) than when K=117 (0.137). See Fig 1 for more details.

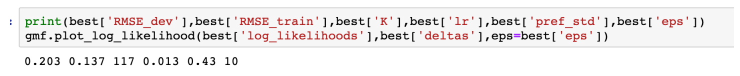
I attempted to **make sense of the latent attributes** of movies by sorting movies based on the component value of β_{MAP} and then inspecting the top 10 movie names and genre tags with the highest values of a given component. Visual inspection was nice but I couldn't see a pattern even when K=19. See Fig 2. I also plotted two attribute dimensions against each other, but didnt notice any obvious patterns. In the future I would like to try PCA or tSNE to reduce latent dimensions and visualize them. See Fig 3.

Question 2

Title: Granular Cross-domain Citation Recommendations using LDA

Abstract: Typical paper recommendation systems offer results on a document level.Yet in practice, different citations are sought after in different sections of the paper. Furthermore, often the most appropriate citations can be from different academic communities. We present a versatile citation recommendation system that can provide accurate inference for smaller texts, recommendations across domains, and usage of known citation networks. First, we evaluate the quality of inference provided by LDA, LDA with word embeddings, and LDA with word embeddings and citation network on the sub-document (section and paragraph) levels using the SemanticScholar dataset. We found that LDA with word embeddings and citation networks performs better on key metrics of mean precision, recall, F-measure, and reciprocal rank. Using our findings, we develop an LDA model augmented with citation networks in a new way. We show that it outperforms other models on the above metrics, especially improving cross-domain recommendations.

Annex



```
: print(best['RMSE_dev'],best['RMSE_train'],best['K'],best['lr'],best['pref_std'],best['eps'])
  gmf.plot_log_likelihood(best['log_likelihoods'],best['deltas'],eps=best['eps'])

  0.203 0.137 117 0.013 0.43 10
```

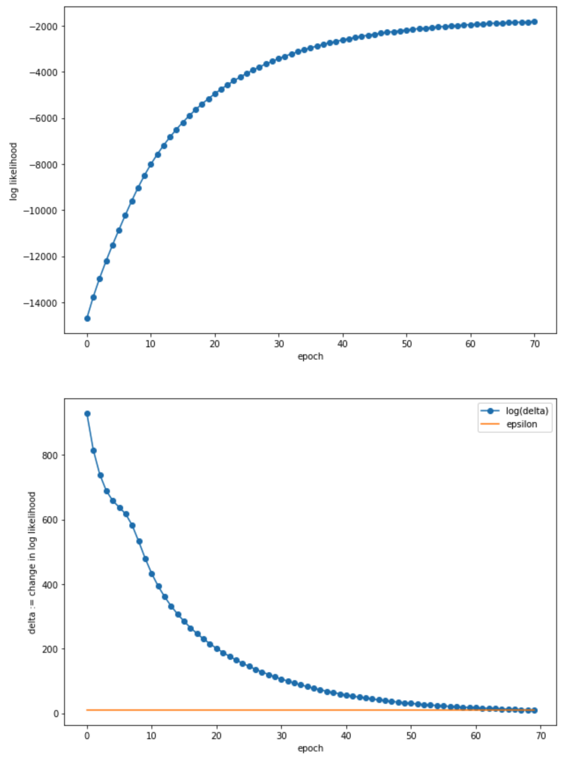


Figure 1: Plot log likelihood vs epoch (above) and delta vs epoch (below) of the hyperparameters that produced the lowest RMSE on the dev set. The corresponding RMSE on the test set was 0.202.

```
      movieId                         title  \
1939     2571              Matrix, The (1999)
2996     4011                   Snatch (2000)
3568     4886            Monsters, Inc. (2001)
6534    54286  Bourne Ultimatum, The (2007)
6772    60069                   WALL·E (2008)

                                              genres
1939                         Action|Sci-Fi|Thriller
2996                         Comedy|Crime|Thriller
3568  Adventure|Animation|Children|Comedy|Fantasy
6534                          Action|Crime|Thriller
6772  Adventure|Animation|Children|Romance|Sci-Fi
      movieId                                title                       genres
257       296                   Pulp Fiction (1994)  Comedy|Crime|Drama|Thriller
277       318  Shawshank Redemption, The (1994)                     Crime|Drama
1211     1610  Hunt for Red October, The (1990)    Action|Adventure|Thriller
2078     2762            Sixth Sense, The (1999)         Drama|Horror|Mystery
3196     4308                   Moulin Rouge (2001)        Drama|Musical|Romance
```

Figure 2: Top 5 movies with highest value in dimension 10 (first) and dimension 20 (second) taken from β_{MAP} with K=117. Notice we can't really tell if all of the movies belong to given genre.

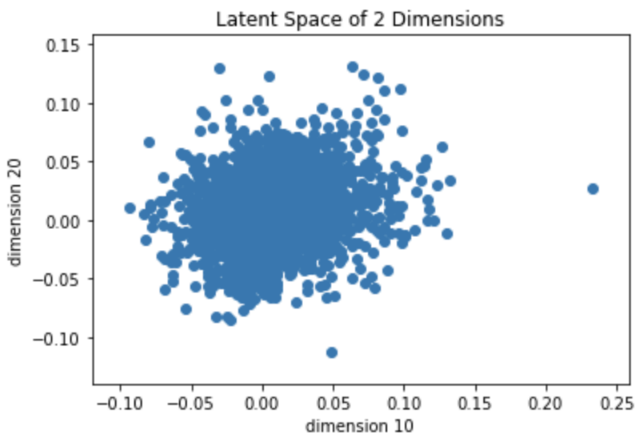


Figure 3: All values of dimension 20 against all values of dimension 10.Notice its hard to see any skew in the data.

Algorithm 6: Coordinate-ascent MAP estimation for matrix factorization.

```
Input: a matrix of ratings $\mathbf{x}$.
Output: MAP estimates of preferences $\boldsymbol{\theta}$ and attributes $\boldsymbol{\beta}$
for each viewer i do
    randomly initialize preferences $\theta_i$
for each item j do
    randomly initialize attributes $\beta_j$
maintain preferences and attributes $(\boldsymbol{\theta}, \boldsymbol{\beta})$
while $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\beta})$ has not converged do
    for each viewer i do
        update preferences $\theta_i \leftarrow \theta_i + \gamma \nabla_{\theta_i} \mathcal{L}$     (Equation 7.6)
    for each item j do
        update attributes $\beta_j \leftarrow \beta_j + \gamma \nabla_{\beta_j} \mathcal{L}$     (Equation 7.7)
return preferences and attributes $(\boldsymbol{\theta}, \boldsymbol{\beta})$
```

Figure 4: MAP estimation algorithm.

$$\nabla_{\theta_i} \mathcal{L} = -(1/\lambda_\theta)\theta_i + \sum_{j \in \mathbf{x}_i} (x_{ij} - \theta_i \cdot \beta_j)\beta_j \tag{7.10}$$

$$\nabla_{\beta_j} \mathcal{L} = -(1/\lambda_\beta)\beta_j + \sum_{i \in \mathbf{x}_j} (x_{ij} - \theta_i \cdot \beta_j)\theta_i. \tag{7.11}$$

Figure 5: Gradient update for MAP estimation for Gaussian Matrix Factorization.