

CSE 258 - Assignment 1

Kaggle username: Haakon Hukkelaas

1. Visiting Prediction

The model I chose to use was to use jaccard similarity to find the most similar user for the user we want to predict for. This model ended up with a accuracy of 0.757 on kaggle. The model has big issues with computing time. Improvements to the model could be incorporating kNN, and extracting better features for the jaccard similarity, however was not done due to time constraints.

1.1 Jaccard similarity model

The model computes the jaccard similarity between all users. The jaccard similarity is based on what categories a user has visited before, where we take all the categories for each user and computing the similarity

$$jaccard = \frac{A \cap B}{A \cup B}$$

To predict for a given user if he is going to visit the business, we look at the k most similar users and check wether or not they have visited the business. If any of these have visited the business, we return True, else return False.

1.2 Computing difficulties

When computing similarities for each user we have to compare each user to all other users. With 20,000 users, this results in 400,000,000 computations, and is becoming an infeasible problem. We also tried to compute it for each prediction, but however with 200,000 test examples this is even worse.

The computing difficulties makes it hard to test out different features, and resulting in a 30 minute compute time each time you change either number of users or features for each user.

1.3 Improvements

Using kNN with this approach could method could improve the

2. Rating prediction

Latent factor model is the model that resulted in the best rating prediction for both local testing data, as well as on Kaggle. The final score was $RMSE = 0,75991$, only 0.00561 worse than the winning submission. As the number of interactions we have is somewhat sparse, we could improve the model by incorporating features, such as considering the bias toward a given category for a user or business. This report gives an overview of the methods used, and the actual code is given in the appendix.

2.1 Latent factor model

The model that resulted in the best result was

$$\min \sum_{u,i \in U,I} (R_{u,i} - \alpha + \beta_u + \beta_i + \gamma_u \gamma_i^T)^2 + \lambda (\sum_{i \in I} \beta_i^2 + \sum_{u \in U} \beta_u^2 + \sum_{u \in U} \|\gamma_u\|_2^2 + \sum_{i \in I} \|\gamma_i\|_2^2)$$

where λ is the regularization constant, α is the default bias, β_u, β_i is the bias for the user and item, and $\gamma_u \gamma_i$ is a dimensionality reduction of the $R_{u,i}$ matrix. Optimizing this can be done by gradient descent, where I used stochastic gradient descent.

The gradient descent should be performed by all the parameters, including α . When updating for a user I got the best result when considering all the items he has interacted with to calculate the gradient.

2.2 Learning speed

I started doing batch gradient descent of the model, which gave a horrible slow learning speed. Changing to updating β_u, γ_u for all the users, then updating β_i, γ_i for all items gave a learning speed considerably faster. α was updated for each epoch.

2.3 Cross validation

To optimize our model on the test set, I used a cross-validation method to choose the number of dimensions K for γ_i, γ_u , and the regularization constant λ to prevent overfitting. In (3.1) in the appendix is given results from cross-validation, where the best results for me was with $\lambda = 4$ and $K = 1$.

2.4 Improvements

The model currently only uses biases towards a user and a business. Incorporating features such as bias towards categories could be implemented, but was not tried for this submission.

3. Appendix

3.1 Rating RMSE vs λ

