CS181 P3: Predicting Music Tastes

The Statistically Significants

Guy Zyskind Yuzhong Meng Qinxia Wang

***Abstract*—This paper summarizes our best model for the 3rd practical: Predicting Music Tastes. Using this approach we were ranked 3rd on Kaggle.**

I. The Data

First, we observed that number of plays for all observations follows a right-skewed distribution (seemingly lognormal or power-law) with a very long tail (Fig. 1A). To make the numbers more amenable to fitting, we first took a log transformation, resulting in a bell-shaped curve (Fig. 1B). This gave us control over the variance of the weights we fitted in our model, which we will describe in detail. It also allowed us to use mean of log(plays) rather than median of log(plays)– means would represent the data poorly for the heavily skewed pre-transformation distribution–and the mean of the log was found to yield better results than the median.

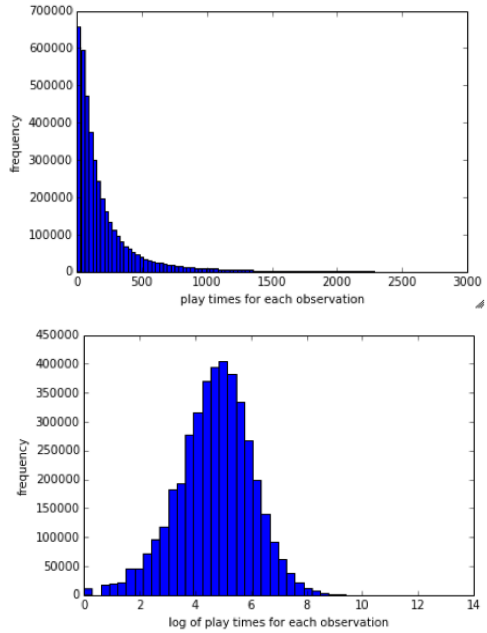

Fig. 1: Log transform of plays give a bell-shaped distribution. A: Right-skewed distribution of plays before log transform. B: Bell curve after log transform.

II. The Model

A. Collaborative Filtering using Matrix Factorization

For collaborative filtering, we used latent factor models and matrix factorization to capture mutual relations across users and artists. From our tests, it turned out that collaborative filtering using matrix factorization is superior to traditional content-based model.

The main idea of our matrix factorization model is to map users and artist information to a latent factor space and the estimated user-artist association $\hat{r}_{ui}$ is modeled as an inner product of the two vectors. In our example, it is the number of plays given each user-artist pair. Here, we have each artist associated with vector q_i, and each user associated with p_u.

$$\log \hat{r}_{ui} = q_i^\top p_u \tag{1}$$

We used a similar version to Koren [2] and Funk's [1] matrix factorization algorithm which also includes the global mean and the user-artist bias b_{ui}. Funk computed this bias explicitly from the samples and Koren treated them as parameters. We received slightly better results when treating them as parameters, although in both cases we obtained results that show that learning the biases is almost the same as analytically computing them. Given this, the refined objective function is given by –

$$min_{p,q,b} = \sum_{(u,i)\in K} (\log r_{ui} - \mu - b_u - b_i - q_i^\top p_u)^2 + \lambda_{p,q}(||q_i||^2 + ||p_u||^2) + \lambda_b(b_i^2 + b_u^2), \tag{2}$$

where b_u is the user bias and b_i is the artist bias. Each of them capture their respective deviation from the mean.

B. Adding Content-based Recommendations

User features such as age, sex and country can influence their preference for artists. We incorporated this per-user information in our model with w_u – a vector of weights for each user corresponding to the user's feature vector x_u. This could be thought of as an extension to the user bias, where b_u is the intercept and w_u are the covariates' weights. Our improved (and final) model is therefore –

$$min_{p,q,b,w} \sum_{(u,i)\in K} (\log r_{ui} - \mu - b_i - b_u - w_u^T x_u - p_u^T q_i)^2 + \lambda_{pq}(||p_u||^2 + ||q_i||^2) + \lambda_{bw}(b_i^2 + b_u^2 + w_u^2), \tag{3}$$

Note that similarly, if we had information about the artists, we could have modeled them using x_i and the weight vector w_i. For x_u, we only used the data given for the practical – (age, sex, country). The latter two were converted to one-hot-encoded categorical variables.

III. Optimizations

A. Stochastic Gradient Descent

We used stochastic gradient descent to minimize the objective function with respect to q, p, b and w. For each user-artist observation in the training set, we have a prediction error of $e_{ui} = \log r_{ui} - \mu - b_i - b_u - w_u^T x_u - p_u^T q_i$. The update rules are:

$$q_i \leftarrow q_i + \gamma(e_{ui} p_u - \lambda_{p,q} q_i) \quad (4)$$
$$p_u \leftarrow p_u + \gamma(e_{ui} q_i - \lambda_{p,q} p_u) \quad (5)$$
$$b_i \leftarrow b_i + \gamma(e_{ui} - \lambda_{b,w} b_i) \quad (6)$$
$$b_u \leftarrow b_u + \gamma(e_{ui} - \lambda_{b,w} b_u) \quad (7)$$
$$w_u \leftarrow w_u + \gamma(e_{ui} x_u - \lambda_{b,w} w_u) \quad (8)$$

In addition, to help get out of local minima, we also made use of a momentum term m in stochastic gradient descent process. The optimal value of m determined by cross-validation is 0.8. The actual learning was done in mini-batches of approximately 400K samples each.

B. Regularization

Given the many parameters in our model and the nature of the data, regularization was key. By splitting the lambda between the collaborative components (p and q) and the content components (b and w), we were able to improve our results. Using CV, we determined that the the optimal results were obtained with $\lambda_{p,q} = 0.1$ and $\lambda_{b,w} = 1$.

We also had to be careful about overfitting in general. We empirically noticed that after 10 epochs, our CV error started to increase while the training error kept going down. We therefore set our maximal number of epochs to 10, which converged reasonably fast.

C. Non-linear Prediction

Our basic prediction function is given by –

$$\hat{r}_{ui} = \exp(\mu + b_i + b_u + w_u^T x_u + q_i^\top p_u) \quad (9)$$

Given that the user-median provided reasonably good performance, we decided to leverage it by passing the predictions through a non-linear function that pushes the results towards the median.

Protocol 1 describes the non-linear function. First, we compute the *median absolute deviation* MAD_u for each user. Then, for every prediction, if it deviates more than c times from MAD_u, then we slightly push it towards the median, as seen in line (4). Function f determines the direction to push to. It is simply a step function that is -1 if the prediction is larger than the median and 1 otherwise. Using CV, we determined that $c = 2$ and $lambda = 0.1$ worked best.

Protocol 1 Non-linearity applied to $\hat{r}_{ui}$

1: **for each** $(u, i, \hat{r}_{ui})$ in X
2: $\quad \hat{\sigma}_{\hat{r}_{ui}} = |\hat{r}_{ui} - median_u|$
3: $\quad$ **if** $\hat{\sigma}_{\hat{r}_{ui}} > cMAD_u$ **then**
4: $\quad\quad \hat{r}_{ui} = \hat{r}_{ui} + \lambda f(\hat{\sigma}_{\hat{r}_{ui}} - cMAD_u)$
5: $\quad$ **end if**
6: **end for**

References

[1] Simon Funk. Netflix update: Try this at home, 2006.

[2] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.