EMI Music Data Science Hackthon: How I Did It

Dell Zhang
DCSIS, Birkbeck, University of London
Malet Street, London WC1E 7HX, UK
dell.z@ieee.org

August 4, 2012

Abstract

This document describes my approach to the EMI Music Data Science Hackathon, a 24-hour competition aimed at building a predictive model about how much a user is going to like a new song.

1 Introduction

My entry in the EMI Music Data Science Hackathon[1] — *zmusic* — could simply be summarised as "chucking everything into a Random Forest", which is also the winning strategy used by Ben Hamner in the previous Data Science Hackthdon[2]. After the competition, I tried the technique of Factorization Machines and found that it would be able to achieve a better performance for this task.

I only used Python[3] for coding throughout the competition. The 64-bit edition Python would be needed to handle the dataset and avoid memory errors. The code for replicating my approach has been made available on GitHub[4].

The rest of this document is organized as follows. In Section 2, I describe the features extracted from the data. In Section 3, I present the learning algorithms employed to build the predictive model. In Section 4, I report the results of my submission and my post-competition experiments. In Section 5, I make concluding remarks.

2 Features

The data used for this competition come from a subset of the EMI One Million Interview Dataset[5]. The data files were first cleaned and encoded manually

[1]http://www.kaggle.com/c/MusicHackathon
[2]http://goo.gl/V8RIJ
[3]http://www.python.org/
[4]https://github.com/dell-zhang/zmusic_code
[5]http://musicdatascience.com/emi-million-interview-dataset/

using Unix tools (cat, cut, split, grep, sort, wc, etc.) and a text editor (search, replace, etc.). The data files were also pre-processed so that all data could be saved in binary format and then loaded quickly through Python's pickle mechanism.

For each training or test example given in the form of a tuple [`artist_id`, `track_id`, `user_id`, `time_id`], I expanded it with the attributes describing that user's (a) demographics, (b) his preferences for music, and (c) his opinions about that EMI artist — basically all the information that is provided about the corresponding user and artist. Then I represented each example as a feature vector: each numerical attribute was represented as one feature; while each categorical attribute (including those `ids`) with k distinctive values was represented as k binary indicator features. Specifically, the features that I have used include: `artist_id` (50 binary indicator features), `track_id` (184 binary indicator features), `user_id` (50928 binary indicator features or 1 real-valued feature, see the explanation below in Section 3.1), `time_id` (24 binary indicator features), `gender` (1 integer feature $\in \{-1, 1\}$), `age` (1 real-valued feature $\in [0, 1]$), `working` (13 binary indicator features), `region` (4 binary indicator features), `music` (5 binary indicator features), `list_own` (1 real-valued feature $\in [0, 1]$), `list_back` (1 real-valued feature $\in [0, 1]$), `q_xx` (19 real-valued features $\in [0, 1]$ corresponding to that user's answers to the 19 questions about his preferences for music), `heard-of` (4 binary indicator features), `own-artist-music` (5 binary indicator features), `like-artist` (1 real-valued feature $\in [0, 1]$), and `w_xx` (81 integer features $\in \{-1, 1\}$ corresponding to whether each of those 81 words was used by that user to describe that EMI artist).

If a categorical attribute is missing, its binary indicator features are just all 0. With respect to numerical attributes, the missing values for the integer features (`gender` and `w_xx`) are filled as 0; while the missing values for the real-valued features (`age`, `list_own`, `list_back`, `q_xx`, and `like-artist`) are filled as -1.

Finally, for each example, the user's `rating` on the given track is considered as the target variable. Thus, this predictive modelling task is formulated as a regression problem.

3 Algorithms

3.1 Random Forest

Random Forest (RF) [1], an *ensemble learning* method [3,7] for classification or regression built on top of decision trees, has kept showing best performances on a variety of real-world data mining problems [2]. If you are not familiar with this technique, please refer to an explanation of RF in layman's terms[6] and a demonstration of RF in action[7].

[6]`http://goo.gl/3Mdu5`

[7]`http://goo.gl/1O5P4`

Due to the great success of RF in many data mining competitions, I decided to take it as my first choice for this problem. I used the RF implementation provided by the Python machine learning library scikit-learn[8], with all the features described in Section 2. There was one serious obstacle though: that RF implementation does not support expressing data in a sparse matrix. To get around of it, I had to represent the categorical attribute `user_id` as one real-valued feature (`user_id`/50928), though in principle it should be represented as 50928 binary indicator features. Hence in the end a RF with 395 features was used to make my submissions.

I only tuned two major parameters of RF via cross-validation: one is the the number of trees in the forest (`n_estimators`), and the other is the size of the random subsets of features to consider when splitting a node (`max_features`). For `n_estimators`, the larger the better performance, but also the longer time the computation will take. A fairly good performance could be achieved by using 10 trees. I finally settled at a RF with 60 trees, as it took about one hour to run on my laptop while the performance improvement brought by using more trees became insignificant. For `max_features`, the lower the greater the reduction of variance, but also the greater the increase in bias. The rule of thumb is to use $\texttt{max\_features} = \texttt{n\_features}$ for regression problems and $\texttt{max\_features} = \sqrt{\texttt{n\_features}}$ for classification problems, where `n_features` is the number of features in the data. Our task is a regression problem, but the target variable values, i.e., ratings, turned out to be highly clustered, ash shown by Figure 1. Therefore it is somewhat similar to a classification problem. According to the cross-validation experiments and the public leaderboard, setting `max_features` to $\sqrt{\texttt{n\_features}}$ instead of its default value `n_features` did improve the performance slightly.

I was not able to take advantage of the multi-core parallel computation functionality provided by the `n_jobs` parameter. Since the main memory could only hold only one copy of the data, running multiple jobs in parallel would make the hard disk busy and the speed would actually be slower.

3.2 Factorization Machines

Although the above RF based method has utilised all the attributes, it has not fully exploited all the information embedded in the data, as it has ignored the *latent factors* of user-track interactions, user-artist interactions, and so on. Such latent factors are known to be very helpful for modelling dyadic data, e.g., in recommender systems, as demonstrated by the Netflix Prize [9] [5]. So in the second half of the competition, I was attempting to apply Singular Value Decomposition (SVD) to this problem. The standard SVD provided by NumPy or SciPy was not useful as it could not handle incomplete matrices. What I needed was Simon Funk's SVD[10] that models observed ratings only. However, I did not get enough time to complete implementing my SVD algorithm.

[8]`http://scikit-learn.org/`

[9]`http://www.netflixprize.com/`

[10]`http://sifter.org/~simon/journal/20061211.html`

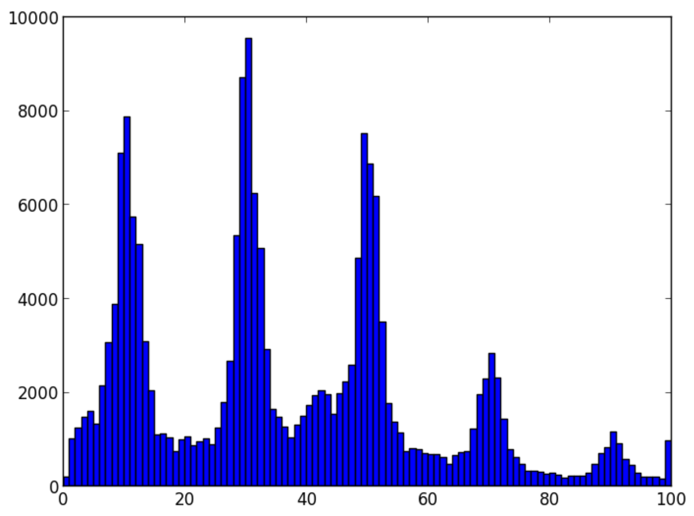

Figure 1: The histogram of ratings in the training data.

After the competition, I discovered the technique of Factorization Machines (FM) [6] developed by Steffen Rendle, which is a generic latent factor model that encompasses most matrix factorization models (incl. SVD) by feature engineering. Furthermore, a software toolkit for FM, libFM[11], is freely available for academic purposes. Since libFM supports sparse data format, I was able to represent the the categorical attribute `user_id` as 50928 binary indicator features, which resulted in 51322 features in total.

I carried out cross-validation experiments using libFM with all those features, and found that it would be able to achieve a better performance for this task. Bayesian inference using Markov Chain Monto Carlo (MCMC) was chosen to be the optimisation method for training the FM with 100 dimensions in 1000 iterations, and the standard deviation parameter for initialising two-way factors was set to 0.25 which yielded a good convergence speed.

4 Results

Table 1 shows the performances of RF (`n_estimators=60, max_features='sqrt'`) and FM (`-method mcmc -dim '1,1,100' -init_stdev 0.25 -iter 1000`), measured by Root Mean Square Error (RMSE).

5 Conclusions

Once again, RF proved to be amazingly powerful, when a predictive model needed to be built quickly on a complex dataset with many different types of

[11]`http://www.libfm.org/`

Table 1: The performances of RF and FM.

RMSE	2-fold cross-validation	public leaderboard	private leaderboard
RF	14.59553	13.76513	13.80559
MF	14.19240	-	-

features. It should be considered as the baseline learning algorithm to start from, for typical classification or regression problems. Moreover, FM exhibited outstanding performance for this task, suggesting the critical importance of identifying the latent factors underlying the interactions between users, artists, and tracks, etc. Since RF and FM address different aspects of the problem, it is very promising to blend their results [4].

6 Acknowledgement

I thank EMI[12] for providing us this valuable dataset and challenging us to this interesting problem. I am grateful to Data Science London[13] and Kaggle[14] for their excellent organisation of this competition.

References

[1] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[2] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *ICML*, pages 161–168, Pittsburgh, PA, USA, 2006.

[3] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2nd edition, 2009.

[4] M. Jahrer, A. Töscher, and R. A. Legenstein. Combining predictions for accurate recommender systems. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 693–702, Washington, DC, USA, 2010.

[5] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 426–434, Las Vegas, NV, USA, 2008.

[6] S. Rendle. Factorization machines with libFM. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57:1–57:22, May 2012.

[12] `http://www.emimusic.com/`
[13] `http://datasciencelondon.org/`
[14] `http://www.kaggle.com/`

[7] G. Seni and J. F. Elder. *Ensemble Methods in Data Mining: Improving Accuracy Through Combining Predictions*. Morgan & Claypool Publishers, 2010.