CS63 Spring 2017
Random Forests for Music Rating Predictions

Sayed Malawi

May 8, 2017

1 Introduction

In this project, an ensemble of random forest regressors were used to predict users' ratings of artists' music given training information about specific users and artists.

The random forest implementation used in this project (RandomForestRegressor from scikit-learn.ensemble) involves both bagging and boosting ensemble methodology. The ensemble is made up of a collection of decision trees, each of which is trained on a bootstrap sample (sampling with replacement) of the training data. Additionally, each tree is constructed by choosing a random subset of the features and finding the best split along that subset, imparting a degree of randomness in the decision tree algorithm. These trees are trained to completion, i.e. leaf nodes being the result of single-feature splits. This splitting approach causes individual trees to be more biased towards specific splits, but improves the variance of the model when the entire ensemble is considered.

The data set used for this project was provided by the now-defunct EMI record label, for a 2012 hackathon hosted online by Kaggle. The goal was to predict users' ratings of music from artists signed to the EMI label. User data (collected through interviews in the United Kingdom) consisted of a total of over one hundred features, including demographics, answers to music preference questions, and word descriptors of specific artists' songs; in the training set, the ratings were provided as well. Since the ratings were on a scale of 0-100, this challenge was approached as a regression problem.

The best performance using five-fold cross-validation was 14.712 (measured by root mean squared error). For comparison, the lowest error as reported on the Kaggle leaderboard was 13.246.

2 Method and Details

The data set consisted of four .csv files: a training set, a test set, a user information file (users.csv), and a user-artist information file (words.csv). The training set contained 188,690 samples, each consisting of unique artist, track, and user IDs, the time the interview was conducted, and the rating the user gave the artist's track. The test set contained the same information minus the rating (the target attribute). The user information file contained seven features corresponding to the 48,645 users' demographics as well as answers (on a 0-100 scale) to nineteen questions regarding listening habits. The user-artist information file consisted of 118,301 lines of interviews conducted with users about specific artists' music, primarily consisting of yes/no answers (0/1) to whether the music matched any of a subset of 82 possible descriptive words.

A significant number of user interviews in both information files are missing (i.e. user ID missing in users.csv, or user-artist pairing from the training set missing in words.csv). The organizers of the competition explained that at least some of the missing data was due to "extenuating circumstances".

The training and user information files were quickly imported into numpy arrays via the genfromtxt() method with a comma delimiter. Normalization of the 0-100 values in users.csv was attempted but yielded worse scores than without normalization. The primary challenge in preparation of the data was addressing NaN values in the converted numpy arrays.

Many of these values were the result of the genfromtxt() method's handling of string field entries, which were changed to NaN in the numerical numpy arrays. This included the first line of each file (field titles) and feature columns with string answers (mostly demographics in users.csv). These rows and columns were all removed. (The feature columns were removed with the intention of being reinstated once simple models were running correctly, but after strong performance using only numerical features, the string features remained deleted in light of the time constraint.)

The other source of NaN values was missing entries in feature columns. In words.csv, each user only responded with regards to a subset of the 82 possible words, leading to missing fields being filled with NaN in the array conversion. Similarly, users occasionally did not respond to every question in users.csv. These feature columns were completed by average value imputation, functionality imported from the scikit-learn library. The mean value of each feature column was inserted in place of each NaN value in said column.

The stripped-down arrays from users.csv and words.csv were moved to dictionaries, the former being indexed with the user ID and the latter with a tuple of user and artist IDs. Each key was mapped to a numpy array containing the features for each sample. Afterwards, the training array was built by combining the relevant user and user-artist information for each sample in the training set. A one-column array was used to hold the known rating labels. To handle the aforementioned cases where the user or user-artist combination was missing from the data, the dictionary get() function was called with a default argument, which was the median feature vector for the respective feature array.

After attempting to use a variety of regressors as detailed in the Results section, the final ensemble used was a random forest of 200 decision trees, splitting the best feature from a random subset of log2(n) features, where n is the number of total features. For three-fold cross-validation, this model took a total of BLANK seconds to train. Each decision tree was trained down to single-feature splits (i.e. to completion).

Since the competition was too old to accept test set submissions on the Kaggle website, the accuracy of the model was tested using three-fold cross-validation, using scikit-learn's cross_val_score function. Three-fold validation was considered appropriate given the relative sizes of the training and test sets. The values reported in the Results section are the averages from this cross-validation.

3 Results

The final model involved a random forest ensemble of 200 trees, each considering log2(n) features during splitting, and with bootstrapping enabled. When performing three-fold cross-validation using this model on the entire training set, a root mean squared error (RMSE) of 14.712 was obtained, with training and validation taking place in 180.0 seconds.

The performance of random forests when compared to a single decision tree (considering all

features when looking for the best split) is shown in Table 1. A variety of depth caps were explored to prevent overfitting. The lowest observed RMSE was 17.217 with depth capped at 20. Although runtimes were impressive, the random forest model performed better with regards to actual error.

Table 1: Decision tree performance.

Algorithm	Max depth	RMSE	Runtime
Random forest	none	14.755	95.05 s
Decision tree	4	18.580	2.92 s
Decision tree	10	17.238	5.94 s
Decision tree	20	17.217	9.73 s
Decision tree	none	17.983	13.30 s

Two other ensemble models were considered: AdaBoost and gradient boosting tree regressors. These models were found to be ill-suited for handling the sheer size of the data being trained upon. In particular, finding optimal parameters for the gradient boosting algorithm was difficult due to the prohibitively long runtime. A sample of results is shown below in Table 2.

Table 2: Comparison of ensemble regressors.

Algorithm	Max depth	# estimators	RMSE	Runtime
Random forest	40	100	14.755	95.05 s
Gradient boosting	3	100	16.386	209.9 s
Gradient boosting	10	100	15.100	3307 s
AdaBoost	-	50	19.193	63.77 s
AdaBoost	-	100	19.249	70.70 s

The maximum depth of the random forest regressors was varied, as shown in Table 3. For the ensemble regressors, it was found that building trees with low depth caps gave less optimal RMSE when compared to the results using individual decision trees. This could be attributed to overfit trees being better suited to the ensemble voting system. However, large depth limits did marginally improve the error while reducing runtime; 40 was found to be the best value.

Table 3: Random forest depth capping.

Max depth	RMSE	Runtime
10	16.836	31.99 s
20	15.426	56.04 s
30	14.858	77.25 s
40	14.738	88.77 s
50	14.743	92.94 s
none	14.755	95.05 s

After setting the depth cap at 40, other parameters varied were the number of classifiers and the function used to determine size of the random subset of features upon which to split. The results of these variations are displayed in Tables 4 and 5, respectively. Increasing the number of estimators could likely improve error indefinitely, but gains are marginal past n = 100. Interestingly, the choice of splitting function had a significant effect on runtime but not on RMSE.

Table 4: Number of estimators (log2 splitting function).

# estimators	RMSE	Runtime
10	15.223	12.11 s
50	14.791	44.74 s
100	14.738	88.77 s
120	14.729	108.1 s
140	14.727	125.5 s
200	14.712	180.0 s

Table 5: Splitting subset function (# estimators = 100).

Function	RMSE	Runtime
all n	14.857	799.6 s
sqrt(n)	14.743	121.5 s
log2(n)	14.738	88.77 s

Lastly, bootstrapping of the data is an optional parameter in scikit-learn's random forest regressor implementation. This parameter is set to true by default. Switching off the bootstrapping functionality resulted in a significantly lower RMSE with all other parameters held equal, as shown in Table 6.

Table 6: Random forest bootstrapping (# estimators = 100).

Bootstrapping?	RMSE	Runtime
Yes	14.738	88.77 s
No	15.426	56.04 s

4 Conclusions

This project involved the use of machine learning algorithms to build a model for prediction of users' music ratings. A variety of algorithms were considered; the best-performing model involved use of a bootstrapped random forest regressor. Using 200 estimators with a max depth of 40 and log2(n) max features, a root mean squared error of 14.712 was obtained, measured via three-fold cross-validation. Given that the winning competition error was 13.246, this model represents a relatively successful implementation of the random forest regressor to the problem.