MUSES: Improving Performance of Music Genre Classification with Ensemble Models

Hai Nguyen
College of Computer and Information Sciences, Northeastern University
nguyen.hai@husky.neu.edu

Abstract

Music genre classification is essential for music information retrieval systems. This process is widely applied to high-demand tasks such as song recommendation and improving the performance for the task is still an open issue. Motivated by that, we study how different classification models perform in music genre recognition task with Free Music Archive dataset, focusing on ensemble models with two representatives XGBoost and ExtraTrees. The results show that using ensemble models has some improvements over other commonly used models. Furthermore, we studied the impact of data pre-processing techniques on classification performance: Principal Component Analysis (PCA) and Synthetic Minority Over-sampling (SMOTE). We found that using PCA degrades the performance, while using SMOTE improves the balanced accuracy for most of our models.

1 Introduction

Genre is a type of musical descriptor that human defined in order to categorize music pieces which have different acoustic characteristics. As music is an essential part of life, nowadays there are demands of organizing large music collections as well as suggesting appropriate songs based on user preferences or specific occasions, in which genre is especially useful. A popular application is song recommendation system, which bases on genres to provide song suggestions for users. Because hand-labeling an enormous number of songs is time-consuming, automatically recognizing genres based on audio features is important and beneficial, especially for music streaming services such as Spotify or Apple Music [1].

Improving the accuracy for music genre classification has become an intriguing problem, having attracted many attentions from researchers and machine learning developers in the recent years. Tzanetakis and Cook analyzed useful musical features and used Gaussian mixture model and k-nearest neighbors to classify genres [2]. Meng et al [3] combined feature vectors in a time window to get a new feature vector which capture temporal information, and classified the data by methods such as Generalized Linear Model classifier. Also, a challenge on Musical Genre Recognition was held in 2018 International Web Conference [4].

For the purpose of studying this interesting problem, we introduce MUSES project, which stands for **MU**sic Genre Cla**S**sification with **E**n**S**emble Models. We investigated some commonly used supervised learning models: Linear Regression, Decision Tree, Support Vector Machine, Neural Networks, beside focusing on ensemble methods. More specifically, we studied two popular ensemble methods XGBoost [5] and ExtraTrees [6] which have achieved state-of-the-art results in recent classification tasks such as job recommendation [7] and in machine learning competitions on Kaggle [8]. Moreover, we analyzed the impact of two data processing techniques, namely Principal Component Analysis [9] and Synthetic Minority Over-sampling [10], on genre prediction.

2 Technical Approach

2.1 Baseline models

We inspected some commonly used popular supervised learning models and used them as baseline models to compare with our focused ensemble models.

(A) *Logistic Regression*
Logistic Regression is a discriminative classification method which models and optimizes the conditional probability $p(y \mid x)$. The optimization problem is finding optimal parameters $\hat{\theta}$ to minimize the loss function (a common choice is cross-entropy loss function). To prevent overfitting, an additional penalty (or regularization) for θ is often incorporated in the loss function: L2-norm in Ridge Regression, L1-norm in LASSO (least absolute shrinkage and selection operator). For multi-class classification problem, each class can have one separated classifier and it is called One-vs-Rest Classifier.

(B) *Decision Tree*
Decision Tree consists of a root node, decision nodes (correspond to features) and leaf nodes (correspond to labels), combined into a classification model in tree structure. To predict the label of a sample, we start from the root node (most distunguishable feature), then based on the comparison between the value of data feature and the root node or a decision node, we follow the branches until reaching a leaf node, which is the predicted data label. To select which features should be used as the root or decision nodes, a criterion such as Gini Index or Entropy is used.

(C) *Support Vector Machine*
Support Vector Machine (SVM) classifies data by constructing decision boundaries that separate data in different classes to minimize the max-margin of all data points. In real cases, high-dimensional data is often not linearly separable. Therefore, it is natural to use SVM with slack variables (Soft-margin SVM), which can be further derived to Dual form for more efficient computation (especially when dataset is small and the number of features is larger than the number of data points). We can also use kernels to reduce complexity of multiplication when basis expansion is used. For MUSES, we incorporated SVM with Linear and RBF kernels. With the goal of multi-class classification, multiple decision boundaries will be created, one for each class to be separated from other classes. (One-vs-Rest SVM)

(D) *Artificial Neural Network*
Artificial Neural Network (ANN) is a structure of neurons where outputs of neurons in previous layers are linearly combined to be used as inputs for neurons in next layers. It consists of an input layer, an output layer and several hidden layers. Each hidden layer applies an activation function to its inputs to get the outputs and passes them to next layer. Figure 1 shows a structure of an ANN with two hidden layers. ANN uses forward-propagation to generate output and backward-propagation with gradient descent to update its parameters. To prevent overfitting, L2-regularization is often used along with the loss function.

2.2 Ensemble models

(A) *XGBoost*
Extreme Gradient Boosting (or XGBoost) [5] is a boosting method in which models are trained sequentially in a way that each model tries to correct its predecessor by optimizing the previous *residual error*. This could result in a much stronger model built from numerous "weak" base models.
XGBoost can be used with linear boosting or tree boosting base model. In this project we used tree boosting as it is a much more popular option for XGBoost in classification tasks. XGBoost has many advantages compared to conventional gradient boosting. One of which is that it constructs trees with full depth first, then eliminates bad node splits with small loss reduction, similar to tree pruning technique for tree-based models. Also, XGBoost incorporates regularization in the cost function, which gradient boosting does not have, thus makes the model robust to overfitting. Moreover, XGBoost also takes advantages of parallel computing to speed up computation for multi-core computers. With those advancements, XGBoost has achieved much success in Machine Learning tasks [11].
To push the model to its limit, we considered tuning the hyper-parameters extensively. Some important hyper-parameters that we chose are (along with their notations in *xgboost* official library [12]):

- Number of boosting trees (*n_estimators*)
- Maximum tree depth (*max_depth*)
- Minimum sum of weights required for child nodes (*min_child_weight*)
- Minimum loss reduction required to make a split (*gamma*)
- L2-regularization (*reg_lambda*)
- The fraction of data points to be randomly sampled for each tree (*subsample*)
- The fraction of features to be randomly sampled for each tree (*colsample_bytree*)

(B) *ExtraTrees*
We also investigated another well-known ensemble method, namely Extremely Randomized Trees (or

ExtraTrees) [6]. This is a bagging method in which multiple decision trees are constructed and trained in parallel. The prediction result is the average prediction results of all trees. To select features for splitting, the threshold (or cut point) for each feature is chosen randomly within its value range. Then based on a predefined criterion, the algorithm chooses the pair of feature with corresponding threshold which has the best score as the splitting node.
Two important parameters which control the complexity of *ExtraTrees* are the number of decision trees and the maximum depth of trees. We will study the impact of these parameters on performance of the method through the validation process.

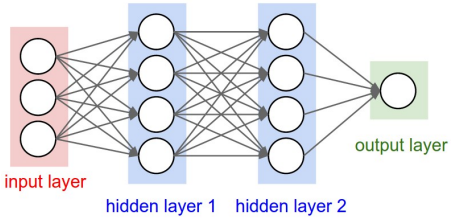


Figure 1: Example of neural network model with 2 hidden layers. Source [13]

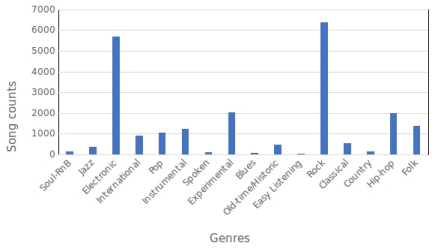


Figure 2: Distribution of genres in FMA training set

2.3 Data processing techniques

(A) *Principal Component Analysis (PCA)*
PCA algorithm [9] tries to project the data in high-dimensional space to the best-fitting affine subspace of smaller dimension. The low-dimensional projections $\hat{z}$ of data as well as optimal parameters $\hat{\mu}$, $\hat{U}$ of the subspace are obtained by optimizing the generalized Lagrange function. PCA is used the most for dimensionality reduction. As the data is high-dimensional, some features might be correlated to others and redundant to be considered for training. Furthermore, too many features could make our models complicated, leading to overfitting. We will apply PCA on both training and test data to see if our models with optimal hyper-parameters could have some improvements on classification performance.

(B) *Synthetic Minority Over-sampling (SMOTE)*
SMOTE [10] is an over-sampling technique that generates synthetic samples in neighboring space between existing samples for minority classes. The algorithm finds k-nearest neighbors for each sample in minority classes, then creates new samples between the original sample and each of its neighbor in the feature space. More specifically, if we represent the original data point as a feature vector $\vec{o}$ and one of its neighbors as a feature vector $\vec{o_n}$, then the new generated synthetic sample will be represented as a feature vector $o_s\vec{y}n$ as follows:

$$o_s\vec{y}n = \vec{o} + (\vec{o_n} - \vec{o}).c$$

with c is a random number between $[0,\ 1]$
As it will be discussed later, the dataset can be highly imbalanced which results in biased models after training phase. Biased models cannot predict data in minority classes well. Some possible ways to address this problem are over-sampling minority classes, or under-sampling majority classes. Under-sampling can cause loss of information and reduce the size of dataset significantly, which is undesirable because of possible overfitting. For over-sampling, one approach is to replicate the original samples multiple times, which would also result in overfitting because as the models learn specific instance in the training set too many times, the decision boundary between classes will become very tight. Therefore, our intuition is to use SMOTE to generate more similar samples, instead of replicating samples. The effect of SMOTE on genre recognition will also be investigated in later sections.

3 Experimental Results

3.1 Experimental settings

For the scope of this project, a subset of 25,000 tracks in the Free Music Archive (FMA) dataset [14] was used. It was splitted in advance into training set and test set with the ratio 6:1. Each sample in the

dataset contains 518 statistical audio features and a genre label. The audio features are used commonly in audio recognition tasks, such as Mel-Frequency Cepstral Coefficients (MFCC) [15] or Chroma [16]. Furthermore, they are all numerical, thus appropriate for wide range of learning models.

25,000 tracks in the dataset are categorized into 16 different genres. Figure 2 shows the distribution of genres in the training set. It is obvious that the classes are highly imbalanced i.e. some classes have much more (or fewer) samples than others. For examples, Electronics and Rock genres are among the most popular classes, each having over 5,000 samples. Meanwhile, classes such as Easy Listening only have around 20 samples. Imbalanced dataset can make our model biased in the training phase and affect the predictions of classes in the test phase, as we will discuss later.

Pre-processing: Before using the dataset for training and testing, we performed feature scaling on all 518 features of the data. The reason is that using some specific features with large values would lead them to be the main predictors in our models, which is not desirable. The standardization scaling was used, which takes a feature value subtracted by the mean then divided by the standard deviation. Also, the genre labels were encoded into integer values ranging from 0 to 15 to be compatible with the output of classification model.

Evaluation metrics: To evaluate the performance of models in both training phase (to select the best hyper-parameters) and in test phase (to compare the models), accuracy is not a good metric in the case of imbalanced dataset. The reason is that a model which returns the most common classes all the time can still get a very high accuracy, while in fact it predicts samples of other classes incorrectly. Therefore, we also use balanced accuracy as another metric for evaluating performance. It is calculated as the average of *recalls* for all the classes. Recall is a fraction of true positives over sum of true positive and false negatives, indicating how accurate a model can predict for each class.

The classification methods are implemented in Python with the help of *scikit-learn* [17] and *xgboost* [12] packages. Additionally, we used *imbalanced-learn* [18] package for convenient SMOTE implementation.

3.2 Hyper-parameter investigation

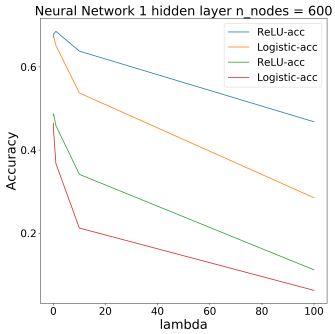
Model	**Hyper-parameters & corresponding values**
Logistic Regression	Regularization: L1, $\lambda : 0.001$
Decision Tree	Maximum tree depth: 21
SVM with Linear kernel	$C : 0.1$
SVM with RBF kernel	$C : 500$
Neural Network 1 hidden layer (600 nodes)	Activation: ReLU, $\lambda(L2) : 0.1$
Neural Network 2 hidden layers	Number of nodes in hidden layers: (600, 100) Activation: ReLU, $\lambda(L2) : 0.01$
ExtraTrees	Number of trees: 3000, Maximum tree depth: 17
XGBoost	n_estimators: 300, max_depth: 9, min_child_weight: 10 gamma: 0.3, subsample: 0.7, colsample_bytree: 0.8, reg_lambda: 1

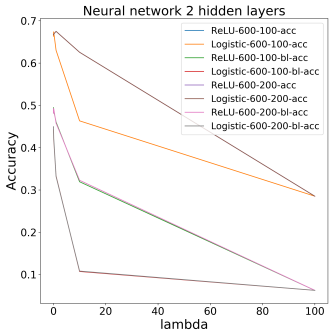
Table 1: Tuned hyper-parameters for the models and their best values from cross-validation

Hyper-parameters control how an algorithm works. For example, regularization helps to deal with overfitting or maximum depth decides the complexity of a tree-based model. They need to be predefined and unchanged during the training process. Therefore, to obtain a desirable model, tuning these hyper-parameters as well as observing how they affect the performance is essential for any Machine Learning tasks.

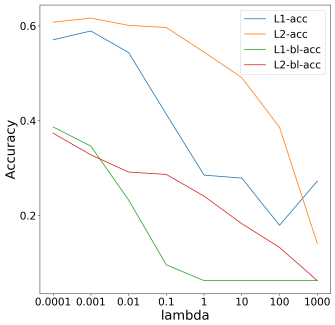
To investigate how different values of hyper-parameters affect classification performance as well as to get the best ones for each model, we used k-fold cross-validation with $k=5$ using *GridSearchCV* of *scikit-learn*. The evaluation result is the average of test results over all folds. Table 1 displays the hyper-parameters that we chose for tuning as well as their best values. Figure 3 illustrates how those hyper-parameters affect accuracy and balanced accuracy in the validation process. Note that for XGBoost, many hyper-parameters were used for tuning to give the best settings for the model, detailed results in parameter tuning for XGBoost are excluded in this report.

It is obvious that when using SVM with RBF kernel, larger values of regularization C provide better results both in terms of accuracy (from lower than 30% with C smaller than 0.01 to approximately 70% with C larger than 10) and balanced accuracy (similar pattern, from lower than 10% to higher than 50% in the same range of C). With Linear kernel the accuracy peaks at $C = 0.01$ ($\approx 67\%$) while the balanced accuracy peaks at $C = 0.1$ ($\approx 46\%$). There is a big gap between accuracy and balanced accuracy curves of Decision Tree. Furthermore, we also see opposite patterns in the two curves: As the maximum tree depth increases, balanced accuracy improves whereas accuracy degrades. In the case of ExtraTrees, we see great improvements when increasing the maximum depth of randomized trees. Meanwhile, increasing the number of trees does not have significant effect when it raises above 1000.

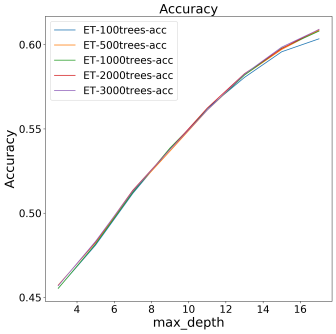


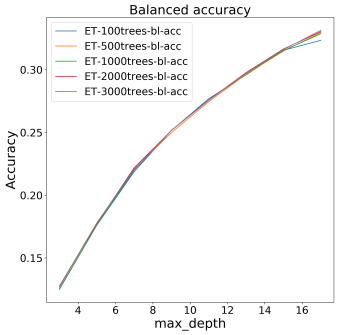


(a) Neural Network 1 hidden layer (left) and 2 hidden layers (right)

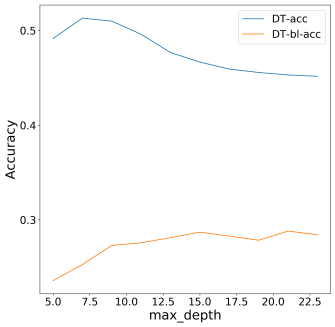


(b) Logistic Regression

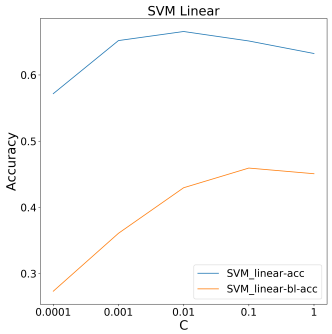


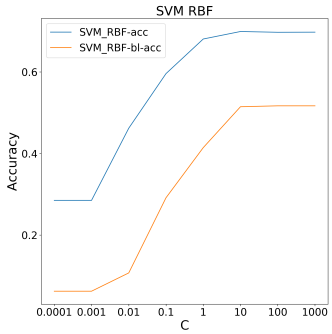


(c) ExtraTrees: accuracy (left) and balanced accuracy (right)



(d) Decision Tree





(e) SVM Linear (left) and SVM RBF (right)

Figure 3: Effect of hyper-parameters to performance of corresponding models in validation phase.

An interesting pattern is that small regularization in Logistic Regression and Neural Network generalizes better both in terms of accuracy and balanced accuracy. This might be caused by two properties: First, the nature of the data in the training and cross-validation sets does not make the regularization

useful for improving generalization in the cases with Logistic Regression and Neural Network; second, the models that we used for tuning are already simple and they do not overfit, so large regularization could not help. One way to inspect this behavior deeper in future work is to make our Neural Network model more complex i.e. adding more hidden layers with more nodes.

3.3 Prediction results

Model	Accuracy (%)	Balanced Accuracy(%)
Logistic Regression	53.75	33
Decision Tree	42.79	23.41
SVM Linear	61.02	38.85
SVM RBF	62.88	40.12
Neural Network 1 hidden layer	61.68	40.36
Neural Network 2 hidden layers	59.46	38.85
ExtraTrees	57.68	28.05
XGBoost	64.32	38.74

Table 2: Accuracy and Balanced accuracy on the test set

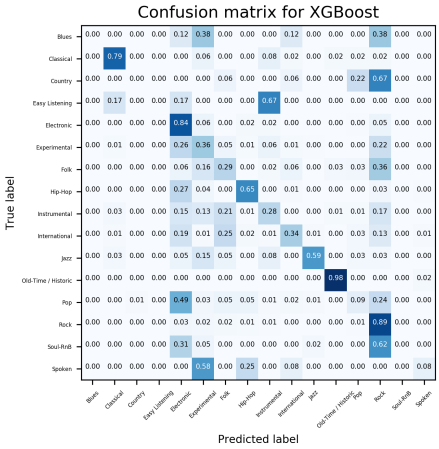


Figure 4: Confusion matrix of XGBoost prediction on the test set

After getting the best hyper-parameters, we train our models with the whole training set and use the trained models to make prediction on data in the test set. Table 2 shows the results of prediction of all models. We can see that XGBoost has the highest accuracy, but lower balanced accuracy than some other models such as SVM RBF. Meanwhile, ExtraTrees even performs worse than SVM and Neural Networks. Overall, our trained models suffer from the impact of imbalanced dataset, resulting in very low balanced accuracy, which is 20% lower compared to the accuracy for all models.

The impact of imbalanced dataset can also be observed in Figure 4 which depicts the confusion matrix of prediction of XGBoost. We can see that in some minority classes such as Easy Listening or Blues, the accuracy is zero, while in most popular classes such as Rock and Electronic the accuracy is higher than 80%. Imbalanced classes in dataset make our model biased and inclined to predicting label of popular classes which results in low balanced accuracy.

3.4 Effects of PCA and SMOTE

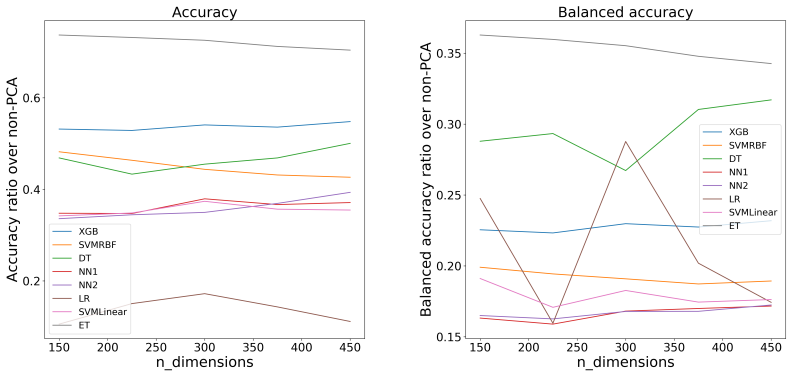


Figure 5: Impact of PCA on accuracy (left) and balanced accuracy (right)

We applied PCA on both training and test set with different values of *n_dimensions*, then used the transformed data to train and evaluate our models. Figure 5 illustrates prediction results of our model with data after PCA. To compare with non-PCA case, we calculated the ratio of new prediction accuracy/balanced accuracy with the old ones that we obtained before with non-PCA dataset. It is clear that the performance of our models degrades significantly when using PCA on dataset, especially the balanced accuracy which reduced more than three times. One thing to note is that ExtraTrees now performs the best among all models. Overall, we can conclude that using PCA to reduce the number of dimensions does not help to improve performance of our models in this task. Instead, we would need some other approaches such as feature engineering to carefully inspect and select the features that effectively represent the music audio data.

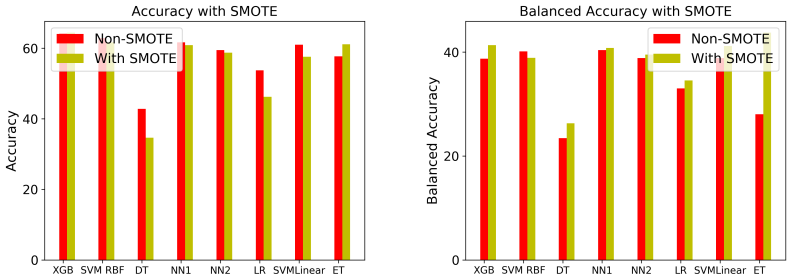


Figure 6: Impact of SMOTE on accuracy (left) and balanced accuracy (right)

SMOTE was used in the training set to obtain a balanced dataset. After that, we took the balanced dataset to train our models and then tested the trained models on the original test set. Figure 6 depicts how SMOTE affects the performance of our models. It is easy to see that using SMOTE boosts the test balanced accuracy for almost all the models (except SVM RBF), especially for the two ensemble models: approximately 3% for XGBoost and more than 10% for ExtraTrees. Interestingly, only the accuracy for two ensemble models do not decline with data using SMOTE, compared to non-SMOTE case. These findings mark an advantage of ensemble methods over other methods in this classification task.

We have learned that better performance can be achieved if the impact of the imbalanced dataset is alleviated. One question we should consider to solve for future work is "Are the classes mutually inclusive?". For example, can some genres such as Easy Listening be grouped into other genres? As the number of instances in minority classes become larger, the dataset will be more balanced. Because the data were labeled beforehand in a fashion that only one label is assigned for one sample, in order to relabel the samples, we need to obtain and analyze the exact information for every track which belongs to minority classes in the data.

4 Conclusion

Music genre classification is a problem of recognizing song genre based on audio characteristics such as rhythmic structure or instrument sound. To address this problem, in MUSES project, we investigate how different classification models perform in recognizing genres, with more focus on two state-of-the-art ensemble models: XGBoost and ExtraTrees. The experimental results show that XGBoost has the best test accuracy, and that XGBoost and ExtraTrees can be used with Synthetic Minority Over-sampling (SMOTE) to effectively deal with the problem of imbalanced dataset. We also found that applying PCA to reduce the number of dimensions of data make the test performance deteriorated and thus it should not be used for this task. Possible improvements for future work are feature engineering and relabeling process on the data, as the former helps to obtain effective and high-quality data features while the latter can reduce the impact of imbalanced classes. Another direction is using Convolution Neural Networks to automatically extract useful features from raw audio data and then we can use some strong classification models on those features such as XGBoost or Deep Neural Networks.

References

[1] M. Johnston. How spotify discovers the genres of tomorrow. https://artists.spotify.com/blog/how-spotify-discovers-the-genres-of-tomorrow.

[2] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10:293–302, 2002.

[3] A. Meng, P. Ahrendt, J. Larsen, and L. K. Hansen. Temporal feature integration for music genre classification. *IEEE Transactions on Audio, Speech, and Language Processing*, 15:1654–1664, 2007.

[4] Www 2018 challenge: Learning to recognize musical genre. https://www.crowdai.org/challenges/www-2018-challenge-learning-to-recognize-musical-genre.

[5] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, New York, NY, USA, 2016.

[6] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning Journal*, 63:3–42, 2006.

[7] A. Pacuk, P. Sankowski, K. Wegrzycki, A. Witkowski, and P. Wygocki. Recsys challenge 2016: Job recommendations based on preselection of offers and gradient boosting. In *Proceedings of the Recommender Systems Challenge*.

[8] Kaggle. https://www.kaggle.com.

[9] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2:559–572, 1901.

[10] N.V. Chawla, K.W. Bowyer, L.O. Hall, and W.P. Kegelmeyer. Smote: Synthetic minority oversampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[11] List of machine learning challenge won by xgboost. https://github.com/dmlc/xgboost/tree/master/demo#machine-learning-challenge-winning-solutions.

[12] xgboost package. https://xgboost.readthedocs.io/en/latest/index.html.

[13] Cs231n. http://cs231n.stanford.edu.

[14] Fma: Free music archive dataset. https://github.com/mdeff/fma.

[15] L. R. Rabiner and B. H. Juang. *Fundamentals of speech recognition*. 1993.

[16] M. Goto. A chorus section detection method for musical audio signals and its application to a music listening station. *IEEE Transactions on Audio, Speech and Language Processing*, 14:17831794, 2006.

[17] scikit-learn package. https://scikit-learn.org/stable/.

[18] imbalanced-learn package. https://imbalanced-learn.org/en/stable/index.html.