Ensemble methods for music genre prediction

December 10, 2017

This is the final report for the data analysis project part of the course Machine Learning Basic Principles. The task of the project is to correctly predict music genres from the given dataset. This report explores the given data and provides analysis on it, addressing issues such as class imbalance and non-normal feature distributions. In addition, a full model development and evaluation is performed and the steps taken are explained and discussed. The focus of our project was to evaluate ensemble methods, however, a simple Support Vector Classifier (SVC) ended up performing extremely well with proper data preprocessing. Ensemble methods are challenging to tune properly, also interpreting their results is more complicated than for simple models.

1 Introduction

The goal of this machine learning project is to evaluate different machine learning algorithms in order to predict a music genre using features generated from the raw audio signal. Recommendation engines are currently extremely popular; many large companies e.g Netflix and Spotify use machine learning algorithms to recommend movies and songs to customers. Even though this project is not strictly speaking a recommendation engine, making automatic genre predictions is part of that problem.

Music genre recognition is highly studied area in machine learning and feature engineering, However most research papers focus mainly on feature extraction from raw audio signals. From machine learning algorithmic point of view, *Support Vector Classifier (SVC)* has been proved to be a good classifier for genre prediction using proper feature engineering [1]. There is an interesting survey paper, which covers different approaches to feature engineering and discusses how different algorithms perform with predicting music genre [2].

From this project we hope to get hands-on experience on approaching real world machine learning problems. The problems we might face are very different from the isolated problems of the weekly assignments in our machine learning major. These expected problems include the need for data preprocessing as well as the problems regarding the accurate evaluation of the model performance. Also, this project is a perfect opportunity to deepen our understanding of the basic machine learning models: observing the performance of traditional models on this kind of data and figuring out the requirements for improving the performance of these models. Also, it is interesting to explore the state of the art solutions to other Kaggle competitions and experiment with the models used in them.

In the next section, we will perform analysis on the data and discuss the key aspects that need to be taken into account during the modeling. In the next section we describe our methods and experiments taken. Finally the obtained results are presented and the results as well as the whole project are discussed.

2 Data analysis

The features represent different descriptive statistics, such as mean, standard deviation or kurtosis for distributions of various sound engineered features. Therefore, the distributions are expected to vary greatly between the features. For example, unconstrained mean values are expected to follow normal distribution due to central limit theorem, standard deviation on the other hand is limited to positive values and follows χ^2-distribution. In addition, some of the features in the data set are uninformative (value 1 for all samples). Four distinct types of feature distributions of the data are visualized in Figure 2.

The normality of a feature distribution is crucial for most linear models as they assume symmetric (gaus-

Figure 1: *Class distribution of the data: the data is clearly extremely imbalanced.*

sian) error terms. Algortihms such as SVC or *K-Nearest Neighbors* (KNN) are sensitive to the normality of the data. For linear models, performing normal transformation by taking the logarithm or using Box-Cox transformation improves the results greatly. For non-linear models, such as *Random Forest (RF)*, *Neural Network (NN)*, neither distribution nor the scale of the data has impact on the predictive performance (although scaling does improve rate of convergence for NN).

The classes in the dataset are highly imbalanced (Figure 1): approximately half of the data consist of class 1 (Pop-Rock) as opposed to only 2% representation of class 10 (Blues). Class imbalance needs to be accounted for when creating the train-test splits in order to preserve the class distribution.

There are 258 features in the dataset in total. This not a terribly large data set, however, based on the visualized distributions, some of these features are most likely redundant and therefore a data preprocessing is performed beforehand. We also spent some time learning about the feature generation process behind the data [5].

3 Methods and experiments

This sections covers the full model construction process, from data preprocessing to model construction. The workflow is visualized in Figure 3.

3.1 Data preprocessing

The data preprocessing consisted of three sections: feature normalization, feature extraction and lastly feature engineering. Feature normalization means scaling the data along the dimensions, in feature extraction the data is transformed into new feature space (possibly reducing dimensions). The objective of feature engineering is to improve model performance by creating completely new features.

Feature normalization

Normalizing the data using a linear transformation is a standard step taken, especially when using linear models. We experimented with both *standard scaling* (reduce mean and divide by standard deviation) as well as *min-max scaling* (subtract smallest value and divide by the distribution length). The difference between these two was negligible and we ended up using standard scaling method for most of our submissions.

In addition to the scaling of the features, we also performed a non-linear *Box-Cox transformation* [3] on the data (Figure 4). The objective of this step is to normalize the data distribution so that linear models can be used for it. Without normalization, the assumptions on normally distributed error terms does not apply and the performance of the models relying on it is impaired. Another option would've been to simply take a logarithm of the data, which achieves similar effect.

Feature extraction

Based on the high amount of features and information acquired from the data exploration, we reasoned it would be necessary to extract useful features and reduce dimensionality. The first method tried was *Principal Component Analysis (PCA)*, for which the correlation matrix approximation was not needed as our data was already normalized. We tried different number of principal components, anywhere between 50-130. This amount of features covered between 50% and 80% of the data variance. As expected, linear models such as SVC with Gaussian kernel and KNN benefit from the dimensionality reduction. On the other hand, dimensionality reduction did not notably improve non-linear and properly regularized models such as RF or NN.

Second feature extraction method used was *Linear Discriminant Analysis (LDA)*. This proved to be substantially better than PCA in all the experiments.

Even though dimensionality reduction increase the accuracy in simple models, extensively reducing complexity might (and most probably will) create an upper bound for the accuracy that our model can achieve. Therefore when we started to experiment with ensemble methods, we did not use LDA preprocessing with every single model. For instance, LDA was used with KNN and SVC but was not used with random forests and multilayer perceptrons that should be able to handle more complex data easier.

Feature engineering

Feature engineering is the process of adding new features to the data, often using additional, heuristic knowledge. However, it also makes sense to compose new features from the existing features without necessarily adding new knowledge. As an example, if a

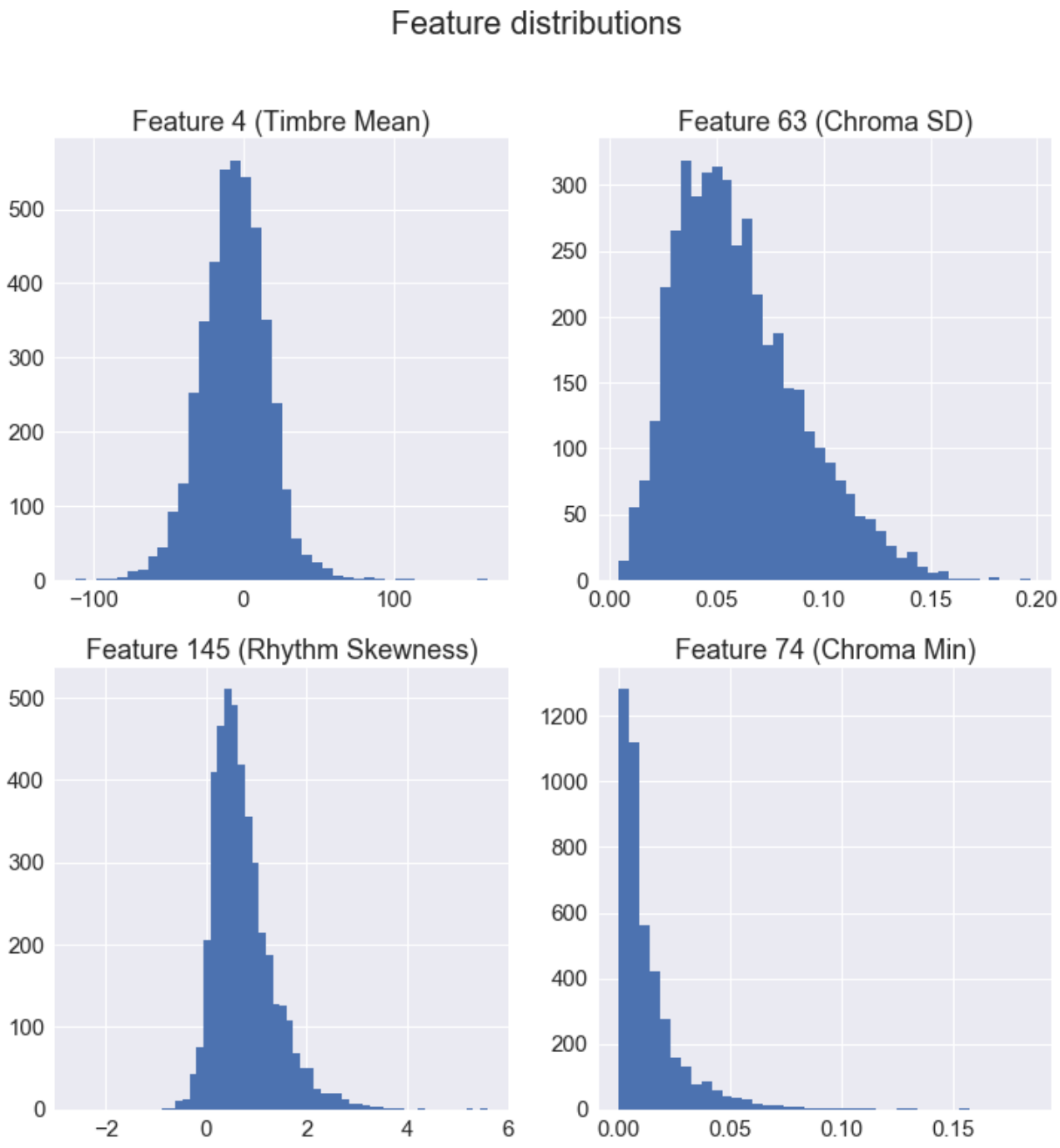


Figure 2: *Feature distributions: features follow various types of normal and non-normal distributions.*

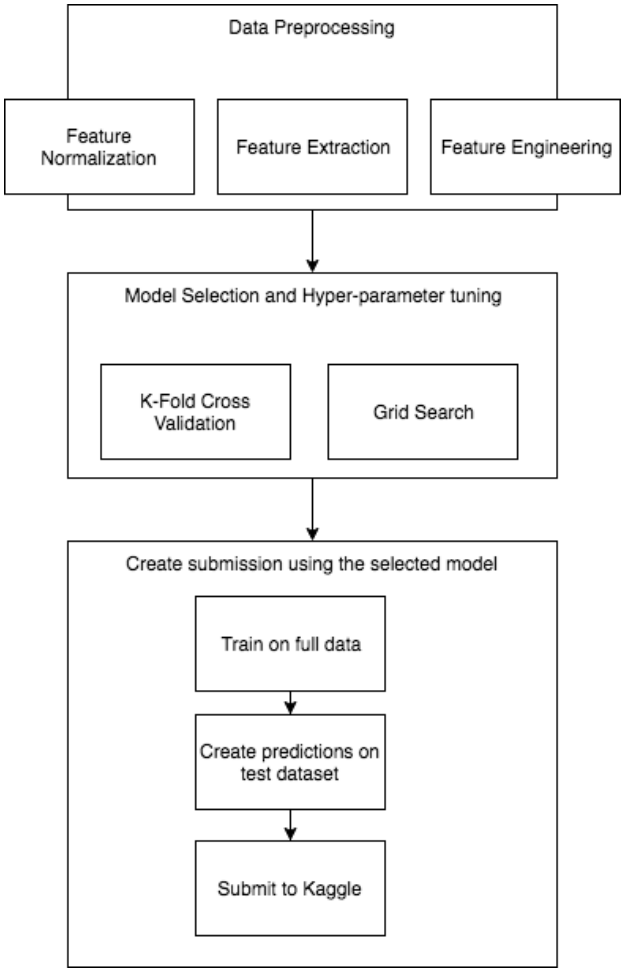


Figure 3: *Experimenting process*

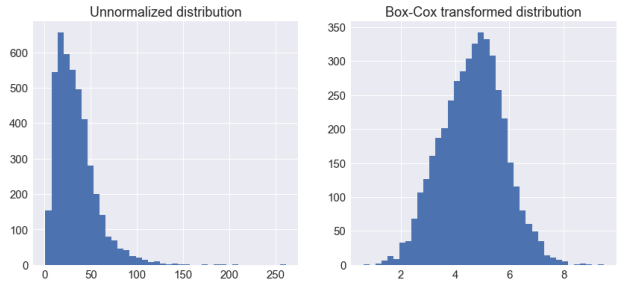


Figure 4: *Data distribution before and after applying Box-Cox transformation*

product of two features has a significant meaning, models are often not capable of interpreting this relation from the raw features and thus adding the feature explicitly makes sense. We added one such feature: the cyclic lagged correlation between chroma mean values. The intuition behind this feature is that it could encapsulate the scales used in the music and therefore provide more dimensions to distinguish the genres.

3.2 Model experiments

The model experiments section addresses the actual model selection and experiment process. The first subsection describes our model evaluation and selection process. After the evaluation section, our experiments have been divided into two sections: standard models and ensemble models. We started the project by experimenting with standard models, such as SVC, KNN or RF, along with data preprocessing. After fine tuning these models for a while, we switched our focus on ensemble models.

Evaluation methods

K-fold cross validation was used for comparing and measuring model performances. Fold amounts 5 and 10 were used. We also used *grid search* to search for best selections of hyper parameters and quickly experiment with various classifiers and preprocessing choices. Later on in the project we started using repeated cross validation in order to increase our confidence on the predictions. With the repeated experiments, both 5- and 10-folds performed quite equally.

The class imbalance caused problems in our initial setup where we used standard randomized cross validation. Due to this imbalance, the smaller classes were not sufficiently sampled for some of the folds which caused extremely unreliable estimates for the accuracy. This issue was fixed by using stratified k-fold cross validation, which preserves the relative class frequencies within folds.

Standard models

As can be seen in Figure 8, we tried multiple different classifiers with different preprocessing steps and compared the results. This step was highly mechanical: a huge grid was built containing most of the standard classifiers, preprocessing steps as well as a range of hyper parameter choices. Best individual accuracies were achieved with Box-Cox normalized features, LDA feature extraction and SVC with gaussian kernel.

We also tried a gradient boosting model called *XGBoost* [8] as an individual model. XGBoost is a nonlinear model that ensembles multiple weak classifiers using boosting. XGBoost is a highly performant often used in highly performing Kaggle submissions [11], [10]. Fine tuning XGBoost ended up being challenging due to high amount of hyper parameters, an issue

we'll address more in the Discussion and Conclusions sections.

Ensemble methods

The idea behind ensemble methods is to train various *base classifiers*, such as KNN, RF or NN, that each capture different aspect of the data and have different decision boundaries. The predictions of these models are then fed to another, often non-linear, *meta classifier* (often XGBoost or NN) that outputs the final prediction, using the so called meta predictions (and possibly additional raw features) as its input. Figure 5 illustrates one such *stacking* [14] setup.

For the ensemble methods we found multiple strategies [13], [7], [14], out of which we tried two different strategies. The simpler one was to use 80% of the data for training the base classifiers and the remaining 20% for the meta classifier. The advantages of this approach is that there is no *information leak*, i.e. the data is never used for training and evaluation at the same time.

In the second approach, the data is split in k folds and the base classifiers are each trained on $k-1$ folds, and meta predictions are created on the last fold. This is repeated k times in order to get out of sample predictions for the whole dataset. This meta prediction dataset is the used to train the meta classifier. same time on the same fold and then each of the first layer models made a prediction for the test fold data. There is a possible information leak in this approach, however it is still more robust than the simpler approach as the combiner gets more data to be used for training. The information leak is supposedly fairly negligible [6] as the base predictions are still made outside of the training set, even though the meta classifier has implicit knowledge on the labels as the whole data is reused on both layers.

When choosing the base classifiers, we tried to find sufficiently performing classifiers that correlate as little as possible with each other [10]. It makes sense to find models that predict different labels well even when their individual accuracy is not necessarily the best. As an example, we found that LDA classifier had good accuracy of 65% but was poor at predicting labels that were infrequent (label 10). However *Naive Bayes classifier* (NB) with 80% PCA had a worse overall accuracy of only 54% but was better at predicting correctly infrequent labels. Intuitively, properly combining the results from such classifiers should result in improved accuracy.

Initially we used NN as the meta classifier, however, we tried multiple variations of this ensemble method for instance by adding NN to base classifier as well and changed our meta classifier to XGBoost which has achieved good results in the literature [11].

4 Results

Overall, the best result was accomplished with SVC using gaussian kernel. During the data preprocessing, features 84-95 were dropped due to lack of information after which the data was normalized using Box-Cox transformation. After normalization, the features were extracted using LDA and classified using SVC with gaussian kernel. On the leaderboard, the accuracy for this model was **0.66778** and logloss **0.17230**. This particular model performed locally very similarly as in the Kaggle (average cross validated, local accuracy for this exact configuration was 0.66688). The confusion matrices for this model can be seen in Figure 6.

Our best stacking ensemble consisted did not manage to outperform the simple SVC model. The highest accuracy for our ensemble model on the leaderboard was **0.65265**. The meta classifiers consisted of KNN (multiple models with varying k), NB, NN, RF as well as Logistic Regression (LR). As the meta classifier we used XGBoost. The data normalization varied across the classifier, non-linear models (RF, NN) used raw data, for linear models Box-Cox and feature extraction was used. Average cross-validated accuracies were above 67% on local tests, however, on the Kaggle submission the accuracy fell well below this. We did not submit this to the logloss competition.

5 Discussion and Conclusions

During our research, we saw that ensemble methods have performed extremely well on many Kaggle classification competitions [10], [11], [12] (similar to this music classification task). We were eager to get our stacking ensemble approach to work and were disappointed when the accuracies were constantly better in our local tests compared to our Kaggle returns. On local evaluation we got accuracies above 67%, but when returned to Kaggle, our results were below 66%. Building these models is fairly complicated compared to simpler models and therefore it is likely our ensemble methods overfit the data heavily due to limited regularization and insufficient hyper parameter tuning. XGBoosting already requires heavy hyper parameter tuning as it is. Another possibility is that there are some errors still in our stacking code.

During the project, we saw that a comprehensive data exploration and feature preprocessing substantially increases the performance of especially linear models. We got a good hands on experience on comparing different models as well as exploring and understanding the features, their distributions and how to prepare them. We also observed some unexpected issues caused by class imbalance in the cross validation sampling that we were able to solve. Perhaps the biggest revelation was that even simple models can produce extremely good performance with properly normalized data. It was surprising that a simple LDA

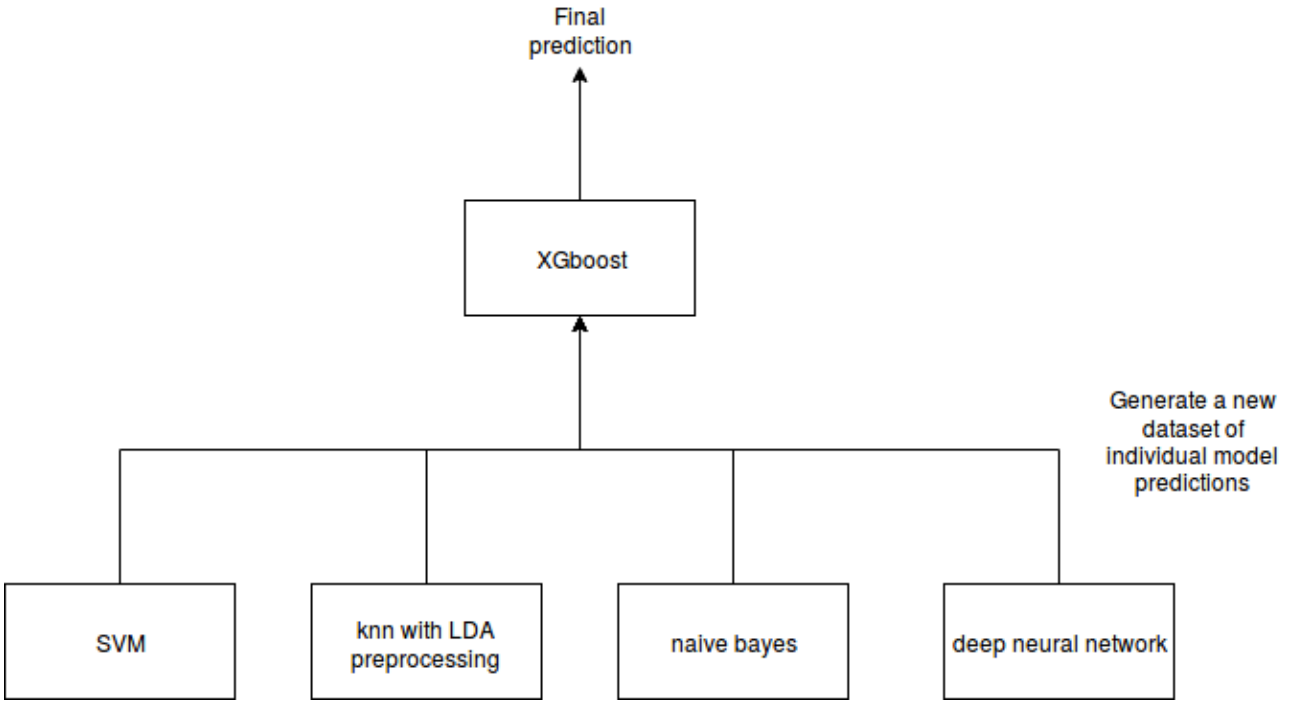
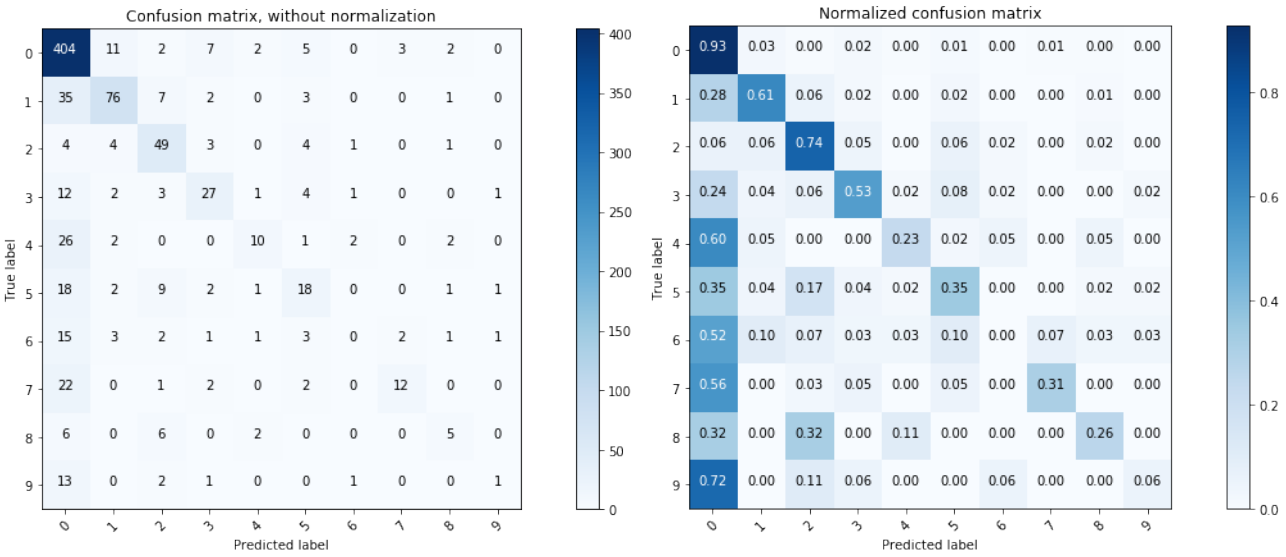


Figure 5: *Structure of one of our ensemble experiments*



(a) *Absolute frequencies* **(b)** *Proportion of true class*

Figure 6: *Confusion matrices for our best submission.*

accompanied with SVC performed so well and ended up reaching high ranks in the leaderboard. We were expecting more complex models to beat it easily. The amount of data is fairly slow and the data consists of processed features (i.e., no raw signals), which makes it possible for the simple models to perform so well. Overall, the project did answer most of our questions regarding the data analysis practices.

When investigating and comparing the two performance measures, classification accuracy and logloss, we found that they both provide a different approach to the same problem. Accuracy measures the conclusive prediction value where as logloss uses the raw predicted probabilities of class belonging to measure the accuracy of the classification. Logloss more heavily penalizes models that are more confident about false prediction than "normal" accuracy metric. The class imbalance should be accounted for when the performance is measured using logloss performance metric. We did not address this properly in our logloss submission and thus the performance was substantially weaker compared to our accuracy submission. For the accuracy measure, the classes do not need to be balanced [4] as long as the test set follows the same relative frequencies. In fact, balancing the dataset (using weighting, oversampling or other methods) overemphasises infrequent classes reducing the overall accuracy.

For future improvements, we would further explore the ensemble methods. When defined, trained, fine-tuned and regularized properly, they should outperform the simpler models in this competition as well. Another approach would be to dig deeper into the processes behind the actual features. Understanding the features and engineering new features is also very likely to improve the results substantially.

References

[1] Li, Tao, Mitsunori Ogihara, and Qi Li. *A comparative study on content-based music genre classification.*. Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 2003

[2] Scaringella, Nicolas, Giorgio Zoia, and Daniel Mlynek. *Automatic genre classification of music content: a survey.*. IEEE Signal Processing Magazine 23.2 (2006): 133-141

[3] Wikipedia. *Power transformation*, https://en.wikipedia.org/wiki/Power_transform#Box.E2.80.93Cox_transformation

[4] Stack Exchange. *When should I balance classes in a training data set?*, https://stats.stackexchange.com/questions/227088/when-should-i-balance-classes-in-a-training-data-set

[5] MIR Group. *Music Information Retrieval*, http://ifs.tuwien.ac.at/mir/audiofeatureextraction.html

[6] Romain Beaudon. *Cross validation strategy when blending/stacking*, https://www.kaggle.com/general/18793

[7] MLWave. *Kaggle Ensembling Guide*, https://mlwave.com/kaggle-ensembling-guide/

[8] XGBoost. *XGBoost*, https://xgboost.readthedocs.io/en/latest/

[9] Wikipedia. *Boosting*, https://en.wikipedia.org/wiki/Boosting_(machine_learning)

[10] Tom Van de Wiele. *Santander Product Recommendation Competition, 2nd Place Winner's Solution Write-Up*, http://blog.kaggle.com/2017/01/12/santander-product-recommendation-competition-2nd-place-winners-solution-write-up-tom-van-de-wiele/

[11] Gilberto Titericz Junior. *Otto Group Product Classification Challenge, Winner Solution*, https://www.kaggle.com/c/otto-group-product-classification-challenge/discussion/14335

[12] Kaggle Team. *Otto Product Classification Winner's Interview: 2nd place*, http://blog.kaggle.com/2015/06/09/otto-product-classification-winners-interview-2nd-place-alexander-guschin/

[13] Anisotropic. *Introduction to Ensembling/Stacking in Python*, https://www.kaggle.com/arthurtok/introduction-to-ensembling-stacking-in-python

[14] Ben Gorman. *A Kaggler's Guide to Model Stacking in Practice*, http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice/

6 Appendix

We are listing raw data from our data exploration as well as accuracies of various experiments. In addition we'll list the source code for one of our stacking experiment.

Indices	Feature	Statistic	Distribution	Comment
0-11	Timbre	Mean	Normal	
12-23	Timbre	SD	Chi Sq	
24-35	Timbre	Min	Varying Beta	Lot's of variation between indices
36-47	Timbre	Max	Varying Beta	Lot's of variation between indices
48-59	Chroma	Mean	Normal	
60-71	Chroma	SD	Chi Squared	
72-83	Chroma	Min	Normal	Heavily Truncated, ≥ 0
84-95	Chroma	Max	Always 1	Useless, can be removed
96-119	Rythm	Mean	Normal	Slightly truncated,≥ 0
120-143	Rythm	SD	Chi Squared	≥ 0
144-167	Rythm	Skewness	Skewed normal	Negative values, skewed
168-191	Rythm	Kurtosis	Chi Squared	≥ 0
192-215	Rythm	Median	Normal	Slightly truncated, ≥ 0
216-239	Rythm	Min	Normal	Heavily Truncated, ≥ 0
240-263	Rythm	Max	Normal	feature 263 is 'always' 0, and otherwise ≥ 0

Figure 7: *Our raw analysis of the dataset and feature meanings.*

Classifier	Accuracy	Comments
random forest (all features)	0.604	
random forest (pca 50% variance)	0.559	Can predict 60% of classes
random forest (pca 80% variance)	0.597	
knn (all features)	0.538	Tradeoff between separating labels and accuracy
knn (pca 50% variance)	0.557	
knn (pca 80% variance)	0.592	
knn (box cox all features)	0.598	Can't predict label 10
knn (box cox 80 % pxa)	0.571	Can't predict labels 7-10
svm (linear kernel) (all features)	0.608	Can predict labels from 2-> better than most others
svm (linear kernel) (pca 50% variance)	0.556	Can only predict label 1 reliably
svm (linear kernel) (pca 80% variance)	0.635	Worse at predicting 2-> than svm with all features
naive bayes (all features)	0.460	Can predict quite well about 60 % of classes
naive bayes (pca 50% variance)	0.514	Can predict quite well about 60 % of classes
naive bayes (pca 80% variance)	0.541	Can predict all classes
Stochastic Gradient Boosting (pca 80 %)	0.625	
Linear Discrimant analysis (all features)	0.648	
Linear Discrimant analysis (all features box-cox)	0.651	Scale and drop useless columns

Figure 8: *Subsection of our initial experiments tried*

```
from functools import reduce
from sklearn.model_selection import StratifiedKFold

X = ...
y = ...

base_classifiers = {'svc': ..., 'nb': ...}

# Cross validation to create out of sample predictions
cv = np.array(list(StratifiedKFold(n_splits=5, shuffle=True, random_state=None).split(X, y)))

# Train the models using cross validation, determine the correlations and accuracies
base_probabilities = []

for train_i, test_i in cv:
    X_train = X[train_i]
    X_test = X[test_i]

    y_train = y[train_i]
    y_test = y[test_i]

    # Fit classifiers
    fitted_classifiers = {key: classifier.fit(X_train y_train)
                          for key, classifier in base_classifiers.items()}
    # Predict probabilities, from the test fold
    probabilities = {key: classifier.predict_proba(X_train)
                     for key, classifier in fitted_classifiers.items()}

    # Create a dictionary with keys as id_classprediction and values as predicted probability
    probabilities = {key + '_' + str(i): probabilities[:,i]
                     for key, probabilities in probabilities.items() for i in range(10)}

    # Preserve the true label for meta classifier
    probabilities['true'] = y_test

    # Preserve the index of the fold
    probabilities['id'] = test_i + 1

    base_probabilities.append(probabilities)

# Merge the features from classifier into single Pandas dataframe
# This is now a separate training data set for the meta classifier
meta_features = reduce(lambda x, y: x.append(y), map(lambda d: pd.DataFrame(d),
                                                     base_probabilities))

# Remove the true labels and IDs from the meta features
y_meta = meta_features['true'].as_matrix().ravel()
id_meta = meta_features['id'].as_matrix().ravel()
X_meta = meta_features.drop('true', axis=1).drop('id', axis=1)

# Classifier to be used as meta classifier
meta_classifier = ...

meta_cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=None).split(X_meta, y_meta)

meta_predictions = []

# Train the meta classifier
for train_i, test_i in cv:
    X_train = X_meta[train_i]
    X_test = X_meta[test_i]

    y_train = y_meta[train_i]
    y_test = y_meta[test_i]

    meta_model = meta_classfier.fit(X_train, X_test)
    meta_predictions.append(meta_model.predict(X_test))
```

Figure 9: *Pseudo Python code of our ensemble process*