Mini Project Report: Music Classification

Anonymous Author(s)
Affiliation
Address
email

Abstract

In this report, we have tried four methods to fit a given data set which contains 750 records of the preferred music of a man and related data. We used four models and compared them by different metrics in the test data set split from the 'training_data.csv', and then we applied the models in the test data and compared their performance. The result shows that the boosting method has the best results on the Leader Board. Besides, we discussed an example of ethics in machine learning applications at the final. **Group numbers: 4.**

1 Introduction

The project aims to predict Andreas Lindholm's preference in a set of music and discuss ethical issues about the methods used for modeling. For prediction, we were provided a dataset of 750 songs with its features and Andreas' favor, and thus the issue can be formed as a binary classification problem, which amounts to finding a sound predictor that maps data points to a predicted value of its class via its features. In this project, several methods that are widely used for classification are implemented, e.g., logistic regression, random forest, K-nearest neighbourhood(KNN), and gradient boosting, and their performances are evaluated using quantitative measures such as accuracy, recall, and precision. We then compare their performance and select the optimal model for its great accuracy and generalization for the new dataset. Finally, we end with a concerning ethical problem about risks of model bias.

2 Methods

There are four methods of classification introduced in the report, including logistic regression, k-nearest neighbor, tree-based methods and boosting.

2.1 Logistic regression

Logistic regression [1] is an example of linear models and it is appealing for its reliability and efficient fitting, but it also has apparent defects, for example, the model ignores the interaction between features because it uses the only linear function, $h^{(w)}(x) = w^T x$ to predict the label based on the input variables x with the optimal weight w. Logistic regression is generally used for binary classification problems, and it takes the sign function to transform the real-valued linear function $h^{(w)}(x)$ into binary labels y.

Its prediction can be interpreted by the probabilistic model mathematically. Suppose that the true label of a sample is a random variable, and a linear regression model can be used to represent its probability as $p(y) = w^T x$. To enable the probability falls into [0,1], we use logistic function

$\sigma(x) = 1/(1+\exp(-x))$, so we have

$$p(y=1;w) = \frac{1}{1+\exp(-h(x))} = \frac{1}{1+\exp(-w^T x)} \tag{1}$$
$$p(y=0;w) = 1 - p(y=1;w)$$

We aims at maximizing the probability of labels and need to find the optimal weights $\hat{w}$ to achieve our goal as the features x is given. This challenge can be solved by maximum likelihood method, and the classification problem is converted as a optimization problem such as

$$\hat{w} = \arg\max \prod_{i=1}^{n} p(y = y^{(i)}, w)$$
$$\Rightarrow \log(\hat{w}) = \arg\min -y \log(\sigma(w^T x)) - (1-y)\log(1-\sigma(w^T x)) \tag{2}$$

$\log(\hat{w})$ is known as logistic loss $L((x,y),w)$, and we intend to find the weight to minimize its average, namely empirical risk $\varepsilon(w)$,

$$\hat{w} = \arg\min(w) = \arg\min \frac{1}{n} \sum_{i=1}^{n} -y^{(i)} \log(\sigma(w^T x^{(i)})) - (1-y^{(i)})\log(1-\sigma(w^T x^{(i)})) \tag{3}$$

2.2 K-nearest neighbors algorithm

The K-nearest neighbors (KNN) classification algorithm is proposed by Cover and Hart in 1968 [2] and it is one of the simplest and robustest non-parametric methods. Its basic idea is that every sample can be represented by its K nearest neighbors. In this report, the KNN algorithm is used to determine the label of unknown samples according to current data with labels.

The KNN algorithm upon the distance between two data points to predict the output. The most common distance metrics is Euclidean distance. The Euclidean distance is the straight line distance between two data points in a plane. The distance between two points in the multidimensional space can be expressed as:

$$d(x,y) = \sqrt{\sum_{j=1}^{J} (x_j - y_j)^2} \tag{4}$$

where x and y are two samples from training set and testing set; J is the dimension of multidimensional space.

2.3 Tree-based methods

2.3.1 Classification trees

Classification Trees is a method that can help people to make decisions by classifying data samples. It can help to decide the importance of the features with different criterion such as Entropy and Gini(Equation 5). A basic decision tree has the node, branch, and left node stated by James et al.[3]. The node represents the value divided according to the division criterion of a feature. The branch denotes whether the next node satisfies the parent node. The left node represents the label or class of the data sample. A tree is to iteratively divide the data into sub-trees according to the standard until it can not be divided, and then determine which category it belongs to. A fully grown classification tree is perfectly fit for all samples but has lower performance on new data, which calls overfitting. Pruning tree is a good way to enhance the accuracy, the normal ways are Reduced Error Pruning and Cost Complexity Pruning[4].

$$Ent(T) = -\sum_{k=1}^{|y|} p_k * \log_2 p_k; Gini(T) = 1 - \sum_{k=1}^{|y|} p_k^2 \tag{5}$$

2.3.2 Random forests

Given a data set T of size N, randomly select N samples in set T with replacement and repeated N times to obtain k data sets of size N. Then construct N classification trees by using N data set, after

that random forests are built. And when N increases to infinite, there are approximately 36.8% of the data will not be picked(Equation 6), and this data can be used to do validation. For a test sample, the decision making was voted by every sub classification tree, the results of voting will be the label of the sample. Random forests can lower the risk of overfitting, so it is not needed to prune the classification trees.

$$\lim_{N\to\infty}(1-1/N)^N \approx 0.368 \tag{6}$$

2.4 Boosting

2.4.1 Two Kinds of Boosting

Boosting is an ensemble learning method. It means that we divide a huge machine learning task into several small tasks, and for each task, we use a classifier to get a probably approximately correct answer. To get the final answer, we combine all the results given by those weak classifiers. With different learning and combination strategies, there are two kinds of boosting methods used in machine learning. One is adaptive boosting, or it could be called Adaboost. It was first published by Freund and Schapire in 1997 [5]. The other one is gradient boosting, and it was published by Friedman [6]. In our experiment, we used the gradient boosting method, and the toolkit is LightGBM developed by Microsoft[7].

2.4.2 Gradient Boosting and Decision Tree

The classification and regression tree(CART) is usually used by gradient boosting method as a base classifier. The method is based on a additive model, which we could list the equation like:

$$f_M(x) = \sum_{m=1}^{M} T(x;\Theta_m) \tag{7}$$

In the equation, $T(x;\Theta_m)$ represents the results of a single decision tree, Θ_m is the parameters of weights, and M is the number of trees. So the final result is a linear addition of the results of the weak classifier. To get each Θ_m, we use the equation:

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i)) + T(x_i;\Theta_m)) \tag{8}$$

In the equation above, if we use square loss function, we would get $L(y, f(x)) = (y - f(x))^2$ and then we get $L(y, f_{m-1}(x) + T(x;\Theta_m)) = [r - T(x;\Theta_m)]^2$, in which $r = y - f_{m-1}x$. So it means that for boosting method, we just need to calculate the residual. However, for a much more normal loss function, we could not get the residual. So gradient boosting method uses the value of the negative gradient of the loss function in the current model to fit the residual.

$$r = -[\frac{\partial L(y, f(x_i))}{\partial f(x_i)}]_{f(x)=f_{m-1}(x)} \tag{9}$$

In conclusion, for each iteration, the boosting method will calculate the residual and then update the corresponding tree. The linear combination of the trees will be the final results for the model.

3 Modeling

3.1 Data pre-processing

The data pre-processing methods vary with each classification methods. For logistic regression, PCA is used to reduce the dimensions of features, and one-hot encoding is used to change the categorical data to numerical data, i.e., mode, but the performance is worse than without any pre-processing, and thus all input has been used in final model. Normalization is used on KNN and Boosting to make sure all the data are scale from 0 to 1, in case of some feature has large weight during calculation. Boosting remove two features ($mode$ and $time\_signature$), since the weight of these two features are really low during the process of training model. For classification tree, there is no difference between whether do one-hot encoding or not, the possible reason is the $mode$, key, $time\_signature$ are not important features. The feature selection method is not implemented in classification trees and random forests, since the methods has ability to pick the important features.

Table 1: Summary Statistics for Performance of Classifiers

Method	Leader Board	Accuracy	F1-score	Recall	Precision
Logistic regression	0.815	0.867	0.897	0.888	0.906
KNN	0.755	0.800	0.730	0.727	0.753
Random forests	0.815	0.847	0.878	0.864	0.892
Gradient Boosting	0.860	0.839	0.894	0.918	0.873

3.2 Parameters tuning

3.2.1 Logistic Regression

There are three hyparameters involve in our parameter tuning for regression model and we mainly use Gridsearch to find the optimal model with the highest accuracy given a set of hyperparameters. To avoid over-fitting, we set the regularization strength $C = [10^{-5}, 10^{5}]$, with less C indicating stronger regularization and the available penalty such as $l1, l2, Elasticnet$to look for more possibilities. As some solvers have no capabilities of handing penalty $Elasticnet$, we set additional solver $saga$ and $liblinear$. The optimal model found the solution using $liblinear$ with regularization $C = 0.20$ and $penalty = l1$.

3.2.2 K-nearest Neighbors

The KNN algorithm is a non-parametric method and it classifies new data based on its proximity to K neighbors from training data. K is the hyperparameter of the algorithm, we use Scikit-Learn Library to implement the classification based on KNN and the default value of K is 5. We get an ideal value of 35 for K according to experiments. Another important issue is how to compute distance, compared with Manhattan distance, Euclidean distance can lead to higher accuracy. Thus, we set k as equal to 35 and p equals 2 which is related to Euclidean distance.

3.2.3 Random forest

Since decision tree is base classifier of random forest, so the process of parameter tuning is not refer in this part. There are three essential parameters in this model, $criterion$, ccp_alpha or max_depth, and $n\_estimators$. When we fix the $criterion$ and $n\_estimators$, and whatever we adjust the ccp_alpha or max_depth, the accuracy will be lower. But we tested on the test dataset, the accuracy of after pruning(0.815) is better than without pruning(0.805), there are not very big difference between this two model, and since the number of the test data is fairly small, so the random forests is not necessary to . The best number of estimators is 100 after verified.

3.2.4 Boosting

There are lots of parameters in the boosting method. In our experiment, we used grid search to find better combination of parameters. We first set $learning\_rate$ to 0.005 to find the best parameters. $Learning\_rate$ controls the speed of iteration, the model will fit more precisely while it is lower, but the times of iteration will increase. Then we are about to change max_depth and num_leaves, which are two parameters to determine the shape of all the child trees. After that, we set $bagging\_fraction$ and $bagging\_freq$, which means how much and how often we samples from the origin data, to avoid overfitting.

4 Evaluation

The result of classification methods shows in Table 1. Accuracy, accuracy from the leader board, and precision are used to evaluate the performance of the classification models. Since accuracy and accuracy from the leader board are the factories to judge the model's performance on the data, and the experimental problem is based on user preferences, precision is also an important factor for this problem.

Logistic regression and KNN are simple classification methods, and logistic regression is easy to underfit, and cannot handle a large number of multi-type features or variables well. The weights of

the feature are the same as KNN but according to later experiments the weights of the features is different, and the experiment data is also imbalanced, so KNN and logistic regression are both not suitable for this problem. Random forests and gradient boosting are stable and did well in our test data and leader-board since the base classifier of these two methods are decision trees. The decision trees can select the important features, so the high dimension of the data works fine to decision tree.

From Table 1, it tells that the performance of KNN still can be improved, which shows low accuracy on both leader board and test data. Logistic regression did well in test data but has lower performance on the leader board which implies that logistic regression is unstable. In principle, the performance of random forests and gradient boosting should be similar, and they do on test data. While their accuracy and precision are close, the random forest shows poor performance on the leader board. **In summary, gradient boosting is decided to use 'in production'**.

5 Discussion of reflection

For task (b), we discussed both the positive side and negative side of whether machine learning engineers have a responsibility to inform and educate the client about the risk, and our thoughts and opinions towards this task.

5.1 Positive sides

The insurance company should be informed of how the model works so they can prepare remedies to avoid potential risks or manage possible consumer harm. As the example shown, although improves the processing efficiency and decreases errors caused by human, this method may inadvertently create unintentional discrimination and produces negative impacts on the protected classes. For the insurance company, the subjective evaluation of insurability is completed by this method, and the decision naturally depends on probabilities derived from the insurance applicant's current situation and previous pattern. If this algorithm learns patterns that may lead to illegal discrimination, for example, charge more for those who are not multilingual learners that imply their education level, then the insurance company should decrease its exposure to bias claims to comply with the law and guarantee its legitimacy[8]. Besides, it would be better for the insurance company to educate employers to identify the discriminatory outputs from the model and prepare defenses before they arise.

For the insurance company managers, the more they know the model, the more they can do for the improvement. It is known that the algorithm matters when we train the model, the source of data itself is also a key to a sound model. The maxim 'Garbage in, garbage out' reveals that the biases often come from the input data[9]. The insurance company, as a user, can provide feedback for further development and possible improvement. For instance, it helps collect data of minor cases that lack sufficient historical data so the model has extensive ability to minimize possible risks of unintended consequences and correct the model. The process can be done by testing unexpected scenarios and constructing controls into defective models to avoid adverse outputs.

It is the engineers' liability according to the IEEE code of ethics[10] for engineers Article 3 and it also protects our clients' privilege as a consumer. Engineers should consider not only how well the model works but also how well the model complies with legal requirements. Moreover, developers should anticipate the possible consequences of their projects with their clients and inform of circumstances that may lead to a conflict of interest, i.e., potential risks. There does not exist an insurance company that wants to apologize to and pay for the regulators and customers for the discriminational effects generated by their technology, so developers should state the potential issues and make an appropriate plan, in compliance with the regulation, to solve the problems before they trigger more problems in practices or bias beyond the insurance company's daily operations.

5.2 Negative sides

As a machine learning engineer, there is no responsibility to inform and educate the clients about some prejudices and biases caused by the machine learning methods, but we have a responsibility to guarantee we use the correct training data. In the given examples, I think the bias caused by the Microsoft Twitter bot and the Google Photo system is caused by the wrong data input. The bot learned some racist words from other people and then it becomes racist, and the photosystem had

little examples of black people so it got the wrong result. So technology is not guilty but we human beings should take responsibility to avoid the results. If we design a chatting bot, we should avoid it learning some bad words, and for the photosystem, we should input enough photo examples of all races to guarantee it could learn them correctly.

Now it comes to the insurance company examples. The given data will show that people who have higher yearly income would like to buy more expensive insurance while the low-income people would buy cheaper one or do not buy insurance. So for the company, we can recommend different products according to different customers, and for the customers, they will be accurately recommended products that are more acceptable to them. I do not think that it is a prejudice for different costumers, in contrast, this is exactly what machine learning should do.

5.3 Summary

Sometimes the side effect of machine learning is indeed creating biases in many aspects, sometimes it comes from unbalanced data and designers' unprofessional skills, etc. Even though Amazon created a biased recruiting engine that did not rate candidates IT-applicants in a gender-neutral way, Amazon abandoned this project [11]. The intention of creating machine learning model is to improve the efficient and eliminate bias since a human has emotion preferences and always make bias decision, but a machine does technically and logically follow the designers' instructions. When the biases happen, the engineers fix it instinctively, and try to make the model unbiased. But adjusting the model means more effort and energy, which requires a high cost of the model. While some clients may not require a high-quality product, the engineer does not have the responsibility to perfect the model. Under some circumstances, if the bias cannot be eliminated, an engineer should specify the possible consequences of the products with their clients [12]. In general, machine learning engineers should make their model unbiased as much as possible, since a biased product cannot convince people and lose people's reliance.

The core of insurance products is to protect society's wealth, remove social evils, reduce risks, and maintain social security benefits. The insurance companies have to inform clients of the potential risks while engineers should do as much as possible to avoid bias. If the insurance companies pull themselves out of social responsibility, people's lives would lose their security and social stability would be reduced. A possibility is that people in the color and the poor can't purchase reasonable insurance products because of the bias in machine learning. It seems absurd but possible that people who need insurance can't be insured.

6 Conclusion

In this report, focusing a predicting users' preference task, we have tried four kinds of machine learning methods to solve the problem. The methods include logistic regression, K-nearest neighborhood, random forests, and gradient boosting. In our experiment, we have split the training data into two parts. We trained our model in 80% data while testing it in the other 20% data. The results show that ensemble methods such as random forests and gradient boosting have better performance, where gradient boosting seems more fit for the data compared with random forests. However, because of the limited size of training data, we could not give a conclusion showing that which model is perfect in general.

According to the feature weights of the boosting, we found that $mode$ and $time\_signature$ have a low weight, so Andreas Lindholm may not care about these two features when choosing favorite music. At the same time, $speechiness$, $loudness$, $duration$, $danceability$, and $acousticness$ have high weights after training the model, so they may be the features Andreas Lindholm caring most when listening to the music.

In conclusion, for the given task, we tried four kinds of machine learning methods and found the best method to fit the data. And according to the weight distribution, we speculated the five features about which Andreas Lindholm more likely to cares. Besides, we discussed the responsibility of machine learning engineers.

References

[1] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[2] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 1967.

[3] Claude Sammut and Geoffrey I. Webb, editors. *Classification Tree*, pages 209–209. Springer US, Boston, MA, 2017.

[4] A. B. Nobel. Analysis of a complexity-based pruning scheme for classification trees. *IEEE Transactions on Information Theory*, 48(8):2362–2368, 2002.

[5] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[6] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[7] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in neural information processing systems*, pages 3146–3154, 2017.

[8] Alan Bundy. Preparing for the future of artificial intelligence, 2017.

[9] H Room. Re: Big data: A tool for inclusion or exclusion? workshop, project no. p145406. 2014.

[10] Edgar Serna-Montoya. Ieee code of ethics for engineers. *Lámpsakos*, (1):65–65, 2009.

[11] Jeffrey Dastin. Amazon scraps secret ai recruiting tool that showed bias against women. *Reuters*, Oct 2018.

[12] IEEE. Ieee code of ethics for engineers. *Revista Digital Lámpsakos*, page 65, 2009.

Appendix

Logistics regression

Listing 1: Logistic regression

```
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.metrics import accuracy_score, f1_score, recall_score, precision_score


# train the model
# read data from the file
data = pd.read_csv('training_data.csv')
# split features and label
x = data.iloc[:,:-1]
y = data.iloc[:,-1]

# split dataset for train and test
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size = 0.2,shuffle = True)
```

```
# initialize a logistic regression model
logreg = LogisticRegression()
# initialize a set of hyperparameters for further tuning
para = {'C': np.logspace(-5,5,8), 'penalty':['l2','l1', 'elasticnet'],
    'solver':['saga', 'liblinear'] }
# implement 5-fold gridsearch to find the optimal model
logSearch = GridSearchCV(logreg,para,cv=5)
# train model using train dataset
logSearch.fit(xtrain,ytrain)
# print the parameters and accuracy of the optimal model
print('best hyperparameters: ', logSearch.best_params_)
print('accuracy ',logSearch.best_score_)

# predict labels for test dataset
pred = logSearch.predict(xtest)
# print metrics and evaluate the performance of the model
print('accuracy', accuracy_score(pred, ytest))
print('precision',precision_score(pred,ytest))
print('recall',recall_score(pred,ytest))
print('f1-score', f1_score(pred, ytest))

# predict labels for new data
newx = pd.read_csv('songs_to_classify.csv')
prednew =logSearch.predict(newx)
# format the ouput
res3 = ''.join(str(label) for label in prednew)
print(res3)
```

K-nearest neighbourhood

Listing 2: K-nearest neighbourhood

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn.preprocessing as skl_pre
import sklearn.linear_model as skl_lm
import sklearn.discriminant_analysis as skl_da
import sklearn.neighbors as skl_nb
import sklearn.model_selection as skl_ms
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
import sklearn.metrics as skl_mt

# load the data
data_path = 'data/training_data.csv'
test_path = 'data/songs_to_classify.csv'
pd.set_option('display.max_columns',30)
music = pd.read_csv(data_path)
test = pd.read_csv(test_path)
music.head()
#one hot
music = music.join(pd.get_dummies(music.key,prefix='k'))
music = music.drop(columns = ['key'])
music = music.join(pd.get_dummies(music.time_signature,prefix='ts'))
music = music.drop(columns = ['time_signature'])
music.head()
test = test.join(pd.get_dummies(test.key,prefix='k'))
test = test.drop(columns = ['key'])
test = test.join(pd.get_dummies(test.time_signature,prefix='ts'))
test = test.drop(columns = ['time_signature'])
```

```
test.head()
#Normalized
mm = skl_pre.MinMaxScaler()
duration = np.array(music['duration']).reshape(-1,1)
music['duration'] = mm.fit_transform(duration)
tempo = np.array(music['tempo']).reshape(-1,1)
music['tempo'] = mm.fit_transform(tempo)

ss = skl_pre.StandardScaler()
loudness = np.array(music['loudness']).reshape(-1,1)
music['loudness'] = ss.fit_transform(loudness)

mm2 = skl_pre.MinMaxScaler()
duration = np.array(test['duration']).reshape(-1,1)
test['duration'] = mm2.fit_transform(duration)
tempo = np.array(test['tempo']).reshape(-1,1)
test['tempo'] = mm2.fit_transform(tempo)

ss2 = skl_pre.StandardScaler()
loudness = np.array(test['loudness']).reshape(-1,1)
test['loudness'] = ss2.fit_transform(loudness)

X_train = music.drop(columns = ['label'])
Y_train = music['label']
X_test = test

np.random.seed(0)
model = skl_nb.KNeighborsClassifier(n_neighbors=35, p=2 )
model.fit(X_train,Y_train)
Y_predict = model.predict(X_test)
print(Y_predict,sep='')
#np.savetxt('1117knn.txt',Y_predict,fmt='%d')
```

Tree-based classification

Classification trees

Listing 3: Classification trees

```
import numpy as np # to do linear algebra

import pandas as pd # to store the data

import sklearn.tree as skl_tr # to create tree model

# grid search for best parameter
from sklearn.model_selection import GridSearchCV

# load the data
song = pd.read_csv('training_data.csv', na_values='?', dtype={'ID':
    str}).dropna().reset_index()
print('Shape Song:\t{}'.format(song.shape))


# the features of the data
attr = ['acousticness', 'danceability', 'duration',
        'energy', 'instrumentalness', 'key', 'liveness',
        'loudness', 'mode', 'speechiness', 'tempo',
        'time_signature', 'valence']
label = 'label'
# the colinearity between different feature by numeric
print(song[attr].corr())

```

```
X = song[attr] # data
y = song[label] # label

# to see whether the 'max_depth' or 'ccp_alpha' has better performance
dt1 = skl_tr.DecisionTreeClassifier()
param_grid = [{'max_depth': [2,3,4,5,6,7,8,9,10,11,12], 'criterion':
    ['gini','entropy']}]
search = GridSearchCV(dt1, param_grid, cv=10)
search.fit(X, y)
print(search.best_params_, search.best_score_)

dt2 = skl_tr.DecisionTreeClassifier()
ccp_alpha = np.arange(0,0.052, 0.002)
param_grid = [{'ccp_alpha': ccp_alpha, 'criterion': ['gini','entropy']}]
search = GridSearchCV(dt2, param_grid, cv=10)
search.fit(X, y)
print(search.best_params_, search.best_score_)

# after validation, we get best parameters combination are 'entropy' and
    'ccp_alpha' = 0.012,
# then we train the model with whole data set and get solution on test data

test = pd.read_csv('songs_to_classify.csv',
                   na_values='?',
                   dtype={'ID': str}).dropna().reset_index()[attr]

predict = search.predict(test)

# format the solution as '001100...'
res = str(str(predict).split(','))
solution = ''
for i in range(len(res)):
  if res[i].isdigit():
    solution += res[i]
print(solution)
```

Random forests

Listing 4: Random forests

```
import pandas as pd # to store the data

from sklearn.ensemble import RandomForestClassifier # to create random forest

# to calculate the accuracy
from sklearn.metrics import accuracy_score

# grid search for best parameter
from sklearn.model_selection import GridSearchCV

from sklearn.metrics import precision_recall_fscore_support


song = pd.read_csv('training_data.csv', na_values='?', dtype={'ID':
    str}).dropna().reset_index()
print('Shape Song:\t{}'.format(song.shape))
print(song.attrs)

attr = ['acousticness', 'danceability', 'duration',
        'energy', 'instrumentalness', 'key', 'liveness',
        'loudness', 'mode', 'speechiness', 'tempo',
        'time_signature', 'valence']
label = 'label'
```

```
# the colinearity between different feature by numeric
print(song[attr].corr())

# To keep our group train and test in same dataset, we divided the data set randomly
# and the ratio is 8:2, train vs test.
trainData = pd.read_csv('train_data.csv', na_values='?', dtype={'ID':
    str}).dropna().reset_index()
testData = pd.read_csv('test_data.csv', na_values='?', dtype={'ID':
    str}).dropna().reset_index()

trainData_X = trainData[attr]
trainData_Y = trainData[label]

testData_X = testData[attr]
testData_Y = testData[label]

# to search the best parameter
rf = RandomForestClassifier()
param_grid = [
{'n_estimators': range(80,120,10), 'max_features' : ['sqrt','log2',None],
    'criterion' :['gini','entropy']}
]
search = GridSearchCV(rf, param_grid, cv=5)
search.fit(trainData_X,trainData_Y)
print(search.best_params_, search.best_score_)
print(search)


test = pd.read_csv('songs_to_classify.csv',
                   na_values='?',
                   dtype={'ID': str}).dropna().reset_index()[attr]

predict = search.predict(test)

# format the solution as '001100...'
res = str(str(predict).split(','))
solution = ''
for i in range(len(res)):
  if res[i].isdigit():
    solution += res[i]
# print out the solution for leader board
print(solution)

pred = search.predict(testData_X)
print('accuracy: ', accuracy_score(testData_Y, pred) )
print('precision, recall, fscore: ', precision_recall_fscore_support(testData_Y,
    pred))
```

Gradient boosting

Listing 5: Using grid search to find the best parameters.

```
import lightgbm as lgb
import numpy as np
from sklearn.model_selection import GridSearchCV

x = np.load('./Version2/totalData_deleteFeatures.npy')
y = np.load('label.npy')
y = np.ravel(y)

parameters = {
    'max_depth': [2, 4, 6, 8, 10],
```

```
    'learning_rate': [0.01, 0.05, 0.1, 0.15],
    'num_leaves': [10, 15, 20, 30],
    'feature_fraction': [0.6, 0.7, 0.8, 0.9, 0.95],
    'bagging_fraction': [0.6, 0.7, 0.8, 0.9, 0.95],
    'bagging_freq': [2, 4, 5, 6, 8],
    'lambda_l1': [0, 0.1, 0.4, 0.5, 0.6],
    'lambda_l2': [0, 10, 15, 35, 40],
    'cat_smooth': [1, 10, 15, 20, 35],
    'min_child_samples': [16, 18, 20, 22, 24],
    'min_child_weight': [0.0005, 0.001, 0.002]
}
gbm = lgb.LGBMClassifier(objective='binary',
                         learning_rate=0.001,
                         n_estimators=500)

gsearch = GridSearchCV(gbm, param_grid=parameters, scoring='accuracy', cv=5)
gsearch.fit(x, y)
print('Best parameters:{0}'.format(gsearch.best_params_))
print('Best scores:{0}'.format(gsearch.best_score_))
print(gsearch.cv_results_['mean_test_score'])
print(gsearch.cv_results_['params'])
```

Listing 6: Training the gradient boosting model.

```
import lightgbm as lgb
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.model_selection import KFold
import joblib

# import data
x = np.load('./Version2/totalData_deleteFeatures.npy')
y = np.load('label.npy')
y = np.ravel(y)

# define KFold validation, here K=5
kf = KFold(n_splits=5, shuffle=True)
accuracy_arr = []

# define the machine learning model
gbm = lgb.LGBMClassifier(objective='binary',
                         learning_rate=0.01,
                         max_depth=8,
                         num_leaves=10,
                         min_child_weight=0.0005,
                         min_child_samples=20,
                         bagging_fraction=0.95,
                         bagging_freq=8,
                         cat_smooth=1,
                         n_estimators=500)

# use the model defined to do the training(5-Flod validation)
# after each iteration, store the accuracy
for train, test in kf.split(x):
    x_train, y_train = x[train], y[train]
    x_test, y_test = x[test], y[test]
    gbm.fit(x_train, y_train, eval_set=[(x_test, y_test)], eval_metric='l2',
        early_stopping_rounds=80)
    y_pred = gbm.predict(x_test, num_iteration=gbm.best_iteration_)
    print('Accuracy is: ', accuracy_score(y_test, y_pred))
```

```
    accuracy_arr.append(accuracy_score(y_test, y_pred))
    print('Recall is: ', recall_score(y_test, y_pred))
    print('Precision is: ', precision_score(y_test, y_pred))
    print('F1 Score is: ', f1_score(y_test, y_pred))
    print('The rmse of prediction is:', mean_squared_error(y_test, y_pred) ** 0.5)
    print('Feature importances:', list(gbm.feature_importances_))

# get the average accuracy and feature weights, then store the model
print('The avarage accuracy is: ' + str(np.mean(accuracy_arr)))
print(accuracy_arr)
print(gbm.feature_importances_)
joblib.dump(gbm, 'gbm.pkl')
```