Music Classification

Mingzhong Deng

March 8, 2019

Abstract

This project uses machine learning algorithms, LDA and Naive Bayes, to train and classify short clips, five seconds, of music by various artists. The music clips are recorded by microphone then moved and read by Matlab. This project is trying to build a classifier that can identify music from different artists.

1 Introduction and Background

Computers cannot identify human hearing as quick as human. Even people have a hard time to identify the genre or the singer from a really short music clip. The purpose of the project is to classify the genre or artist of different music clips with machine learning algorithms, so the computers can make this task possible.

There are three tests in the project. The first test is the classification among Michael Jackson (pop), Soundgarden (rock) and Beethoven (classical), three different types of bands. The second test is the classification among three Seattle bands, Soundgarden, Alice & Chains, and Pearl Jam with the same genre, rock. The third test is the classification among classical, hiphop, and pop music.

2 Theoretical Background

There are three algorithms used in this project, Linear Discriminant Analysis, Supporting Vector Machine, and Naive Bayes algorithm.

Linear discriminant analysis (LDA) is a generalization of Fischer's linear

discriminant method, a method to find a linear combination of features that characterizes or separates two or more classes of objects. LDA makes some simplifying assumptions about data: 1.The data is Gaussian, that each variable is is shaped like a bell curve when plotted. 2. Each attribute has the same variance, that values of each variable vary around the mean by the same amount on average. With these assumptions, the LDA model estimates the mean and variance from your data for each class. It is easy to think about this in the univariate (single input variable) case with two classes. LDA makes predictions by estimating the probability that a new set of inputs belongs to each class. The class that gets the highest probability is the output class and a prediction is made. The model uses Bayes Theorem to estimate the probabilities. The LDA has some of the limitations too. It is unstable with well separated casses. logistic regression can become unstable when the classes are well separated. It is also unstable with few examples. Logistic regression can become unstable when there are few examples from which to estimate the parameters.

Naive Bayes is a classification algorithm for binary (two-class) and multi-class classification problems. The technique is easiest to understand when described using binary or categorical input values. Naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features. Naive Bayes is a popular (baseline) method for text categorization, the problem of judging documents as belonging to one category or the other with word frequencies as the features. With appropriate pre-processing, it is competitive in this domain with more advanced methods including support vector machines. Learning a naive Bayes model from your training data is fast. Training is fast because only the probability of each class and the probability of each class given different input (x) values need to be calculated. No coefficients need to be fitted by optimization procedures.

3 Algorithm Implementation and Development

The algorithm for extracting original data and data after PCA is the same for the four tests of the project.

- Load songs files of each artist.

- Create the spectrogram of each five second clip of each artist from HW2.
- Perform SVD econ to extract the significant feature of each artist.
- Use the compressed data from V matrix from SVD for traing and testing data for the two machine learning algorithms.
- Randomly split data into training and testing sets, create labels for the training data.
- Perfrom LDA and Naive Bayes algorithm to train the model.
- Get the accuracy with the testing data.

4 Computational Results

After running the dataset with songs composed by Michael Jackson, Soundgarden, and Beethoven, the accuracy of prediction by the Naive Bayes model is 46.67%, the accuracy of prediction of the test data is 66.67%. The result of prediction is not too good, but acceptable. Due to the difference between the three artists, the result should be better.

After running the dataset with songs composed by Pearl Jam, Soundgarden, and Alice & Chains, the accuracy of prediction by the Naive Bayes model is only 20.00%, the accuracy of prediction of the test data is 46.67%. The result of prediction is not good. Due to the similarity among the three artists in the same genre and location of band, the result is reasonable but there are a lot of rooms for improvement.

After running the dataset with songs composed by classical, HipHop, and pop music, the accuracy of prediction by the Naive Bayes model is 73.33%, the accuracy of prediction of the test data is 73.33%. The result of prediction is surprisingly good. It is obvious that classical music is distinguishable comparing to the other two genre, there are a lot of similarity between HipHop and pop music. Both LDA and Naive Bayes models did well to classify the three genres.

5 Summary and Conclusions

In the project, each of the three datasets are analyzed with two machine learning algorithms. Different results are received based on the test cases. The first test's data have an obvious differences between different labeled data, but the outcome is not as good as expected. The second test's data are really similar in terms of sounds, rhythm, and vocal voices. It is understandable to have a low accuracy comparing to the other test cases, but the outcome of the result is looking too low. The third test's data gives the best result but there should be some room for improvement. For future improvement, I should collect more data with more variety, for example, for Michael Jackson's song, I should collect some of his fast rhythm and slow rhythm song, different period during the songs. Because the algorithm and preprocessing of the data works well for the third case, but not so well the first and second cases. Collecting more data for training should be a method for improvement.

6 References

Data-Driven Modeling & Scientific Computation

`https://en.wikipedia.org/wiki/Linear_discriminant_analysis`
`https://machinelearningmastery.com/linear-discriminant-analysis-for-machine-learning/`
`https://en.wikipedia.org/wiki/Naive_Bayes_classifier`
`https://machinelearningmastery.com/naive-bayes-for-machine-learning/`

Appendix A - MATLAB functions

- `audioread`: audioread(filename) reads data from the file named filename, and returns sampled data, y, and a sample rate for that data, Fs.
- `svd`: [U,S,V] = svd(A) performs a singular value decomposition of matrix A, such that $A = U * S * V'$.

- `fitcnb`: returns a multiclass naive Bayes model (Mdl), trained by the predictors in table Tbl and class labels in the variable Tbl.ResponseVarName.
- `fitcecoc`: returns a full, trained, multiclass, error-correcting output codes (ECOC) model using the predictors in table Tbl and the class labels in Tbl.ResponseVarName. fitcecoc uses K(K 1)/2 binary support vector machine (SVM) models using the one-versus-one coding design, where K is the number of unique class labels (levels). Mdl is a ClassificationECOC model.
- `classify`: classify(sample,training,group) classifies each row of the data in sample into one of the groups in training. sample and training must be matrices with the same number of columns. group is a grouping variable for training. Its unique values define groups; each element defines the group to which the corresponding row of training belongs. group can be a categorical variable, a numeric vector, a character array, a string array, or a cell array of character vectors. training and group must have the same number of rows. classify treats ¡undefined¿ values, NaNs, empty character vectors, empty strings, and ¡missing¿ string values in group as missing data values, and ignores the corresponding rows of training. The output class indicates the group to which each row of sample has been assigned, and is of the same type as group.

Appendix B - MATLAB code

```
clear all; close all; clc
%%
A = [];
a = 100;

for i = 1:25
    fname = strcat("alice (" ,int2str(i));
    fname = strcat(fname, ").m4a");
    y = audioread(char(fname));
    y = single(y);
    v = y'/2;
    dur = 5;
```

```
Fs = length(v)/dur;
L = length(v)/Fs;
k=(2*pi/(2*L))*[0:(length(v)-1)/2 -(length(v)-1)/2:-1];
ks=fftshift(k);

if rem(length(k), 2) > 0
    ks = ks(length(ks)/2:end);
else
    ks = ks(length(ks)/2-1:end);
end

k = single(k); ks = single(ks);
tfinal = length(v)/Fs;
t = single(1:length(v))/Fs;
tslide = single(0:0.1:length(v)/Fs);

Sgt_spec = [];
Spec1 = [];

for ii = 1:length(tslide)
    g = exp(-a*(t-tslide(ii)).^2);
    sg = g.*v;
    sgt = fft(sg);
    sgt = fftshift(sgt);
    len = length(sgt);
    sgt = sgt(int64(len/2):end);
    sgt = single(sgt);
    Sgt_spec = [Sgt_spec abs((sgt))];
end
temp = zeros(1, 8388608);
for i = 1:length(Sgt_spec)
    temp(i) = Sgt_spec(i);
end
res = zeros(1, 1048576);
for i = 1:1048576
    res(i) = temp(4*i);
end
A = [A; res];
```

```
    dummy = 0;
end
%%
alice = A;
save('alice.mat', 'alice');


clear all; close all; clc

load alice.mat;         % alice = A;
load beethoven.mat;     % beethoven = A;
load classical.mat;     % classical = A;
load hiphop.mat;        % hiphop = A;
load mj.mat;            % mj = A;
load pearljam.mat;      % pearljam = A;
load pop.mat;           % pop = A;
load soundgarden.mat;   % soundgarden = A;

%%

dat = [alice' beethoven' classical' hiphop' mj' pearljam' pop' soundgarden'];
[u,s,v] = svd(dat, 'econ');

%%

modes = 10;
alice = v(1:25,1:modes);
beethoven = v(26:50,1:modes);
classical = v(51:75,1:modes);
hiphop = v(76:100,1:modes);
mj = v(101:125,1:modes);
pearljam = v(126:150,1:modes);
pop = v(151:175,1:modes);
soundgarden = v(176:200,1:modes);

q1=randperm(25);

alice_train = alice(q1(1:20),:);
```

```
alice_test = alice(q1(21:end),:);
beethoven_train = beethoven(q1(1:20),:);
beethoven_test = beethoven(q1(21:end),:);
classical_train = classical(q1(1:20),:);
classical_test = classical(q1(21:end),:);
hiphop_train = hiphop(q1(1:20),:);
hiphop_test = hiphop(q1(21:end),:);
mj_train = mj(q1(1:20),:);
mj_test = mj(q1(21:end),:);
pearljam_train = pearljam(q1(1:20),:);
pearljam_test = pearljam(q1(21:end),:);
pop_train = pop(q1(1:20),:);
pop_test = pop(q1(21:end),:);
soundgarden_train = soundgarden(q1(1:20),:);
soundgarden_test = soundgarden(q1(21:end),:);

%%

X = [mj; soundgarden; beethoven];
Y = [ones(25,1); 2*ones(25,1); 3*ones(25,1)];
rng(1);
CVMdl = fitcnb(X,Y,'Holdout',0.20);
CMdl = CVMdl.Trained{1};          % Extract trained, compact classifier
testIdx = test(CVMdl.Partition); % Extract the test indices
XTest = X(testIdx,:);
YTest = Y(testIdx);

idx = randsample(sum(testIdx),15);
label = predict(CMdl,XTest);
% table(YTest(idx),label(idx),'VariableNames',...
%     {'TrueLabel','PredictedLabel'})
bar(label)
cnt = 0;
for i = 1:length(YTest)
    if YTest(i) == label(i)
        cnt  = cnt + 1;
    end
end
```

```
accuracy = cnt/length(YTest)
%%

X = [pearljam; soundgarden; alice];
Y = [ones(25,1); 2*ones(25,1); 3*ones(25,1)];
rng(1);

CVMdl = fitcnb(X,Y,'Holdout',0.20);
CMdl = CVMdl.Trained{1};          % Extract trained, compact classifier
testIdx = test(CVMdl.Partition); % Extract the test indices
XTest = X(testIdx,:);
YTest = Y(testIdx);

idx = randsample(sum(testIdx),15);
label = predict(CMdl,XTest);
% table(YTest(idx),label(idx),'VariableNames',...
%     {'TrueLabel','PredictedLabel'})
cnt = 0;
for i = 1:length(YTest)
    if YTest(i) == label(i)
        cnt  = cnt + 1;
    end
end
accuracy = cnt/length(YTest)


%%

X = [classical; hiphop; pop];
Y = [ones(25,1); 2*ones(25,1); 3*ones(25,1)];
rng(1);

CVMdl = fitcnb(X,Y,'Holdout',0.20);
CMdl = CVMdl.Trained{1};          % Extract trained, compact classifier
testIdx = test(CVMdl.Partition); % Extract the test indices
XTest = X(testIdx,:);
YTest = Y(testIdx);
```

```
idx = randsample(sum(testIdx),15);
label = predict(CMdl,XTest);
% table(YTest(idx),label(idx),'VariableNames',...
%     {'TrueLabel','PredictedLabel'})
cnt = 0;
for i = 1:length(YTest)
    if YTest(i) == label(i)
        cnt  = cnt + 1;
    end
end
accuracy = cnt/length(YTest)

%% fitcsvm multipal labels

X = [mj; soundgarden; beethoven];
Y = [ones(25,1); 2*ones(25,1); 3*ones(25,1)];
rng(1);
CVMdl = fitcecoc(X,Y,'Holdout',0.20);
CMdl = CVMdl.Trained{1};          % Extract trained, compact classifier
testIdx = test(CVMdl.Partition); % Extract the test indices
XTest = X(testIdx,:);
YTest = Y(testIdx);

idx = randsample(sum(testIdx),15);
label = predict(CMdl,XTest);
% table(YTest(idx),label(idx),'VariableNames',...
%     {'TrueLabel','PredictedLabel'})
% bar(label)
cnt = 0;
for i = 1:length(YTest)
    if YTest(i) == label(i)
        cnt  = cnt + 1;
    end
end
accuracy = cnt/length(YTest)

%%
```

```
X = [pearljam; soundgarden; alice];
Y = [ones(25,1); 2*ones(25,1); 3*ones(25,1)];
rng(1);
CVMdl = fitcecoc(X,Y,'Holdout',0.20);
CMdl = CVMdl.Trained{1};          % Extract trained, compact classifier
testIdx = test(CVMdl.Partition); % Extract the test indices
XTest = X(testIdx,:);
YTest = Y(testIdx);

idx = randsample(sum(testIdx),15);
label = predict(CMdl,XTest);
% table(YTest(idx),label(idx),'VariableNames',...
%     {'TrueLabel','PredictedLabel'})

cnt = 0;
for i = 1:length(YTest)
    if YTest(i) == label(i)
        cnt  = cnt + 1;
    end
end
accuracy = cnt/length(YTest)

%%

X = [classical; hiphop; pop];
Y = [ones(25,1); 2*ones(25,1); 3*ones(25,1)];
rng(1);
CVMdl = fitcecoc(X,Y,'Holdout',0.20);
CMdl = CVMdl.Trained{1};          % Extract trained, compact classifier
testIdx = test(CVMdl.Partition); % Extract the test indices
XTest = X(testIdx,:);
YTest = Y(testIdx);

idx = randsample(sum(testIdx),15);
label = predict(CMdl,XTest);
% table(YTest(idx),label(idx),'VariableNames',...
%     {'TrueLabel','PredictedLabel'})
% bar(label)
```

```
cnt = 0;
for i = 1:length(YTest)
    if YTest(i) == label(i)
        cnt  = cnt + 1;
    end
end
accuracy = cnt/length(YTest)

%%

Xtrain = [mj_train; soundgarden_train; beethoven_train];
Xtest = [mj_test; soundgarden_test; beethoven_test];
Ytrain = [ones(20,1); 2*ones(20,1); 3*ones(20,1)];
Ytest = [ones(5,1); 2*ones(5,1); 3*ones(5,1)];
[ind err] = classify(Xtest, Xtrain, Ytrain);
% table(Ytest,ind,'VariableNames',...
%     {'TrueLabel','PredictedLabel'})
cnt = 0;
for i = 1:length(Ytest)
    if Ytest(i) == ind(i)
        cnt  = cnt + 1;
    end
end
accuracy = cnt/length(Ytest)

%%

Xtrain = [soundgarden_train; pearljam_train; alice_train];
Xtest = [soundgarden_test; pearljam_test; alice_test];
Ytrain = [ones(20,1); 2*ones(20,1); 3*ones(20,1)];
Ytest = [ones(5,1); 2*ones(5,1); 3*ones(5,1)];
[ind err] = classify(Xtest, Xtrain, Ytrain);
% table(Ytest,ind,'VariableNames',...
%     {'TrueLabel','PredictedLabel'})
cnt = 0;
for i = 1:length(Ytest)
    if Ytest(i) == ind(i)
        cnt  = cnt + 1;
```

```
    end
end
accuracy = cnt/length(Ytest)

%%

Xtrain = [classical_train; hiphop_train; pop_train];
Xtest = [classical_test; hiphop_test; pop_test];
Ytrain = [ones(20,1); 2*ones(20,1); 3*ones(20,1)];
Ytest = [ones(5,1); 2*ones(5,1); 3*ones(5,1)];
[ind err] = classify(Xtest, Xtrain, Ytrain);
% table(Ytest,ind,'VariableNames',...
%     {'TrueLabel','PredictedLabel'})
cnt = 0;
for i = 1:length(Ytest)
    if Ytest(i) == ind(i)
        cnt  = cnt + 1;
    end
end
accuracy = cnt/length(Ytest)
```