AMATH 482 - Winter Quarter
Homework 3: Music Genre Identification

Vicky Norman

March 1, 2018

Abstract

This project explores the use of the K nearest neighbour and Naive Bayes algorithms to classify songs from different genres by taking training sets of 5 second clips of songs. The accuracy of each algorithm is presented for each case and for various numbers of features. The results show that there is not much improvement in the accuracy when all the songs come from different genres as apposed to the same genre.

1 Introduction and Overview

This project aims to explore how classification algorithms can be used on songs of different genres to make predictions about the genre of music that a particular song clip belongs to. Two different classification algorithms are used, and their results are discussed in three separate cases. The first case takes music from three different bands, of three separate genres to build a test and train set. A statistical model is built using each of the classification algorithms and then it is tested to see how well it classifies a new set of song clips from the same three bands. The second case is used to compare the accuracy of the genre classification by taking samples of music from three different bands within the same genre. This comparison is of interest to see if the classification algorithms are better at classifying songs which seemingly have less in common with each other.

The third case takes music from three different genres, but the clips come from a variety of different artists. This is the most interesting case since we can see if it's possible for the classifier to pick up on similarities in the music when the singer and even instruments are different. This is more similar to how humans can identify the genre of music just by listening to a song.

2 Theoretical Background

Naive Bayes Algorithm

The Naive Bayes classifier uses a set of supervised learning algorithms based on applying Bayes' theorem, so called because of the 'naive' assumption of independence between every pair of features. Bayes theorem is as follows:

$$\mathbb{P}(c_k|\mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}|c_k)\mathbb{P}(c_k)}{\mathbb{P}(\mathbf{x})}$$

1. $\mathbb{P}(c_k|\mathbf{x})$ The Posterior Probability - The probability of belonging to a class given a location in the coordinate system
2. $\mathbb{P}(\mathbf{x}|c_k)$ The Likelihood - The probability of being in a location given belonging to a certain class
3. $\mathbb{P}(c_k)$ The Prior Probability of the class

4. $\mathbb{P}(\mathbf{x})$ The Prior Probability of the predictor

The algorithm fits a Gaussian distribution for each of the points from the training set. The prediction is made by deciding which class maximises the posterior probability.

K nearest neighbours

The k-nearest neighbour algorithm is used to classify pattern data. It takes as input the k nearest neighbours of a point and outputs which class it is being assigned to. This type of algorithm is called instance-based learning as it is only classifying based on the limited information it is given, and all computation is done at the time of classification. A common problem with nearest neighbour classification is when the distribution of the classes is not even, for example when there are far more points in one class than the others, this can affect how test points are classified. Because of this problem it is sometimes beneficial to give each neighbour a weight of the distance from the current point, thus the class of the points closest to the test point will have more influence than those further away. As our data has equal sized classes this adjustment would be unnecessary.

For this algorithm to work it requires a set of data points and their corresponding 'labels', i.e. it must know which class each of the points belongs to. The algorithm then finds the Euclidean distance from the test point to the training points, the Euclidean distance is the straight-line distance between the two points in the Euclidean plane. It will then classify the test point based on a majority vote of the classification of the k nearest points. It's therefore important to think carefully about the value of k, choosing a large value of k can make the classification less skewed by neglecting the influence of any outliers but it will also make the boundaries of the classes less well defined. As our data set is relatively small it would not be sensible to choose a very high value for k as this would make the decisions fairly inaccurate.

3 Algorithm Implementation and Development

In order to use the in-built classification functions available in MATLAB we first need to extract some data to work with. After reading the files into MATLAB we want to manipulate the data so that we have just a few features from each song clip which we can then use to train the classifier algorithm. To reduce the data to these features we perform singular value decomposition on the data. Before doing this the data is transformed into a spectrogram which represents the time frequency of the data. The algorithm was performed with several different values of the v components of the svd and the results are compared below.

A summary of the algorithm steps is as follows:

1. Take the spectrogram of each song.
2. Split the song into a series of 5 second clips.
3. Reshape the data into a singular column vector.
4. Form a large matrix with one song per column.
5. Compute the SVD of the matrix.
6. Randomly allocate most of the songs to training data and the rest to test data.
7. Apply the Naive Bayes or k-nearest neighbor algorithm to the v values of the training data.
8. Use inbuilt MATLAB functions to predict the class of each test clip.
9. Calculate the accuracy of the prediction algorithm.
10. Repeat the training and test steps 1000 times with different random permutations of training and test data.

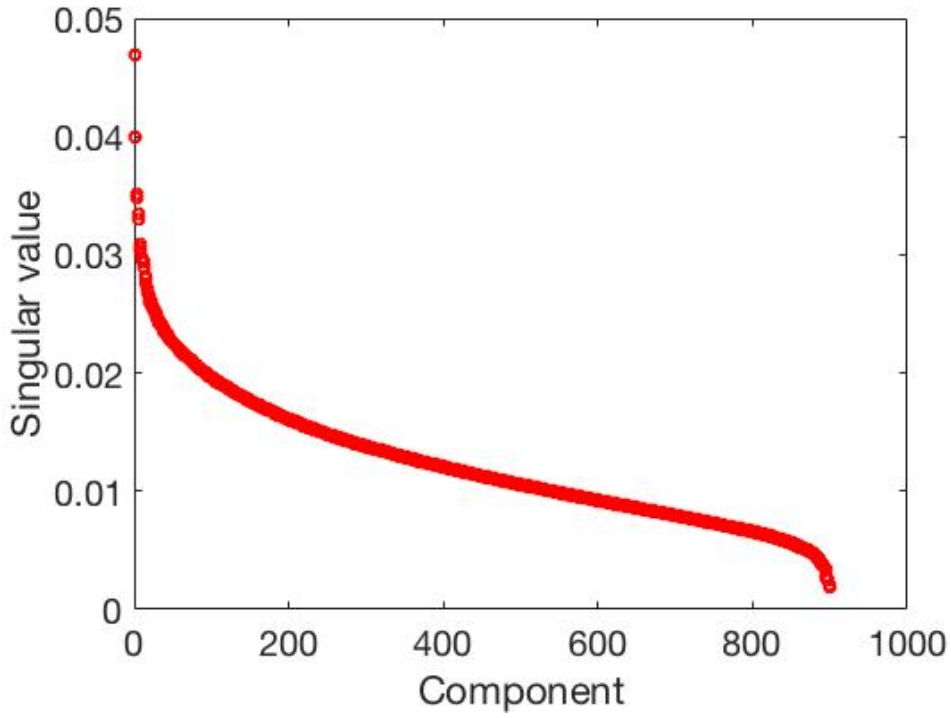

Figure 1: Singular value of each component, ordered by size. Case 1.

4 Computational Results

4.1 Band Classification

Figure 1 shows a plot of the singular values of each component in this case. Unlike previous cases we've seen there is no one component that has a magnitude much greater than the rest, suggesting that the songs do not have a particular feature that is very similar in all of the clips. The results of each algorithm are detailed in table 1. The highest accuracy was achieved in this case by using the nearest neighbour classifier with one nearest neighbour and 3 v values. This accuracy was around 81%.

4.2 The case for Seattle

Figure 2 shows the plot of each component for case 2. As with the first case there is no one feature that is of much higher magnitude than the rest. The results of each algorithm are detailed in table 1. The highest accuracy was achieved in this case by using the nearest neighbour classifier with one nearest neighbour and 4 v values. This accuracy was around 81.5%.

4.3 Genre Classification

Figure 3 shows the components of the song clips from case 1. As with case 1 and 2 there is no one feature that has a magnitude much higher than the rest. The results of each algorithm are detailed in table 1. The highest accuracy was achieved in this case by using the nearest neighbour classifier with one nearest neighbour and 5 v values. This accuracy was around 81%.

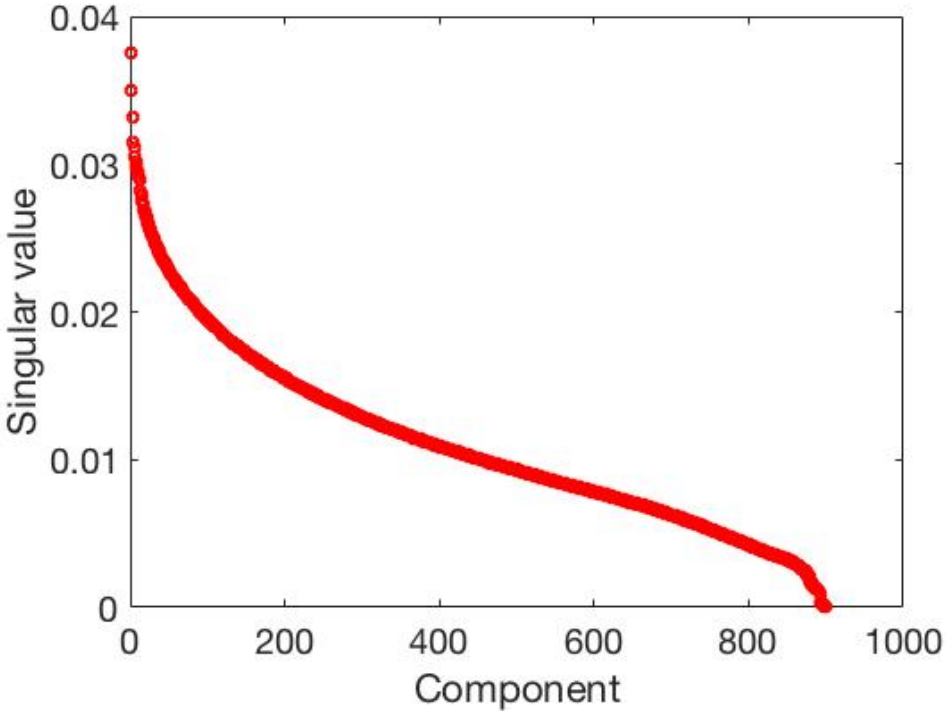

Figure 2: Singular value of each component, ordered by size. Case 2.

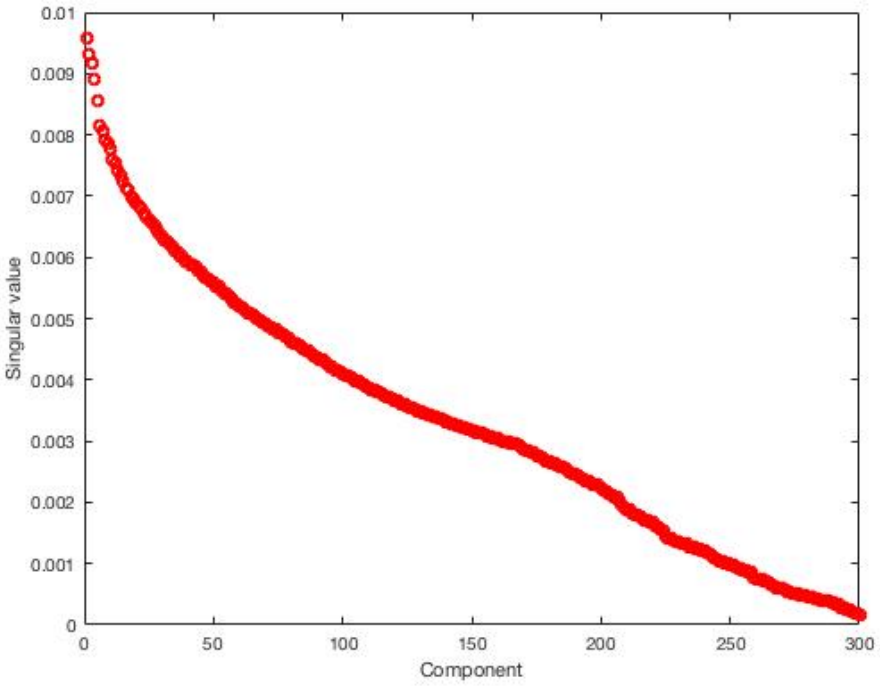

Figure 3: Singular value of each component, ordered by size. Case 3.

Table 1: Table of results from Naive Bayes and Nearest Neighbour classifiers for different number of V values and N values.

	Naive Bayes				Nearest Neighbour 1 neighbour			
V values	3	4	5	6	3	4	5	6
Case 1	0.5784	0.5827	0.5803	0.6067	0.8137	0.8135	0.8044	0.8113
Case 2	0.5728	0.5700	0.5955	0.6066	0.8144	0.8156	0.8104	0.8099
Case 3	0.5983	0.5953	0.6318	0.6190	0.7676	0.7991	0.8075	0.8033

	Nearest Neighbour 2 neighbours				Nearest Neighbour 3 neighbours			
V values	3	4	5	6	3	4	5	6
Case 1	0.6395	0.6643	0.6679	0.6691	0.6659	0.6910	0.6711	0.6585
Case 2	0.6324	0.6652	0.6723	0.6652	0.6607	0.6712	0.6568	0.6652
Case 3	0.6300	0.6709	0.6674	0.6905	0.5724	0.5992	0.6281	0.6331

Table 1 shows the results of the two learning algorithms for all cases. The results were calculated for various numbers of v values, from three values to six as detailed in the table. Overall the results were best for the nearest neighbour algorithm with only one neighbour. Moreover the results generally improved when more v values or features were used to train the classifier, this is intuitive however using more values slows down computation.

5 Summary and Conclusions

The highest accuracy results were achieved by the k nearest neighbour classifier with one neighbour, this is an interesting result since classifying the training data based on just the one nearest neighbour is an example of over fitting, generally over fitting during the learning algorithm can cause the classifier to incorrectly classify the test data. Since the data set used here is relatively small this could explain why over fitting hasn't been an issue here.

Interestingly the accuracy results were not drastically different in each of the three cases. This would suggest that although one would think that classifying songs of different genres would be easier than classifying songs within the same genre for the learning algorithms used it doesn't make that much difference.

6 Appendix

A

Appendix A MATLAB functions used and brief implementation explanation.

- s = spectrogram(x) returns the short-time Fourier transform of the input signal, x. Each column of s contains an estimate of the short-term, time-localized frequency content of x.
- s = svd(A) returns the singular values of matrix A in descending order.
- p = randperm(n) returns a row vector containing a random permutation of the integers from 1 to n inclusive.
- X = ones(sz1,...,szN) returns an sz1-by-...-by-szN array of ones where sz1,...,szN indicates the size of each dimension. For example, ones(2,3) returns a 2-by-3 array of ones.
- Mdl = fitcknn(Tbl,Name,Value) fits a model with additional options specified by one or more name-value pair arguments, using any of the previous syntaxes. For example, you can specify the tie-breaking algorithm, distance metric, or observation weights.
- Mdl = fitcnb(Tbl,ResponseVarName) returns a multiclass naive Bayes model (Mdl), trained by the predictors in table Tbl and class labels in the variable Tbl.ResponseVarName.
- cpre = predict(nb,test) classifies each row of data in test into one of the classes according to the NaiveBayes classifier nb, and returns the predicted class level cpre. test is an N-by-nb.ndims matrix, where N is the number of observations in the test data. Rows of test correspond to points, columns of test correspond to features. cpre is an N-by-1 vector of the same type as nb.CLevels, and it indicates the class to which each row of test has been assigned.

B

Appendix B MATLAB codes The code below was used to perform the classification algorithm. It can be modified to classify different song clips and use different learning algorithms.

```
monkeys = [dir('/Users/vick/Docs/AMATH482/hw3/ArcticMonkeys')];

Y = [];

for i = 4:13
    [y, Fs] = audioread(monkeys(i).name);
    for j = 1:30
        start = 44100*j;
        fin = start + 44100*5;
        yclip = y(start:fin,1);
        z = spectrogram(yclip);
        ry = reshape(z,[],1);
        Y = [Y, ry];
    end
end


%%

eminem = [dir('/Users/vick/Docs/AMATH482/hw3/Eminem')];

Y1 = [];
```

```
for i = 4:13
    [y, Fs] = audioread(eminem(i).name);
    for j = 1:30
        start = 44100*j;
        fin = start + 44100*5;
        yclip = y(start:fin,1);
        z = spectrogram(yclip);
        ry = reshape(z,[],1);
        Y1 = [Y1, ry];
    end
end

%%

SClub = [dir('/Users/vick/Docs/AMATH482/hw3/SClub7')];

Y2 = [];

for i = 3:12
    [y, Fs] = audioread(SClub(i).name);
    for j = 1:30
        start = 44100*j;
        fin = start + 44100*5;
        yclip = y(start:fin,1);
        z = spectrogram(yclip);
        ry = reshape(z,[],1);
        Y2 = [Y2, ry];
    end

end

%%

X = [Y Y1 Y2];

[u, s, v] = svd(X,'econ');

%%

plot(diag(s)/sum(diag(sc)), 'ro', 'Linewidth', [2]);
xlabel('Component')
ylabel('Singular value')

%%

plot3(v(1:300,4),v(1:300,5),v(1:300,6),'ro')
hold on
plot3(v(301:600,4),v(301:600,5),v(301:600,6),'bo')
hold on
plot3(v(601:900,4),v(601:900,5),v(601:900,6),'go')

%%
```

```
plot(v(1:300,3),v(1:300,2),'ro')
hold on
plot(v(301:600,3),v(301:600,2),'bo')
hold on
plot(v(601:900,3),v(601:900,2),'go')


%%

v1 = real(v(1:300,1:6));
v2 = real(v(301:600,1:6));
v3 = real(v(601:900,1:6));

accuracy = [];

for i = 1:1000

q1 = randperm(300);
q2 = randperm(300);
q3 = randperm(300);

xtrain = [v1(q1(1:290),:); v2(q1(1:290),:); v3(q1(1:290),:)];
xtest = [v1(q1(291:end),:); v2(q2(291:end),:); v3(q3(291:end),:)];

labels = [ones(290,1); 2*ones(290,1); 3*ones(290,1)];
labels2 = [ones(10,1); 2*ones(10,1); 3*ones(10,1)];

kn = fitcknn(xtrain, labels, 'NumNeighbors', [3]);

pre = kn.predict(xtest);

score = 0;

for i = 1:length(pre)
    if labels2(i) == pre(i)
        score = score + 1;
    end
end

accuracy = [accuracy score/length(pre)];

end

mean(accuracy)
```