A Rough Path Musical Genre Classifier

Victor Gardner

10 December 2021

Abstract

Independent project report. In this paper, I train a neural net to classify songs into one of ten genres. In order to extract features from the raw waveform, I perform the signature transform inspired by rough path theory on the raw data.

1 Introduction

The raw data of time series is not inherently interesting on its own. Rather, when analyzing a time series, we typically look for trends, correlation between increments of the time series, or periodicity of the series, all of which can occur together.

While these methods work well for time series prediction, if we are rather faced with a time series classification problem, things become more difficult. For example, in [GT02], the authors train classification algorithms to learn to which of ten different genres a piece of music belongs to. The main problem overcome in the paper is not time series modeling of the waveform of a piece of music. Rather, it is extracting features from the raw time series data of the waveform from which a machine learning algorithm can be trained.

In [GT02], the authors extract feature sets corresponding to pitch, timbre, and rhythm before attempting to train a musical genre classifier. Compared to attempting to learn from the raw data, this seems like a high-level approach to musical genre classification. Moreover, it seems tailor-maid for music classification, and not generalizable to other time-series classification problems. I was curious to see if there is some middle-ground between the possibly senseless enterprise of learning on raw time series data and learning on high-level, tailor-made features extracted from the data. Ideally, this middle ground would be more generalizable to other time series classification problems.

In this paper, I attempt to do just that. In order to extract learnable features from time series data, I use the signature transform from rough path theory on course time steps on the waveform. I then attempt to use these extracted features to train a neural net to classify songs into one of ten genres. It should be noted that the signature transform has found use extracting features for neural nets before, as in [Gra13], so I am not completely in the dark.

The next section of this paper gives a brief introduction to rough path theory, and in particular, to the signature transform. Next, I train a neural net on the features extracted from the signature transform. The feature extraction and the neural net are described in section 3. In section 4, I describe the dataset I used and the results of my neural net, comparing those results to the results from [GT02] and the results from other algorithms I trained on the signature transform of the data. Finally, in section 5, I discuss the errors and mistakes I made along the way. The code I wrote at attached at the end of this paper after the bibliography.

2 A Brief Overview of Rough Path Theory

2.1 Theory

The following exposition is largely adapted from the first two chapters of [PKF20].

Suppose that we are trying to understand a dynamical system of the form

$$\dot{Y}_t = Y_t \, dX_t \tag{1}$$

Then, we can naïvely write

$$Y_t = Y_0 + \int_0^t Y_s \, dX_s. \tag{2}$$

For this integral, if X and Y are too rough (Holder tontinuous of exponent less than $1/2$), the usual constructions of the integral produce nonsense. This is where rough path theory comes in.

At this point, for simplicity, we will assume that Y, as a process, is controlled by X, and that $X \in \mathcal{C}^\alpha([0,T])$ for some $\alpha \in (1/3, 1/2)$ (where $\mathcal{C}^\alpha$ represents the space of Hölder-continuous functions of exponent α). The general theory follows along the lines presented here. For Y to be "controlled" by X means that there is some process $Y' \in \mathcal{C}^\alpha([0,T])$ and some process $R^Y \in \mathcal{C}^{2\alpha}([0,T])$ such that

$$Y_{s,t} = Y'_s X_{s,t} + R^Y_{s,t}, \tag{3}$$

where $X_{s,t}$ is defined as the increment $X_t - X_s$. Then, for small time-steps $|t-s|$, we may write

$$Y_{s,t} = \int_s^t Y'_s X_{s,r}\, dX_r + \int_s^t R^Y_{s,r}\, dX_r. \tag{4}$$

Due to the regularity of R^Y and X, if we construct the second integral in (4) using Riemann-Stiljes type techniques, this term disappears. Therefore, the only thing we need to figure out is

$$\int_s^t X_{s,r} \otimes dX_r. \tag{5}$$

At this point, the reader likely has the question: "what's with the tensor?" Essentially, when X takes values in a Banach space, we need to capture the way every component of X interacts with every other component of X in order to develop a robust solution theory. Thus far, I have left out tensors from my integrals for the clarity of this informal heuristic discussion, but from now on I will leave them in.

Rough path theory takes this step by postulating the secondary, two-parameter process $\mathbb{X}_{s,t}$, and then defining the integral in (5) as

$$\mathbb{X}_{s,t} = \int_s^t X_{s,r} \otimes dX_r. \tag{6}$$

It is important to remember that the integral on the right is defined in terms of the process $\mathbb{X}$, *not* the other way around. Once $\mathbb{X}_{s,t}$ satisfies certain algebraic conditions, such as Chen's relation:

$$\mathbb{X}_{s,t} - \mathbb{X}_{s,u} - \mathbb{X}_{u,t} = X_{s,u} \otimes X_{u,t}, \tag{7}$$

a robust and path-wise solution theory for (3) can be constructed.

The question presents itself: What if $X \in C^\alpha([0,T])$ for $\alpha \in (0, 1/3]$? That is, how can we construct a solution theory for paths of arbitrarily low regularity? For this answer, we turn to the signature transform of the path X. The signature transform of the path X is defined as

$$S(X)_{s,t} = \left(\mathbf{1}, X_{s,t}, \mathbb{X}_{s,t}, \int_{s<t_1<t_2<t} X_{s,t_1} \otimes dX_{t_1} \otimes dX_{t_2}, \dots \right). \tag{8}$$

However, because each level of the signature transform beyond $\mathbb{X}_{s,t}$ can be considered as a path controlled by X, only $\mathbb{X}_{s,t}$ needs to be defined; the rest may be calculated. Essentially, with the information contained in the signature transform, we may define solutions for paths of arbitrarily low regularity controlled by paths that are themselves of arbitrarily low regularity.

It is this aspect of rough path theory - the signature transform - that is of interest in extracting features from time series for machine learning applications.

2.2 Application

Applications of the Signature of a path come from two properties of the signature. The first of these properties is known as the shuffle product. Before we define the shuffle product, we must first note that addition of signatures can be defined coordinate-wise, and products can be defined using the linearity of the tensor product.

While so far I have used the full signature transform, in practice for applications we are often interested in iterated integrals of specific coordinates. If $X : [0, T] \to \mathbb{R}^d$ is a path and $I = (i_1, \dots, i_k)$ is a multi-index with each $i_\ell \in \mathbb{R}^d$, then we define

$$S(X)^I_{s,t} = \int_{s<t_1<\cdots<t_k<t} X^{i_1}_{s,t_1}\, dX^{i_2}_{t_1} \dots dX^{i_k}_{t_k} \tag{9}$$

We are now ready to define the shuffle product, as in [IC16]. Let $I = (i_1, \dots, i_k)$ and $J = (j_1, \dots, j_m)$ be two multi-indices. Define the multi-index R as

$$(r_1, \dots, r_k, r_{k+1}, \dots, r_{k+m}) = (i_1, \dots, i_k, j_1, \dots, j_m). \tag{10}$$

The shuffle product $I \sqcup\!\sqcup J$ is defined as the set

$$I \sqcup\!\sqcup J = \{(r_{\sigma(1)}, \dots, r_{\sigma(k+m)}) \colon \sigma \in \text{Shuffle}(k, m)\} \tag{11}$$

where Shuffle(k, m) is itself defined as the set of permutations on $k + m$ elements such that $\sigma^{-1}(1) < \sigma^{-1}(2) < \cdots < \sigma^{-1}(k)$ and $\sigma^{-1}(k + 1) < \cdots < \sigma^{-1}(k + m)$ (in other words, σ literally splices together I and J).

Now that we know what the shuffle product is, it's time for our first Theorem, taken from [IC16].

Theorem 1. For a path $X : [s,t] \to \mathbb{R}^d$, and two multi-indices $I = (i_1, \dots, i_k)$ and $J = (j_1, \dots, j_m)$, where each i_ℓ and j_ℓ are in $\{1, \dots, d\}$, we have the following identity:

$$S(X)^I_{s,t} \otimes S(X)^J_{s,t} = \sum_{K \in I \sqcup J} S(X)^K_{s,t}. \tag{12}$$

□

In other words, products of lower-order terms in the signature can be represented as the sum of higher-order terms in the signature. Those familiar with the Stone–Weierstrass theorem from real analysis may see where this is going.

This next theorem is taken from [Lyo14].

Theorem 2. Coordinate iterated integrals, as features of paths, span an algebra that separates Signatures and contains the constants.

□

These two theorems together suggest that linear combinations of the coordinate iterated integrals that comprise the signature transform should be able to approximate any (continuous) function on spaces of continuous paths. In other words, these two theorem are the justification for the algorithm outlines in the next section.

3 Methods

In this section, I discuss the model I created, and I explain the choice of hyper-parameters I used.

The classification problem in machine learning is essentially tying to approximate a non-linear function as best as possible. That is, if $F : X \to C$ is a classifier from a set of features X to a set of classes C, then F is certainly a non-linear function. If similar features implies similar classification, then F must ultimately be continuous. Using this fact, the justification from the previous section that coordinate iterated integrals as features of paths separate the signatures associated with those paths (Theorem 2), and the fact from the previous section that non-linear functions of the signature can be approximated by linear combinations of the signature (Theorem 1), I developed the following model.

The first step of the model is feature extraction. For this step, after truncating a waveform at 600,000 samples, I split this time series into 300 sub-intervals of equal length. I calculated the first two non-constant levels of the signature transform on each subinterval using a Riemann-Stiljes sum approximation, summing across each data point in the sub-interval. Assuming that a waveform has locally symmetric ascending and descending parts, the error between the approximation and the true value of the integral should be minimal. Because this data is one-dimensional, the first two levels of the signature transform are just $X_{s,t}$ and $\int_s^t X_{s,r}\, dX_r$.

The main constraint on the number of levels of the signature transform I calculated was computation time. Unfortunately, calculating deeper levels of the signature transform would have taken days of computing time, so I was stuck with two levels.

Once this feature extraction was complete, I built a neural net as follows. The input layer has 600 nodes (one for each of the two levels of the signature transform calculated on each sub-interval). In order to create local linear combinations of the signature transform (that is, linear combinations of some element with its neighbors), I applied a 1 dimensional convolution layer of length 5; I had a width-2 padding layer to each side to make sure this layer output a set of 600 features. After the convolution layer, I applied a sigmoid activation function. The next layer was a linear layer of domain-dimension 600 and range-dimension 128. This was done so that global linear combinations of the signature transform could be introduced to the model. After applying a sigmoid activation function once more, the final layer was a linear layer with domain-dimension 128 and range-dimension 10, for each of the ten genres. Finally, I applied the softmax function to this final layer in order to create a probability distribution on the ten possible genres; the genre with the highest probability is considered the prediction of the neural net.

The sigmoid activation function was chosen after some experimentation. Other activation functions, such as relu, produced NANs while running the neural net. This is likely because the numbers produced at each layer of the neural net were too big. The sigmoid activation function essentially normalized the weights to be between 0 and 1, acting as something similar to a Gaussian CDF.

The number of nodes for the linear layers was chosen to create a neural net that acted like a funnel. The important part, from a theoretical perspective, was to create linear combinations of the elements of the signature transform.

To train the model, I used the standard backpropogation error with a mean-squared error (MSE) loss function. The MSE loss function was used because I was trying to train on the distance between to probability measures (the probability measure generated by the neural net and the probability measure that is a dirac mass on the correct genre), and the MSE is equivalent to the Wasserstein earth-mover's metric, which is a natural metric on the space of probability measures. To reduce overfitting, I shuffled the training set every epoch.

The computation time of backpropogation is the reason that I split the time interval into 300 equal-length sub-intervals. While I was curious to see if an algorithm could learn on coarse-resolution signature transforms, when I tried to make the sub-intervals the resolution of the waves in the waveform, training my algorithm took all night, severely limiting the possibility of experimenting with other aspects of the neural net.

To summarize, the model is as follows:

1 Extract features using signature transform

  1.1 split time interval into 300 sub-intervals

  1.2 perform signature-transform 2 levels deep

2 Neural Net

  2.1 input layer: 600 nodes

  2.2 1D Convolution layer

  2.3 sigmoid activation

  2.4 linear layer: 600 to 128

  2.5 sigmoid activation

  2.6 linear layer: 128 to 10

  2.7 softmax to create probability distribution

3 Train neural net with MSE error.

  3.1 shuffle training set every epoch

4 Experiments

4.1 Setup and Metrics

The dataset I used was the same one used in [GT02], available as the GTZAN dataset in PyTorch. This dataset consists of 1,000 thirty-second clips of songs from ten genres; each genre has 100 songs in the dataset. The genres are as follows: blues, classical, country, disco,

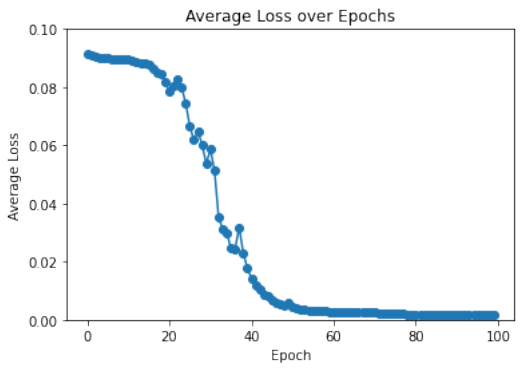


Figure 1: Average loss per epoch while training the neural net.

hiphop, jazz, metal, pop, reggae, rock. The tracks are all 22050Hz Mono 16-bit audio files in .wav format [Leb].

The dataset was split into train and test sets, where the train set had 700 songs and the test set had 300 songs. The genres were evenly represented in both the train and set sets, so each genre had 70 songs in the training set and 30 songs in the testing set.

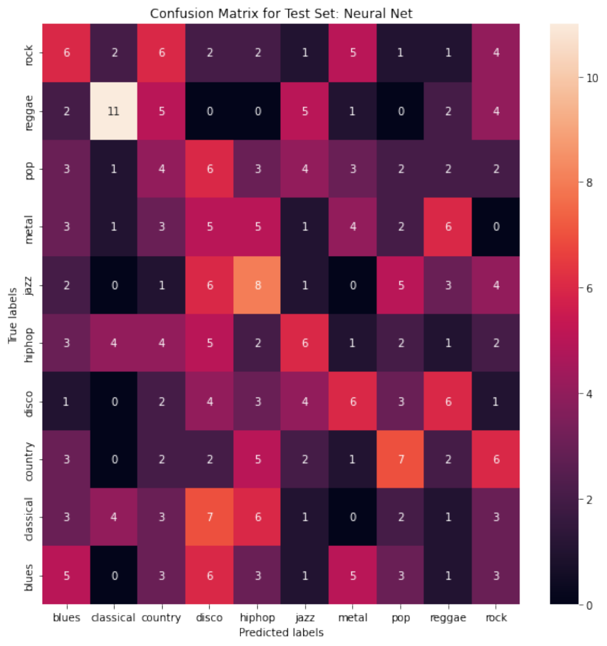
In order to determine just how effective my neural net is at categorizing musical genre, I created two naive-learning algorithms in addition to the nerual net. One is a k-means clustering algorithm for time series that uses dynamic time warping (DTW) as a distance function, and the other is a simple logistic regression. I performed both of these algorithms on the signature transform features, both because I wanted to see how other algorithms would do on the signature-features and because training on the raw data is prohibitively expensive. Moreover, I used the paper [GT02] as a reference for how good a genre classifier can be when learned on coarse features tailor-made for the problem at hand.

For the k-means unsupervised classifier, I used the DTW metric because it's merely the L^2-distance of aligned paths, and the signature transform is merely a path (albeit a complex one). I initialized the algorithm with 5 random restarts, choosing the output with the lowest k-means objective.

For each model I trained, I created confusion matrices and accuracy scores. For the neural net, I also tracked the average loss over each epoch.

4.2 Experimental Results

After some experimentation, I decided to train the neural net for 100 epochs. The average loss per epoch is graphed in Figure 1.



True \ Predicted	blues	classical	country	disco	hiphop	jazz	metal	pop	reggae	rock
rock	6	2	6	2	2	1	5	1	1	4
reggae	2	11	5	0	0	5	1	0	2	4
pop	3	1	4	6	3	4	3	2	2	2
metal	3	1	3	5	5	1	4	2	6	0
jazz	2	0	1	6	8	1	0	5	3	4
hiphop	3	4	4	5	2	6	1	2	1	2
disco	1	0	2	4	3	4	6	3	6	1
country	3	0	2	2	5	2	1	7	2	6
classical	3	4	3	7	6	1	0	2	1	3
blues	5	0	3	6	3	1	5	3	1	3

Figure 2: Confusion matrix for the Neural Net on the training set.

At this point, the reader may suspect that I have drastically overfit my model. In fact, once we see the prediction results in confusion-matrix form for both training and test sets (Figures 2 and 3), this conclusion seems to be supported.

The neural net produced an accuracy of 0.19 on the test set and an accuracy of 0.987 on the training set. Interestingly, when I trained the neural net for newer epochs in order to reduce overfitting, the accuracy on the test set *decreased*. I am not sure what this means for neural nets trained on features extracted from signature transforms.

For comparison, when I ran a k-means clustering algorithm on the training set and assigned labels to the the majority class represented in the cluster, the accuracy was 0.236. The accuracy of a logistic regression on the training set was 1.0, while the accuracy of the logistic regression on the test set was 0.173. All of these accuracies pale in comparison to the accuracy achieved in [GT02], where the authors achieved an accuracy of 0.61 from extracting music-specific features such as timbre, beat, and pitch. Confusion matrices for each of these algorithms in addition to bar graphs for the neural net are contained in the an appendix after the bibliography but before the code (outside of the 10 page limit. Graphs take up a lot of room.)

However, all of these scores are significantly better than random chance. This suggests that there *is* useful information contained in the signature transform. However, I am not sure

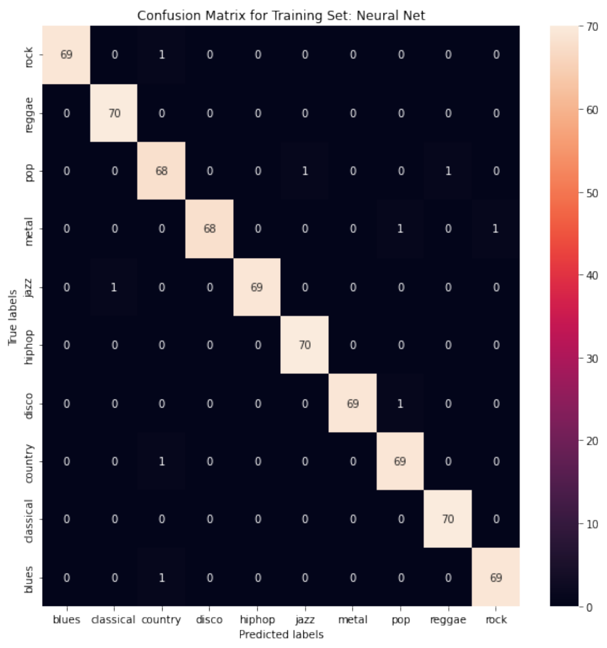


Figure 3: Confusion matrix for the Neural Net on the test set.

what that information is. Interpretability of neural nets and other ML algorithms trained on the signature transform does seems like an interesting problem, however, although at the moment I am not mathematically mature enough to tackle it.

5 Errors and Mistakes

As mentioned previously, the main issue I encountered was running time. I'll have to learn more about parallel programming, but it would be interesting to see what happens when the signature transform is done on shorter timescales and to a greater depth. For musical applications specifically, the signature transform has already been used to create efficient compression algorithms in [TJL05], so we know that the signature is capable of storing valuable information about a piece of music. However, extracting meaning from that information seems to be a different issue entirely.

References

[Gra13] Benjamin Graham. Sparse arrays of signatures for online character recognition. 2013.

[GT02] Perry Cook George Tzanetakis. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293–302, 2002.

[IC16] Andrey Kormilitzin Ilya Chevyrev. A primer on the signature method in machine learning. *ArXiv*, 2016.

[Leb] Jakob Leben. Gtzan genre collection. `http://marsyas.info/downloads/datasets.html`. Accessed 8 December 2021.

[Lyo14] Terry Lyons. Rough paths, signatures and the modelling of functions on streams. *Arxiv*, 2014.

[PKF20] Martin Hairer Peter K. Friz. *A Course on Rough Paths With an Introduction to regularity Structures*. Springer, 2 edition, 2020.

[TJL05] Nadia Sidorova Terry J. Lyons. Sound compression – a rough path approach. 2005.

6 Appendix: Confusion Matrices for K-means and Logistic Regression

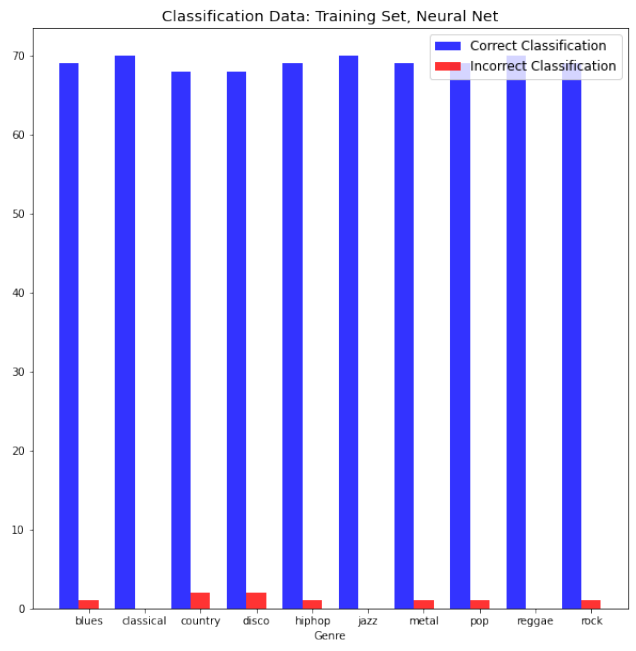


Figure 4: Bar chart for the Neural Net on the training set.

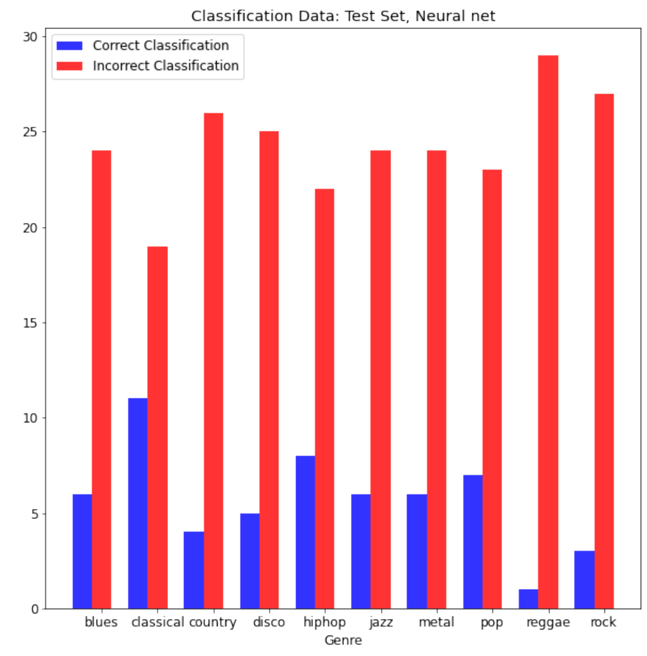


Figure 5: Bar chart for the Neural Net on the test set.

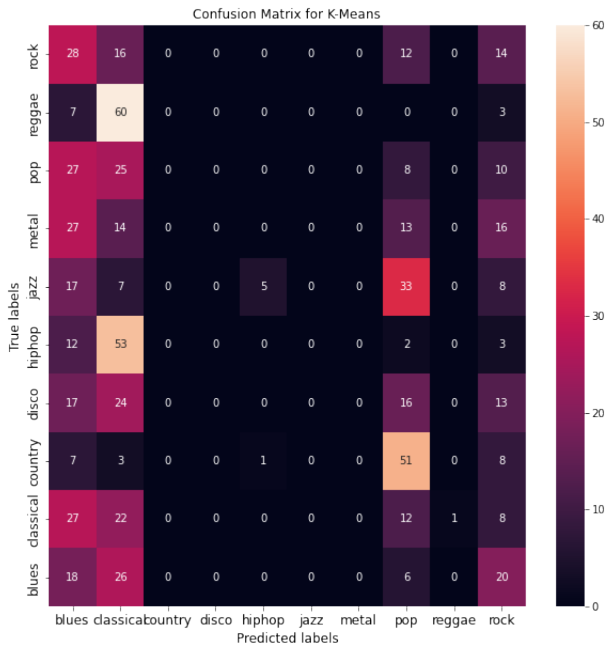


Figure 6: Confusion matrix for k-means on training set

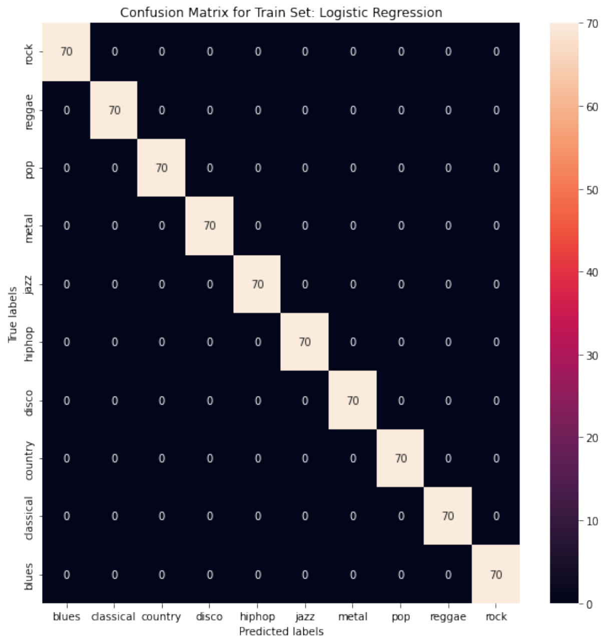


Figure 7: Confusion matrix for logistic regression on training set

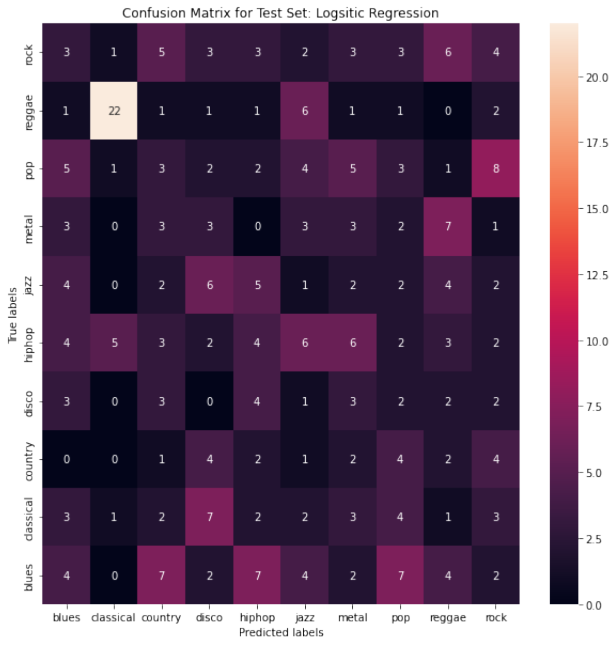


Figure 8: Confusion matrix for logistic regression on test set.