MUSIC MOOD CLASSIFICATION

BY NIKHITA KANDIKONDA and VENKATA SATYA PRANEETH TAMVADA

1. INTRODUCTION

This report documents the classification of music genre developed by Nikhita and Praneeth (Group 16) for Machine Learning final project presentation.

There are many characteristics of a song for which it gets classified into its respective genre. Energy, Instrumentalness, Bits per Minute, Loudness, Tempo etc. are some of the key features for genre classification. The aim of our project is to classify each input (music track) with different features as mentioned above, into its respective category like 'Party', 'Dinner', 'Workout' and 'Sleep'. To implement this idea, we collected the music data from Kaggle website. The dataset contains different attributes like Tempo, BPM, Loudness and other relevant features which helps classify our inputs. We began the analysis by cleaning the data and splitting the dataset to test and train purposes. Afterwards, we implemented two machine learning algorithms: Multi-Layer Perceptron and Support Vector Machines to classify the inputs into their respective genres based on input features. Subsequently after building the models, we used evaluation metrics like Accuracy, Confusion Matrix and Classification report. Then, we assessed the models based on best results given by the above-mentioned metrics.

The document is organized as follows: Description of dataset and algorithms, Experimental setup, Results, Summary, References and appendix.

2. DESCRIPTION OF DATA SET

The source of our dataset is Kaggle website and Spotify music application. As we are implementing a classification problem, the four classes we choose are:

Dinner: Songs that sound good when played in a dinner setting or at a restaurant.

Sleep: Songs that promote sleep when they are played.

Party: Songs that sound good when played at a party.

Workout: Songs that sound good when one is exercising/ working out.

Description of our dataset columns:

Input Variables:

Id: The id of the Spotify track. (String)

Name: the name of the track. (String)

Artist: The artist name of the track. (String)

Acousticness: control sound so that you can hear only the sounds you want to hear. (Numeric)

Danceability: the dancing beats for the track. (Numeric)

duration_ms: duration of the track. (Numeric)

Energy: energy measure of the track. (Numeric)

Instrumentalness: the level of instruments used in the track. (Numeric)

Key: describes the scale of the track. (Numeric)

Liveness: the liveness measure of the track. (Numeric)

Loudness: the loud features of the track. (Numeric)

Mode: the mode scale of the track. (Numeric)

Speechiness: the amount of wordings or lyrics in a track. (Numeric)

Tempo: the tempo feature of the track. (Numeric)

time_signature: this is a feature used for creating track with a old time music and mood. (Numeric)

Valence: defined by Spotify as a measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. (Numeric).

***Output Variable (Target)**:*

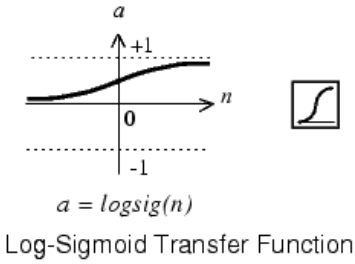
Genre: the genre of the track.

3. DESCRIPTION OF ALGORITHMS

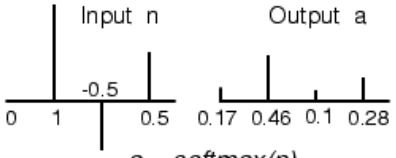
3.1 Neural Network

The study of human brain is 1000 years old. Human brain is a collection of thousands of neurons which harness our thinking process. A neuron (neuron or nerve cell) is an electrically excitable cell that processes and transmits information through electrical and chemical signals via the synapses. The connections between neurons carry an activation signal of changing magnitude. The first neural networks came in 1943 where they modeled a simple neural network with electrical circuits. A neural network system is a collection of inputs, outputs and simple processing units called the neurons. The processing units have internal parameters called weights and altering these weights would help us achieve desired output. These weights summed up with the bias are passed through the activation function which is called the transfer function. The neural network consists of a network of interconnected neurons present in different layers called

the input layer, hidden layer and output layer. The input layer has input neurons which transfer data via synapses to the hidden layer, and similarly the hidden layer transfers this data to the output layer via more synapses. The Transfer function used in the hidden layer for training is log-Sigmoid and output layer transfer function is softmax, which is used for classification.



Log-Sigmoid Transfer Function



Softmax Transfer Function

The output of the hidden layer is input to the output layer. The output of the hidden layer and the output of output layer is given by

$$a^1 = logsig(W^1 p + b^1) \tag{3.1}$$

$$a^2 = softmax(W^2 a^1 + b^2) \tag{3.2}$$

Where,

p – input to network

W^1, b^1 $-$ $weights\ and\ bias\ of\ first\ layer$

$W^2, b^2 - weights\ and\ bias\ of second\ layer$

The architecture of the network is shown below

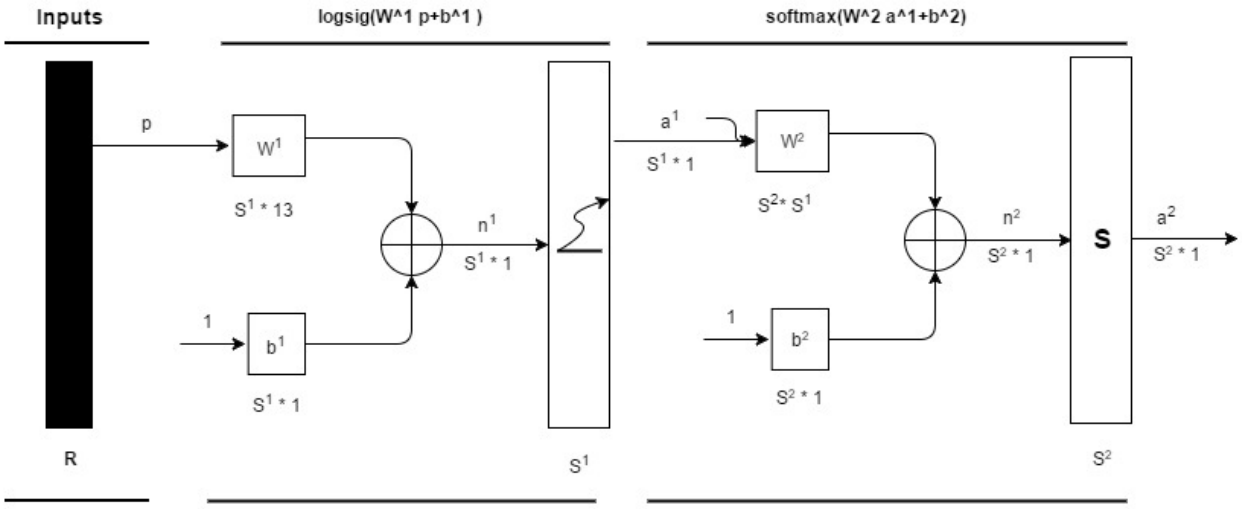


Figure 3.1

The forward propagation of the network is done using equation 3.1 and 3.2.

Next the network tries to adjust the network parameters to minimize the mean square error $E[(t-a)^2]$ by backpropagating the errors. We have used the steepest descent algorithm to optimize the error surface. The sensitivities of each layer are backpropagated where, the sensitivities of the output layer and hidden layer are given by equation

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t}-\mathbf{a}) \tag{3.3}$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1}, \text{ for } m = M-1, \ldots, 2, 1. \tag{3.4}$$

Finally, the weights and bias are updated in each iteration using the following equation

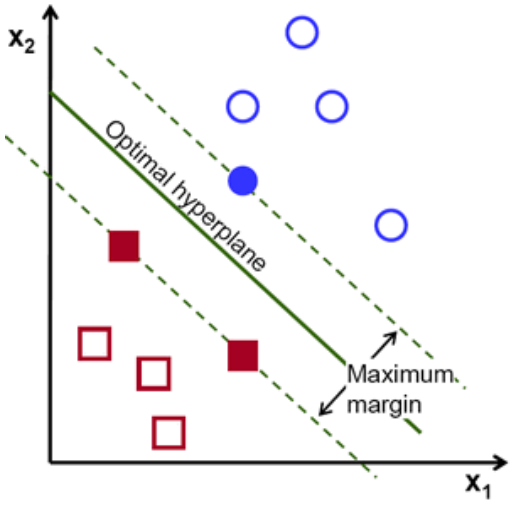
$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha\mathbf{s}^m(\mathbf{a}^{m-1})^T,$$

$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha\mathbf{s}^m. \tag{3.5}$$

The network is trained until it converges and the final weight and bias are used to form the decision boundaries.

3.2 Support Vector Machine

Support Vector Machine is a supervised learning algorithm, which aims at classification of data. The features are plotted as a point in n-dimensional space and classification is done by drawing a hyperplane between the classes. The right hyperplane is identified by finding a line that maximizes the margin of data where, margins are the lines passing through the nearest point of the class. From the figure below, the dotted lines are the margins and the optimal hyperplane is the decision line.



For any given dataset, SVM tries to identify the hyperplane W·x+b=0 and the equations of margins given as W·x+b=−1 for negative class and W·x+b=1 for positive class.
The margin equations can be written as

$$y_i(\text{W}\cdot X_i\text{+b}-1) >= 0, \text{ where } \{X_i, y_i\} \in R^n \text{ and i = 1....n}$$

The width of the margins is given by $\frac{2}{|w|}$. The objective is to maximize the width of the margin which is equal to minimizing |w|. This give us the following optimization problem:

$$\text{Minimize in (w,b) } \|w\|$$

$$\text{subject to } y_i(\text{W}\cdot X_i\text{+b}-1) >= 1 \quad \text{(for any i=1,...,n)}$$

We introduce Lagrange multiplier α_i to solve convex optimization and we get Wolfe dual of the optimization problem:

$$L(w,b) = \frac{1}{2} \| \text{W} \|^2 - \sum_{i=1}^{n} \alpha_i y_i [(x_i^T W + b) - 1]$$

We find the values of W and b by taking the derivatives with respect to W and b and equate them to 0. After we solve them we get the values of W and b as

$$w = \sum_{i=1}^{n} \alpha_i y_i X_i$$

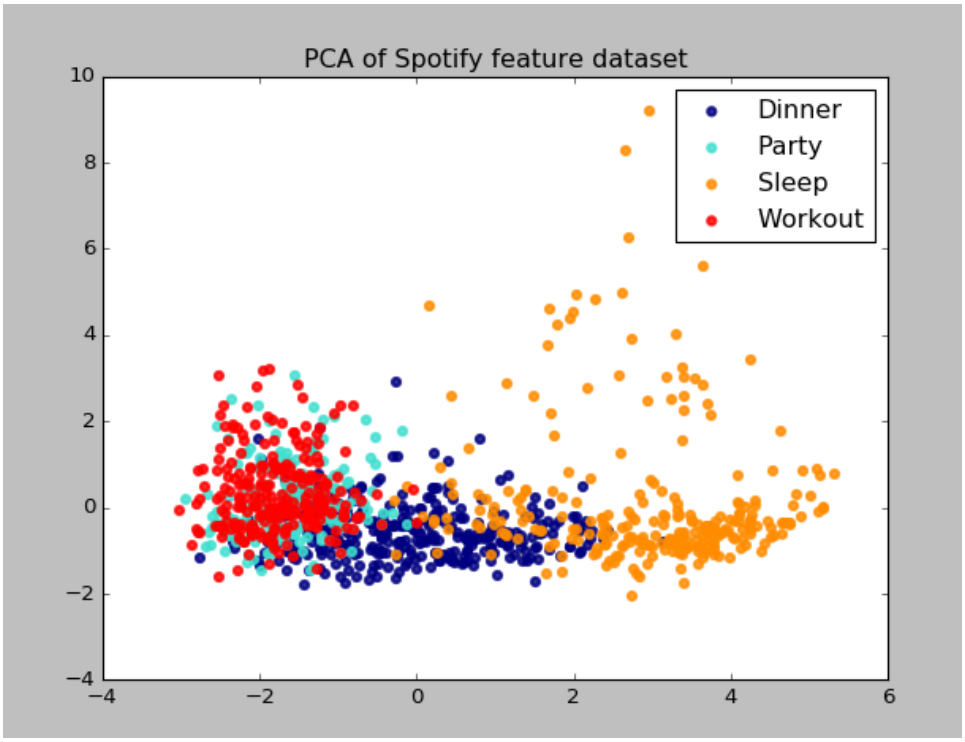
$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

For non-linear data, kernel transformations are applied to make the data linearly separable and SVM is applied on the transformed data.

4. EXPERIMENTAL SETUP

The data has 13 features which are categorized into 4 classes. The data is reduced to 2-d using sklearn PCA decomposition.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2) #Create a PCA object that reduces the dimentions to 2
X_r = pca.fit(X_train).transform(X_train) #fit the data
Y = np.array(y_train)
colors = ['navy', 'turquoise', 'darkorange', 'red']
lw = 2
target_names = ['Dinner','Party','Sleep','Workout']
for color, i, target_name in zip(colors, [1, 2, 3, 4], target_names):
    plt.scatter(X_r[Y == i, 0], X_r[Y == i, 1], color=color, alpha=.8, lw=lw,
                label=target_name)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.title('PCA of Spotify feature dataset')
plt.show()
```



From the 2-d distribution of the data, it can be seen clearly that Dinner and Sleep categories are clearly separable but party and workout categories overlap.

Sklearn train_test_split package was used to split the data into training and testing with a test size of 0.3 and the stratified split was used to divide the data set into equal distribution of classes in both the splits.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(audio_features, audio_genere,
test_size=0.3, random_state=1,stratify=audio_genere)
```

The data was then normalized using preprocessing.StandardScaler() function of the Sklearn package. The data is transformed using

Standardization: $z=(x-\mu)/\sigma$

with mean: $\mu=\frac{1}{N}\sum_{i=1}^{N}(x_i)$

and standard deviation: $\sigma=\sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2}$

Min-Max scaling: $Xnorm=\frac{X-Xmin}{Xmax-Xmin}$

```
scaler = preprocessing.StandardScaler().fit(X train)
X train = scaler.transform(X train)
X_test = scaler.transform(X_test)
```

4.1 Neural Network

The training data set was used to train the neural network shown in the figure 3.1. the method GrisSearchCV of the sklearn model_selection package was used to determine the best hyper

parameters with a stratified 3-fold cross validation. The parameters used to optimize the network are

```
gs = GridSearchCV(nn, param_grid={
    'n_iter': [100,500,1000],
    'learning_rate': [0.01, 0.001],
    'hidden0__units': [10, 20, 5],
    'hidden0__type': ["Rectifier", "Sigmoid", "Tanh"]},refit=True)
gs.fit(X train,y train)
```

The model is refit after it finds the best hyperplane parameters. The best estimators of the network are hidden0_units =5, n_iter=500, hidden_type = Sigmoid, learning_rate = 0.001. The neural network was refit with hidden layer containing 5 (S1) neurons and output layer contains 4 (S2) neurons. Sknn package is used to train the network with a learning rate of 0.001 using 1000 iterations and a validation size of 0.3. The hidden layers have a drop out of 0.2 where 20% of the data is randomly dropped out to prevent underfitting of the data. Stochastic gradient descent is used for the learning rule with a learning rate of 0.001. The cost function of mean categorical entropy is used to minimize the error of the network while training the network. A callback function is used to store the train errors and validation errors after each epoch.

```
def store_stats(avg_valid_error, avg_train_error, **_):
    valid_errors = []
    train_errors = []
    valid errors.append(avg valid error)
    train errors.append(avg train error)

nn = Classifier(
layers=[
    Layer('Sigmoid', units=5,dropout=0.10),
    Layer("Softmax")],
    learning rate=0.001,
    n_iter=1000,
    valid size=0.3,
    callback={'on_epoch_finish': store_stats})
nn.fit(X_train,y_train)
```

The validation and train errors were plotted to determine early stopping point of the network.

```
plt.figure()
plt.plot(range(len(train_errors)),train_errors,color="b",label="training scores")
plt.plot(range(len(valid_errors)),valid_errors,color="r",label="validation scores")
plt.legend(loc="best")
plt.title("Train Validation Error")
plt.show()
```

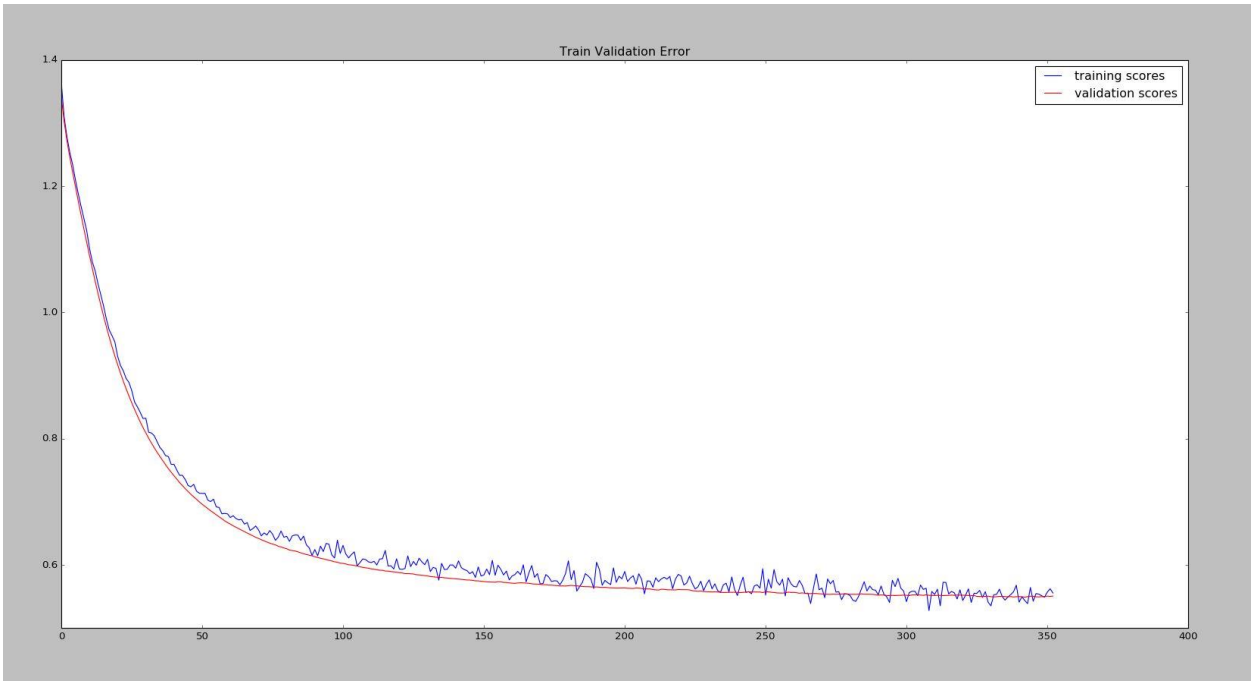


Figure 3.2

From the figure 3.2, the train and validation errors are decreasing and the validation errors are less than or equal to train errors. This is a implies the model is not under or over fit. We have chosen to early-stop at epoch 350 and refit the model.

4.2 SVM

The normalized training data is used to train a SVM model. The data is not linearly separable, hence different kernel functions and hyperplane parameters are used to find the optimal hyperplane using GridSearchCV with a stratified 3-fold cross validation. The parameters used to optimize the model are

```
parameter_candidates = [{'C': [1, 5, 10, 15, 20, 50, 100], 'gamma': [0.5, 0.05, 0.005, 0.01,
0.001, 0.0001, 0.00001], 'kernel': ['rbf','poly','sigmoid']}]
svm_grid_model = GridSearchCV(estimator=svm.SVC(), param_grid= parameter_candidates, refit= True)
svm_grid_model.fit(X_train, y_train)
```

The model is refit after it finds the best hyperplane parameters. The best estimators of the model are c=100, Gamma = 0.001 and Kernel = rbf. The rbf kernal is used to transform the input vectors using the following formula

$K(x,x') = \exp(-\gamma||x - x'||^2)$ where $||x - x'||^2$ is the squared euclidean diatance and $\gamma > 0$.

Here, γ defines how far the influence of the single training example reaches and low value is ‘far’ and high value is ‘close’. Gamma defines the radius of influence of the support vector in simple terms. C value is used to tradeoff between missclassification and simplicity of the decision surface. A low c makes decision surface smooth and high c makes it classify all the training examples by selecting more number of support

vectors. There has to be a significant tradeoff between c and gamma to get the best support vectors. In this training eaxample we use a c of 100 and gamma of 0.001.

The neural network model and SVN model will be used to predict for testing data set and will be compared to the expected output. The various metrics that will be used to evaluate the model are confusion matrix, precision recall and f1 score, number of misclassifications and accuracy of the models.

5. RESULTS

Figure 5.1 shows the confusion matrix of Neural Network

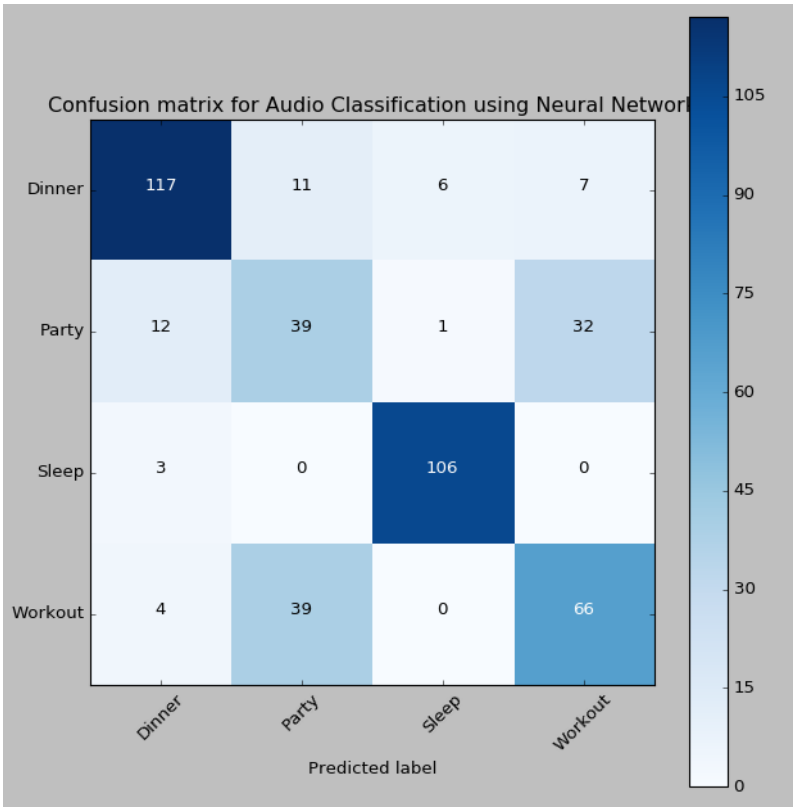


Figure 5.1

Figure 5.2 shows the confusion matrix of SVM

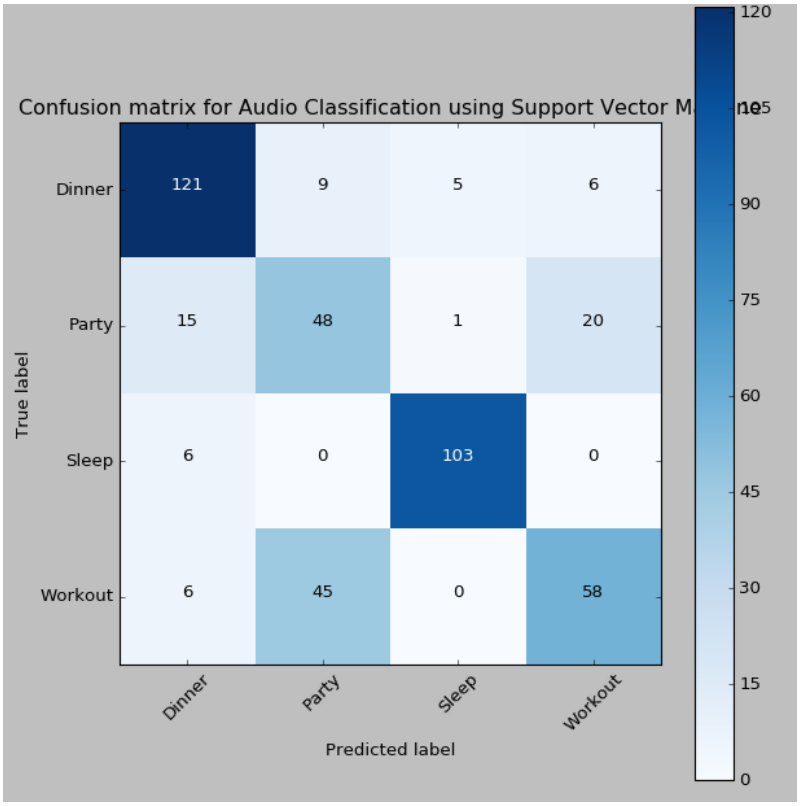


Figure 5.2

From figure 5.1 and 5.2, it can be observed that both the algorithms classify the data points similarly. Dinner and Sleep categories are classified almost correctly by both the models. Class party and workout have a lot of misclassifications. This is mainly because both the classes have similar features and also a song can be both party and a workout song.

Below table shows the summary of Neural Network and SVM

	Neural Network	SVM
# of misclassifications	115	113
Precision	0.74	0.75
Recall	0.74	0.74
F1 score	0.74	0.74
Accuracy	74.04%	74.5%

Table 5.1

Precision of an algorithm is calculated as

$$\frac{TP}{TP+FP} = \frac{\text{positive predicted correctly}}{\text{all positive predictions}}$$

Recall of an algorithm is calculated as

$$\frac{TP}{TP+FN} = \frac{\text{positive predicted correctly}}{\text{all positive observations}}$$

F1 Score of an algorithm is calculates as the harmonic mean of precision and recall

$$= 2\,\frac{\text{Precision}*\text{Recall}}{\text{Precision}+\text{Recall}}$$

Accuracy of an algorithm is calculated as

$$\frac{TP+FN}{TP+FP+FN+TN} = \frac{\text{Correct predictions}}{\text{all predictions}}$$

6. SUMMARY AND CONCLUSION

From the results discussed in the earlier section, both the algorithms classify similarly with accuracy of 74%. Dinner and Sleep classes are classified almost correctly by both the algorithms. On the other hand, party and workout have similar features, hence they are not classified accurately by both the algorithms. From the above results, we can conclude that Dinner and sleep songs are very different and workout and party songs are almost similar. Even in day to day life we have party and workout songs which are loud and have high bpm. Hence, we can conclude that party and workout songs can be made into a single category.

The model can be improved further by studying each feature of the songs and identifying which features are common to workout and party moods. The model can be then improved by considering only the features that are unique to these classes. This could improve the accuracy and decrease the misclassifications.

Thought Neural Networks and SVM have the same accuracy and classify similarly, SVM is much quicker than Neural Networks. We can also further explore other classification algorithms like Naïve Bayes, Stochastic Gradient decent and random forest and compare with the existing model.

7. REFERENCES

(n.d.). Retrieved May 01, 2017, from

https://www.kaggle.com/aniruddhaachar/audio- features

Sklearn.svm.SVC. (n.d.). Retrieved May 01, 2017, from

http://scikitlearn.org/stable/modules/generated/sklearn.svm.SVC.html

Confusion matrix. (n.d.). Retrieved May 01, 2017, from

http://scikitlearn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py

Sklearn.model_selection.GridSearchCV¶. (n.d.). Retrieved May 01, 2017, from

http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Sklearn.metrics.zero_one_loss¶. (n.d.). Retrieved May 01, 2017, from

http://scikit-learn.org/stable/modules/generated/sklearn.metrics.zero_one_loss.html#sklearn.metrics.zero_one_loss

Sklearn.neural_network.MLPClassifier¶. (n.d.). Retrieved May 01, 2017, from

http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

Hagan, M. T., Demuth, H. B., Beale, M. H., & Jesús, O. D. (2016). Neural network design. S. l.: S. n.

RBF SVM parameters¶. (n.d.). Retrieved May 01, 2017, from

http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

Sklearn.decomposition.PCA¶. (n.d.). Retrieved May 01, 2017, from

http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html