Music Genre Classification

-ANIKET BHATIA

-SHUBHANG BHATNAGAR

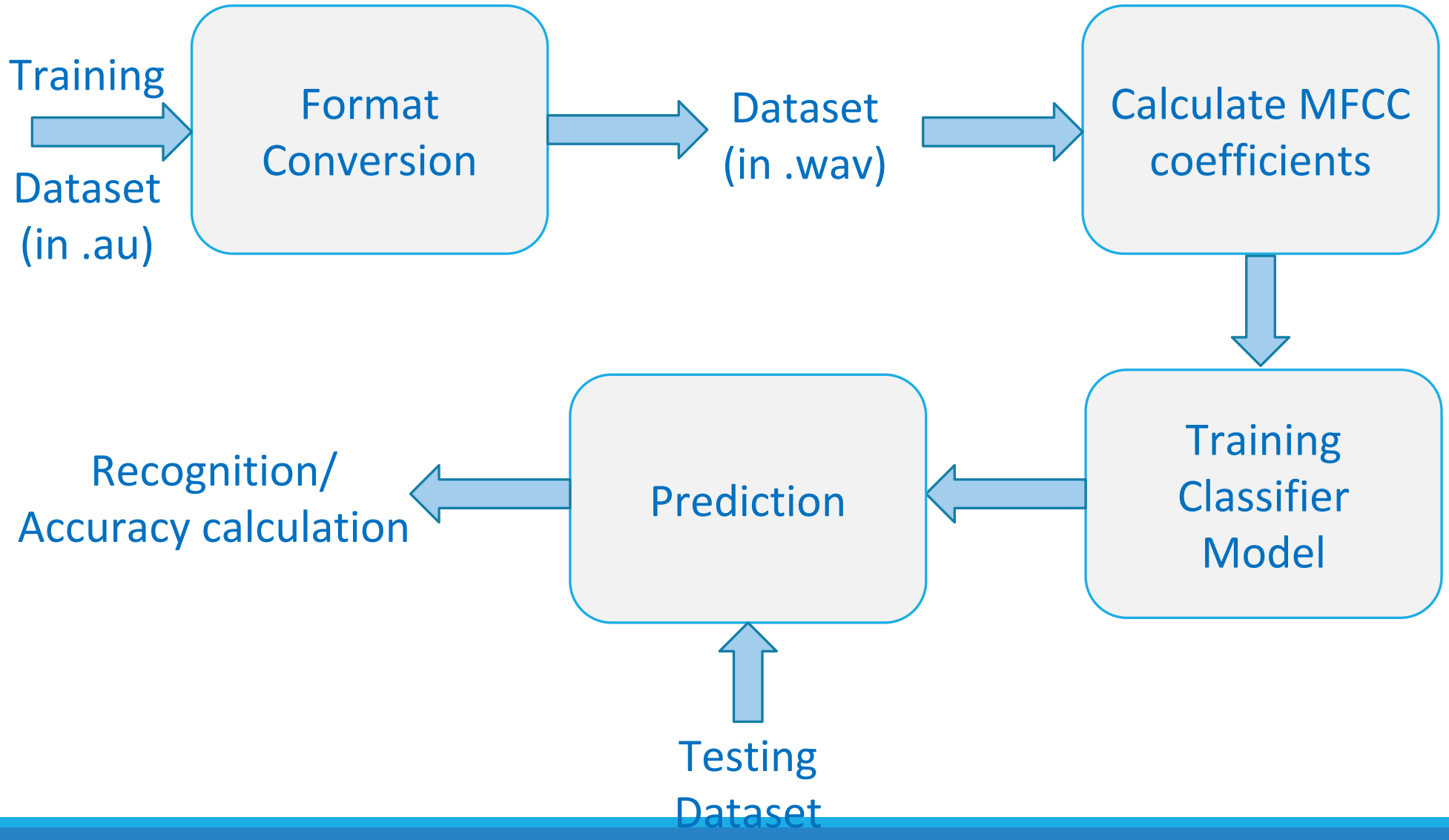
-VINEET ASHOK KOTARIYA

-CHINMAY GURJARPADHYE

Goals of the Project

- → To classify given set of music files into different classes like ‘Metal’, ‘Pop’, ‘Classical’ etc.
- → To use different models like Feed-Forward Neural Networks, Convolutional neural networks, Support Vector Machines, k-Nearest neighbours, k-means clustering etc.
- → To optimize the hyper-parameters in each classifier/model to maximize the accuracy.
- → To compare the different classifiers/models and find the best one for music genre classification.

Block diagram



Features

- → The MFCC coefficients intuitively represent the power of the signal in various frequency bands to which the human ear is the most sensitive. This is better than taking a Fourier transform of the song as here, we use the mel scale which is weighted according to the human ears sensitivity, which is maximum around 3-4 kHz.
- → We extract the MFCC coefficients using the given code for each second and take the mean and standard deviation of each coefficient across the 30 second time period.
- → We also use the number of zero crossings as feature along with the above mentioned coefficients
- → Other features included: Mean and standard deviation of Spectrum centroid, contrast, bandwidth, roll off, etc.

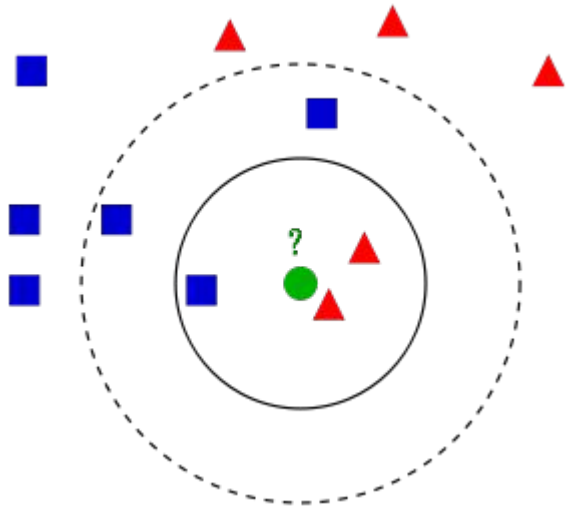
Pre-processing

- We used the GTZAN dataset for all of our classifiers. It has 100 songs of 10 genres. We only used 6 genres (due pre-processing data limitations) and from each genre, we put 60 into the training and 40 into the testing set.
- In pre-processing we first converted the .au files in the dataset to .wav files.We then extracted the MFCC (Mel-frequency cepstral coefficients) coefficients from the songs using the open- source librosa software. We used a code from github to preprocess the data and used this data in all of the classifiers. The link for it is- https://github.com/alikaratana/Music-Genre-Classification/blob/master/CreateDataset.py
- The 13 MFCC coefficients are calculated for each 0.1 second in each music file, this would mean 13 by 3000 attribute matrix for each training example.
- In the given paper, they have used delta and acceleration values to which would make it 39 by 3000 matrix. They claim that these new attributes increase the accuracy by over 10%
- To accelerate the execution we have to do some sort of dimensionality reduction. PCA being one of the options.
- We instead took mean and standard deviations of the 3000 values of jth coefficients (As the data was readily available) and did not use the delta and acceleration values.

Experiments

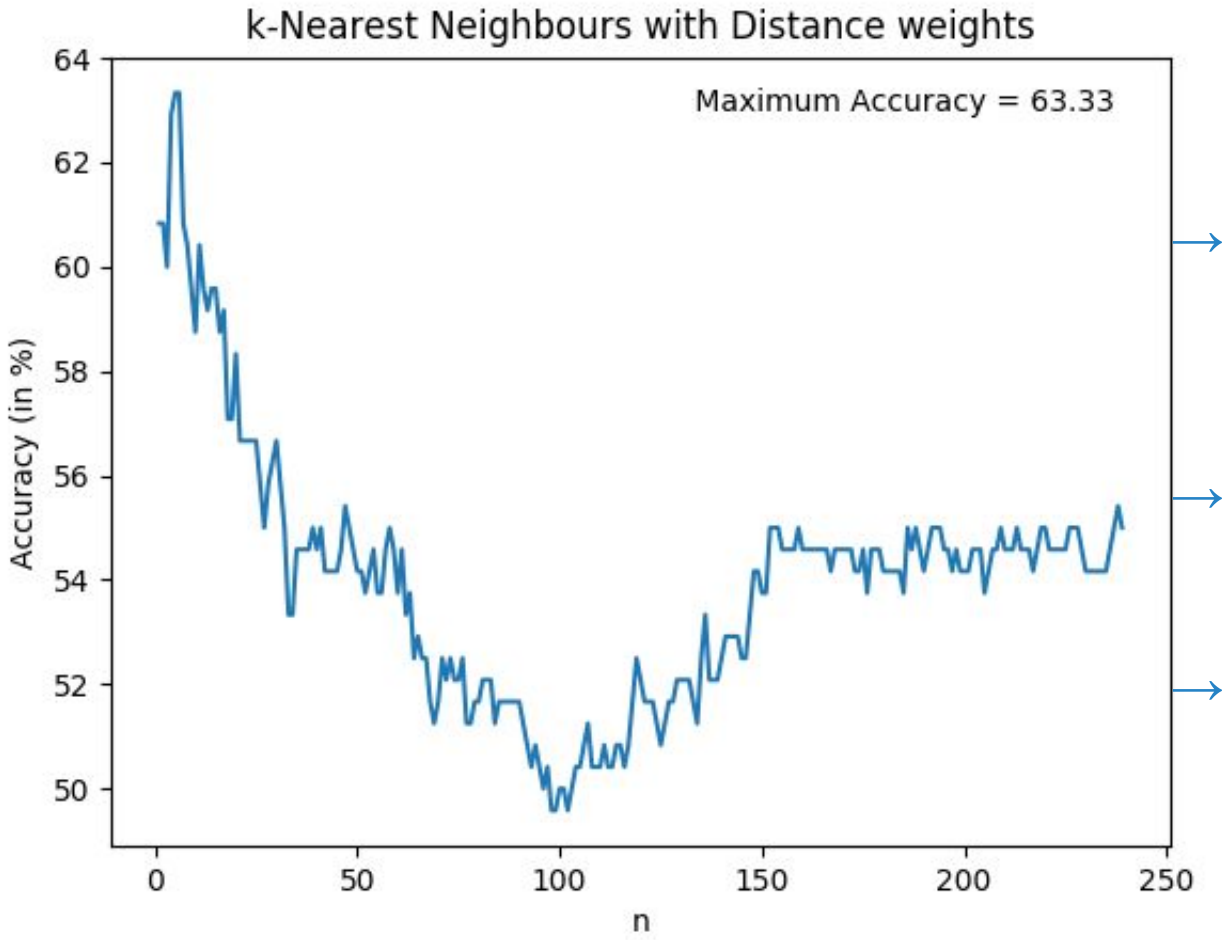
- → We wrote our code in python (in a Text editor(Atom)) and used different freely available libraries like numpy, pandas, matplotlib among other ML libraries
- → 1000+ Lines of code
- → URL pointing to our code snippets: https://colab.research.google.com/drive/1l9fDpBeHXHe_9prp9zlLAgIoBHC9oGps
- → We ran our code in two environments: (a) python3 in terminal (b) Google Collab
- → (We made requisite change in our code while jumping from one platform to another)
- → Following slides show our experimental results.

k-Nearest Neighbours



→ We used two types of weight functions:
→ Distance and Uniform

K-NN: Trends with n (Weight function: Distance)



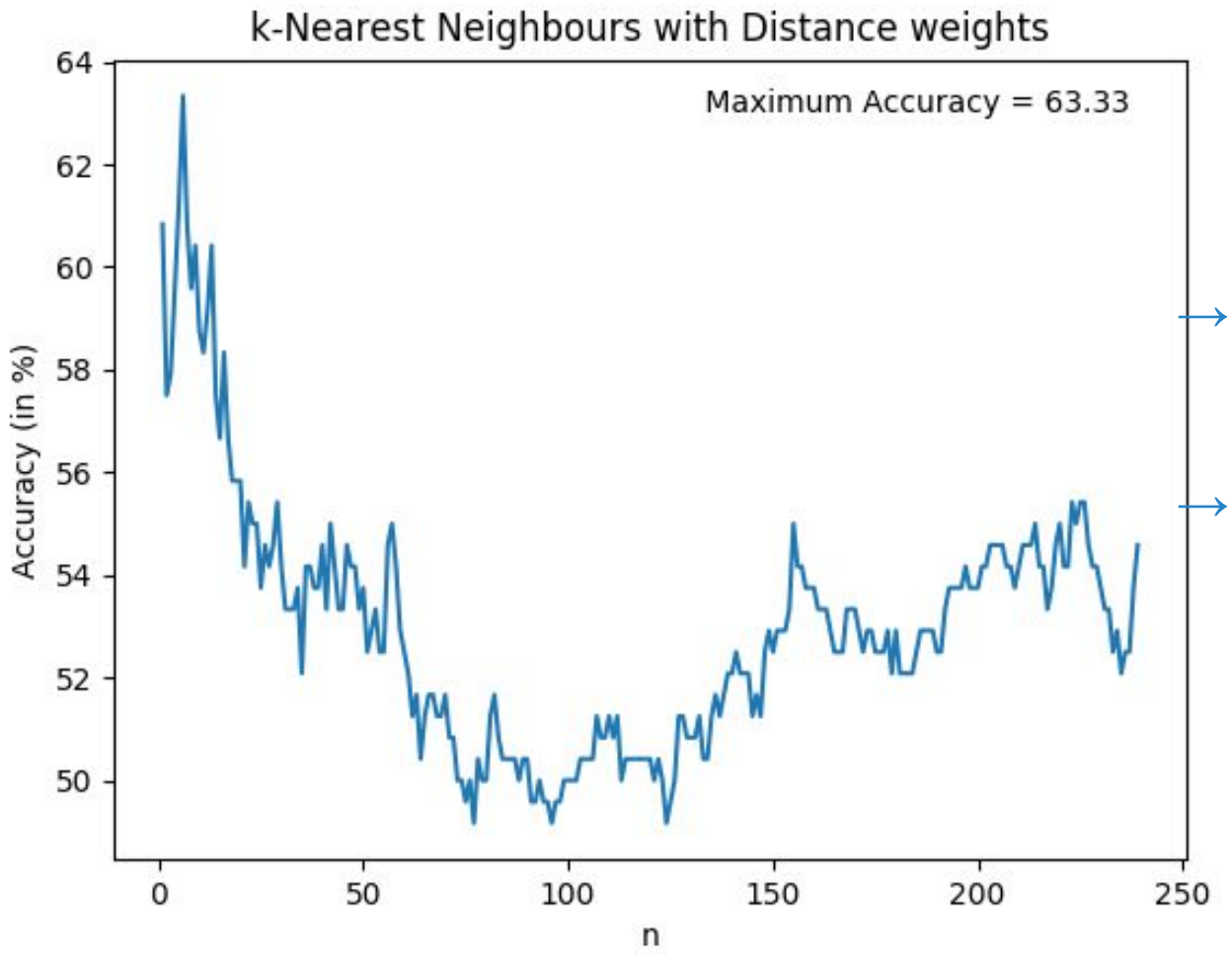
→ We achieve a maximum accuracy of 63.33% when taking 5 nearest neighbours into account and weighing each neighbour by the distance function.

→ The accuracy diminishes as expected when we use a very large number of neighbours as the estimate becomes cruder.

→ For very small n, it is more like overfitting with the training dataset.

→ **Maximum Accuracy : 63.33% at n=5**

K-NN: Trends with n (Weight function: Uniform)



→ Similar plot when we change the weight function to uniform

→ Unlike in distance function, the accuracy keeps fluctuating at higher n, as weights for each is equal and small outliers also produce a big difference.

→ **Maximum Accuracy : 63.33% at n=6**

K-Means Clustering

- → K-means clustering is an algorithm which divides the data set into a number of groups which are more similar to each other than the points in the other groups.
- → In totality we have 6 genres (i.e. we can go up to 6 clusters, if the entire data set is taken). So we should take 6 clusters, and it would seem that we are done!
- → But upon taking 6 clusters the achieved accuracy is just 47.92% -which is very poor! Hence the job of clustering is not as trivial as it seems! Thus we need to put in a lot more head ;)

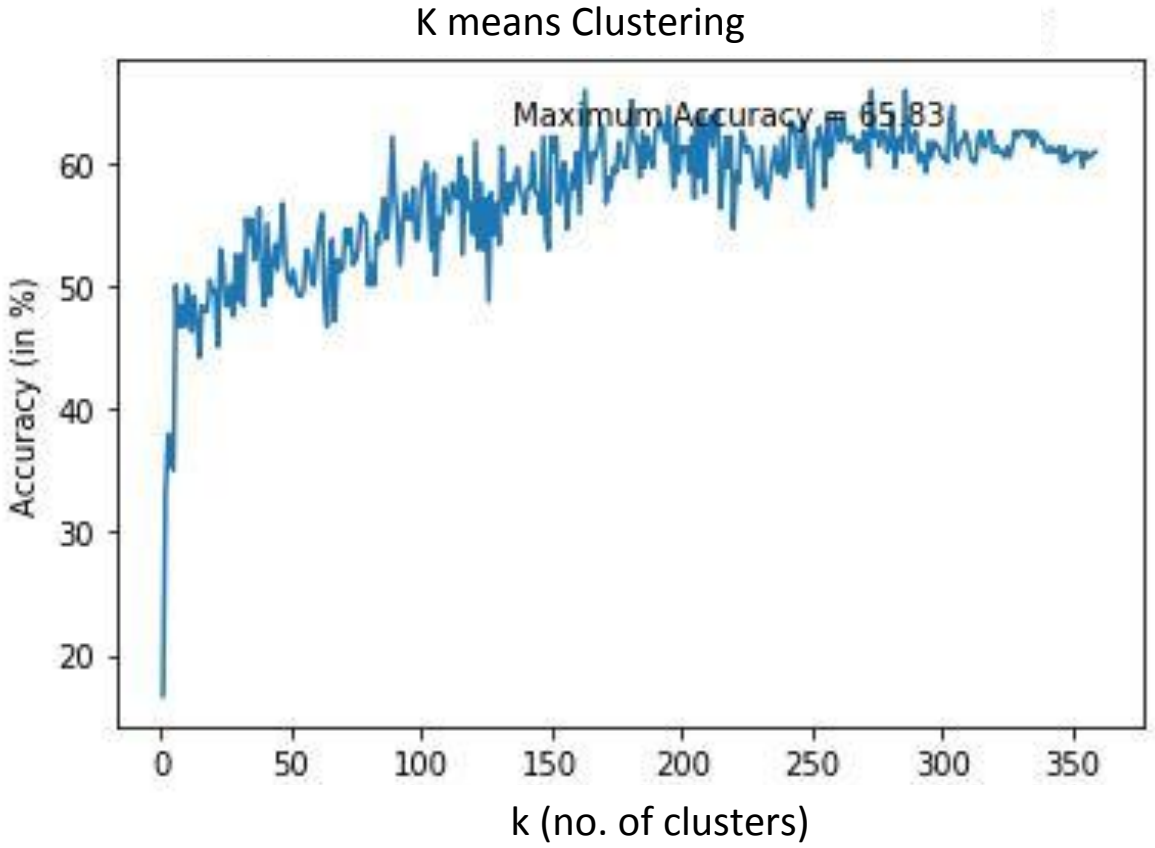
K-Means Clustering

Algorithm:

- → Consider a total of k clusters (k>>6, say 30,50 or even 80!)
- → Apply K means clustering on these k clusters with k enters now available to us.
- → Now each genre can be specified by more than one cluster. For mapping each cluster to a genre, consider the array in which labels 0-29 have been assigned to the training data set.
- → Now for a particular label consider all data points. Look at their genre and assign the mode (maximum frequency of occurrence) of the genre (class label) to that particular cluster label.
- → This way more than one cluster label can be mapped to one genre.
- → Now cluster labels are assigned to each test data point, and the mapping from the cluster label to the song genre is applied to get a genre for each data point and then the error is computed by comparing each label with its predicted label
- → ‘k’ now hence is a hyper-parameter which needs to be tuned to improve accuracy

K-Means Clustering Results:

1. With all 6 genres

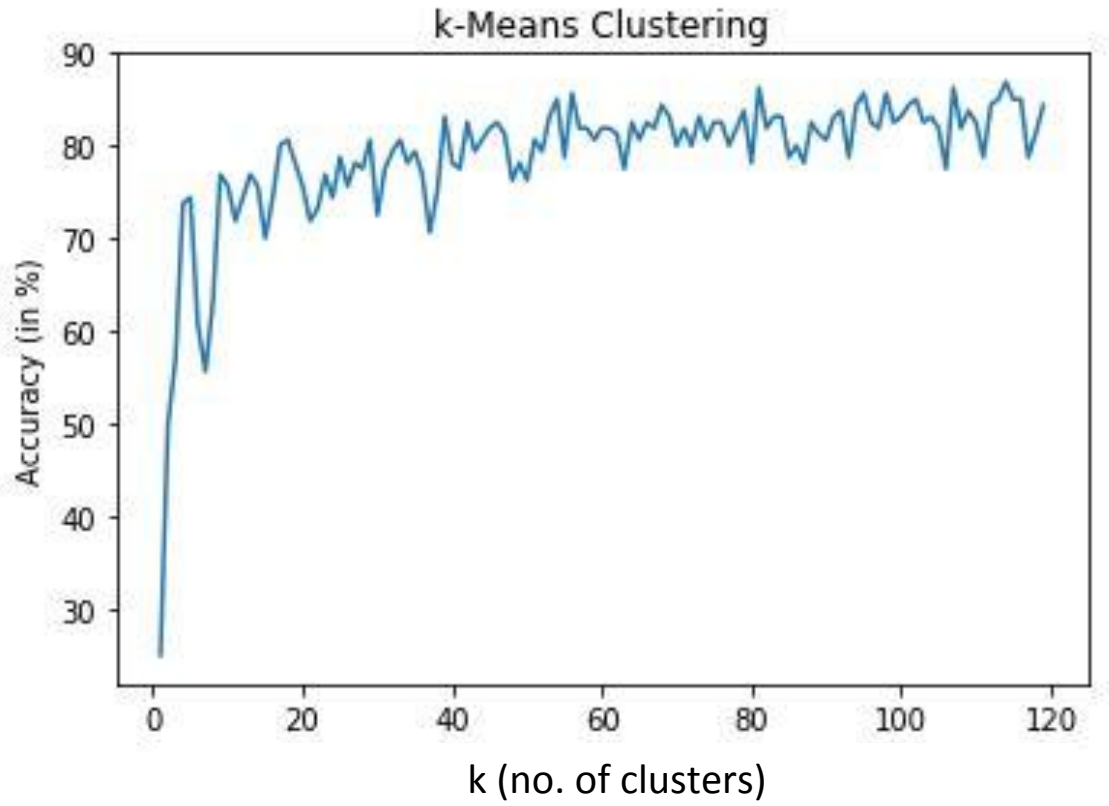


→ We achieve a maximum accuracy of 65.83%

→ Here we observe that for k=360 (the total no. of data set i.e. the maximum no. of clusters possible) the accuracy (60.83%) exactly matches with K nearest neighbour algorithm for K=1, which is intuitively satisfying.

K-Means Clustering Results:

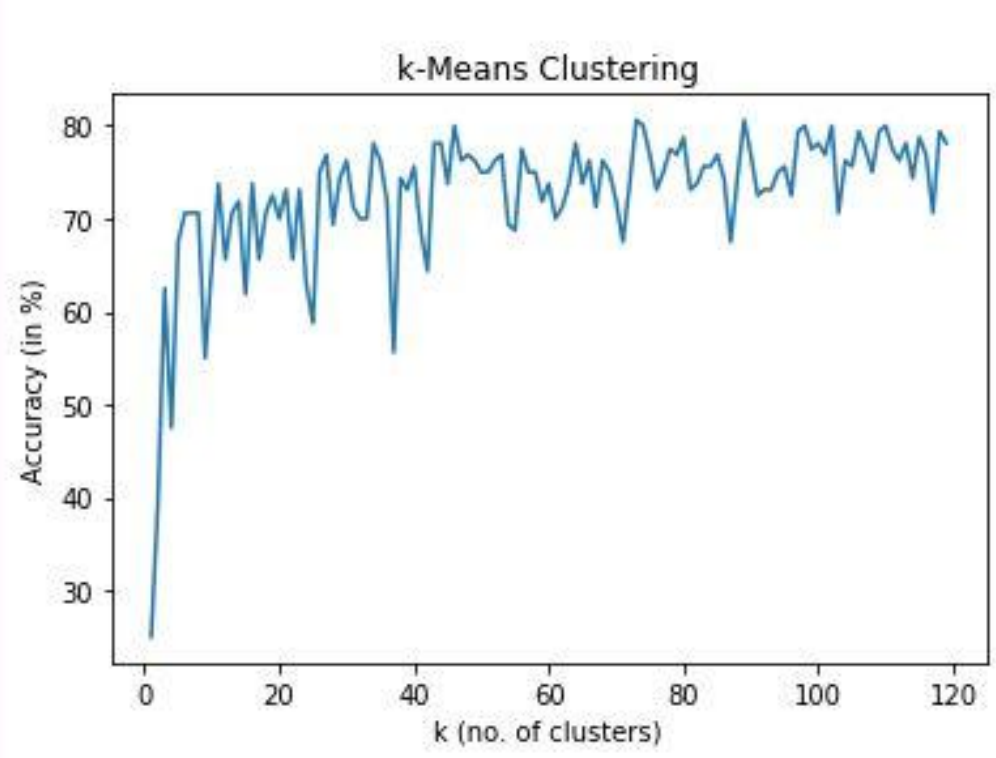
2. With 4 genres- Blues, Classical, Metal, Pop



→ We achieve a maximum accuracy of 86.88%

K-Means Clustering Results:

3. With 4 genres- Blues, Classical, Metal, Reggae



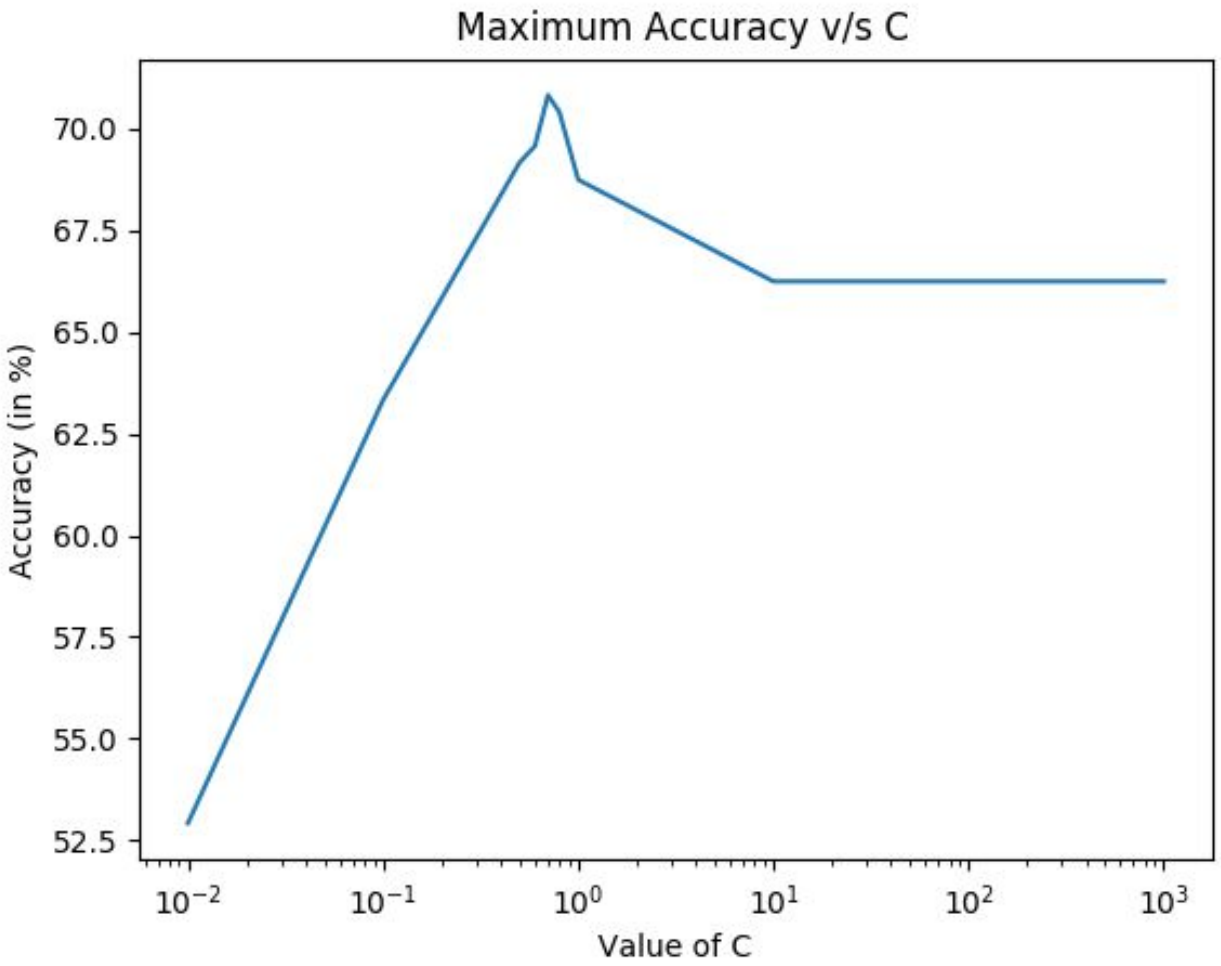
→ We achieved a maximum accuracy of 80.62%

Support Vector Machine(SVM)

Support vector machines are a very robust class of classifiers based on the hinge loss. They can classify varied sets of data using a trick called the kernel trick-which makes them much more dynamic than most other loss regularization classifiers. One of the most popular kernel- RBF has been used by us in the project and it has the equation-

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right)$$

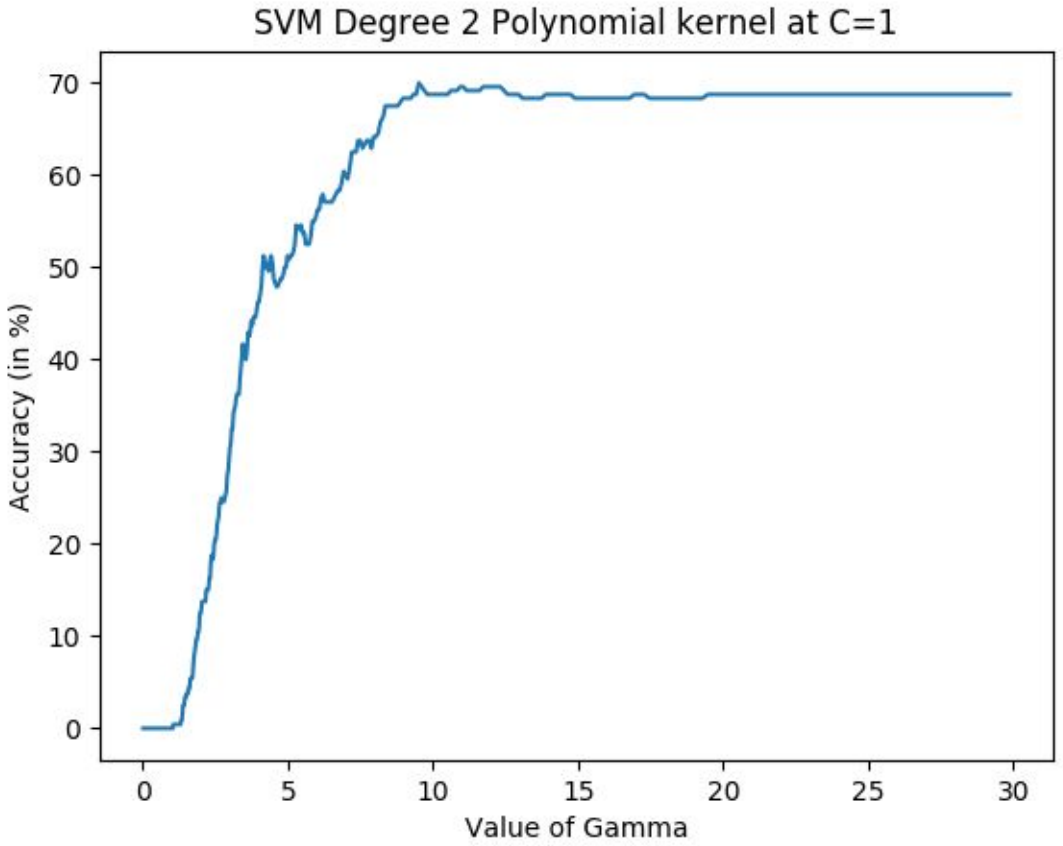
SVM linear kernel: Trends with C



- Here we are plotting the overall accuracy with the constant multiplied to the loss function C
- As expected, the accuracy initially increases, as the loss function gets higher emphasis
- But at very large value of C, the accuracy almost saturates as the loss function dominates the $|w^2|$ objective and after a certain value of C is exceeded it doesn't matter if C is higher.

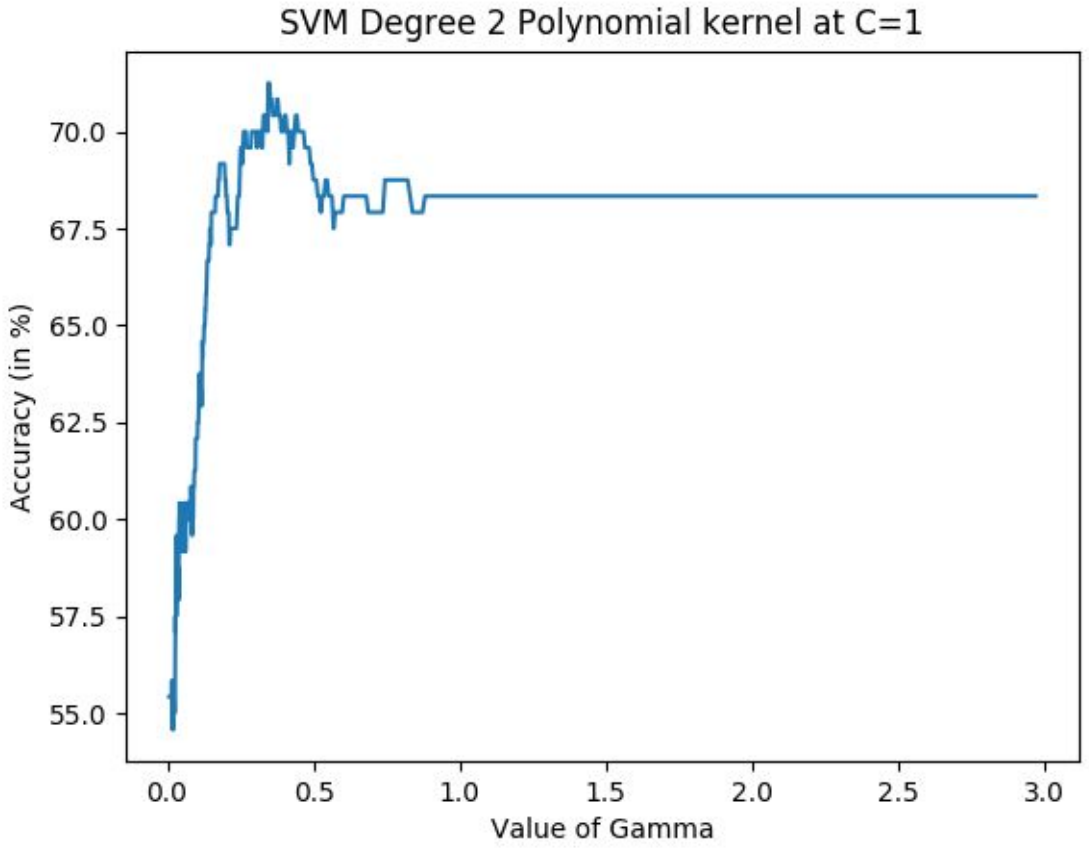
→ **Maximum Accuracy : 70.08%**

SVM degree 2 polynomial kernel: Trends with C
Bias = -10



→ **Maximum Accuracy : 69.98%**

SVM degree 2 polynomial kernel: Trends with C
Bias = 8

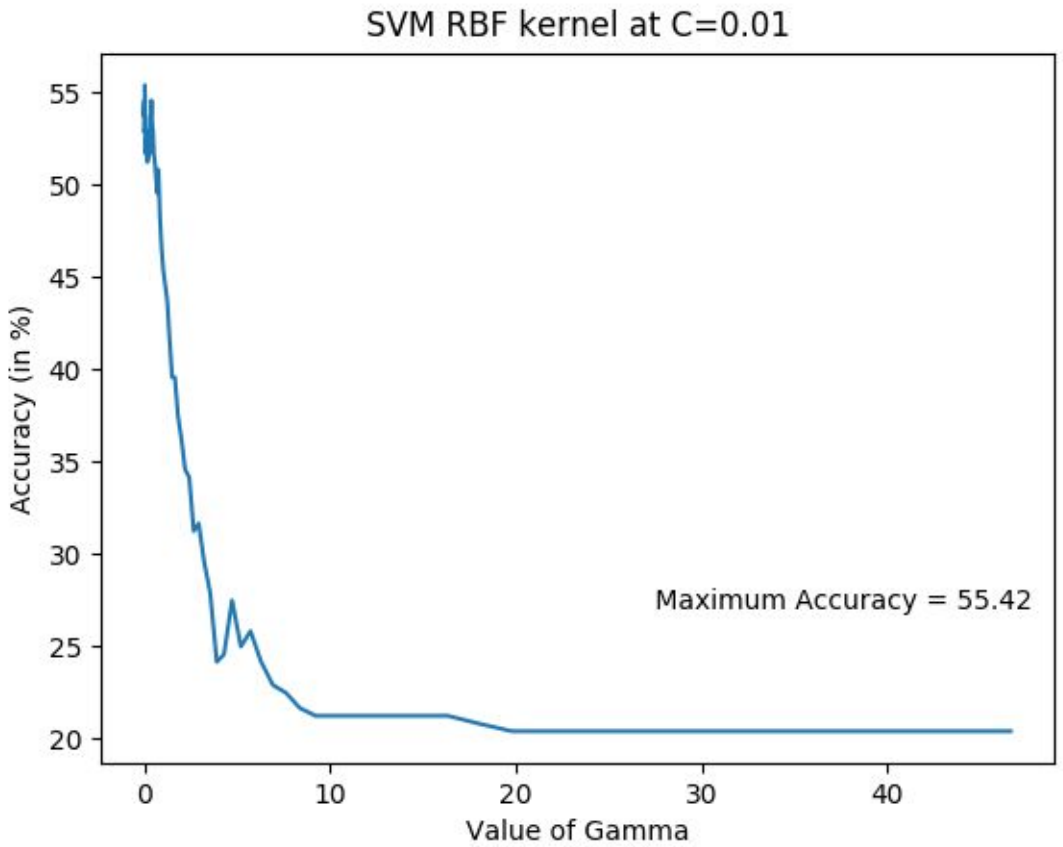


→ As expected, the accuracy of a second degree SVM kernel is higher than that for a linear kernel as it gives a higher order representation of the underlying function

→ The maximum accuracy peaks at Coeff = 8

→ **Maximum Accuracy : 72.08%**

SVM RBF kernel: Trends with Υ

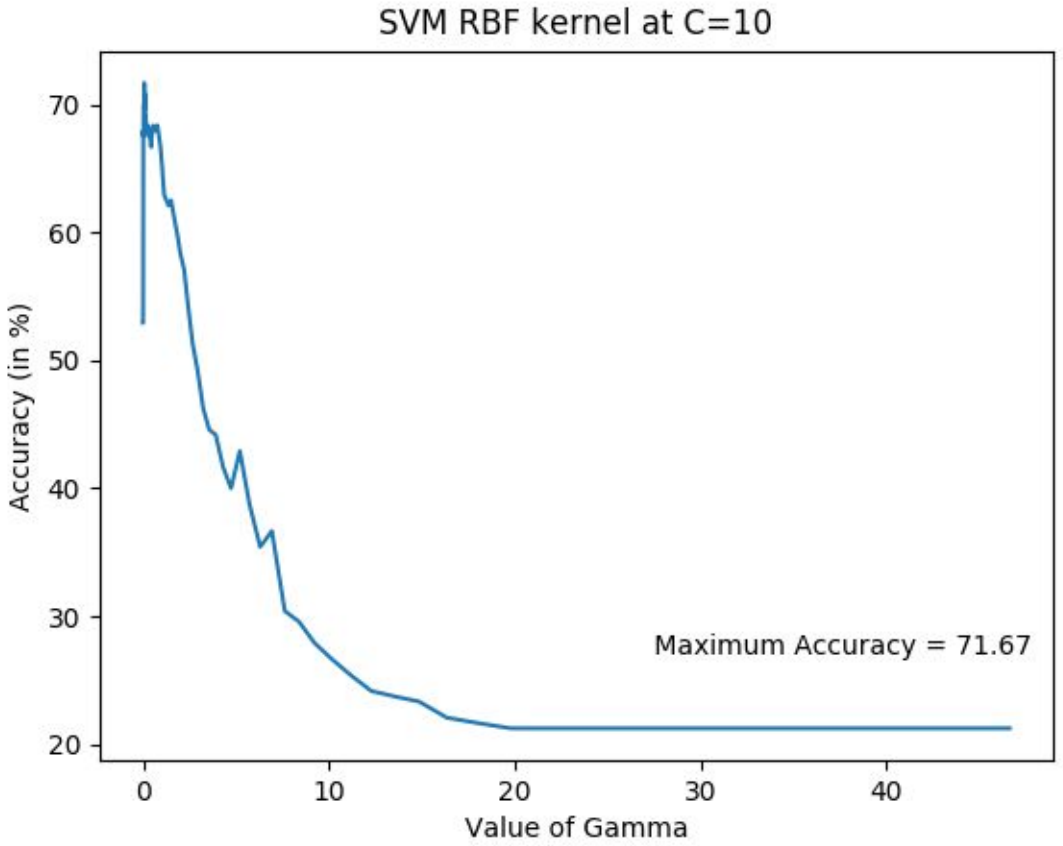


→ The accuracy of the RBF kernel is almost the same a second degree SVM kernel, it doesn't give better accuracy

→ As expected, the accuracy rate plummets at higher value of Υ as we start to overfit the training data set

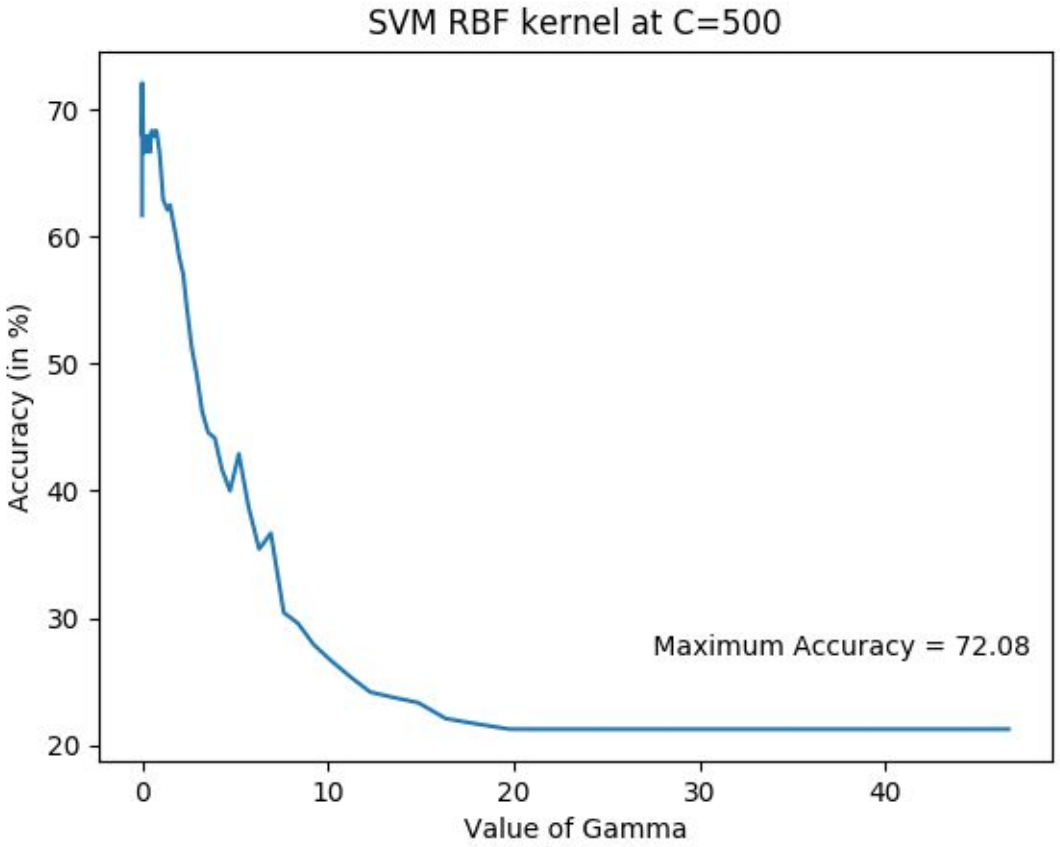
→ **Maximum Accuracy : 55.42%**

SVM RBF kernel: Trends with Υ



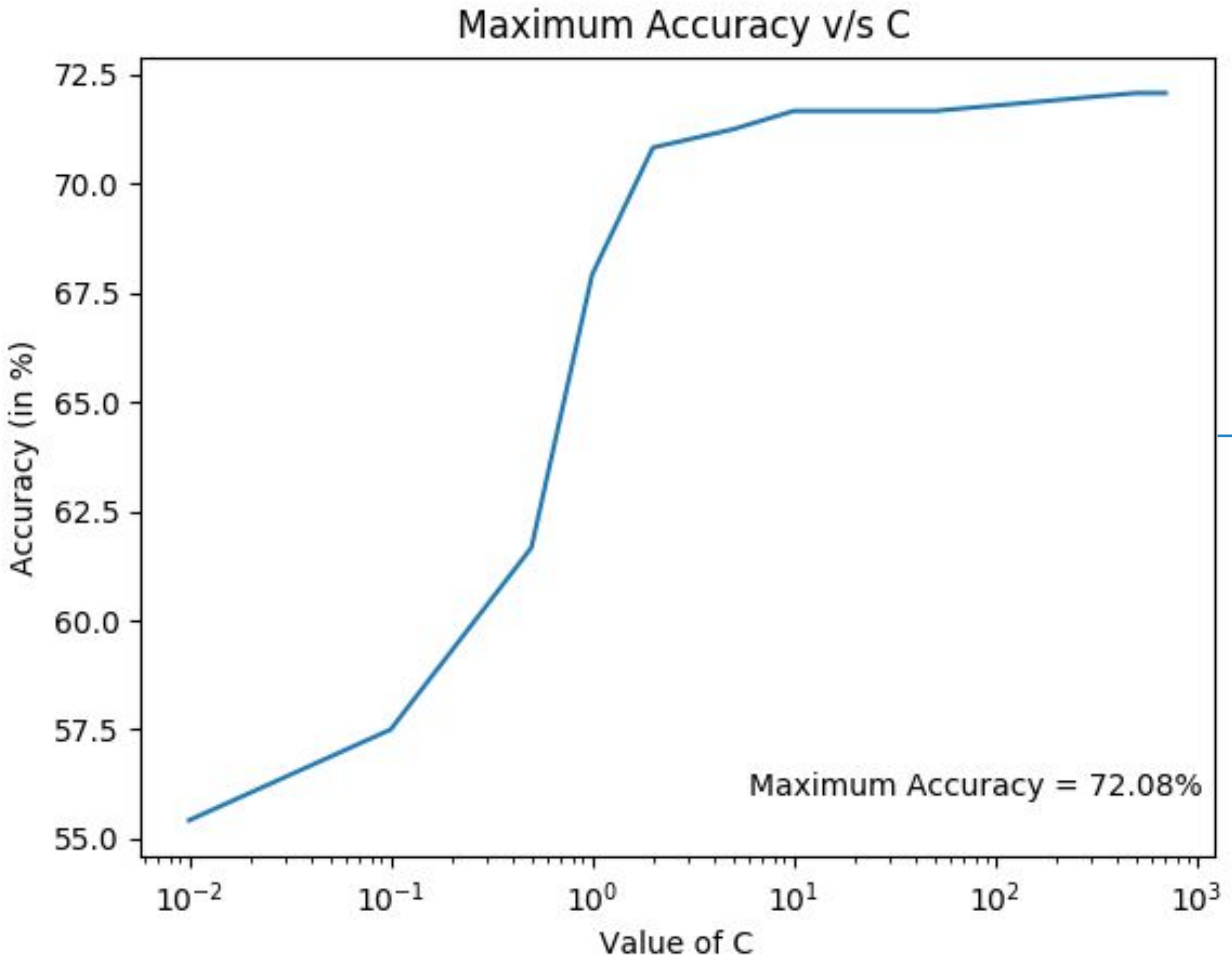
→ **Maximum Accuracy : 71.67%**

SVM RBF kernel: Trends with Υ



→ **Maximum Accuracy : 72.08%**

SVM RBF kernel: Trends with C (For γ=350)



→ Like in the linear kernel case, the accuracy increases initially and the saturates at higher values of C (Because of exactly same argument)

→ **Maximum Accuracy : 72.08%**

Using only 3 Genres (Metal, Classical, Pop)

→**We get an overall accuracy level of 99.16% for Υ = 0.79**

→Metal : 97.5%

→Classical: 100%

→Pop : 100%

→**For 6 Genres best overall accuracy = 72.08% for Υ = 0.715**

→Blues: 35.0%

→Classical : 97.5%

→Hiphop : 72.5%

→Metal : 90.0%

→Pop : 87.5%

→Reggae : 27.50%

Neural Networks

Neural network’s are a very important class of classifier’s,as they can easily model non linear decision boundaries too.We tried to classify the songs into the 6 genre’s using a feedforward neural network. We decided to go for a 4 layer deep network for the classification. We also add a softmax layer at the end for classification

I used coded the neural networks in tensorflow using the keras API.

Before giving the training data as input, we encode the genre types using one hot encoding,so that we can get easily get the prediction’s from the softmax layer.

I used Gradient Descent and Adam Optimizer’s for optimizing the process of minimizing the loss

Approaches Used

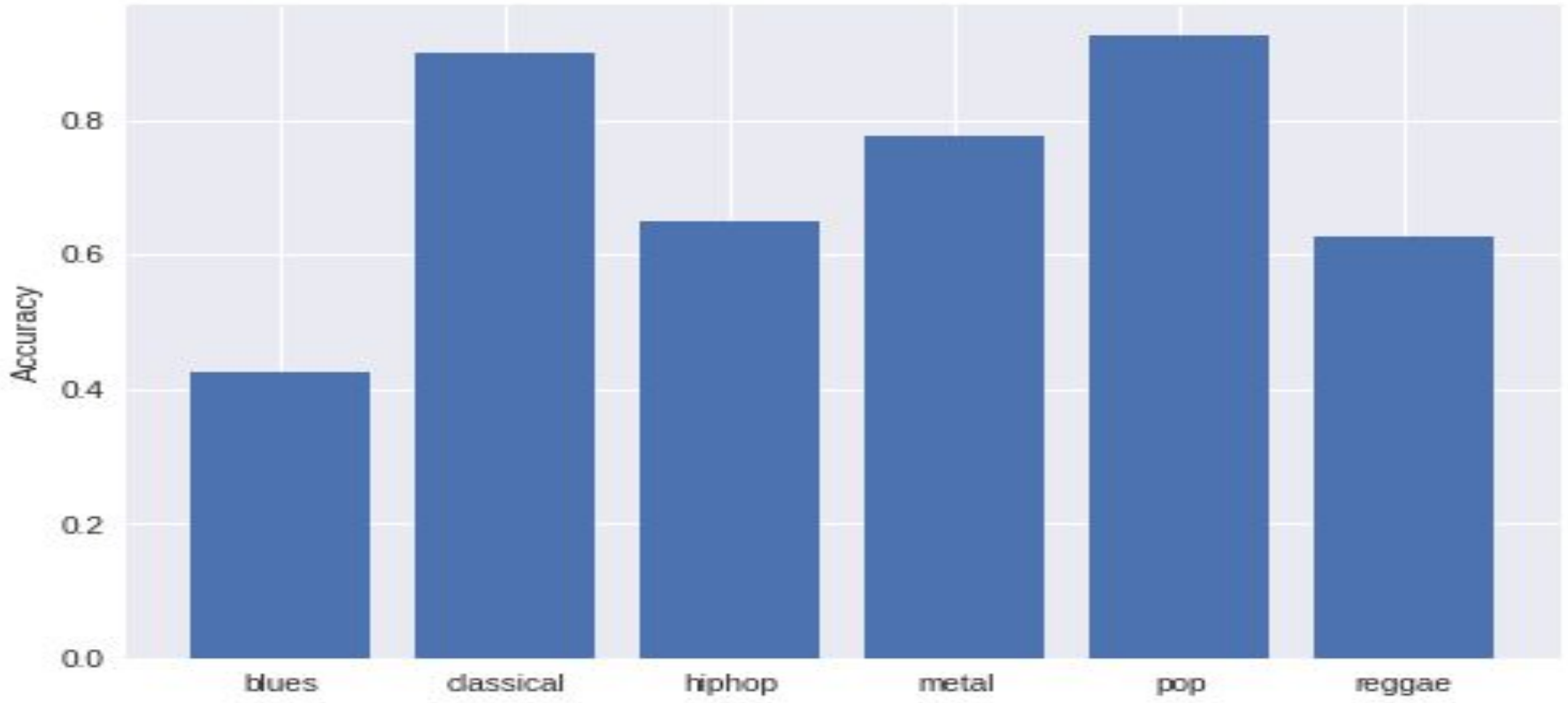
I tried 4 approaches to the problem-

1) Using a feedforward network with no regularisation
2) Using a L2 regularised network
3) Using a network regularized by dropout layers

In all of them,the activation function is Relu. Also,I have randomly initialized the weights in all the 3 networks

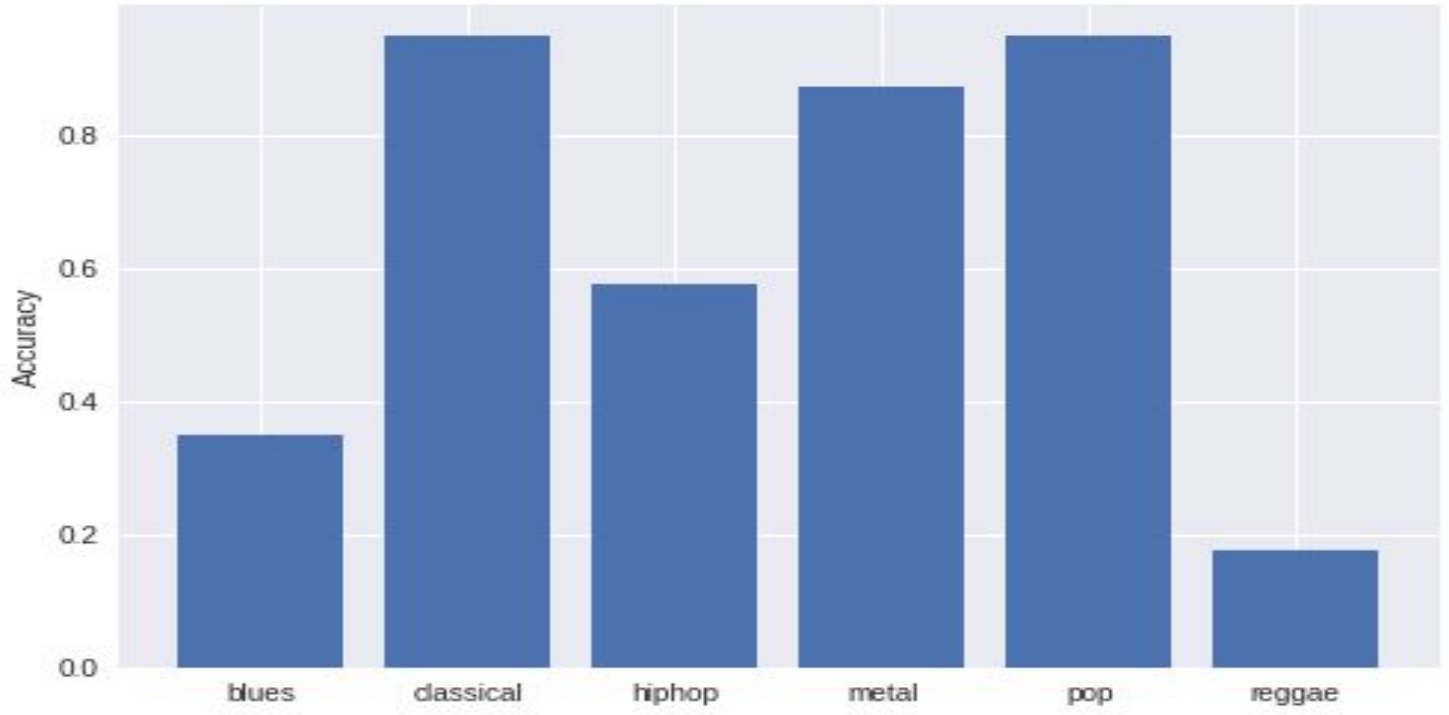
Feedforward network without regularisation

The maximum accuracy I got is 73.77% overall. The individual accuracies of each of the genre's are-



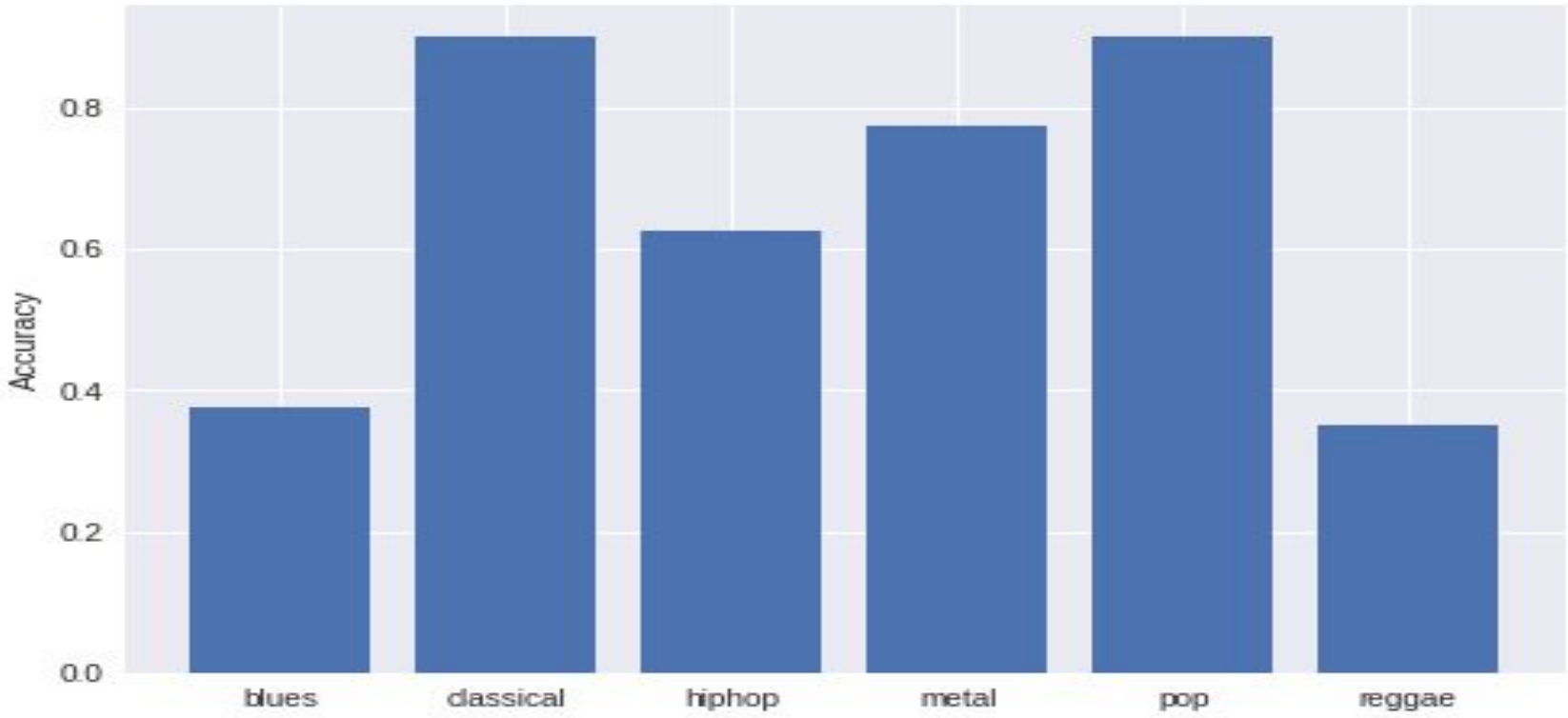
Feedforward network with L2 regularisation

The maximum accuracy I got in this case was 65.11%. This means that the L2 regulariser was not very effective and even wrongly penalized some correctly classified instances. It decreased the accuracy of the reggae class to only 18%



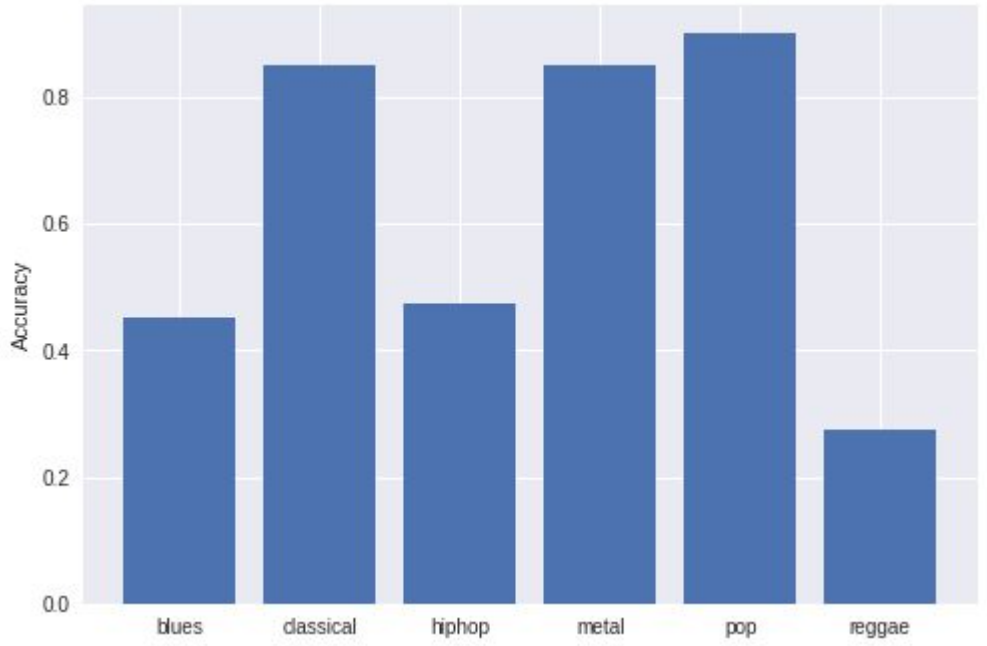
Feedforward network with dropout regularisation

The maximum accuracy I got was 66% using dropout layer's to prevent overfitting.This method was also not very effective as it reduced the test accuracy. But this was much more effective than the L2 regulariser as it did not affect any class(eg reggae) very adversely.



1D Convolutional Neural Networks (CNN)

The accuracy we got is 63.34% overall. However, the result is un-optimized and there is still scope for improvement. The individual accuracies of each of the genre's are-



Effort

- Fraction of time spent on each sub-part:

→Acquiring data and Pre-processing: 40%

→Neural Network: 25%

→SVMs: 15%

→k-NNs and K-means clustering: 15%

→Report: 5%

- Most Challenging part:

→Pre-processing and Architecture of the Neural network (and CNNs)

- Fraction of Work done by each member:

→Aniket: 30% Shubhang: 30% Vineet: 30% Chinmay: 10%

References

→Karpov, I.. “Hidden Markov Classification for Musical Genres”, http://www.music.mcgill.ca/~ich/classes/mumt611_06/similarity/GenreHMM.pdf

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

→https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

→https://stackoverflow.com/questions/47709248/using-pydub-to-convert-audio-from-au-to-wav

Acknowledgement

→Marsyas v0.2 “GTZAN Genre Collection” Dataset, “http://marsyasweb.appspot.com/download/data_sets/”

→Github user Alikaratana User: https://github.com/alikaratana we used adopted his code for pre-processing the Music (.au) files to get the dataset in attribute form.

→Prof. Sunita Sarawagi for their guidance.

“There is no real ending. It’s just the place where

-Frank Herbert