**FINAL PROJECT REPORT**

**PROBLEM STATEMENT AND BACKGROUND:**

Music emotions are felt and expressed in ways that can barely even be explained by adjectives because they are so central to the human experience. The few music emotions that can be put into words include emotions like: joy, nostalgia, peacefulness, power, sadness, tenderness, tension, transcendence, and wonder. We chose this project to see if we can actually predict the likelihood that the song will be able to stir these emotions, given simple features of the song itself- It’ll be interesting to see the mathematical relationship of liking a song to features and characteristics related to the artist and the song.

The project aims to classify whether a song will be a hit or not, and will it reach Top-Charts based on several music and artist attributes as predictor variables. It will be crucial for us to predict the popularity of a song and we will be using various classification models in our analysis to predict the probability of a particular song reaching the Top Charts.

**PREPARING THE DATASET:**

We are using the music data from the Million Song Dataset. The Million Song Dataset (MSD) is a musician feature dataset which contains more than one million contemporary songs. It contains:

• 280 GB of data

• 1,000,000 songs/ﬁles

• 44,745 unique artists

• 7,643 unique terms (Echo Nest tags)

• 2,321 unique music brainz tags

• 43,943 artists with at least one term

• 2,201,916 asymmetric similarity relationships

• 515,576 dated tracks starting from 1922

The data is stored using HDF5 format to efﬁciently handle the heterogeneous types of information such as audio features in variable array lengths, names as strings, longitude/latitude, similar artists, etc.  [HDF5](http://www.hdfgroup.org/HDF5/) is a format developed by NASA to handle 1) large 2) heterogeneous 3) hierarchical datasets. The data can be compressed (10%-15% more that matfiles), and the I/O speed is still impressive. Each song is described by a single ﬁle.

Importing the data was the first challenging part of this project. We only used the 2GB subset data from the million song data to train and test our models. The subset data zip file provided by echonest and labrosa had a lot of subfolders consisting of 10,000 files in total where each file represents one track with all the related information (artist information, release information, audio analysis of the track, etc).

We used the **rhdf5** package in R to import the subset data. A glimpse of the R code to import these rhdf5 files and the steps followed to merge all the files can is summarized below:

**SUMMARY TO IMPORT THE SUBSET DATA:**

**R CODE TO READ THE INDIVIDUAL H5 FILES AND MERGING THEM IN A SINLE DATASET:**

function\_data\_combined <- function(directory)

{

mylist <- list.files(path = directory, full.names = TRUE)

res <- lapply(mylist, h5read, name="/analysis/songs")

return(res)

}

myData <- function\_data\_combined("C:/Users/arpit/Documents/MillionSongSubset/Merged\_files")

library(plyr)

data\_final<-ldply(myData, rbind)

**DATA WRANGLING:**

Both the subset and the entire dataset were fairly large. The subset is 2.5GB while the full Million Songs dataset is 270GB. We initially tried building our models directly using the HDF5 files. However, that made it difficult to do data cleaning, imputation and statistical analyses. We switched to a different method based on the observation that a lot of the data in the dataset was not relevant to us (such as the actual audio tracks) so we filtered the data from the HDF5 files and converted it into a much more compact dataframe. We filtered the files by only extracting features relevant to our problem statement. For the 10,000 song subset, these included the following fields: artist\_familiarity, artist\_hotttnesss, artist\_id, artist\_latitude, artist\_longitude, artist\_location, artist\_name, song\_id, song\_hotttnesss, title, artist\_terms, artist\_terms\_freq, artist\_terms\_weight, danceability, duration, energy, key, loudness, mode, tempo ,year . By applying this filter, we were able to convert the 2.5GB HDF5format dataset into a single R dataframe.

**EXPLORATORY DATA ANALYSIS:**

We began the process of data exploration by examining the dataset in aggregate and computing basic univariate statistics (count, mean, standard deviation, min, max) for each of the explanatory variables. A few of interesting bivariate relationships have also been discussed below.The DataExplorer package in R is a very helpful package is R which eases the explanatory data analysis process.

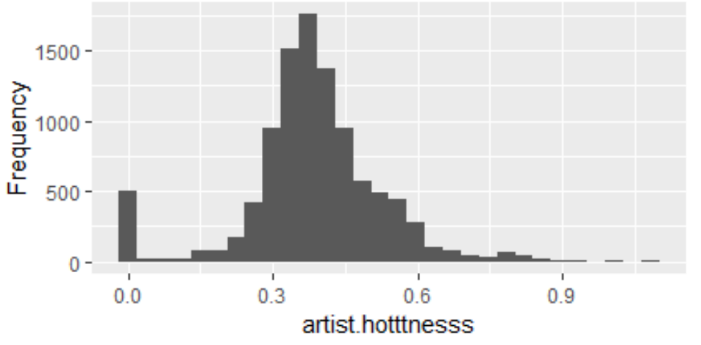
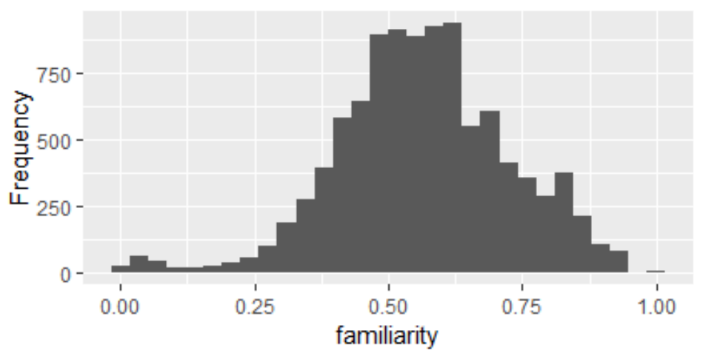
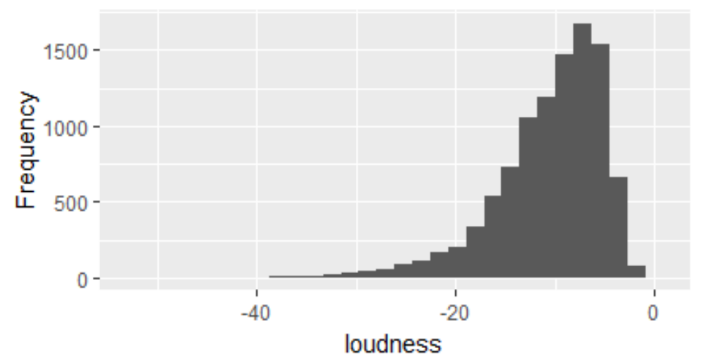
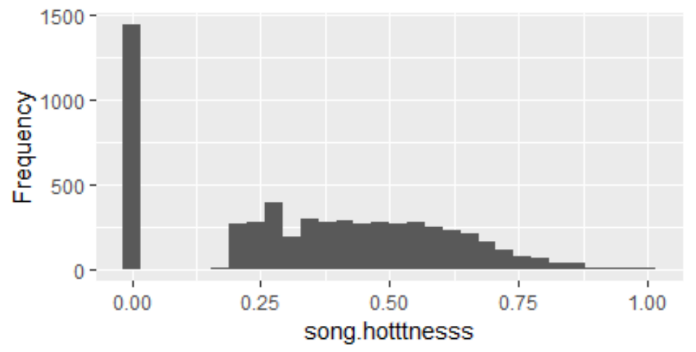
**UNIVARIATE STATISTICS AND EXPLORATION:**

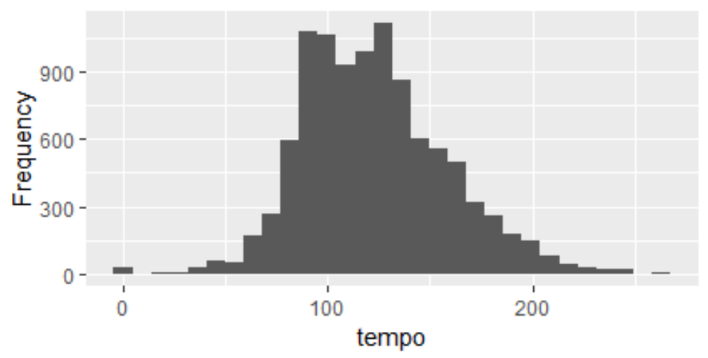
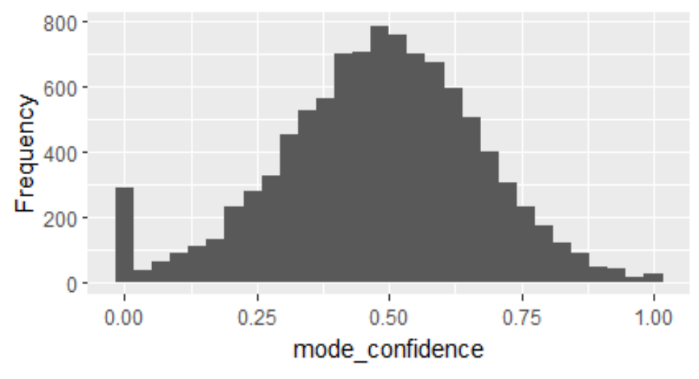
Some basic univariate statistics of all the numeric variables was the first step we did.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Mean** | **Median** | **Standard Deviation** | **Range** | **Minimum** | **Maximum** |
| *artist.hotttnesss* | 0.39 | 0.38 | 0.14 | 1.08 | 0.00 | 1.08 |
| *artist\_mbtags\_count* | 0.52 | 0.00 | 0.88 | 9.00 | 0.00 | 9.00 |
| *bars\_confidence* | 0.24 | 0.12 | 0.29 | 8.86 | 0.00 | 8.86 |
| *bars\_start* | 1.07 | 0.79 | 1.72 | 59.74 | 0.00 | 59.74 |
| *beats\_confidence* | 0.61 | 0.69 | 0.32 | 1.00 | 0.00 | 1.00 |
| *beats\_start* | 0.43 | 0.33 | 0.81 | 72.25 | -60.00 | 12.25 |
| *duration* | 240.62 | 223.06 | 246.08 | 22048.96 | 1.04 | 22050.00 |
| *end\_of\_fade\_in* | 0.76 | 0.20 | 1.86 | 43.12 | 0.00 | 43.12 |
| *familiarity* | 0.57 | 0.56 | 0.16 | 1.00 | 0.00 | 1.00 |
| *Key* | 5.37 | 5.00 | 9.67 | 904.80 | 0.00 | 904.80 |
| *key\_confidence* | 0.45 | 0.47 | 0.33 | 19.08 | 0.00 | 19.08 |
| *latitude* | 37.16 | 37.16 | 9.54 | 110.93 | -41.28 | 69.65 |
| *longitude* | -63.93 | -63.93 | 30.89 | 337.20 | -162.44 | 174.77 |
| *loudness* | -10.48 | -9.38 | 5.40 | 52.21 | -51.64 | 0.57 |
| *mode* | 0.69 | 1.00 | 0.46 | 1.00 | 0.00 | 1.00 |
| *mode\_confidence* | 0.48 | 0.49 | 0.19 | 1.00 | 0.00 | 1.00 |
| *start\_of\_fade\_out* | 229.88 | 213.86 | 112.02 | 1834.82 | -21.39 | 1813.43 |
| *tatums\_confidence* | 0.51 | 0.50 | 0.33 | 9.23 | 0.00 | 9.23 |
| *tatums\_start* | 0.30 | 0.19 | 0.51 | 12.25 | 0.00 | 12.25 |
| *tempo* | 122.90 | 120.16 | 35.20 | 262.83 | 0.00 | 262.83 |
| *terms\_freq* | 224.89 | 1.00 | 22392.16 | 2239217.00 | 0.00 | 2239217.00 |
| *time\_signature* | 3.56 | 4.00 | 1.27 | 7.00 | 0.00 | 7.00 |
| *time\_signature\_confidence* | 0.60 | 0.55 | 8.99 | 898.89 | 0.00 | 898.89 |
| *Year* | 934.70 | 0.00 | 996.65 | 2010.00 | 0.00 | 2010.00 |
| *song.hotttnesss* | 0.34 | 0.36 | 0.25 | 1.00 | 0.00 | 1.00 |

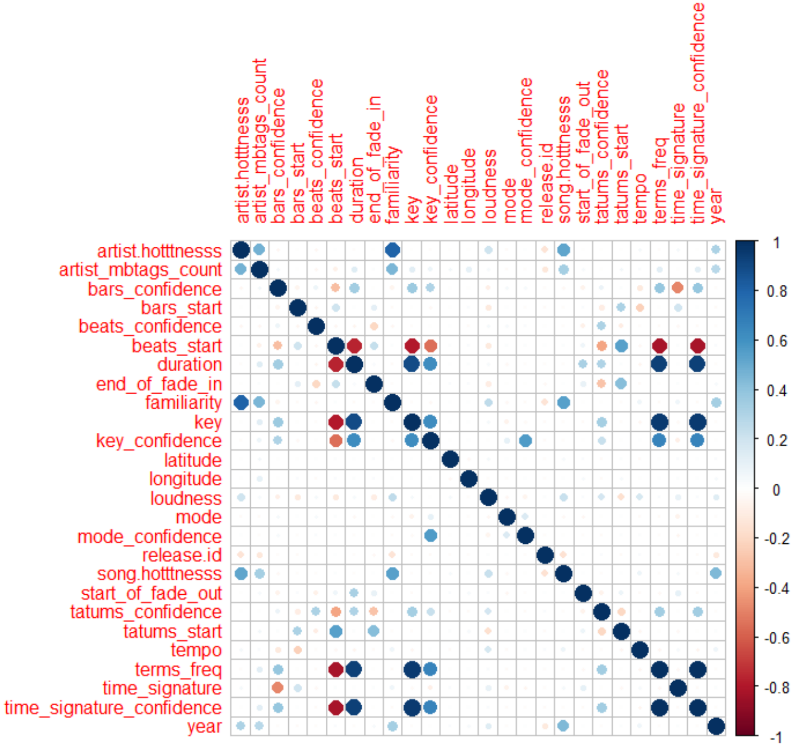
We plotted some graphs to get a sense of the distribution of different fields in our dataset. As

can be seen from Figure 1, key fields as as tempo, artist familiarity, artist hotttnesss, song.hotttnesss, song duration all follow a gaussian distribution , which is important for distance metric based models such as SVM, kNN,etc.



**BIVARIATE ANALYSIS:**



To get a sense of the extent of linear relationship between song hotness and other explanatory variables we used the corrplot package in R and found that artist familiarity and artist hotness had a high positive correlation with the song hotness.

Loudness of the song also had a mild positive correlation with the song hotness- the louder the song, the greater the chances that it’ll hit the top charts.

**DATA CLEANING:**

The data explorer package in R gives a nice summary of the statistics on missing observations and columns in the data. We saw that in the subset data, we had about 58.5% observations with missing data and in totality only 1.2% observations actually had missing data.

We used the basic univariate statistics to determine which numeric fields required cleaning. Post analyzing the data, we found that the main point of contention was that that the most important variable – song hotttnessswas missing from a large number of data points. Attempting to impute or categorically fill a value for year would be futile, and we cannot impute *hotttnesss* since it is what we are trying to predict. We decided to use the spotify API in R to fill the missing values in the song popularity variable. The spotiftr library in R requires a user to create a profile on developer.spotify website and use the client id and clientsecret id to get an access token for a successful connection for data extraction. Post removing the observations with missing song hotness, the 5,604 records which contained our prediction variable song\_hotness label now was increased to 7490 and the dimension of our final dataset we used had 7590 rows.

We also had a lot of variable with missing values for which we used various modelling techniques rather than imputing by using univariate statistics such as mean or median. During our bivariate exploratory data analysis we found that artist hotness and artist familiarity had a high positive correlation with a Pearson correlation coefficient of 0.82 so we used simple linear regression to impute artist familiarity and artist hotness as we had non missing values for familiarity where artist hotness was missing and vice versa. For missing values in the explanatory variables related to the audio features such as end\_of\_fade\_in, tempo, loudness, beats\_start, beats\_end we used kNN by using all the non missing audio features and genre information extracted from the song terms.

We checked for duplicate keys using the song\_id as our unique record identifier. There were no duplicate records. We checked for statistical anomalies using the basic statistics described previously. The only anomalies were in the energy and danceability columns, which we dropped.

**FEATURIZATION:**

Our featurization involves two main components: feature selection and feature engineering. We

explain our reasons for the features we have selected and for the features we plan to generate

below. These features are a combination of filtering down all features, and generating new ones.

**Feature Engineering**:

* The songs had two very unique attributes named artist\_mb\_tags and terms with values like Pop, rock, Indie, etc. in both – one related to the arist and the other related to the song.The artist\_tags had a very low fill rate,but the terms related to the song had a 100% fill rate.We felt that this may aid to the prediction in a lot of ways so we converted this to a document term matrix and appended the same to our original dataset using tf-idf feature creation. The term frequency normalizes the occurrence of the terms in a single row and the idf adjusts for the sparseness of the occurrence across observations in the data.The quanteda package in R was used to create the document term matrix and the bag of words.
* The year data had a lot of 0 values, so we imputed its value as not available in the missing and converted the year into decades and converted into a factor variable rather than using the year variable as a factor variable as it is.

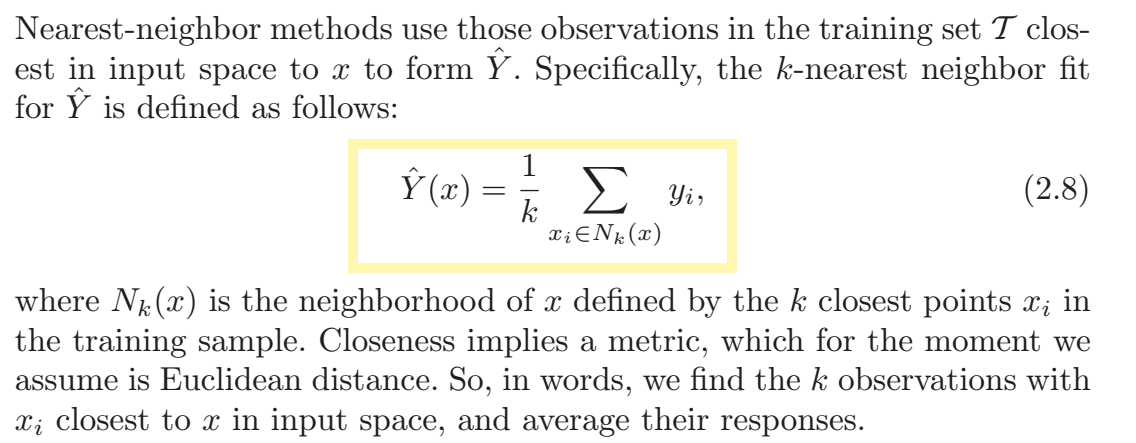
**Feature Selection**:

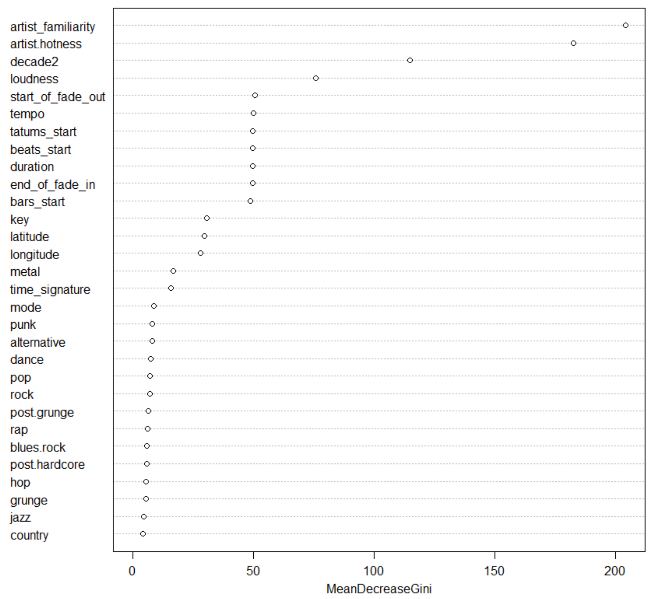
For extracting important variables especially for distance based models such as kNN, we used feature selection techniques to extract only the important explanatory variables in our problem.We began by using a large subset of the features available (initially selected based on intuition of what features are likely to be predictive, as well as quality and availability of data for that feature) but then used ablation on a Random Forest model to determine the optimal feature set. The optimal feature set included the following features: artist\_familiarity, artist\_hotttnesss, artist\_terms , duration, tempo, loudness, artist\_location, artist\_name, decade and some term variables such as pop, rock, punk, metal, dubstep etc.

**MODELING:**

The song\_hotttness variable was used to create the dependent variable for the classification problem- a threshold value of 0.5 was used to tag a song as a hit or not hit. The class distribution of the hit variable was close to about 30% (hit). Hence,we distributed our data into 75% training and 25% test and created our models on the training data and thereby calculated the generalized error performance of each model using cross-validation. We went ahead and created our classification models using the a couple of classification algorithms including knn, decision trees, random forest,Naïve Bayes and support vector machines.

**KNN:**





For k nearest neighbours algorithm, we used random forest for feature selection and used only a subset of variables selected in line with the variable importance we got from our random forest model (shown on the left).

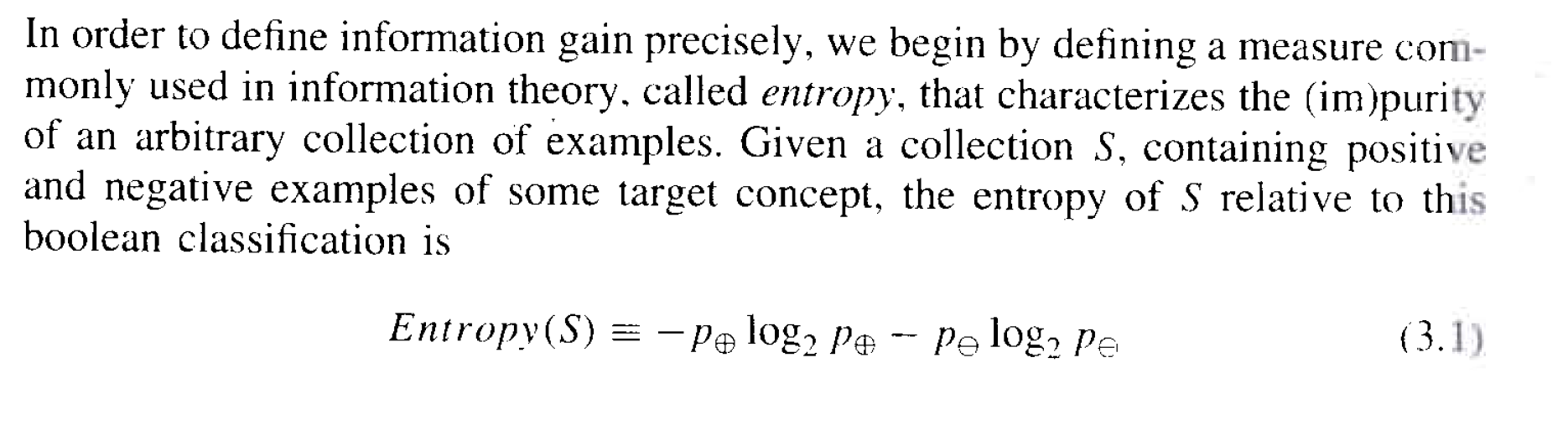
For preprocessing the data for KNN we first normalized the data in order to remove the bias on distance calculation towards the variables having a wider range and we tried both Euclidean and Manhattan distance measures in order to compute the distances.The categorical variables were handled by using 1-

In order to find out the optimal value of k for our k nearest neighbours algorithm, we used the traincontrol and train function in caret package in R to get optimal k with the minimum training and validation error and found that k=9 minimised the generalized error.

|  |  |  |
| --- | --- | --- |
| **kNN** | *Actual* | |
| *Predicted* | **Not hit** | **Hit** |
| **Not hit** | 1313 | 265 |
| **Hit** | 164 | 156 |

The confusion matrix corresponding k=9 translates to an accuracy of ~78% with a precision and recall of 49% and 39% respectively for the hit class and 83% and 89% for the not-hit class.

**Decision Trees:** For the decision tree classification we used the rpart package in R to classify the songs and tuned the cost complexity pruning (cp) and the minsplit value parameter to find out optimal length of the tree which gives minimum generalization error.



The e1071 package in R hosts the tune function which was used to find out the optimal value of cp and minsplit. The optimal parameters corresponding the best decision tree used information gain as the splitting criterion and cp=0.02 and minsplit=30.

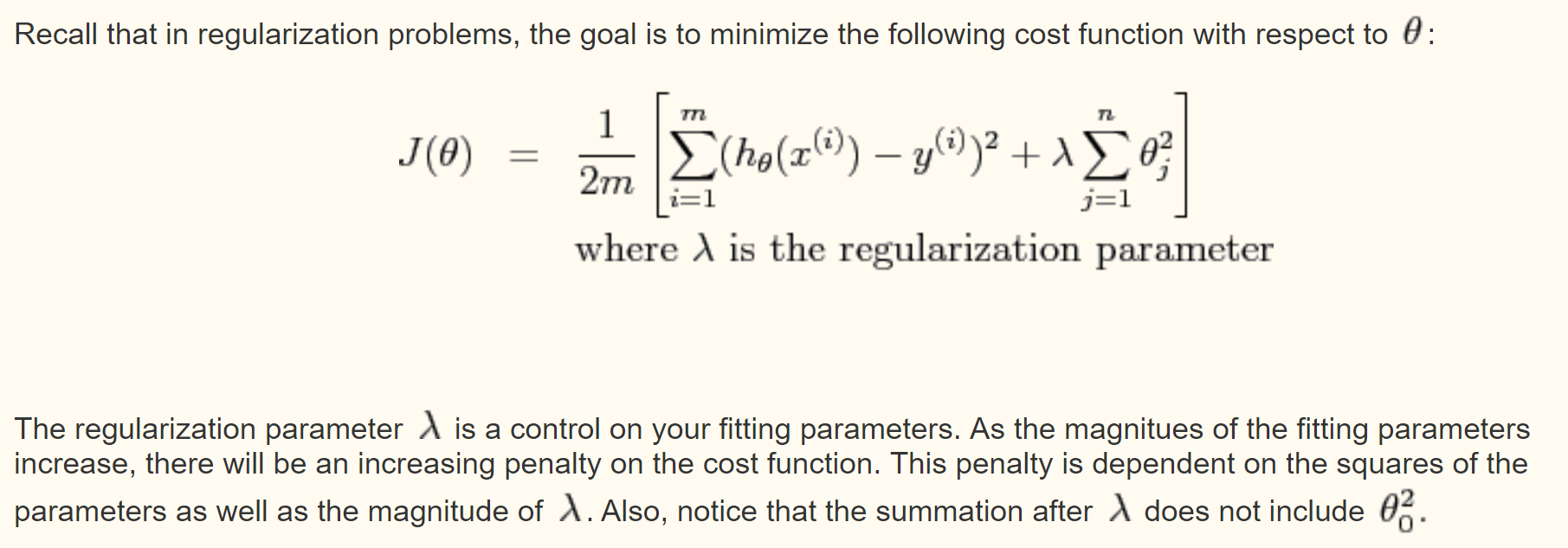
The rule based learner pointed out artist\_hotness, decade , loudness and tempo to be the most important predictor variables.

|  |  |  |
| --- | --- | --- |
| **D-Trees** | *Actual* | |
| *Predicted* | **Not hit** | **Hit** |
| **Not hit** | 1338 | 217 |
| **Hit** | 139 | 204 |

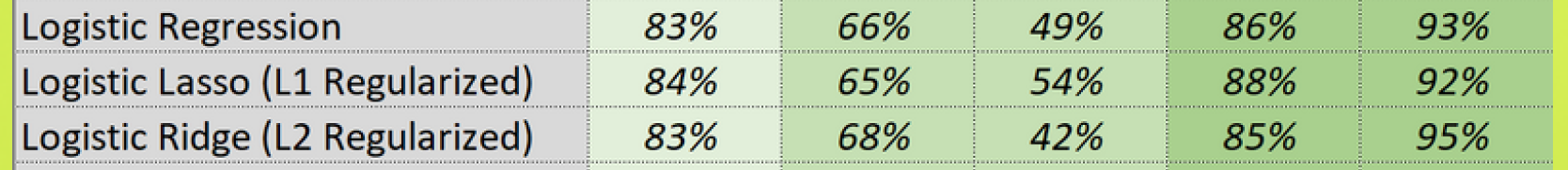
The confusion matrix corresponding the best decision tree model translates to an accuracy of ~81% with a precision and recall of 60% and 48% respectively for the hit class and 86% and 91% for the not-hit class.

**Logistic Regression:**

For the logistic regression model we first performed the classification without regularization followed by a ridge and lasso regression by setting alpha=0 and alpha =1 and optimized the lambda value using cv.glmnet package in R. L1 regularized logistic regression requires solving a convex optimization problem. However, standard algorithms for solving convex optimization problems do not scale well enough to handle the large datasets encountered in many practical settings. In an **L2 regularization:**



The best result received from running the logistic regression models pre and post regularization (L1 and L2) can be summarized below:



**Random forest:**

Random Forests grows many classification trees. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).Each tree is grown as follows:

1. If the number of cases in the training set is N, sample N cases at random - but *with replacement*, from the original data. This sample will be the training set for growing the tree.
2. If there are M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
3. Each tree is grown to the largest extent possible. There is no pruning.

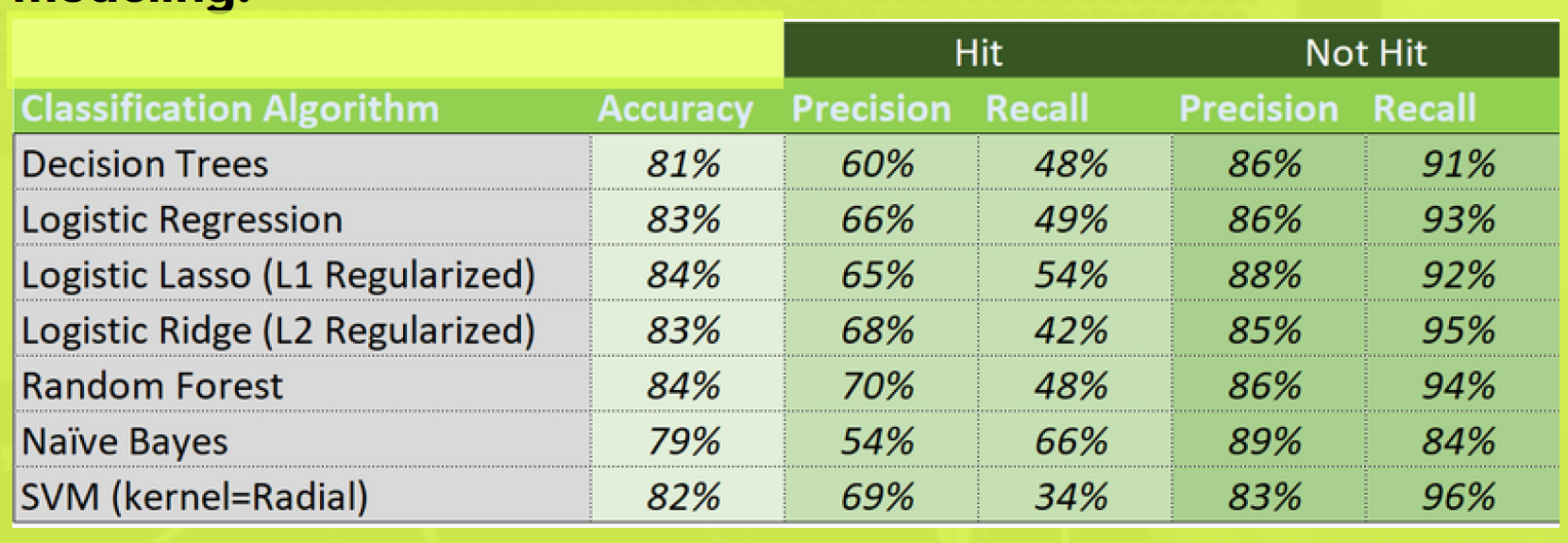
In the original paper on random forests, it was shown that the forest error rate depends on two things:

* The *correlation*between any two trees in the forest. Increasing the correlation increases the forest error rate.
* The *strength*of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.

Reducing m reduces both the correlation and the strength. Increasing it increases both. Somewhere in between is an "optimal" range of m - usually quite wide. Using the oob error rate (see below) a value of m in the range can quickly be found. This is the only adjustable parameter to which random forests is somewhat sensitive.

The mtry and ntree parameters of the random forest algorithm (randomforest package) were optimized in R to generate 500 trees with mtry=18 variables

Show only random forest:

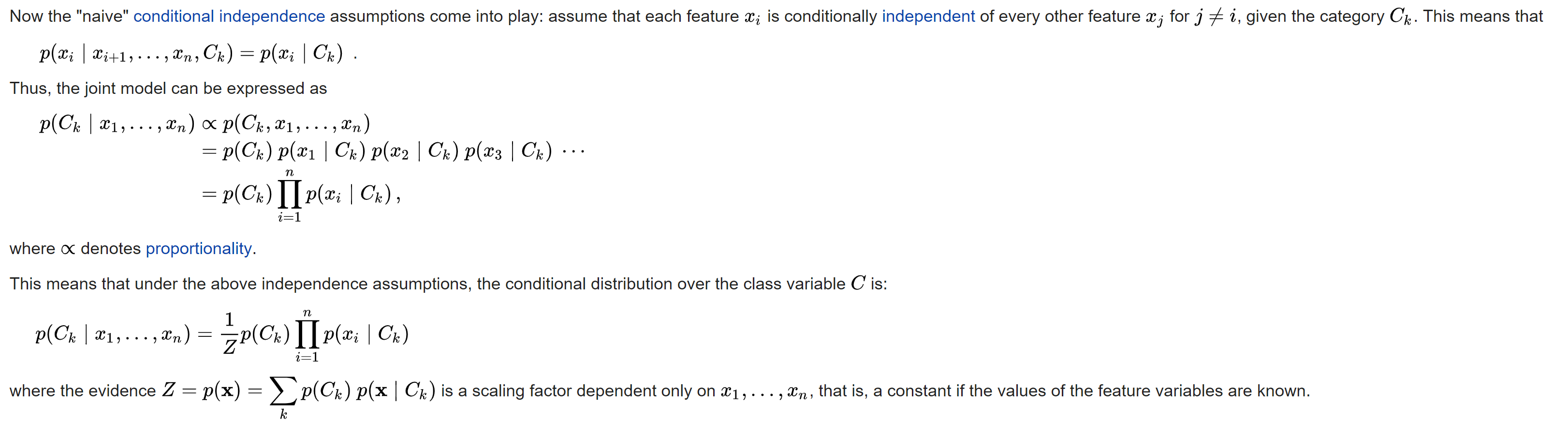


|  |  |  |
| --- | --- | --- |
| **D-Trees** | *Actual* | |
| *Predicted* | **Not hit** | **Hit** |
| **Not hit** | 1390 | 227 |
| **Hit** | 87 | 194 |

The confusion matrix corresponding the optimized random forest model translates to an accuracy of ~84% with a precision and recall of 70% and 48% respectively for the hit class and 86% and 94% for the not-hit class.

**Naïve Bayes:**

Naïve bayes remains a popular (baseline) method for [text categorization](https://en.wikipedia.org/wiki/Text_categorization), the problem of judging documents as belonging to one category or the other (such as [spam or legitimate](https://en.wikipedia.org/wiki/Spam_filtering), sports or politics, etc.) with [word frequencies](https://en.wikipedia.org/wiki/Bag_of_words) as the features. We used naïve bayes for our problem, considering that we had a lot of sparse columns related to all the song terms in our data.



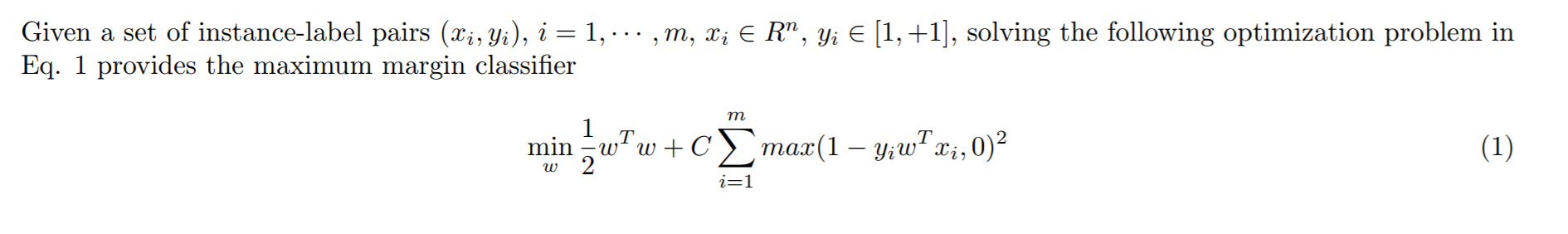
**Source: Wikipedia**

Post preprocessing the data by binning the numerical variables into 5-10 bins depending on the range and spread, we ran the naïve bayes model on the training data and got the following performance on the generalized test dataset:

|  |  |  |
| --- | --- | --- |
| **D-Trees** | *Actual* | |
| *Predicted* | **Not hit** | **Hit** |
| **Not hit** | 1236 | 142 |
| **Hit** | 241 | 279 |

The confusion matrix corresponding the optimized random forest model translates to an accuracy of ~79% with a precision and recall of 54% and 66% respectively for the hit class and 89% and 84% for the not-hit class.

**Support vector machines:**



Where C>0 is the penalty parameter

We used linear and radial kernels to train and our data with varying C values and found radial basis kernels with C= to be doing the best.

A summary of the varying the C values and the error rate is given below: