Music Recommendation System

Mukesh Gupta
IIIT, Delhi
mukesh14149@iiitd.ac.in

Pankaj Anuragi
IIIT, Delhi
pankaj14073@iiitd.ac.in

Tarun Kumar
IIIT, Delhi
tarun14110@iiitd.ac.in

Abstract

Music Recommendation is useful and has become an increasingly relevant problem in recent years, since a lot of music is now sold and consumed digitally. We have taken a Kaggle problem in which we will be asked to predict the chances of a user listening to a song repetitively after the first observable listening event within a time window was triggered. KKBOX provides the dataset in which we evaluated different classification algorithms on their ability to predict whether user will listen the song again in a specified time.

1. Introduction

Motivation

The problem here is to predict the chances of a user listening to a song repetitively after the first observable listening event. This solution of this problem can affect the overall accuracy of music recommendation systems. This solution will answer some questions like if listeners will like a new song or a new artist? And, how would it know what songs to recommend brand new users? Which song to recommend among the already listened list?

Problem Statement

To solve the problem of predicting the chances of a user listening to a song repetitively after observable listening event within a month and build a better music recommendation system. The dataset is from KKBOX, Asia's leading music streaming service, holding the world's most comprehensive Asia-Pop music library with over 30 million tracks.

1. Related Work.

The problem here is a bit unique. Our problem is to predict whether a user will listen to the already listened song again or not. There is generally already lot of work done in recommendations systems, but not specific just analyzing an aspect of recommendation system (like ours).

2. Dataset and Features

KKBOX provides a training data set consists of information of the first observable listening event for each unique user-song pair within a specific time duration. Dataset contains train file (which describe user with which song he/she listened from which device), song file (which contains the information of the songs) and members (which contain information of the user). At the end after merging out dataset contains (7377418 data points and 24 features, 1 target)

Features	Description
user_id	User's Unique id
city	User's city
bd:age	User's age
gender	User's gender
registered_via	Through which it registered at kkbox
registration_init_time	Registration date of account
expiration_date	Expiration date of account
song id	Songs's Unique id
song_length	song's length in millisecond
song_year	Song releasing year
genre_ids	genre category
artist_name	song artist name
composer	song composer
lyricist	lyrist of the song
language	song's language
source_system_tab	name of the tab where the event was triggered
source_screen_name	name of the layout a user sees
source_type	Entry point a user first plays music on mobile apps could be album, online-playlist, song etc

Problem with Dataset

1. Members dataset contains ages where '0' age members are 19932 out of 34403 members, and there are some negative age and more than 1000 age, we deal with this by putting mean of age range (10 - 100).

2. There are many empty values in many of the features, we deal this by putting Nan value there.

Preprocessing the data

Additional Features
Feature like registration date and expiration date (e.g. 20170831) does not give any meaning, so we decided to break it down into 3 features (date, month, year) and we added 2 more features registration and expiration number of days which is the number of days from choose initial date (1/1/2000).

Label Encoding
Dataset contains many features who don't have numeric values and algorithms like logistic regression, distance based methods such as KNN, support vector machines, tree based methods etc. in sklearn needs numeric arrays, So Label Encoder is used which basically map the strings to label with values 0 to n_classes-1

Dataset Evaluation
After calculating skewness and kurtosis of the dataset which gives approximately 0 value for most of the features implies that data has a normal distribution.

Out-lier Removal Techniques
Using various normality check(using skewness, kurtos, scipy normal test) on this dataset, we found this data is normally distributed so we apply two outlier techniques to remove outliers from the dataset

- **Z-Score Technique**: Used Z-score technique to detect outlier detection, it basically used standard deviation and mean of a group of data to measure central tendency and dispersion. Once we have centred and rescaled the data, anything which is beyond from threshold (threshold is set based on visualization of all data's Z-Score(Figure3.2.1)) should be considered as an outlier.

- **Interquartile range(IQR)**: It is useful in detecting the presence of outliers. Outliers are individual values that fall outside of the overall pattern of the rest of the data.

After applying these techniques, dataset (7377418 data points and 23 features 1 target) has been converted to Dataset (5843949 data points and 23 features 1 target), around 15 lakh outliers has been removed.

Standardization
Scales for different features are wildly different so we apply standardization technique to rescale the features so that they are centred around mean and standard deviation of 1, features being on different scales can alter the speed to updating the weights.

Feature Selection
In total our final dataset contains 24 features and many of which were not relevant for prediction. As a result, using all features suffered from over-fitting. In order narrow down the number of features and find the most relevant features, we conducted several feature selection algorithms.

- **Using Recursive feature elimination**: In this method estimator is trained on the initial set of features, then it removes the least important feature from current set of features and then again repeat this until given number of features is eventually reached.

- **Correlation Attribute**: In this method each feature is correlated with the class, a feature of a subset is good if it is highly correlated with the class, it uses ranker search method to rank feature based on correlation. We used cross validation (k= 10) to get the feature rank.

- **CFSSubsetEval**: In this method it evaluates the individual feature predictive ability with the degree of redundancy between them. It chooses those features which has high correlation with the class and having low inter-correlation. It uses a forward or backward step-wise strategy to navigate attribute subsets.

- **Information gain**: In this method those features selected that contribute more information will have a higher information gain value whereas those that do not add much information will have a lower score and can be removed.

- **WrapperSubsetEval**: In this method a powerful learning technique (J48 decision tree) is used and evaluate the performance of the algorithm on the dataset with different subsets of attributes selected. The subset that results in the best performance is taken as the selected subset.

Based on all of these we order all of them with increasing rank and then remove 6 most common features which comes in the bottom of all the ranking list generated by above techniques.
Features like expiration_date, registration_date, gender, registration month, registration_via and registration_year has been removed.

3. Classifier's analysis and result

We used AUC ROC metrics to evaluate our models. To avoid over-fitting and get bet analysis of the model, we used Cross validation technique for parameter tuning with k-fold = 5. Grid search over all the possible parameters is not efficient for such a large dataset. So, we applied grid search by varying small number of parameters at a time. Parameters which are interdependent on each other are taken together.

Parameter tuning of some models are done on subset of dataset. We used randomly selected 1000000 lakh samples.

Gaussian NB : We started with GaussianNB as it is simple, light to train, small memory footprint.
The Naive Bayes algorithm is an intuitive method that uses the probabilities of each attribute belonging to each class to make a prediction.

Logistic Regression : As our data is too large we used logistic regression with grid search to get the optimum parameter theta that maximizes the likelihood function.

These above classifiers are giving some decent ROC score, but we may want to increase the ROC score by trying out multiple instances of the same classifier with different parameters. So, that a linear classifier can be combined into non-linear classifiers. Adaboost is one of the technique for the same, which combines multiple weak classifier.

Boosting Techniques
AdaBoost
(Paramaters tuned on Subset of Dataset)
It was the first really successful boosting algorithm developed for binary classification. And most common algorithm used with AdaBoost are decision trees with one level. Because these trees are so short and only contain one decision for classification, they are often called decision stumps.

Classifiers	AUC ROC Score	Parameters
Naïve Bayes	0.522556	Default
Linear Regression	0.55742	Copy_x=true, normalize=true, fit_intercept=true
Logistic Regression	0.55746	C = 0.1
Decision Tree	0.556	max_features='sqrt', minimum_samples_split=10 , max_depth=13
Ada boosting	0.68	base_estimator__criterion: 'gini',base_estimator__splitter': 'best',
Gradient Boosting	0.58049	Need to tune parameter
Xgboost	0.66972	Objective='binary:logistic', eta=0.1, max_depth=14, silent=1, subsample=0.75, colsample_bytree=0.75, eval_metric=auc
LightGBM	0.71006	learning_rate=0.1,num_leaves=256,max_depth=14,num_boost_round=800,min_child_weight=16,bagging andeature_action=1,max_bin=228
Deep Neural Network		activation='softsign', epochs=100,batch_size=60, maxnorm=1,dropout_rate=0.1,lr=0.001,momentum=0.8, kernel_initializer='normal', hidden_layer = [10 8 3], optimizer='Adamax'

Adaboost gives better ROC score than previous techniques. So, here we want to try out some other boosting technique also to get an idea of the boosting techniques on this dataset. The other most common used boosting technique is Gradient boosting.

Gradient Boosting
(Parameters tuned on Subset of Dataset)
Gradient Boosting generates learners during the learning process. It build first learner to predict the value of samples and calculate the loss (difference between the outcome of the first learner and the real value). It will build the second learner to predict the loss after the first step. The step continues to learn the third, fourth…, until the certain threshold.

Gradient boosting gives better ROC score than Adaboost. So, we thought of trying out a variant of gradient boosting, i.e Extreme gradient boosting.

Xgboost
Xgboost is more flexible i.e has more customizable parameters. It is faster (at least than adaboost and gradient boost). Initially Xgboost gives ROC score of 0.61108 which is huge jump than other techniques. So, we decided to go with this techniques and did parameters tuning.
Started with high learning rate i.e 0.1 and found the optimum number of trees for this learning rate. We used the XGBoost's “cv” function, which performs cross-validation at each boosting iteration and thus returns the optimum number of trees required. Then, tuned tree-specific parameters (max_depth, min_child_weight, gamma, subsample, colsample_bytree) for decided

learning rate and number of trees. After this point, there was overfitting in our model. So, wetune regularization parameters (lambda, alpha) for xgboost which helped to reduce model complexity and enhance performance. Finally, Lowered the learning rate and decide the optimal parameters . We used early stopping to prematurely stop the training of an XGBoost model at an optimal epoch. we got ROC score of 0.66972 on Kaggle submission

LightGBM
(Parameters tuned on Subset of Dataset)
To avoid Memory error issues and training time issues we used LightGBM algorithm, which is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm. It splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree level wise rather than leaf-wise. While tuning parameter we analyses that by increases learning rate its accuracy increases but after some time it decreases drastically. So at learning rate 0.1 we got best train score, after that model started over-fitting.
By increaseing max depth and no. of leaves train score is increasing but after certain value, model started over-fitted.

Deep Neural Network
(Paramters tuned on Subset of Dataset)
A deep neural network is an ANN with multiple hidden layers between the input and output layers. It is a very powerful technique due to its ability to learn and model non-linear and complex relationships. It is notoriously difficult to configure because it has lot of parameters to tune. We evaluate model for each combination of parameters on subset of the data set.

- Activation Function

Activation function controls the non-linearity of individual neurons and when to fire. We got best ROC score by using softsign activation function.

- Batch Size and Number of Epochs

Batch size in iterative gradient descent is the number of patterns shown to the network before the weights are updated and Number of epochs is the number of times that the entire training dataset is shown to the network during training.
Because both the parameters depends on each other we tuned both parameters simultaneously. By increasing batch size we got nearly same ROC score, but by increasing no. of epochs ROC score increased.

- Dropout Regularization and Max Norm

Tuning the dropout rate for regularization in an effort to limit over-fitting and improve the model's ability to generalize. We combined dropout and max norm parameter to get best result.

- Learning Rate and Momentum

Learning rate controls how much to update the weight at the end of each batch and the momentum controls how much to let the previous update influence the current weight update.

- Kernel_initializer

There are different techniques of network weight initialization, It may be better to use different weight initialization schemes according to the activation function used on each layer.

- Number of Hidden Layer

Hidden layers makes it possible for the network to exhibit non-linear behavior, generally 2 hidden layers will enable the network to model any arbitrary function. But due to large dataset we tuned to check whether to take 2 or 3 hidden layer. And 3 hidden layers are slightly better for our dataset.

- Number of Neurons in the Hidden Layer

Number of neurons in a layer controls the representational capacity of the network, at least at that point in the topology.

- Optimization Algorithm

Optimization algorithm used to train the network, each with default parameters. There are different algorithm like SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam.

4. Problems faced

- Difficult to process dataset, getting memory issues.
- Deep neural network and some of the algorithm required more memory so we randomly sampled 10 lakh data-points did parameter tuning of our model on that subset of the data set.
- One of the table of dataset i.e "Members" have 50 percent empty rows or nan value in most of the features.
- Dataset loading problems due to heavy size
- Taking too much time to apply classifiers and tuning

6. Contributions

1. Mukesh Gupta :- Data Preprocessing, DNN, LightGBM
2. Pankaj Anuragi :- Linear, Logistic Regressiong, Decision Tree, Gradient Boosting, Adaboost.
3. Tarun Kumar Yadav:- Data Preprocessing, DNN, XGBoost

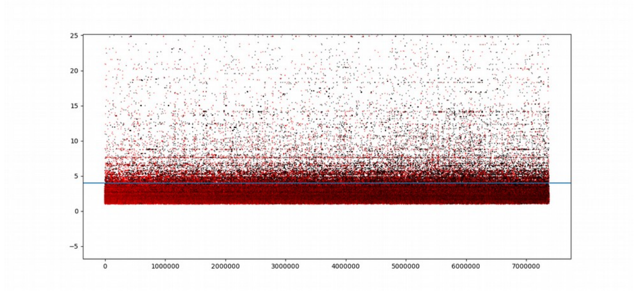

Figure3.2.1 : X-axis >Datapoint and Y-axis >ZScore

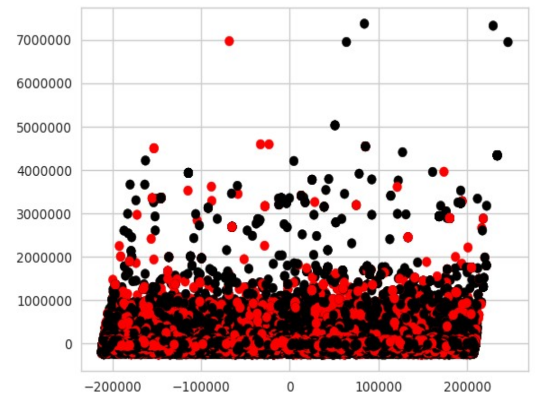

Figure 1:- Raw Data Visualization using PCA

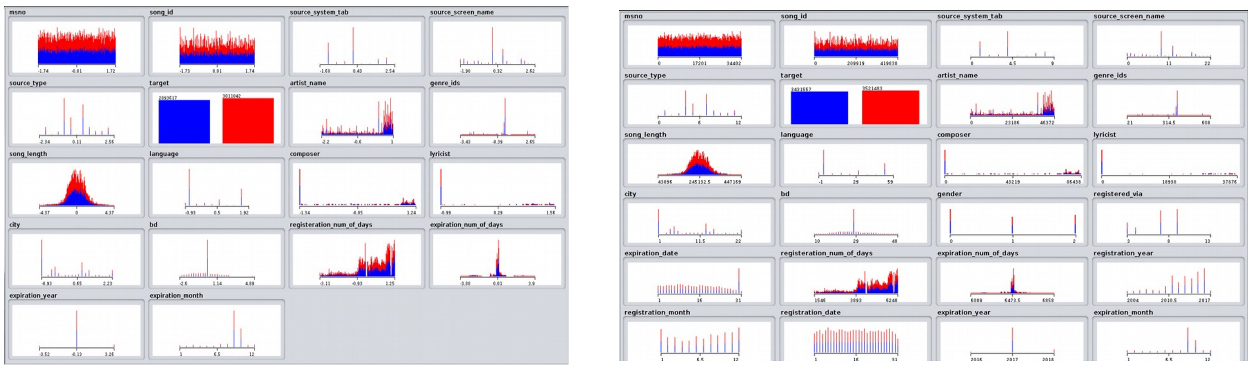

Figure 2:- Visualization after Preprocessing and before Preprocessing

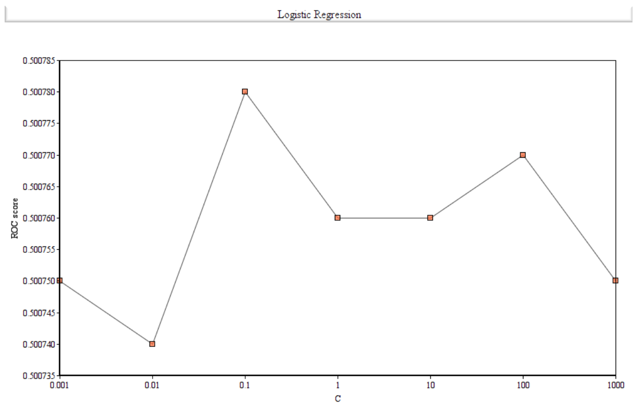

Illustration 1: Logistic(C param vs accuracy)

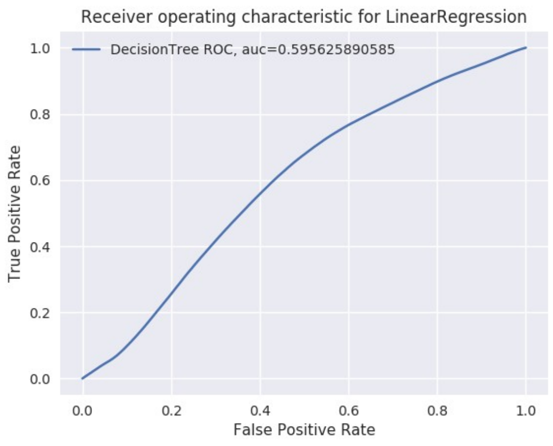

Illustration 3: ROC curve of Linear Regression

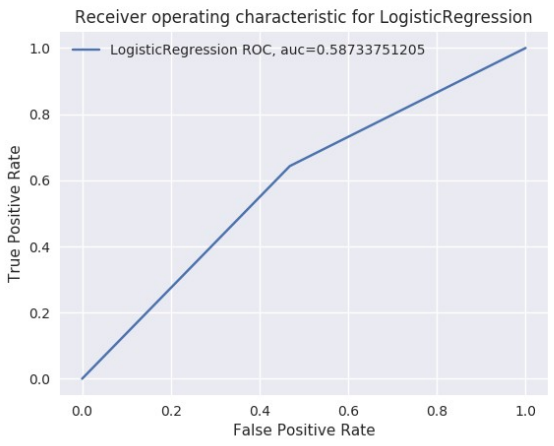

Illustration 2: ROC Curve of LogisticRegression

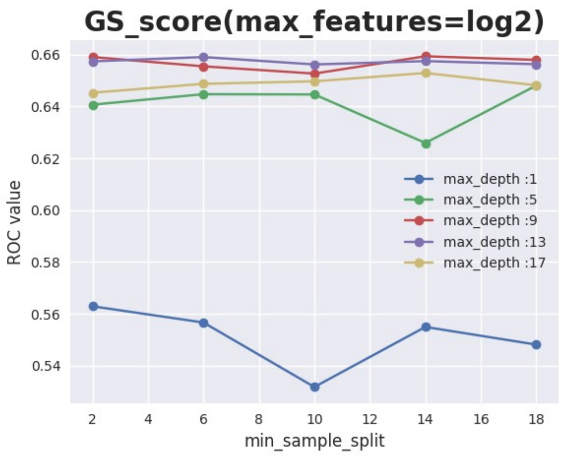

Illustration 4: Decision Tree(max features = log2)

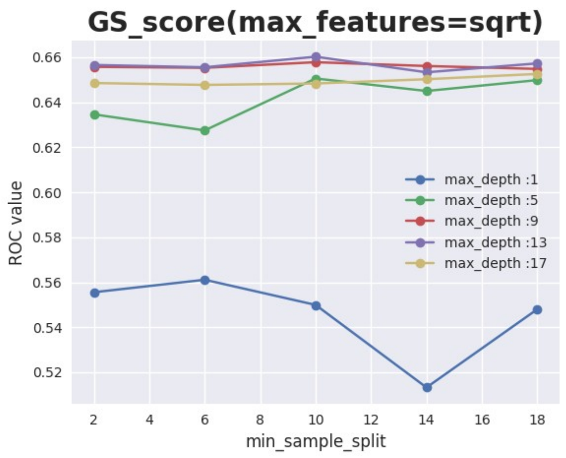

Illustration 5: Decision Tree(max features=sqrt)

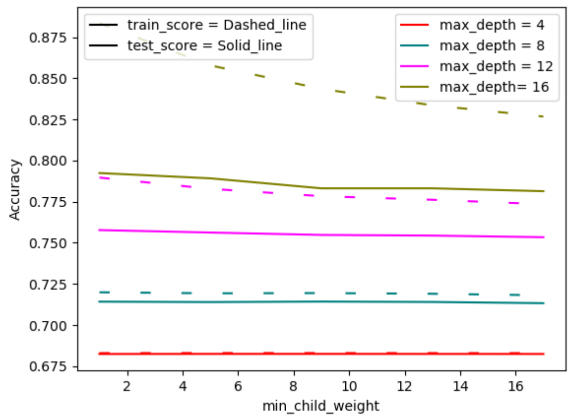

Illustration 8: XGBoost(Min-child and max_depth)

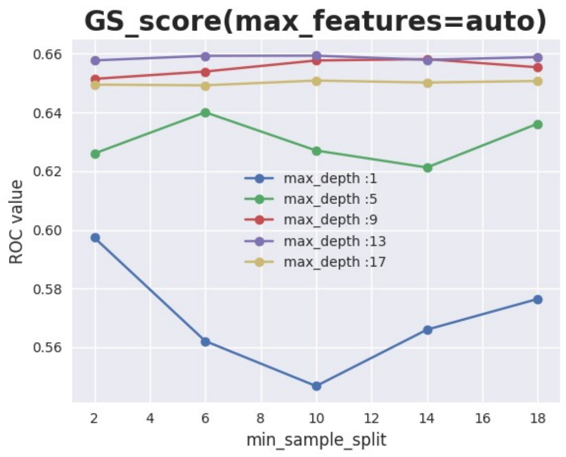

Illustration 6: Decision Tree(max features = auto)

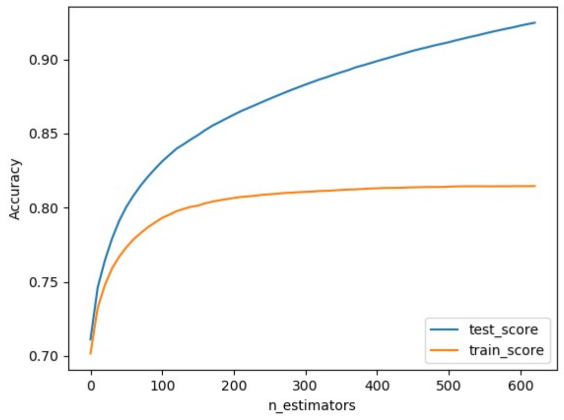

Illustration 9: XGBoost(n_estimator vs Score)

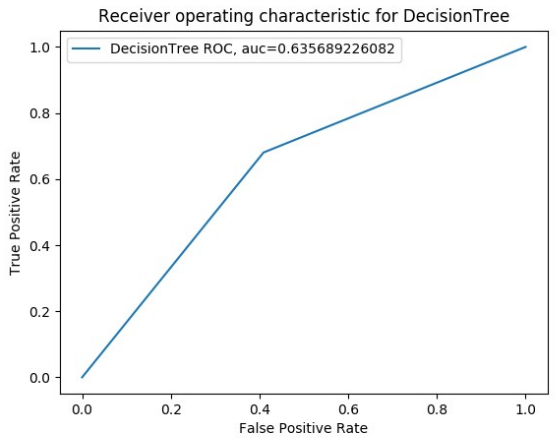

Illustration 7: Decision Tree(ROC curve)

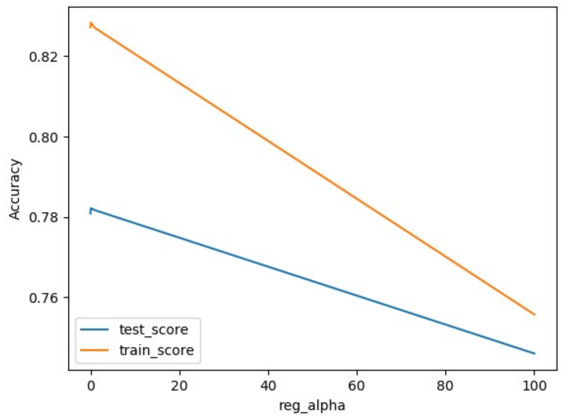

Illustration 10: XGBoost(Regularisation vs Score)

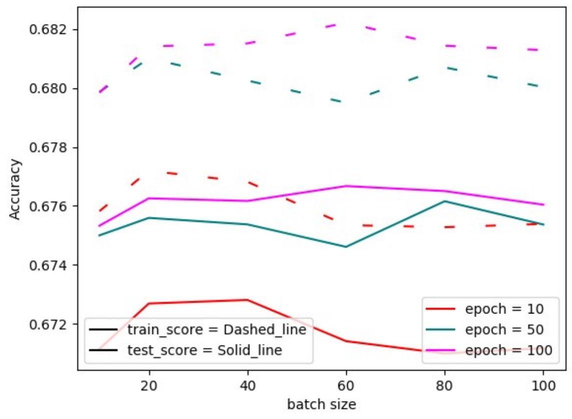

Illustration 13: DNN(Batch Size and Epoch)

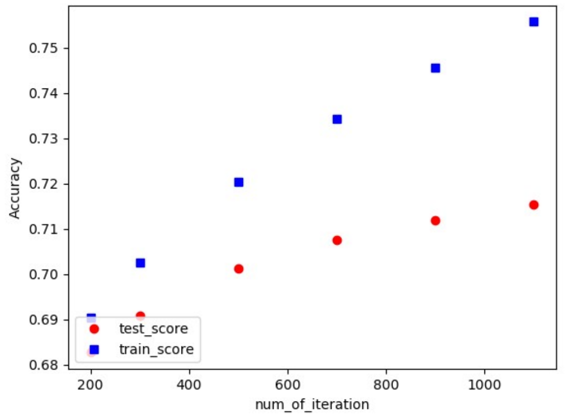

Illustration 11: LightGBM(num_of_iteration vs accuracy)

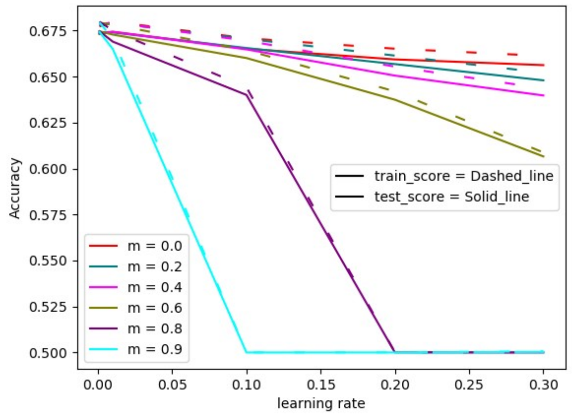

Illustration 14: DNN(learning_rate vs Accuracy)

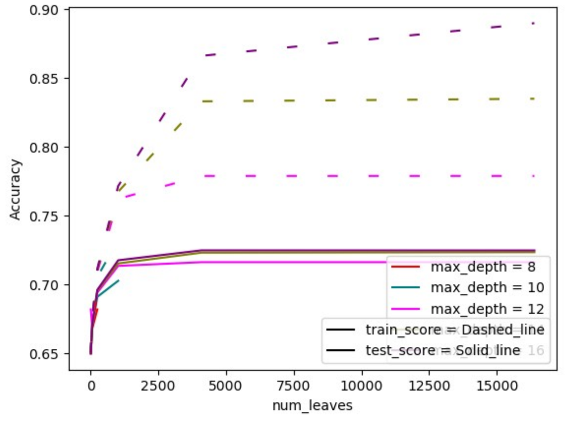

Illustration 12: LightGBM(Num_of_leaves and Max_Depth)

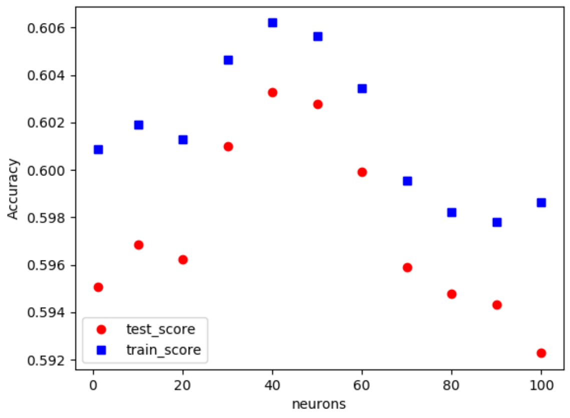

Illustration 15: DNN(neurons vs accuracy)

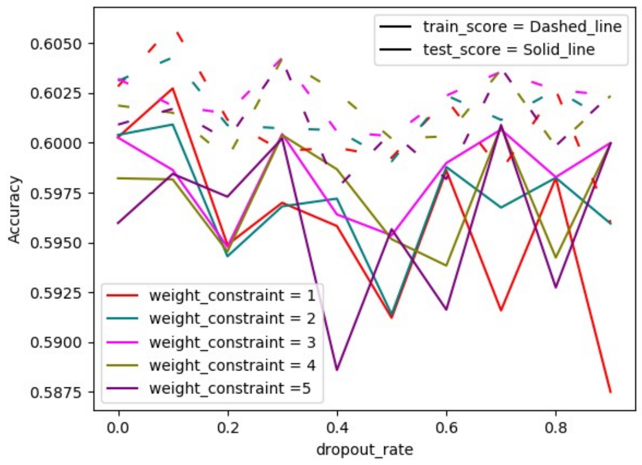

Illustration 16: DNN(Dropout rate and weight constraint)