Music Recommendation System : EE627 – Final Project

How does Spotify know what song you might want to listen to next? How does the Play Store pick an app just for you? Magic? No, in both cases, an ML-based recommendation model determines how similar music and apps are to other things you like and then serves up a recommendation.

Prashant Kumar | Ritesh Panditi | Master’s in Applied Artificial Intelligence

Recommendation Systems: Why? What? How?

- Recommendation systems play a very important role in a world of ever-increasing consumer facing databases because they offer a solution to the Paradox of Choice among users.
- Some other examples of Recommendation Systems: Netflix - Content, Amazon – Products, YouTube- Content, App Store – Applications, LinkedIn - people .
- They are systems built using Math, code and data to bring the most relevant ‘recommendation’ to the consumer.
- Broadly built using one of two strategies or by symbiotically implementing both:
- Content-based filtering uses item features to recommend other items like what the user likes, based on their previous actions or explicit feedback.
- Collaborative filtering uses similarities between users and items simultaneously to provide recommendations. These models can recommend an item to user A based on the interests of a similar user B. Furthermore, the embeddings can be learned automatically, without relying on hand-engineering of features. Neighborhood methods and Latent Factor Methods are two widely used collaborative based filtering methods.

Data format

The Music data is stored in a hierarchical structure as : Track -> Artist -> Album -> Genre 1 , Genre 2 ….. Genre K

Training Data

62M ratings scores

•296K items

•249K users

Test Data

•100K test users, 6 tracks per user

•Test user is in the training user set

In the **training** data, the **format** given is:

•user 1 : item id : ratings , user 2 : item id : ratings …. user m : item id : ratings

•Here, each user has rated x number of items (which correspond to either track id, album id , artist id or genre id) and the rating for each one of those items.

In the **test** data, the **format** given is:

•user 1 – track id - rating , user 2 – track id - rating …. user n – track id - rating

•Here, each user is given 6 tracks of which predictions are to be made to check which 3 tracks will the user will like and the 3 tracks he'll dislike. The objective of the system is to predict each user's ratings for their 6 given tracks.

Objective of the Music Recommendation System

Data Files

- trainItem2.txt - the training set
- testItem2.txt - the test set
- trackData2.txt -- Track information formatted as: TrackId | AlbumId| ArtistId | Optional GenreId_1|...|Optional GenreId_k
- albumData2.txt -- Album information formatted as: AlbumId |ArtistId | Optional Genre Id_1|...|Optional Genre Id_k
- artistData2.txt -- Artist listing formatted as: ArtistId
- genreData2.txt -- Genre listing formatted as: GenreId
- testTrack_hierarchy.txt -- UserID|TrackID|Album|Artist|Genre1|Genre2|...

Aim

- Each user is given 6 tracks of which predictions are to be made to check which 3 tracks will the user will like and the 3 tracks he'll dislike. The aim of the system which is built using Software tools like Python, PySpark, Hadoop and various libraries is to predict 3 tracks out of 6 that are most similar to the user's music taste.
- The different files given can be used to gather information and provide recommendations using various methods.

Data – Preprocessing

A snapshot of our initial Training data:

"trainIdx2_matrix.txt" data format

The data file "trainIdx2_matrix.txt" shows the user rating history with different tracks.

It contains 3 columns with the separator "|", i.e.,

userID | itemID | score

```
199808|248969|90
199808|2663|90
199808|28341|90
199808|42563|90
199808|59092|90
199808|64052|90
199808|69022|90
199808|77710|90
199808|79500|90
```

For example, the first row in the records

199808 (userID) | 248969 (itemID) | 90 (score)

A snapshot of our initial Test data:

"testTrack_hierarchy.txt" data format

For layout format in the file "testTrack_hierarchy.txt" is shown below

UserID|trackID|Album|Artist|Genre1|Genre2|Genre3 | ....

```
199810|208019|209288|None
199810|74139|277282|271146|113360|173467|173655|192976|146792|48505|133159
199810|9903|None|None|33722|123396|79926|73523
199810|242681|190640|244574|61215|17453|274088
199810|18515|146344|33168|19913|48505|154024
199810|105760|93458|11616|131552|173467|48505|133159
199812|276940|201356|163237|287681
199812|142408|112725|275191|158282|173467|242383|207648|48505|133159
199812|130023|226816|275191|158282|242383|207648|19913
```

For example, in the second row,

```
199810|74139|277282|271146|113360|173467|173655|192976|146792|48505|133159
```

199810 (user ID) | 74139 (track ID) |277282 (album ID) | 271146 (artist ID) | 113360 (genre 1) | 173467 (genre 2) | 173655 (genre 3) | 192976 (genre4) | 146792(genre 5) | 48505(genre 6) | 133159(genre 7)

- We can see that the Training data has 3 columns in all its entries with each user and all their ratings for different items (track, artist, album or genre) .
- We can see that the Test data has at least 4 columns, but can have more, depending on the number of genres.

Processing Output - without genre

jupyter output1_no_genre.txt

File Edit View Language

```
1  199810|208019|0.0|0.0
2  199810|74139|0.0|0.0
3  199810|9903|0.0|0.0
4  199810|242681|0.0|0.0
5  199810|18515|0.0|70.0
6  199810|105760|0.0|90.0
7  199812|276940|0.0|0.0
8  199812|142408|100.0|100.0
9  199812|130023|100.0|100.0
10 199812|29189|0.0|0.0
11 199812|223706|0.0|100.0
12 199812|211361|0.0|0.0
13 199813|188441|0.0|90.0
14 199813|20968|0.0|0.0
15 199813|21571|90.0|90.0
16 199813|79640|0.0|90.0
17 199813|184173|0.0|70.0
18 199813|111874|0.0|0.0
19 199814|122375|0.0|0.0
20 199814|189043|75.0|75.0
21 199814|122429|0.0|0.0
22 199814|52519|0.0|0.0
23 199814|232332|100.0|100.0
24 199814|262193|75.0|75.0
25 199815|64345|0.0|50.0
```

- We run the Output_test_v1.ipynb file to get output1_no_genre.txt which contains the userID | item_id | rating 1 | rating 2 where rating 1 , 2 correlate to the music data hierarchical structure (album and artist rating respectively) .
- Item_id corresponds to the tracks in the test dataset which are also present in the training dataset .
- E.g., in row 5, the data pertains to user 199810 who has rated 70 for the artist associated to the track 18515.
- This output is the first file being used to build our models

Processing Output – with genre

```
199810|208019|0|0|0
199810|74139|0|0|80.0
199810|9903|0|0|0
199810|242681|0|0|0
199810|18515|0|70|0
199810|105760|0|90|80.0
199812|276940|0|0|0
199812|142408|100|100|80.0
199812|130023|100|100|80.0
199812|29189|0|0|80.0
199812|223706|0|100|80.0
199812|211361|0|0|0
199813|188441|0|90|80.0
199813|20968|0|0|80.0
199813|21571|90|90|0
199813|79640|0|90|80.0
199813|184173|0|70|80.0
199813|111874|0|0|80.0
199814|122375|0|0|0
199814|189043|75|75|0
199814|122429|0|0|0
199814|52519|0|0|0
199814|232332|100|100|0
199814|262193|75|75|0
199815|64345|0|50|65.0
```

- This file is generated by running the prelim_tuning.ipynb file.
- It is structured as the previous output's hierarchy with a 5th additional column consisting the average rating of the genres.

Matrix Factorization | Model I

Matrix factorization is a simple embedding model. Given the feedback matrix A $\in R^{m \times n}$, where m is the number of users (or queries) and n is the number of items, the model learns:

- A user embedding matrix $U \in \mathbb{R}^{m \times d}$, where row i is the embedding for user i.
- An item embedding matrix $V \in \mathbb{R}^{n \times d}$, where row j is the embedding for item j.

Results from Matrix Factorization and importance of PySpark

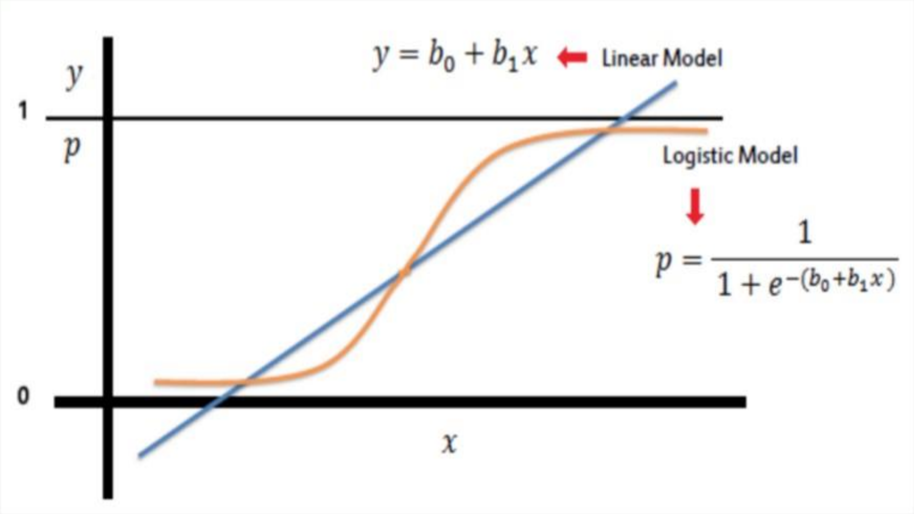
- The results of Matrix Factorization was a low-test score of 0.6000, due to the data being very sparse.
- Also, the Pandas library although powerful, is not as efficient to handle large quantities of data and perform computations.
- Hadoop is the best solution for storing and processing Big Data because Hadoop stores huge files in the form of (HDFS) Hadoop distributed file system without specifying any schema.
- It is highly scalable as any number of nodes can be added to enhance performance. In Hadoop data is highly available if there is any hardware failure also takes place.
- Spark is also a good choice for processing a large amount of structured or unstructured datasets as the data is stored in clusters. Spark will conceive to store the maximum amount of data in memory so it can spill to disk. It will store a part of the dataset in memory and therefore the remaining data on the disk.
- In the implementation of the rest of the models, PySpark has been used to handle and work with the large quantities of data more easily.

Logistic Regression | Model II:

- Logistic Regression is a statistical model that is used to perform a binary classification task using the log function.
- It models a relationship between predictor variables and a categorical response variable,
- Here between the user's music preference through what he already heard and rated being related to the new tracks he is to be recommended .
- The formula for Logistic Regression, which roots from conditional probability, is shown .
- In the image, y is the response variable, the 3 songs the user will like. x_i represents the independent variables, or the user's previous ratings for various tracks, albums, artists and genres.

❑ Logistic regression : Classification

$$\text{Prob.}(\mathbf{y}=1) = \frac{1}{1+\exp(-[b_0 + b_1\mathbf{x}_1 + b_2\mathbf{x}_2 + \cdots + b_n\mathbf{x}_n])}$$



Logistic Regression Implementation

These are two files used to implement the logistic regression

- This is the test2.txt file that has the track ID and ground truth values for each user.

	userID	trackID	ground_truth
0	200031	30877	1
1	200031	8244	1
2	200031	130183	0
3	200031	198762	0
4	200031	34503	1
...	...	...	...
5995	212234	137371	0
5996	212234	42375	0
5997	212234	277867	1
5998	212234	83093	1
5999	212234	239143	1

6000 rows × 3 columns

- This is the same output file that we obtained by processing the data, the columns are now labeled for readability .

	userID	trackID	album_score	artist_score
0	199810	208019	0.0	0.0
1	199810	74139	0.0	0.0
2	199810	9903	0.0	0.0
3	199810	242681	0.0	0.0
4	199810	18515	0.0	70.0
...	...	...	...	...
119995	249010	72192	0.0	0.0
119996	249010	86104	0.0	0.0
119997	249010	186634	90.0	90.0
119998	249010	293818	0.0	0.0
119999	249010	262811	90.0	90.0

120000 rows × 4 columns

Observations and Results: Logistic Regression

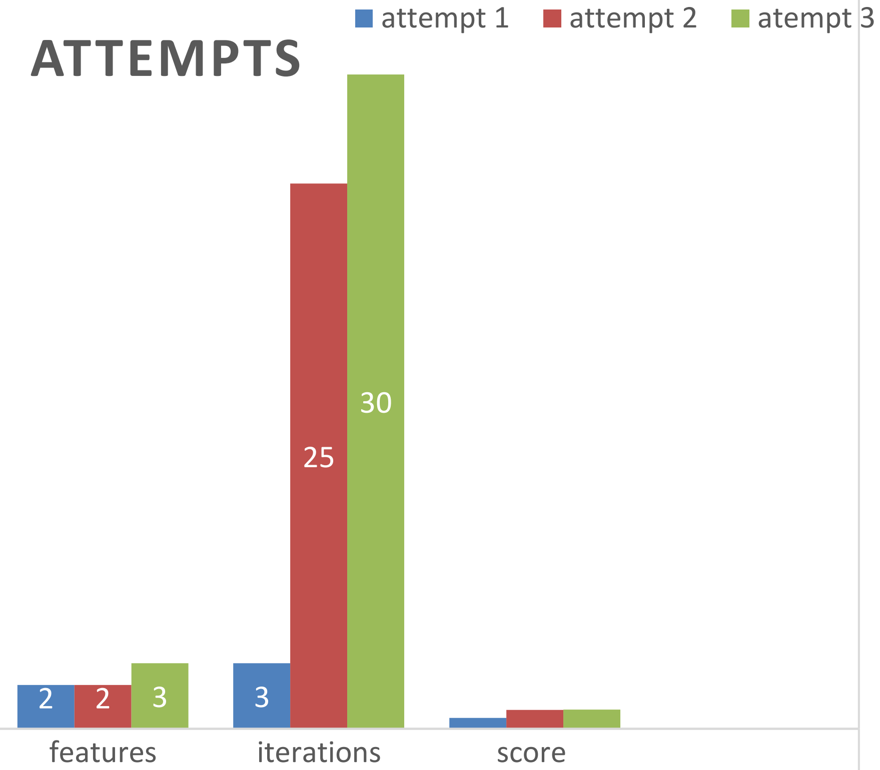
As we run the Logistic_Regression.ipynb file, we observe the following:

```
# Model- Logistic Regression

from pyspark.ml.classification import LogisticRegression

start_time = time.time()

lr = LogisticRegression(featuresCol = 'features', labelCol = 'label', maxIter=30)  # initialize a logistic regression model
lr_model = lr.fit(train_df)                                                         # fit the training data with the model

end_time = time.time()
elapsed_time = end_time - start_time
print(f'Done! Time elapsed - {elapsed_time:.2f} seconds.')
```

```
Done! Time elapsed - 4.97 seconds.
```

```
lr_model.coefficients
```

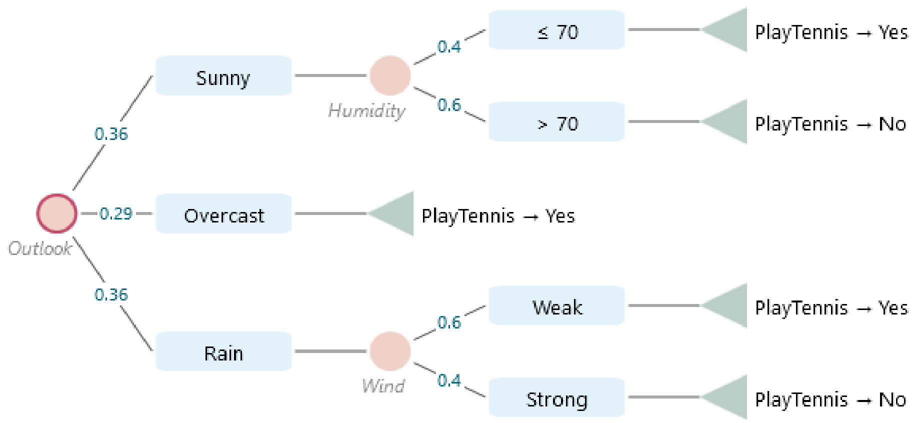
```
DenseVector([0.0312, 0.0295])
```



	features	iterations	score
attempt 1	2	3	0.5
attempt 2	2	25	0.85866
atempt 3	3	30	0.87036

Decision Trees | Model III:

- **Decision Trees** are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piecewise constant approximation



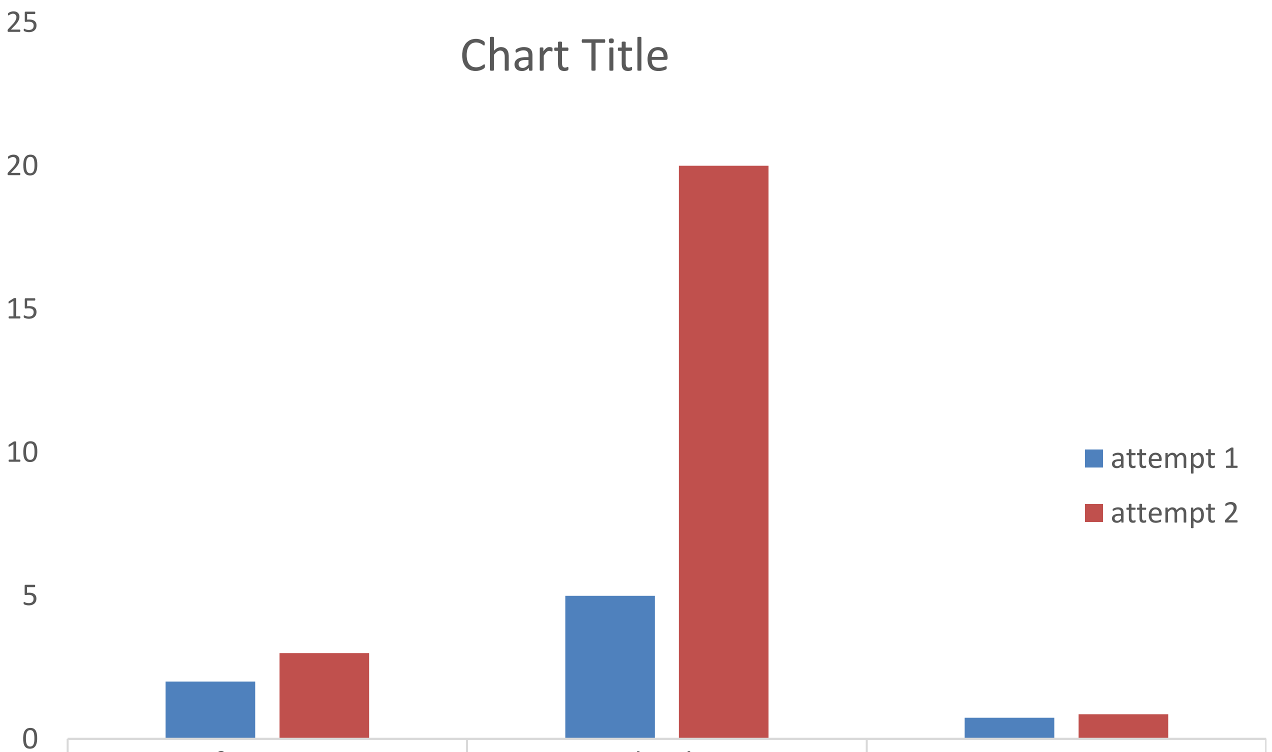
Observations and Results: Decision Trees

```
# Decision Tree

start_time = time.time()

dt = DecisionTreeClassifier(featuresCol='features', labelCol='label', maxDepth=30)
dt_model = dt.fit(train_df)

end_time = time.time()
elapsed_time = end_time - start_time
print(f'Done! Time elapsed - {elapsed_time:.2f} seconds.')
```

```
Done! Time elapsed - 2.14 seconds.
```

```
predictions_dt = dt_model.transform(test_df)
evaluator = MulticlassClassificationEvaluator(labelCol='label', predictionCol='prediction', metricName='accuracy')
accuracy = evaluator.evaluate(predictions_dt)    # evaluate decision tree model on predictions
print(f'Test Error = {1.0 - accuracy:.2%}')
```

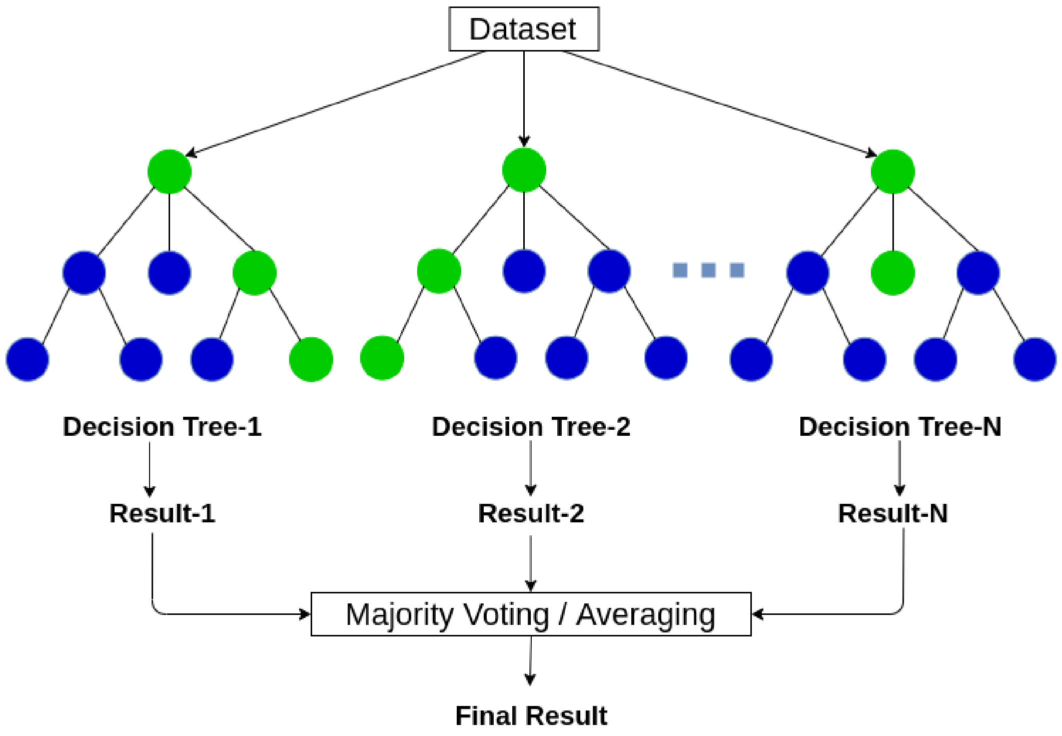
```
Test Error = 14.56%
```



	features	depth	score
attempt 1	2	5	0.74152
attempt 2	3	20	0.85983

Random Forest | Model IV:

- Random forest is a **Supervised Machine Learning Algorithm** that is used widely in Classification and Regression problems. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.
- One of the most important features of the Random Forest Algorithm is that it can handle the data set containing **continuous variables** as in the case of regression and **categorical variables** as in the case of classification. It performs better results for classification problems.
- Random forest is most accurate **ensemble classifier** and works efficiently on huge dataset. It can effectively predict the missing data accurately, even in situations **where large portions of data are missing** and without pre-processing. It combines bagging and random feature selection. Random forest contains decision trees that are combined individual learners



Observations and Results: Random Forest

```
# Random Forest

from pyspark.ml.classification import RandomForestClassifier

start_time = time.time()

rf = RandomForestClassifier(featuresCol='features', labelCol='label')
rf_model = rf.fit(train_df)

end_time = time.time()
elapsed_time = end_time - start_time
print(f'Done! Time elapsed - {elapsed_time:.2f} seconds.')

predictions_rf = rf_model.transform(test_df)

evaluator = MulticlassClassificationEvaluator(labelCol='label', predictionCol='prediction', metricName='accuracy')
accuracy = evaluator.evaluate(predictions_rf)    # evaluate random forest model on predictions
print(f'Test Error = {1.0 - accuracy:.2%}')
```

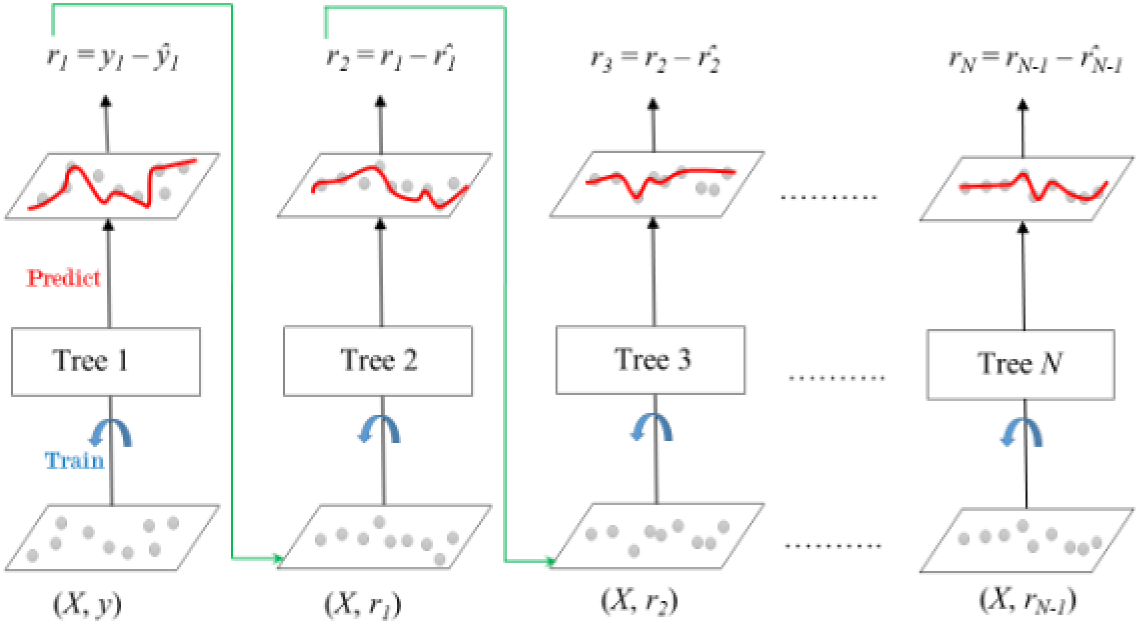
```
Done! Time elapsed - 2.18 seconds.
Test Error = 14.62%
```

We used 2 features and got a test score of 0.74100

Gradient Boosting | Model V:

- Gradient boosting works by building simpler (weak) prediction models sequentially where each model tries to predict the error left over by the previous model.
- Each predictor corrects its predecessor's error.
- The weights of the training instances are not tweaked, instead, each predictor is trained using the residual errors of predecessor as labels.
- Formula:

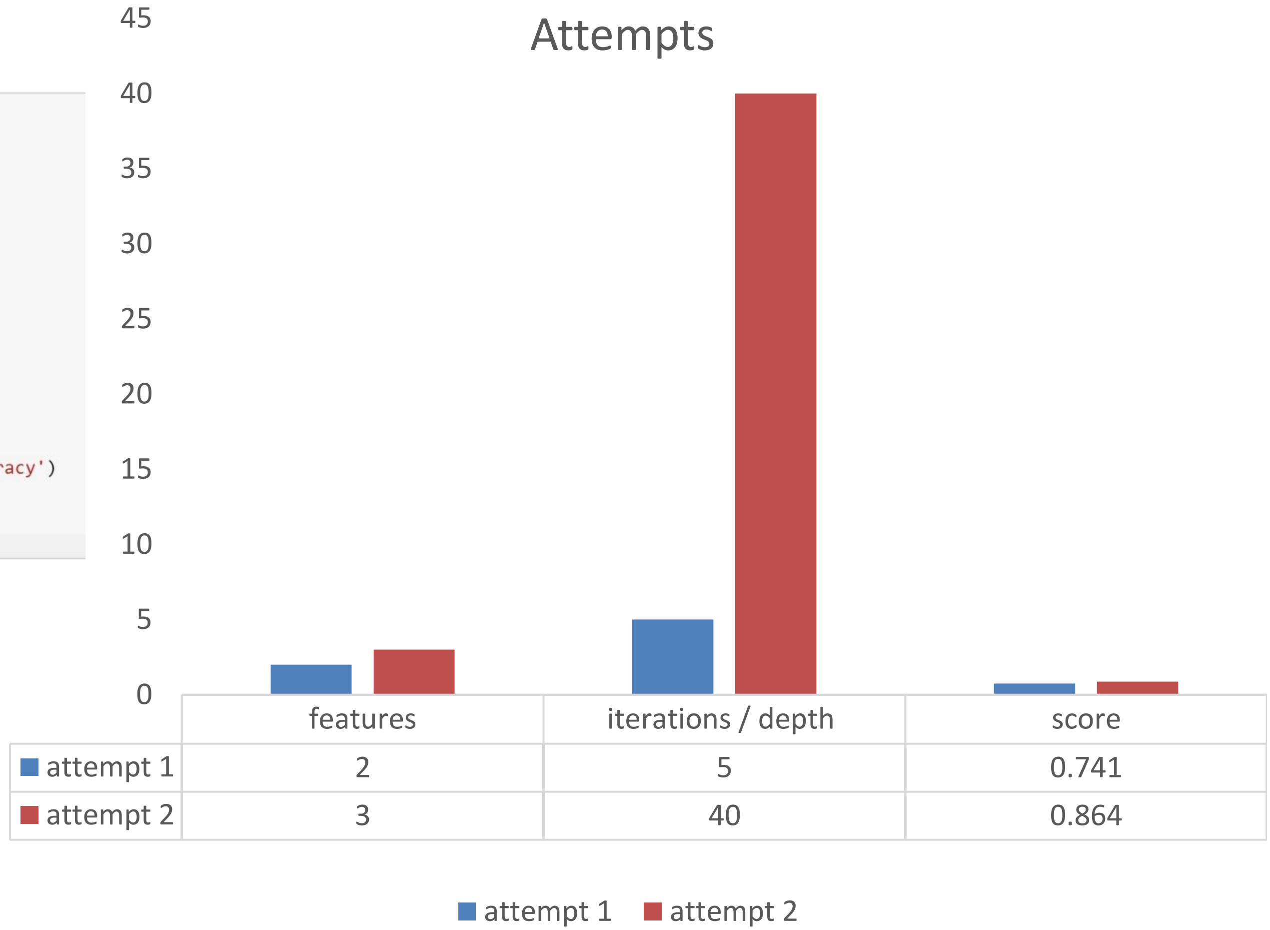
  y(pred) = y1 + (eta * r1) + (eta * r2) + ... + (eta * rN)
- eta is the learning rate
- rN is the residual error of Tree N



Observations and Results: Gradient Boosting

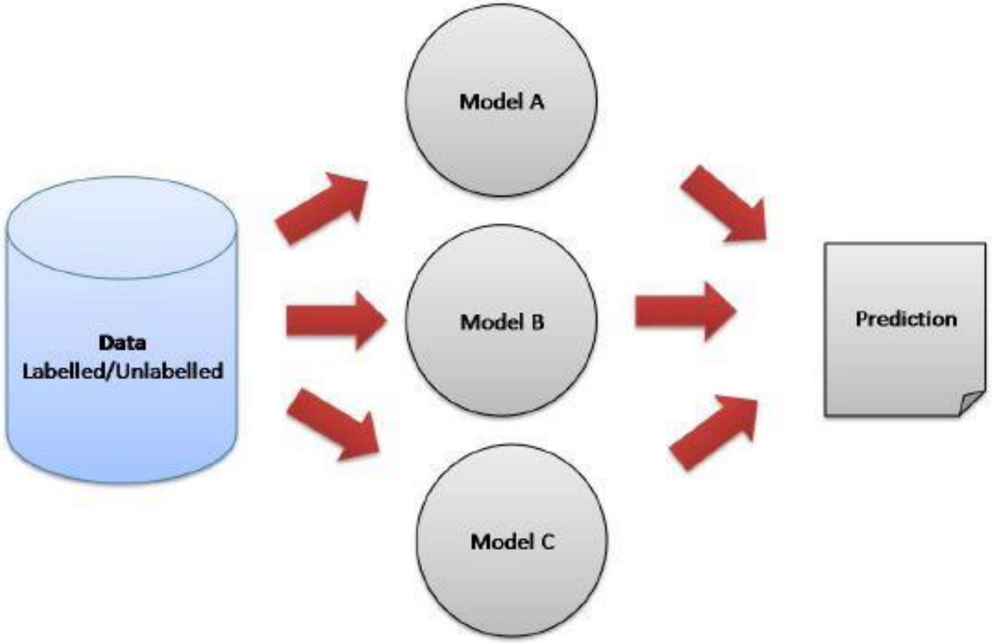
```
# Gradient Boosted Tree

from pyspark.ml.classification import GBTClassifier

start_time = time.time()

gbt = GBTClassifier(maxIter=100)
gbt_model = gbt.fit(train_df)

end_time = time.time()
elapsed_time = end_time - start_time
print(f'Done! Time elapsed - {elapsed_time:.2f} seconds.')

predictions_gbt = gbt_model.transform(test_df)

evaluator = MulticlassClassificationEvaluator(labelCol='label', predictionCol='prediction', metricName='accuracy')
accuracy = evaluator.evaluate(predictions_gbt)    # evaluate random forest model on predictions
print(f'Test Error = {1.0 - accuracy:.2%}')
```

```
Done! Time elapsed - 61.75 seconds.
Test Error = 14.62%
```

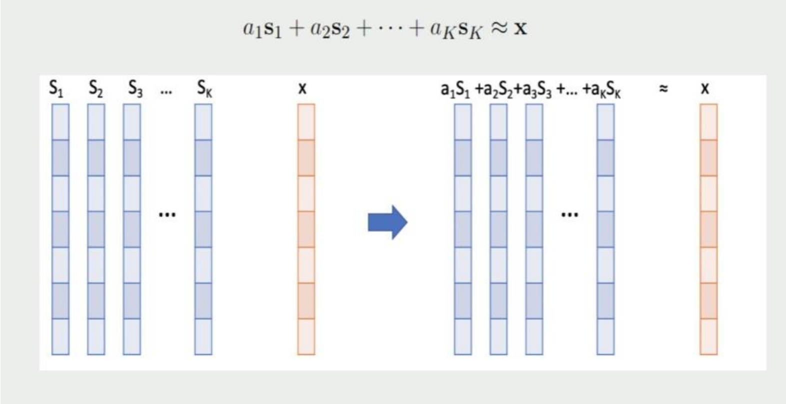


Ensemble Learning

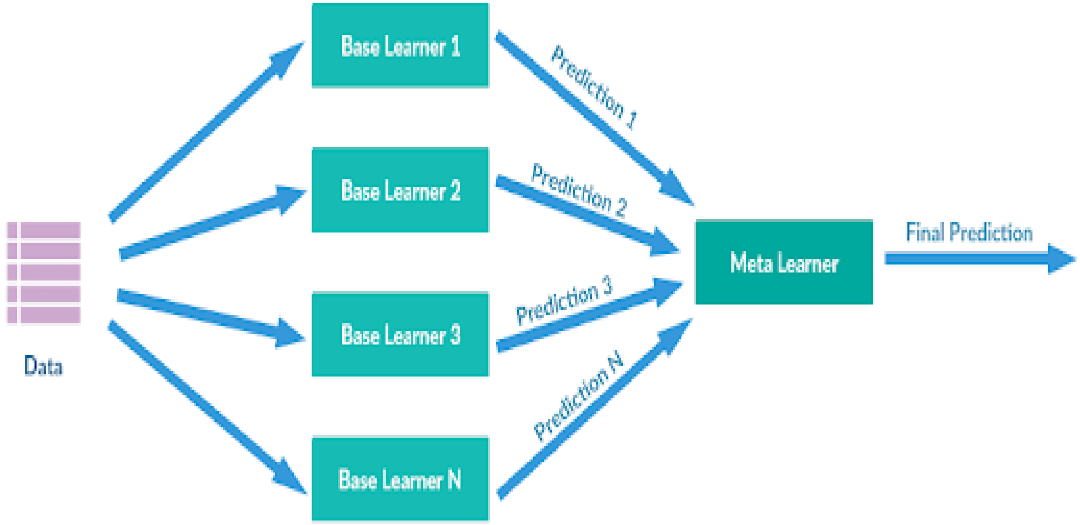
- The ensemble methods in machine learning combine the insights obtained from multiple learning models to facilitate accurate and improved decisions.
- In learning models, noise, variance, and bias are the major sources of error. The ensemble methods in machine learning help minimize these error-causing factors, thereby ensuring the accuracy and stability of machine learning (ML algorithms.
- We have seen a significant improvement of the score with the use of Ensemble Learning



Ensemble Learning yields the best results overall as it takes into consideration the results from all the different models used to predict the 'recommendation' and uses all the results to create the best outcome.



Given submitted solution vectors s1, s2, ⋯ , sK , we look for a set of weights to combine a new vector which is close to the true vector x as much as we can.



Test Score Progression

Method	Features	Parameters	Score
Sample Submission	None	None	0.5000
Logistic Regression	2 (Album + Artist Score)	Iterations = 3	0.74255
Decision Tree	2 (Album + Artist Score)	Depth = 5	0.74152
Gradient Boost	2 (Album + Artist Score)	Iterations = 5	0.74177
Random Forest	2 (Album + Artist Score)	None	0.74100
Logistic Regression	2 (Album + Artist Score)	Iterations = 25	0.84866
Alternating Least Squares	2 (Album + Artist Score)	Iterations = 10, rank = 5	0.60702
Ensemble	All above o/p	NA	0.84908
Logistic Regression	3 (Album + Artist+ Genre Score)	Iterations = 30	0.87036
Decision Tree	3 (Album + Artist+ Genre Score)	Depth = 20	0.85983
Gradient Boost	3 (Album + Artist+ Genre Score)	Iterations = 40	0.86419
Alternating Least Squares	3 (Album + Artist+ Genre Score)	Iterations = 30, rank = 20	0.72302
Final Ensemble	All above o/p	NA	0.87139

STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

stevens.edu