Music Recommendation Project

Zifeng Zhou
Chao Zhao
Jiajun Liu

Overview

Nowadays, technology companies often use recommender algorithm to recommend products, music, movie, etc. Recommender algorithm is really useful and helps all those companies make a huge profit. Now, after learning the factorial matrix and other useful algorithms. It's your turn to solve the problem which Spotify, Amazon, and Yahoo Music faced every day - recommend music to their customers.

Understanding the Dataset

TrainIterm2.txt

the training set testIterm2.txt - the test set sample_ submission.csv - a sample submission file in the correct format

TtrackData2.txt

Track information formatted as: <'TrackId'>|<'AlbumId'>|<'ArtistId'>|<'Optional GenreId_1'>|...|<'Optional GenreId_k'>

AlbumData2.txt

Album information formatted as: <'AlbumId'>|<'ArtistId'>|<'Optional GenreId_1'>|...|<'Optional GenreId_k'>

ArtistData2.txt

Artist listing formatted as: <'ArtistId'>

GenreData2.txt

Genre listing formatted as: <'GenreId'>

Project objective:
Using different algorithms to train the dataset in order to recommend new music for users

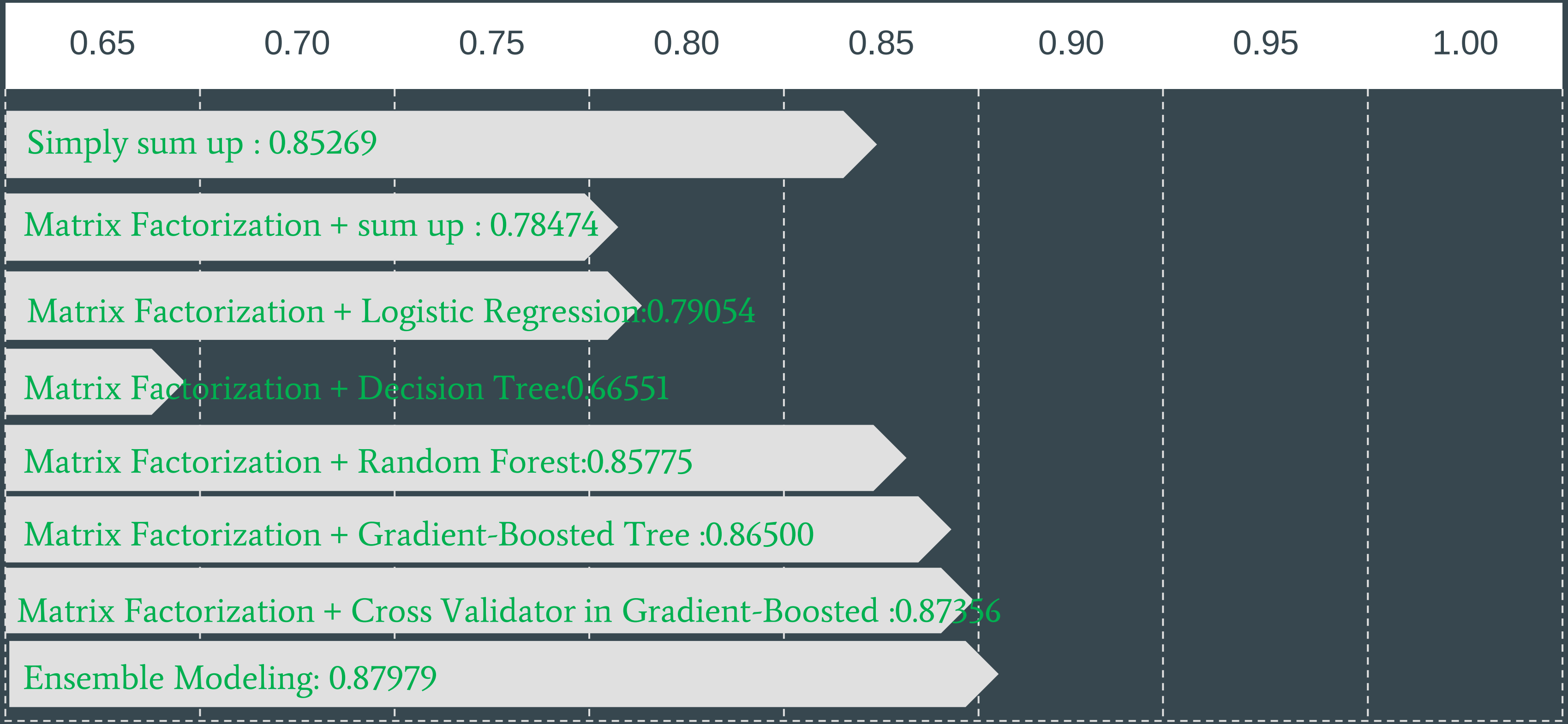
Algorithms Used

Algorithm #1	• Simply sum up all the existing rating
Algorithm #2	• Matrix Factorization + sum up all the rating
Algorithm #3	• Matrix Factorization + Logistic Regression
Algorithm #4	• Matrix Factorization + Decision Tree

Algorithms Used

Algorithm #5	• Matrix Factorization + Random Forest
Algorithm #6	• Matrix Factorization + Gradient-Boosted Tree
Algorithm #7	• Matrix Factorization + Cross Validator in Gradient-Boosted
Algorithm #8	• Ensemble Modeling

Accuracy of Prediction



Summary of Methods

Simply sum up all the existing rating

- At beginning, we read all the ratings from trainItem2.txt.
- Then we merge the userID, trackID, albumID, artistID, genreID, existing score together.
- We sum up all the existing score for one trackID, and every trackID will have a total score
- Sort trackID by the total score for every user
- We choose the top 3 trackID’s predictor as 1, and the other as 0.
- Generate a file called ’ prediction’ to calculate accuracy on Kaggle.

Summary of Methods

Simply sum up all the existing rating

```
def score():
    for user in testing_item:
        scores = [0 for n in range(6)]
        testing_item[user] = dict(zip(testing_item[user], scores))
        for trackID in testing_item[user]:
            for each in track_data[trackID]:
                if(each in training_item[user]):
                    testing_item[user][trackID] += int(training_item[user][each])
        #print(testing_item[user])
```

We sum up all the existing score for one trackID, and every trackID will have a total score

```
import pandas as pd

training_item = read_training()
testing_item = read_testing()
track_data = read_track()
score()
a = []
b = []
for every_user in testing_item:
    for every_trackID in testing_item[every_user]:
        a.append(every_user+'_'+every_trackID)
        b.append(testing_item[every_user][every_trackID])
for counter in range(0,len(b),6):
    location=[0,0,0]
    for index in range(0,6):
        if b[counter+index]>b[location[0]]:
            location[2]=location[1]
            location[1]=location[0]
            location[0]=counter+index
        elif b[counter+index]>b[location[1]]:
            location[2]=location[1]
            location[1]=counter+index
        elif b[counter+index]>b[location[2]]:
            location[2]=counter+index
        else:
            continue
    for i in range(counter,counter+6):
        if i in location:
            b[i]=1
        else:
            b[i]=0
```

Sort trackID by the total score for every user and choose the top 3 trackID's predictor as 1, and the other as 0.

Summary of Methods

Matrix Factorization + sum up all the rating

- At beginning, we read all the ratings from trainItem2.txt.
- We want to build a matrix based on the item and user to know that a user's rating on every user.
- Because a user couldn't rate all the item, so the matrix is a sparse matrix.
- We use spark.als to build the matrix.
- We merge the userID ,trackID, albumID, artistID, genreID together as the test data to predict all users' rating on every item
- We sum up all the score for one trackID, and every trackID will have a total score
- Sort trackID by the total score for every user
- We choose the top 3 trackID's predictor as 1, and the other as 0.
- Generate a file called ' mf_sum.csv' to calculate accuracy on Kaggle.

Summary of Methods

Matrix Factorization + sum up all the rating

```
from pyspark.mllib.recommendation import ALS, MatrixFactorizationModel, Rating
train_data = sc.textFile("trainItem.data")
train_ratings = train_data.map(lambda l: l.split(',')).map(lambda l: Rating(int(l[0]), int(l[1]), float(l[2])))
```

```
rank = 10
numIterations = 10
model = ALS.train(train_ratings, rank, numIterations)
```

Use spark.als to train the model based on the trainItem.data, the trainItem.data is rated userID+rated ItemID+existing rating

```
testFile = sc.textFile("trackItem.data")
test_ratings = testFile.map(lambda l: l.split(','))\
 .map(lambda l: Rating(int(l[0]), int(l[1]), float(l[2])))
```

```
testdata = test_ratings.map(lambda p: (p[0], p[1]))
print(testdata.count())
predictions = model.predictAll(testdata).map(lambda r: ((r[0], r[1]), r[2]))
```

Use spark.als to train the model based on the trackItem.data, the trainItem.data is all the userID+ItemID

Summary of Methods

Matrix Factorization + sum up all the rating

```
score=[]
for i in range(len(UserID)):
    rating=album[i]+artist[i]+genre[i]
    score.append(rating)
```

We sum up all the existing score and predicted score for one trackID, and every trackID will have a total score

```
track=[]
Score_new=[]
Score_temp=[]
count=0
for i in range(len(score)):
    count+=1
    track.append(str(UserID[i])+'_'+str(TrackID[i]))
    Score_temp.append(score[i])
    if count%6==0:
        Score_temp.sort(reverse=True)
        for j in range(count-6,count):
            if score[j] in Score_temp[:3]:
                Score_new.append(1)
            else:
                Score_new.append(0)
        Score_temp=[]
```

Sort trackID by the total score for every user and choose the top 3 trackID's predictor as 1, and the other as 0.

Summary of Methods

Matrix Factorization + Logistic Regression

- From ‘test2.txt’, we could get the data which contains the ground-truth of the track ID recommendations.
- We get the rating on every item from the built matrix before
- We merge the userID, trackID, predictor from the data above and album rating, artist rating and genre rating together as the train data
- Similarly, we could get the test data.
- Use pipeline to read the train data and test data.
- from pyspark.ml.classification import LogisticRegression to train the data and we set the maxIter is 10.
- Use the test data to get the prediction and probability
- From the prediction and probability, we could get two probability, if the latter probability is more than the previous probability, the prediction will be 1,otherwise we will get 0. So we compared the latter probability.
- Generate a file called ’ pre_lr2.csv’ to calculate accuracy on Kaggle.

Summary of Methods

Matrix Factorization + Logistic Regression

```
from pyspark.ml.feature import OneHotEncoderEstimator,StringIndexer, VectorAssembler
categoricalColumns = ['UserID','TrackID']
stages = []
for categoricalCol in categoricalColumns:
    stringIndexer = StringIndexer(inputCol = categoricalCol, outputCol =
    categoricalCol + 'Index')
    encoder=OneHotEncoderEstimator(inputCols=[stringIndexer.getOutputCol()],
    outputCols=[categoricalCol + "classVec"])
    stages += [stringIndexer, encoder]
label_stringIdx = StringIndexer(inputCol = 'rating', outputCol = 'label')
stages += [label_stringIdx]
numericCols = ['album','artist','genre']
assemblerInputs = numericCols
assembler = VectorAssembler(inputCols=assemblerInputs,
outputCol="features")
stages += [assembler]
```

```
from pyspark.ml import Pipeline
pipeline = Pipeline(stages = stages)
pipelineModel = pipeline.fit(df)
df = pipelineModel.transform(df)
selectedCols = ['label', 'features'] + cols
df = df.select(selectedCols)
df.printSchema()
```

Use pipeline to read the data.

Summary of Methods

Matrix Factorization + Logistic Regression

```
train=df
test=df2
print("Training Dataset Count: " + str(train.count()))
print("Test Dataset Count: " + str(test.count()))
```

```
Training Dataset Count: 6000
Test Dataset Count: 120000
```

Choose the train data and test data.

```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(featuresCol = 'features', labelCol = 'label',maxIter=10)
lrModel = lr.fit(train)
predictions = lrModel.transform(test)
predictions.select('UserID_test','TrackID_test', 'label', 'rawPrediction',
'prediction', 'probability').show(10)
```

```
+-----------+------------+-----+--------------------+----------+--------------------+
|UserID_test|TrackID_test|label|       rawPrediction|prediction|         probability|
+-----------+------------+-----+--------------------+----------+--------------------+
|     199810|      208019|  0.0|[0.69742575813580...|       0.0|[0.66761678270990...|
|     199810|       74139|  0.0|[0.48014493890673...|       0.0|[0.61778209939732...|
|     199810|        9903|  0.0|[0.61348783697000...|       0.0|[0.64873601371575...|
|     199810|      242681|  0.0|[0.03987051092837...|       0.0|[0.50996630751566...|
|     199810|       18515|  0.0|[-0.0886615330947...|       1.0|[0.47784912524978...|
|     199810|      105760|  0.0|[-0.8401843162418...|       1.0|[0.30149596628667...|
|     199812|      276940|  0.0|[-2.6006701238144...|       1.0|[0.06909530476480...|
|     199812|      142408|  0.0|[-1.5988261813982...|       1.0|[0.16814573614892...|
|     199812|      130023|  0.0|[-1.5386556129745...|       1.0|[0.17673079408379...|
|     199812|       29189|  0.0|[-1.5165598014193...|       1.0|[0.17996866487025...|
+-----------+------------+-----+--------------------+----------+--------------------+
```

Use Logistic Regression to train the data and get the prediction.

Summary of Methods

Matrix Factorization + Logistic Regression

```
probability=[]
userID=[]
trackID=[]
for temp in predictions.collect():
    probability.append(temp[-2][1])
    userID.append(temp[2])
    trackID.append(temp[3])
```

```
track=[]
Score_new=[]
Score_temp=[]
count=0
for i in range(len(probability)):
    count+=1
    track.append(str(userID[i])+'_'+str(trackID[i]))
    Score_temp.append(probability[i])
    if count%6==0:
        Score_temp.sort(reverse=True)
        for j in range(count-6,count):
            if probability[j] in Score_temp[:3]:
                Score_new.append(1)
            else:
                Score_new.append(0)
        Score_temp=[]
```

We compared the latter probability. Sort trackID by the latter probability for every user. Then We choose the top 3 trackID’s predictor as 1, and the other as 0.

Summary of Methods

Matrix Factorization + Decision Tree

- From ‘test2.txt’, we could get the data which contains the ground-truth of the track ID recommendations.
- We get the rating on every item from the built matrix before
- We merge the userID, trackID, predictor from the data above and album rating, artist rating and genre rating together as the train data
- Similarly, we could get the test data.
- Use pipeline to read the train data and test data.
- from pyspark.ml.classification import DecisionTreeClassifier to train the data and we set the maxdept is 3.
- Use the test data to get the prediction and probability
- From the prediction and probability, we could get two probability, if the latter probability is more than the previous probability, the prediction will be 1,otherwise we will get 0. So we compared the latter probability.
- Generate a file called ’ pre_dt2.csv’ to calculate accuracy on Kaggle.

Summary of Methods

Matrix Factorization + Decision Tree

```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(featuresCol = 'features', labelCol = 'label',maxIter=10)
lrModel = lr.fit(train)
predictions = lrModel.transform(test)
predictions.select('UserID_test','TrackID_test', 'label', 'rawPrediction',
'prediction', 'probability').show(10)
```

UserID_test	TrackID_test	label	rawPrediction	prediction	probability
199810	208019	0.0	[0.69742575813580...	0.0	[0.66761678270990...
199810	74139	0.0	[0.48014493890673...	0.0	[0.61778209939732...
199810	9903	0.0	[0.61348783697000...	0.0	[0.64873601371575...
199810	242681	0.0	[0.03987051092837...	0.0	[0.50996630751566...
199810	18515	0.0	[-0.0886615330947...	1.0	[0.47784912524978...
199810	105760	0.0	[-0.8401843162418...	1.0	[0.30149596628667...
199812	276940	0.0	[-2.6006701238144...	1.0	[0.06909530476480...
199812	142408	0.0	[-1.5988261813982...	1.0	[0.16814573614892...
199812	130023	0.0	[-1.5386556129745...	1.0	[0.17673079408379...
199812	29189	0.0	[-1.5165598014193...	1.0	[0.17996866487025...

Use Decision Tree to train the data and get the prediction.

Summary of Methods

Matrix Factorization + Random Forest

• From ‘test2.txt’, we could get the data which contains the ground-truth of the track ID recommendations.
• We get the rating on every item from the built matrix before
• We merge the userID, trackID, predictor from the data above and album rating, artist rating and genre rating together as the train data
• Similarly, we could get the test data.
• Use pipeline to read the train data and test data.
• from pyspark.ml.classification import RandomForestClassifier to train the data
• Use the test data to get the prediction and probability
• From the prediction and probability, we could get two probability, if the latter probability is more than the previous probability, the prediction will be 1,otherwise we will get 0. So we compared the latter probability.
• Generate a file called ’ pre_rf2.csv’ to calculate accuracy on Kaggle.

Summary of Methods

Matrix Factorization + Random Forest

```python
from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'label')
rfModel = rf.fit(train)
predictions = rfModel.transform(test)
predictions.select('UserID_test','TrackID_test', 'label', 'rawPrediction',
'prediction', 'probability').show(10)
```

```
+-----------+------------+-----+--------------------+----------+--------------------+
|UserID_test|TrackID_test|label|       rawPrediction|prediction|         probability|
+-----------+------------+-----+--------------------+----------+--------------------+
|     199810|      208019|  0.0|[14.0467976338286...|       0.0|[0.70233988169143...|
|     199810|       74139|  0.0|[15.8473663153623...|       0.0|[0.79236831576811...|
|     199810|        9903|  0.0|[10.2863177872613...|       0.0|[0.51431588936306...|
|     199810|      242681|  0.0|[14.1491831065770...|       0.0|[0.70745915532885...|
|     199810|       18515|  0.0|[12.4395448951316...|       0.0|[0.62197724475658...|
|     199810|      105760|  0.0|[2.39766727784864...|       1.0|[0.11988336389243...|
|     199812|      276940|  0.0|[10.9356449525111...|       0.0|[0.54678224762555...|
|     199812|      142408|  0.0|[2.49603042543080...|       1.0|[0.12480152127154...|
|     199812|      130023|  0.0|[2.49603042543080...|       1.0|[0.12480152127154...|
|     199812|       29189|  0.0|[9.88872711111153...|       1.0|[0.49443635555557...|
+-----------+------------+-----+--------------------+----------+--------------------+
only showing top 10 rows
```

Use Random Forest
to train the data and get the
prediction.

Summary of Methods

Matrix Factorization + Gradient-Boosted Tree

• From ‘test2.txt’, we could get the data which contains the ground-truth of the track ID recommendations.
• We get the rating on every item from the built matrix before
• We merge the userID, trackID, predictor from the data above and album rating, artist rating and genre rating together as the train data
• Similarly, we could get the test data.
• Use pipeline to read the train data and test data.
• from pyspark.ml.classification import GBTClassifier to train the data
• Use the test data to get the prediction and probability
• From the prediction and probability, we could get two probability, if the latter probability is more than the previous probability, the prediction will be 1,otherwise we will get 0. So we compared the latter probability.
• Generate a file called ’ pre_gbt2.csv’ to calculate accuracy on Kaggle.

Summary of Methods

Matrix Factorization + Gradient-Boosted Tree

```
from pyspark.ml.classification import GBTClassifier
gbt = GBTClassifier(maxIter=10)
gbtModel = gbt.fit(train)
predictions = gbtModel.transform(test)
predictions.select('UserID_test','TrackID_test', 'label', 'rawPrediction',
'prediction', 'probability').show(10)
```

UserID_test	TrackID_test	label	rawPrediction	prediction	probability
199810	208019	0.0	[-0.0770435642131...	1.0	[0.46155425541634...
199810	74139	0.0	[0.72295643578683...	0.0	[0.80936862385671...
199810	9903	0.0	[0.05914171938033...	0.0	[0.52953643076989...
199810	242681	0.0	[0.74071329988719...	0.0	[0.81478796348015...
199810	18515	0.0	[-0.4442236349852...	1.0	[0.29143036030506...
199810	105760	0.0	[-0.7832599241730...	1.0	[0.17271308095706...
199812	276940	0.0	[0.31202938686409...	0.0	[0.65114108954880...
199812	142408	0.0	[-1.0387861494968...	1.0	[0.11129586191939...
199812	130023	0.0	[-1.1850884089672...	1.0	[0.08547531668983...
199812	29189	0.0	[0.28248220860500...	0.0	[0.63760043068490...

Use Gradient-Boosted Tree to train the data and get the prediction.

Summary of Methods

Matrix Factorization + Cross Validator in Gradient-Boosted

• From ‘test2.txt’, we could get the data which contains the ground-truth of the track ID recommendations.
• We get the rating on every item from the built matrix before
• We merge the userID, trackID, predictor from the data above and album rating, artist rating and genre rating together as the train data
• Similarly, we could get the test data.
• Use pipeline to read the train data and test data.
• Use GBTClassifier, ParamGridBuilder, CrossValidator to train the data • Use the test data to get the prediction and probability
• From the prediction and probability, we could get two probability, if the latter probability is more than the previous probability, the prediction will be 1,otherwise we will get 0. So we compared the latter probability.
• Generate a file called ’ pre_cvgbt2.csv’ to calculate accuracy on Kaggle.

Summary of Methods

Matrix Factorization + Cross Validator in Gradient-Boosted

```python
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
from pyspark.ml.evaluation import BinaryClassificationEvaluator
paramGrid = (ParamGridBuilder()
.addGrid(gbt.maxDepth, [2, 4, 6])
.addGrid(gbt.maxBins, [20, 60])
.addGrid(gbt.maxIter, [10, 20])
.build())
evaluator = BinaryClassificationEvaluator()
cv = CrossValidator(estimator=gbt, estimatorParamMaps=paramGrid,
evaluator=evaluator, numFolds=5)
# Run cross validations. This can take about 6 to 10 minutes since it
#is training over 20 trees!
cvModel = cv.fit(train)
predictions = cvModel.transform(test)
predictions.select('UserID_test','TrackID_test', 'label', 'rawPrediction',
'prediction', 'probability').show(10)
```

```
+-----------+------------+-----+--------------------+----------+--------------------+
|UserID_test|TrackID_test|label|       rawPrediction|prediction|         probability|
+-----------+------------+-----+--------------------+----------+--------------------+
|     199810|      208019|  0.0|[0.21528148985829...|       0.0|[0.60600809477371...|
|     199810|       74139|  0.0|[0.69694748301130...|       0.0|[0.80121332434369...|
|     199810|        9903|  0.0|[0.10547533785238...|       0.0|[0.55254296563363...|
|     199810|      242681|  0.0|[0.66765474929792...|       0.0|[0.79171753192905...|
|     199810|       18515|  0.0|[-0.7240297020172...|       1.0|[0.19030040530103...|
|     199810|      105760|  0.0|[-0.6293812255112...|       1.0|[0.22118700294670...|
|     199812|      276940|  0.0|[0.25691602524122...|       0.0|[0.62570436602802...|
|     199812|      142408|  0.0|[-1.5109349834914...|       1.0|[0.04644758301517...|
|     199812|      130023|  0.0|[-1.5109349834914...|       1.0|[0.04644758301517...|
|     199812|       29189|  0.0|[-0.0347013169714...|       1.0|[0.48265630260806...|
+-----------+------------+-----+--------------------+----------+--------------------+
only showing top 10 rows
```

Use Use GBTClassifier, ParamGridBuilder, CrossValidator to train the data and get the prediction.

Summary of Methods

Ensemble Modeling

$$\mathbf{s}_{\text{ensemble}} = a_1\mathbf{s}_1 + a_2\mathbf{s}_2 + \cdots + a_K\mathbf{s}_K = \mathbf{S}\cdot\mathbf{a}_{\text{LS}} = \mathbf{S}\left(\mathbf{S}^T\mathbf{S}\right)^{-1}\mathbf{S}^T\mathbf{x}$$

- Based on the formula above, we collect the accuracy and the submissions before. These submitted solutions are s1; s2; ⋯ ; sK
- We changed the binary set {0,1} in those submissions to {-1,1}.
- $$\mathbf{S}^T\mathbf{x} = \begin{bmatrix} \mathbf{s}_1^T\mathbf{x} \\ \mathbf{s}_2^T\mathbf{x} \\ \vdots \\ \mathbf{s}_K^T\mathbf{x} \end{bmatrix} = \begin{bmatrix} N(2P_1 - 1) \\ N(2P_2 - 1) \\ \vdots \\ N(2P_K - 1) \end{bmatrix}$$

  where $P_1,\ P_2,\ \cdots,\ P_K$ are all your scores from the Kaggle submissions.

Summary of Methods

Ensemble Modeling

```
csvlist=['prediction.csv','mf_sum.csv','pre_cvgbt2.csv','pre_dt2.csv','pre_gbt2.csv','pre_lr2.csv','pre_rf2.csv']
```

```
P = [0.85269,0.78474,0.87356,0.66551,0.86500,0.79054,0.85755]
stx = []
for i in range(7):
    stx.append(120000*(2*P[i]-1))
sTx = np.array([stx]).T
s = []
for a in range(120000):
    s.append([])
```

```
count=0
for i in csvlist:
    file=pd.read_csv(i)
    l=file['Predictor'].tolist()
    for j in range(len(l)):
        s[j].append(l[j])
    count=count+1
S=np.array(s)
```

```
for i in range(len(S)):
    for j in range(7):
        S[i][j] = 2*S[i][j]-1
```

Calculate the $S^T x$ using the submissions before

Changed the binary set {0,1} in those submissions to {-1,1}.

Summary of Methods

Ensemble Modeling

```
tra = np.linalg.inv(S.transpose().dot(S))
```

Calculate the $(S^T S)^{-1}$

```
S_temp = np.dot(S, tra)
```

Calculate the $S(S^T S)^{-1}$

```
S_ensemble = np.dot(S_temp, sTx)
```

Calculate the $S(S^T S)^{-1} S^T x$

Thoughts and Conclusion

After using various machine learning algorithms, we get different results with diverse accuracy. Among these algorithms, Matrix Factorization merged with Cross Validator in Gradient-Boosted Tree gets the best result. In contrast, Matrix Factorization and Decision Tree get the worst one. But after all, the result gets boost up when we ensemble them together.

Through this project, we get more familiar with some machine learning algorithms and especially the Pyspark. It helps a lot when we need to handle datasets with large size.