Compensation Prediction

Team #13

Junyuan Bao
Boyu Chen
Yu Fang

Introduction

Firstly, we decided to predict compensation for accidents using spark machine learning tools and API.

Then, we built random forest model, gradient boosting tree model and multilayer perceptron model to solve this problem.

Also , by keep tuning hyper-parameters like number of layers and number of iterations, we get a fair good result on validation test and MAE.

Finally, we compared and evaluated these three different models and made an conclusion about them.

Exploratory Data Analysis

Using spark sql to identify missing data.

Using spark sql to identify data outliers.

Using spark sql to commpute basic data statistics like mean and stdev.

Visulizing data using zeppelin.

Sampling data with dataframe to visualize data points and understand the nature of our data.

Pre-processing dataset and filter invalid data.

Methodology

Using Gradient Boost Tree, Random Forests, multilayer perceptron and other supervised machine learning methods.

Using Spark MLlib decentralized machine learning framework.

Using Spark SQL for EDA.

Data Features

Number Of Data

```
data.count()

res11: Long = 188319
```

Loss Column Statistics

```
ds.select($"loss").describe().show()

+-------+------------------+
|summary|              loss|
+-------+------------------+
|  count|            188318|
|   mean| 0.4957170178808214|
| stddev|0.22248753959411222|
|    min|          0.179722|
|    max|          0.844848|
+-------+------------------+
```

General Methods

Train-Test Split:

We divide Dataset into two parts, one for training(70%), one for testing(30%).

```
val splits = data.randomSplit(Array(0.7, 0.3))
val (trainingData, validationData) = (splits(0), splits(1))
```

General Methods cont.

Grid Search:

In order to obtain the best model, we need to do this to select the best hyper-parameter combination during training.

```
algoNumTrees: Seq[Int] = Seq(20,30,40),
algoMaxDepth: Seq[Int] = Seq(4,6,8,10),
algoMaxBins: Seq[Int] = Seq(32,36,40),
numFolds: Int = 10,
```

```
val paramGrid = new ParamGridBuilder()
  .addGrid(algo.numTrees, params.algoNumTrees)
  .addGrid(algo.maxDepth, params.algoMaxDepth)
  .addGrid(algo.maxBins, params.algoMaxBins)
  .build()
```

General Methods cont.

K-fold Cross Validation:

Every time we select k-1 of K subsets of input data for training, remain 1 for validation.

```
numFolds: Int = 10,
```

```
val cv = new CrossValidator()
  .setEstimator(pipeline)
  .setEvaluator(new RegressionEvaluator)
  .setEstimatorParamMaps(paramGrid)
  .setNumFolds(params.numFolds)
```

Random Forest Solution

Advantages:

when number of features is high and give feature importance.

The speed of this Algorithm is faster among others and it can apply parallelization which correspond to the idea of Spark.

Disadvantage:

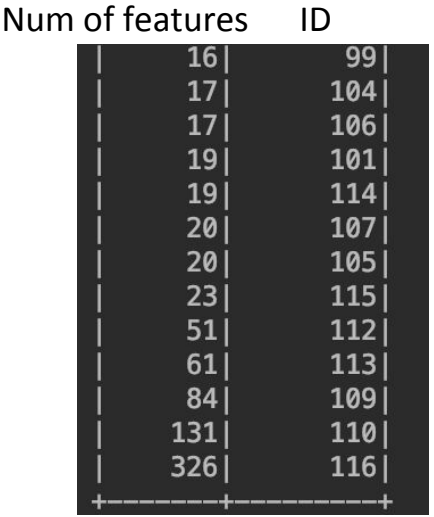
It performs not that good when some features have too many types (indexes)

Useful Methods When Apply Random Forest

Data re-format:

In order to apply random forest algorithm, we need to extract feature columns for training and remove columns that have too many feature types.

According to sorting result of features' type numbers, we will remove bottom five feature columns that has too many feature types.



Detail Features of specific column

```
val dsSample2 = ds.sample(true, .10)
dsSample2.groupBy("cat113").count.orderBy("cat113").show()
```

```
+------+-----+
|cat113|count|
+------+-----+
|     A|  267|
|    AA|  117|
|    AB|   31|
|    AC|   59|
|    AD|  151|
|    AE|   72|
|    AF|  934|
|    AG|  134|
|    AH| 1803|
|    AI|  442|
|    AJ|   10|
|    AK|  677|
|    AL|  119|
|    AM|  124|
```

```
val dsSample2 = ds.sample(true, .10)
dsSample2.groupBy("cat116").count.orderBy("cat116").show()
```

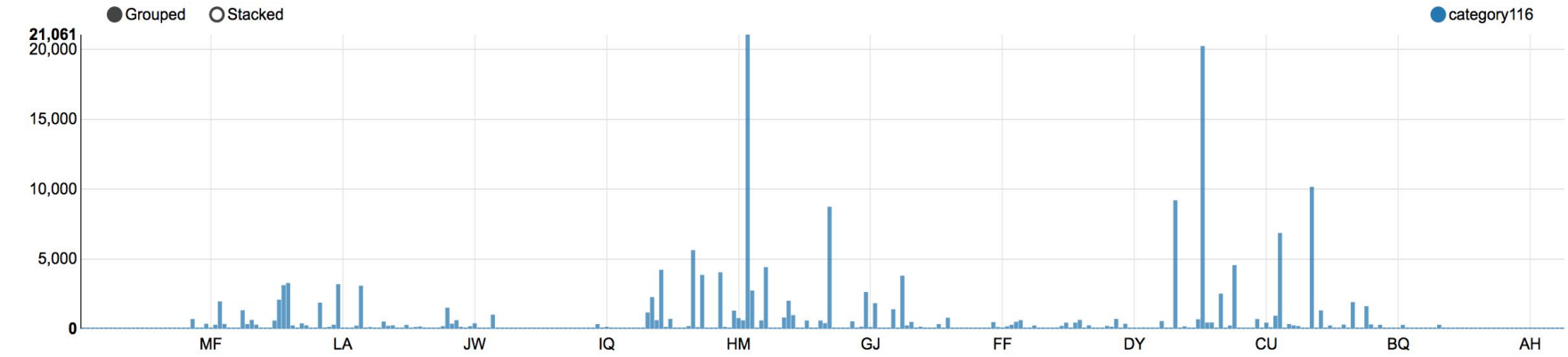
```
+------+-----+
|cat116|count|
+------+-----+
|     A|    5|
|     D|    1|
|     F|   21|
|     G|   31|
|     H|  266|
|     I|  722|
|     J| 2314|
|     K| 4397|
|     L| 1616|
|     M| 1199|
|     N| 2208|
|     O| 2722|
|     P| 2120|
|     Q|  790|
|     R|  204|
```

Showing one column with distribution of features

```
%sql
select cat116, count(*) as category116
from train
group by cat116
order by cat116 desc
```

FINISHED

settings



Random Forest Solution cont.

Building Pipline

StringIndexer: Change string type category columns into indieces.

VectorAssembler: Compose feature columns into one feature vector.

RandomForestRegressor: The Random forest model we want to train our data on.

```
// Building the Pipeline for transformations and predictor
val pipeline = new Pipeline().setStages((stringIndexerStages :+ assembler) :+ algo)
```

Random Forest Conclusion

MAE is 1324, not that far from the goal 1100, we may still change hyper-parameters.

```
================================================================
Param trainSample: 1.0
Param testSample: 1.0
TrainingData count: 131754
ValidationData count: 56564
TestData count: 125546
================================================================
Param algoNumTrees = 24
Param algoMaxDepth = 8
Param algoMaxBins = 32
Param numFolds = 10
================================================================
Training data MSE = 3932951.676671522
Training data RMSE = 1983.1670823890563
Training data R-squared = -0.12540229367001232
Training data MAE = 1301.6908525241693
Training data Explained variance = 8541394.45120469
================================================================
Validation data MSE = 4188007.057722932
Validation data RMSE = 2046.4620831383445
Validation data R-squared = -0.22977779577846014
Validation data MAE = 1324.4648041115904
Validation data Explained variance = 8182858.078965224
================================================================
```

Multi-Layer Perception Classifier

In addtion to regression, we also tried classification method as MLP.

MLP is a basic neural network model that has input layer, output layer and hidden layer, with activation function that transform model into non-linear form to simulate any non-linear process of data.

Multi-Layer Perception Classifier cont.

If we want to convert the problem into a classification problem, we have to change the target variable into category variable.

We can find a series of buckets and map target value into those buckets with appropriate bucket size.

So we apply QuantileDiscetizer to transform continous variable to category variable when building pipline.

```
val discretizer = new QuantileDiscretizer()
  .setInputCol("label")
  .setOutputCol("newlabel")
  .setNumBuckets(100)
  .fit(data.select(data.col( colName = "label")))
```

Multi-Layer Perception Classifier Result

Though we put 100 neurons in each hidden layer, we still achieve a bad result after training for a long time, maybe we should change bucket size or number of layers to do further testing.

```
==========================================================================
Param trainSample: 1.0
Param testSample: 1.0
TrainingData count: 132012
ValidationData count: 56306
TestData count: 125546
==========================================================================
Param trainlayers = [I@47396929
Param maxTrainBlockSize = 128
Param maxTrainIter = 20
Param numFolds = 10
==========================================================================
Training data MSE = 1.731381035872 49E7
Training data RMSE = 4160.986704944501
Training data R-squared = -13283.228736114621
Training data MAE = 2988.5563720722403
Training data Explained variance = 1.741732049040085E7
==========================================================================
Validation data MSE = 1.71055184337211 92E7
Validation data RMSE = 4135.881820569973
Validation data R-squared = -13098.452491904593
Validation data MAE = 2983.094138812916
Validation data Explained variance = 1.721006561930186 7E7
==========================================================================
```

"Gradient Boosting Tree" Intro

Gradient Boosting Tree(GBT) and Random Forest(RF) both are ensemble learning methods and predict (regression or classification) by combining the output from individual trees.

GBT build trees one at a time, where each new tree helps to correct errors made by previously trained tree.

"Gradient Boosting Tree" Data Form

training dataset : dropp off the category with too many type

```
cat25,cat26,cat27,cat28,cat29,cat30,cat31,cat32,cat33,cat34,cat35,cat36,cat37,cat38,cat39,cat40,cat41,ca
A,A,A,A,A,A,B,A,D,B,B,D,D,B,D,C,B,D,B,A,A,A,A,A,D,B,C,E,A,C,T,B,G,A,A,I,E,G,J,G,BU,BC,C,AS,S,A,O,LB,0.72
A,A,A,A,A,A,A,A,A,D,B,B,D,D,A,B,C,B,D,B,A,A,A,A,A,D,D,C,E,E,D,T,L,F,A,A,E,E,I,K,K,BI,CQ,A,AV,BM,A,O,DP,0.3
A,A,A,A,A,A,A,A,A,D,B,B,B,D,B,D,C,B,B,B,A,A,A,A,A,D,D,C,E,E,A,D,L,O,A,B,E,F,H,F,A,AB,DK,A,C,AF,A,I,GK,0.26
,A,A,A,A,B,A,A,A,D,B,B,D,D,D,B,C,B,D,B,A,A,A,A,A,D,D,C,E,E,D,T,I,D,A,A,E,E,I,K,K,BI,CS,C,N,AE,A,O,DJ,0.3
,A,A,A,B,A,A,A,A,D,B,D,B,D,B,B,C,B,B,C,A,A,A,B,H,D,B,D,E,E,A,P,F,J,A,A,D,E,K,G,B,H,C,C,Y,BM,A,K,CK,0.273
,A,A,A,B,A,A,A,A,D,B,D,B,D,B,B,C,B,B,B,A,A,A,A,A,D,D,D,E,C,A,P,J,D,A,A,E,E,H,F,B,BI,CS,A,AS,AE,A,K,DJ,0.
,A,A,A,A,A,A,A,A,D,B,B,D,D,B,D,C,B,B,B,A,A,A,A,A,D,D,D,E,C,A,P,J,A,A,C,E,E,H,F,B,BI,DK,A,J,AF,A,K,DJ,0.4
```

"Gradient Boosting Tree" Data Form

separate feature column & continuous column

```
val stringIndexerStages = trainingData.columns.filter(isCateg)
  .map(c => new StringIndexer()
    .setInputCol(c)
    .setOutputCol(categNewCol(c))
    .fit(trainInput.select(c).union(testInput.select(c))))
```

"Gradient Boosting Tree" Data Form

features column => vector (String Indexer)

```
// VectorAssembler for training features
val assembler = new VectorAssembler()
  .setInputCols(featureCols)
  .setOutputCol("features")
```

"Gradient Boosting Tree" Pipeline

Estimator Algorithum:

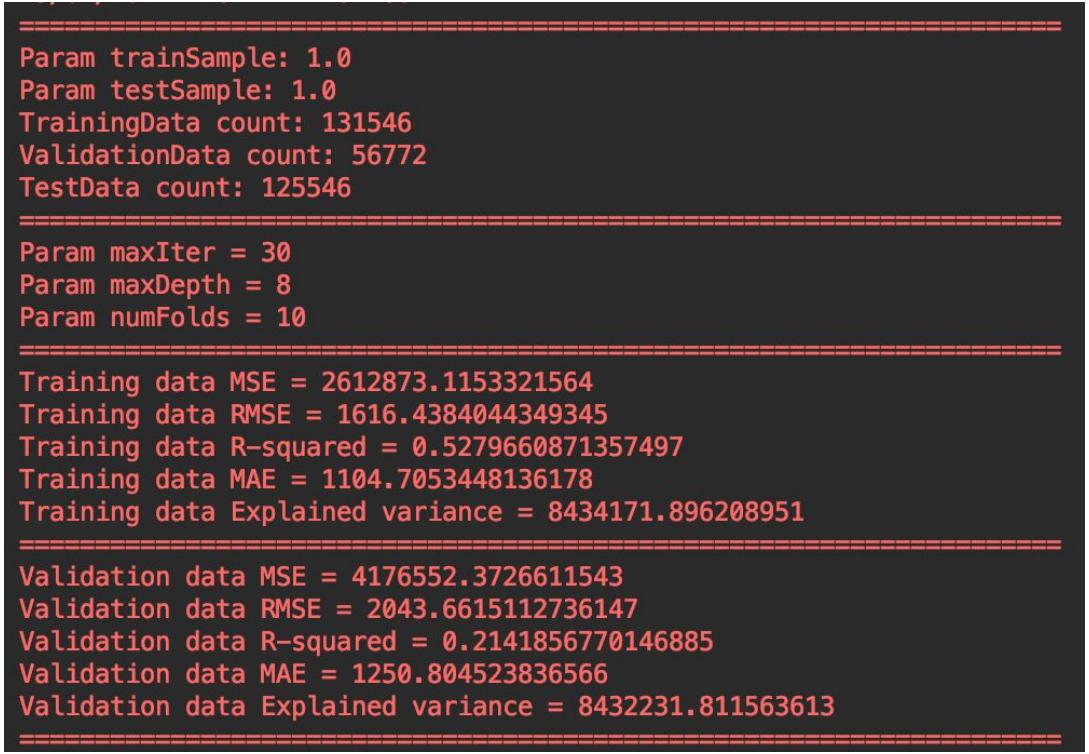
```
// Estimator algorithm
val algo = new GBTRegressor().setFeaturesCol("features").setLabelCol("label")
```

Set up the pipeline:

```
// Building the Pipeline for transformations and predictor
val pipeline = new Pipeline().setStages((stringIndexerStages :+ assembler) :+ algo)
```

"Gradient Boosting Tree" Conclusion

MAE is 1250, it achieve better performance compared to the Random Forest. But in the real environment. It took much longer time to complete the training.



Conclusion

Current achievement:

We got a current best MAE of 1290 which is much closer to the goal of 1100. We may adjust the hyper-parameters to obtain better result.

Pre-Plan Acceptance Criteria

Reach a "Mean Absolute Error(MAE)" over 1100. (Measured by Kaggle)

Our system will also show the performance gap between the "random forest regression" and "gradient tree regression" models.

Thank you