Siddharth Sharma

Kaggle Bluebook for Bulldozers Competition :

http://www.kaggle.com/c/bluebook-for-bulldozers

Introduction

The `Blue Book for Bulldozers' bidding sales prices prediction data set comprises of two main data sets, the Machine Appendix data and training data set. The Machine Appendix data set consisted mainly Machine related characteristics and details. The data in Machine Appendix is index with unique MachineID feature that uniquely identifies a Machine. There are 13 features in this data set that helps into recognizing the machine specific details like Model ID, Manufactured year, base model, secondary model, etc. Training data set provides details regarding bidding and machine intrinsic details. It comprises of 60 dimensions that includes bidding related information e.g. Sales Price, Sales ID, Auctioneer ID, Sales Date, State etc. and machine characteristics relevant to price of machine e.g. Machine Usage, Machine Hours Current Meter, Product Class Description, Product Group etc.

Data Analysis

In data analysis phase we discovered the following details about the provided data sets,

Missing Values :

Many features in merged train and machine appendix data consisted of majority of values as missing.

Usage Band, Product Size, Drive System, Forks, Pad_Type, Ride Control, Stick, Transmission, Turbocharged, Blade Width, Enclosure_Type, Pushblock, Ripper, Scarifier, Tip_Control, Tire_Size, Hydraulics_Flow, Track_Type, Undercarriage_Pad_Width, Stick_Length, Thumb, Pattern_Changer, Grouser_Type, Blade_Type, Travel_Controls, Differential_Type, Streeing_Controls

These features have missing values in range 50% to 75%. Even critical features like auctioneer id and model id had quite large proportion of values missing.

Handling Missing Values

To handle missing values we adopted 4 kinds of mechanisms:

a. **Randomly filling missing values** : e.g. to fill in missing values and inconsistent values in year made feature, we tried to take sample of data with same ModelId and then select a random value of the year made.

```
model<-train[i,3] #fetched model id for this particular year

year_model<-which(train[,3] == model) #these rows have the same model id
```

```
sample_year<-train[year_model,6] #getting year manufactured for same model trucks

year<-sample(sample_year[sample_ids],1)
```

b. **Average :** Another approach we tried to tackle missing values for numerical dimensions is to take mean of the existing values and replace missing values with them

```
Missing_val<-mean(data$colname,na.rm=TRUE).
```

c. **K Nearest Neighbors :** In this approach we tried to find a relevant sub data set of dimensions that are semantically related to the dimensions in question and tried to run 3 nearest neighbor on instances where the desired dimension value is missing. The missing value was filled by the average value of 3 nearest neighbors.

e.g. MachineHoursCurrentMeter's missing values can be found by using k nearest neighbor approach. Make a data frame with attributes sales price, modelid,sales year - year made,MachineHoursCurrentMeter for all values for which MachineHoursCurrentMeter is not missing. Then taking instances where MachineHoursCurrentMeter is missing and finding 3 nearest neighbors.

For this approach we would have to **normalize** dimensions using underlined approach

$$Zscore = (x - mean) / sd$$

d. **Replacing missing values with -1:** The three approaches defined above will create correlation in data since there are lot of dimensions for which majority values are missing. If we replace them with average or using other kind of average measures this will lead to statistical correlation and will reduce the performance of classification techniques like Decision Trees. To avoid this we can introduce a new class in dimensions that have missing value.

e. **Using AMELIA and Imputation R packages :** Due to the enormity of the data set, memory exceptions occurred.

Inconsistent Values: In features like Year Made we have nearly 38189 values for which there's an inconsistent value of 1000 present, which doesn't make any semantic sense. These values were replaced with -1, which represent a new class.

Unique Features: The training data consists of features like Sales ID which uniquely identifies each bidding instance. These features together with Machine ID were dropped.

Aggregated Features: Features like **fiModelDesc** and **fiProductClassDesc** were representing combined values of other features hence weren't providing any information gain for prediction models.

Types of Features : The data has following types of attributes

a. Numerical (SalesPrice, MachineHoursCurrentMeter etc.)
b. Ordered (Usage Band, Primary Lower/Upper)
c. Categorical (State, ModelID , ProductGroup)
d. Binary (Ride_Control, Sacrifier)

Feature Creation: The following features were created for information gain to regression algorithm

a. Age : This feature represents the age of the machine at the time of sale (SalesYear-MfgYear)
b. Sales Day, Sales Moth, Sales Year

Data Preparation

Binary Vectorization

Machine Learning models like SVM, Neural Networks, RandomRegressor (in Python) and even linear regression based approaches need input data as numerals. On the contrary majority of features in Train/Test data set are categorical attributes. Furthermore the R library for RandomForest accepts only 32 distinct categorical levels in a dimension and the data set contains many dimensions with more than even 100 levels. To counter this problem the approach of Binary Vectorization was taken.

model.matrix function in R adds a new dimension pertaining to each level in a feature and populates the instance with 0 and 1s. Features like ModelID, fiBaseModel have levels in thousands, so the likes of these dimensions were dropped since they will lead the dimension to explode.

After performing Binary Vectorization our **dimensions exploded from 52 to more than 800.**

Feature Mapping

Another technique adopted to convert factors to numeric value was feature mapping. Here the level index was mapped to each categorical value. In this way a unique number was provided to each factor in a dimension by **mapLevels** functionality of R. Though this approach has the drawback of providing higher numerical values to some factors and less to others though all factors should be weighted equally.

Converting to base 32

defactor32 is a R functionality that was used to bound the distinct numbers of factors in a feature to less than 32. This technique resulted in addition of dimensions to accommodate the remainder and divisor of each mapped factor (e.g. there were 50 states, one dimension was added to accommodate 50 mode 32 and other as divisor 32 or 1).

After applying Binary Vectorization the dimensions of the test/train data set exploded to more than 800. To reduce the dimensions **Principal Component Analysis** was applied.

PCA tries to discover components (dimensions) along which the projection of data vectors leads to maximum variance. We tried to get components that cover 90% of the variance in the data using R's

princomp function. **After applying PCA we got early 200 components that were able to cover 90% of the variance in data.**

Handling Big data

One of the persistent problems faced during the project was running machine learning regression models on *data set with instances more than four hundred thousand*. Models like random forest were throwing memory exceptions when ran on local machine. We took the following approaches to handle the problem of large scale machine learning

1. **Random Sampling**: In this approach a random sample of the data set with size equivalent to hundred thousand was taken from training data set and models like regression tree and random forest were trained on it.
2. **Temporal Sampling:** The training data consisted of bidding prices of Bulldozers that were time stamped. In this approach data instances belonging to **Sales Year > 2008** were taken as training data.
3. **Cross Validation:** The training data was sampled into 5 folds and in each iteration a regression tree was trained on one fold and tested on rest four.
4. **Using VCL images** with **16 GB** RAM

Modeling Approaches

1. Regression Trees

Since the data set consisted mainly of categorical values, our initial approach was mostly dedicated towards using **regression trees** to train the model. Since the data was time stamped and consisted of bidding information, we dropped data whose sales year was less than 2008. This reduced our training data set size to around 100,000.

5 fold cross validation was used to find the best modeling parameters. Since the data set was large and computation on 4 folds to train was time intensive, **the tree was trained on 1 fold and tested on 4 folds**.

grid = **expand.grid**(list(cp=c(0.01,.05,.1,.2),maxdepth=c(5,10,15,20,30)))

The expand grid was provided range of values of complexity parameters and max depth parameters. To train the tree we used rpart function and provide SalePrice (numerical value) to make it a regression tree.

cv.tree <- **rpart**(SalePrice~.,data=cv.train,control=rpart.control(cp=grid[j,1],maxdepth=grid[j,2]))

The root mean square log error of each tree was stored along with the tuning parameters of that tree and the best model was chosen to compute predictions on test data set. **The RMSLE error on test data was around .9.**

Drawbacks of Regression Tree : Regression tree uses two functions to predict value of the dependent variables 1. Recursive splits to group data in the leaf node 2. Use simple regression model to fit the local data at the leaf to minimize least squares error.

Since the bidding **data has temporal dimension**, though Regression tree can perform relatively good on similar kind of data i.e. belonging to same time period but it will not be able to project future values.

2. **Ensemble Regression tree:** This approach consisted of training *10 regression trees* on random sample of sizes 50,000 each. For test data the *average of these 10 values* was considered. The RMSLE was still more than .9 for this approach.

3. **GBM:** Gradient boosted Machine was used with *interaction.depth=10, shinkage=001,ntress=200. The RMSLE in this case was . 7* . This technique uses boosting as sampling technique. Opposed to bagging, boosting assigns data probabilistically to samples. While training the data instances which are classified wrong are selected more often in other samples so that they can be classified better in next run.

4. **Random Forest :** Random Forest was used on the following data sets

    a. **Data set with binary vectorization reduced by PCA:** In this data set we have binary attributes for all categorizes in a feature. Random forest threw **memory exception** in this case since dimensions were more than 200 and data instances were more than four hundred thousand.

    b. **100,000 random sampled data:** RMSLE was around .9346

    c. **Data set after with Sales Year >= 2008:** It nearly took more than 24 hours to train the random forest and the RMSLE on test data was around .91

5. **Using Random Forest with Neural network :**

    In this experiment we divided the training data set in two parts.

```
colnames1<-c("Drive_System"
 ,"ProductGroup" ,"state0" ,"state1","PrimarySizeBasis" ,"fiManufacturerID","Tire_Size","auctioneerID",
"UsageBand","ProductSize","Age","YearMade")

colnames2<-setdiff(colnames(data.train),colnames1)
```

colnames1 consist of all dimensions that have **global effect** on price of a Bulldozer i.e. no matter what type of machine it is these parameters will affect the Sales Price in similar manner., whereas **colname2** contains **features that are more intrinsic to the machine**. In this way features of training data are segregated into two different sets.

Two random forests were trained on these two data sets suing random sampling. The predictions of these random forests on the other fold of training data is used to train a neural network with two input nodes, three hidden nodes and one output node.

The input to neural network is normalized via using z score. The output will be in range 0-1 since Neural Network uses Sigmoid Function to scale output. The output of the neural network is rescaled as

Value= output * standard deviation + mean

The neural network is trained using feedback algorithm i.e. providing scaled value of actual sales price.

```
rf1 <- randomForest(SalePrice~.,data=data.train.s1,mtry=5,ntree=150)
rf2 <- randomForest(SalePrice~.,data=data.train.s2,mtry=5,ntree=150)

pred.values1 <- predict(rf1,newdata=data.test.s1)  # prediction value of first random forest
pred.values2 <- predict(rf2,newdata=data.test.s2)  # prediction value of second random forest

Y<-data.train$SalePrice[-rids.train]
df=data.frame("Y"=Y,"X1"=pred.values1,"X2"=pred.values2)

net <- neuralnet(Y  ~ X1 + X2,data=df, hidden=3,  threshold=0.01 ,linear.output=FALSE,
learningrate=0.01,algorithm="backprop",err.fct="sse",rep=100,stepmax=100000)
```

The RMSLE of this ensemble learning approach was around .9 which was equivalent to RMSLE retained by earlier models.

Analysis of models and results

1. The bidding data consists of temporal dimension and *none of the model trained did **time series analysis** on sales price*. Model like Random forest and decision trees performs well when tested on data with similar temporal features but don't perform well on data belonging on test data where sales year is in different range.

2. Due to *dominance of categorical features* we were **unable to use model like neural network or SVM that can model non linear functions** and can be used to exploit temporal aspects e.g. recurrent neural networks have been used to model time series analysis of housing prices and stock prices.

3. *Mapping Categorical values to numbers* (levels indexes) was semantically incorrect since it gives more importance to some categories and less to other. It even removes ordering among categories in ordinal values e.g. Machine Usage high, low etc

4. Binary Vectorization is even though a handy technique it *exploded the dimensions in test data set* which further resulted in **sparse data matrix** which is tough to train on machine learning models.

5. GBM's performance was relatively better than cross validation: GBM sampled data in such a way that it gives more importance to data instances that has least square error on predicted value or were classified wrong. In this way while doing ensemble learning, GBM resulted in better results than cross validation using rpart.