20 November 2018
21:24

XGBoost

XGBoost is a fast and efficient algorithm and has been used to by the winners of many data science competitions. It's a boosting algorithm and you may refer this article to know more about boosting. XGBoost works only with numeric variables and we have already replaced the categorical variables with numeric variables. There are many tuning parameters in XGBoost which can be broadly classified into General Parameters, Booster Parameters and Task Parameters.

- **General parameters** refer to which booster we are using to do boosting. The commonly used are tree or linear model

- **Booster parameters** depend on which booster you have chosen

- **Learning Task parameters** that decide on the learning scenario, for example, regression tasks may use different parameters with ranking tasks.

Let's have a look at the parameters that we are going to use in our model.

1. **eta**: It is also known as the learning rate or the shrinkage factor. It actually shrinks the feature weights to make the boosting process more conservative. The range is 0 to 1. Low eta value means the model is more robust to overfitting.

2. **gamma**: The range is 0 to ∞. Larger the gamma more conservative the algorithm is.

3. **max_depth**: We can specify maximum depth of a tree using this parameter.

4. **subsample**: It is the proportion of rows that the model will randomly select to grow trees.

5. **colsample_bytree**: It is the ratio of variables randomly chosen to build each tree in the model.

```
param_list = list(

        objective = "reg:linear",
        eta=0.01,
        gamma = 1,
        max_depth=6,
        subsample=0.8,
        colsample_bytree=0.5
        )
```

```
dtrain = xgb.DMatrix(data = as.matrix(train[,-c("Item_Identifier", "Item_Outlet_Sales")]), label= train$I
tem_Outlet_Sales)
dtest = xgb.DMatrix(data = as.matrix(test[,-c("Item_Identifier")]))
```

Cross Validation

We are going to use the xgb.cv() function for cross validation. This function comes with the xgboost package itself. Here we are using cross validation for finding the optimal value of nrounds.

```
set.seed(112)
xgbcv = xgb.cv(params = param_list,
               data = dtrain,
               nrounds = 1000,
               nfold = 5,
               print_every_n = 10,
               early_stopping_rounds = 30,
               maximize = F)
```

```
[1] train-rmse:2746.725586+7.336395 test-rmse:2746.741797+30.998116
Multiple eval metrics are present. Will use test_rmse for early stopping.
Will train until test_rmse hasn't improved in 30 rounds.

[11]    train-rmse:2537.260938+5.858499 test-rmse:2539.031055+31.050055
[21]    train-rmse:2349.231885+4.482735 test-rmse:2352.873389+32.255671
[31]    train-rmse:2182.393750+4.819384 test-rmse:2187.824902+31.588163
[41]    train-rmse:2033.432080+3.195964 test-rmse:2040.415454+31.588752
[51]    train-rmse:1900.760058+2.740688 test-rmse:1910.174609+30.442020
[61]    train-rmse:1784.256690+2.877070 test-rmse:1796.284033+28.750707
[71]    train-rmse:1680.479932+2.497050 test-rmse:1695.349048+28.331918
[81]    train-rmse:1590.351367+2.354304 test-rmse:1607.833301+27.908130
[91]    train-rmse:1510.894165+2.035948 test-rmse:1531.593726+27.522115
[101]   train-rmse:1441.319312+1.659380 test-rmse:1465.053589+29.087921
[111]   train-rmse:1381.578052+2.426386 test-rmse:1408.426611+27.691651
[121]   train-rmse:1329.565503+2.454451 test-rmse:1359.327344+27.161061
[131]   train-rmse:1283.176050+2.888625 test-rmse:1316.178931+26.197404
[141]   train-rmse:1243.624048+2.574635 test-rmse:1280.237305+25.250655
[151]   train-rmse:1210.155518+2.584130 test-rmse:1250.134522+25.117012
[161]   train-rmse:1180.639380+2.815261 test-rmse:1224.006177+24.242971
[171]   train-rmse:1155.042627+2.457568 test-rmse:1201.782520+23.791665
[181]   train-rmse:1133.263232+2.564969 test-rmse:1183.207934+22.977824
[191]   train-rmse:1114.468555+2.585744 test-rmse:1167.656421+22.294357
[201]   train-rmse:1098.238403+3.042034 test-rmse:1154.633814+21.729746
[211]   train-rmse:1083.767114+3.474945 test-rmse:1143.161670+21.285152
[221]   train-rmse:1071.447070+3.508526 test-rmse:1134.116235+20.871062
```

```
[401]   train-rmse:976.771387+2.411750   test-rmse:1089.900439+18.755224
[411]   train-rmse:973.951404+2.436468   test-rmse:1089.861890+18.818395
[421]   train-rmse:971.220996+2.519783   test-rmse:1089.869263+18.860485
[431]   train-rmse:968.606470+2.528438   test-rmse:1089.797754+18.815133
[441]   train-rmse:966.202258+2.767049   test-rmse:1089.844604+18.789729
[451]   train-rmse:963.874219+2.858829   test-rmse:1089.895117+18.779128
Stopping. Best iteration:
[430]   train-rmse:968.889343+2.497103   test-rmse:1089.786206+18.836296
```

Model Training

As per the verbose above, we got the best validation/test score at the 430th iteration. Hence, we will use nrounds = 430 for building the XGBoost model.

Model Training

As per the verbose above, we got the best validation/test score at the 430th iteration. Hence, we will use nrounds = 430 for building the XGBoost model.

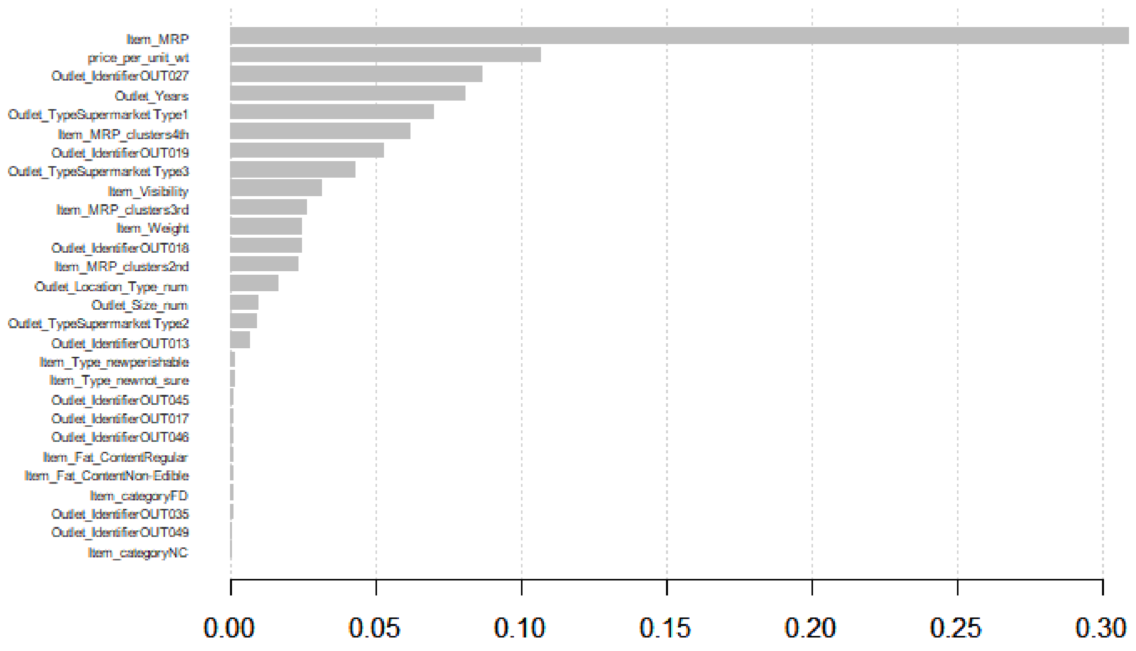
```
xgb_model = xgb.train(data = dtrain, params = param_list, nrounds = 430)
```

Leaderboard score: 1154.70

This model has even outperformed the RandomForest model.

Variable Importance

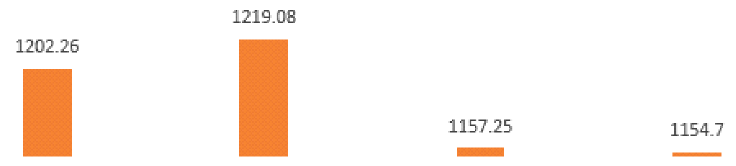
```
var_imp = xgb.importance(feature_names = setdiff(names(train), c("Item_Identifier", "Item_Outlet_Sale
s")),
                         model = xgb_model)
xgb.plot.importance(var_imp)
```



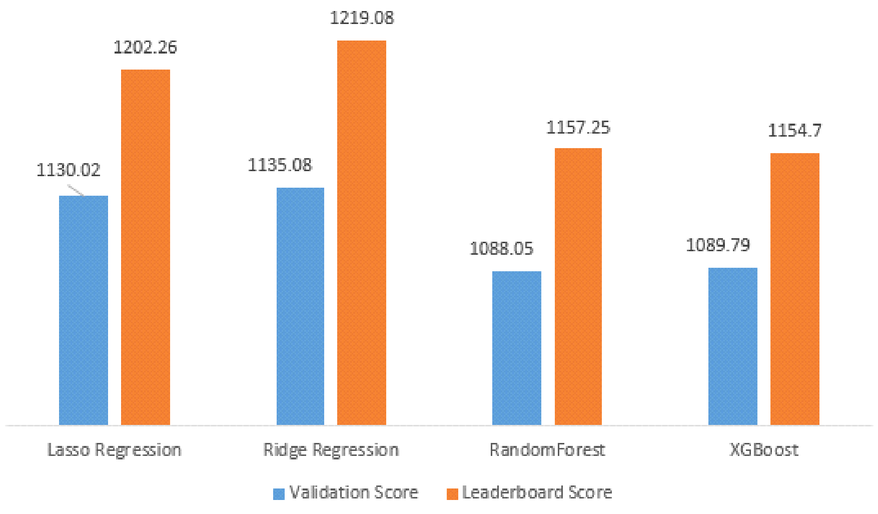
Again the features created by us, like price_per_unit_wt, Outlet_Years, Item_MRP_Clusters, are among the top most important variables.

Summary

After trying and testing 5 different algorithms, the best score on the public leaderboard has been achieved by XGBoost (1154.70), followed by RandomForest (1157.25). However, there are still a plenty of things that we can try to further improve our predictions.



After trying and testing 5 different algorithms, the best score on the public leaderboard has been achieved by XGBoost (1154.70), followed by RandomForest (1157.25). However, there are still a plenty of things that we can try to further improve our predictions.



What else can be tried…

There are still quite a many things that can be tried to improve our models' predictions. We can create and add more variables, try different models with different subset of features, etc. Some of the ideas based on our EDA are listed below:

1. We can transform the target variable to reduce its skewness.
2. We can also make independent vs independent variable visualizations to discover some more patterns.
3. We found out four groups of Item_MRP in EDA. We can create a separate model for each of these groups and then combine the predictions.
4. Based on the violin plots, we can group the categories of the categorical variables which are quite similar in shape and size.
5. We can train the XGBoost model using grid search as done in the case of RandomForest.
6. We can even use different algorithms such as LightGBM or neural network based models.
7. Model ensembling can also be of great help in improving the model's performance.