10/21/2018
University of Illinois at Urbana-Champaign
IE598 - Machine Learning in Finance

IE598 - MLF FINAL PROJECT

FALL 2018

by Joseph Loss, Yuchen Duan, Ruozhong Yang, Fengkai Xu, and Biao Feng

TABLE OF CONTENTS

Contents

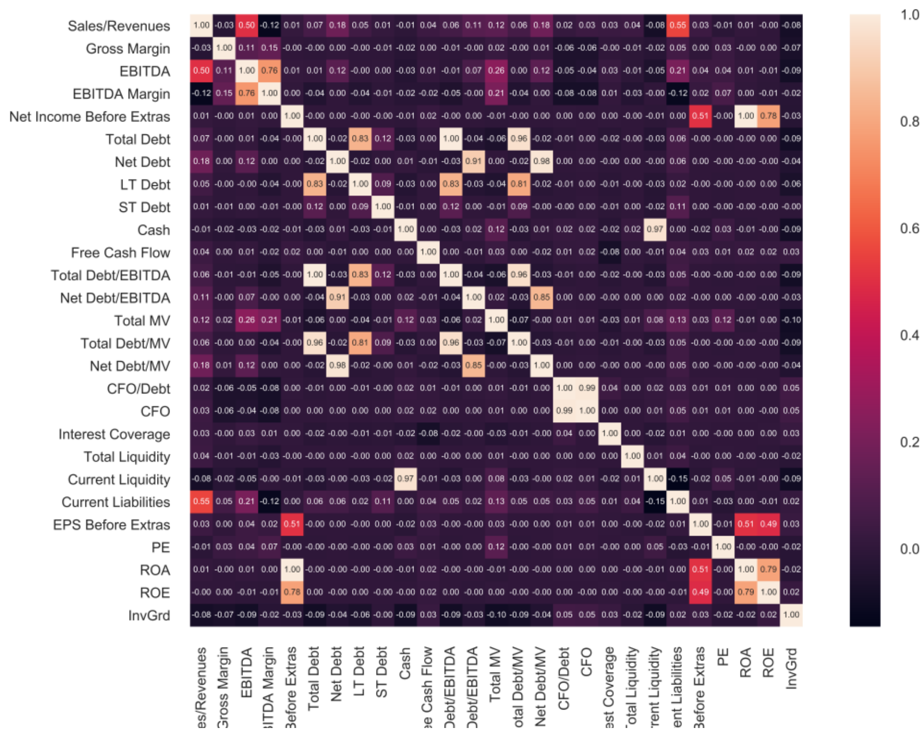
Chapter 1: Moody's Bond Rating Classifier

EXPLORATORY DATA ANALYSIS

Here is what our data looks like:

```
RangeIndex: 1700 entries, 0 to 1699
Data columns (total 29 columns):
Sales/Revenues              1700 non-null float64
Gross Margin                1700 non-null float64
EBITDA                      1700 non-null float64
EBITDA Margin               1700 non-null float64
Net Income Before Extras    1700 non-null float64
Total Debt                  1700 non-null float64
Net Debt                    1700 non-null float64
LT Debt                     1700 non-null float64
ST Debt                     1700 non-null float64
Cash                        1700 non-null float64
Free Cash Flow              1700 non-null float64
Total Debt/EBITDA           1700 non-null float64
Net Debt/EBITDA             1700 non-null float64
Total MV                    1700 non-null float64
Total Debt/MV               1700 non-null float64
Net Debt/MV                 1700 non-null float64
CFO/Debt                    1700 non-null float64
CFO                         1700 non-null float64
Interest Coverage           1700 non-null float64
Total Liquidity             1700 non-null float64
Current Liquidity           1700 non-null float64
Current Liabilities         1700 non-null float64
EPS Before Extras           1700 non-null float64
PE                          1700 non-null float64
ROA                         1700 non-null float64
ROE                         1700 non-null float64
InvGrd                      1700 non-null int64
Rating                      1700 non-null object
Class                       1700 non-null int64
dtypes: float64(26), int64(2), object(1)
memory usage: 385.2+ KB
```

Also, we have a correlation matrix:

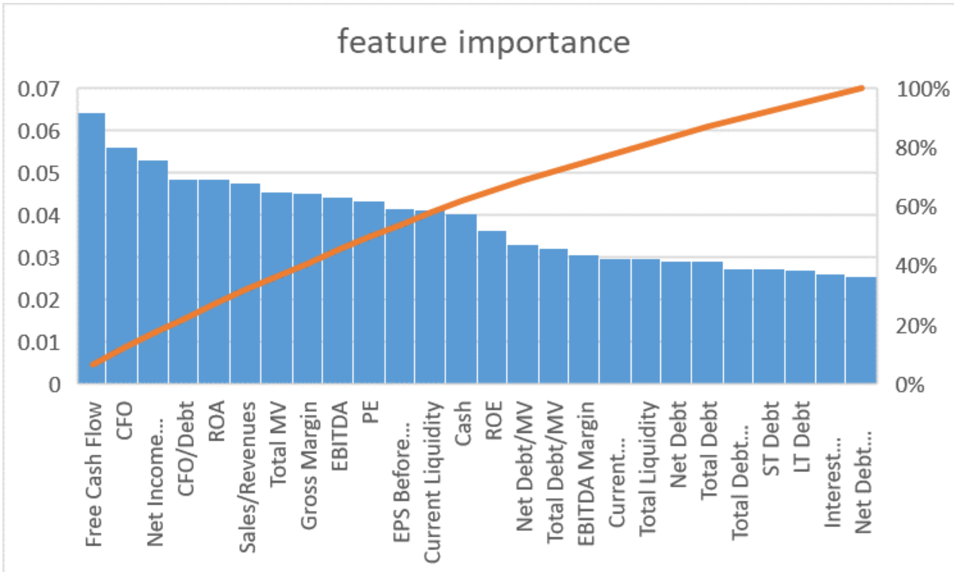


PREPROCESSING & FEATURE EXTRACTION/SELECTION

The preprocessing part combine some steps that need to be done before we try to fit our model:

1. Split the test and train database via train_test_split (with test_size = 0.1 and random_state = 42)
2. Standardize features via StandardScaler for better model performance.

We also calculate the importance of each feature and select 13 of them for our models.



MODEL FITTING & EVALUATION (BINARY & MULTICLASS)

1. Model 1

The first model is the KNN model.

2. Model 2

The second model is the Random Forest model.

3. Model 3

The third model is the Decision Tree model.

4. Model 4

The forth model is the Logistic Regression model.

We will discuss those models in the hyperparameter tuning and ensemble parts.

HYPERPARAMETER TUNING

We deal with different parameters via GridSearchCV function, the range of each model's parameter is from 1 to 100. Here is the best result for each model:

binary		muticlasses		percents
KNN	0.8	KNN	0.458823529	0.573529
RandomForest	0.858823529	RandomForest	0.676470588	0.787671
Decision tree	0.794117647	Decision tree	0.447058824	0.562963
Logistic Regression	0.741176471	Logistic Regression	0.247058824	0.333333

From the table, it is easy to find the multi-classes task lead to poor prediction (multi_lr score is about 1/3 compare to the binary one). There are several improvements can be done for better models, we will discuss them at the conclusion.

ENSEMBLING

Our team used the ensemble method for binary classification (chosen method does not support multi-class classification). Result showed below:

binary		
ROC AUC:	0.73(+/-0.05)	[KNN]
ROC AUC:	0.9(+/-0.02)	[RandomForest]
ROC AUC:	0.75(+/-0.05)	[Decision tree]
ROC AUC:	0.89(+/-0.02)	[Majority voting]

CONCLUSIONS

The best result for binary model is 0.89 (after ensembling) and the best for multiclass is 0.67. There are several things we can do to improve our model:

1. Dimension reduction

We can reduce the dimension of our model for better prediction, but in doing so, we must be careful that we don't accidentally remove important information in doing so.

2. Internal relationships

Some features are highly correlated, we can find them and just use one of them. Besides, many features have internal relationships, thus, some of them may actually talk about the same thing.

3. Weight adjustment

Although those models adjust weights of each feature automatically, people from accounting major may hold different view of those weights.

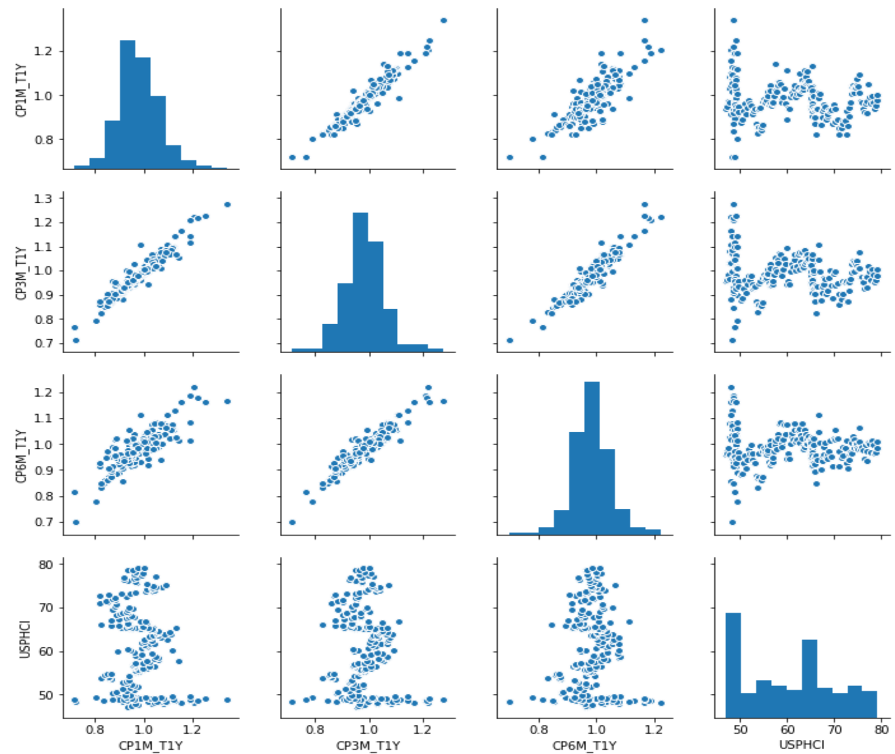
Chapter 2: USPHCI Economic Activity Forecast

EXPLORATORY DATA ANALYSIS

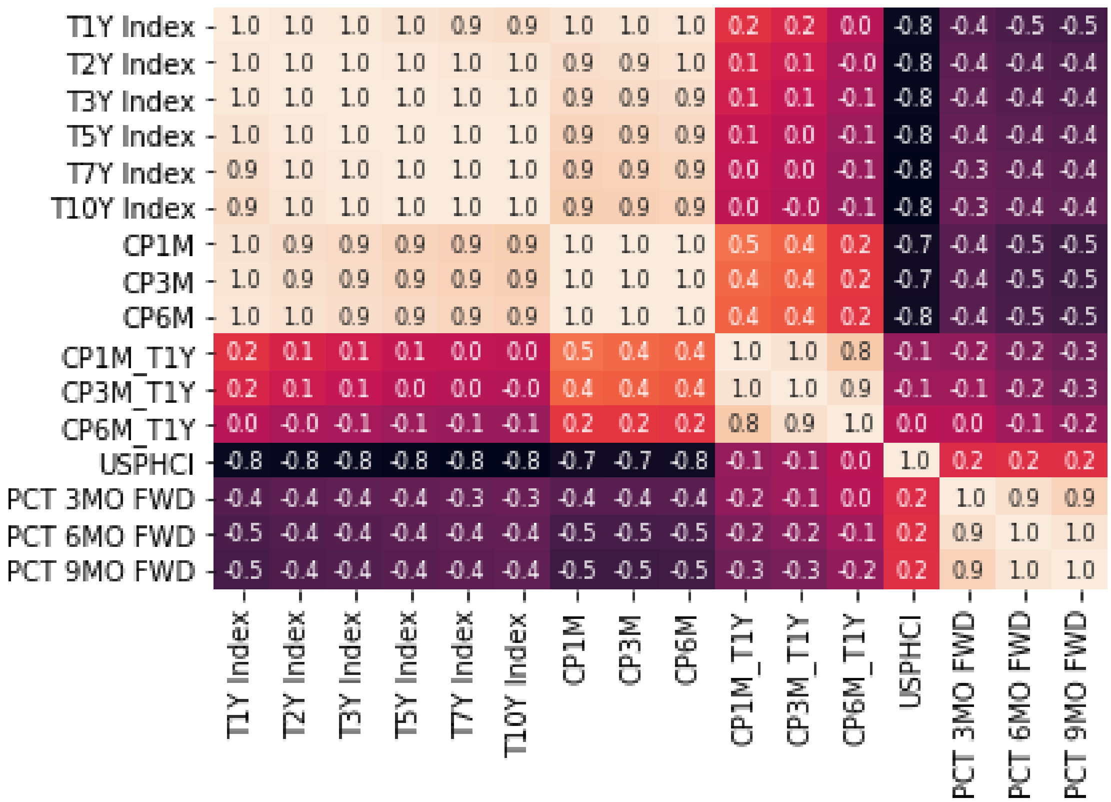
Our dataframe is composed of:

```
Int64Index: 223 entries, 0 to 222
Data columns (total 16 columns):
T1Y Index      223 non-null float64
T2Y Index      223 non-null float64
T3Y Index      223 non-null float64
T5Y Index      223 non-null float64
T7Y Index      223 non-null float64
T10Y Index     223 non-null float64
CP1M           223 non-null float64
CP3M           223 non-null float64
CP6M           223 non-null float64
CP1M_T1Y       223 non-null float64
CP3M_T1Y       223 non-null float64
CP6M_T1Y       223 non-null float64
USPHCI         223 non-null float64
PCT 3MO FWD    223 non-null float64
PCT 6MO FWD    223 non-null float64
PCT 9MO FWD    223 non-null float64
dtypes: float64(16)
memory usage: 29.6 KB
```

Figure 1: Plotting variables to get a feel for the relationships of the dataset

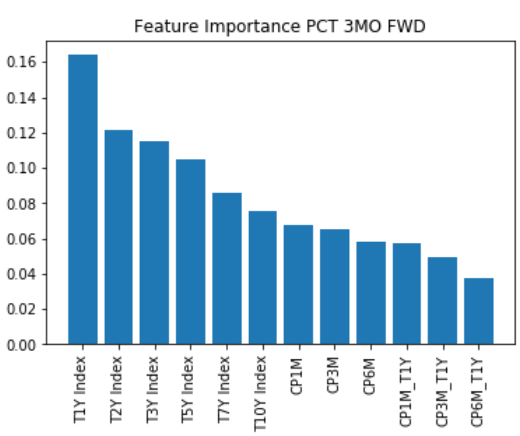


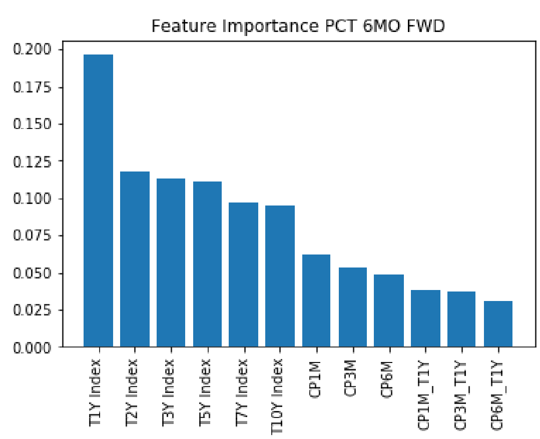
Also, we plotted a matrix to see the relationships for the entire dataset:

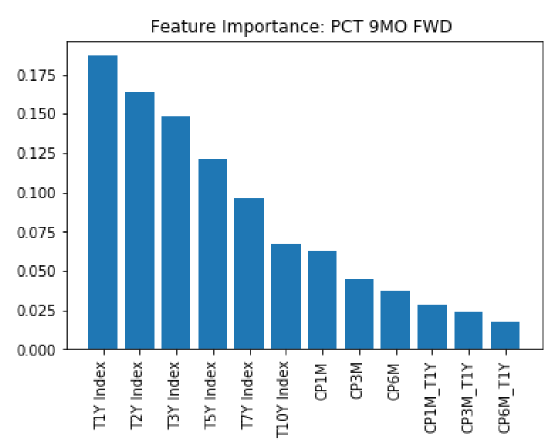


PREPROCESSING & FEATURE EXTRACTION/SELECTION

We can see the importance of each feature in relation to the 3MO, 6Mo, and 9MO Forward Rate:







```
3MO FWD RATE - Feature Importance
 1) T1Y Index                      0.163890
 2) CP1M_T1Y                       0.121097
 3) T10Y Index                     0.114853
 4) T3Y Index                      0.104408
 5) T2Y Index                      0.085722
 6) CP1M                           0.075296
 7) CP3M                           0.067459
 8) CP6M_T1Y                       0.065129
 9) T5Y Index                      0.057837
10) T7Y Index                      0.057455
11) CP6M                           0.049308
12) CP3M_T1Y                       0.037545

6MO FWD RATE - Feature Importance
 1) T1Y Index                      0.195985
 2) CP1M                           0.117591
 3) CP3M                           0.112635
 4) T10Y Index                     0.110856
 5) CP1M_T1Y                       0.096567
 6) CP6M                           0.095394
 7) T3Y Index                      0.061621
 8) T5Y Index                      0.053676
 9) T7Y Index                      0.048926
10) T2Y Index                      0.038705
11) CP6M_T1Y                       0.037129
12) CP3M_T1Y                       0.030916

9MO FWD RATE - Feature Importance
 1) CP1M                           0.186953
 2) CP3M                           0.164119
 3) CP6M                           0.148786
 4) T10Y Index                     0.121164
 5) T1Y Index                      0.095936
 6) CP1M_T1Y                       0.067408
 7) T7Y Index                      0.063060
 8) T5Y Index                      0.044385
 9) T3Y Index                      0.037779
10) CP6M_T1Y                       0.028688
11) CP3M_T1Y                       0.024268
12) T2Y Index                      0.017453
```

MODEL FITTING & EVALUATION

1. Model 1

We use Linear Regression for 3-month prediction.

2. Model 2

We use Ridge Regression for 6-month prediction.

3. Model 3

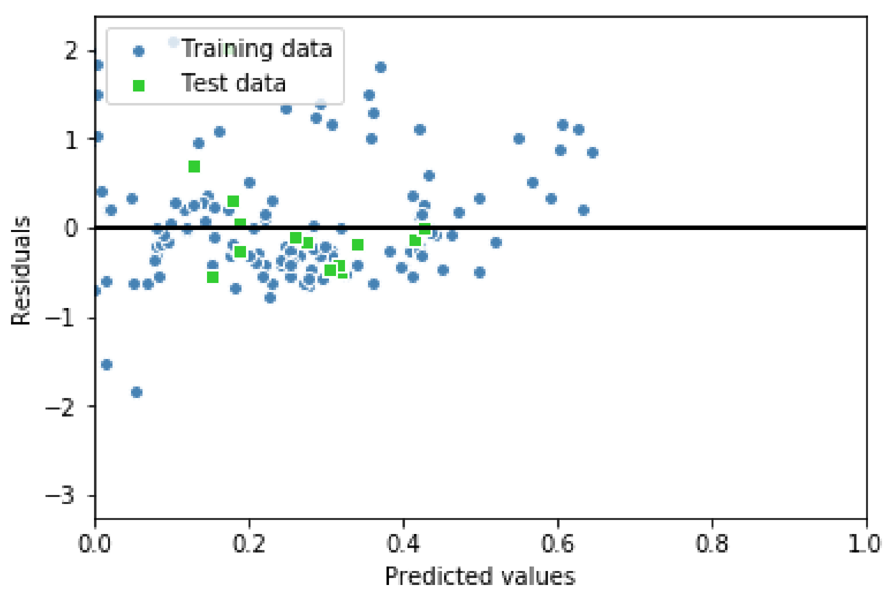
We use Lasso Regression for 9-month prediction.

We will go into detail about the performance of each model with their predictions in the following chapter on HyperParameter Tuning & Evaluating on the Test Set.

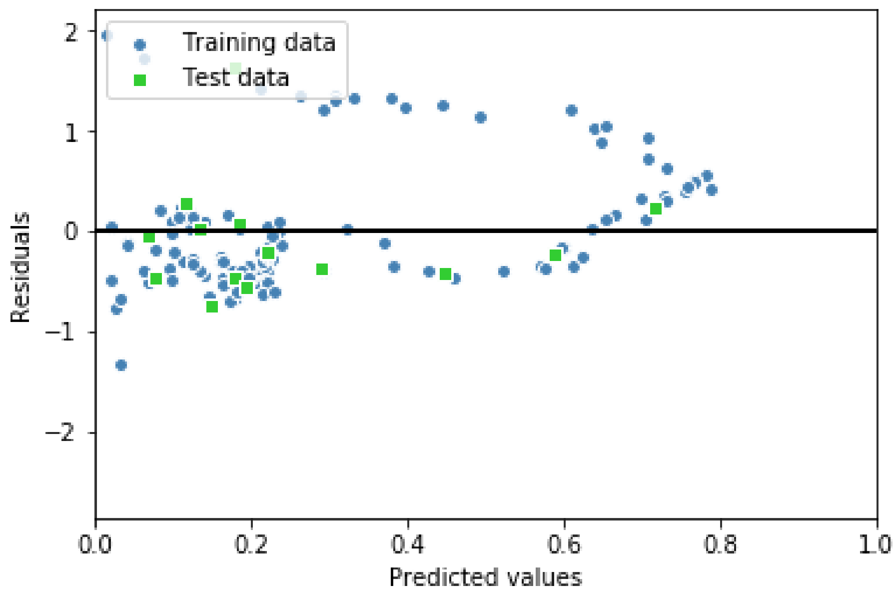
HYPERPARAMETER TUNING

In the first case (linear regression), we cannot change the parameter, in the second and third cases, we change the alpha(ridge from 10^-3 to 10^0, lasso from 10^-6 to 10^-3).We only show those images for the best model of each case and show the rest of them in a table.

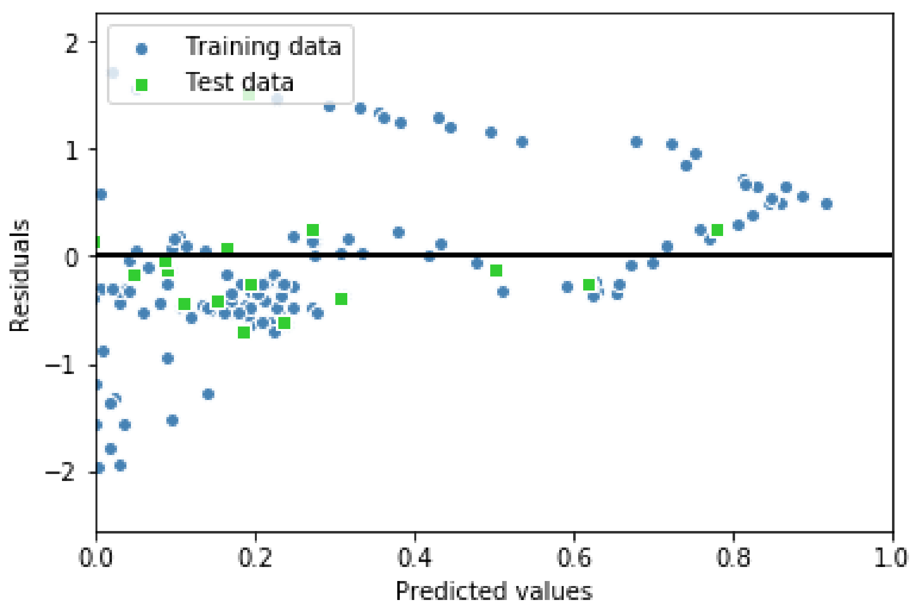
Linear Regression:



Ridge Regression:(Ridgealpha: 0.010)



Lasso Regression:(Lassoalpha: 0.000100)



The following table contains the performance metrics for each model:

ridge						
alpha	MSE train	MSE test	R^2 Train	R^2 test	Slope	Intercept
0.001	0.769	0.477	0.248	0.398	-1.022	-0.018
0.01	0.774	0.471	0.243	0.405	-0.599	-0.017
0.1	0.788	0.496	0.229	0.374	-0.202	-0.016
1	0.808	0.543	0.209	0.314	-0.087	-0.015
Lasso						
alpha	MSE train	MSE test	R^2 Train	R^2 test	Slope	Intercept
0.000001	0.715	0.416	0.296	0.509	-0.825	-0.014
0.00001	0.715	0.416	0.296	0.509	-0.816	-0.014
0.0001	0.716	0.414	0.295	0.512	-0.714	-0.014
0.001	0.726	0.425	0.286	0.499	-0.264	-0.013
Linear						
MSE train	MSE test	R^2 Train	R^2 test	Slope	Intercept	
0.823	0.619	0.194	0.239	-3.219	-0.02	

ENSEMBLING

We utilized a Gradient Boosting Regressor for our Ensemble Learning Methodology. After some parameter tweaking, we found that the Boosting algorithm vastly outperformed all previous models. In stark contrast to the table above, the ensembled GBR model reported a mean-squared-error of 0.31, and impressively scored an R-Squared value of 0.64 (nearly doubling the performance of the Linear and Ridge Regression models)

CONCLUSIONS

In conclusion, we found that out of the three original models, Lasso performed the best. This is likely attributed to the fact that LASSO can reduce some (if not all) of the coefficients of the model to zero, depending upon its regularization parameter, lambda. Ridge regression can only penalize the coefficient sizes; it will not remove them from the model.

Our final model, the ensemble learning methodology in which we used Gradient Boosting Regression, clearly performed the greatest. This is due to key concept of boosting, which is to focus on the training samples that are harder to classify and learn from misclassified training samples, thus teaching itself through trial-and-error to improve the ensemble performance.

Appendix

GITHUB REPOSITORY

IE598_F18_MLF_GROUP_PROJECT (LINK)