IE598 Group Project

Carolina Carvalho Manhães Leite
Xuehui Chao
Khavya Chandrasekaran

CHAPTER 1
Credit Score Problem

1) Introduction / Exploratory Data Analysis

In the first chapter we're going to analyze a Credit Score Problem. Our dataset is called "MLF_GP1_CreditScore", and originally has 1700 observations and 26 financial/accounting metrics for a set of firms in different industries. Our main goal here is to use the explanatory variables to predict:

- Model 1: Whether the company is considered "Investment Grade" or not;
- Model 2: The Moody's rating for each company.

From the variables listed above we can already assume these are classification problems (since both have a finite range for their target variables). The only difference between them is that Model 1 is a binary classification and Model 2 is a multiclass one (still a classification, though).

The first step before actually building those models would be to get to know the database and perform some exploratory data analysis. As we already said, the complete dataset has 1,700 rows/observations and 28 columns (26 features + 2 targets: "InvGrd" and "Rating"). We calculated the descriptive statistics for each one of the variables and also plotted some scatterplot matrices – we won't display all of the results here for the sake of clarity, but some aspects have caught our attention:

Table 1. Descriptive Statistics for the variable "ST Debt".

Count	1,700
Mean	3.14
Std	51.98
Min	-1
25%	-0.34
50%	0.04
75%	0.65
max	2,038

The variable "ST Debt" (Short Term Debt) has mean of 3.14 and a median of 0.04 (note that the minimum is -1). Still, its maximum value is 2,038. This may suggest the presence of outliers in our sample. But there's the problem of "absolute" features: this variable is purely the Short Debt, and not a ratio involving this feature. Let's take a look at another example:

Table 2. Descriptive Statistics for the variable "Total Debt/MV".

Count	1,700
Mean	1.27
Std	22.79
Min	-0.94
25%	-0.21
50%	-0.02
75%	0.24
max	676.44

Here we have the ratio "Total Debt/MV", which represents the Total Debt of the company divided by its Market Value. Even though it's a ratio we still can detect some possible outliers (the maximum value is 676.44, while the median is -0.02 and the minimum value is -0.94).

We'll use some techniques to delete those who we identify as outliers. One useful metric is the Interquartile range (IQR), a measure of statistical dispersion equal to the difference between the 75th and 25th percentiles. We calculated the 25th (Q1) and 75th (Q3) percentiles for each variable, computed their IQR and applied a rule to identify outliers: values that are lower than (Q1 – 1.5IQR) or greater than (Q3 + 1.5IQR) were considered outliers and consequently removed from our sample. This led to a drop in the number of observations from 1700 to 468, which is significant. More than this, by performing some descriptive statistical analysis on the binary target variable after the removal of outliers, we see that we deleted all of the "non-investment grade" cases:

Table 3. Descriptive Statistics for the variable "InvGrd" after deleting the outliers.

Count	468
Mean	1
Std	0
Min	1
25%	1
50%	1
75%	1
max	1

We've just removed the information that would help us explain/predict the cases that are not investment grade! Still another argument against the elimination of those records: 468 observations may be enough for us to build a binary classification model, but perhaps it's too low of a number to construct a multiclass model. We'd potentially disappear with some classes, or end up with classes with very few observations, not statistically significant. After considering all of those reasons, we decided not to remove the outliers.

One last analysis was to compute the frequency of each value in both the binary and the multitarget targets:

Table 4. Frequency of the values for the binary target ("InvGrd")

0 (Non-investment grade)	413
1 (Investment grade)	1287

Table 5. Frequency of the values for the multiclass target ("Rating")

Aaa	2
Aa2	65
Aa3	174
A1	122
A2	156
A3	31
Baa1	179
Baa2	326
Baa3	232
Ba1	17
Ba2	125
Ba3	108
B1	69
B2	48
B3	37
Caa1	9

From **Table 5**, we see that we have too many different possibilities for the multiclass target. Even though is possible to build multiclass classification models, their performance decreases with the number of classes. We then aggregated some of this classes, resulting in a new multiclass target variable, "Rating2":

Table 6. Frequency of the values for the aggregated multiclass target ("Rating2").

1 (AAA/AA)	241
2 (A)	309
3 (BAA)	737
4 (BA)	250
5 (B/CAA)	163

One interesting observation about the clustering above: we didn't create one specific category for the AAA's because we only had 2 of its observations, the same happened for the Caa1 (with 9 observations). We then mixed them with the next/previous categories.

2) Preprocessing / Feature Extraction / Feature Selection

The only preprocessing method we applied here was the standardization of the features; we already saw in the previous section why we're not deleting the outliers. Besides, we didn't perform any more work on the existing variables because:

- We don't have as many variables that would make the problem unfeasible (only 26);
- Since we're also building a multiclass classification model, the process of selecting variables could be potentially dangerous (we could be eliminating the only variable that explains/isolates one particular class of the target);
- We already have a good representation of variables in our features' space: from "absolute" values (EBITDA, Short Term Debt) to ratios involving those variables;
- PCA will already be covered in the 2nd chapter of this report.

We then used "train_test_split" to split the dataset in the training and testing samples, using 15% as our test size.

3) Model Fitting and Evaluation

As a first attempt, we tried different classifiers (with their default parameters) to get a baseline idea of the performance of the models and identify which one of them seems more "promising". In this step we used a K-Fold cross validation method for every technique, with a fixed value of 10 folds. In order to decide which model's parameters are going to be optimized, we used both the accuracy score and the Matthews correlation coefficient to compare the alternatives.

We understood that the accuracy alone would not be sufficient to indicate whether a model is performing or not (we can have a high accuracy and yet only predict well the target with more entries in the dataset). The Matthews coefficient is a great way to balance recall and precision for both the positive and the negative values (thing that the F1 score, another metric usually applied in machine learning, doesn't take into consideration). The Matthews coefficient (MCC) can be directly calculated from the confusion matrix; it ranges from -1 to 1, with 1 representing a perfect prediction and 0 a prediction that's no better than a random one.

$$MCC = \frac{TP \; x \; TN - FP \; x \; FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

I. BINARY CLASSIFICATION

In the first step we tried the following classifiers (with their default parameters):

- Logistic Regression with L2 penalization;
- Decision Tree Classifier;
- KNN;
- SVC.

The results for the 10 K-Fold cross-validation method are compiled in the following table:

Table 7. Results of different binary classifiers.

Method	Mean acc. 10-Fold	Std Dev acc. 10-Fold	MCC
Logistic Regression	0.753	0.032	0.156
Decision Tree	0.803	0.053	0.389
KNN	0.793	0.048	0.408
SVC	0.769	0.03	0.183

Taking both metrics into consideration (mean accuracy and MCC), we see that the KNN model is the best one (even without being optimized); its mean accuracy may be slightly lower than the Decision Tree, but when we look at the MCC, **we conclude that the KNN has a better balance between precision and recall.**

II. MULTICLASS CLASSIFICATION

In the first step we tried the following classifiers (with their default parameters). Besides the trees (that can more easily handle multiclass targets, after all the ultimate idea is to "isolate them" in pure leaves), other types of techniques (as the logistic regression) may request some adaptation. In those cases, we used the OvR approach ("one versus rest"), in which we split the data set into "target A" x "target non-A" (and model this as a binary classification problem), then "target B" x "target non-B"(with its respective binary classification model), so on and so forth.

- Logistic Regression;
- Decision Tree Classifier;
- KNN;
- SVC.

The results for the 10 K-Fold cross-validation method are compiled in the following table:

Table 8. Results of different binary classifiers.

Method	Mean acc. 10-Fold	Std Dev acc. 10-Fold	MCC
Logistic Regression	0.436	0.044	0.033
Decision Tree	0.528	0.095	0.383
KNN	0.474	0.051	0.300
SVC	0.443	0.026	0.151

Some interesting things are worth mentioning here: confirming our thesis that a multiclass problem is more difficult to be addressed (more than one model in some cases, fewer observations per class etc), the Logistic Regression didn't converge with the default number of iterations (100). We tried with 1,000, still without success, only to find that we'd only achieve convergence with 10,000 iterations.

Taking both metrics into consideration (mean accuracy and MCC), we see **that the Decision Tree is the best option (even without being optimized).**

4) Hyperparameter Tuning

After defining the more promising models for both the binary and the multiclass classification problems, we can proceed to their hyperparameters tuning. Remembering our earlier results:

- Binary classification: we'll optimize a KNN model;
- Multiclass classification: we'll optimize a Decision Tree.

I. BINARY CLASSIFICATION

For the binary classification we found out that the KNN was the model with higher accuracy and still balanced when taking metrics as precision and recall in consideration. The model we build before wasn't the one with the best performance yet. The main parameter of a KNN model is the number of neighbors we're considering. We used the default parameters for the classifier (i.e., 5 neighbors), and now we're going to vary this number in order to find the optimal one. Here are the results for the training and testing samples using the 10 K-Fold cross validation procedure:

Table 9. Results for the training and testing samples for the KNN model in the binary classification problem.

Number of neighbors	Mean acc. 10-Fold (train)	Mean acc. 10-Fold (test)
2	0.932	0.778
3	0.901	0.803
4	0.880	0.776
5	0.851	0.793
6	0.845	0.778
7	0.824	0.781
8	0.821	0.776
9	0.809	0.774
10	0.812	0.770
15	0.783	0.767
20	0.778	0.765
25	0.773	0.767
30	0.771	0.767
35	0.769	0.761
40	0.768	0.760
45	0.765	0.755
50	0.763	0.753
75	0.750	0.747
100	0.749	0.748
150	0.750	0.749
200	0.749	0.749

The pattern may be easier to see in a plot:

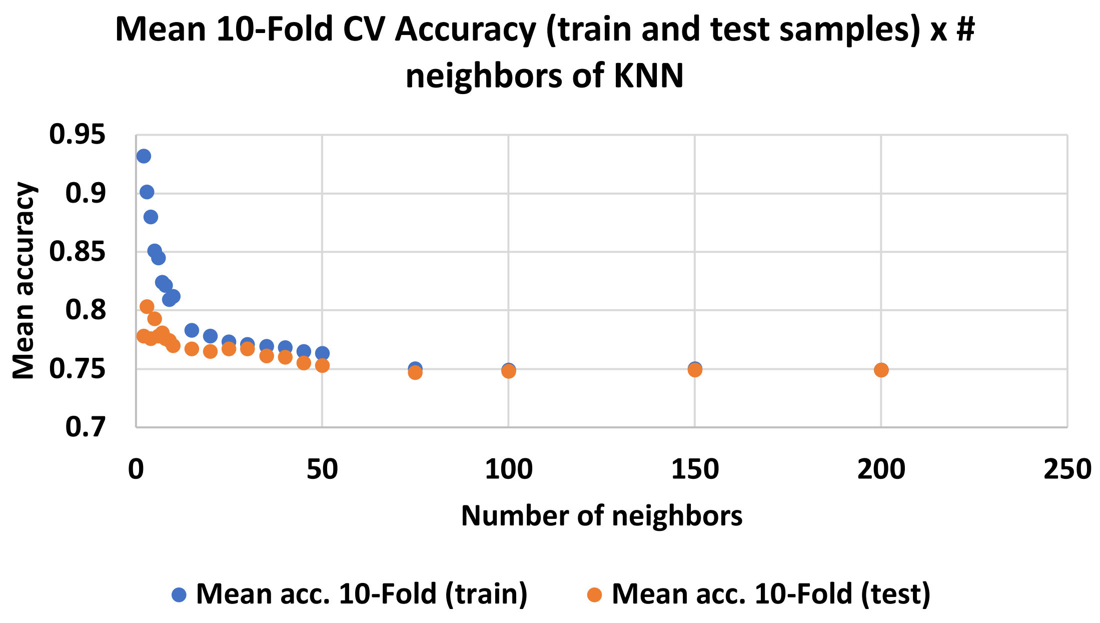


Figure 1. Mean 10-Fold CV Accuracy (train and test samples) x number of neighbors of KNN.

As expected, as the number of neighbors get higher, the model tends to be more specific (we're modeling the noise now), and the accuracy gets lower for both the training and the testing sets. The optimal value of K was chosen as the highest accuracy score in the testing sample, leading to **K* = 3. For this model, we have an accuracy (on the real test set, the one that wasn't used in the CV procedure) of 0.8313 and a Matthews coefficient of 0.41. Both values are higher than the baselines we had established earlier. The classification report can be seen below:**

Table 10. Classification report for the KNN (binary model).

	Precision	Recall
0	0.79	0.67
1	0.92	0.96

Recall and precision "live in conflict", in a way that when one is very high, usually the other one is too low. From the table above, we seem to have found the balance between these two metrics (as confirmed by the Matthews coefficient calculated before).

II. MULTICLASS CLASSIFICATION

For the multiclass problem we found out that the Decision Tree Classifier was the model with higher accuracy yet balanced when taking metrics as precision and recall in consideration. The model we build before wasn't the one with the best performance yet. The main parameter of a Decision Tree may be considered the maximum depth of each tree. We used the default parameters for the classifier (i.e., we didn't specify a maximum depth, which means that nodes were expended until all leaves were pure or less than a minimum threshold of samples), and now we're going to vary this number in order to find the optimal one. Here are the results for the training and testing samples using the 10 K-Fold cross validation procedure:

Table 11. Results for the training and testing samples for the Decision Tree model in the multiclass classification problem.

Maximum Depth	Mean acc. 10-Fold (train)	Mean acc. 10-Fold (test)
2	0.443	0.431
3	0.461	0.424
4	0.492	0.442
5	0.536	0.461
10	0.807	0.520
15	0.958	0.530
20	0.997	0.531
25	1.000	0.539
30	1.000	0.530
40	1.000	0.522
50	1.000	0.525
75	1.000	0.521
100	1.000	0.514

Plotting those numbers:

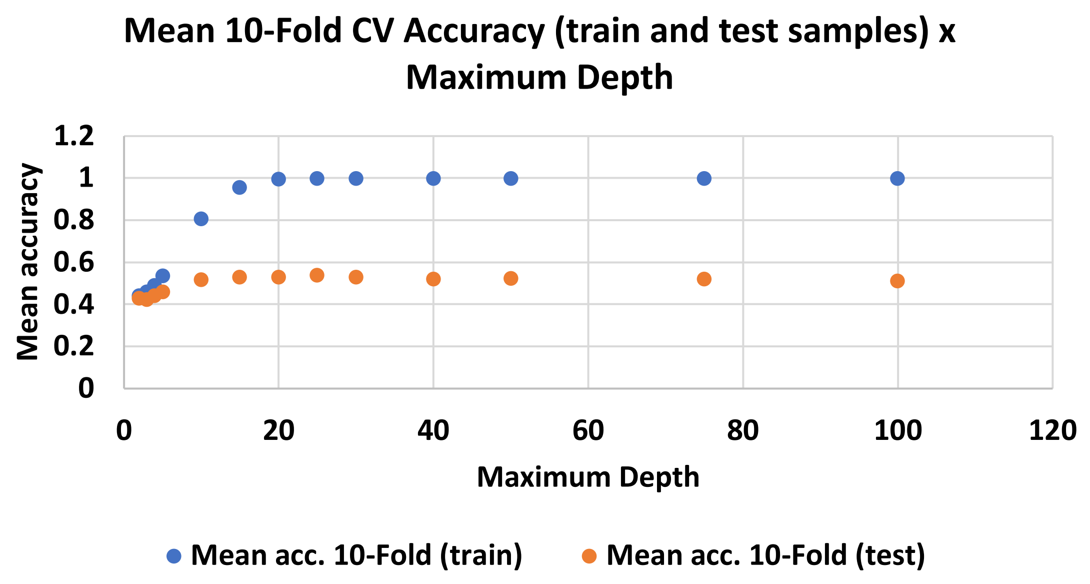


Figure 2. Mean 10-Fold CV Accuracy (train and test samples) x maximum depth of the Decision Tree.

As expected, as the maximum depth get higher, the model performs better and the accuracy gets higher, up to the point that we have an accuracy of 1 in the training set. This is not accompanied by an accuracy of 1 in the testing set, which gets stable at aprox. 0.5. This difference between the training and testing sets may indicate a case of overfitting, and the need of further changes in the trees' hyperparameters. For our purposes, we'll stick with the maximum value (in the test set) found above: **max_depth* = 25. For this model, we have an accuracy (on the real test set, the one that wasn't used in the CV procedure) of 0.576 and a Matthews coefficient of 0.40. Both values are higher than the baselines we had established earlier. The classification report can be seen below:**

Table 12. Classification report for the Decision Tree Classifier of the multiclass model.

	Precision	**Recall**
1	0.55	0.62
2	0.42	0.43
3	0.70	0.69
4	0.55	0.46
5	0.35	0.38

Recall and precision "live in conflict", in a way that when one is very high, usually the other one is too low. From the table above, we seem to have found the balance between these two metrics (as confirmed by the Matthews coefficient calculated before).

5) Ensembling

For both classification models we decided to use a Random Forest as Ensembling technique. The Random Forest is an ensemble of decision trees (which were already good candidates for both problems, as we've seen before). The hyperparameters tuned were:

- Bootstrap ('true' or 'false'), indicating whether bootstrap samples are used when building trees;
- Max_depth (5, 10, 15, 20 or 25), which is the maximum depth of each tree;
- N_estimators (20, 50 or 100), indicating the number of trees in the forest.

Since we have (2 x 5 x 3) = 30 combinations of parameters, we won't display all of the results in tables/charts, but only the significative/optimum ones.

I. BINARY CLASSIFICATION

- Best parameters' set:
  - Bootstrap: false
  - Max_depth: 20
  - N_estimators: 50
- Accuracy of the test set (15%) for the optimized model: 0.90
- Matthews coefficient (test set) for the optimized model: 0.67

II. MULTICLASS CLASSIFICATION

- Best parameters' set:
  - Bootstrap: false
  - Max_depth: 25
  - N_estimators: 100
- Accuracy of the test set (15%) for the optimized model: 0.75
- Matthews coefficient (test set) for the optimized model: 0.62

6) Conclusions

From all of the results shown above, we can conclude that for both problems (binary and multiclass), the Ensembling worked out better than the individual models themselves (what was expected). In both cases we achieved an accuracy over 70%, being as high as 90% for the binary classification. This illustrates the difference between a binary and a multiclass problem: when we have more than one class in our target variable it usually is more difficult to get high performances; in some types of models, for example, we have to build more than one individual model (in an approach known as "OvR"- one versus rest). Another interesting aspect is the balance between precision and recall. Here we tried to maintain a balance between both of them though the Matthews coefficient (which also takes into consideration the negative evaluations in the classification problem). Since we're analyzing a credit risk model, maybe we could have been more restrict when evaluating the precision (and more leninent about the recall): the opportunity cost of losing one misclassified prospective client is lower than the cost of acting on a bad client (who may turn into a default case).

CHAPTER 2
Economic Cycle problem

1) Introduction / Exploratory Data Analysis

In this chapter, we are going to analyze MLF_GP2_EconCycle dataset, which contains 223 monthly observations of the US Treasury bond yield curve, the commercial paper yield curve and the USPHCI Economic Activity Index.

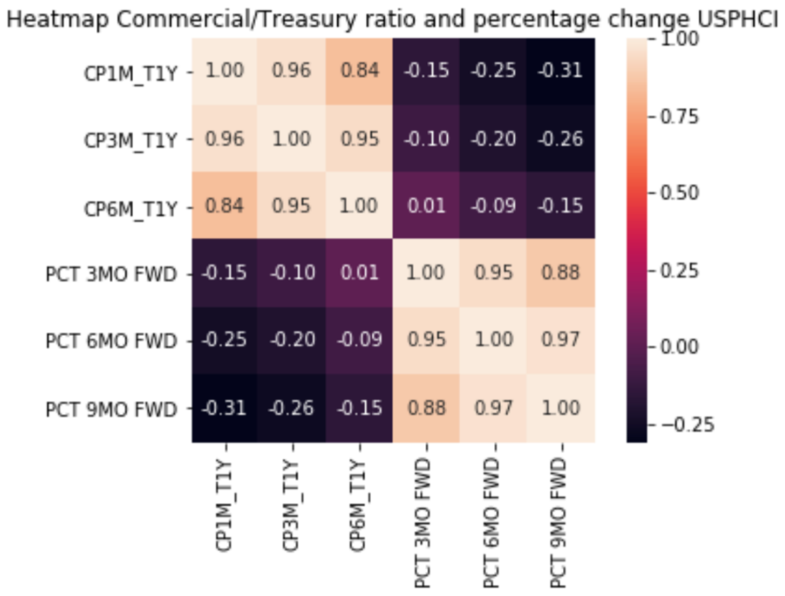
Since the data is continuous, we are going to build regression model using the techniques we learnt from course. According to the original dataset and instruction, the commercial paper yield curve divided by Treasury bond yield curve are CP_M_T_Y columns, which means those columns with only Treasury bond yield curve and Commercial paper yield curve are already influenced in these compound columns. However, there are only three columns with the compound information. In order to capture all the information, we make the compound columns with the division computation and add to the predictors column.

The targets are three columns which represent the changes in the USPHCI for three months forwards, six months forwards and nine months forwards.

Since it's the regression relationship between the predictors and targets, we what to know there linear relationship using the scatter plot and heatmap.

Here we only select columns with 'CP1M_T1Y','CP3M_T1Y','CP6M_T1Y', 'PCT 3MO FWD','PCT 6MO FWD','PCT 9MO FWD'.

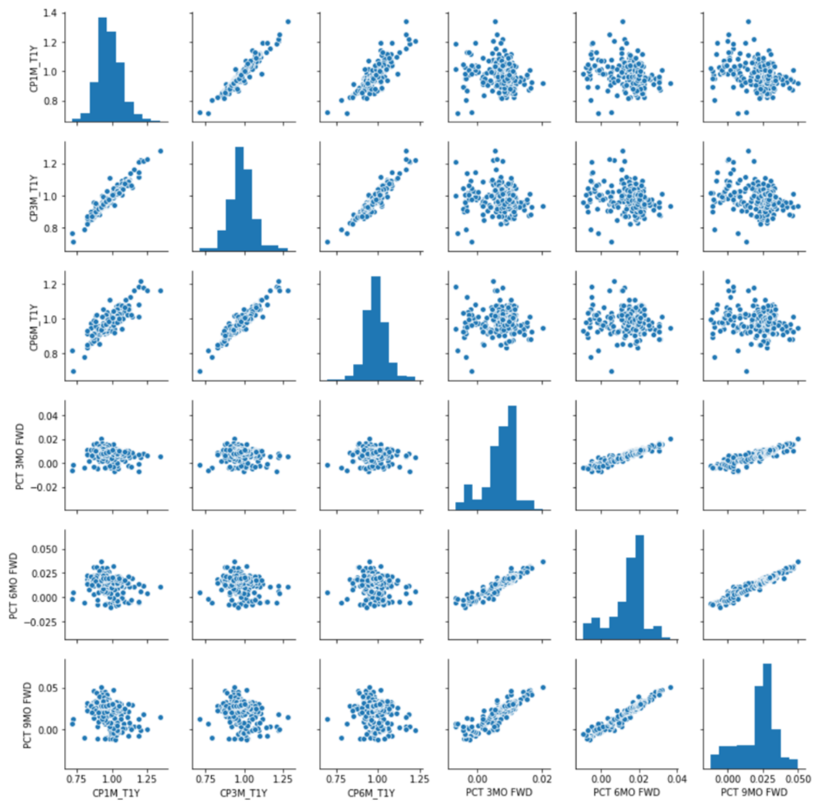
Image1. Heatmap



	CP1M_T1Y	CP3M_T1Y	CP6M_T1Y	PCT 3MO FWD	PCT 6MO FWD	PCT 9MO FWD
CP1M_T1Y	1.00	0.96	0.84	-0.15	-0.25	-0.31
CP3M_T1Y	0.96	1.00	0.95	-0.10	-0.20	-0.26
CP6M_T1Y	0.84	0.95	1.00	0.01	-0.09	-0.15
PCT 3MO FWD	-0.15	-0.10	0.01	1.00	0.95	0.88
PCT 6MO FWD	-0.25	-0.20	-0.09	0.95	1.00	0.97
PCT 9MO FWD	-0.31	-0.26	-0.15	0.88	0.97	1.00

From the heatmap, we noticed the relationship between our targets and the potential predictors are slightly negative and near 0.

Image2. Scatter plot



From the scatter plot, we notice since our dataset only has 223 samples, the relationships between potential predictors and targets are not very obvious.

2) Preprocessing / Feature Extraction / Feature Selection

In this part, we are going to do some prepocessing about our data and select proper features to predict our targets. Since the data have all different scales, we first use standard scaler for both X and y data. Then we split them into 0.4 size of test set. We also name 3 targets as: y1 = 'PCT 3MO FWD', y2='PCT 6MO FWD', y3='PCT 9MO FWD', which will be more convienent for our furture steps.

We also want to know how to select features for future prediction, so we do the Ridge and Lasso regression in this part.

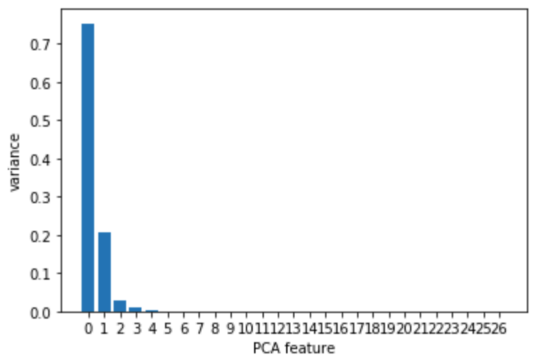
The result for Lasso regression, the coefficient for each of the X features with the three predictors with alpha as 0.0 is:

Table1: coefficient using Lasso regression

X_features	PCT 3MO FWD	PCT 6MO FWD	PCT 9MO FWD
T1Y Index	-0.006535	-0.012524	-0.020432
T2Y Index	-0.000431	0.002238	0.011206
T3Y Index	0.000789	-0.000608	-0.002978
T5Y Index	0.000636	0.001643	-0.000295
T7Y Index	0.001504	0.002643	0.003227
T10Y Index	0.004109	0.006936	0.008602
CP1M	0.005610	0.007101	0.005906
CP3M	-0.003039	-0.003562	-0.001786
CP6M	-0.003428	-0.005389	-0.005426
CP1M_T1Y	-0.208627	-0.400937	-0.477820
CP3M_T1Y	0.046799	0.098663	0.142750
CP6M_T1Y	0.103199	0.212184	0.253251
CP1M_T2Y	0.005551	-0.020908	-0.128530
CP3M_T2Y	-0.008092	-0.015841	-0.014966
CP6M_T2Y	-0.000491	-0.018376	-0.055672
CP1M_T3Y	0.040254	0.128636	0.242647
CP3M_T3Y	-0.013827	-0.023471	-0.017374
CP6M_T3Y	-0.003529	-0.018019	-0.036754
CP1M_T5Y	0.092936	0.174435	0.257379
CP3M_T5Y	-0.014884	-0.026797	-0.021243
CP6M_T5Y	-0.004803	-0.021347	-0.031784
CP1M_T7Y	0.043411	0.087092	0.087040
CP3M_T7Y	-0.017692	-0.031370	-0.027343
CP6M_T7Y	-0.010272	-0.032774	-0.047622
CP1M_T10Y	0.004454	0.013408	-0.020930
CP3M_T10Y	-0.020978	-0.035705	-0.032284
CP6M_T10Y	-0.016935	-0.043188	-0.060765

From the coefficienct, we can choose at least 3 features as our predictors, which is CP1M_T1Y, CP6M_T1Y, CP1M_T5Y. This can be one way of suggestion how to select features, However, we can also use PCA to extract the features.

Image 3. PCA feature extraction, explained variance



```
[0.75375772 0.95917308 0.98761052 0.99712438 0.9984989  0.99925584
 0.99955664 0.99971799 0.9998721  0.9999315  0.99996392 0.99998139
 0.99999219 0.99999748 0.99999842 0.99999931 0.99999952 0.9999997
 0.99999985 0.99999994 0.99999997 0.99999999 0.99999999 1.
 1.         1.         1.        ]
```

From this image, we noticed it's feasible to extract three features to build our model. Here, we prepossessing our data into PCA with n_components as 4 to extract data and go to the next section.

3) Model Fitting and Evaluation

I) LINEAR REGRESSION

In the first model, we try to use the basic linear regression model to predict X_features with three y targets.

Table 2. linear regression model

	intercept	X1	X2	X3	X4	MSE	R^2	RMSE
Y1	**0.006**	-0.073	-0.077	0.036	-0.223			
Y1_train						**0.852**	**0.148**	**0.923**
Y1_test						**0.781**	**0.219**	**0.884**
Y2	**0.001**	**-0.1154**	**-0.08**	**0.0345**	**-0.132**			
Y2_train						**0.835**	**0.165**	**0.914**
Y2_test						**0.698**	**0.302**	**0.835**
Y3	**0.003**	**-0.0961**	**-0.06**	**0.04**	**-0.12**			
Y3_train						**0.777**	**0.223**	**0.882**
c						**0.646**	**0.345**	**0.804**

From the result we can see the linear relationship is not very obvious.

II) Decision tree regressor

Table 3. decision tree regression performance

	MSE	R^2	RMSE
Y1_train	**0.674**	**0.326**	**0.821**
Y1_test	**0.844**	**0.167**	**0.913**
Y2_train	**0.617**	**0.383**	**0.785**
Y2_test	**0.695**	**0.305**	**0.834**
Y3_train	**0.517**	**0.483**	**0.719**
Y3_train	**0.612**	**0.388**	**0.782**

From the performance table, we found the decision tree regression have better performance than linear regression in general.

III) SVR

Table 3. Support vector regressor performance

	MSE	R^2	RMSE
Y1_train	**0.868**	**0.132**	**0.932**
Y1_test	**0.824**	**0.176**	**0.908**
Y2_train	**0.823**	**0.177**	**0.907**
Y2_test	**0.746**	**0.254**	**0.864**
Y3_train	**0.806**	**0.194**	**0.898**
Y3_test	**0.697**	**0.303**	**0.835**

From the performance table, we found SVR model isn't as good as decision tree regressor. Among three models, we found the best one would be decision tree regression. We are going to use decision tree as our basic model to do the following section.

4) Hyperparameter Tuning

Firstly, we build random forest model to see the performance.

Table 4. Random Forest Regressor performance

	MSE	**R^2**	**RMSE**
Y1_train	**0.077**	**0.923**	**0.277**
Y1_test	**0.584**	**0.416**	**0.764**
Y2_train	**0.071**	**0.929**	**0.267**
Y2_test	**0.476**	**0.524**	**0.690**
Y3_train	**0.078**	**0.922**	**0.280**
Y3_test	**0.393**	**0.607**	**0.627**

From the performance table, we notice that the performance for random forrest regressor with y3 has best R^2.

In order to find better parameeters to fit the model, we use gridsearchCV with the random forest regressor as estimator to tuning parameter.

The best model for y1 is:

- **n_estimators: 50**
- **max_features: auto**
- **min_samples_leaf: 2**

The R^2 score**: Train set: 0.834**
Test set: 0.375

The best model for y2 is:

- **n_estimators: 50**
- **max_features: auto**
- **min_samples_leaf: 2**

The R^2 score**: Train set: 0.874**
Test set: 0.518

The best model for y3 is:

- **n_estimators: 100**
- **max_features: auto**
- **min_samples_leaf: 2**

The R^2 score**: Train set: 0.880**
Test set: 0.588

5) Ensembling

Here, we using GradientBoostingRegressor to do the ensembling step. And we get our best R^2 score during the prediction of y3. Here we only shows the result for y3.

The best parameter for y3 model is:

- **Max depth : 7**

- **Min_samples_split :5**
- **Min_samples_leaf : 1**
- **Max_features:0.75**
- **N_estimators:60**
- **Max_leaf_nodes:14**

The R^2 score: Train set: 0.994
Test set: 0.618
MSE score: Train set: 0.006
Test set: 0.382
RMSE score : Train set: 0.080
Test set: 0.618

6) Conclusions

Even after a lot of tuning parameters step, we still found our final model is not good enough for having 0.8 or higher R^2. The best model we can get from regression is random forest model with specific parameters.
This might because we don’t have enough observations for us to fit the model, which we only have 223 samples. Also from the scatter plot, we already notice there might not be a very stronge relationship between our targets and our predictors.

Appendix

Chapter1: **https://github.com/Khavya-Chandrasekaran/IE598_F19_GP**

Chapter2: https://github.com/xuehuic2/IE598MLF_Group_project_chapter2