Finance and Machine Learning

CSS 100 Group Project

Heng-Chia Liu, Beibei Du, Dominic Ricci, and Sarah Kagzi

In this Presentation

End-to-End Machine Learning Project

1. Introduction - The Big Picture

- Wealth management is one of the most important factors for people to improve their life quality and stability.
- Financial assets are the most convenient because they provide high liquidity and return on investment.
- “Risk” has become the major concern that stops people from investing because a bad investment or speculation can harm wealth significantly.
- Our goal - help investors make a better decision in their security selection and trading process
- Our focus - stock investments

1. Introduction - Framing the Problem

- In our project, we will focus on finding securities that will outperform the market
- We will first convert the daily log-return data to daily Sharpe Ratio (a risk/return metric commonly used in finance to assess the performance of an asset) for all S&P 500 components and classify the "market beaters".
- These market beaters are classified by comparing the Sharpe ratio for a particular stock versus the Sharpe ratio of the whole SP&Y for that time period.

1. Introduction - Our Project Plan

Components:

1. Fama-French five-factor model - explains between 71% and 94% of the cross-section variance of expected returns for 5 fundamental factors.
   We will use the FF5F to estimate the factors exposure and relevant statistics for each of the S&P 500 components.
2. Feature Engineering - The features we construct are
   a. factor exposures
   b. statistical significance measurements
   c. Time-series possibility of beating the market
3. Classification - Ensemble Learning using
   a. SGD Classifier
   b. Logistic Regression Classifier
   c. Support Vector Machine Classifier
   d. Random Forest Classifier
   e. Aggregation
4. Target variable - binary outcome of the next trading period indicating whether a security beat the market in that time period

2. Getting the Data

- We use the **S&P 500** which contains the 500 largest publicly traded companies in the U.S. as the benchmark for market beaters.
- The reason for adopting the S&P as the benchmark is that if we can't select an investment that can outperform the market portfolio, we can just simply invest in the market portfolio (the S&P) instead.
- We collected fundamental data from annual reports of all companies in the S&P 500 from Yahoo Finance through an unofficial API "yfinance".
- The financial information we extracted are listed on the following slide

- Annual Market Capital(Size),
- Price to Book Ratio(Value),
- Return on Equity(Profitability),
- Asset Growth Rate(Investment)

2. Getting the Data

The companies that have higher market capital than the median will be classified as Big (B), and those that have lower market capital than the median will be classified as Small(S).

For Value, Profitability and Investment, the companies obtain values higher than top 25 percentile will be classified as High(H), Robust(R),and Aggressive(A), and those obtain values lower than bottom 25 percentile will be classified as Low(L), Weak(W), and Conservative(C), respectfully

3. Preparing the Data for Machine Learning ~~al~~gorithms

Examples included in the code:

1. **FF_assign_label(index_comp_info)** :- Assigns a label indicating the companies' characteristics, Big cap, High value...
2. **FF_factor_classifier(index_comp_info_with_label)** :- Classifies companies to 18 groups for later return calculation
3. **FF_classes_return(market_components_return, list_of_group_info, axis=True)** :- Calculates the value-weighted return of each group above
4. **FF_calc_factors(classes_return_df, df = True)** :- Calculates 5 factors (5 Return Spreads)

4. Selecting a Model - Fama French

- Fama French five factor model captures the average return in a portfolio of securities
- Equation:

$$R_{i_t} - R_F = a_i + b_i(R_{M_t} - R_{F_t}) + + s_i SMB_t + \\ h_i HML_t + r_i RMW_t + c_i CMA_t + e_{i_t}$$

4. Selecting a Model – Fama French

THE FIVE FACTORS:

- $R_m - R_f$
- SMB
- HML
- RMW
- CMA

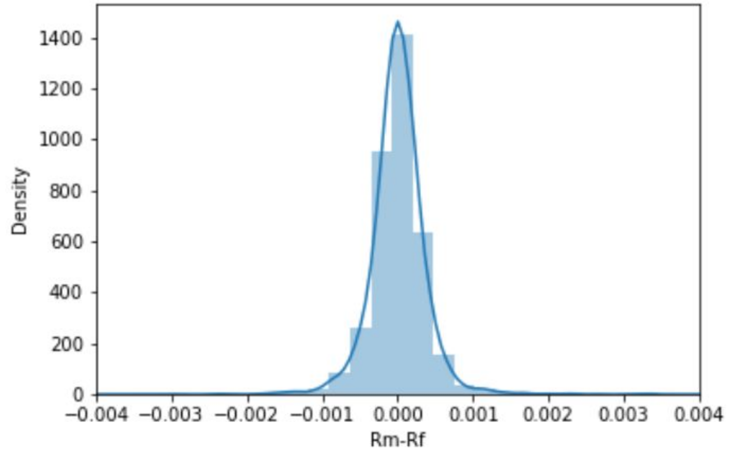


Figure 4. Rm-Rf Distribution Follows Standard Normal Distribution

4. Selecting a Model – Fama French

- $R_m - R_f$
- SMB
- HML
- RMW
- CMA

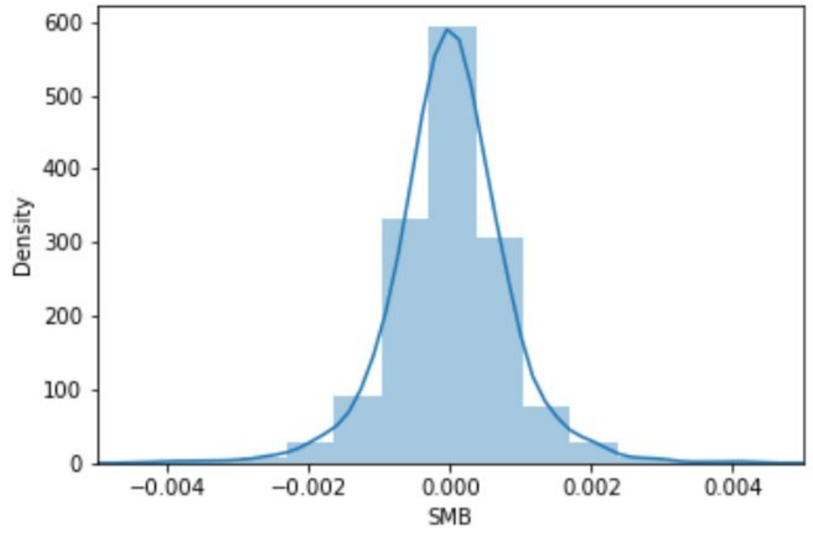


Figure 5. SMB Distribution Follows Standard Normal Distribution

4. Selecting a Model – Fama French

- $R_m - R_f$
- SMB
- HML
- RMW
- CMA

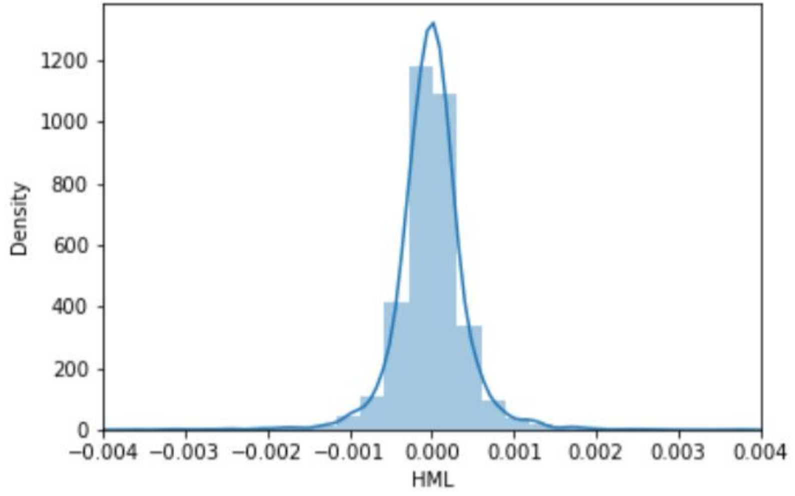


Figure 6. HML Distribution Follows Standard Normal Distribution

4. Selecting a Model – Fama French

- $R_m - R_f$
- SMB
- HML
- RMW
- CMA

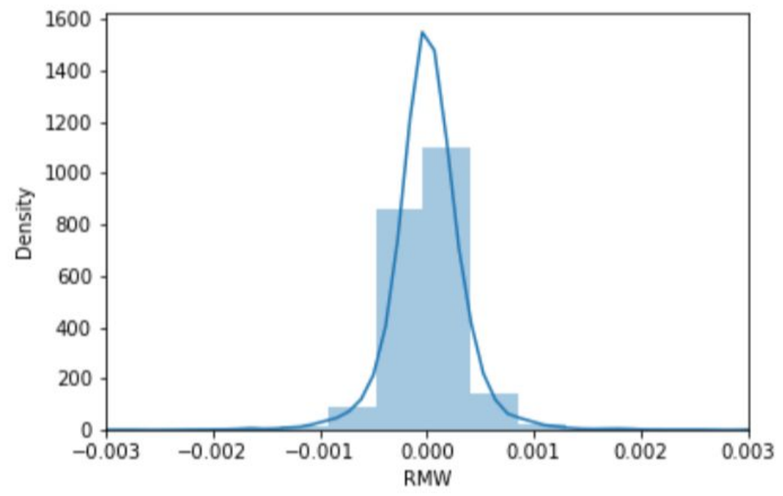


Figure 7. RMW Distribution Follows Standard Normal Distribution

4. Selecting a Model – Fama French

- $R_m - R_f$
- SMB
- HML
- RMW
- CMA

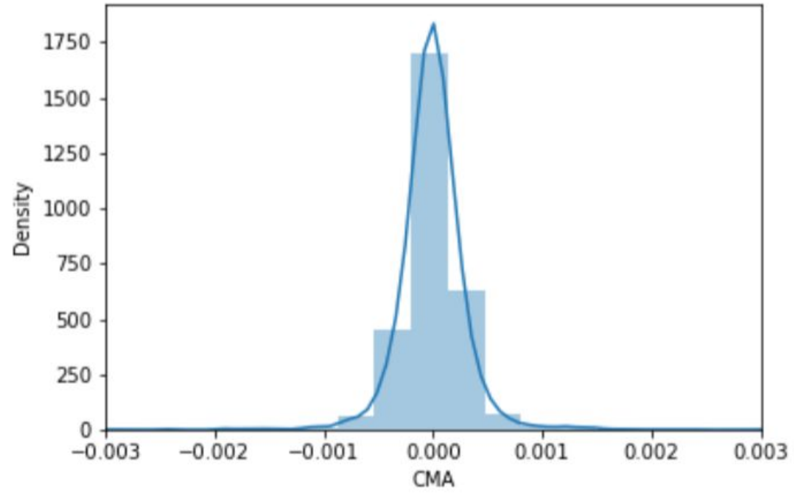
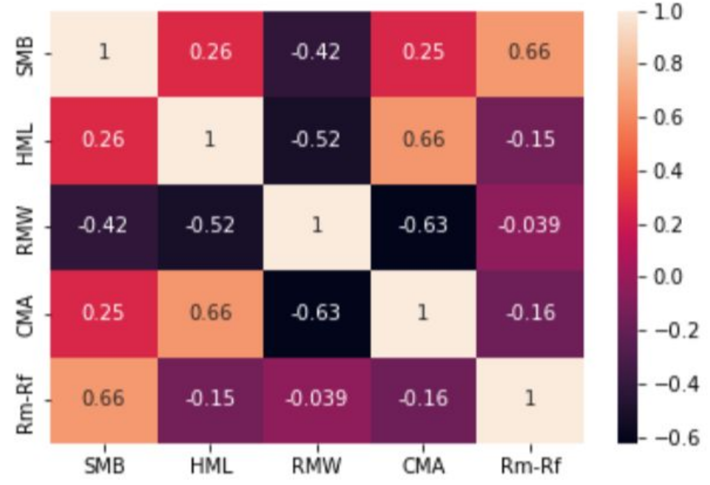


Figure 8. CMA Distribution Follows Standard Normal Distribution

4. Selecting a Model – Fama French



	SMB	HML	RMW	CMA	Rm-Rf
SMB	1	0.26	-0.42	0.25	0.66
HML	0.26	1	-0.52	0.66	-0.15
RMW	-0.42	-0.52	1	-0.63	-0.039
CMA	0.25	0.66	-0.63	1	-0.16
Rm-Rf	0.66	-0.15	-0.039	-0.16	1

Correlation Between 2 min Fama French Factors

CORRELATION BETWEEN THE 5 FACTORS:

In the 2 minute return scale, RMW shows a strong correlation between CMA and HML. As we suspect multicollinearity will exist in linear regression model, we will also implement Ridge, Lasso, and Elastic Net with optional polynomial features to estimate the factors exposures (Coefficient of the 5 Factors).

5. Fine Tuning our Model Using Machine Learning Methods

5. A. Feature Engineering - Finding Market Beaters

Our first goal was constructing a data frame which identified what securities in the SP&Y 500 beat the market in a time interval. To do this we collected data from the yfinance library of stock return percentages over a 2 minute interval using the percent change formula below:

$$R_{t,i} = \frac{r_{t-1,i} - r_{t,i}}{r_{t-1,i}}$$

$$Sharpe_{t,i} = \frac{R_{t,i} - R_f}{\sigma_i}$$

5. A. Feature Engineering - Finding Market Beaters

This formula is used to compute security returns and sharpe ratios. Each 'i' index denotes an individual security, while each 't' denotes a specific time interval. We then compute for the market as a whole over out 2 minute data frame when *i* = *m* . We then return dummy variables based on the comparison below

$$\text{if } Sharpe_{t,i} \leq Sharpe_{t,m} \implies d_{t,i} = 0$$

$$\text{if } Sharpe_{t,i} > Sharpe_{t,m} \implies d_{t,i} = 1$$

These dummy-variables will then be used in the time-series model in order to train and test our data frame.

5. A. Iterative Regression Methods

As described in the previous section the Fama-French model is extremely reliable when it comes to estimating return based on market factors. This is why our algorithm relies on an iterative regression function.

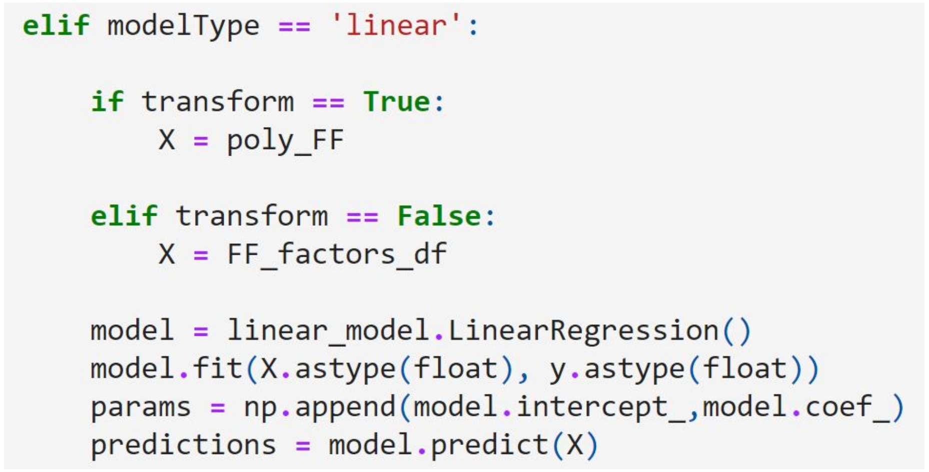
This function attempts to find the values of the coefficients of $b_{t,i}$, $h_{t,i}$, $r_{t,i}$, and $c_{t,i}$ from the last 150 observations, then stores them in a dataframe with their respective performance values. These coefficients can be determined by the following four models:

1. Linear Model

We run the regression:

$$R_{t,i} - R_F = a_{t,i} + b_{t,i}(R_{t,m} - R_F) + s_{t,i}SMB_t + \\ h_{t,i}HML_t + r_{t,i}RMW_t + c_{t,i}CMA_t + e_{t,i}$$

The coefficients and their respective p-values are stored in a data frame which will be used in classification to find the market winners.

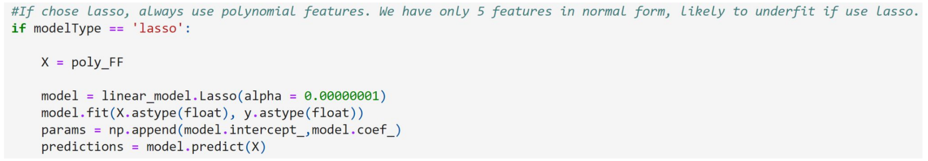


```
elif modelType == 'linear':

    if transform == True:
        X = poly_FF

    elif transform == False:
        X = FF_factors_df

    model = linear_model.LinearRegression()
    model.fit(X.astype(float), y.astype(float))
    params = np.append(model.intercept_,model.coef_)
    predictions = model.predict(X)
```

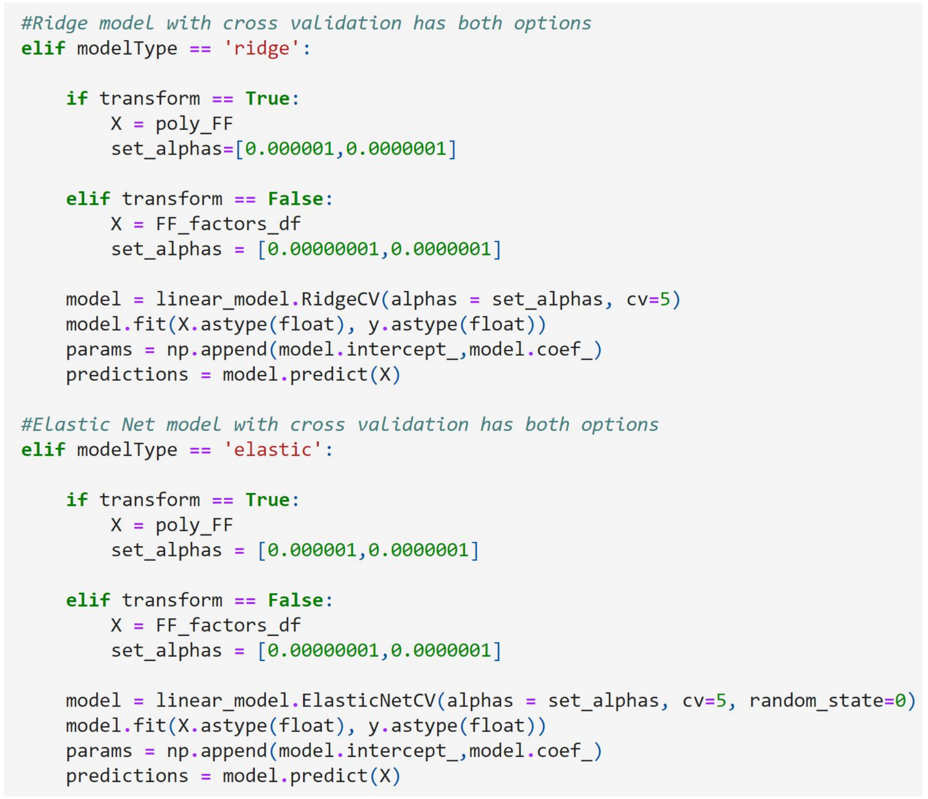
2. Other Models: Lasso, Ridge, Elastic Net

We also used other linear models to estimate coefficients, the Elastic Net model with interaction polynomial features transformation performs the best. We believe this result is due to following reasons:

1. Multi-colinearity exists between Fama-French return factors.
2. Linear relationship is not enough to explain return by its factors.



```python
#If chose lasso, always use polynomial features. We have only 5 features in normal form, likely to underfit if use lasso.
if modelType == 'lasso':

    X = poly_FF

    model = linear_model.Lasso(alpha = 0.00000001)
    model.fit(X.astype(float), y.astype(float))
    params = np.append(model.intercept_,model.coef_)
    predictions = model.predict(X)
```



```python
#Ridge model with cross validation has both options
elif modelType == 'ridge':

    if transform == True:
        X = poly_FF
        set_alphas=[0.000001,0.0000001]

    elif transform == False:
        X = FF_factors_df
        set_alphas = [0.00000001,0.0000001]

    model = linear_model.RidgeCV(alphas = set_alphas, cv=5)
    model.fit(X.astype(float), y.astype(float))
    params = np.append(model.intercept_,model.coef_)
    predictions = model.predict(X)

#Elastic Net model with cross validation has both options
elif modelType == 'elastic':

    if transform == True:
        X = poly_FF
        set_alphas = [0.000001,0.0000001]

    elif transform == False:
        X = FF_factors_df
        set_alphas = [0.00000001,0.0000001]

    model = linear_model.ElasticNetCV(alphas = set_alphas, cv=5, random_state=0)
    model.fit(X.astype(float), y.astype(float))
    params = np.append(model.intercept_,model.coef_)
    predictions = model.predict(X)
```

5. B. Time Series Model

- Our goal is to predict whether a given security or portfolio will outperform the market in the next trading period. To do so, we estimate factor exposure at a certain time t_n considering the most recent observations only, then use the exposures and relevant statistics to predict the outcome at t_{n+1} .

- We will roll the model and make prediction for at least 10 trading intervals, and will collect accuracy metrics of time-series prediction.

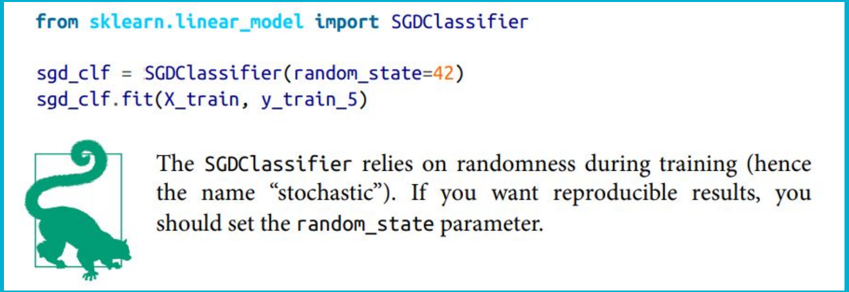
- We use ensemble learning in order to use classifiers in combination.
    - Stochastic Gradient Descent Classifier,
    - Logistic Regression Classifier
    - Support Vector Machine Classifier
    - Random Forest Classifier

- Finally, we use the Voting Classifier and Bootstrap Aggregation to combine these classifiers into our final classifier

1.

1. Stochastic Gradient Descent Classifier

- takes the training set and calculates the gradient based off of one randomly selected instance at every iteration.
- We chose to use the SGD because unlike the Batch Gradient Classifier, it works well on large datasets, and unlike the Mini Batch Gradient Descent Classifier, it is good at avoiding local minima - this is a crucial aspect because of how textured the financial stock data can be.



```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(random_state=42)
sgd_clf.fit(X_train, y_train_5)
```

The `SGDClassifier` relies on randomness during training (hence the name "stochastic"). If you want reproducible results, you should set the `random_state` parameter.

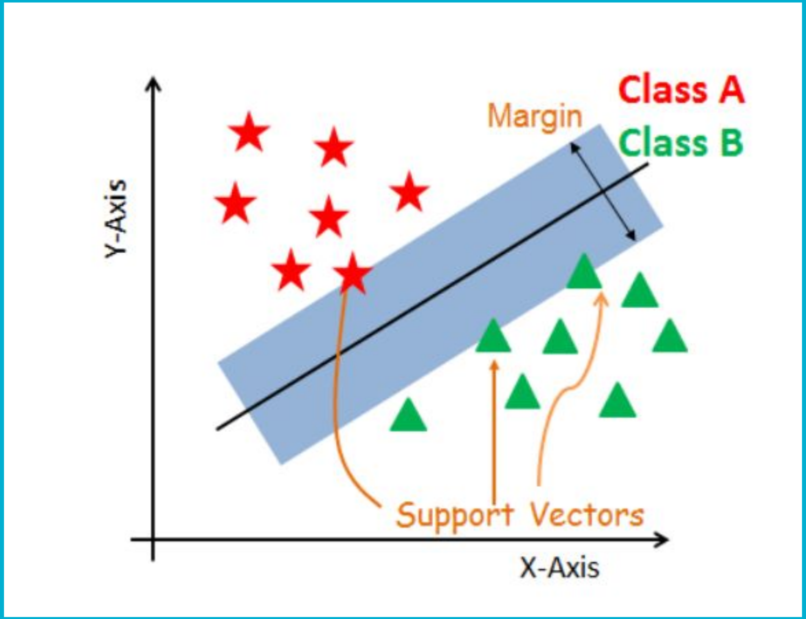
2. Logistic Regression Classifier

- Logistic Regression Classifier estimates the probability of one binary classifier belonging to a certain class. If the probability is greater than 0.5, it returns 1, and if not, it returns 0. This is extremely useful for our project because it helps us classify into different classes (which in our case is whether or not individual instances of stocks perform better than the average). Other than the fact that it's a binary classifier, we also value the fact that its classification is based on the minimization of the cost function, the "log loss". As mentioned in the textbook, "HandsOn Machine Learning", while this cost function does not have a specific equation for minimization, the partial derivatives (Equation 5) give a gradient which we can then use the SGD classifier with in order to get a more accurate classifier for the project.

$$\frac{\partial}{\partial\theta_j} J(\theta) = \frac{1}{m}\Sigma[\sigma(\theta^T x^i) - y^i)x_j^i]$$

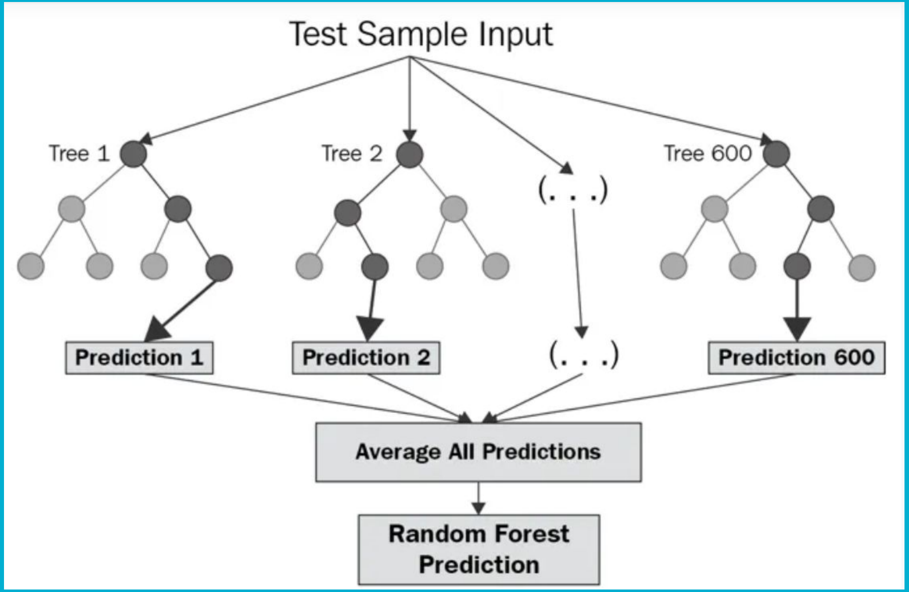
3. Support Vector Classifier

- SVM is a binary classifier that performs classifications on multiple binary classifiers. We chose to incorporate an SVM classifier because we wanted to take advantage of the powerful nature of the Gaussian RBF Kernel. Since we are working with a large and complex dataset, from the computational and run-time sense, it would be expensive and time consuming to use a similarity features method, but the Gaussian RBF allows us to obtain a similar result without adding those similarity features. Thus, for the purposes of our project, it was a powerful tool that we wanted to incorporate in our classifier.



4. Random Forest Classifier

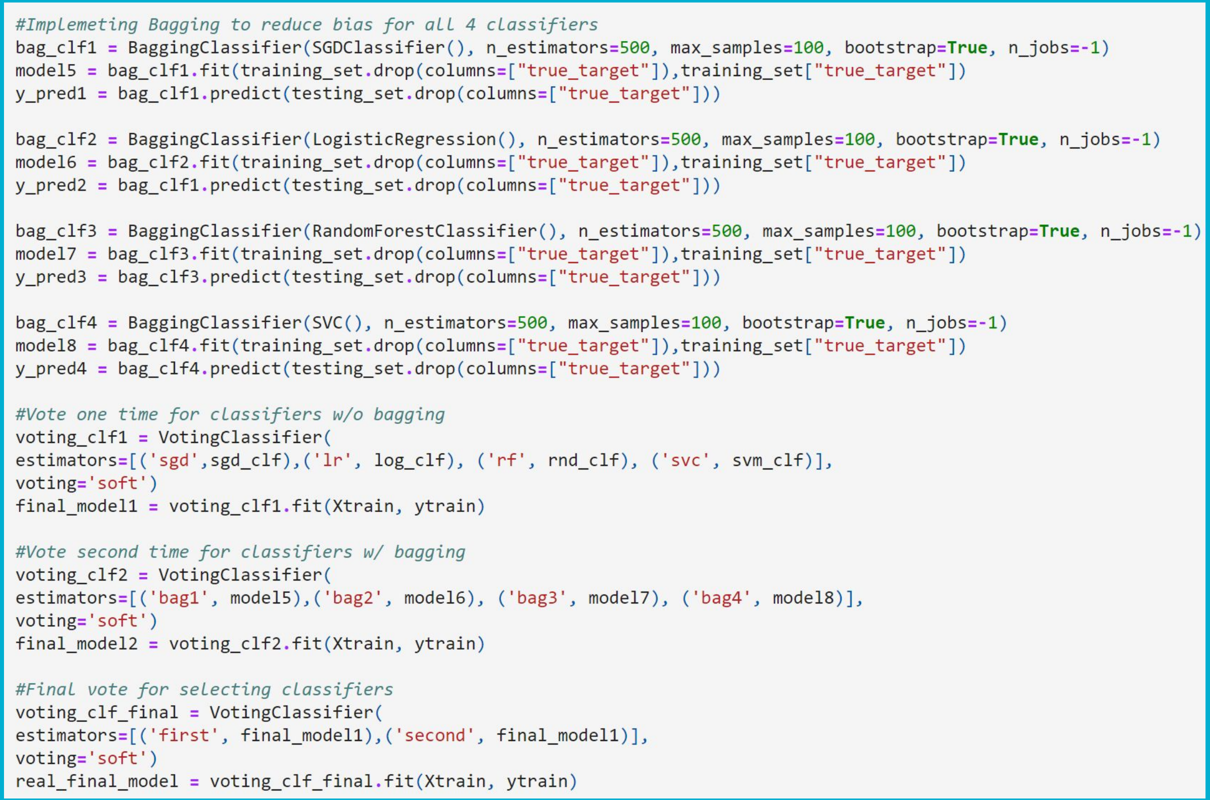
- Random Forest Classifier also classify binary variables to different classes by the hidden decision trees. Through the bagging method, we have many sets of training set and put them into the Decision Tree Classifier, and the hyper-parameters will control how the algorithm ensemble itself.After running multiple Decision Trees, we will output one final result from the previous Decision Tree Classifiers. And this is the reason why this algorithm is called Random Forest Classifier. The Decision Tree Classifiers are multiple questions that require binary answers. And the max depth is dependent on how many questions we have to ask. The bagging and random feature generated from the algorithm allows for the maximized randomness and be representative. A more accurate result will be generated by the multiple Decision Tree Classifiers.



5. B. Time Series Model - Ensemble Learning

First, we use four bagging classifiers on the aforementioned four models, and then by utilizing voting, we combine the four classifiers with bagging and without bagging to get one final classifier.

This is called the wisdom of the crowd according to the textbook. Thus we can see how multiple classifiers can benefit the final result and a better result is generated by ensemble some possible algorithms.



```
#Implemeting Bagging to reduce bias for all 4 classifiers
bag_clf1 = BaggingClassifier(SGDClassifier(), n_estimators=500, max_samples=100, bootstrap=True, n_jobs=-1)
model5 = bag_clf1.fit(training_set.drop(columns=["true_target"]),training_set["true_target"])
y_pred1 = bag_clf1.predict(testing_set.drop(columns=["true_target"]))

bag_clf2 = BaggingClassifier(LogisticRegression(), n_estimators=500, max_samples=100, bootstrap=True, n_jobs=-1)
model6 = bag_clf2.fit(training_set.drop(columns=["true_target"]),training_set["true_target"])
y_pred2 = bag_clf1.predict(testing_set.drop(columns=["true_target"]))

bag_clf3 = BaggingClassifier(RandomForestClassifier(), n_estimators=500, max_samples=100, bootstrap=True, n_jobs=-1)
model7 = bag_clf3.fit(training_set.drop(columns=["true_target"]),training_set["true_target"])
y_pred3 = bag_clf3.predict(testing_set.drop(columns=["true_target"]))

bag_clf4 = BaggingClassifier(SVC(), n_estimators=500, max_samples=100, bootstrap=True, n_jobs=-1)
model8 = bag_clf4.fit(training_set.drop(columns=["true_target"]),training_set["true_target"])
y_pred4 = bag_clf4.predict(testing_set.drop(columns=["true_target"]))

#Vote one time for classifiers w/o bagging
voting_clf1 = VotingClassifier(
estimators=[('sgd',sgd_clf),('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
voting='soft')
final_model1 = voting_clf1.fit(Xtrain, ytrain)

#Vote second time for classifiers w/ bagging
voting_clf2 = VotingClassifier(
estimators=[('bag1', model5),('bag2', model6), ('bag3', model7), ('bag4', model8)],
voting='soft')
final_model2 = voting_clf2.fit(Xtrain, ytrain)

#Final vote for selecting classifiers
voting_clf_final = VotingClassifier(
estimators=[('first', final_model1),('second', final_model1)],
voting='soft')
real_final_model = voting_clf_final.fit(Xtrain, ytrain)
```

5. B. Time Series Model - Ensemble Learning

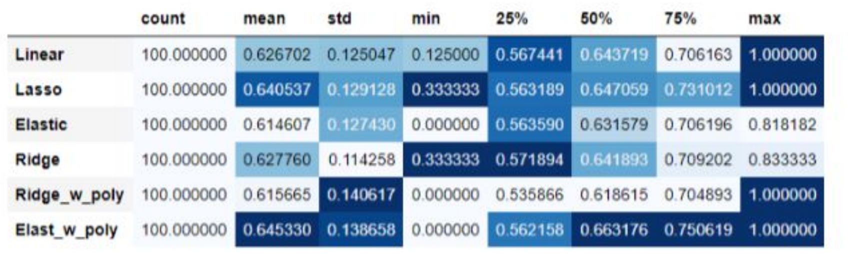
BAGGING (BOOTSTRAP AGGREGATING)

- ~~In~~ order to diversify our classifier, we decided to use Bagging to train the classifier on different subsets of the training set.
- Aggregates by taking Mode
- We chose bagging over pasting because while both bagging and pasting permit that training instances be picked up in the sample several times for many predictors, bagging additionally allows for the same predictor to pick up training instances in its sample multiple times.
- Aggregating using the bagging classifier reduces variance compared to the original predictors by themselves
- Thus, with the bagging classifier, our model will generalize better than just the predictors alone.
- However, despite all these benefits, since bagging diversifies the subsets that each predictor is trained on multiple times, there is a risk that bagging increases the bias. Thus, in order to protect our model from this, we also use voting classifiers to choose the best option - with bagging or without bagging.

VOTING CLASSIFIER

- We use the Voting Classifier to combine multiple models that we have used into one to predict the values.
- Relies on the Law of Large Numbers to arrive at the result that even with weak predictors that are only slightly better at predicting than a random guess, by correctly predicting the class with the majority "votes", the model can still reach an accuracy as high as 75% .
- Since our initial classifiers are very different from each other in their methods and technique of classification, we can rest assured that they will not make correlated errors, thus preventing a majority vote going to the wrong class.
- Moreover, to make our model more robust, we use soft voting, which selects the class with the highest probability since this adds more nuance to the voting classification procedure by giving more weight to the votes we know are accurate.
- In this way, we ensured that our ensemble's accuracy is high, and while calculating the precision and recall of our model, we got precision in the range of 0.61 - 0.65 (with the highest F1 score being 0.60)

6. Conclusion



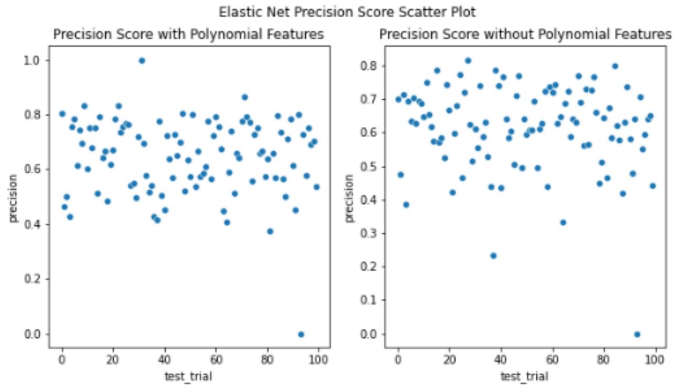
	count	mean	std	min	25%	50%	75%	max
Linear	100.000000	0.626702	0.125047	0.125000	0.567441	0.643719	0.706163	1.000000
Lasso	100.000000	0.640537	0.129128	0.333333	0.563189	0.647059	0.731012	1.000000
Elastic	100.000000	0.614607	0.127430	0.000000	0.563590	0.631579	0.706196	0.818182
Ridge	100.000000	0.627760	0.114258	0.333333	0.571894	0.641893	0.709202	0.833333
Ridge_w_poly	100.000000	0.615665	0.140617	0.000000	0.535866	0.618615	0.704893	1.000000
Elast_w_poly	100.000000	0.645330	0.138658	0.000000	0.562158	0.663176	0.750619	1.000000

Summary of Precision Scores for different Feature Estimates

To sum up, we found that the factors exposures are consistently useful determinants to predict the market beaters after we use 4 regression models and optional polynomial features transformation to estimate the factors exposures. While keeping the parameters and classification algorithm constant, all 6 different feature estimations result in at least 0.6 precision score on average in their 100 testing trials. Estimating features using Elastic Net with polynomial features provide us the highest average precision (0.645), whereas using plain Elastic Net provide us the lowest average precision (0.615).

6. Conclusion

By looking at the precision scatter plot of all models, we didn't find any evidence of serial correlation between trials on all models, so we believe our accuracy score is robust over time.

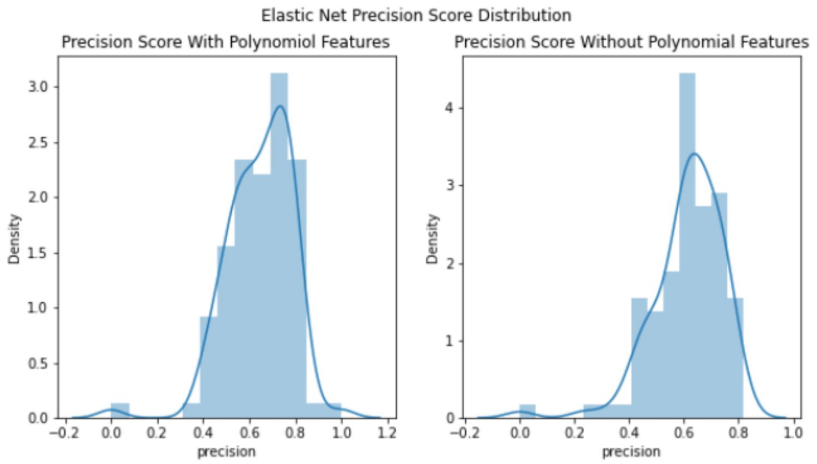


Precision Score Scatter (Best Performer and Worse Performer)

6. Conclusion

Although we observed a trade off between the level of precision score and its standard deviation, which means that well-performing estimators generally associate with instability, such an instability is mainly due to very small amount of outliers.

Moreover, by comparing the best and worst performers, we concluded that the model using Elastic Net with polynomial features perform the best since its distribution is more centered and yield better accuracy while obtaining the same amount of significant outliers.



Precision Distribution of Both Models

Summary

1. The Big Picture
2. Getting the Data
3. Preparing the Data for Machine Learning algorithms
4. Selecting a Model - Fama French
5. Fine Tuning our Model Using Machine Learning Methods
    a. Feature Engineering
    b. Time Series Model (Classification)
6. Conclusion and Results