Machine Learning Engineer Nanodegree

Capstone Project

Joe Importico

March 17, 2019

I. Definition

Project Overview

Forecasting asset prices is a practice that has been adopted by many and in recent years has been modernized by a group of investors known as Quants. Many investors of this ilk create systematic models with a core focus of understanding the behavior of an asset's price movement and how those prices will evolve over time. For any researcher hoping to achieve success in that endeavor, they are first faced with determining what data should be used for their research, as well as how they will go about acquiring it. Financial data is a scarce asset for many, especially for those not working in academia or at a large institutional asset manager, and it shines a light on the disadvantage retail traders face when beginning to tackle an already difficult problem in the form of forecasting asset prices.

Luckily, there are a few platforms and publicly available data sources that have been made available to the broader public to distribute the opportunity to take part in these types of research studies. For my capstone project, I will be using a number of these resources, such as Yahoo Finance to query historical asset prices and Quantopian for backtesting my signal in a simulated trading environment.

Problem Statement

In this project I have constructed a classification model that uses several price-based attributes to predict returns for a cross-sectional universe of assets. The broader project is comprised of the following steps (*note: these are documented here at a high-level*):

1. Collect Data
    a. Query historical pricing data for a universe of publicly traded companies and sector ETF's between 2003-12-31 and 2018-12-31.
    b. Import the Fama & French five factor model the Fama/French Data Library[1]
    c. Filter out companies trading at less than $5 per share
2. Feature Engineering
    a. Create time-series exposures to sectors and styles
    b. Treat time-series asset returns for outliers to normalize the return streams
    c. Construct several technical factors using the data collected in step 1.a
3. Exploratory Data Analysis

    a. Explore the features space, comprised of sector, style, and technical factors, for common start and end dates, outliers, and other potential transformations that should be applied to the data.
4. Model Construction
    a. Binarize the median 63-day future return around the median value for the universe each day. This boolean (1/0) value will serve as the target variable.
    b. Cross-validate the data while preserving spatial and temporal dependencies.
    c. Run a series of classification algorithms and evaluate the best model based on the F1 score.

There are a few, but important, considerations I want to point out with respect to the adjustments that have made throughout this analysis. The first is with respect to stationarity. Many statistical techniques have a deep-rooted assumption that the data being used is stationary, which, unfortunately, financial data is not. It is for that reason that I switched my target variable from closing price to a 63-day return horizon (the 63-day return horizon was chosen to reflect a three-month holding period). This adjustment provides the benefit of working with a more stationary time-series; however, returns can, especially in more volatile markets, have a fair amount of noise associated with them, so once the target was transformed into a return, it was then binarized relative to the median 63-day return horizon of the universe. Assets with a return greater than the median return received a value of one, otherwise zero. Other adjustments, such as the temporal dependencies that need to be accounted for, will be addressed later in this paper.

Metrics

Viewing the accuracy of a model is a common method for evaluating a set of candidate models. This is an effective metric for many research projects, however it is not well suited to evaluate the promise of our models given how our target variable has been structured. With 50% of the observations falling into each class, we'll want to pay closer attention to the number of accurate predictions we make, and for that reason we'll be using the F1 score which provides a balance of evaluating both precision and recall. We are concerned with both components of the F1 score, so we'll leave our value of beta equal to .5.

$$\text{precision} = \frac{tp}{tp + fp},$$

$$\text{recall} = \frac{tp}{tp + fn},$$

$$F_\beta = (1 + \beta^2)\frac{\text{precision} \times \text{recall}}{\beta^2 \text{precision} + \text{recall}}.$$ [2]

II. Analysis

Data Exploration

As mentioned in the *Project Overview*, acquiring clean financial datasets can be costly for retail investors, so the dataset being used for this analysis was collected using publicly available resources. The majority of the data was obtained using the pandas-datareader[3]. Using that API, a series of open, high, low, close, and volume (OHLCV) information was queried for many publicly traded securities. In addition to the OHLCV data that was queried for the assets in this study, the Fama/French Data Library and historical OHLCV data for a variety of sector ETF's were also used to create features. The Fama & French and sector ETF features were constructed via OLS regression, where their return streams were regressed against each assets daily returns over a fixed rolling window of time. This is a common approach used to approximate an assets exposure to a variety of attributes, and is a great proxy for incorporating financial metrics and sector classifications when the underlying data is either too expensive or difficult to otherwise acquire.

Beta to Sectors	
Communication Services (XLC)	Health Care (XLV)
Consumer Discretionary (XLY)	Materials (XLB)
Consumer Staples (XLP)	Industrials (XLI)
Energy (XLE)	Technology (XLK)
Financials (XLF)	Utilities (XLU)

Beta to Fama & French 5 Factor Model
CMA (Conservative Minus Aggressive)
HML (High Minus Low)
MKT-RF (Market Net of Risk-Free Rate)
RMW (Robust Minus Weak)
SMB (Small Minus Big)

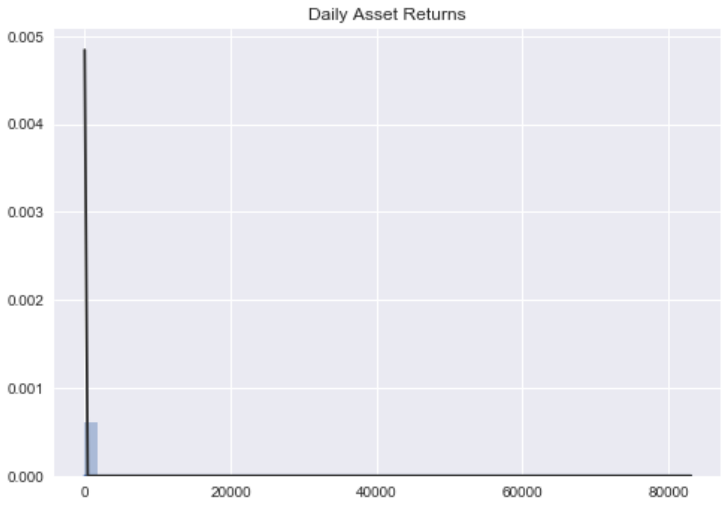
Prior to creating the time-series exposures for these features, we analyzed the distributions of the inputs (e.g. the daily return streams). As shown in the table and plot below, we found a large degree of noise in the daily return streams, which if left untreated would produce unstable coefficients that wouldn't accurately depict the relationship between the assets in our universe and the features listed in the tables above (e.g. sectors and styles). The table and histogram below detail the extreme outliers present in the daily asset returns. Looking to the descriptive statistics to the left, we see a few interesting figures:

1) The minimum return of -100% implies we have companies that went out of business during a single trading period
2) The median company returned 0% day-over-day
3) The maximum one-day return seems implausible for a single trading period to say the least.

Having said that, it's not surprising to see the high degree of noise that is shown below, and it highlights the importance of controlling for outliers prior to building features dependent on these return streams.

Asset Returns:

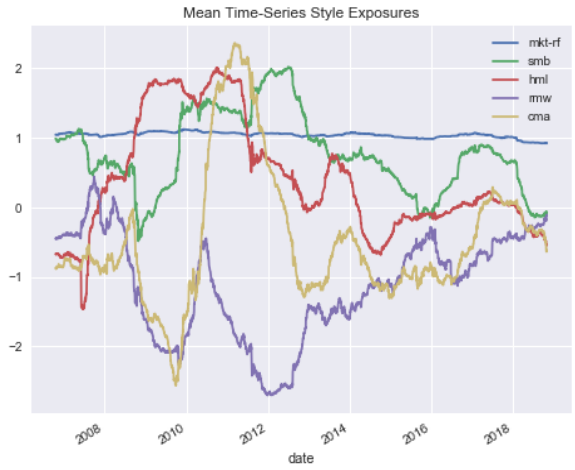
count	1847723.00
mean	0.19
std	81.05
min	-1.00
25%	-0.01
50%	0.00
75%	0.01
max	83063.52



To keep things simple, we decided to evaluate two popular approaches to treat the outliers present in the daily return streams. In the first method, we transformed asset returns into z-scores and excluded figures beyond +/- 3 standard deviations from the mean. In the second, we applied percentile bounds. In that case, we identified an upper and lower bound and clipped returns beyond those thresholds, replacing values higher or lower than the bound with the value for that percentile. We found a greater level of stability when using the percentile bounds relative to the z-score method and ended up going that route to normalize the distribution of asset returns. Following that procedure, we arrived at a cleaner distribution as shown in the figure below.

Once the asset returns were treated for outliers, we moved onto computing our time-series exposures for sectors and styles. The visualizations below illustrate the output of that process by plotting the mean value to those attributes over time. Looking at the plots below, there are a few interesting observations that can be made:

1) Sector betas all began to converge to below 1 beginning in 2012
2) There is a steady, low beta to Utilities which may reflect that sectors' conservative nature
3) The beta to Energy companies oscillates a bit and may reflect volatility in energy prices
4) There are countless thematic cycles present in the style exposures, however we see beta to market pegged at ~1 which resonates with our intuition over a long-time horizon



Finally, technical factors were constructed, which consisted of the features shown below. The TA-Lib[4] library came in very handy when computing these features.

Technical Factors	
Average True Range (ATR)	Percent above and below the 52-week high and low price
Directional Movement Index (DX)	Relative Strength Index (RSI)
Long-Term Momentum: Price change over several horizons	Short-Term Reversal: Price change over several near-term horizons
Money Flow Index	Stochastic Oscillator
Moving Average Signal Convergence Divergence (MACD)	Volatility: Standard deviation of returns over several horizons
Percent above and below the 30 and 200 days moving average price	

Exploratory Visualization

The accompanying Jupyter notebook details several visualizations I relied on to better understand the feature space. Most of the relationships that were surfaced during the exploratory data analysis segment highlighted three things:

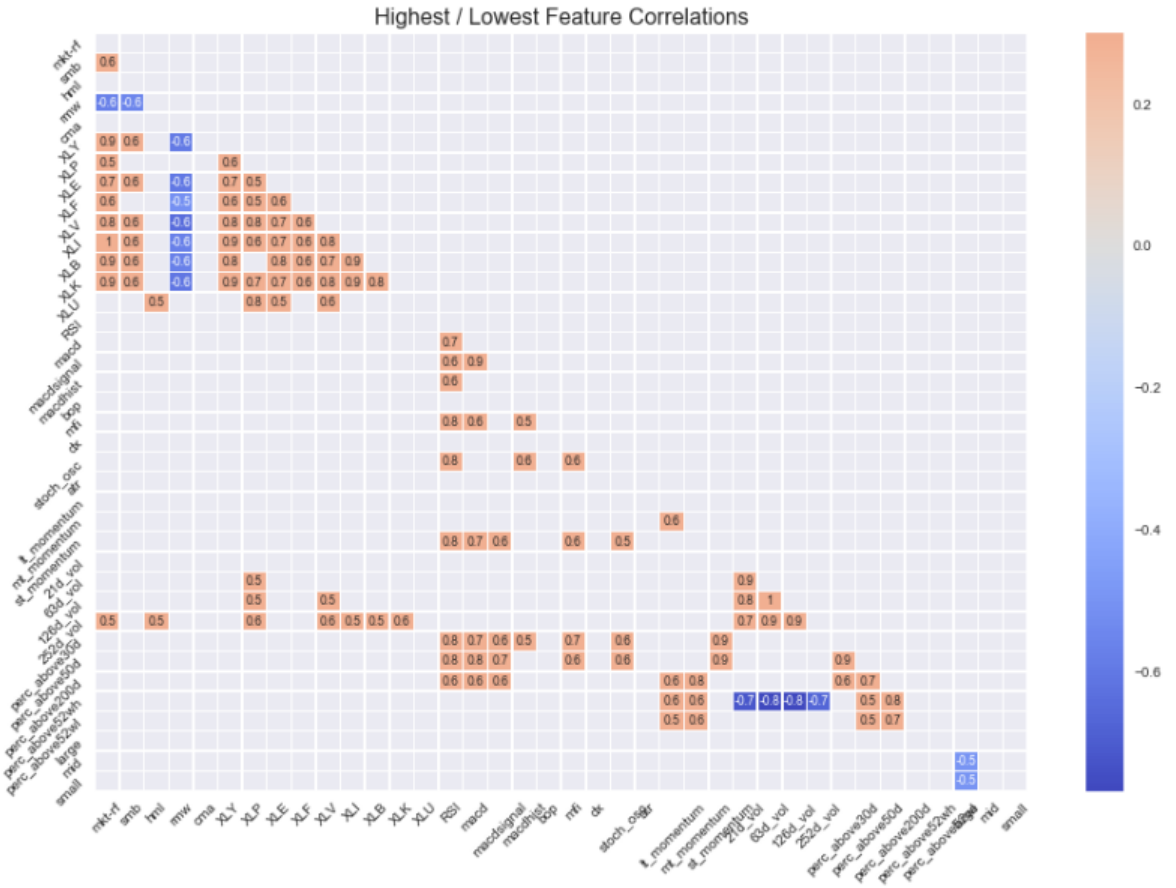
1) Outliers were present and severely influenced many of the features

2) Many features are highly correlated with others
3) Some, but certainly not all, features were best described by a Guassian distribution

These are common shortfalls that we could, and likely should, expect to run into when working with financial time-series. These are addressed later in the analysis by utilizing the interquartile range (IQR) method for treating outliers affecting the feature space, and by log-normally adjusting the following features:

- dx
- 21d_vol
- 63d_vol
- 126d_vol
- 252d_vol
- perc_above52wh
- perc_above52wl

After making those adjustments, I wanted to plot the highest and lowest correlations present within the feature space where 'high' and 'low' were subjectively defined as correlations greater than 50% and less than -50%. What immediately stands out is the high degree of correlation present across sectors (XL*) over the testing period. With the exception of several technical features, we are able to produce a feature space with a fair amount of diversification, however, we see a large degree of concentration stemming from our asset exposures to GICS sectors[1] (see the appendix for more information).



Algorithms and Techniques

Our objective is to predict which companies will outperform and underperform the median company in the universe over the subsequent 63 trading days. Since forecasting continuous return streams can be a bit of a moving target, we decided to create a classification problem by binarizing the returns around the median return of the universe, allowing us to focus on discrete quantities which have a higher probability of being more stationary than their continuous counterparts.

The following algorithms have been selected to solve the classification problem at hand:

1. Logistic Regression
2. Random Forest Classifier
3. AdaBoost Classifier
4. Feed Forward Neural Network
5. Long Short-term (LSTM) Memory Neural Network

Logistic Regression

Logistic regression is a great first model to pursue when solving binary classification problems. With few parameters to tune, it is a simple algorithm to implement and helpful in determining if a more complex model is needed.

The Logistic Regression model will be used to predict the probability that the target variable will be of one class or the other (e.g. zero or one) given the feature space. The following hyperparameters will be tuned to identify the best model:

- The C parameter represents the inverse of regularization strength. Smaller values of C result in regularizing the problem more while larger values place a larger focus on minimizing the cross-entropy of the model.
- L1 & L2 Regularization
    - L1 regularization, commonly referred to as lasso regression, can act as a form of feature selection, shrinking less important features to zero. L1 regularization penalizes the absolute value of the weights.
    - L2 regularization, commonly referred to as ridge regression, minimizes the impact of less important features but does not eliminate them from the algorithm. L2 regularization penalizes the sum of square weights.

Random Forest Classifier

Random Forest classifier is an ensemble algorithm that stems from the decision tree family. This model creates a series decision trees and randomly samples those models based on the training data. This model was chosen over decision trees as the random sampling process often results in a final model that helps produce more accurate and stable predictions.

RandomizedSearchCV[5] from scikit-learn was used to search the hyperparameter space shown below:

- Number of estimators: 10 linear points from 10 to 400
- Max number of features: Square root of the number of features
- Maximum depth: 11 linear points from 10 to 110

- Minimum number samples required to split a node: A list of the following values: 2, 5 ,15, 30
- Minimum number of samples per leaf: A list of the following values: 1, 2, 4

This model was chosen for its flexibility, ease of use, and robustness with respect to combatting overfitting. Additional information on this model and the corresponding hyperparameters can be found here.

AdaBoost Classifier

The AdaBoost classifier is an ensemble method that creates a strong model from numerous weak models. At a high level, this is accomplished by training a model and then creating a second model that attempts to strengthen the first model by correcting errors. These subsequent models are iteratively added to strengthen the initial model until the errors have been eliminated or the maximum number of models has been reached.

RandomizedSearchCV from scikit-learn was used to search the hyperparameter space shown below:

- Number of estimators:
    - 10 linear points from 10 to 400
- Learning rate:
    - 10 linear points from .001 to 1

Feed Forward Neural Network

A feed forward neural network was then chosen as the first of two deep learning models for its ability to capture non-linear relationships present within the data. This model also presents us with a straightforward architecture to work with and relatively low computational complexity when compared to the LSTM we implemented as our final model.

The following parameters were used to design the final architecture for the feed forward neural network:

- 5 hidden layers with 64 units (neurons)
- Dropout probability: 40%
- Epochs: 100
- Batch size: 32
- Activation function: Sigmoid
- Loss function: Binary crossentropy
- Optimizer: Adam

Similar to the Logistic, Random Forest, and AdaBoost models, the feed forward neural network was trained using the TimeSeriesSplit[6] class from scikit-learn.

Long Short-Term Memory (LSTM) Neural Network

The final model chosen for this project was a long short-term memory (LSTM) neural network. LSTM networks are an extension of recurrent neural networks and a popular choice when working with time-series data. This models' ability to map the recent past to the present when making predictions are the

primary advantage I wanted to incorporate and test relative to the feed forward model we discussed earlier.

Below is the architecture I used for this model:

- 4 hidden layers with 50 units (neurons)
- Dropout probability: 40%
- Epochs: 100
- Batch size: 32
- Activation function: Sigmoid
- Loss function: Binary crossentropy
- Optimizer: Adam

As mentioned above for the feed forward model, this model was also trained using the TimeSeriesSplit class.

Benchmark

Since each stock has an equal probability of earning a return that is greater or less than the median value for the period, the appropriate naïve model for this analysis will be to predict that 50% of the companies will fall into each category.

III. Methodology

Data Preprocessing

Several preprocessing steps were performed on the data, many of which were mentioned in the *Exploratory Visualization* section. In total, the following preprocessing steps were carried out on the dataset:

1. Percentile bounds were applied as an upper/lower threshold to treat outliers for each asset's daily returns. This method was applied prior to engineering the time-series features (e.g. sectors & styles).
2. The interquartile range (IQR) method was applied to all features to increase the robustness to outliers.
3. The MinMaxScaler()[7] from scikit-learn was then applied to normalize the feature space prior to being run through the selected algorithms.

Implementation

Before reviewing the implementation of these models, I think it is important to acknowledge the importance a clean data pipeline plays in aiding a successful research process. Practitioners spend the majority of their time collecting, cleaning, and transforming data before machine learning models are ever run, and this project was no different. Up to this point we have performed the following steps in order ensure our data has been collected and treated in a way that makes it suitable for running machine learning models.

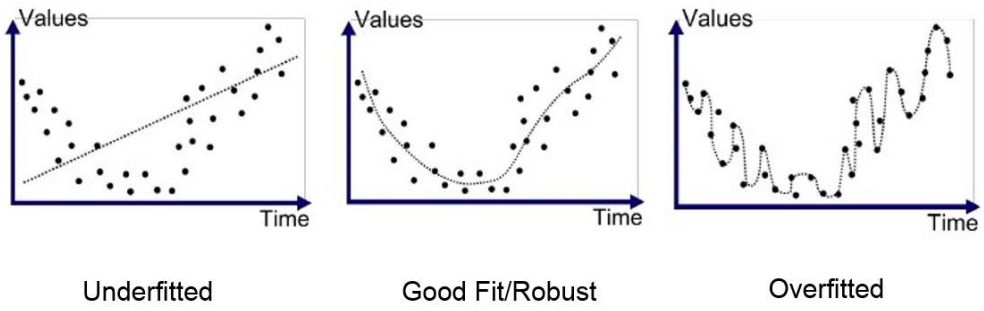
1. Collected and cleaned the dataset
2. Engineered features
3. Explored relationships
4. Identified and treated outliers
5. Normalized distributions where needed
6. Constructed a binary target variable

Now that we've detailed the data processing workflow that has been conducted up to this point, we can focus our attention on the remaining components of the project:

1. Cross-validation
2. Model construction and tuning

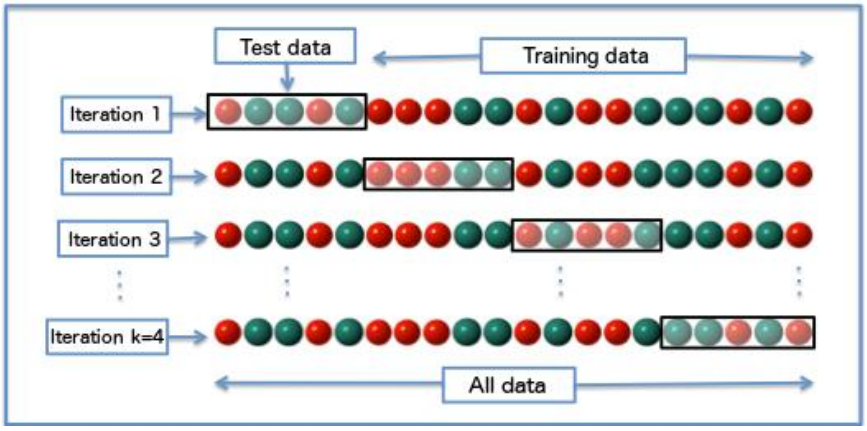
Cross-validation

Cross-validation is an integral component of any machine learning model and a lot of thought needs to be given to how the problem space can be generalized to avoid under and overfitting. The former has the potential to add bias to the results while the latter can introduce variance (link for more information on bias and variance). The image below provides a great illustration of what we're after (in the middle) as well as what we are looking to avoid.



Source: https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76

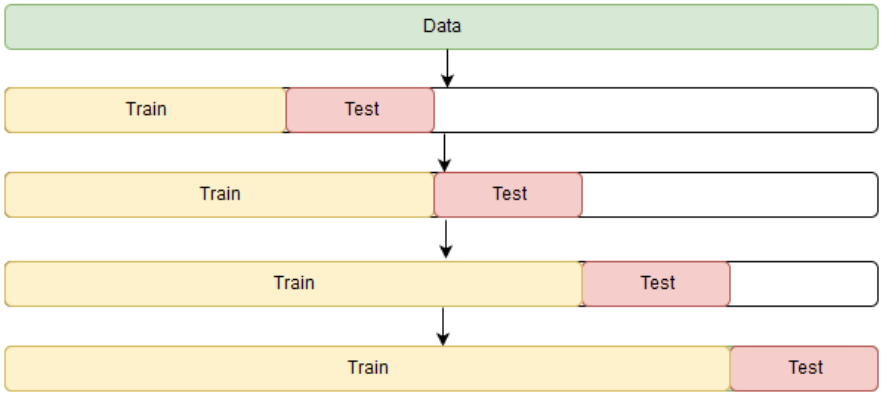
A common approach to cross-validation is k-folds, where our dataset is divided into evenly spaced partitions with one subset held out as the 'test set'. That test data is the out of sample data that will be used to measure the accuracy of the final model being deployed.



Source: https://en.wikipedia.org/wiki/Cross-validation_(statistics)

This technique, while popular, isn't appropriate for our use case as it fails to preserve the temporal features of our dataset. A walk-forward cross-validation approach, like TimeSeriesSplit from scikit-learn, offers us a more appropriate way to cross-validate our model that maintains the path dependent nature of the data. For that reason, we'll be moving forward with that method to train and test our models.

Please reference the image below for an illustration of how that method can be used in practice.



Model Construction & Tuning

As described in the *Algorithms and Techniques* section, the following algorithms have been chosen to forecast future asset returns:

1. Logistic Regression
2. Random Forest Classifier
3. AdaBoost Classifier
4. Feed Forward Neural Network
5. Long Short-term (LSTM) Memory Neural Network

Each model was trained using a proprietary Python module, **FinancialML_utils**, which was built for this project. This library contains a user defined function (UDF) called **BatchWalkForwardCV** takes care of the preprocessing work, model training and evaluation, and generating predictions. The order of operations for training, tuning, evaluating, and forecasting are as follows:

1. Divide the dataset into rolling one-month intervals for training, then use that model to produce predictions for the following year.
2. Utilize a Pipeline[8] from scikit-learn to normalize the data (applying MinMaxScaler) and fit the model.
3. Evaluate a parameter grid to search the parameter space using RandomizedSearchCV.
4. Fit the model using the parameters that resulted in the highest F1 score.
5. Return predictions for each asset and a verbose summary of the models' results.

Refinement

Manipulating the cross-validation window resulted in the largest improvement across all models. The final training and validation split trained each model on a 21-day period and then used the resulting model for the next 264 days before training on another 21-day period. Initially, the cross-validation windows were set to train the models for 6 to 12 months at a time, however given the size of the dataset (> 500 companies per day), the problem became intractable from a computational perspective. Decreasing the size of the training period to 21 days enabled each of these models to run and learn the data while also preserving the markets evolution over time. The most important aspect of setting the rolling windows of time was to ensure the training set was refreshed annually. The size of the window mattered much less in my experience, especially since we are able to work on daily data, a one-month window of time provided each model with an abundant sample size.

In addition to tuning the rolling training and test periods, I found large gains in performance with the Random Forest model when increasing the number of iterations for RandomizedSearchCV from sci-kit learn. Initial values of 5 to 10 iterations resulted in models that rarely produced F1 scores greater than 50%, however increasing those values to above 30, 40, and 50 resulted in much higher scores. I settled on a number of iterations equal to 40 as I felt it was the best balance of performance in terms of both the F1 scores being produced and the time it took to train the model.

IV. Results

Model Evaluation and Validation

Predicting stock returns using machine learning models is quite different when compared to many of the other applications that have been reviewed thus far in our course. Relative to most of the target variables we have predicted up to this point, stock returns are amorphous by nature and as a result have a much lower criteria to constitute success. For example, in an earlier lesson we set out to build an image classification model to predict images of cats. In that scenario, each target variable is a cat, and while they have slight degrees of variability there are a number of constants that do not change over time. By contrast, the stock market can change markedly over time, especially when predicting a continuous variable like forward stock returns. Referring back to the image classification example, a cat is always a cat – that property holds yesterday, it holds today, and we can be reasonably sure it will hold tomorrow, whereas stock returns are more akin to predicting a moving target. Relating this back to the image classification example, stock returns could be a cat yesterday, a dog today, and a bicycle tomorrow. To curb the effects of that analogy, I have decided to binarize the future stock return around

the median return in the universe, assigning stocks outperforming the median a value of one, else zero. Even with assistance by re-engineering the problem, this is still a very complicated problem to solve for so our hurdle rate for success is 50% vs. the much higher percentages we would be looking for with other classical problems like image classification.

With most financial models focused on future stock performance, it is rare to see accuracy scores much greater than 50%. In fact, small gains above 50% can result in much larger performance gains when constructing portfolios. With that, we aren't looking for individual periods with scores of 80-90%, but rather we hope to see persistency above 50% for a prolonged period, which I'm happy to report we do when evaluating the data out of sample.

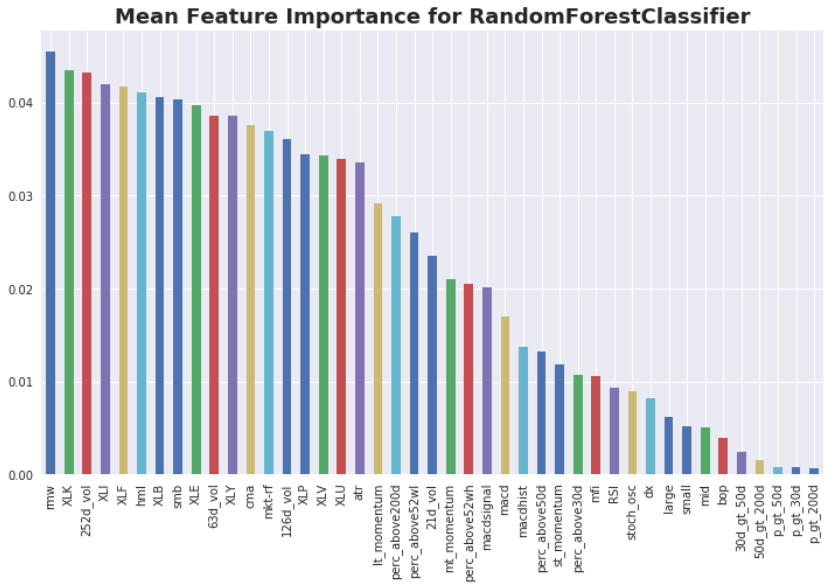
As can be seen in the table below, each of our models outperformed the benchmark of 50%. Given the size of our dataset and computational and monetary expense of running the neural networks, we are going to focus our analysis on the Logistic, Random Forest, and AdaBoost models.

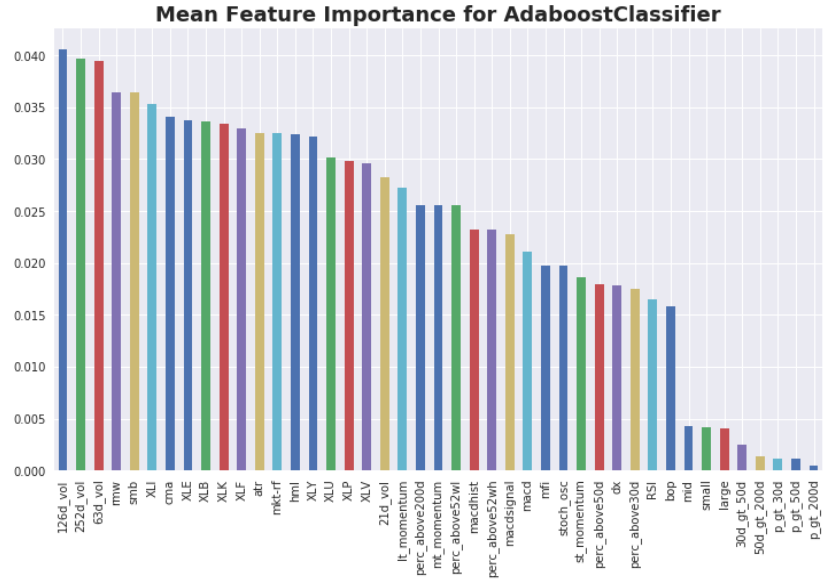
The following can be seen from the table below:

- The Logistic Regression shows promise in individual periods but also the highest variability in its predictions.
- The Random Forest achieves a mean accuracy of 53.1% over a 10-year period. The difference in the F1 scores may seem small, but even a percentage increase over a prolonged period can have dramatic effects on performance, which we will see later.

Period (in years)	Logistic Test Scores	Random Forest Test Scores	AdaBoost Test Scores	Feed Forward Test Scores	LSTM Test Scores
1	38.7%	48.4%	49.6%	49.1%	49.9%
2	56.6%	52.5%	50.1%	55.9%	55.2%
3	52.5%	51.5%	46.7%	50.6%	53.2%
4	58.3%	57.9%	58.2%	54.5%	53.1%
5	45.3%	49.4%	48.3%	45.8%	48.8%
6	50.5%	56.5%	56.6%	52.6%	52.3%
7	55.0%	54.5%	54.5%	52.1%	52.7%
8	55.4%	54.5%	54.4%	54.3%	54.3%
9	37.7%	51.6%	50.6%	52.0%	51.9%
10	55.5%	54.2%	54.6%	52.2%	53.0%
Mean	**50.5%**	**53.1%**	**52.4%**	**51.9%**	**52.4%**
Standard Deviation	**7.1%**	**2.8%**	**3.6%**	**2.8%**	**1.8%**

The feature importance plots generated from the Random Forest and AdaBoost classifiers were also quite telling. The average feature importance for both classifiers were different in order, but generally both placed the most emphasis on sectors, Fama & French style exposures, and volatility. Due to the models' time varying nature, a k-best approach was not taken as that could arbitrarily penalize models in periods where factors with low scores could have had a higher influence on the results for that period.





Each of the five algorithms changes a bit due to a new model being trained each year. I encourage the reader to refer to the corresponding Jupyter notebook to learn more about the specifications of each model over each of the 10 test periods.

Justification

Each of the five models resulted in superior performance relative to the stated benchmark. Most importantly, the models were stable, producing F1 scores greater than 50% in the majority of the out of sample periods. As we will discuss in a later section, those results translate into superior performance when making allocations to securities predicted to outperform the median value of the universe.

IV. Conclusion

Free-Form Visualization

One of the best ways to validate the performance of predictive models is to test how well they work out of sample, and that's exactly what we did for each of our models. The tables below summarize the performance of equally weighted portfolios formed on the out of sample predictions from each of the five models. The out of sample period begins on 2008-02-06 and ends on 2018-07-31. Each model assigns securities in the universe a score of zero, predicting it will underperform the market, and one, predicting it will outperform the market over the subsequent 63 trading days. Securities with common scores are aggregated and held constant, at an equal weight, for 63 days at which point the performance and other statistics are measured to evaluate the efficacy of the models' predictions.

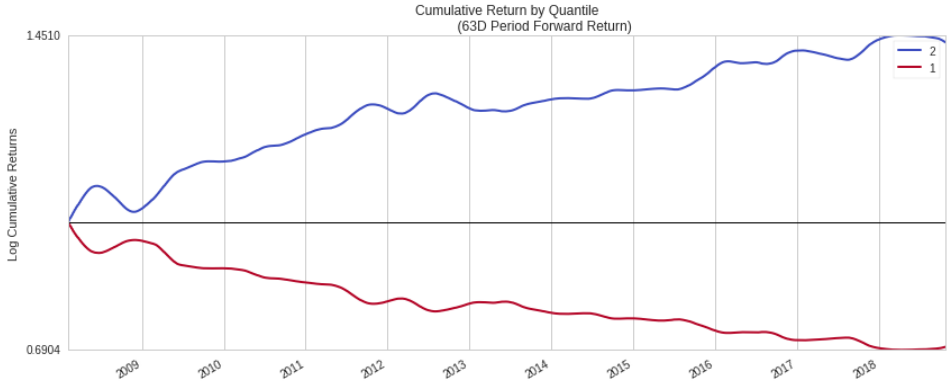
The *Returns Analysis* table measures abnormal market returns. Said differently, we are measuring the performance earned/loss above and beyond that of the market return. As an example, let's turn our attention to the best performing model, the Random Forest classifier. That model produces an annualized return of 4.4% above the market over the 10-year horizon from 2008 – 2018. In addition to outperforming the market, it's accomplishing that feat with minimal beta to market, which mutes the effect any swings in the market will have on our model portfolios performance.

Returns Analysis	RandomForest_63D	AdaBoost_63D	FeedForwardNN_63D	LSTM_63D	Logit63D
Ann Alpha	0.044	0.04	0.036	0.023	0.019
Beta	0.041	0.037	0.032	0.112	0.102
Mean top quantile ret (bps)	42.537	38.198	36.586	29.765	17.615
Mean bottom quantile ret (bps)	-38.219	-35.856	-29.555	-32.612	-35.095
Mean spread ret (bps)	80.757	74.054	66.141	62.377	52.71

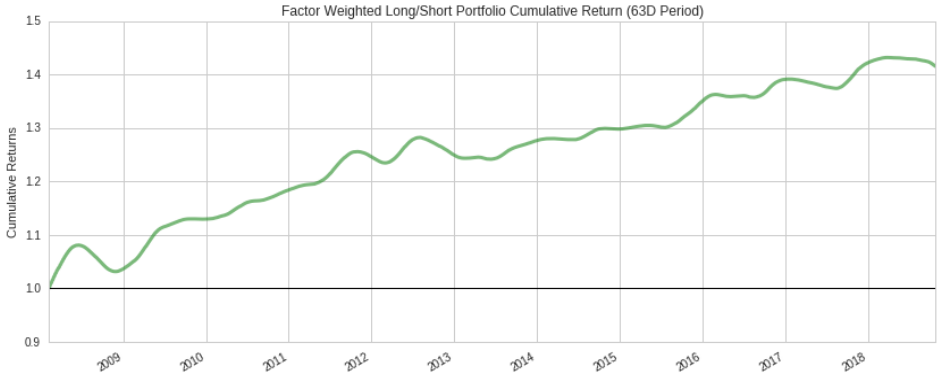
The information coefficient (IC) is a common metric used to gauge the predictive strength of a factor. The IC represents the ranked correlation between the signal and the corresponding future 63-day return. Here we see the Random Forest model produces the highest IC at 6.5%. Each model produced an impressive IC over a 10 year stretch – a good benchmark value would be a statistically significant IC between 3.5% and 4%, which we see the majority of our models surpass.

Information Analysis	RandomForest_63D	AdaBoost_63D	LSTM_63D	FeedForwardNN_63D	Logit63D
IC Mean	0.065	0.06	0.047	0.042	0.037
IC Std	0.184	0.196	0.142	0.171	0.156
Risk-Adj IC	0.353	0.307	0.331	0.246	0.235
t-stat (IC)	18.146	15.752	17.011	12.651	12.069
p-value (IC)	0	0	0	0	0
IC Skew	0.887	0.67	0.482	-0.009	0.198
IC Kurtosis	1.167	0.55	0.751	-0.171	0.073

The plot below displays the performance of the two portfolios formed by the Random Forest classifier's predictions. Securities forecasted to outperform the market were placed into the second quantile and securities forecasted to underperform were placed into the first.



The next plot displays the difference in return and is commonly referred to as the 'spread return'. This return is reflective of a long/short investor who takes a long position in the securities forecasted to outperform and a short position in the securities forecasted to underperform. Each security is then assigned an equal weight and held for the next 63-days, at which point the process is repeated.



Reflection

The process applied for this project can be summarized using the following steps:

1. A hypothesis was formed, and the requisite inputs were listed to support testing that hypothesis
2. A dataset was created, transformed, and cleaned using publicly available resources
3. Features were engineered, explored, and treated for outliers
4. A benchmark model was specified
5. A target variable was identified and constructed
6. Multiple classifiers were trained and tuned until optimal settings were found
7. Each model was evaluated both in and out of sample
8. Out of sample predictions were saved and explored to evaluate their efficacy

I found steps 2, 3, and 6 to be the most challenging aspects of this project. Steps 2 and 3 required more time and care than I had initially anticipated. The time it took to procure a custom dataset and engineer features over a 10-year period was, at times, very time consuming. In addition to the time it took to create the dataset, a lot of care had to be taken to ensure the dataset was accurate over a number of rolling periods. Throughout this process, there were a number of times where features had to be completely recreated to ensure they aligned with the right asset and time period.

Step 6 was difficult from a computational perspective. Training a model on a dense dataset like this was expensive and time consuming. Each of the five models were trained using virtual machines from AWS, and even then, the training times were substantial, especially I moved onto making predictions with neural networks. The neural networks took more than 10 hours at a time to train, and for that reason I ended up implementing architectures that were less robust than I would have liked. With more time and budget on AWS, I would have liked to create more intricate models. While the neural networks could have been improved upon, I enjoyed how much I was able to learn about building different architectures for time-series problems like the one I decided to solve for.

I found the importance of feature engineering and structuring the cross-validation model to be the most surprising aspect of this project. The features included in this project and the frequent training of new models resulted in the largest gains in performance as measured by the F1 score. Tuning the hyperparameters of the various models had a large affect as well, but those were expected more than the changes in the features and cross-validation approaches.

I am very pleased with the final version of the models I implemented. I am confident each model could be improved upon, but each shows promise in predicting the cross-section of asset returns over a 63-day horizon. More than that, each model shows persistency in making predictions with a reasonable amount of variability in the F1 score. In addition to the persistent F1 scores, these models produce statistically significant returns out of sample.

Improvement

There are several improvements that could be made to potentially enhance the performance of these models. Some of these include:

- **Variance/Bias Trade-off:** The Random Forest, AdaBoost, Feed Forward, and LSTM models all suffered from a high degree of variance. The final models could have been structured differently to generalize the problem space more.
- **Cross-validation:** Our models are being trained once a year for a month at a time before making predictions for the next 264 days. This is most likely suboptimal and could be improved upon by increasing the frequency of training new models and shortening the amount of time a model is used. Given more computational resources, I would like to experiment with training new models each quarter on a rolling three-year window.
- **Features:** This analysis uses a self-made dataset with freely available data. I have little doubt that using institutional quality data would not only save a massive amount of time, but it would also add to the quality of the final model.
- **Creating Labels:** I used a basic assumption to create binary labels for each security. There are several other approaches that may result in a more robust model. A great resource that explores a number of these methods can be found in Marcos Lopez de Prado's book: Advances in Financial Machine Learning.

Appendix:

1 It is widely cited throughout financial literature that asset allocation makes up a meaningful percentage of portfolio performance, and while the actual percentage has been debated, the impact of a robust asset allocation plan has not. Sector allocations are a large part of many investors asset allocation plan and for that reason I have decided to leave those features in the analysis despite their seemingly high correlations with one another. Another way to explore those relationships while dampening the correlations would be to generate their return streams net of market. That goes a bit beyond the scope of this assignment but could prove to be a worthwhile modification.

References:

1 Fama/French Data Library
2 https://scikit-learn.org/stable/modules/model_evaluation.html#precision-recall-f-measure-metrics
3 https://pandas-datareader.readthedocs.io/en/latest/
4 TA-Lib
5 https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
6 https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html
7 https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html
8 https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

Helpful Articles:

Logistic Regression:

- https://medium.com/greyatom/logistic-regression-89e496433063
- https://medium.com/datadriveninvestor/l1-l2-regularization-7f1b4fe948f2
- https://simple.wikipedia.org/wiki/Logistic_Regression
- https://machinelearningmastery.com/logistic-regression-for-machine-learning/
- https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c

Random Forest:

- https://www.datacamp.com/community/tutorials/random-forests-classifier-python#algorithm
- https://medium.com/machine-learning-101/chapter-5-random-forest-classifier-56dc7425c3e1

AdaBoost:

- https://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning

Feed Forward Neural Network:

- https://www.learnopencv.com/understanding-feedforward-neural-networks/

Long Short-Term Memory (LSTM) Neural Network:

- https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5