Darian Bailey
6/22/2016

Predicting Daily Direction of Futures Markets from Price Action Using ML Techniques

Definition

Project Overview

The derivatives market is estimated at $1.2 quadrillion, 10 times the size of the world GDP (How Big is the Derivatives Market | Investopedia). A good portion of the derivatives market takes the form of futures contracts, an instrument that guarantees delivery of an asset in the future. Futures markets can be shorted and bought and they allow investors to lever their accounts and trade with more money than they have available. An entire industry is based off of building models that can accurately predict when to buy and sell futures contracts. In this project we attempted to build several of these models.

In this project we create several models to predict the direction of several equity futures markets. We use daily price action data[1] taken from CQG, a paid service, and make our prediction after the open of each day. We test several methods for model creation and try to beat a benchmark on a test sample. Finally we construct rudimentary equity curves using test sample predictions of several of the better models.

Problem Statement

Futures markets are "auction market[s] in which participants buy and sell commodity and futures contracts for delivery on a specified future date." (Investopedia - Futures Market Definition, 2003) In this project we create a system that can predict the direction of days in several futures market (i.e. whether the first trades of the day were below or above the price of the final trades of the day) based only on daily historical price data. Our goal in this project was not to create a trade system, which would require much more detailed input data and risk protection, but rather create a basic system to test the viability of using various machine-learning techniques to develop such a trading platform on such limited data. At the end of the project we wanted to be able to recommend what type of learning techniques would be worth building into a trading system.

The basic system we create takes the data we have available, historical price data, and outputs binary labels that specify whether the day is up or down (1 or 0). This is a supervised time series classification problem so we will be using classifiers to predict the classification of current days based on historical data. We train using a train set composed of days in the past, and test using a test set composed of more recent days.

[1] Only the market's open, high, low, close and volume

Metrics

To measure the performance of our learning techniques we used the accuracy_score() method found in sklearn (Sklearn.metrics.accuracy_score) on a test sample kept apart from our validation data. The test sample was the last 400 days of each market. The accuracy score is the number of correctly guessed days divided by the total number of days.

After determining the best performing models by accuracy we construct a rudimentary trading simulation and determine how the models would perform in each market if they were trading at full size using 1000 dollars in a fixed account (no compounding).

Analysis

Data Exploration

Though good intraday futures data is often hard to find cheaply, daily data is widely available.[2] We use daily data from CQG in the following markets:

- EP: Emini S&P 500
- EMD: EMINI SP MIDCAP 400
- ENQ: IMM EMINI NSDQ
- YM: CBT MINI DOW 5
- TFE: NYF SM RUSS2K

Each day in each market has the following features:

- Date: the Date
- Market: the Market the day occurs in
- Open: the average price of the first trades of a day
- High: the highest price of an executed trade on a day
- Low: the lowest price of an executed trade on a day
- Close: the average price of the last trades of a day
- Volume: the amount of contracts traded on a day

In this project we build a system that makes predictions of sign of the change (Close-Open) on a given day in a given market. We want to make the prediction shortly after the open so we have the following information available on each day:

- Date: the Date
- Market: the Market the day occurs in
- Today's Open
- All previous Highs
- All previous Lows
- All previous Closes

[2] A good resource is https://www.quandl.com/collections/futures

- All previous daily Volumes

Before we start exploring data and training we need to do some data cleaning. Two steps are required for every model.

Rolling: Futures data is divided into overlapping contracts which last about 1-6 months depending on when they expire. There is often a difference in prices of these contracts even though they share the same underlying commodity. For example a contract of crude oil scheduled for July delivery will trade at a different price than a contract scheduled for august delivery. The first thing we need to do to get data in a usable format is form one continuous contract. There are many ways to do this. We knit them together by backward adjusting[3]. This is in a custom application outside python and the scope of this project.

Creating a Master Index: Since we analyzed multiple futures markets it is convenient to have all days line up on one index. We forward fill our data when markets don't trade. More information on the method can be found in preprocessor.py

Exploratory Visualization

Once the data is clean we can display it. Here are the first 5 days of Emini SP Midcap:

	Market	DayIndex	Open	High	Low	Close	Volume
0	EMD	0.0	715.2	724.7	706.2	722.7	36862
1	EMD	1.0	724.3	729.0	722.4	728.6	22599
2	EMD	2.0	728.0	729.9	725.3	728.1	20060
3	EMD	3.0	732.6	737.5	729.0	737.1	19130
4	EMD	4.0	737.8	744.4	737.6	742.5	17069

We can also look at summary statistics for the data.
Here are the means and variances for all the data:
Means:
Open 3074.006777
High 3095.190089
Low 3050.677860
Close 3074.081011
Volume 397231.730652

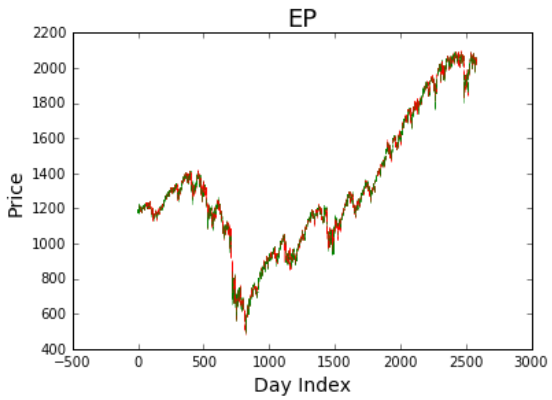
Variances:
Open 1.572036e+07

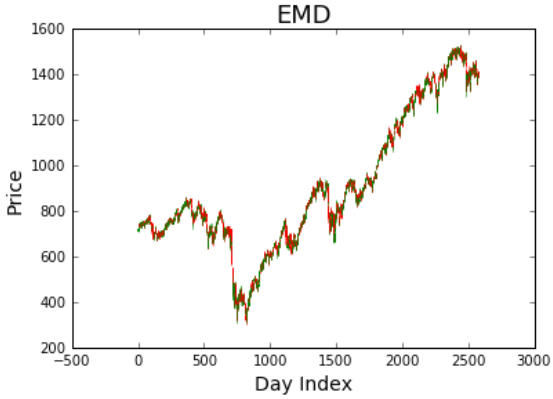
[3] A description of how to backward adjust data can be found here http://www.premiumdata.net/support/futurescontinuous2.php

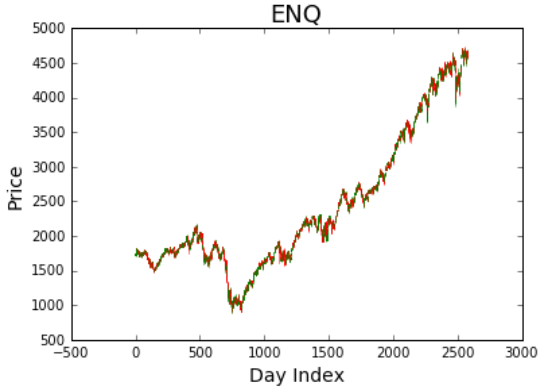
High 1.589596e+07
Low 1.552909e+07
Close 1.572104e+07
Volume 3.693845e+11

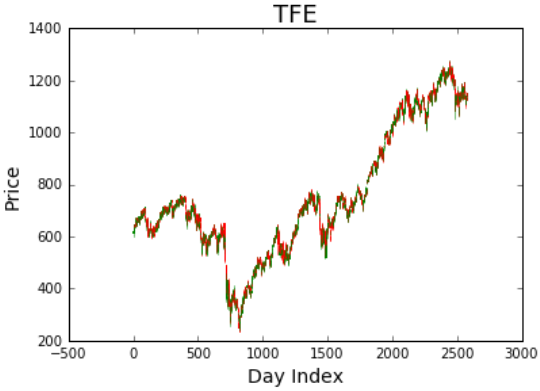
In addition to putting data in a table we chart it in both bar chart and histogram format in order to determine what other preprocessing might be necessary.

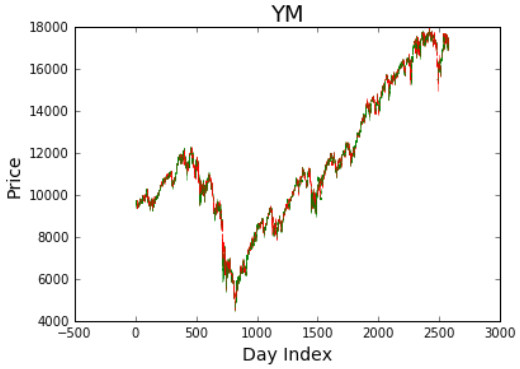
Candlestick Charts:







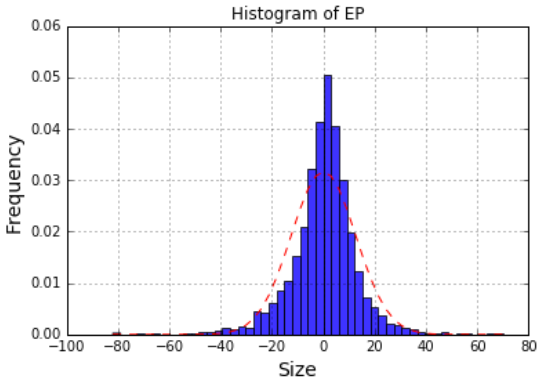


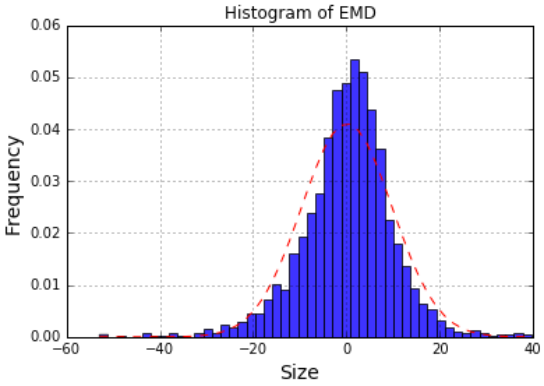


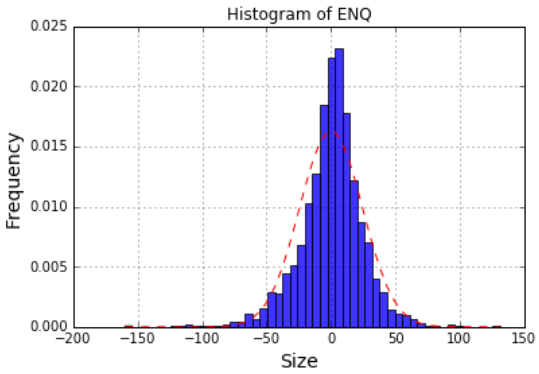
The candlestick charts are a standard way of displaying market data. On the y-axis we have the price of the contract and on the x-axis we see the day index we

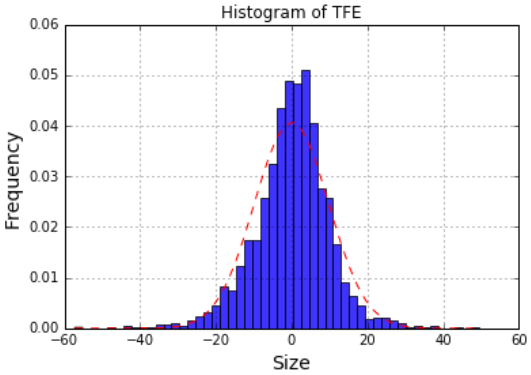
made in the data-cleaning step. They are green if the close is above the open and red if the close is below the open. We are trying to predict the color of the candle in our project. From the charts we can see that green and red candles are evenly distributed through the data set. To get a better idea of the distribution of the candles we can graph the change on each day in a histogram.

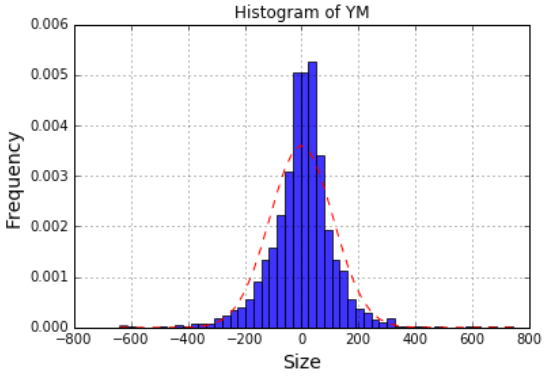
Histograms:











Histograms are useful in this problem because we can see a visual representation of the distribution of daily changes. We are trying to predict which days have a change less than 0 and which days have a change greater than zero. From the histogram we can see the mean of each market is slightly above 0. Thus we know immediately that predicting every day has a positive change will have accuracy higher than 50%. In addition the histograms have high kurtosis and no skew. Benoit Madelbrot suggests that they follow a Cauchy distribution (Mandelbrot 2004).

The charts do not show evident correlation between historical data and the direction of the current day. So we know from the visualization that it is necessary to engineer features for the data set. This project's feature engineering is described in the "Data Preprocessing" section of this report.

Algorithms and Techniques

This is a time series classification problem and we have the following methods evidently available:

- Decision Trees
- SVMs
- Neural Nets

To start we compare decision trees and SVMs. Decision Trees or their ensemble equivalent, random forests, should make sense for this problem as intersections of features in certain ranges may lead to specific labels. For example the market may have a down day after a large upward change followed by a large upward gap greater than a certain value. Traders often have specific levels they will buy and sell at. SVM's, or SVM's with grid search, make sense if we are able to use a hyper plane to divide the data more directly using different kernels.

Once we have compare SVMs and Decision trees we have the option of performing principle component analysis to scale the data. This will matter for SVM's in particular as they need to have equal sensitivity to all inputs.

Finally we can use the universal function approximation of the neural net. Neural nets are state of the art for almost every machine learning task and hypothetically we should be able to perform better on this task using a neural net. However architecture matters a great deal as does training time. We have the option to explore several complex neural network structures, but the scope of this project only covers simple nets that have 1-3 layers, drop out and other regularizations.

Benchmark

We divide the data into 3 sets: a training set, a validation set which consist of the penultimate 400 days and the test set which was the final 400 days. We only touch the test set to determine whether or not our algorithm beat the both benchmarks. This will satisfy the robustness requirements of the project:

-A weak benchmark: the algorithm has above 50% accuracy in the test sample

-a stronger benchmark,: the algorithm's trading simulation produces more PNL than a simple long only strategy in the test sample.

We consider several factors when we set our benchmarks. First investors can go long or short in a futures market, so from that perspective futures are a zero sum game. Any algorithm that attains more than 50% accuracy would be classified as beating a benchmark. However in the exploratory visualization section we see an historical upward trend. So we added a second stage to our benchmark; one that measures dollar performance of a trading strategy using our model against a historically good strategy.

Methodology

Data Preprocessing

(All code described in this section can be found in preprocessor.py and each model's file)

We perform two stages of preprocessing: A general stage, where we used the same techniques for every model (feature creation, Train test split), and a model specific stage that uses different techniques depending on the model (PCA, whitening, dummy variable creation).

In the general stage we remove 400 days from the end of each market to act as a test set. We then cut an additional 400 days to serve as a validation set. With these sets we create many features that summarized historical data: volatility measures, distances of opens from moving averages, and distances of opens from previous closes, and others. Here is the list of features and their purposes:

- Yesterday's True Range: a measure of yesterday's volatility. It is max(day before yesterday's close, yesterday's high)- min(day before yesterday's close, yesterday's low)
- ATR10: a measure of the environmental volatility. It is the average of the previous 10 true ranges
- Moving Average X: Checks for mean reverting phenomena and includes data from previous days in current day. Shows where the price is in relation to the moving average of the last X days; it is scaled by dividing by ATR10.
- Gap: a summary of yesterdays close and todays open. It is the distance of today's open from yesterday's close scaled by ATR10.

We also create a binary label that stored whether the day was up or down to test our models.

In the model specific stage we mainly format the data for to allow for model implementation. We also train different models either individually on markets or on all markets together depending on previous accuracy of similar models with each training method. We create 2 data sets using the features: one of scaled features only and one of all features. This is important for SVMs because we want to run them on a scaled dataset so the models overly influenced by certain features. Finally we perform PCA on the datasets for certain models on both datasets. This is another way to feature scale for SVMs. Neural nets may benefit from this as well, but in

theory they should be able to capture principle components more effectively in their higher levels. PCA may just remove useful data. ZCA would be a solution to this, but is outside the scope of this project.

Implementation and Refinement

(Code implementation can be found in main.ipynb or main.html)

We implement SVCs, decision trees and neural nets in this project. Each implementation is trained on a train set and validated on a validation set. The best performing models are then trained on the train and validation sets together and benchmarked using the test set. We start by training the default instances of SVC and decision trees from sklearn on both normalized features only and all the features together. We train the instances on all the markets together and each market individually to see if signals persisted across markets. SVCs perform better than decision trees in every permutation.

Because the results skew heavily in favor of the SVC's we stop investigating decision trees and concentrate our efforts on SVCs. We run several grid searches on complexity and kernel types (C=[1,2,3,5,7,9,15], kernel =[rbf, linear]). The computing power available is unfortunately only able to calculate RBF and any complexity greater than 3 is found to over fit.[4] We train models on markets individually and all markets together. Although we find it does not make a difference for SVC's. We also run an SVC with PCA grid search. Many of our SVC's received the same high accuracy on the validation set: .567082294264. This was our highest score and it occurred on SVCs with and without market batching and with and without PCA; SVC's run with both datasets (normalized and all) achieve this score.

After testing SVC's and Decision trees we move on to neural networks. We decide not to batch data because SVC's are able to achieve high scores when they train on all markets together. According to Stanford's cs231 (Stanford), non-convolutional neural nets do not benefit from more than 3 layers. We test neural nets with 2 and 3 layers with and without drop out and different activation functions. We also attempt performing PCA on the data using the optimal number of features found in the SVC grid search, but as expected that method only served to remove useful data. We then attempt PCA using the same number of features as found in our full set and the neural nets achieved better accuracy. SVC's receive the highest accuracy of any technique.

[4] These complexity tests are not in main.ipynb as they are not particularly valuable. To test higher complexity, you can alter the tuned_parameters variable in line 50 of svm.py.

Accuracy scores can be found in the table below:

Technique	Data Set	Batched Data?	Train Accuracy	Validation Accuracy
Decision Tree	Full	No	0.9881	0.5451
Decision Tree	Full	Yes	0.9814	0.4847
SVC	Full	No	1	0.5671
SVC	Full	Yes	1	0.5671
Decision Tree	Scaled	No	0.9875	0.4953
Decision Tree	Scaled	Yes	0.9842	0.4848
SVC	Scaled	No	0.8536	**0.5671**
SVC	Scaled	Yes	0.8780	**0.5671**
Gridsearch SVC	Full	Yes	1.0	**0.5671**
SVC with PCA	Full	No	0.5587	0.5526
SVC with PCA	Full	Yes	0.6096	0.5536
SVC with PCA	Scaled	No	0.5340	0.5341
SVC with PCA	Scaled	Yes	0.5942	0.5406
GridSearch SVC with PCA	Full	No	1.0	**0.5671**
2 Layer NN	Full	No	.5261	0.5667
PCA 2 Layer NN	Full	No	0.5170	0.5147
PCA same number of features with 2 layer NN	Full	No	0.5257	0.5566
3 layer NN	Full	No	0.5322	0.5177
3 Layer NN with tanh activation	Full	No	0.5340	0.5197
3 layer NN with dropout	Full	No	0.5340	0.5182
3 layer NN with tanh and dropout	Full	No	0.5339	0.5197

Results

Model Evaluation and Validation and Justification

Based on the previous table we decide to use an SVC with Grid search and PCA for our final model. We could have chosen any of the models in with results in bold, but we pick that model because it uses the full dataset and non-batched data so it would be the easiest to implement in other futures markets. The final model has an RBF kernel because the linear system does not calculate in a feasible time on the computing power available. To test the robustness of the model we turn to the test set. Unlike the validation set the test set has not been fit to or examined at all by any of the models. It is a wholly different data set than we've used up until this point. It is a good measure for robustness because for the test set we will train on the union of the train and validation data and we will test the model on the untouched test set.

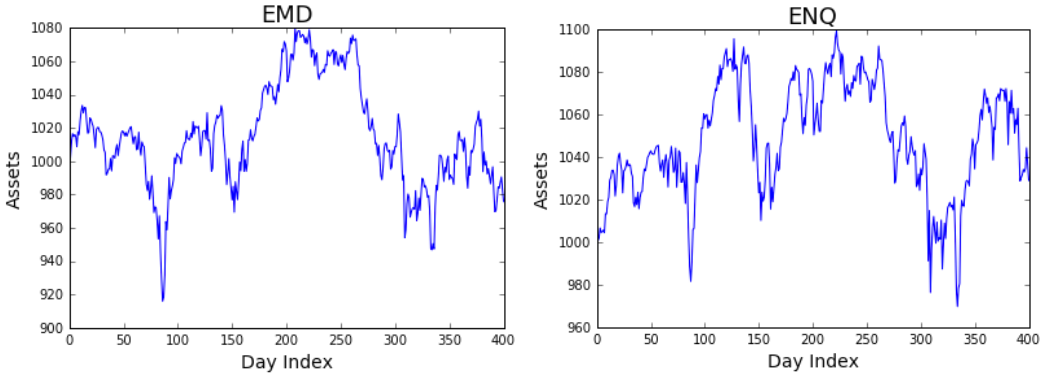
When we test our chosen model on the test data after training on the union of train and validation it received a score of 0.5197. It beats the first benchmark, but not by a significant margin. For the second benchmark it receives the same amount of dollars on the equity curve as simply going long, average pnl of 1.7059 across all markets. The model does not beat the second benchmark.

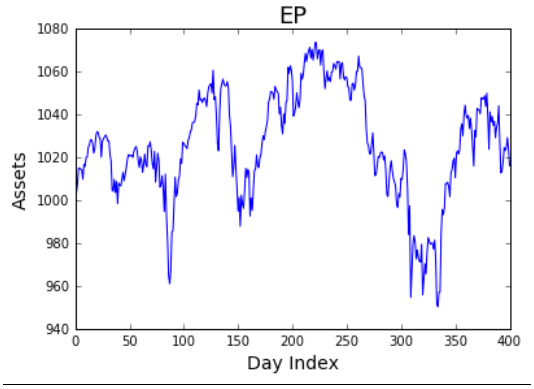
Overall the model predicts a good strategy for equities (going long), but it failed to generate any better trading ideas from the data available. It is not significant enough to solve the problems. Ideas for improvement can be found in the "improvement" section.

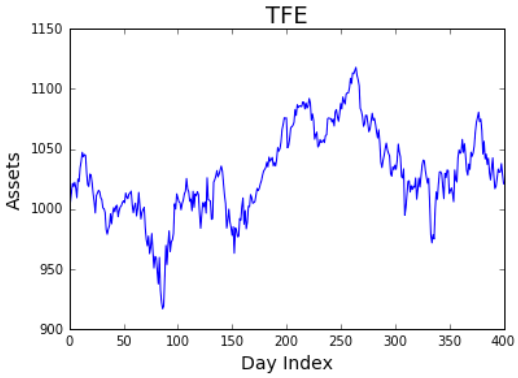
Conclusion

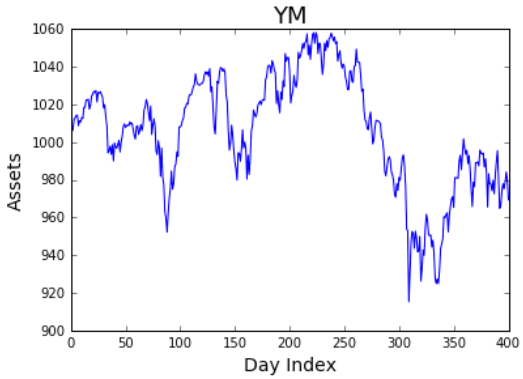
Free Form Visualization

In the below we can see the equity curves for the final model on the test sample. Our strategy does not scale with volatility as a better system might. Thus we see the effect of either going long or going short for a full day. This is sometimes very volatile as we can see on day 75 and other times there is no movement. If we examine the equity curves closely we can see they are the same as simply going long (buying every day) as both strategies generate an average pnl of 1.7059 across all markets. We graph going long every day in main.ipynb and doing so also generates the following graphs:

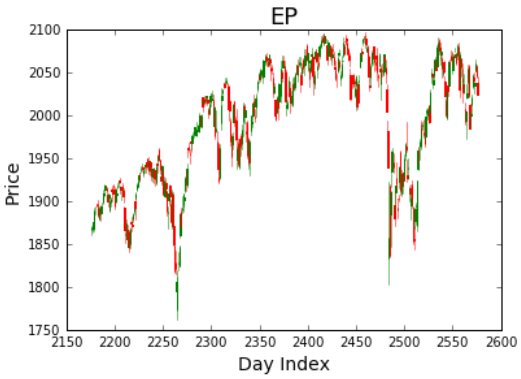


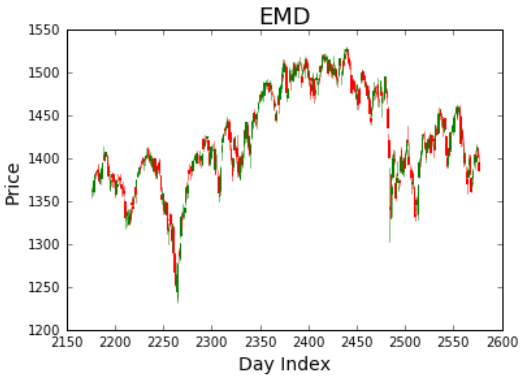


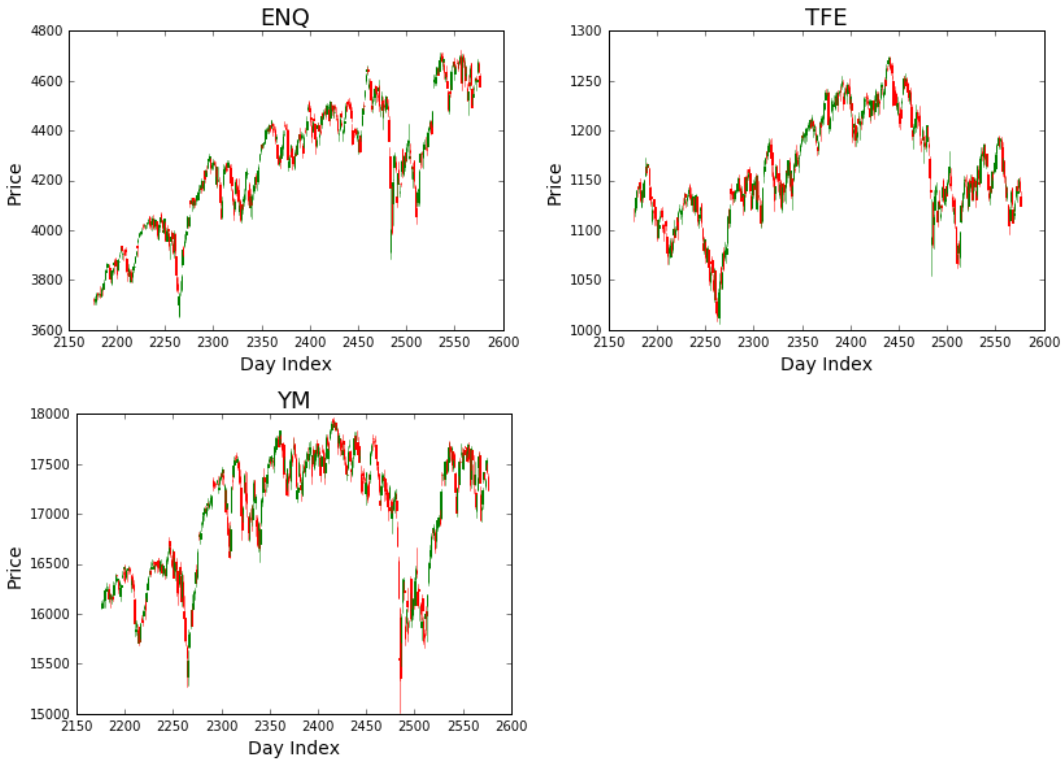




In the above graphs we assume that the strategy has 1000 dollars to invest every day. We tracked the performance over time and plotted it in each market. Visualizations for neural nets can be found in main.ipynb and main.html as well. We can see below that they track the performance of the markets themselves:







The only difference in the movement of the equity curves and the markets themselves is that the equity curves ignore gaps between the close and the open. What this means is despite all the engineered features and grid searches, our chosen model decided that the best strategy was just to go long.

Reflection

We started this project by performing a good amount of cleaning. It was a difficult process given the idiosyncrasies of futures data. We also created several features in an attempt to extract more useful signals. We found this a particularly interesting aspect of the project as we were attempting to summarize historical movement using only one or two numbers. For those features we calculated distance from moving averages, moving averages of volatility and distances from previous closes scaled by volatility.

After feature engineering we tried several techniques to predict change on the day. SVC's, neural nets and Decision trees were all tested. The final model was chosen based on its performance on a validation set. Finally we tested the final model on the test set. It exceeded the first benchmark by scoring 0.5197, but failed to perform better than simply going long as both strategies yielded a pnl of 1.7059 .

Obviously we were disappointed in the results, but they made sense given the data available. Successful trading algorithms have many more natural features available (fundamentals etc.). In addition the models we tested mainly looked at the data from the previous day, historical data before the previous day was included only in the form of moving averages of volatility and price.

We performed an exhaustive search of machine learning techniques available, but failed to find anything useful because of the limited data. Recurrent neural nets may have yielded some results. More information on that is in the "improvements" section. Many of the models, including our final choice overfit to training data.

The models in their current states should not be used to solve these problems.

Improvement

We were unable to generate a good model because of the data available. To improve on this project we could do any of the following 3 steps: Add more engineered features, add fundamental features such as average price to earnings, or use recurrent neural nets such as LSTMs to extrapolate from historical trends.

If we wanted to add more engineered features we could track how many days up or down out of a sequence of days have occurred, or we could use any of the technical indicators widely used by traders.[5] We don't believe that we would see very much improvement from that given the poor performance of the current model.

If we wanted to add more natural features we could add the average price to earnings, the sector makeup of the companies on each index, or any other number of features. The simplest way to add features may be to feed all the market data in together so models can use the price movement of the S & P to predict movement of the Dow similar to statistical arbitrage.

Finally using a recurrent neural net such as LSTM may yield better results. It would be equivalent to feeding in all historical data into a model at once.

[5] Many of these can be found on investopedia.com

Citations

"Convolutional Neural Networks for Visual Recognition." Stanford, n.d. Web.
"Futures Market Definition | Investopedia." *Investopedia*. N.p., 23 Nov. 2003. Web. 22 June 2016.
"How Big Is the Derivatives Market? | Investopedia." *Investopedia*. N.p., 27 May 2015. Web. 03 July 2016.
Mandelbrot, Benoit B., and Richard L. Hudson. *The (mis)behavior of Markets: A Fractal View of Risk, Ruin, and Reward*. New York: Published by Basic, 2004. Print.
"Sklearn.metrics.accuracy_score¶." *Sklearn.metrics.accuracy_score — Scikit-learn 0.17.1 Documentation*. N.p., n.d. Web. 22 June 2016.