Darian Bailey
6/22/2016

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

Definition

Project Overview

In this project we created several models to predict the direction of several equity futures markets. We used daily price action data[1] and made our prediction after the open of each day. We tested several methods for model creation and tried to beat a benchmark on a test sample. Finally we constructed 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 aimed to 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 to test the viability of using various machine-learning techniques to develop such a system 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.

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 constructed a rudimentary trading simulation and determined how the models would perform in each market if they were traded at full size using 1000 dollars in a fixed account (no compounding).

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

Analysis

Data Exploration

Though good intraday futures data is often hard to find cheaply, daily data is widely available.[2] We used 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 had 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 built a system that made predictions of sign of the change (Close-Open) on a given day in a given market. We wanted to make the prediction shortly after the open so we had 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
- All previous daily Volumes

Before we started exploring data and training we needed to do some data cleaning. Two steps were 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 needed to do to get data in a usable format is form one continuous contract. There are many ways to do this. We knitted them together by backward adjusting[3]. This was in a custom application outside python and the scope of this project.

---

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

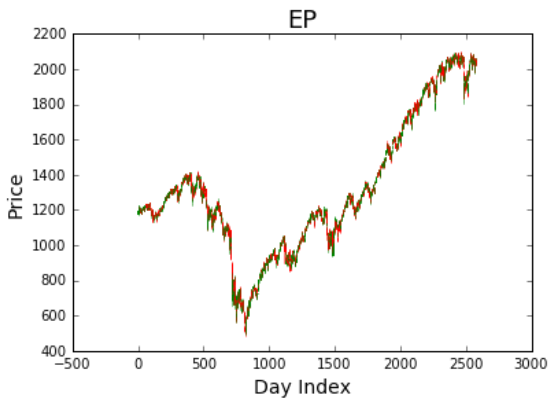
[3] A description of how to backward adjust data can be found here http://www.premiumdata.net/support/futurescontinuous2.php

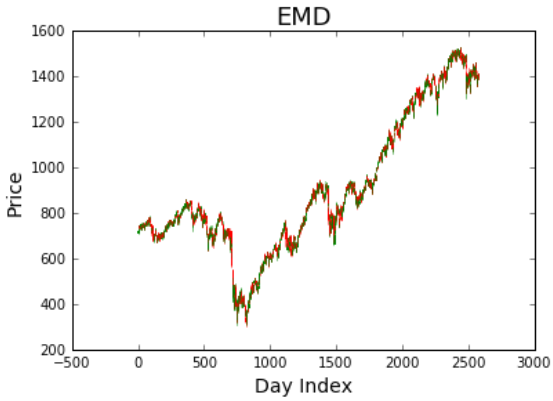
Creating a Master Index: Since we analyzed multiple futures markets it is convenient to have all days line up on one index. We forward fill our data when markets don't trade. More information on the method can be found in preprocessor.py

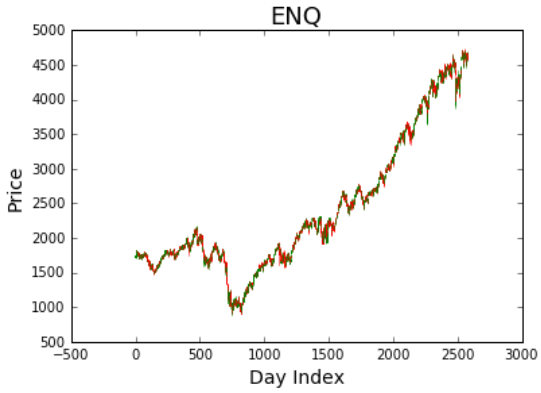
Exploratory Visualization

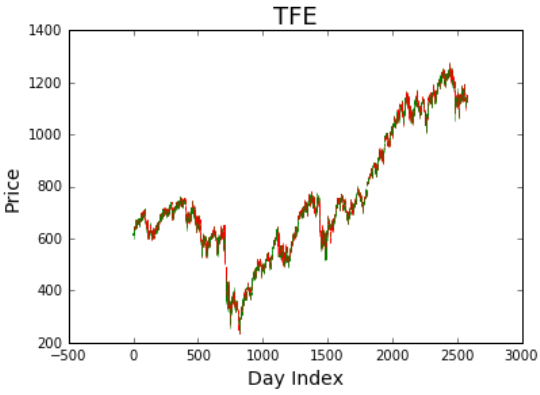
Once we cleaned the data we charted it in both bar chart and histogram format in order to determine what other preprocessing might be necessary.

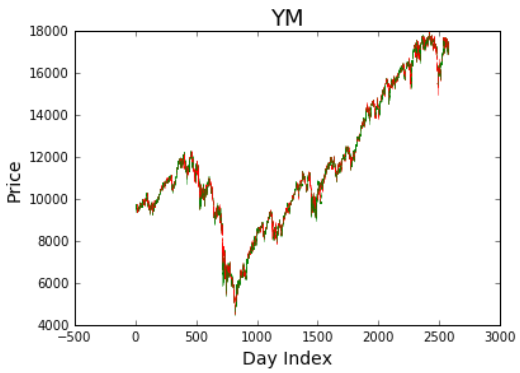
Bar Charts:



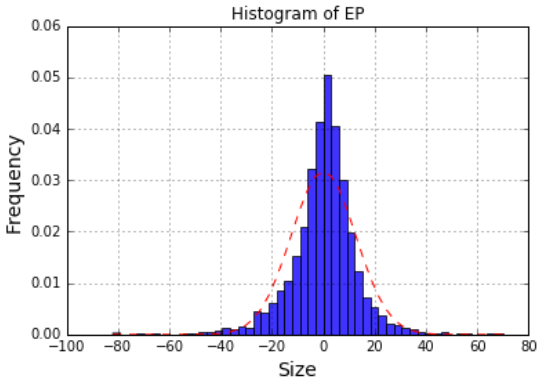


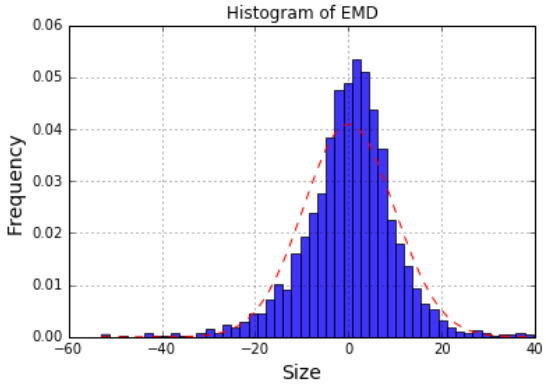


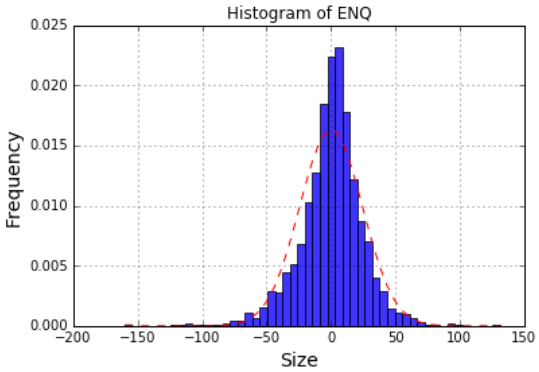


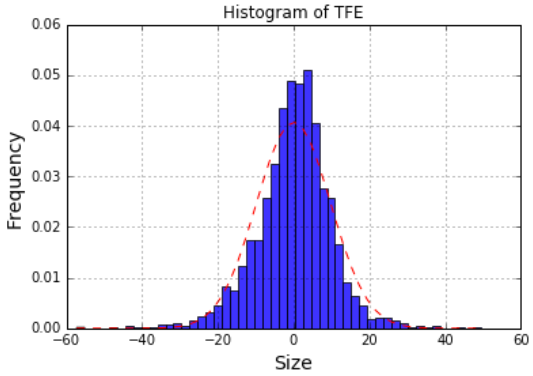


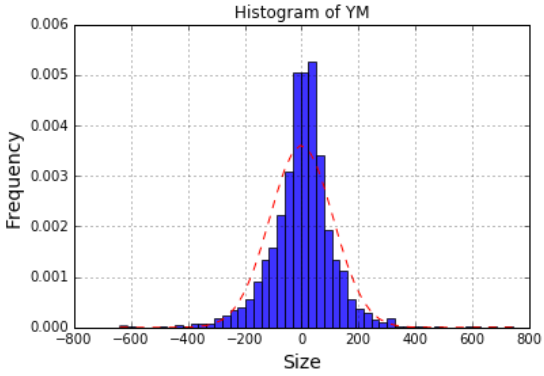
Histograms:


Histogram of EP
Frequency
Size
0.06
0.05
0.04
0.03
0.02
0.01
0.00
−100 −80 −60 −40 −20 0 20 40 60 80


Histogram of EMD
Frequency
Size
0.06
0.05
0.04
0.03
0.02
0.01
0.00
−60 −40 −20 0 20 40


Histogram of ENQ
Frequency
Size
0.025
0.020
0.015
0.010
0.005
0.000
−200 −150 −100 −50 0 50 100 150


Histogram of TFE
Frequency
Size
0.06
0.05
0.04
0.03
0.02
0.01
0.00
−60 −40 −20 0 20 40 60


Histogram of YM
Frequency
Size
0.006
0.005
0.004
0.003
0.002
0.001
0.000
−800 −600 −400 −200 0 200 400 600 800

The charts demonstrate an upward trend with 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. We determined from the visualization that it would be necessary to engineer features for to improve visualization. This project's feature engineering is described in the "Data Preprocessing" section of this report.

Algorithms and Techniques

We attempted many different algorithms in this project. Since it was a classification problem we attempted: Random Forest, SVC, SVC with PCA, and several neural nets. Each technique was trained on a train data set and validated on a validation data set.
The algorithms were chosen based on their suitability to the classification problem and their relatively low training time.

Benchmark

We divided the data into 3 sets: a training set, a validation set which consisted of the penultimate 400 days and the test set which was the final 400 days. We only used the test set to determine whether or not our algorithm beat the both benchmarks:

-A weak benchmark: the algorithm had to have above 50% accuracy in the test sample

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

We considered 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 attained more than 50% accuracy would be classified as beating a benchmark. However in the exploratory visualization section we saw an historical upward trend. So we added a second stage to our benchmark; one that measured 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 performed 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 used different techniques depending on the model (PCA, whitening, dummy variable creation).

In the general stage we removed 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 created many features that summarized historical data: volatility measures, distances of opens from moving averages, and distances of opens from

previous closes, and others. We also created a binary label that stored whether the day was up or down to test our models.

In the model specific stage we mainly formatted the data for to allow for model implementation. We also trained different models either individually on markets or on all markets together depending on previous accuracy of similar models with each training method. Finally we performed PCA on the datasets for certain models.

Implementation and Refinement

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

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

Because the results skewed heavily in favor of the SVC's we stopped investigating decision trees and concentrated our efforts on SVCs. We ran several grid searches on complexity and kernel types. The computing power available was only able to calculate RBF and any complexity greater than 3 was found to over fit.[4] We trained models on markets individually and all markets together. Although we found it did not make a difference for SVC's. We also ran 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) achieved this score.

After testing SVC's and Decision trees we moved on to neural networks. We decided not to batch data because SVC's were able to achieve high scores when they trained on all markets together. According to Stanford's cs231 (Stanford), non-convolutional neural nets do not benefit from more than 3 layers. We tested neural nets with 2 and 3 layers with and without drop out and different activation functions. We also attempted 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 attempted PCA using the same number of features as found in our full set and the neural nets achieved better accuracy. SVC's received 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 decided 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 picked that model because it used the full dataset and non-batched data; it would be the easiest to implement in other futures markets. The final model has an RBF kernel because the linear system did not calculate in a feasible time on the computing power available.

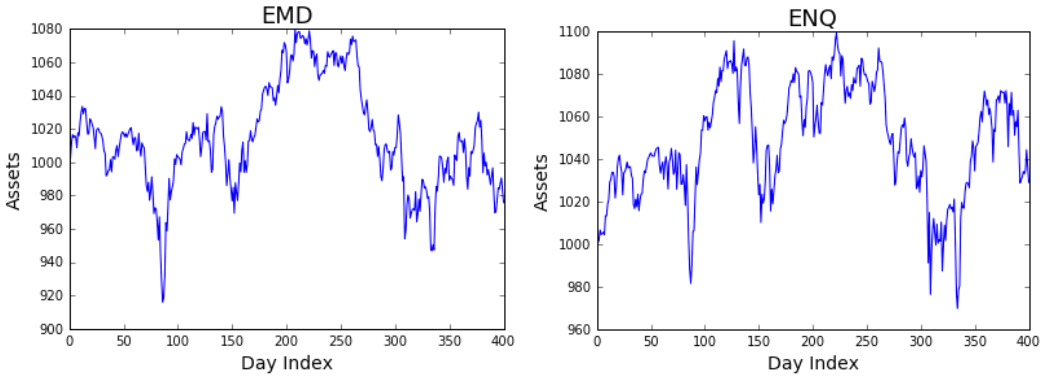
When we tested our chosen model on the test data it received a score of 0.5197. It beat the first benchmark, but not by a significant margin. For the second benchmark it received the same amount of dollars on the equity curve as simply going long. The model cannot be considered to beat the second benchmark.

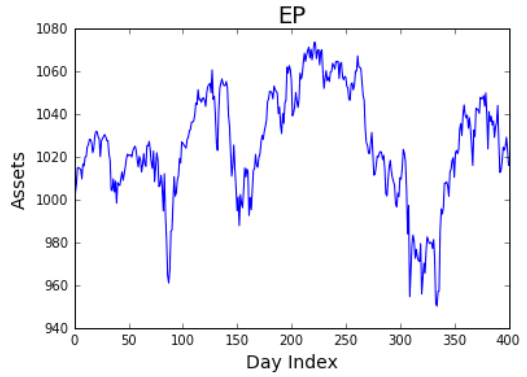
Overall the model predicted a good strategy for equities (going long), but it failed to generate any better trading ideas from the data available. It is not significant enough to have solved the problems. Ideas for improvement can be found in the "improvement" section.

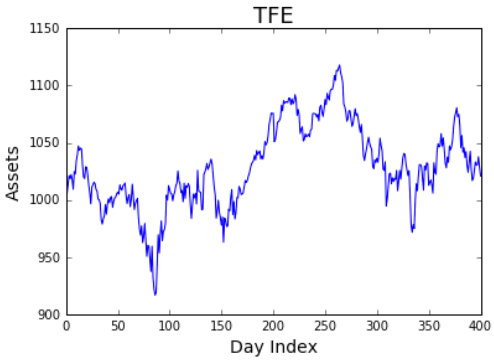
Conclusion

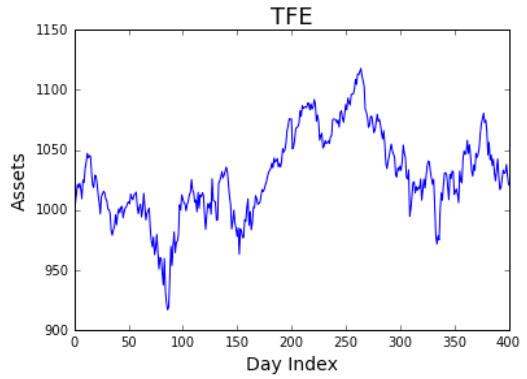
Free Form Visualization

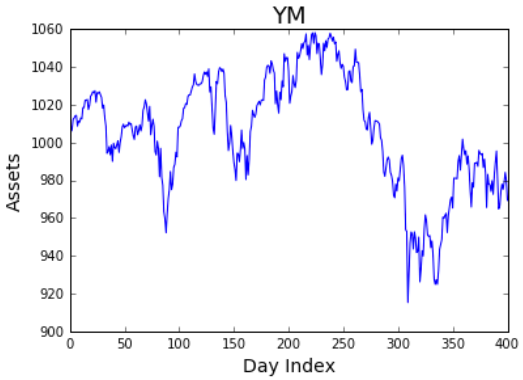
In the below we can see the equity curves for the final model. They are the same as simply going long (buying every day)











In these graphs we assumed 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.

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.

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.

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.