pyTrading: A Framework for the Creation and Analysis of Automated Trading Strategies

Mark Hamilton MARK.HAMILTON@YALE.EDU
Yale University, New Haven, CT 06520 USA

Abstract

pyTrading is a Python package designed to aid in the creation and analysis of automated stock trading strategies. pyTrading has portfolio classes to aid financial management as well as strategy classes that can automatically manage your finances. The package also contains novel additions to the popular Python machine learning package scikit-learn (Pedregosa et al., 2011). Through this work, I have created a class that transforms any scikit-learn regressor into a time-series regressor or a sequence to sequence regressor. This substantially generalizes linear dynamical systems auto-regression to a large class of nonlinear models already implemented in scikit-learn. This time-series regression technique was used in conjunction with a strategy object to yield a strategy that substantially outperforms the average "Buy and Hold" strategy.

1. Overview

Through this project I have created an testing and analysis framework for stock investment strategies. This framework focuses on modularity and code reuse. Each piece is designed to serve a well defined purpose, and maximize extensibility and flexibility. This work is designed with developers in mind, and much of its value comes from its interface and straightforward way to add more strategies. The project contains Portfolio, Strategy, and TimeSeriesRegressor classes for reasoning about the stock market and portfolio management. The pyTrading framework can be found at: `https://github.com/mhamilton723/pyTrading`

Proceedings of the 30^{th} International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

1.1. Portfolio Class

The portfolio class keeps track of all transactions, and has parameters for percent and flat rate commission. The portfolio exposes several member functions for buying and selling tickers at a listed price, and abstracts away commission calculations and safety checks. The portfolio also contains several utilities for calculating prices, checking if tickers are owned, and adding and subtracting capital from the balance.

1.2. Strategy Class

The strategy class consists of a general agent based framework for managing portfolios. A strategy 'runs' on an iterator of stock data, each day it will update the day's prices and its internal memory state, and then act on the data it observed by choosing to buy or sell stocks that day. This simulates a real world trading environment. Strategy objects interface with underlying portfolio objects which are defined as members of each strategy object. Class extension maximizes code re-use and makes defining strategies flexible and intuitive. Each strategy inherits two main functions from the ABC. The first function will is called "*observe_datum*", which reads the next value of stock data iterator and updates the strategy's internal memory state. The second is called "act" and uses the strategy's internal state to decide whether to buy or sell stocks. Many strategies can be easily encoded in this framework. Already, pyTrading has weighted, multi-stock versions of buy and hold and momentum strategies.

1.3. Informed Buy and Hold Strategies

In addition to the buy and hold, and momentum strategies, pyTrading has a general class of strategies called informed buy and hold strategies. These strategies wait for a period of time to gather data before analyzing it to choose a portfolio to buy and hold. Currently, pyTrading has two main classes which inherit from this base class: BestChangeBuyAndHold-

Strategy and TSEBuyAndHoldStrategy. The former chooses top stocks based on which stocks had the highest change in the waiting period. The latter fits a time series estimator to the system of stocks (I used the s&p500 stocks, but this is not hard-coded) and then forecasts the prices into the future and picks the top k results from the forecast. Each of these strategies have flags to weight the resulting portfolio uniformly, proportional to the change, or proportional to the log of the change (for positive changes). The time series estimator strategy is completely general in the sense that it can use any time series estimator, linear or otherwise. Because of this, the TSEBuyAndHoldStrategy parametrizes a space of strategies as large as the space of sci-kit learn regressor object. In tests, this strategy significantly outperforms the standard buy and hold strategy. The strategy is also interpretable if its internal predictor has an interpretable set of parameters such as the lasso, decision tree, or linear regression.

1.4. Utilities

pyTrading also had several utilities for the automatic processing of time series data. More specifically, 'pyTrading.utils' has methods for splitting datasets, and creating cross validation sets for statistically robust backtesting (time_series_cv and cascade_cv). I also have created a python decorator that intelligently caches intensive computations, and will automatically update the cache if the calling parameters have changed. This has served very useful for decreasing redundant calculations and exponentially speeding development time.

1.5. A Strategy Independent Back-Testing Function

pyTrading.utls contains a backtesting function that is strategy independent. pyTrading uses the unified strategy interface discussed in 1.2 to define backtesting functions at a high level of abstraction. More broadly, this allows one to create other testing functions compatible with any type of strategy as I do in the ipython notebooks. This allows for a powerful strategy comparison platform with very intuitive and simple interface.

2. The Time Series Estimator (TSE)

In addition to creating a automated trading strategy platform, I have added a novel type of estimator to the popular machine learning package sci-kit learn. This estimator transforms any sci-kit learn regressor into a sequence to sequence auto-regressor or time series auto-regressor. This greatly extends the class of problems solvable by sci-kit learn. I am currently trying to integrate this code into the next release of sci-kit learn. In the meantime this can be found within pyTrading or as a standalone repository at https://github.com/mhamilton723/TimeSeriesRegressor

2.1. Theory

The time series estimator generalizes auto-regressive models where one is looking for a function that maps sequences to sequences. Consider the sequences of vectors, x_i, y_i where :

$x_i \in \mathbb{R}^m, y_i \in \mathbb{R}^n, i \in [1...I]$, and $n, m, I \in \mathbb{N}^+$

To learn a sequence to sequence mapping, one is looking to find a function: $f \in \mathcal{F} \subset \mathbb{R}^{m \times w} \to \mathbb{R}^n$

such that:

$$f = \arg\min_{g \in \mathcal{F}} \sum_{i=w+1}^{I} loss(g(x_{i-w}, x_{i-w+1}, ..., x_{i-1}), y_i)$$

where usually:

$loss(a, b)$ is the mean squared error.

To model linear dynamical systems, the function space $\mathcal{F}$ is taken to be the space of $m \times n$ matrices. In general however, $\mathcal{F}$ can be any nonlinear function space. Hence the time series regression problem can be simplified to the classical regression problem on an altered domain. Namely $\mathbb{R}^{m \times w}$ where $w$is the number of steps to look back in time instead of $\mathbb{R}^m$.

2.2. Integration with scikit-learn

The TSE is designed to integrate seamlessly with the all estimators and functions defined in scikit-learn. Any estimator that maps vectors to vectors can be used as an internal or 'base_model' for the TSE. Furthermore, even if an sci-kit learn estimator can only map vectors to numbers, this can be transformed into a m-vector to n-vector mapping by creating n m-vector to scalar maps and concatenating their results. This feature can be accessed by using the "parallel_models" flag in the TSE constructor.

The TSE can be used within a pipeline as a final regressor object. This allows the TSE to be composed with other transformers, for instance a scaler, PCA, or Auto-Encoder. For stock market regression, one could use the canonical scalers, which convert prices to log(prices) or convert prices to percent changes from the previous day. Pipelines can also be used *within* a TSE to improve the sequence to sequence mapper. For example, one could use a PCA or Auto-Encoder to reduce the dimensionality of the data, effectively transforming the high-dimensional space of 500+ stock

prices to a lower-dimensional space that captures the most salient features of the market.

Furthermore, TSEs are compatible with all sklearn hyper-parameter search methods including GridSearchCV and GaussianProcessCV (still in development but will be added soon). One can tune the parameters of the TSE, such as the number of previous data points to use 'n_prev' and the model used in the regression 'base_estimator'. Additionally, one can tune the parameters of the base estimator itself because any excess parameters supplied to the TSE are passed directly into the base estimator.

However, one must take caution when using cross validated search mechanisms. Traditional K-fold or Leave-one-out CV techniques randomly sample the dataset, or break the data into folds which are then recombined to form multiple training and testing sets. This random sampling assumes i.i.d data and destroys it's temporal ordering, making any results found on the cross validation sets not applicable to the original problem. Luckily, the TimeSeriesEstimator module has several time-series cross validation techniques to overcome this issue. The function 'time_series_cv' splits the data up into non-overlapping folds, leaving the final %'test_fraction' of the dataset for validation. 'cascade_cv' generates overlapping folds with the final test fraction of the dataset for validation, this option is optimal for small datasets or large numbers of folds.

3. Usage

Please see the pyTrading Demo ipython notebook at https://github.com/mhamilton723/pyTrading/blob/master/PyTrading%20Demo.ipynb for an interactive demo of the portfolio and strategy objects. Please see the TimeSeriesEstimator Demo ipython notebook at https://github.com/mhamilton723/pyTrading/blob/master/Time%20Series%20Estimator%20Demo.ipynb for a tutorial of basic usage, grid search usage, prediction, forecasting ahead, statistically jittered forecasting as in figure 1, and other functionality.

4. Results

Please see the pyTrading Demo ipython notebook mentioned in 4 at for an interactive demo of the strategies' returns and for a detailed quantification of their effectiveness. Please see the TimeSeriesEstimator Demo ipython notebook for several results on their effectiveness at predicting the stock market as in figure 2, the quantitative optimization of several parameters, and statistically justified model selection. Select plots

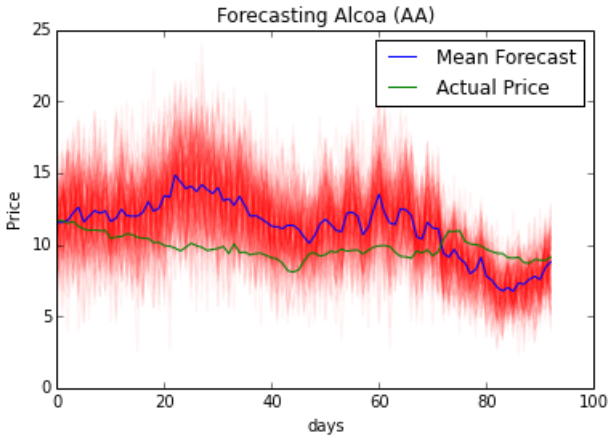

Figure 1. 200 statistically jittered forecasts of Alcoa ('AA') made with a linear time series regressor. This feature allows one to visualize the approximate uncertainty of the forecast

are shown here for convenience.

4.1. Explaining the Decision

As one can see in figures 2 and 3, the linear TSE significantly outperforms the mean buy and hold strategy yielding a return of $\approx 50\%$ over the course of 260 days. This model was only trained for 100 days of data and was not the best performing forecaster on the dataset, but was shown here as fast running proof of concept. It can also justify its decisions by comparing its results to the space of other buy and hold strategies as shown in figure 3. Furthermore, a more statistically robust study of performance of the time series estimator was undertaken in 4. It found that the TSE suggests an above average performing stock on average. This decision system can also actively justify its decisions by using the forecast function to see where the regressor predicts the market will be arbitrarily far ahead. It can then and reporting the predicted changes in price. The use of jittered forecasts as in figure 1 can yield approximate measures of risk. These measures of risk can easily be incorporated into a strategy in the form of a discounting factor or expected value calculation. The use of the back-test function helps tie this belief with reality as one can actively observe each strategy's performance in a real world situation.

5. Currently under development

In addition to the software presented above there are several new developments that will be added to the software package as soon as I complete them.

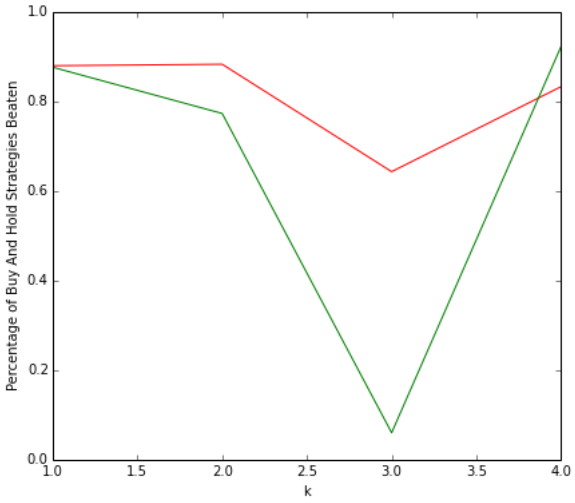

Figure 2. The fraction of buy and hold strategies beaten by a linear (red) and decision tree (green) time series regression informed buy and hold strategy. K represents the number of stocks chosen for the portfolio

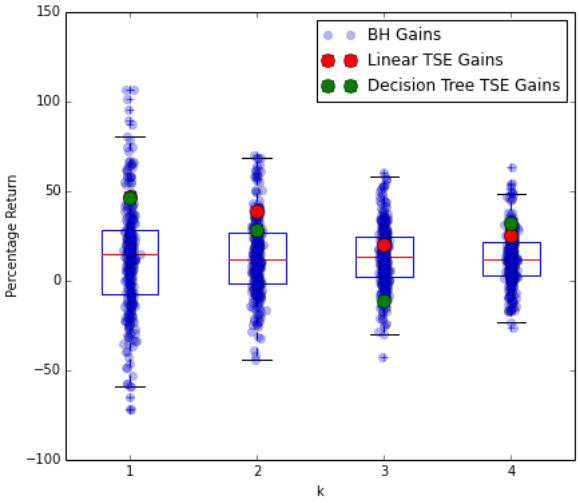

Figure 3. Boxplots of the returns of buy and hold strategies sampled uniformly from the space of possible strategies (the space of k-ticker subsets of s&p500 stocks). Returns made by a linear TSE informed buy and hold strategy are shown in red, decision tree results shown in green

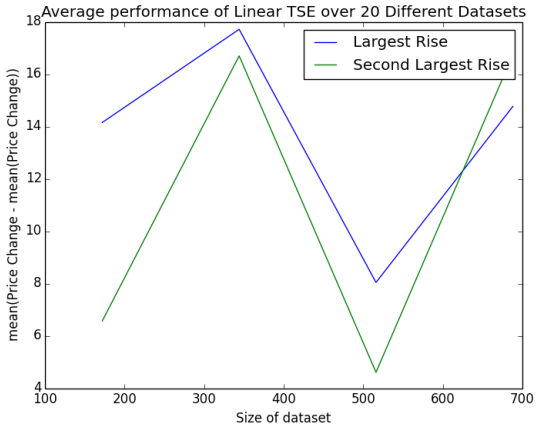

Figure 4. An analysis of 20 different datasets taken from 2009-1-1 to 2015-11-1. The metric on the y axis captures the ability for the TSE to predict stocks that do better than the average stock.

5.1. GPU Enabled Recurrent Neural Nets in Theano and Keras

In the pyTrading repository I have included several scripts for creating and running GPU enabled recurrent neural networks on the s&p500 data (see the rnn folder). These nonlinear function estimators have performed very well in language learning tasks and industrial time series analysis. Furthermore, these estimators are universal which means that they can approximate any computable function or program given enough data and neurons. I have created a very modular python script which creates and runs many different RNN architectures including SimpleRNN, Long Short Term Memory Networks (LSTMs) (Gers et al., 2000) and Grated Recurrent Units (GRUs). Almost every parameter of the nets can be controlled via the command line and all computationally intensive components can be cached. It's command line interface allows for running many parallel jobs on clusters. I also have created several scripts for automatically submitting hundreds of parallel jobs on Yale's omega cluster. The script can be modified for other clusters that use PBS or Torch. The neural nets are built using Keras, which is built on Theano, which is a language for automatic differentiation and deep learning research. This framework automatically compiles and simplifies models making them easy to scale to multiple CPUs or a GPUs. The reason these nets are not currently in the framework is because they do not yet forecast data well. I would like to add overshooting to these nets to improve their forecasting.(Zimmermann et al., 2012) I would also like to implement the Historically Consistent Neural Network (HCNN) in theano. This ar-

chitecture is specifically designed to model time series regression of a dynamical system like the stock market. (Zimmermann et al., 2012)

6. Code

All code for pyTrading can be found in the git repository:

https://github.com/mhamilton723/pyTrading

The above link also includes the TimeSeriesEstimator, for a standalone version of the TSE please see:

https://github.com/mhamilton723/TimeSeriesRegressor

7. Acknowledgements

This project would not have been accomplished without Professor Sahand Negabahn and Catherine Holland who both contributed helpful and guiding discussions. Thanks also to Stephen Slade, Ronghui Gu, and Kun Ren for approving and reviewing this project.

References

Gers, Felix A, Schmidhuber, Jürgen, and Cummins, Fred. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

Pedregosa, Fabian, Varoquaux, Gaël, Gramfort, Alexandre, Michel, Vincent, Thirion, Bertrand, Grisel, Olivier, Blondel, Mathieu, Prettenhofer, Peter, Weiss, Ron, Dubourg, Vincent, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

Zimmermann, Hans-Georg, Tietz, Christoph, and Grothmann, Ralph. Forecasting with recurrent neural networks: 12 tricks. In *Neural Networks: Tricks of the Trade*, pp. 687–707. Springer, 2012.