Documentation for Arbit – Ben Lackey

Arbit is a program trading platform written mostly in Python. Arbit uses a variety of third party components as well.

Contents

[Core Idea 2](#_Toc228639018)

[Algorithm 2](#_Toc228639019)

[Predictors 2](#_Toc228639020)

[Naïve Bayes 2](#_Toc228639021)

[E(symbol)/E(bucket) 2](#_Toc228639022)

[Ideas 2](#_Toc228639023)

[Architecture 2](#_Toc228639024)

[Yahoo! Finance 3](#_Toc228639025)

[To Do 3](#_Toc228639026)

[DNS 3](#_Toc228639027)

[Back Testing Arbit 4](#_Toc228639028)

[Server 4](#_Toc228639029)

[Configure NTP 4](#_Toc228639030)

[Configure DHCP, TFTP and PXE 4](#_Toc228639031)

[Configure EMS on Server 4](#_Toc228639032)

[Create Live CD 5](#_Toc228639033)

[To Do 6](#_Toc228639034)

[Node 6](#_Toc228639035)

[Run Time Arbit 6](#_Toc228639036)

[Ameritrade API 7](#_Toc228639037)

[Trade Engine 7](#_Toc228639038)

[To Do 8](#_Toc228639039)

[Long Term Enhancements 8](#_Toc228639040)

# Core Idea

I know more about machine learning than most finance people. Likewise, I know more about finance than most machine learning people. Finally, I don’t have a lot of money to invest. I should be able to make money trading with techniques that hedge funds couldn’t use because they would move markets. Day traders couldn’t use these techniques because they wouldn’t know how to code them.

My techniques have to be unique. Acting as a liquidity provider is by no means unique in itself. Day traders use technical indicators that assume mean reversion or momentum to do this. Instead, I assume a stochastic model that does not depend on mean reversion.

I use a form of Naïve Bayes with some predictors as input to pick volatile stocks. The classifier attempts to find a stock that will have a high 2% above its open on a given day. I buy at open, place a sell order at open+2%, and then sell any remaining stock just before the market closes.

# Algorithm

Different versions of arbit have always had an overriding similarity: they all output a value per symbol per day. A calling program takes s=Max[p(symbol, day)] for a given day across a set of symbols. The calling program then buys on the next trading day.

## Predictors

### E(Peaks)

Avg(peaks)

Std(Peaks)

Graph the two, maybe figure out something about for p(win| avg, std) is distributed

### Naïve Bayes

Use a naïve bayes classifier to combine a number of different predictors. The problem is this doesn’t have a risk component. So, even though it often wins 80-90% of the time, the losses make the limit trend to 0.

### E(symbol)/E(bucket)

This is equivalent to the expected return for a symbol over some window because the E(bucket) term cancels out.

### E(symbol)/E(bucket)

Use this strategy with both a long and a short position

## Ideas

* Analogous to feature selection, symbol selection. Cross validate to avoid overfitting.
  + Assumes that classifier isn’t reliable

# Architecture

There are two versions of Arbit:

* runtime version that automatically trades during the day
* back testing version that uses multiple compute nodes to quickly back test large amounts of data

## NASDAQ

NASDAQ.com hosts lists of the symbols traded on the NASDAQ, NYSE and AMEX. The symbols.py module grabs those lists from there. This is used because Ameritrade doesn’t provide any symbol lists.

## Yahoo! Finance

Yahoo finance provides historical data going back decades. Unfortunately only interday data is available. For this reason, the current version of arbit is using the Ameritrade API instead. The Yahoo! Finance functionality will remain in quotesYahoo.py, but it’s not being maintained right now.

## Ameritrade API

The ameritrade.py module wraps parts of Ameritrade’s API. This is used to get intraday data and to perform trades. The quotesAmeritrade.py module wraps ameritrade.py in a cleaner interface for grabbing the intraday data.



Figure : Ameritrade API Documentation

Ameritrade maintains forums for the API at: [http://apiforums.tdameritrade.com](http://apiforums.tdameritrade.com/tda-board/ubbthreads.php)



Figure : Arbit Network Diagram

## DNS

It’d be nice to have a way to access the server remotely. This would allow updates to the server while I’m traveling, and make it possible for remote nodes to work on a JMS queue. I see two options:

* dynDNS
  + need a domain name (it’d be hard to set up a secondary for nonplatonic.com) A little grand, perhaps:
    - arbitfund.com
    - arbitcapital.com
  + BT hub allows this, but then Yesler would need to DHCP from there while the nodes DHCP from Yesler. Not sure I want two DHCP servers on the same network.
* Real Hosting

# Back Testing Arbit



Figure : Arbit Back Testing Architecture

## Server

Arbit uses JMS to distribute work between any number of worker nodes. When a node boots up, it gets a DHCP lease from the server, and then downloads a kernel via PXE. The kernel bootstraps up using TFTP until it can download an Ubuntu live cd over nfs. To set all that up, you need to do a couple things:

### Configure NTP

Set up network time.

### Configure DHCP, TFTP and PXE

…………still needs some cleanup with BT hub

<https://wiki.koeln.ccc.de/index.php/Ubuntu_PXE_Install>



Figure : dhcp conf file goes in /etc/dhcp3

### Configure EMS on Server

First, install TIB\_ems\_5.1.2\_linux24gl23\_x86.tar.gz as root.

EMS requires libstdc++.so.5. To get it run:

sudo apt-get install libstdc++5

Try running tibemsd64. You probably need to chmod 777 /opt/tibco/ems/5.1/bin to get permission to write the conf and log files

The EMS C libraries need to be available in the path for python to use them. To do this for all users, add to ~/.bashrc:

export LD\_LIBRARY\_PATH= /opt/tibco/ems/5.1/lib/64: /opt/tibco/ems/5.1/lib

Now, set emsd to start when the server does:

cd /etc/init.d

sudo chmod +x emsd.sh

sudo update-rc.d emsd.sh defaults



Figure : EMS start script goes in /etc/init.d

Create two queues using /opt/tibco/ems/5.1/bin/tibemsadmin:

create queue arbit.work.request

create queue arbit.work.response

### Create Live CD

The Live CD tells the node to mount an NFS on the server that has the client.py program as well as stock data from Yahoo! Finance. The client.py program then subscribes to a JMS queue that contains work items. The live cd was customized as described here:

<https://help.ubuntu.com/community/LiveCDCustomization>



Add to .bashrc:

export LD\_LIBRARY\_PATH=/opt/tibco/ems/5.0/lib

Figure : Config files for Server

The live cd needs a little custom configuration:

* Set a root password
* Put JMS libraries in library path
* Set up an init.d job
  + Mount NFS on Yesler
  + Copy arbit \*.py files from NFS
  + Copy market data from NFS
  + Start 4x client.py
* lm-sensors
  + CPU temperature
  + <http://ubuntuforums.org/showthread.php?t=2780>
* sshd
  + <http://www.cyberciti.biz/faq/ubuntu-linux-openssh-server-installation-and-configuration/>
* nfs client
  + <http://www.ubuntugeek.com/nfs-server-and-client-configuration-in-ubuntu.html>



Figure : pxe config for server goes in /var/lib/tftpboot/pxelinux.cfg

Finally, the server needs to run server.py which farms work out to the nodes. server.py publishes a bunch of work items to the queues and then quits. The work is collated using simulate.py which pulls finished items from the JMS server, writes them to disk, and then calculates return on any items on disk.

### To Do

* Add logic to start different numbers of client.py depending on the number of cores available. Right now it’s always 4.

## Node

The node is very simple to configure. It needs to be plugged in somewhere it can see Yesler’s DHCP server. It also needs to be configured to boot from the network in the BIOS settings.

# Run Time Arbit

The run time version of Arbit is a single python program that runs within a single process and calls classifier.py, grabs symbols from the NASDAQ, and invokes the Ameritrade API.



Figure : Arbit Run Time Architecture

The runtime version does not currently update its stock data. Instead, downloader.py does this as an asynchronous batch job.



Figure : downloader.py architecture

## Trade Engine

Using the Ameritrade API requires a simple trade engine. The first part is a download process that is scheduled to run every night. This process also runs when the downloader first starts.



Figure 8: download

The download is followed by a pretrade that runs the classifier. Right now this happens at 3am, though it should probably follow directly from the download. These are currently in separate programs: arbit.py and downloader.py



Figure : pretrade

In the morning, arbit wakes up and opens a position. It then enters a sell order to sell any stock purchased at a slight profit.



Figure : openPositions

The final component runs just before the market closes. It modifies the sell order if it hasn’t filled to a limit order at the current market price.



Figure : closePositions

### To Do

* Send an alert if a position is opened
* Send an alert if the position is closed
* Send an alert at end of day
  + Closed and Won
  + Closed and Lost
  + Open and Lost
    - Margin OK
    - Margin Call (because we may be on 4x SRO day-trader margin and only get 2x SMA overnight)
    - These cases need to be fixed by hand

# Long Term Enhancements

* Better Market Data
  + Historical options data
  + Fundamentals
* Hosting
  + Hideously expensive
  + Better uptime
  + Lower latency
    - Even collocating to the US would probably help
* Asset Management
  + A single server could manage multiple Ameritrade accounts
  + need financial certification
  + more capital available
    - for what?
    - shouldn't mine grow on its own?
  + Issues with secrets, intellectual property
* Trading Cabal
  + Share IT costs
    - Hosting
    - Development
    - Feeds
      * Could afford Bloomberg, Reuters
      * Trading Tech
      * Introduction to clearers – Marex?
  + Keep algorithms private
* More Instruments
  + Ameritrade
    - Derivatives
    - Bonds
  + UK
    - Open UK Brokerage Account with API support
      * Need to do before I move back to the States
      * Interactive Brokers
        + <http://www.interactivebrokers.co.uk/>
        + <http://www.interactivebrokers.com/>
  + Asia
    - Tokyo
    - Shenzhen
    - Hong Kong