Part I

Factor Models, Fama-French, and Pipeline

Oversimplification

Linear Regression:

Height = Nutrition + Sleep + Genetics (+ Error)

Linear Factor Models in Finance

$$R_i = a_i + b_{i1}F_1 + b_{i2}F_2 + \ldots + b_{iK}F_K + \epsilon_i$$

Linear Factor Models in Finance

$$R_i = a_i + b_{i1}F_1 + b_{i2}F_2 + \ldots + b_{iK}F_K + \epsilon_i$$

For an equity i:

R_i = returns on equity i
a_i = some constant
b_{ix} = change in return per unit change in F_x
F_x = value of factor x
e_i = random error

Fama-French Three-Factor Model

$$r = R_f + \beta_3(K_m - R_f) + b_s \cdot SMB + b_v \cdot HML + \alpha$$

Linear Regression Model

Purpose: to describe returns of an
equity over some time period

Fama-French Three-Factor Model

$$r = R_f + \beta_3(K_m - R_f) + b_s \cdot SMB + b_v \cdot HML + \alpha$$

Returns: (Rm-Rf), SMB, HML

Fama-French

$R_m - R_f$ = Returns of market - risk free returns

Risk free rate of return = 0

$R_m - R_f = R_m$ = S&P500 (SPY)

Fama-French

Partitioning our Universe

Market Cap

50th Percentile

$$\frac{\text{Book Value}}{\text{Market Cap}}$$

70th Percentile

30th Percentile

Small Value	Big Value
Small Neutral	Big Neutral
Small Growth	Big Growth

Fama-French

SMB: Small Minus Big

$$\begin{aligned} SMB = \tfrac{1}{3}\ &(\text{Small value returns} + \\ &\ \text{Small neutral returns} + \\ &\ \text{Small growth returns}) \\ -\ \tfrac{1}{3}\ &(\text{Big value returns} + \\ &\ \text{Big neutral returns} + \\ &\ \text{Big growth returns}) \end{aligned}$$

Fama-French

HML: High Minus Low

HML = ½ (Small value returns +
Big value returns)
- ½ (Small growth returns +
Big growth returns)

Fama-French

Partitioning our Universe

Market Cap

50th Percentile

$$\frac{\text{Book Value}}{\text{Market Cap}}$$

	Small Value	Big Value
70th Percentile		
	Small Neutral	Big Neutral
30th Percentile		
	Small Growth	Big Growth

Pipeline

- ➢ Effectively: map-reduce on securities data
- ➢ Before: algorithms limited to 500 securities
- ➢ Now: unlimited

- ➢ Compute on large universes
- ➢ Filter universes down to desired sets

Pipeline

Pipeline allows us to swiftly work with literally every security in our database.

This enables us to use large-scale strategies. This Fama-French implementation is a “Hello World” for a new class of strategies that users can deploy on Quantopian.

Fama-French & Pipeline

```
import pandas as pd
import numpy as np
from quantopian.algorithm import attach_pipeline, pipeline_output
from quantopian.pipeline import Pipeline
from quantopian.pipeline import CustomFactor
from quantopian.pipeline.data.builtin import USEquityPricing
from quantopian.pipeline.data import morningstar

# time frame on which we want to compute Fama-French
normal_days = 31
# approximate the number of trading days in that period
# this is the number of trading days we'll look back on,
# on every trading day.
business_days = int(0.69 * normal_days)
```

Fama-French & Pipeline

```
class Returns(CustomFactor):
    """
    this factor outputs the returns over the period defined by
    business_days, ending on the previous trading day, for every security.
    """
    window_length = business_days
    inputs = [USEquityPricing.close]
    def compute(self,today,assets,out,price):
        out[:] = (price[-1] - price[0]) / price[0] * 100

```

Fama-French & Pipeline

```
class MarketEquity(CustomFactor):
    """
    this factor outputs the market cap of every security on the day.
    """
    window_length = business_days
    inputs = [morningstar.valuation.market_cap]
    def compute(self,today,assets,out,mcap):
        out[:] = mcap[0]
```

Fama-French & Pipeline

```
class BookEquity(CustomFactor):
    """
    this factor outputs the book value of every security on the day.
    """
    window_length = business_days
    inputs = [morningstar.balance_sheet.tangible_book_value]
    def compute(self,today,assets,out,book):
        out[:] = book[0]

```

Fama-French & Pipeline

```
class CommonStock(CustomFactor):
    """
    this factor outputs 1.0 for all securities that are either common stock or SPY,
    and outputs 0.0 for all other securities. This is to filter out ETFs and other
    types of share that we do not wish to consider.
    """
    window_length = business_days
    inputs = [morningstar.share_class_reference.is_primary_share]
    def compute(self,today,assets,out, share_class):
        out[:] = ((share_class[-1].astype(bool)) | (assets == 8554)).astype(float)
```

Fama-French & Pipeline

```python
def initialize(context):
    """
    use our factors to add our pipes and screens.
    """
    pipe = Pipeline()
    attach_pipeline(pipe, 'ff_example')

    common_stock = CommonStock()
    # filter down to securities that are either common stock or SPY
    pipe.set_screen(common_stock.eq(1))
    mkt_cap = MarketEquity()
    pipe.add(mkt_cap,'market_cap')

    book_equity = BookEquity()
    # book equity over market equity
    be_me = book_equity/mkt_cap
    pipe.add(be_me,'be_me')

    returns = Returns()
    pipe.add(returns,'returns')
```

Fama-French & Pipeline

```
def before_trading_start(context,data):
    """
    every trading day, we use our pipes to construct the Fama-French
    portfolios, and then calculate the Fama-French factors appropriately.
    """
    spy = sid(8554)

    factors = pipeline_output('ff_example')

    # get the data we're going to use
    returns = factors['returns']
    mkt_cap = factors.sort(['market_cap'], ascending=True)
    be_me = factors.sort(['be_me'], ascending=True)

    # to compose the six portfolios, split our universe into portions
    half = int(len(mkt_cap)*0.5)
    small_caps = mkt_cap[:half]
    big_caps = mkt_cap[half:]

```

Fama-French & Pipeline

```
thirty = int(len(be_me)*0.3)
seventy = int(len(be_me)*0.7)
growth = be_me[:thirty]
neutral = be_me[thirty:seventy]
value = be_me[seventy:]

# now use the portions to construct the portfolios.
# note: these portfolios are just lists (indices) of equities
small_value = small_caps.index.intersection(value.index)
small_neutral = small_caps.index.intersection(neutral.index)
small_growth = small_caps.index.intersection(growth.index)

big_value = big_caps.index.intersection(value.index)
big_neutral = big_caps.index.intersection(neutral.index)
big_growth = big_caps.index.intersection(growth.index)

# take the mean to get the portfolio return, assuming uniform
# allocation to its constituent equities.
sv = returns[small_value].mean()
sn = returns[small_neutral].mean()
sg = returns[small_growth].mean()
```

Fama-French & Pipeline

```
    bv = returns[big_value].mean()
    bn = returns[big_neutral].mean()
    bg = returns[big_growth].mean()

    # computing Rm-Rf (Market Returns - Risk-Free Returns). we take the
    # rate of risk-free returns to be zero, so this is simply SPY's returns.
    # have to set an initial dummy value
    context.rm_rf = float('nan')
    if spy in returns.index:
        context.rm_rf = returns.loc[spy]

    # computing SMB
    context.smb = (sv + sn + sg)/3 - (bv + bn + bg)/3

    # computing HML
    context.hml = (sv + bv)/2 - (sg + bg)/2
```

Fama-French & Pipeline

```python
def handle_data(context, data):
    # print the Fama-French factors for the period defined by business_days
    # ending on the previous trading day.
    print(context.rm_rf, context.smb, context.hml)
    pass
```

Fama-French & Pipeline

```
2015-06-03 PRINT (0.014195135800133027, 1.2599008403438299, -0.9471552363179645)
2015-06-04 PRINT (1.4504547630445195, 2.5198264016819163, -1.8197691681797725)
2015-06-05 PRINT (1.0527327789261152, 2.37287351462154, -1.07017850008036)
2015-06-08 PRINT (0.44046536122946689, 3.6514499863493604, -0.3552867615539843)
2015-06-09 PRINT (-1.4885171533881514, 4.203448894150004, 0.4023551935638068)
2015-06-10 PRINT (-1.025592327049996, 3.5553186568700523, 0.2938412431344256)
2015-06-11 PRINT (0.47151838445418603, 4.0522657848885055, 0.2922592676949245)
2015-06-12 PRINT (0.73795467529994829, 4.773849401249658, 1.6162412398212527)
2015-06-15 PRINT (-1.0414212336836191, 5.502867894598888, 2.5629469846331916)
2015-06-16 PRINT (-1.590812820633499, 6.020220813657052, 2.4118590774471764)
2015-06-17 PRINT (-1.3280773382139062, 5.303317754635235, 2.2474150212253714)
2015-06-18 PRINT (-1.1220130510304616, 5.779978868701205, 3.052113537180644)
2015-06-19 PRINT (-0.051677158695857546, 5.136293592480323, 2.1427459741081942)
2015-06-22 PRINT (-0.93507683730575564, 4.722604655863224, 1.1440187817968483)
2015-06-23 PRINT (-0.018337754032817739, 4.621740125803042, 0.47718701604673)
2015-06-24 PRINT (1.1064293491658268, 4.623566956736383, 0.10815295522888801)
```

Implementation

Ken French's data library: to double-check

July 2015:

	Rm	SMB	HML
Ken French	1.54	-4.16	-4.50
Me	1.53	-4.67	-4.33

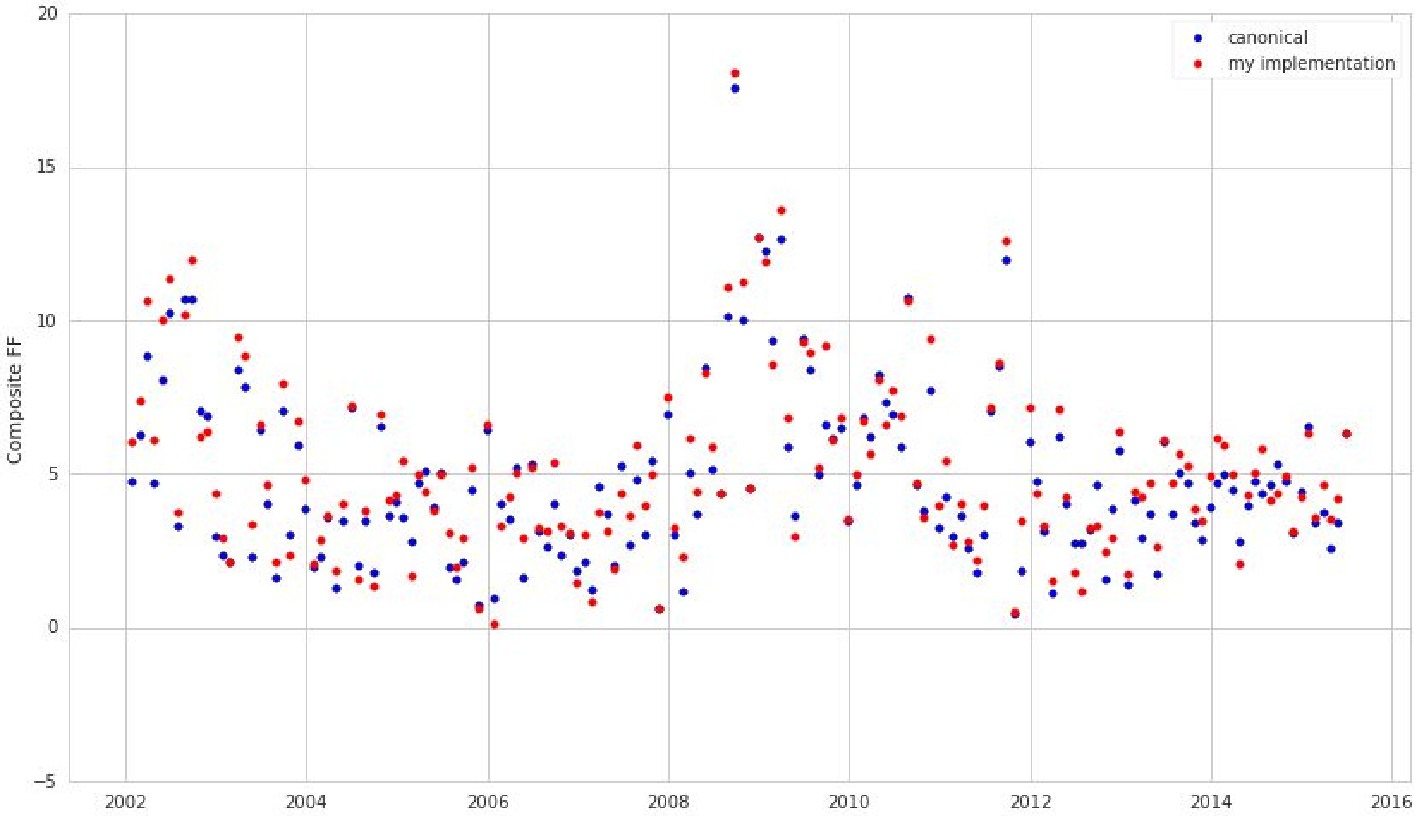
Implementation

Ken French’s data library: to double-check

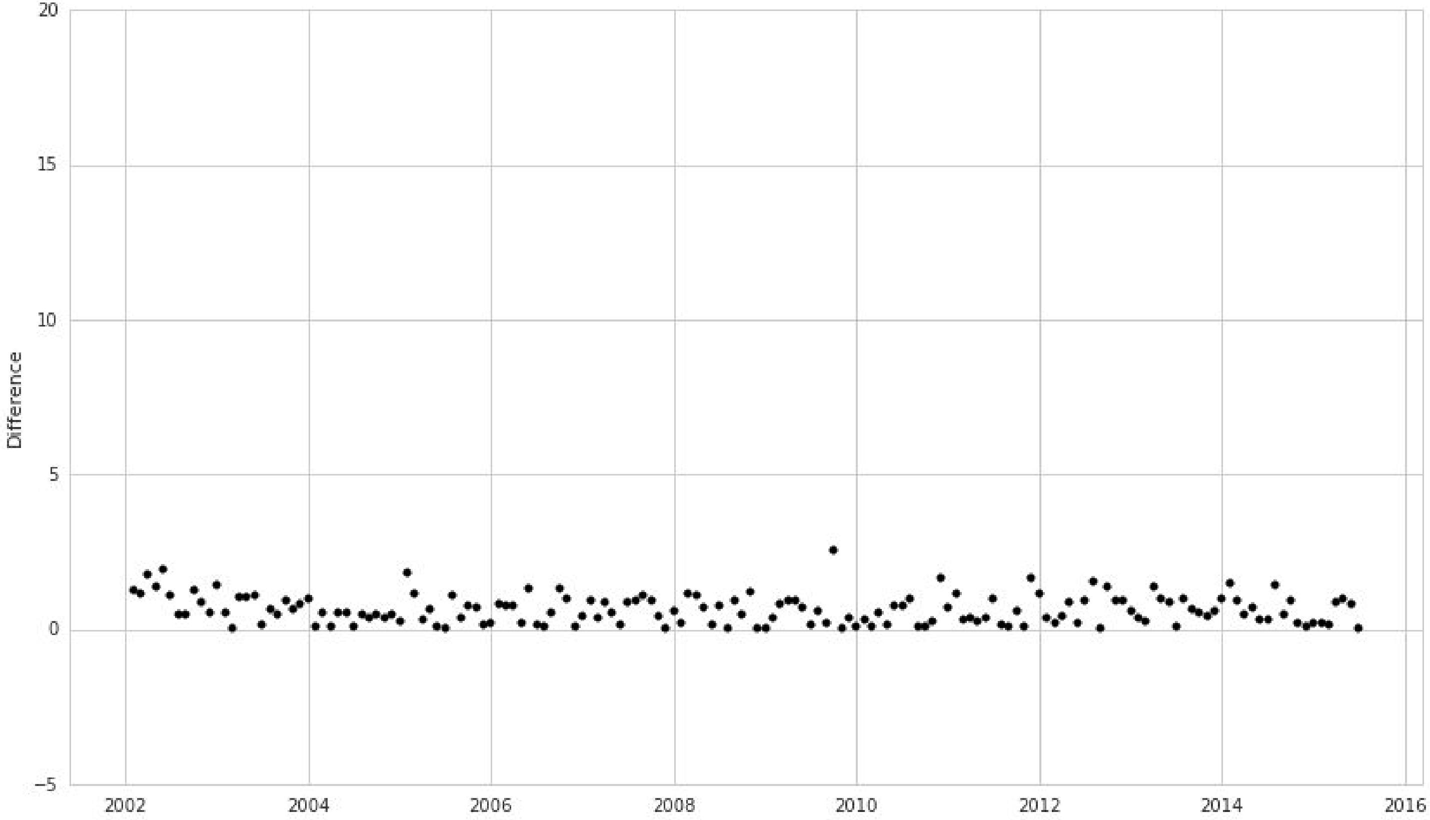
August 2014 - August 2015:

	Rm	SMB	HML
Ken French	11.28	0.91	-18.88
Me	10.04	-0.68	-19.99

Monthly Fama-French Comparison



Monthly Fama-French Comparison



Is Our Implementation Better?

Ken French’s implementation uses only NYSE, NASDAQ, and AMEX.

We use data from over 12 exchanges.

Applications

- ➢ Evaluate strategies with FF factor correlation
- ➢ Coming: Pipeline in research
- ➢ Pyfolio: regression against FF factors

Part II

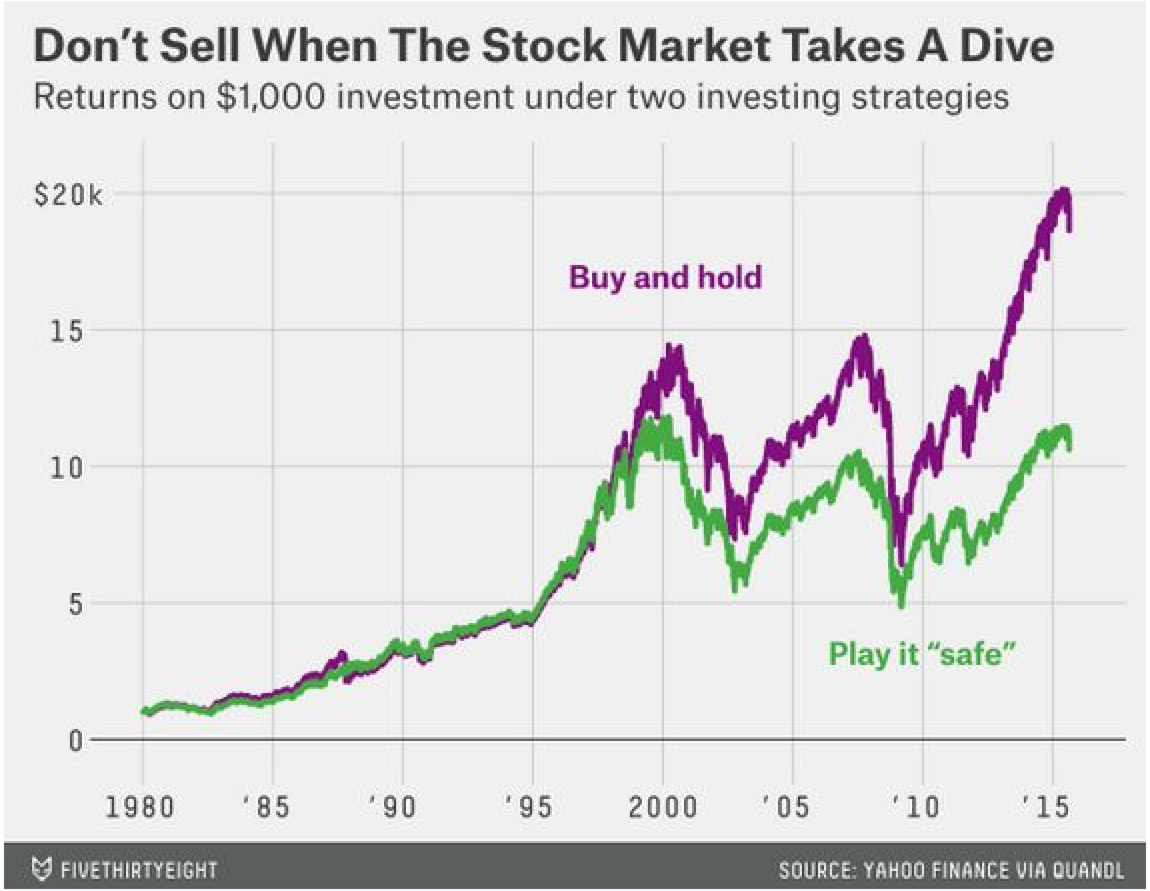
Parameter Optimization in Research

Quantopian Research

- ➢ iPython platform
- ➢ run algorithms
- ➢ import datasets
- ➢ use Python’s data-analytic libraries

Motivation

Zoom: 1d 5d 1m 3m 6m YTD 1y 5y 10y All
May 07, 2015 - Nov 06, 2015 +1.73 (0.83%)
210
200
190
$1.03
$1.03
May 2015
Jun 2015
Jul 2015
Aug 2015
Sep 2015
Oct 2015
Volume (mil / 1d)
510
255


Don't Sell When The Stock Market Takes A Dive
Returns on $1,000 investment under two investing strategies
$20k
15
10
5
0
1980
'85
'90
'95
2000
'05
'10
'15
Buy and hold
Play it "safe"
FIVETHIRTYEIGHT
SOURCE: YAHOO FINANCE VIA QUANDL

That’s just one strategy.

It’s not sufficiently rigorous.

We want more general results.

Does there exist a simple, profitable strategy

for lay investors that sells on market downturns

and buys back on rebounds?

Parameter Optimization

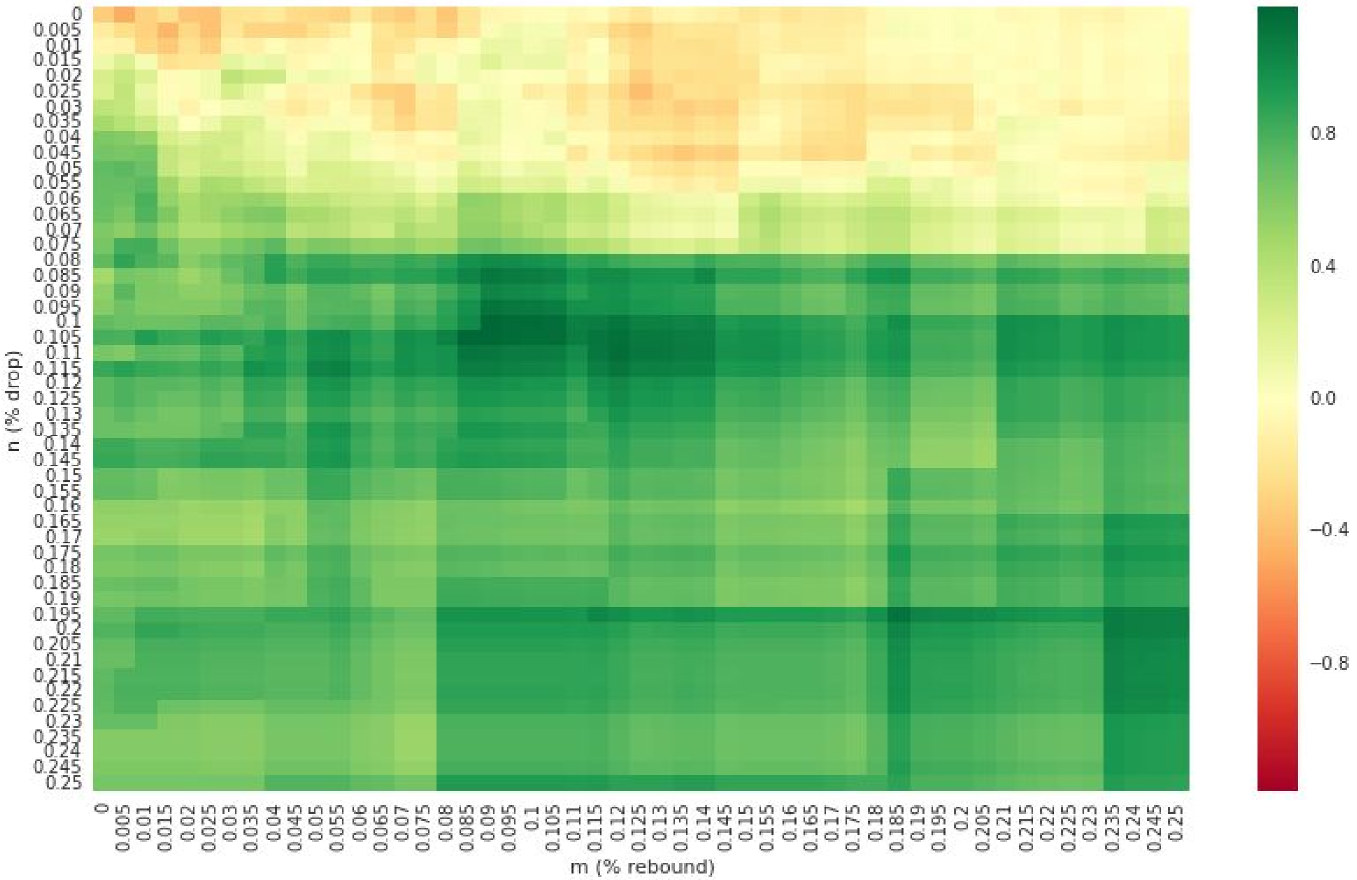
- ➢ Goal: optimize the percentage downturns and rebounds to sell and buy on
- ➢ Space: [0, 25%] Downturns
- ➢ [0, 25%] Rebounds
- ➢ Increments of 0.5%
- ➢ Benchmark: SPY returned ~85% between January 2002 and February 2015

```python
def initialize(context):
    context.n = n
    context.m = m
    context.day = 0
    context.max_price = 0
    context.min_price = 1000000
    context.spy = symbol('SPY')
    context.holding = False
    context.first_target = 0

def handle_data(context, data):
    price = data[context.spy].close_price

    # on the first day, buy as many shares as possible
    if context.day == 0:
        context.max_price = price
        target_number = context.portfolio.cash / price
        context.first_target = target_number
        order(context.spy,target_number)
        context.holding = True
        context.day +=1

    if context.holding:
        if price > context.max_price:
            context.max_price = price
        elif price < context.max_price * (1 - context.n):
            context.min_price = price
            held = context.portfolio.positions[context.spy].amount
            order(context.spy,-held)
            context.holding = False
    else:
        if price < context.min_price:
            context.min_price = price
        if price > context.min_price * (1 + context.m):
            context.max_price = price
            target_number = context.portfolio.cash / price
            order(context.spy,target_number)
            context.holding = True
```

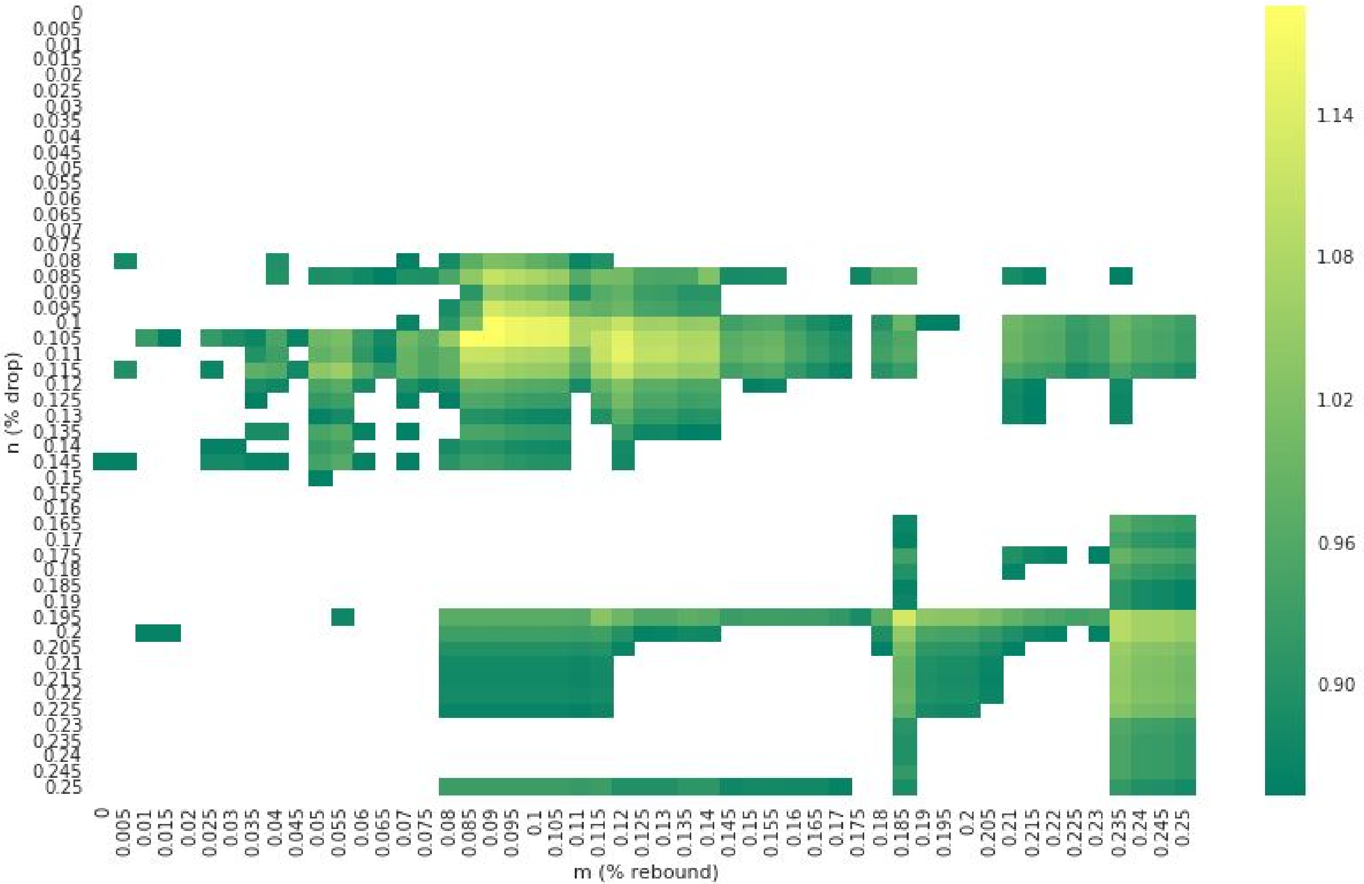
Iteration: 2500 backtests

```
# list of n- and m-values
ms = [0.000, 0.005, 0.010, 0.015, 0.020, 0.025, 0.030, 0.035, 0.040, 0.045, 0.050, 0.055, 0.060, 0.065,
      0.070, 0.075, 0.080, 0.085, 0.090, 0.095, 0.100, 0.105, 0.110, 0.115, 0.120, 0.125, 0.130, 0.135,
      0.140, 0.145, 0.150, 0.155, 0.160, 0.165, 0.170, 0.175, 0.180, 0.185, 0.190, 0.195, 0.200, 0.205,
      0.210, 0.215, 0.220, 0.225, 0.230, 0.235, 0.240, 0.245, 0.250]

ns = [0.000, 0.005, 0.010, 0.015, 0.020, 0.025, 0.030, 0.035, 0.040, 0.045, 0.050, 0.055, 0.060, 0.065,
      0.070, 0.075, 0.080, 0.085, 0.090, 0.095, 0.100, 0.105, 0.110, 0.115, 0.120, 0.125, 0.130, 0.135,
      0.140, 0.145, 0.150, 0.155, 0.160, 0.165, 0.170, 0.175, 0.180, 0.185, 0.190, 0.195, 0.200, 0.205,
      0.210, 0.215, 0.220, 0.225, 0.230, 0.235, 0.240, 0.245, 0.250]

backtest_count = 0
returns = pandas.DataFrame(index=ns,columns=ms)

for n in ns:
    for m in ms:
        algo_obj = TradingAlgorithm(initialize=initialize,
                                    handle_data=handle_data)
        perf_manual = algo_obj.run(data)
        backtest_count += 1
        print("Backtest {0} completed...").format(backtest_count)
        # grab the most recent return. careful to assign column-first
        returns[m][n] = perf_manual.returns.sum()
```

Heatmap: 2500 backtests



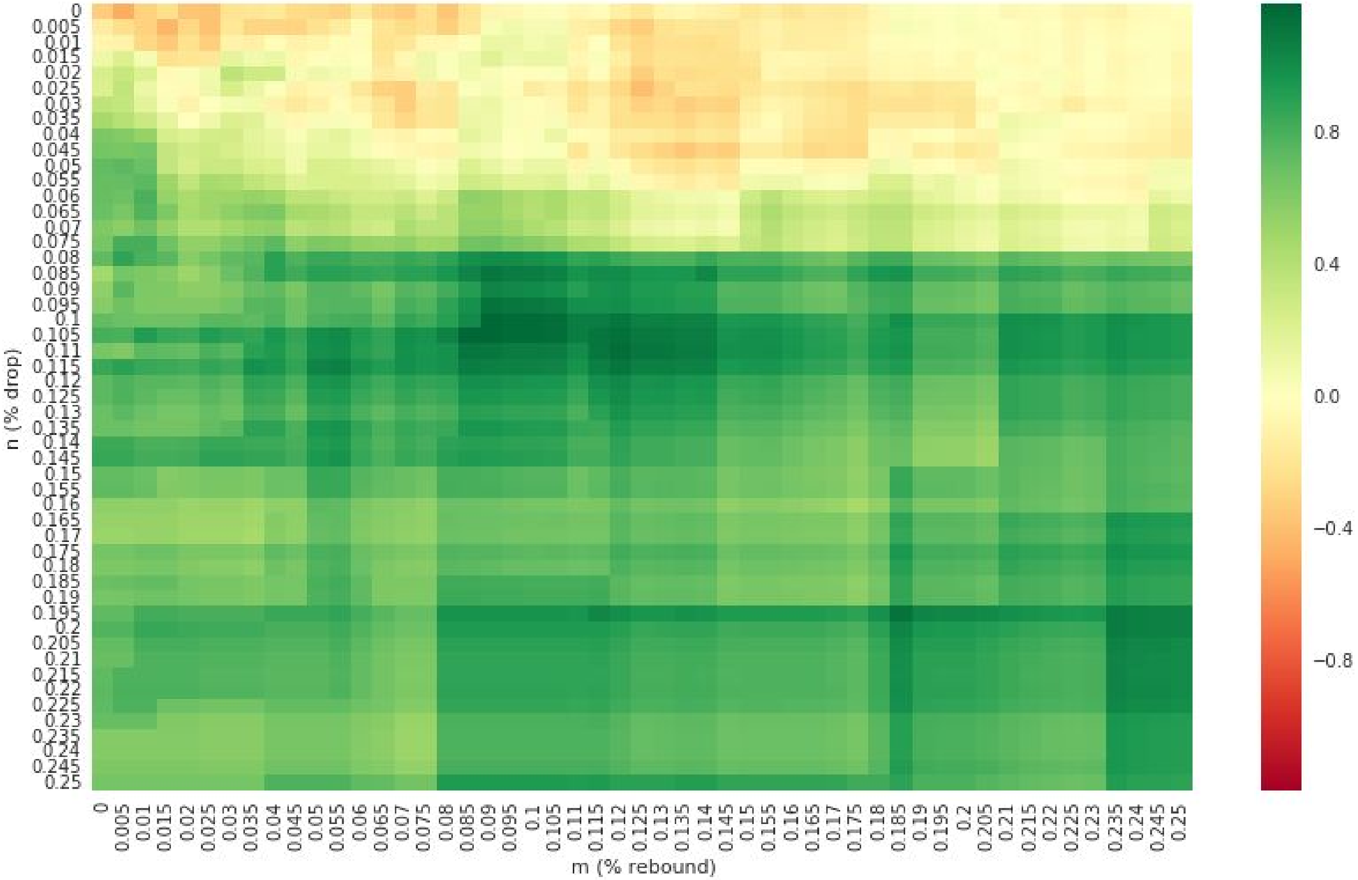
Threshold: Benchmark-Beating



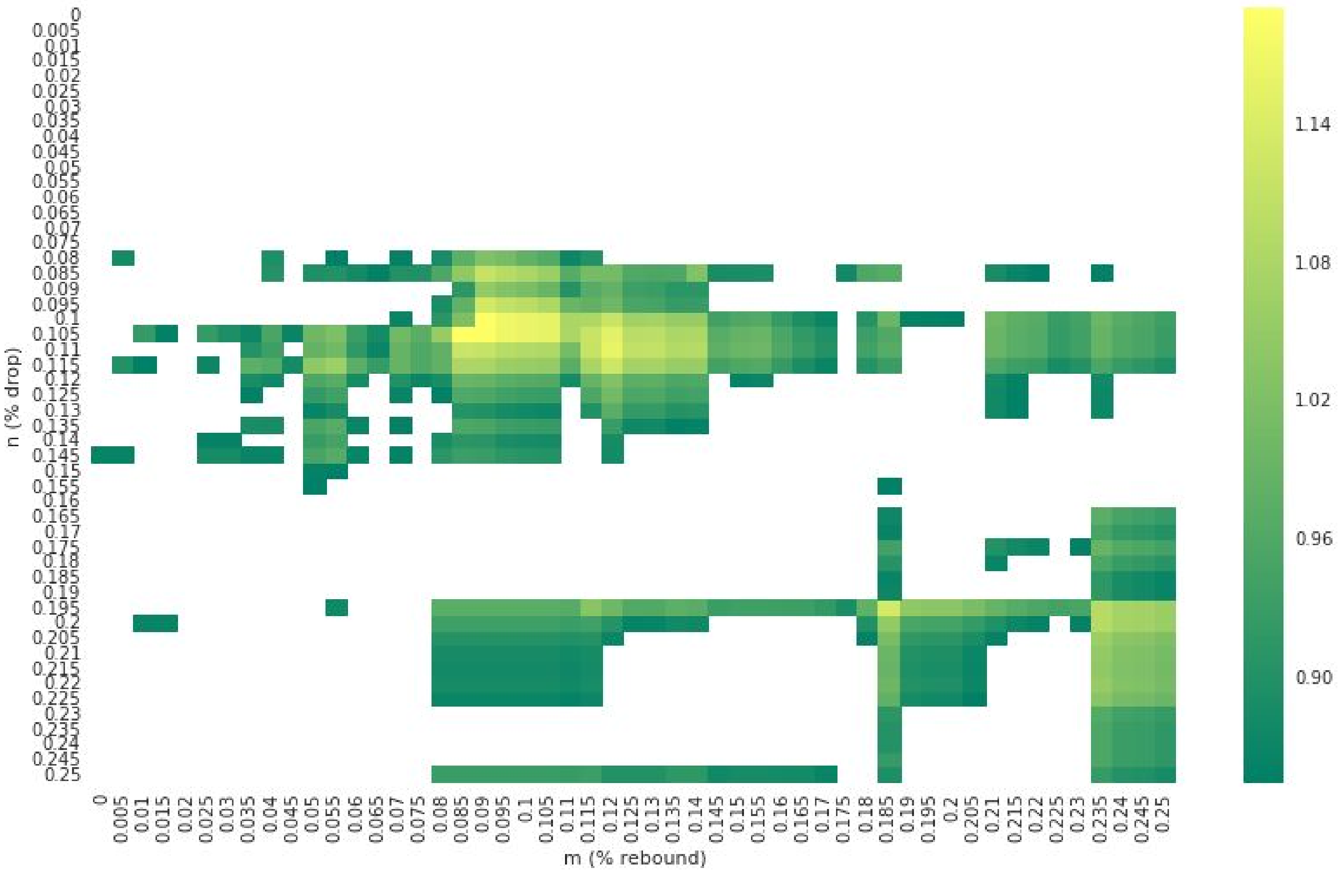
While we’re liquidated, why not pick up some risk-free returns?

Let’s return 2% annually on any cash we hold (to simulate buying treasury bills, etc.)

Another 2500 backtests

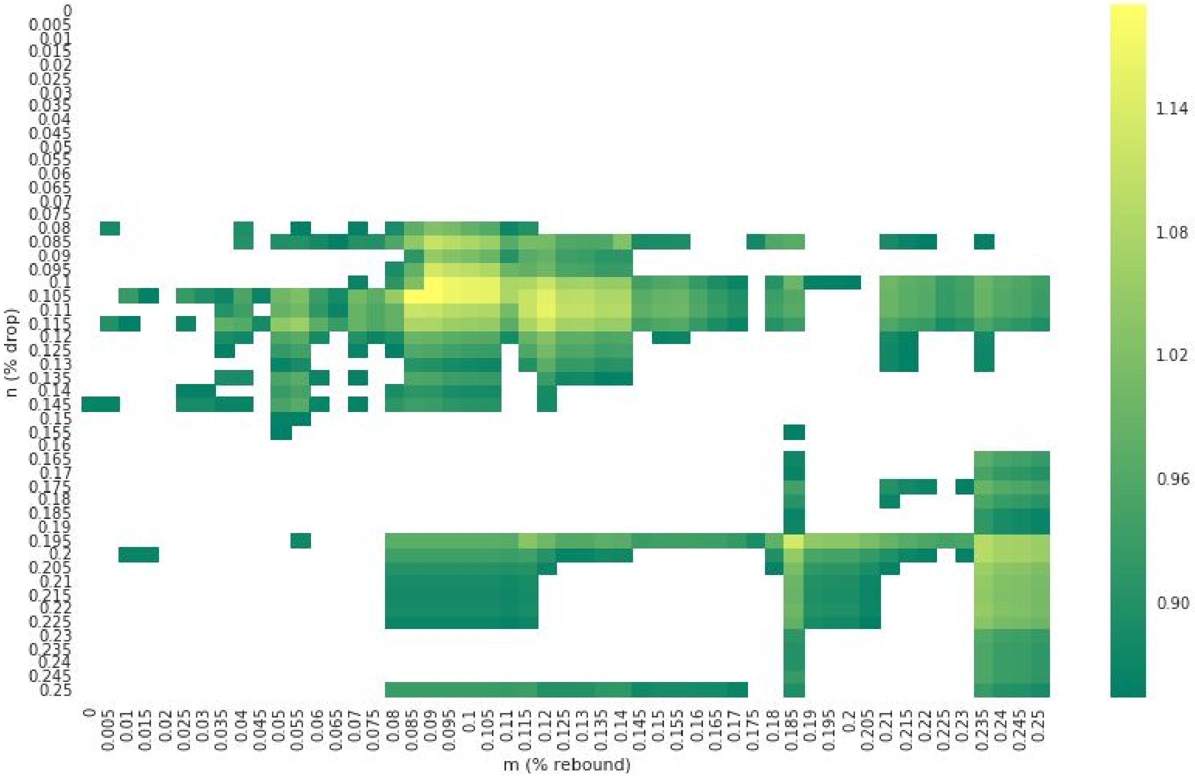


Threshold: Benchmark-Beating



Overfit to the Recession

Giveaway: best returns next to mediocre ones



Conclusions

- Parameter optimization is powerful, but dangerous!
- Next economic downturn: will it be structurally similar enough to the previous one for this model to be reliable? Probably not.
- Does there exist a simple strategy for lay investors to ensure market-beating returns over times of economic downturn? Maybe, but not this one.
- If a lay investor is holding SPY during an economic downturn, they may as well hold on.

Questions?

contact@johnloeber.com