Lecture 5: Backtesting Portfolio Strategies

Computational Finance

Portfolios and Portfolio Strategies

Portfolio Weights

- A **portfolio** is a collection of assets
  - Represented by a set of weights $\mathbf{w} = (w_1, \ldots, w_N)$
  - Weights w_i indicate how much you invest in asset i relative to other assets
- **Long** position: $w_i > 0$
  - **Long-only portfolio**: $w_i > 0$ for all assets i in the portfolio
- **Short** position: $w_i < 0$
  - Borrow asset i and sell it (then buy it back later at a hopefully lower price)
- One of the assets can be a risk-free asset (often called “cash”), in which case $w_f \neq 0$
  - $w_f > 0$: invest at the risk-free rate
  - $w_f < 0$: borrow at the risk-free rate – **leverage**

What do weights sum to? Typically 1 (or 100%)

- $\sum_{i=1}^{N} w_i = 1$
- Weights indicate what fraction of our investment is allocated to each asset
- E.g., I have $100 million to invest. I can invest in
  - Long-only portfolio: $60M in Microsoft, $40M in Walmart
    - Weights: $w_M = 0.6, w_W = 0.4$
  - Leveraged portfolio: $80M in Microsoft, $50M in Walmart, -$30M in the risk-free asset
    - Borrow $30M (“short cash”), combine with $100M, invest the total $130M
    - Weights: $w_M = 0.8, w_W = 0.5, w_f = -0.3$
  - Long-short portfolio: $80M in Microsoft, $50M in Walmart, -$30M in General Electric
    - Instead of borrowing $30M, short $30M of GE stock
    - **Long leg**: $w_M = 0.8, w_W = 0.5$ sums to 1.3
    - **Short leg**: $w_G = -0.3$

Self-financing portfolios: weights sum to 0

- What if I start with *none* of my own money?
  - Borrow \$100M, invest \$100M in the portfolio
  - Weights: $w_M = 0.6, w_W = 0.4, w_f = -1$
- Can sell risky assets short instead of borrowing to finance the long leg?
  - Sell \$130M short of GE, invest \$80M in Microsoft and \$50M in Walmart
  - $w_M = 0.8/1.3, w_W = 0.5/1.3, w_G = -1.3/1.3$
- **Self-financing portfolio**: Weights sum to 0
  - The short leg finances the long leg
  - Weights now represent the fraction of each *leg* allocated to each asset
    - Long leg weights sum to 1: $\sum_i w_i = 1$ for all $w_i > 0$
    - Short leg weights sum to -1: $\sum_i w_i = -1$ for all $w_i < 0$

Self-financing portfolios (continued)

- Any portfolio has a self-financing equivalent: finance it by borrowing at the risk-free rate
  - E.g., make $w_M = 0.6, w_W = 0.4$ a self-financing portfolio by adding $w_f = -1$
- How realistic are self-financing portfolios?
  - Would you lend me $100M cash to invest in a portfolio of risky assets with 0 of my own money?
  - Would you lend me $100M in shares of GE to invest in a portfolio of risky assets with 0 of my own money?
  - Nevertheless, useful theoretical construct to study the relative performance of different assets or portfolios
  - It makes money if and only if the long leg outperforms the short leg
  - E.g.: SMB (long small firms, short big firms), HML (long value firms, short growth firms)
- Connection with CAPM and APT: $E[R_p] - R_f = \beta(E[R_m] - R_f)$
  - Left-hand side: self-financing portfolio that borrows at R_f and invests in portfolio p earning $E[R_p]$ in expectation
  - Right-hand side: self-financing portfolio that borrows at R_f and invests in the market earning $E[R_m]$ in expectation

Portfolios vs. Portfolio Strategies

- Portfolio is a *static* concept
  - Weights describe how much we *currently* have invested in each firm
  - If we just bought the portfolio, or if we just re-balanced, then we chose the weights
    - Otherwise, they reflect the current allocations after changes since our last action
- Portfolio strategy is a *dynamic* concept
  - Approach to changing portfolio weights over time
  - Rebalancing: buying and selling assets to achieve a target portfolio
- A **rules-based portfolio strategy** follows a set of pre-specified rules to determine how to rebalance
  - Instead of the portfolio manager using their discretion and judgment to decide how to trade
  - E.g. fixed weights strategy: buy and hold the same portfolio weights forever
    - Even this simple strategy requires rebalancing! Since portfolio weights will change as prices change
  - E.g. buy all stocks in the S&P 500 in proportion to their market capitalization
    - Again, we need to rebalance to maintain these weights as market capitalization of each firm changes

Fixed Weights vs. Value Weights

▶ Code

- **Fixed weights strategy**: buy and hold the same portfolio weights forever
  - E.g. $w_M = 0.6, w_W = 0.4$
  - Holding weights fixed takes work! Need to rebalance to maintain fixed weights
  - Today, we use $100M to implement the strategy by buying
    - 200 thousand shares of Microsoft at $300 a share = $60M
    - 250 thousand shares of Walmart at $160 a share = $40M
  - In a month, Microsoft is at $350 a share, Walmart is at $170 a share
    - Value of Microsoft position: 200K $\times$ $350 = $70.0M
    - Value of the Walmart position: 250K $\times$ $170 = $42.5M
    - Microsoft is now 62.2% of the portfolio, Walmart is 37.8%
    - Need to sell 7,143 shares of Microsoft, buy 14,706 shares of Walmart to get back to 60/40 split
- **Value weights**: portfolio weights are proportional to the market cap of the stock
  - If MSFT stock price goes up by more than WMT, MSFT weight increases
  - A “set it and forget it” **passive** strategy

Backtesting

What is backtesting?

- Before implementing a rules-based strategy, useful to see if it would have been profitable in the past
  - Not a guarantee of future performance, but a useful check
- **Backtesting**: testing a rules-based portfolio strategy on historical data to see how it would have performed in the past
  - Requires historical data on asset returns and any additional information needed to implement the rule
  - E.g., strategy: every year form a value-weighted portfolio of the 10 firms with the largest sales
    - Need annual accounting data on sales for all firms to find the top 10
    - Need market capitalization data to form value weights
- Important to make sure you don't accidentally assume clairvoyance (ability to predict the future)
- Data used to form weights at time t should be available at time t
  - Otherwise, easy to construct a great strategy that buys stocks that will do well later!
  - As general notation, we can represent expected returns at the next period (e.g., the next month, day, year or whatever period we're measuring) using the conditional notation $E[R_{t+1}|I_t]$
    - Translated: "The expected return at time t+1 given the information available at time t"

Backtesting the SMB Strategy

- Our goal: use data from 1926-2022 to test the performance of the SMB strategy
  - Long smallest (bottom 80%) firms, short largest (top 10%) big firms
    - Similar to (but not exactly the same as) the Fama-French SMB factor
  - Form *value-weighted* portfolios based on market cap at the end of June, hold for a year
- How we will measure performance:
  - Average monthly returns
  - Monthly Sharpe ratio
  - Monthly CAPM alpha
- Data needed
  - Monthly stock returns on the universe of U.S. stocks from 1926-2022
  - Annual end-of-June market capitalization data for all stocks
  - all available on CRSP through **WRDS**

Approach: 3 steps

- Step 1: Identify stocks to invest in
  - Find 80th and 90th percentile cutoffs for market cap at the end of each June
  - Use cutoffs to identify stocks to invest in
- Step 2: Calculate weights and portfolio returns:
  - Given the stocks to invest in for that year,
  - Use end-of-June market caps to determine July weights and portfolio returns
  - Use end-of-July market caps to determine August weights and portfolio returns
  - … and so on until June of the next year
- Step 3: Calculate performance metrics
  - Absolute: Average returns, standard deviation (volatility), Sharpe ratio
  - Relative: Regress portfolio returns on market excess returns to get CAPM alpha

Step 1: Identify stocks to invest in (Example: 2022)

```
import pandas as pd
df = pd.read_csv('data/dfrebalance2022.csv')
df.head()
```

	date	permno	prc	shrout	ret	mktcap
0	2022-06-30	22765	0.800000	4741	0.125969	3792.800057
1	2022-06-30	22772	1.280000	3163	-0.268571	4048.639910
2	2022-06-30	91627	2.339900	1799	-0.120702	4209.480130
3	2022-06-30	85540	0.714799	5912	-0.140762	4225.891611
4	2022-06-30	16377	0.200000	22700	-0.428571	4540.000068

```
df['indicator'] = 0
df.loc[df['mktcap']<=df['mktcap'].quantile(0.8), 'indicator'] = 1   # long leg
df.loc[df['mktcap']>=df['mktcap'].quantile(0.9), 'indicator'] = -1  # short leg
```

Step 2: Calculate weights and portfolio returns

- Portfolio return: $R_{p,t} = \sum_i w_{i,t} R_{i,t}$
  - $R_{i,t}$: return on asset i in month t (e.g., July return is from June 30 to July 31)
  - $w_{i,t}$: weight on asset i in month t (e.g., **July** weight is based on **June** 30 market cap)
- Value-weighted portfolio: weights proportional to market cap
  - Weights for long leg: $w_{i,t} = \frac{mktcap_{i,t-1}}{\sum_{j \in \text{long leg}} mktcap_{j,t-1}}$
  - Weights for short leg: $w_{i,t} = -\frac{mktcap_{i,t-1}}{\sum_{j \in \text{short leg}} mktcap_{j,t-1}}$
  - Time t weights depend on market caps at time $t-1$: dynamic calculation
- Need to re-do every month
  - Value-weighted portfolio doesn't require trading during the year, but
  - Weights change every month: if stock A goes up by more than stock B, stock A's weight increases

Step 2 Example

▶ Code

- Example with a long-only portfolio
  - 3 Months: June, July, August
  - 2 Stocks: Microsoft (MSFT) and Nvidia (NVDA)
  - market cap (at the end of the month)
  - return (from previous month to current month)
- Calculate July portfolio return:
  - Weights based on June market cap
    - $w_{MSFT,7} = \frac{m_{MSFT,6}}{m_{MSFT,6}+m_{NVDA,6}} = \frac{2.4}{2.4+0.8} = 0.75$
    - $w_{NVDA,7} = \frac{m_{NVDA,6}}{m_{MSFT,6}+m_{NVDA,6}} = \frac{0.8}{2.4+0.8} = 0.25$
  - Portfolio return:
    - $R_{p,7} = w_{MSFT,7} \times R_{MSFT,7} + w_{NVDA,7} \times R_{NVDA,7}$
    - $R_{p,7} = 0.75 \times 0.125 + 0.25 \times 0.25 = 0.15625$

```
display(dfex)
```

	t	i	m	R
0	6	MSFT	2.4	0.060
1	6	NVDA	0.8	0.080
2	7	MSFT	2.7	0.125
3	7	NVDA	1.0	0.250
4	8	MSFT	2.6	-0.037
5	8	NVDA	1.1	0.100

Step 2 Example: Implementing in Python (1)

Calculate weights $w_{i,t}$ and components of the portfolio return $w_{i,t}R_{i,t}$

```
dfex['lagged_m'] = dfex.groupby('i')['m'].shift(1)
dfex['w'] = dfex['lagged_m'] / dfex.groupby('t')['lagged_m'].transform('sum')
dfex['Rp'] = dfex['w'] * dfex['R']
display(dfex)
```

```
   t     i    m      R  lagged_m        w        Rp
0  6  MSFT  2.4  0.060       NaN      NaN       NaN
1  6  NVDA  0.8  0.080       NaN      NaN       NaN
2  7  MSFT  2.7  0.125       2.4  0.75000  0.093750
3  7  NVDA  1.0  0.250       0.8  0.25000  0.062500
4  8  MSFT  2.6 -0.037       2.7  0.72973 -0.027000
5  8  NVDA  1.1  0.100       1.0  0.27027  0.027027
```

Step 2 Example: Implementing in Python (2)

Add up components of the portfolio return $w_{i,t}R_{i,t}$ within each month

```
portret = dfex.groupby('t')['Rp'].sum()
display(portret)
```

```
t
6    0.000000
7    0.156250
8    0.000027
Name: Rp, dtype: float64
```

- Even though June was (correctly) all `nan`s, `sum` ignored them and produced zeros.
- We will have to deal with this when we implement the strategy on the full dataset

Step 3: Calculate performance metrics

- Straightforward to calculate average returns and standard deviations
- Sharpe ratio
  - Earlier, we defined the Sharpe ratio as $\frac{E[R_p]-R_f}{\sqrt{Var[R_p]}}$
  - Now, we know that the numerator is the expected return on a self-financing strategy
    - Borrow at R_f, invest in portfolio p earning $E[R_p]$ in expectation
    - The denominator is the standard deviation of the self-financing strategy
  - SMB is already a self-financing portfolio, so its Sharpe ratio is just $\frac{E[R_p]}{\sqrt{Var[R_p]}}$
- CAPM alpha
  - Estimate: $R_{p,t} = \alpha_p + \beta_p(R_{m,t} - R_{f,t}) + \epsilon_{p,t}$
  - Left-hand side does not have $-R_{f,t}$ because SMB is already a self-financing portfolio

Let's implement the strategy!

See backtesting_smb.ipynb

- Uses a dataset of monthly stock returns from 1926-2022 in `data/stockdata.zip`
  - Big data! Almost 3 million rows, 47 MB compressed (158 MB uncompressed)
- Uses Professor Elenev's module `kenfrench` for importing the Fama-French factors from Ken French's website