columbia-final-project

November 28, 2023

1 Portfolio optimization example

1.1 Load Packages

[1]:
```
import pandas
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle
from statsmodels.formula.api import ols
from scipy.stats import gaussian_kde
import scipy
import scipy.sparse
import patsy
from statistics import median
import bz2
import math
```

1.2 Load Data

The model data has already been pre-processed from text files and saved into pickle files. Loading and parsing the raw files (parsing text into numbers) can slow down your backtest significantly, which is why it's important to pre-process the data beforehand.

[2]:
```
model_dir = '/Users/gordonritter/Dropbox/teaching/pickle_data/'

def sort_cols(test):
    return(test.reindex(sorted(test.columns), axis=1))

frames = {}
for year in [2004,2005,2006]:
    fil = model_dir + "pandas-frames." + str(year) + ".pickle.bz2"
    frames.update(pd.read_pickle(fil))

for x in frames:
    frames[x] = sort_cols(frames[x])

covariance = {}
```

```
for year in [2003,2004,2005,2006]:
    fil = model_dir + "covariance." + str(year) + ".pickle.bz2"
    covariance.update(pd.read_pickle(fil))
```

ID: a unique identifier that can be used to link stocks across time

1DREVRSL: very short-term reversal, potential alpha factor but probably too fast-moving to be tradable

STREVRSL: short-term reversal, potential alpha factor

LTREVRSL: long-term reversal, potential alpha factor

BETA: risk factor computed from CAPM beta regression

EARNQLTY: earnings quality, potential alpha factor

EARNYILD: earnings yield (blend of forecasted earnings and historical earnings divided by market cap)

GROWTH: mix of historical and forecasted earnings growth

INDMOM: industry momentum (defined as relative historical outperformance or underperformance of the other stocks in the same industry)

LEVERAGE: financial leverage of the company's balance sheet, usually a risk factor

LIQUIDTY: factor with high loadings for very liquidly traded names; usually a risk factor

MGMTQLTY: management quality, potential alpha factor which looks at quantitative measures of how well-run a company is by its management

MOMENTUM: 12-month growth in stock price, usually a risk factor

PROFIT: profitability, potential alpha factor

PROSPECT: based on skewness of the return distribution, potential risk factor

RESVOL: risk factor computed from residual volatility

SEASON: seasonality-based alpha factor

SENTMT: news sentiment alpha factor

SIZE: risk factor based on log(market capitalization)

VALUE: risk factor based on ratio of tangible book value to current price

SpecRisk: specific risk is another name for predicted residual volatility. We called this the D matrix in our discussion of APT models.

TotalRisk: predicted total vol, including factor and idiosyncratic contributions, annualized

Ret: asset's total return on the next day after the factor loadings are known, suitable as the Y vector in a regression analysis

Yield: the dividend yield of the asset

HistBeta: historically estimated CAPM beta coefficient PredBeta: model-predicted beta coefficient in the future

IssuerMarketCap: aggregate market capitalization of the company (all share classes from the same issuer, e.g. for Google would include both Alphabet A and C)

BidAskSpread: bid-offer spread (average for the day)

CompositeVolume: composite trading volume for the day

DataDate: the date when the data would have been known, as of the close

Many of the remaining columns are industry factors, of which a full list is given below.

[3]:
```
industry_factors = ['AERODEF', 'AIRLINES', 'ALUMSTEL', 'APPAREL', 'AUTO',
       'BANKS','BEVTOB', 'BIOLIFE', 'BLDGPROD','CHEM', 'CNSTENG',
        'CNSTMACH', 'CNSTMATL', 'COMMEQP', 'COMPELEC',
       'COMSVCS', 'CONGLOM', 'CONTAINR', 'DISTRIB',
       'DIVFIN', 'ELECEQP', 'ELECUTIL', 'FOODPROD', 'FOODRET', 'GASUTIL',
       'HLTHEQP', 'HLTHSVCS', 'HOMEBLDG', 'HOUSEDUR','INDMACH', 'INSURNCE',␣
 ↪'INTERNET',
        'LEISPROD', 'LEISSVCS', 'LIFEINS', 'MEDIA', 'MGDHLTH','MULTUTIL',
       'OILGSCON', 'OILGSDRL', 'OILGSEQP', 'OILGSEXP',
        'PAPER', 'PHARMA', 'PRECMTLS','PSNLPROD','REALEST',
       'RESTAUR', 'ROADRAIL','SEMICOND', 'SEMIEQP','SOFTWARE',
        'SPLTYRET', 'SPTYCHEM', 'SPTYSTOR', 'TELECOM', 'TRADECO', 'TRANSPRT',␣
 ↪'WIRELESS']

style_factors = ['BETA','SIZE','MOMENTUM','VALUE','LEVERAGE','LIQUIDTY']
```

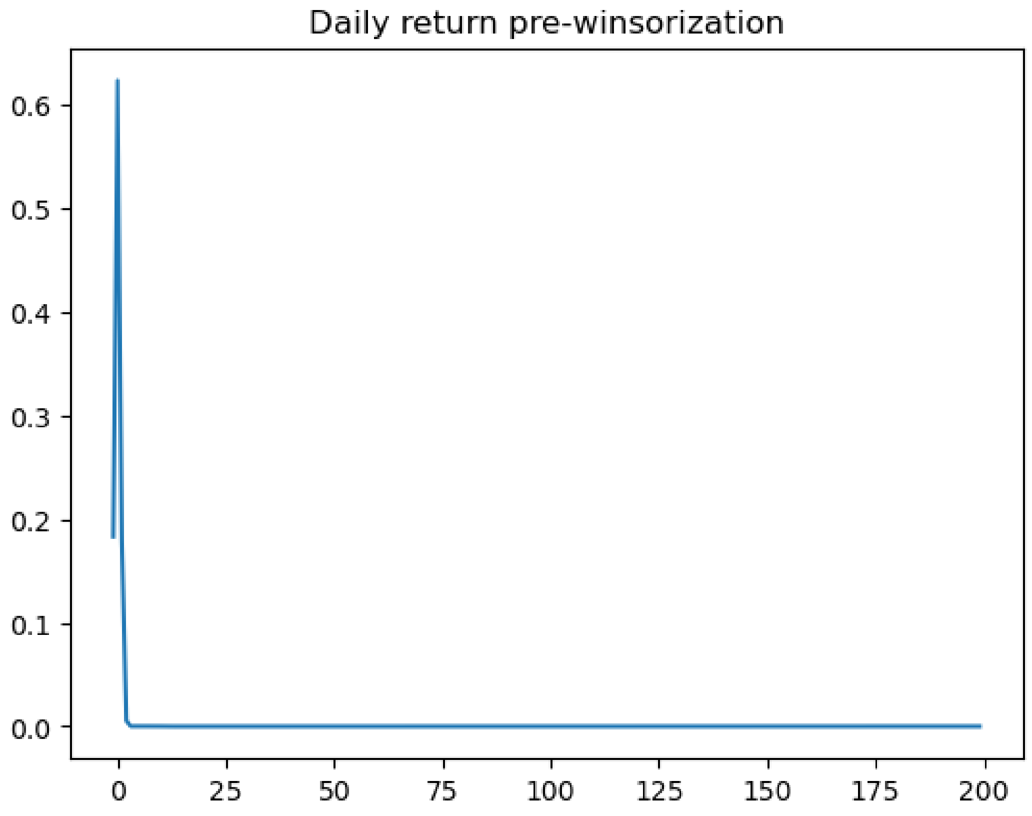
1.3 Data Cleaning and Winsorization

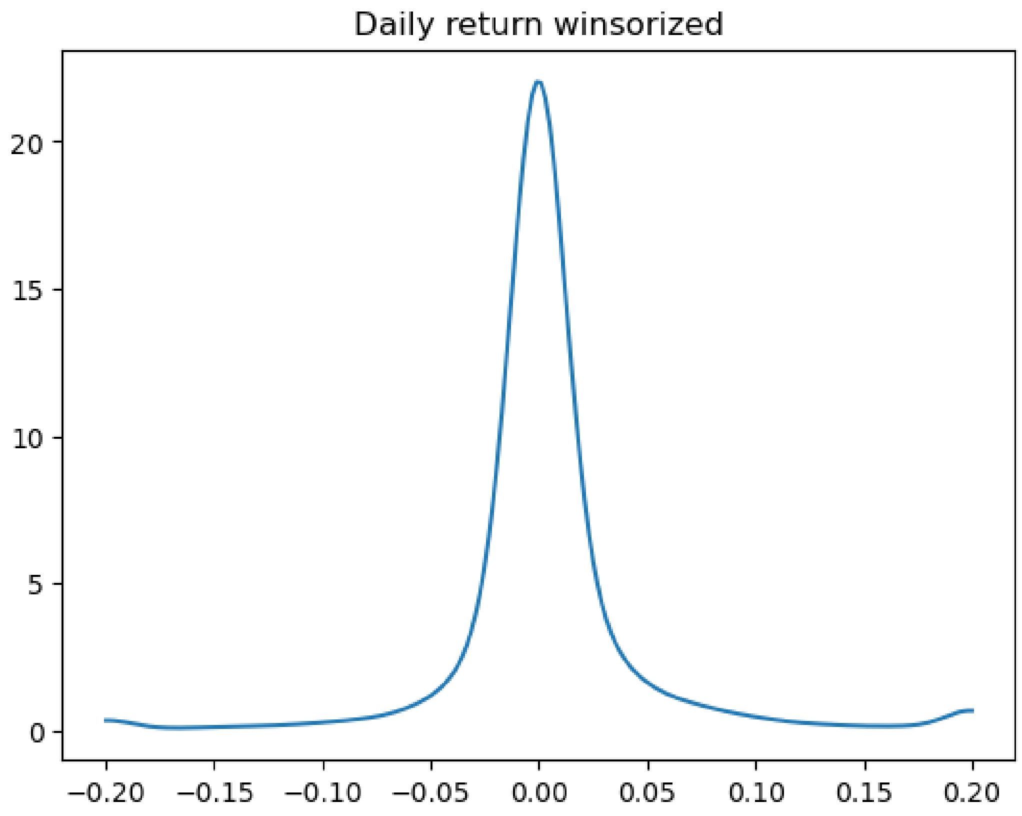
The distribution of many statistics can be heavily influenced by outliers. A simple approach to robustifying parameter estimation procedures is to set all outliers to a specified percentile of the data; for example, a 90% winsorization would see all data below the 5th percentile set to the 5th percentile, and data above the 95th percentile set to the 95th percentile. Winsorized estimators are usually more robust to outliers than their more standard forms.

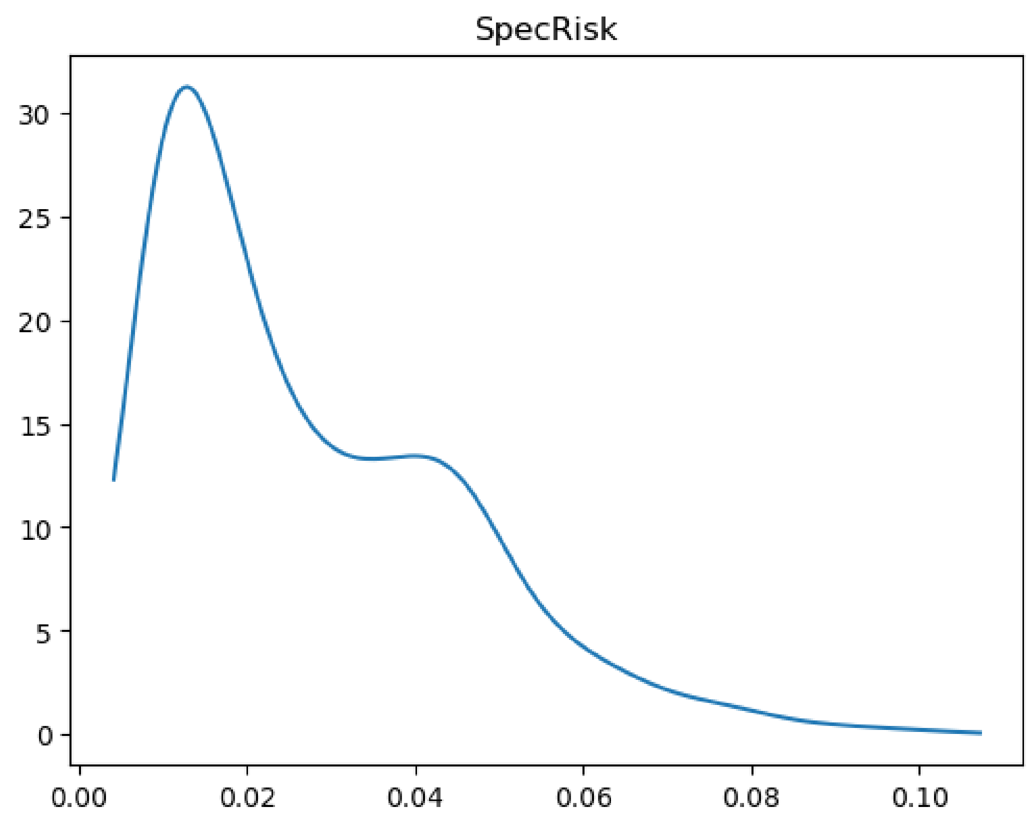
[4]:
```
def wins(x,a,b):
    return(np.where(x <= a,a, np.where(x >= b, b, x)))

def clean_nas(df):
    numeric_columns = df.select_dtypes(include=[np.number]).columns.tolist()

    for numeric_column in numeric_columns:
        df[numeric_column] = np.nan_to_num(df[numeric_column])

    return df
```

We can check the distribution of returns with a density plot, both before and after winsorization to observe the effect of trimming the tails.

[5]:
```
def density_plot(data, title):
    density = gaussian_kde(data)
    xs = np.linspace(np.min(data),np.max(data),200)
    density.covariance_factor = lambda : .25
    density._compute_covariance()
    plt.plot(xs,density(xs))
    plt.title(title)
    plt.show()

test = frames['20040102']
density_plot(test['Ret'], 'Daily return pre-winsorization')
density_plot(wins(test['Ret'],-0.2,0.2), 'Daily return winsorized')

D = (test['SpecRisk'] / (100 * math.sqrt(252))) ** 2
density_plot(np.sqrt(D), 'SpecRisk')
```




Daily return winsorized
20
15
10
5
0
−0.20
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15
0.20



2 Factors

2.1 Factor Exposures and Factor Returns

Arbitrage pricing theory relaxes several of the assumptions made in the course of deriving the CAPM. In particular, we relax the assumption that all investors do the same optimization and hence that there is a single efficient fund. This allows the possibility that a CAPM-like relation may hold, but with multiple underlying sources of risk.

Specifically, let $r_i, i = 1, \dots, n$ denote the cross-section of asset returns over a given time period $[t, t+1]$. In a fully-general model, the multivariate distribution $p(\mathbf{r})$ could have arbitrary covariance and higher-moment structures, but remember that for n large there is typically never enough data to estimate such over-parameterized models.

Instead, we assume a structural model which is the most direct generalization of the CAPM:

$$r_i = \beta_{i,1} f_1 + \beta_{i,2} f_2 + \cdots + \beta_{i,p} f_p + \epsilon_i, \quad \epsilon_i \sim N(0, \sigma_i^2)$$

If $p = 1$, this reduces to the Capital Asset Pricing Model (CAPM) in a rather direct way.

With $p > 1$, the model starts to differ from the CAPM in several very important aspects. In the CAPM, we were able to identify the single efficient fund by arguing that its weights must equal the market-capitalization weights. Hence we were given for free a very nice proxy for the single efficient fund: a capitalization-weighted basket such as the Russell 3000. Hence in the $p = 1$ case we had a convenient proxy which could be used to impute the return f_1, which we called r_M. Also $\beta_{i,1}$ could be estimated, with no more than the usual statistical estimation error, by time-series regression.

If $p > 1$ then the underlying assumptions of that argument break down: there is no longer any simple way to identify f_j nor $\beta_{i,j}$ $(j = 1, \ldots, p)$. We shall return to the estimation problem in due course.

To avoid confusion with the CAPM, and its simplistic β coefficient (which is still sometimes used in larger multi-factor models), it is conventional to make the following notation change: $\beta_{i,j}$ becomes $X_{i,j}$ and so the model equation becomes

$$r_i = X_{i,1}f_1 + X_{i,2}f_2 + \cdots + X_{i,p}f_p + \epsilon_i, \quad \epsilon_i \sim N(0, \sigma_i^2)$$

It's difficult to simultaneously estimate both all components $X_{i,j}$ and all risk-source returns f_j, so one usually assumes one is known and calculates the other via regression. In what follows, we focus on the approach where X is known, and the f_j are assumed to be hidden (aka latent) variables.

The structural equation is more conveniently expressed in matrix form:

$$R_{t+1} = X_t f_{t+1} + \epsilon_{t+1}, \quad E[\epsilon] = 0, \quad V[\epsilon] = D$$

where R_{t+1} is an n-dimensional random vector containing the cross-section of returns in excess of the risk-free rate over some time interval $[t, t+1]$, and X_t is a (non-random) $n \times p$ matrix that can be calculated entirely from data known before time t. The variable f in denotes a p-dimensional random vector process which cannot be observed directly.

Since the variable f denotes a p-dimensional random vector process which cannot be observed directly, information about the f-process must be obtained via statistical inference. We assume that the f-process has finite first and second moments given by

$$E[f] = \mu_f, \quad \text{and} \quad V[f] = F.$$

The primary outputs of a statistical inference process are the parameters μ_f and F, and other outputs one might be interested in include estimates of the daily realizations $\hat{f}_{t+1}$.

The simplest way of estimating historical daily realizations of $\hat{f}_{t+1}$ is by least-squares (ordinary or weighted, as appropriate), viewing the defining model equation as a regression problem.

[6]:
```python
def get_estu(df):
    """Estimation universe definition"""
    estu = df.loc[df.IssuerMarketCap > 1e9].copy(deep=True)
    return estu

def colnames(X):
    """ return names of columns, for DataFrame or DesignMatrix """
    if(type(X) == patsy.design_info.DesignMatrix):
        return(X.design_info.column_names)
    if(type(X) == pandas.core.frame.DataFrame):
        return(X.columns.tolist())
    return(None)
```

```python
def diagonal_factor_cov(date, X):
    """Factor covariance matrix, ignoring off-diagonal for simplicity"""
    cv = covariance[date]
    k = np.shape(X)[1]
    Fm = np.zeros([k,k])
    for j in range(0,k):
        fac = colnames(X)[j]
        Fm[j,j] = (0.01**2) * cv.loc[(cv.Factor1==fac) & (cv.
 ↪Factor2==fac),"VarCovar"].iloc[0]
    return(Fm)

def risk_exposures(estu):
    """Exposure matrix for risk factors, usually called X in class"""
    L = ["0"]
    L.extend(style_factors)
    L.extend(industry_factors)
    my_formula = " + ".join(L)
    return patsy.dmatrix(my_formula, data = estu)
```

2.1.1 Helpful code to show how to get X, F, D matrices

[7]:
```python
my_date = '20040102'

# estu = estimation universe
estu = get_estu(frames[my_date])
estu['Ret'] = wins(estu['Ret'], -0.25, 0.25)

rske = risk_exposures(estu)
F = diagonal_factor_cov(my_date, rske)
X = np.asarray(rske)
D = np.asarray( (estu['SpecRisk'] / (100 * math.sqrt(252))) ** 2 )

kappa = 1e-5

candidate_alphas = [
    'STREVRSL', 'LTREVRSL', 'INDMOM',
    'EARNQLTY', 'EARNYILD', 'MGMTQLTY', 'PROFIT', 'SEASON', 'SENTMT']
```

2.1.2 Problem 0.

All of the below pertain to the estimation universe as defined above. Modify the daily data frames, removing all non-estimation-universe rows, before continuing.

2.1.3 Problem 1.

Residual returns Within each daily data frame, let Y denote the residuals of the variable `Ret`, with respect to the risk model. In other words, define

$$Y := \text{Ret} - XX^{+}\text{Ret}$$

where X^{+} denotes the pseudoinverse, and X is constructed as above (ie. using the risk_exposures function). Augment the data frames you have been given, by adding a new column, Y, to each frame. Be sure to winsorize the `Ret` column prior to computing Y as above. You do not have to save the augmented data, unless you want to. In other words, the modification that adds column Y can be done in-memory.

2.1.4 Problem 2.

Model selection Split your data into a training/validation set D_{train}, and an ultimate test set (vault), D_{test}. Do not split within a single day; rather, some dates end up in D_{train} and the rest in D_{test}. This will be the basis of your cross-validation study later on.

It will be helpful to join together vertically the frames in the training/validation set D_{train} into a single frame called a panel. For the avoidance of doubt, the panel will have the same columns as any one of the daily frames individually, and the panel will have a large number of rows (the sum of all the rows of all the frames in D_{train}).

Consider list of candidate alpha factors given above. Find a model of the form

$$Y = f(\text{ candidate alphas }) + \epsilon$$

where Y is the residual return from above. Determine the function $f()$ using cross-validation to optimize any tunable hyper-parameters. First, to get started, assume f is linear and use lasso or elastic net cross-validation tools (e.g. from sklearn). Then, get creative and try at least one non-linear functional form for f, again using cross-validation to optimize any tunable hyper-parameters.

2.1.5 Problem 3.

Efficient portfolio optimization Code up the efficient formula for portfolio optimization discussed in lecture, based on the Woodbury matrix inversion lemma.

2.1.6 Problem 4.

Putting it all together Using the helpful code example above, and using the output of the function f as your final alpha factor, construct a backtest of a portfolio optimization strategy. In other words, compute the optimal portfolio each day, and dot product it with `Ret` to get the pre-tcost 1-day profit for each day. Use the previous problem to speed things up. Create time-series plots of the long market value, short market value, and cumulative profit of this portfolio sequence. Also plot the daily risk, in dollars, of your portfolios and the percent of the risk that is idiosyncratic.