Machine-learning for Portfolio Management and Trading: An Introduction with Scikit-Learn

Sylvain Champonnois[1]

December 7, 2023

[1] sylvain.champonnois@gmail.com, latest version at https://github.com/schampon/skfin.

Contents

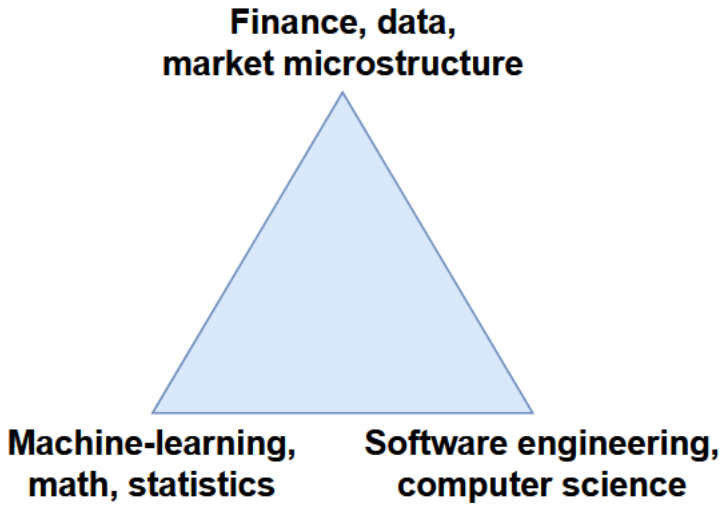
Chapter 1

Introduction

1.1 What hedge funds do

This material is an introduction to using machine-learning for portfolio management and trading. Given the centrality of programming in hedge funds today, the concepts are exposed using only `jupyter` notebooks in `python`. Moreover, we leverage the `scikit-learn` package (also known as `sklearn`) to illustrate how machine-learning is used in practice in this context.

As shown by the illustration below, we cover here topics that span Finance (market microstructure, portfolio construction, data, etc), Machine-learning (including mathematics and statistics) and Software engineering/Computer science.

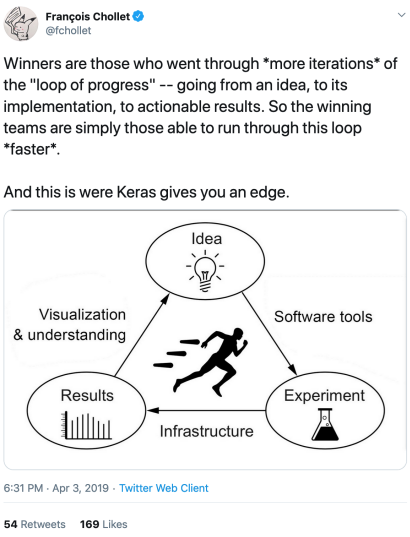


We are interested in how quantitative hedge funds operate in practice. Today, quantitative hedge funds are essentially *consumers* of data – they ingest all sorts of datasets and extract information used to systematically buy or sell securities. Researchers and portfolio managers are deeply involved in the process of `data ingestion` and `information extraction`, but they do not directly decide which securities are bought or sold – instead algorithms do.

Because these processes of `data ingestion` and `information extraction` are so central to quantitative hedge fund operations, they have become `software companies` – a lot of the intellectual

property (IP) of hedge funds is embedded in the code they write. And in that sense, hedge funds are not so different from other data-science based technology companies. (And in fact the hiring has become very similar, with a lot of interest in profiles out of Computer Science, Machine-Learning, Data engeeniring, Statistics, etc).

Another point is the Hedge Fund industry is very competitive and given that the frontier of knowledge is today moving very quickly (because of active research in Data, ML, hardware, software, etc) not keeping up with progress runs the risk of lagging by the competition. More precisely, innovation implies comming up with new ideas, testing them, deploying these ideas in trading systems; learning from them – and then deploying newer ideas after that. Of course, this kind of "loop of progress" (see Francois Chollet (04/03/2019)) is common to all AI-impacted industries.

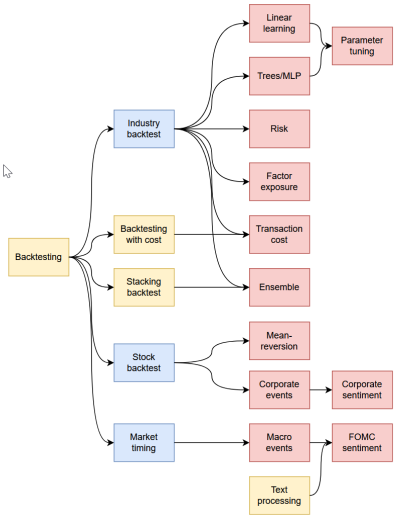


1.2 Overview

Across notebooks, we cover three main empirical use-cases: timing industry returns; timing stock returns; and timing the market (ie. timing the Standard and Poor's 500 index). For each use-case, there is a set of toy-datasets (described in the `Data` section).

Moreover:

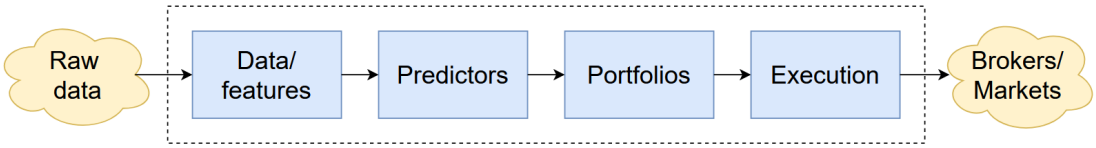
- for the industry return backtest, we illustrate several main concepts: learning (with linear models, boosted trees, and MLP neural nets); risk; factor exposures; transaction costs; ensemble;
- for the stock return backtest, we illustrate mean-reversion, trading around corporate events and sentiment (in earning calls);
- for the market timing backtest, we also illustrate traing around macroeconomic events, and extracting sentiment in the statements of the Federal Open Market Committee (FOMC).



1.3 Quant workflow

The graph below shows the typical data workflow of a quantitative fund:

- Raw data is acquired generally by a Data team and possibly transformed into usable features.
- Data quality is checked, in particular that timestamp of each observation is valid and has not been revised in a way that injects future information.
- From these features, predictors of asset returns are derived.
- Given a single predictor (or a set of many predictors), porfolios are constructed: these portfolios represent the ideal positions of a fund given the asset forecasts, but also risk forecasts (and possibly, transaction cost forecasts).
- When these ideal positions change from one day to the next (because the underlying data has been updated), the difference in positions initiate trades that are then executed on asset exchanges or with brokers.



1.4 MLOps for backtests

MLOps (machine learning operations) represents a set of practices for the deployment of ML models in production. For quant hedge funds, there are two main concepts that we describe here:

- pipelines
- backtests

1.4.1 Pipelines

Pipeline:

> A machine-learning pipeline is an end-to-end description of the automated flow of data from raw inputs to a desired output. Each step represents a transformation of the data, possibly with a fitted model.

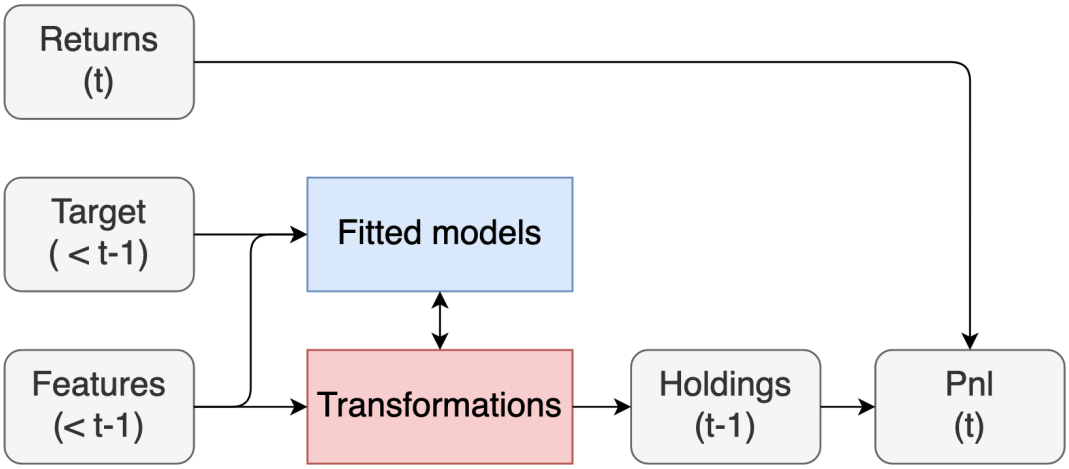
The diagram below illustrates a pipeline for a quant fund. The end point (to the right of the diagram) are the positions or `holdings` in a set of traded securities – and combined with the returns on these securities, the pnl of a given strategy can be computed. The entry point (to the left of the diagram) are `features`. A set of `transformations` (pre-determined in the `pipeline`) are applied to these `features` to produce the desired `holdings`. Some `transformations` in the `pipeline` are "fixed" while others depend on `fitted models` (e.g. a ML predictor of returns or a risk model).

In the diagram, we emphasize the timing of these different objects:

- for a pnl at time t, the `features` and `target` include only information up to $t-1$ so that the holdings are known in $t-1$ and can accrue returns over the period t.

The following equation summarizes this point:

$$pnl_t = holdings_{t-1} \times returns_t.$$



1.4.2 Scikit-learn

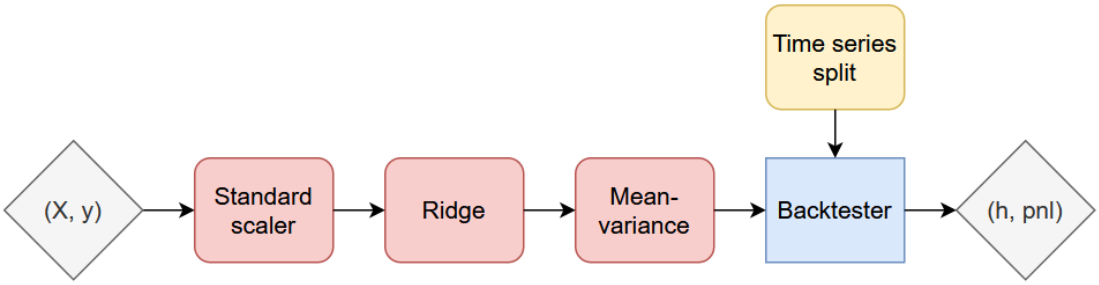
The following notebooks and notes are largely based on `scikit-learn`. `scikit-learn` is an extremely powerful (and widely used) package for machine-learning. In particular, it provides a "grammar" for pipelines where each transformation or estimator class has the `fit/transform/predict` functions with arguments as (`X`, `y`) where `X` represents the features and `y`, targets.

A look-ahead bias occurs when data dated at t includes information only available after t; in contrast, point-in-time data ensures that data dated at t is based on only information up to date t. A backtest is a method to simulate a strategy using point-in-time historical data and evaluate its profitability.

In order to illustrate how to use pipelines à la `scikit-learn` for quantitative portfolio management, we introduce in the next sections two objects:

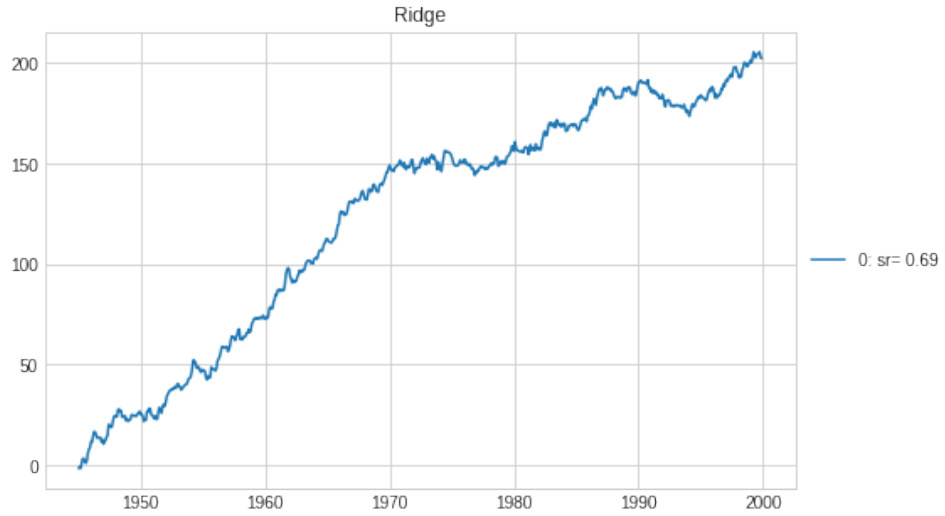
- a `Mean-variance` estimator that computes positions from a predictor;
- a `Backtester` class that fits a `scikit-learn` estimator over rolling windows (as defined by the `TimeSeriesSplit` class).

The `Backtester` class runs the rolling window simulation so that only information up to date $t-1$ is used to determined the holdings at that date. The following graph shows an example of a learning pipeline (with a `StandardScaler` and `Ridge` steps before compute the positions with the `Mean-variance` estimator) that we will use.



[8]:
```
from skfin import Backtester, MeanVariance, Ridge
from skfin.datasets import load_kf_returns
from skfin.plot import line
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

estimator = make_pipeline(StandardScaler(with_mean=False), Ridge(), MeanVariance())

returns_data = load_kf_returns(cache_dir="data")
ret = returns_data["Monthly"]["Average_Value_Weighted_Returns"][:"1999"]

transform_X = lambda x: x.rolling(12).mean().fillna(0)
transform_y = lambda x: x.shift(-1)
features = transform_X(ret)
target = transform_y(ret)

pnl_ = Backtester(estimator).train(features, target, ret)
line(pnl_, cumsum=True, title="Ridge")
```

```
INFO:skfin.datasets:logging from cache directory: data/12_Industry_Portfolios
```



The graph above shows the annualized sharpe ratio (as `sr`): given that the strategy is monthly, this is the montly pnl average divided by the monthly pnl standard deviation and "annualized" by multiplying by the square-root of the number of observation per year ($\sqrt{12}$).

1.5 Python package

All the classes and functions used across notebooks are stored in `.py` files using the `jupyter magic`: `%%writefile`. And these files are structured as a repository on `github` that can be cloned from the command line (once in the correct directory):
`> git clone https://github.com/schampon/skfin.git && cd skfin`

The following script helps create an environment with the proper packages:

[9]:
```
%%writefile ../create_env.sh
conda create python=3.9 --name  skfin -c https://conda.anaconda.org/conda-forge/ -y
conda activate skfin

pip install -r requirements.txt
pip install -e .
python -m ipykernel install --user --name skfin --display-name "Python (skfin)"
```

```
Overwriting ../create_env.sh
```

In practice, this is done from the command line with:

```
./create_env.sh
```

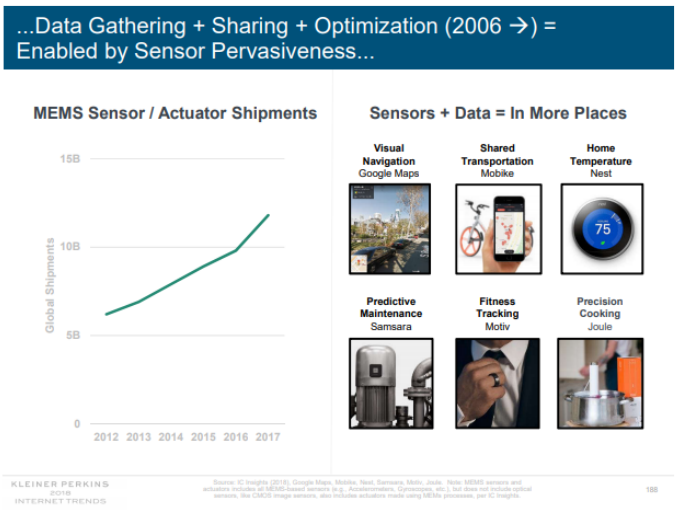
Chapter 2

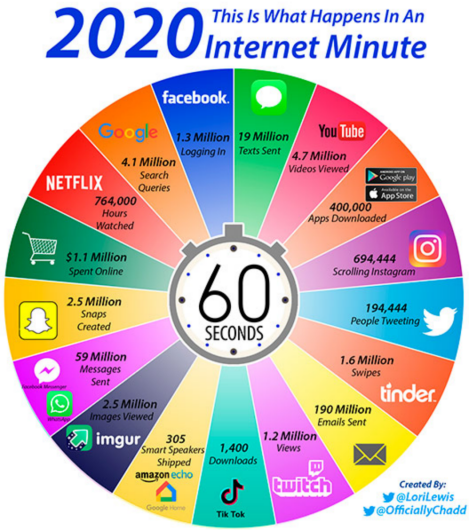
Data

In the prevous section, we described hedge funds, and more generally the buy-side industry, as `consumers of data`. In practice hedge funds have been a beneficiary of the exponential growth in the number of available datasets. We first describe the main classes of datasets that have been made available in the industry. We then introduce some of the illustrative `toy` datasets that will be used across notebooks.

2.1 Data deluge

There are now sensors everywhere in the physical world and most of the online interactions are tracked – leading a `data deluge` (e.g. see Mary Meeker (2018) on internet trends or Lori Lewis (2022)). On the one hand, the cost of sensors has dropped so much that these sensors can be put practically everywhere and record data from the physicial world. On the other hand, the volume and velocity of data only has also exploded.



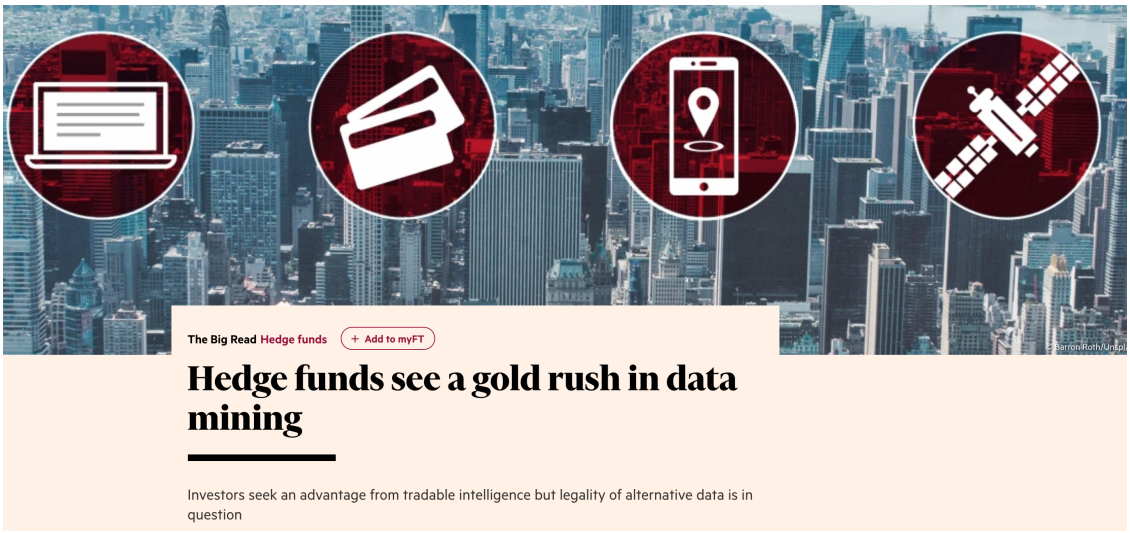


2.1.1 Data scouting

The article "Hedge funds see a gold rush is data mining" FT (08/28/2017) describes use-cases associated with four types of alternative data:

- web traffic
- credit card transaction
- geolocalisation
- satellite imaging

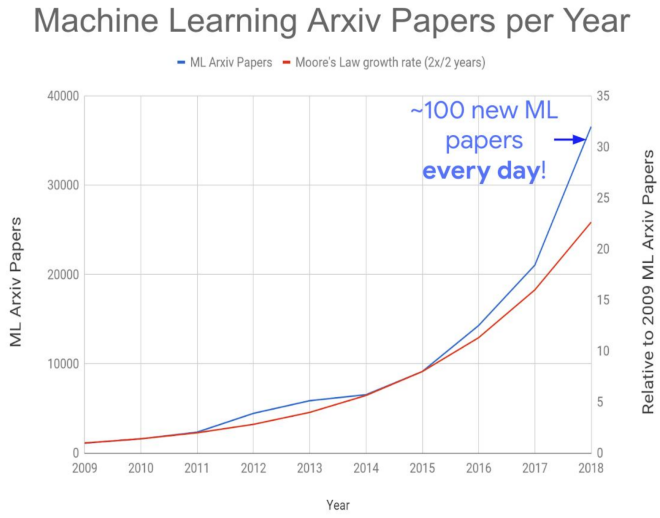
In each of these cases, the main idea is to use alternative data to get a noisy, but real-time forecast of company fundamentals (e.g. quarterly sales).



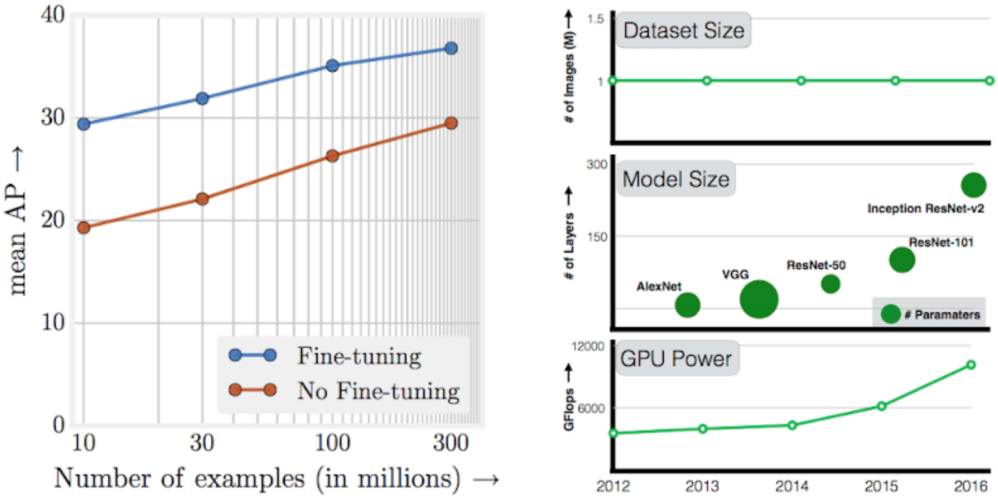
Obviously, predicting company fundamentals is only part of the story of what drives stock prices in financial markets. On the other side, understanding the market participants, in particular retail traders, is also important as shown in China Bloomberg (06/15/2019) or in the US with the Game Stop saga FT(01/29/2021).

2.1.2 ML research

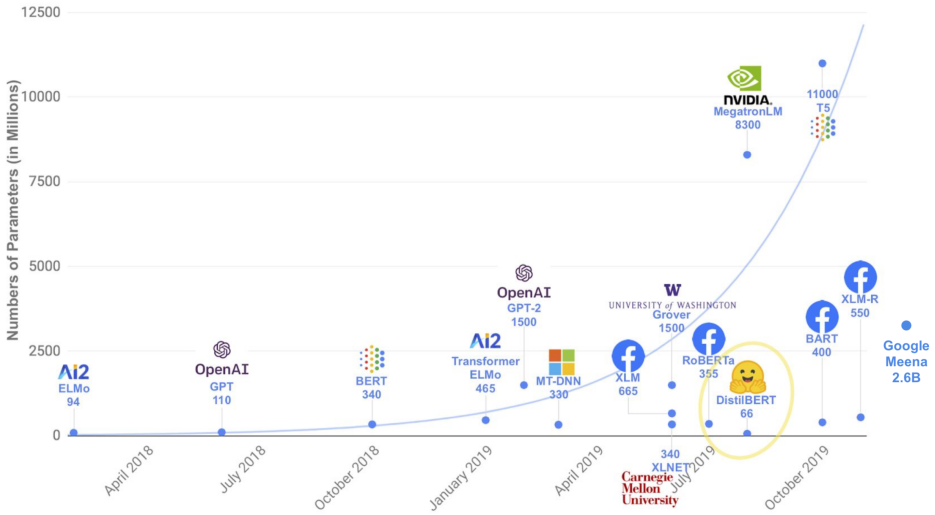
ML research has become a race with new ideas coming out with an increasing speed – e.g. as illustrated by the number of papers published on the scientific paper repository arxiv.com (Jeff Dean (06/02/2019)).



The success of deep-learning depends on: i) model capacity, ii) computational power, iii) dataset size. Sun, Shrivastava, Singh, Gupta (2017) note that the size of the largest dataset has remained somewhat constant over the last few years.



A particular success of deep-learning has been on Natural Language Processing (NLP) and there too, the size of the largest models has increased dramatically. Victor Sanh, Lysandre Debut, Julien Chaumond, Thomas Wolf (2019).



2.2 Datasets

In particular, we look at several sources of data:

- Ken French's data library
- Berkshire Hathaway financials
- Stock returns
- Federal Open Market Committee (FOMC) statements
- the sentiment dictionary created by Loughran and McDonalds
- US filings to the U.S. Securities and Exchange Commission (SEC) called 10Qs and 10Ks
- Long term stock market data put together by Amit Goyal

2.2.1 data helper functions

In this section, we use the `jupyter` command `%%writefile` to construct a module of data helper function.

[8]:
```
%%writefile ../skfin/datasets.py
import logging
import os
import re
import subprocess
import sys
import warnings
from io import BytesIO
from zipfile import ZipFile

import numpy as np
import pandas as pd
import requests
from bs4 import BeautifulSoup
from tqdm.auto import tqdm
```

```
warnings.filterwarnings("ignore", category=UserWarning, module="openpyxl")

from skfin.data_utils import clean_directory_path, load_dict, save_dict
from skfin.dataset_dates import load_fomc_change_date
from skfin.dataset_mappings import mapping_10X, symbol_dict

logging.basicConfig(stream=sys.stdout, level=logging.INFO)
logger = logging.getLogger(__name__)


def clean_kf_dataframes(df, multi_df=False):
    """
    extract the annual and monthly dataframes from the csv file with specific formatting
    """
    idx = [-2] + list(np.where(df.notna().sum(axis=1) == 0)[0])
    if multi_df:
        cols = ["  Average Value Weighted Returns -- Monthly"] + list(
            df.loc[df.notna().sum(axis=1) == 0].index
        )
    returns_data = {"Annual": {}, "Monthly": {}}
    for i in range(len(idx)):
        if multi_df:
            c_ = (
                cols[i]
                .replace("-- Annual", "")
                .replace("-- Monthly", "")
                .strip()
                .replace("/", " ")
                .replace(" ", "_")
            )
        if i != len(idx) - 1:
            v = df.iloc[idx[i] + 2 : idx[i + 1] - 1].astype(float)
            v.index = v.index.str.strip()
            if len(v) != 0:
                if len(v.index[0]) == 6:
                    v.index = pd.to_datetime(v.index, format="%Y%m")
                    if multi_df:
                        returns_data["Monthly"][c_] = v
                    else:
                        returns_data["Monthly"] = v
                    continue
                if len(v.index[0]) == 4:
                    v.index = pd.to_datetime(v.index, format="%Y")
                    if multi_df:
                        returns_data["Annual"][c_] = v
                    else:
                        returns_data["Annual"] = v
    return returns_data


def load_kf_returns(
    filename="12_Industry_Portfolios", cache_dir="data", force_reload=False
):
    """
    industry returns from Ken French:
    https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html
    """

    if filename == "12_Industry_Portfolios":
        skiprows, multi_df = 11, True
```

```
    if filename == "F-F_Research_Data_Factors":
        skiprows, multi_df = 3, False
    if filename == "F-F_Momentum_Factor":
        skiprows, multi_df = 13, False
    if filename == "F-F_Research_Data_Factors_daily":
        skiprows, multi_df = 4, False

    output_dir = clean_directory_path(cache_dir) / filename
    if (output_dir.is_dir()) & (~force_reload):
        logger.info(f"logging from cache directory: {output_dir}")
        returns_data = load_dict(output_dir)
    else:
        logger.info("loading from external source")
        path = (
            "http://mba.tuck.dartmouth.edu/pages/faculty/ken.french/ftp/"
            + filename
            + "_CSV.zip"
        )
        r = requests.get(path)
        files = ZipFile(BytesIO(r.content))

        df = pd.read_csv(files.open(filename + ".CSV"), skiprows=skiprows, index_col=0)
        if "daily" in filename:
            returns_data = {
                "Daily": df.iloc[:-1].pipe(
                    lambda x: x.set_index(pd.to_datetime(x.index))
                )
            }
        else:
            returns_data = clean_kf_dataframes(df, multi_df=multi_df)

        logger.info(f"saving in cache directory {output_dir}")
        save_dict(returns_data, output_dir)
    return returns_data


def load_buffets_data(cache_dir="data", force_reload=False):
    """
    data from Stephen Lihn: site: https://github.com/slihn
    """
    filename = clean_directory_path(cache_dir) / "ffdata_brk13f.parquet"

    if (filename.is_file()) & (~force_reload):
        logger.info(f"logging from cache directory: {filename}")
        df = pd.read_parquet(filename)

    else:
        logger.info("loading from external source")
        path = "https://github.com/slihn/buffetts_alpha_R/archive/master.zip"
        r = requests.get(path)
        files = ZipFile(BytesIO(r.content))

        df = pd.read_csv(
            files.open("buffetts_alpha_R-master/ffdata_brk13f.csv"), index_col=0
        )
        df.index = pd.to_datetime(df.index, format="%m/%d/%Y")
        logger.info(f"saving in cache directory {filename}")
        df.to_parquet(filename)
    return df
```

```
def load_sklearn_stock_returns(cache_dir="data", force_reload=False):
    """
    data from scikit-learn
    """
    filename = clean_directory_path(cache_dir) / "sklearn_returns.parquet"
    if (filename.is_file()) & (~force_reload):
        logger.info(f"logging from cache directory: {filename}")
        df = pd.read_parquet(filename)

    else:
        logger.info("loading from external source")
        url = "https://raw.githubusercontent.com/scikit-learn/examples-data/master/
↪financial-data"
        df = (
            pd.concat(
                {
                    c: pd.read_csv(f"{url}/{c}.csv", index_col=0, parse_dates=True)[
                        "close"
                    ].diff()
                    for c in symbol_dict.keys()
                },
                axis=1,
            )
            .asfreq("B")
            .iloc[1:]
        )

        logger.info(f"saving in cache directory {filename}")
        df.to_parquet(filename)
    return df


def get_fomc_urls(from_year=1999, switch_year=None):
    if switch_year is None:
        from datetime import datetime

        switch_year = datetime.now().year - 5
    calendar_url = "https://www.federalreserve.gov/monetarypolicy/fomccalendars.htm"
    r = requests.get(calendar_url)
    soup = BeautifulSoup(r.text, "html.parser")
    contents = soup.find_all(
        "a", href=re.compile("^/newsevents/pressreleases/monetary\d{8}[ax].htm")
    )
    urls_ = [content.attrs["href"] for content in contents]

    for year in range(from_year, switch_year):
        yearly_contents = []
        fomc_yearly_url = (
            f"https://www.federalreserve.gov/monetarypolicy/fomchistorical{year}.htm"
        )
        r_year = requests.get(fomc_yearly_url)
        soup_yearly = BeautifulSoup(r_year.text, "html.parser")
        yearly_contents = soup_yearly.findAll("a", text="Statement")
        for yearly_content in yearly_contents:
            urls_.append(yearly_content.attrs["href"])

    urls = ["https://www.federalreserve.gov" + url for url in urls_]
    return urls


def sent_cleaner(s):
```

```
    return s.replace("\n", " ").replace("\r", " ").replace("\t", " ").strip()


def bs_cleaner(bs, html_tag_blocked=None):
    if html_tag_blocked is None:
        html_tag_blocked = [
            "style",
            "script",
            "[document]",
            "meta",
            "a",
            "span",
            "label",
            "strong",
            "button",
            "li",
            "h6",
            "font",
            "h1",
            "h2",
            "h3",
            "h5",
            "h4",
            "em",
            "body",
            "head",
            "sup",
        ]
    return [
        sent_cleaner(t)
        for t in bs.find_all(text=True)
        if (t.parent.name not in html_tag_blocked) & (len(sent_cleaner(t)) > 0)
    ]


regexp = re.compile(r"\s+", re.UNICODE)


def feature_extraction(corpus, sent_filters=None):
    if sent_filters is None:
        sent_filters = [
            "Board of Governors",
            "Federal Reserve System",
            "20th Street and Constitution Avenue N.W., Washington, DC 20551",
            "Federal Reserve Board - Federal Reserve issues FOMC statement",
            "For immediate release",
            "Federal Reserve Board - FOMC statement",
            "DO NOT REMOVE:  Wireless Generation",
            "For media inquiries",
            "or call 202-452-2955.",
            "Voting",
            "For release at",
            "For immediate release",
            "Last Update",
            "Last update",
        ]

    text = [
        " ".join(
            [
                regexp.sub(" ", s)
```

```
                for i, s in enumerate(c)
                if (i > 1) & np.all([q not in s for q in sent_filters])
            ]
        )
        for c in corpus
    ]

    release_date = [pd.to_datetime(c[1].replace("Release Date: ", "")) for c in corpus]
    last_update = [
        pd.to_datetime(
            [
                s.replace("Last update:", "").replace("Last Update:", "").strip()
                for s in c
                if "last update: " in s.lower()
            ][0]
        )
        for c in corpus
    ]
    voting = [" ".join([s for s in c if "Voting" in s]) for c in corpus]
    release_time = [
        " ".join(
            [s for s in c if ("For release at" in s) | ("For immediate release" in s)]
        )
        for c in corpus
    ]

    return pd.DataFrame(
        {
            "release_date": release_date,
            "last_update": last_update,
            "text": text,
            "voting": voting,
            "release_time": release_time,
        }
    )


def load_fomc_statements(
    add_url=True, cache_dir="data", force_reload=False, progress_bar=False, from_year=1999
):
    """
    https://www.federalreserve.gov/monetarypolicy/fomccalendars.htm
    """
    filename = clean_directory_path(cache_dir) / "fomc_statements.parquet"
    if (filename.exists()) & (~force_reload):
        logger.info(f"logging from cache file: {filename}")
        statements = pd.read_parquet(filename)
    else:
        logger.info("loading from external source")
        urls = get_fomc_urls(from_year=from_year)
        if progress_bar:
            urls_ = tqdm(urls)
        else:
            urls_ = urls
        corpus = [
            bs_cleaner(BeautifulSoup(requests.get(url).text, "html.parser"))
            for url in urls_
        ]
        statements = feature_extraction(corpus).set_index("release_date")
        if add_url:
            statements = statements.assign(url=urls)
```

```
        statements = statements.sort_index()
        logger.info(f"saving cache file {filename}")
        statements.to_parquet(filename)
    return statements


def load_loughran_mcdonald_dictionary(cache_dir="data", force_reload=False):
    """
    Software Repository for Accounting and Finance by Bill McDonald
    https://sraf.nd.edu/loughranmcdonald-master-dictionary/
    """
    filename = (
        clean_directory_path(cache_dir)
        / "Loughran-McDonald_MasterDictionary_1993-2021.csv"
    )
    if (filename.exists()) & (~force_reload):
        logger.info(f"logging from cache file: {filename}")
    else:
        logger.info("loading from external source")
        id = "17CmUZM9hGUdGYjCXcjQLyybjTrcjrhik"
        url = f"https://docs.google.com/uc?export=download&confirm=t&id={id}"
        subprocess.run(f"wget -O '{filename}' '{url}'", shell=True, capture_output=True)
    return pd.read_csv(filename)


def load_10X_summaries(cache_dir="data", force_reload=False):
    """
    Software Repository for Accounting and Finance by Bill McDonald
    https://sraf.nd.edu/sec-edgar-data/
    """
    filename = (
        clean_directory_path(cache_dir)
        / "Loughran-McDonald_10X_Summaries_1993-2021.csv"
    )
    if (filename.is_file()) & (~force_reload):
        logger.info(f"logging from cache directory: {filename}")
    else:
        logger.info("loading from external source")
        id = "1CUzLRwQSZ4aUTfPB9EkRtZ48gPwbCOHA"
        url = f"https://docs.google.com/uc?export=download&confirm=t&id={id}"
        subprocess.run(f"wget -O '{filename}' '{url}'", shell=True, capture_output=True)

    df = pd.read_csv(filename).assign(
        date=lambda x: pd.to_datetime(x.FILING_DATE, format="%Y%m%d")
    )
    return df


def load_ag_features(cache_dir="data", sheet_name="Monthly", force_reload=False):
    """
    load features from Amit Goyal's website:
    https://sites.google.com/view/agoyal145
    """
    filename = clean_directory_path(cache_dir) / "PredictorData2021.xlsx"
    if (filename.exists()) & (~force_reload):
        logger.info(f"logging from cache file: {filename}")
    else:
        id = "1OArfD2Wv9IvGoLkJ8JyoXS0YMQLDZfY2"
        url = f"https://docs.google.com/uc?export=download&confirm=t&id={id}"
        subprocess.run(f"wget -O '{filename}' '{url}'", shell=True, capture_output=True)
    return (
```

```
        pd.read_excel(filename, sheet_name=sheet_name)
        .assign(date=lambda x: pd.to_datetime(x.yyyymm, format="%Y%m"))
        .set_index("date")
    )
```

```
Overwriting ../skfin/datasets.py
```

2.2.2 Ken French data: industry returns

[9]:
```
from skfin.datasets import (
    load_buffets_data,
    load_kf_returns,
    load_sklearn_stock_returns,
)
```

[10]:
```
%%time
returns_data = load_kf_returns(filename="12_Industry_Portfolios", force_reload=True)
```

```
INFO:skfin.datasets:loading from external source
INFO:skfin.datasets:saving in cache directory data/12_Industry_Portfolios
CPU times: user 161 ms, sys: 14.9 ms, total: 176 ms
Wall time: 880 ms
```

Reloading from a cache directory is faster!

[11]:
```
%%time
returns_data = load_kf_returns(filename="12_Industry_Portfolios", force_reload=False)
```

```
INFO:skfin.datasets:logging from cache directory: data/12_Industry_Portfolios
CPU times: user 45.4 ms, sys: 10.8 ms, total: 56.2 ms
Wall time: 58.4 ms
```

[12]:
```
returns_data_SMB_HML = load_kf_returns(filename="F-F_Research_Data_Factors")
```

```
INFO:skfin.datasets:logging from cache directory: data/F-F_Research_Data_Factors
```

[13]:
```
returns_data_MOM = load_kf_returns(filename="F-F_Momentum_Factor")
```

```
INFO:skfin.datasets:logging from cache directory: data/F-F_Momentum_Factor
```

[14]:
```
returns_data_DAILY = load_kf_returns(filename="F-F_Research_Data_Factors_daily",␣
↪force_reload=True)["Daily"]
```

```
INFO:skfin.datasets:loading from external source
INFO:skfin.datasets:saving in cache directory
data/F-F_Research_Data_Factors_daily
```

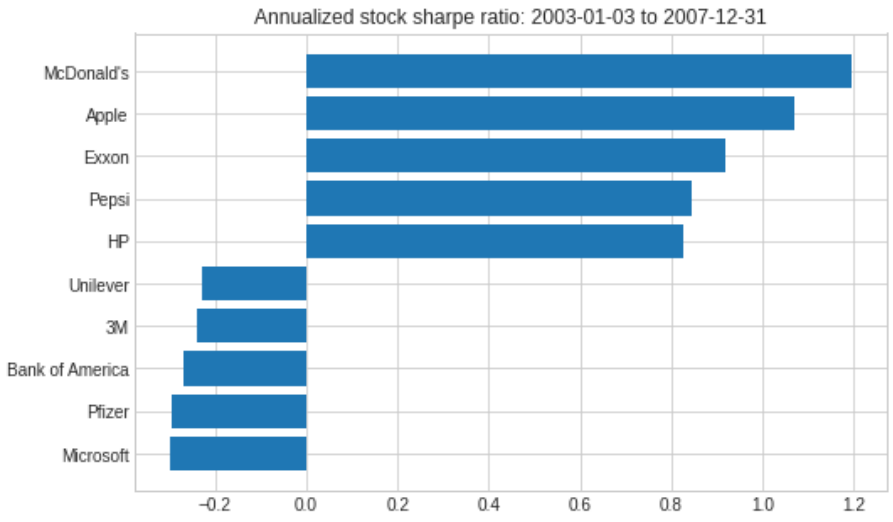
2.2.3 Stock returns (2003-2007)

[15]:
```
%%time
returns_data = load_sklearn_stock_returns(force_reload=True)
```

```
INFO:skfin.datasets:loading from external source
INFO:skfin.datasets:saving in cache directory data/sklearn_returns.parquet
CPU times: user 3.19 s, sys: 46.6 ms, total: 3.24 s
Wall time: 15.2 s
```

[16]:
```
from skfin.datasets import symbol_dict
from skfin.metrics import sharpe_ratio
from skfin.plot import bar
```

[17]:
```
start_date, end_date = returns_data.index[0].strftime("%Y-%m-%d"), returns_data.index[
    -1
].strftime("%Y-%m-%d")

df = (
    returns_data.pipe(sharpe_ratio)
    .rename(index=symbol_dict)
    .sort_values()
    .pipe(lambda x: pd.concat([x.head(), x.tail()]))
)
bar(
    df,
    horizontal=True,
    title=f"Annualized stock sharpe ratio: {start_date} to {end_date}",
)
```



2.2.4 13F Berkshire Hathaway

[18]:
```
%%time
df = load_buffets_data(force_reload=True)
```

```
INFO:skfin.datasets:loading from external source
INFO:skfin.datasets:saving in cache directory data/ffdata_brk13f.parquet
CPU times: user 122 ms, sys: 4.42 ms, total: 126 ms
Wall time: 1 s
```

2.2.5 FOMC Statements

[19]:
```
from skfin.datasets import load_fomc_statements
```

[20]:
```
%%time
statements = load_fomc_statements(force_reload=True)
```

```
INFO:skfin.datasets:loading from external source
INFO:skfin.datasets:saving cache file data/fomc_statements.parquet
CPU times: user 25.5 s, sys: 645 ms, total: 26.2 s
Wall time: 1min 37s
```

[21]:
```
%%time
statements = load_fomc_statements(force_reload=False)
```

```
INFO:skfin.datasets:logging from cache file: data/fomc_statements.parquet
CPU times: user 6.43 ms, sys: 4.71 ms, total: 11.1 ms
Wall time: 8.52 ms
```

2.2.6 Loughran-McDonalds sentiment dictionary and regulartory filing summaries

[22]:
```
from skfin.datasets import load_loughran_mcdonald_dictionary, load_10X_summaries
```

[23]:
```
%%time
filing_summaries = load_10X_summaries(force_reload=False)
```

```
INFO:skfin.datasets:logging from cache directory: data/Loughran-
McDonald_10X_Summaries_1993-2021.csv
CPU times: user 2.73 s, sys: 1.04 s, total: 3.76 s
Wall time: 3.78 s
```

[24]:
```
%%time
lm = load_loughran_mcdonald_dictionary(force_reload=False)
```

```
INFO:skfin.datasets:logging from cache file: data/Loughran-
McDonald_MasterDictionary_1993-2021.csv
CPU times: user 121 ms, sys: 25 ms, total: 146 ms
Wall time: 171 ms
```

2.2.7 Goyal long-term market data

[25]:
```
from skfin.datasets import load_ag_features
```

[26]:
```
%%time
ag = load_ag_features(force_reload=False)
```

```
INFO:skfin.datasets:logging from cache file: data/PredictorData2021.xlsx
CPU times: user 704 ms, sys: 17 ms, total: 721 ms
Wall time: 721 ms
```

Chapter 3

Mean-variance estimators

A portfolio is the ownership of a set of securities that give rights to a stream of payments. The goal of Markowitz portfolio optimisation is to determine the portfolio holdings that maximise the expected return of the portfolio under a risk constraint. Mathematically, it boils down to maximising a "mean-variance objective". In this section, we review the mathematics of Markowitz portfolio optimisation and how to implement in `python` "mean-variance optimisers" that follow the scikit-learn "fit/predict" api.

3.1 Markowitz portfolio optimisation

The portfolio universe is made of N assets:

- r is a vector of returns for each asset (with size N);
- α is the asset return forecast: $\alpha = E(r)$;
- V is the return covariance matrix that measures that correlation between each asset: $V = Var(r)$;
- h is a vector of positions/holdings in each of these assets.

Note: for a vector v, we denote v^T as the transpose of v.

A mean-variance objective is an objective that trades off the portfolio expected return (as $h^T\alpha$) against the portfolio expected risk (as h^TVh). In practice, it is written as:

$$U = h^T\alpha - \frac{h^TVh}{2\lambda},$$

where λ is the risk-tolerance.

Lemma [mean-variance]: the allocation h that maximizes the mean-variance objective is

$$h = \lambda V^{-1}\alpha.$$

The ex-ante risk is $h^TVh = \lambda^2\alpha^TV^{-1}\alpha$ and the ex-ante Sharpe ratio is

$$S = \frac{h^TE(r)}{\sqrt{h^TVh}} = \sqrt{\alpha^TV^{-1}\alpha}.$$

Corollary: The maximisation of the sharpe ratio is equivalent (up to a scaling factor) the mean-variance optimisation.

The mean-variance formula is extended to account for the linear constraints

$$Ah = b.$$

To do so, we introduce the Lagrangian $\mathcal{L}$ (and Lagrange multiplier ξ)

$$\mathcal{L} = h^T \alpha - \frac{h^T V h}{2\lambda} - (h^T A^T - b^T)\xi$$

The Lagrange multiplier ξ is a `tuning parameter` chosen exactly so that the constraint above holds. At the optimal value of ξ, the constrained problem boils down to an unconstrained problem with the adjusted return forecast $\alpha - A^T \xi$.

Lemma: the allocation that maximizes the objective $h^T \alpha - \frac{h^T V h}{2\lambda}$ under the linear constraint $Ah = b$ is

$$h = V^{-1} A^T \left(A V^{-1} A^T \right)^{-1} b + \lambda V^{-1} \left[\alpha - A^T \left(A V^{-1} A^T \right)^{-1} A V^{-1} \alpha \right]$$

Proof: the first-order condition is

$$\frac{\partial \mathcal{L}}{\partial h} = \alpha - \frac{Vh}{\lambda} - A^T \xi = 0 \Leftrightarrow h = \lambda V^{-1} [\alpha - A^T \xi]$$

The parameter ξ is chosen so that $Ah = b$

$$b = Ah = \lambda A V^{-1} [\alpha - A^T \xi] \Rightarrow \xi = [A V^{-1} A^T]^{-1} \left[A V^{-1} \alpha - \frac{b}{\lambda} \right]$$

The holding vector under constraint is

$$h_\lambda = \underbrace{V^{-1} A^T \left(A V^{-1} A^T \right)^{-1} b}_{\text{minimum variance portfolio}} + \underbrace{\lambda V^{-1} \left[\alpha - A^T \left(A V^{-1} A^T \right)^{-1} A V^{-1} \alpha \right]}_{\text{speculative portfolio}}$$

- The first term is what minimises the risk $h^T V h$ under the constraint $Ah = b$ (in particular, it does not depend on expected returns or risk-tolerance).
- The second term is the speculative portfolio (it is sensitive to both inputs).

The efficient frontier is the relation between expected portfolio return $h^T \alpha$ and portfolio standard deviation $\sqrt{h^T V h}$ for varying level of risk-tolerance

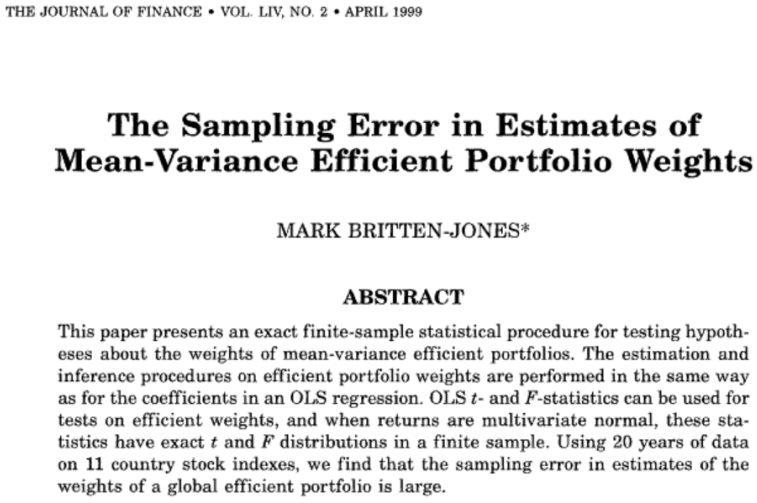
$$(x, y) \mapsto \left(h_\lambda^T \alpha, \sqrt{h_\lambda^T V h_\lambda} \right)$$

When $b = 0$, the efficient frontier between $h_\lambda^T \alpha$ and $\sqrt{h_\lambda^T V h_\lambda}$ is a line through $(0, 0)$; otherwise, it is a parabolic curve.

We focus on pure "alpha views" – that is, long-short "cash-neutral" portfolios where the sum of holdings is zero. In this case $b = 0$ and $A = \mathbf{1}$ where

$$\mathbf{1} = \begin{bmatrix} 1 & \dots & 1 \end{bmatrix}.$$

3.2 A shortcut to compute unconstrained mean-variance weights



THE JOURNAL OF FINANCE • VOL. LIV, NO. 2 • APRIL 1999

The Sampling Error in Estimates of Mean-Variance Efficient Portfolio Weights

MARK BRITTEN-JONES*

ABSTRACT

This paper presents an exact finite-sample statistical procedure for testing hypotheses about the weights of mean-variance efficient portfolios. The estimation and inference procedures on efficient portfolio weights are performed in the same way as for the coefficients in an OLS regression. OLS t- and F-statistics can be used for tests on efficient weights, and when returns are multivariate normal, these statistics have exact t and F distributions in a finite sample. Using 20 years of data on 11 country stock indexes, we find that the sampling error in estimates of the weights of a global efficient portfolio is large.

Trick to compute unconstrained mean-variance weights just with the pnl of different assets

- X: pnl of K assets over T days – so that the shape of X is $[T \times K]$.
- y: vector of ones of size T.

Lemma [Mark Britten-Jones]: the markowitz weights of are proportional to the slope coefficient of a regression of the vector of ones y on the pnls X *with no intercept*.

Proof: the coefficient of the regression with no intercept is given by

$$b = (X^T X)^{-1} X^T y.$$

The mean of the pnls is given by $\mu = \frac{1}{T} X^T y$. The variance of the pnls is $V = \frac{1}{T} X^T X - \mu\mu^T$

Using the Woodbury identity (https://en.wikipedia.org/wiki/Woodbury_matrix_identity), we have:

$$b = (V + \mu\mu^T)^{-1}\mu = \left[V^{-1} - \frac{V^{-1}\mu\mu^T V^{-1}}{1 + \mu^T V^{-1}\mu}\right]\mu = \frac{V^{-1}\mu}{1 + \mu^T V^{-1}\mu}.$$

This implies that the OLS slope coefficient b is proportial to the mean-variance holdings $V^{-1}\mu$.

3.3 Mean-variance estimators

In the following python file, we introduce several functions:

- a function that computes mean-variance holdings for batches
- a `MeanVariance` class that follows the `sklearn` api
- a `Mbj` class that computes unconstrained mean-variance weights with the Britten-Jones (1999) trick.

[3]:
```
%%writefile ../skfin/mv_estimators.py
import numpy as np
import pandas as pd
from skfin.metrics import sharpe_ratio
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.linear_model import LinearRegression


def compute_batch_holdings(pred, V, A=None, past_h=None, constant_risk=False):
    """
    compute markowitz holdings with return prediction "mu" and covariance matrix "V"

    mu: numpy array (shape N * K)
    V: numpy array (N * N)

    """

    N, _ = V.shape
    if isinstance(pred, pd.Series) | isinstance(pred, pd.DataFrame):
        pred = pred.values
    if pred.shape == (N,):
        pred = pred[:, None]
    elif pred.shape[1] == N:
        pred = pred.T

    invV = np.linalg.inv(V)
    if A is None:
        M = invV
    else:
        U = invV.dot(A)
        if A.ndim == 1:
            M = invV - np.outer(U, U.T) / U.dot(A)
        else:
            M = invV - U.dot(np.linalg.inv(U.T.dot(A)).dot(U.T))
    h = M.dot(pred)
    if constant_risk:
        h = h / np.sqrt(np.diag(h.T.dot(V.dot(h))))
    return h.T


class MeanVariance(BaseEstimator):
    def __init__(self, transform_V=None, A=1, constant_risk=True):
        if transform_V is None:
            self.transform_V = lambda x: np.cov(x.T)
        else:
            self.transform_V = transform_V
        self.A = A
        self.constant_risk = constant_risk

    def fit(self, X, y=None):
        self.V_ = self.transform_V(y)
```

```
    def predict(self, X):
        if self.A==1:
            T, N = X.shape
            A = np.ones(N)
        else:
            A = self.A
        h = compute_batch_holdings(X, self.V_, A, constant_risk=self.constant_risk)
        return h

    def score(self, X, y):
        return sharpe_ratio(np.sum(X * y, axis=1))


class Mbj(TransformerMixin):
    """
    Computing unconstrained mean-variance weights with the Britten-Jones (1999) trick.
    """

    def __init__(self, positive=False):
        self.positive = positive

    def fit(self, X, y=None):
        m = LinearRegression(fit_intercept=False, positive=self.positive)
        m.fit(X, y=np.ones(len(X)))
        self.coef_ = m.coef_ / np.sqrt(np.sum(m.coef_**2))
        return self

    def transform(self, X):
        return X.dot(self.coef_)


class TimingMeanVariance(BaseEstimator):
    def __init__(self, transform_V=None, a_min=None, a_max=None):
        if transform_V is None:
            self.transform_V = lambda x: np.var(x)
        else:
            self.transform_V = transform_V
        self.a_min = a_min
        self.a_max = a_max

    def fit(self, X, y=None):
        self.V_ = self.transform_V(y)

    def predict(self, X):
        if (self.a_min is None) & (self.a_max is None):
            h = X / self.V_
        else:
            h = np.clip(
                X / np.sqrt(self.V_), a_min=self.a_min, a_max=self.a_max
            ) / np.sqrt(self.V_)
        return h
```

```
Overwriting ../skfin/mv_estimators.py
```

[4]:
```
from skfin.datasets import load_kf_returns
from skfin.mv_estimators import MeanVariance, compute_batch_holdings

returns_data = load_kf_returns(cache_dir="data", force_reload=True)
ret = returns_data["Monthly"]["Average_Value_Weighted_Returns"][:"1999"]
```

```
INFO:skfin.datasets:loading from external source
```

```
INFO:skfin.datasets:saving in cache directory data/12_Industry_Portfolios
```

[5]:
```
T, N = ret.shape
A = np.ones(N)
```

[6]:
```
h = compute_batch_holdings(ret.mean(), ret.cov(), A, past_h=None)
```

[7]:
```
np.allclose(h.dot(A), [0.0])
```

[7]:
```
True
```

[8]:
```
A = np.stack([np.ones(N), np.zeros(N)], axis=1)
A[0, 1] = 1
```

[9]:
```
h = compute_batch_holdings(pred=ret.mean(), V=ret.cov(), A=A, past_h=None)
```

3.4 Pnl metrics

The performance of a strategy is evaluated through the profit-and-loss (pnl). The time convention is:

- holdings h_t and returns r_t are known for period t – ie. at the end of period t.
- so to compute pnl with forward-looking information, the holdings must only depend on information up to $t-1$
- in practice, we will have

$$pnl_t = h_{t-1} \times r_t.$$

First, the measure that captures the gain of a strategy $E[pnn_t]$ for a level of risk $Var[pnl_t]$ is given by the Sharpe ratio:

$$sr = \sqrt{N_{\text{periods per year}}} \times \frac{E[pnl_t]}{\sqrt{Var[pnl_t]}},$$

where $N_{\text{periods per year}}$ is an annualization factor that accounts for the number of observation per year (e.g. 12 for a monthly strategy, 260 for a strategy with pnls only on business days).

[10]:
```
%%writefile ../skfin/metrics.py
import numpy as np


def test_monthly(df):
    return int(len(df) / len(df.asfreq("M"))) == 1


def test_bday(df):
    return int(len(df) / len(df.asfreq("B"))) == 1


def test_day(df):
    return int(len(df) / len(df.asfreq("D"))) == 1


def sharpe_ratio(df, num_period_per_year=None):
    num_period_per_year = None
```

```
    if test_monthly(df):
        num_period_per_year = 12
    if test_bday(df):
        num_period_per_year = 260
    if test_day(df):
        num_period_per_year = 365
    if num_period_per_year is None:
        return np.nan
    else:
        return df.mean() / df.std() * np.sqrt(num_period_per_year)


def drawdown(x, return_in_risk_unit=True, window=36, num_period_per_year=12):
    dd = x.cumsum().sub(x.cumsum().cummax())
    if return_in_risk_unit:
        return dd.div(x.rolling(window).std().mul(np.sqrt(num_period_per_year)))
    else:
        return dd
```

```
Overwriting ../skfin/metrics.py
```

Chapter 4

Backtesting

In this section, we construct a backtest using industry data. More precisely, we use data from Ken French's data library to construct a simple industry momentum return predictor.

The goal of a backtest is to assess the validity of a trading predictor at any point in the past. In particular, it is crucial to avoid any forward-looking bias – in which information available only after time t is mistakingly used at time t. In practice, the predictors are estimated over `rolling` (or `expanding`) windows. We implement rolling window estimation with the `sklearn` `TimeSeriesSplit` object.

For backtesting, visualisation is very important and we make use of some plotting functions introduced in the Appendix:

[1]:
```
from skfin.plot import bar, heatmap, line
```

4.1 Industry momentum backtest

The setup for predicting industry returns is the following:

- the assets are industries;
- the return forecast α is estimated using rolling-window returns (over L months, $L = 12$) preceding a given date;
- no `look-ahead bias`: at each date, only information up that date is used;
- such a strategy goes long past "winners" (industries with higher-than-average returns) and goes short "losers" (industries with lower-than-average returns) $\Rightarrow$ Momentum strategy;
- this strategy is often implemented by skipping the most recent month to avoid the `1-month reversal" effect`.

The article "Do Industries Explain Momentum" (1999) by Moskowitz and Grinblatt in the *Journal of Finance* document that indeed there is momentum in industry returns – past industry returns help predict statistically and economically future industry returns.

THE JOURNAL OF FINANCE • VOL. LIV, NO. 4 • AUGUST 1999

Do Industries Explain Momentum?

TOBIAS J. MOSKOWITZ and MARK GRINBLATT*

ABSTRACT

This paper documents a strong and prevalent momentum effect in industry components of stock returns which accounts for much of the individual stock momentum anomaly. Specifically, momentum investment strategies, which buy past winning stocks and sell past losing stocks, are significantly less profitable once we control for industry momentum. By contrast, industry momentum investment strategies, which buy stocks from past winning industries and sell stocks from past losing industries, appear highly profitable, even after controlling for size, book-to-market equity, individual stock momentum, the cross-sectional dispersion in mean returns, and potential microstructure influences.

Table III
Industry Momentum Trading Profits

		Panel A: No Gap					Panel B: Month Skipped				
L	H =	1	6	12	24	36	1	6	12	24	36
12	Wi	0.0202	0.0164	0.0143	0.0130	0.0138	0.0194	0.0164	0.0141	0.0128	0.0139
	Lo	0.0117	0.0111	0.0116	0.0132	0.0139	0.0128	0.0110	0.0118	0.0131	0.0138
	Wi – Lo	0.0085	0.0053	0.0026	−0.0002	−0.0001	0.0066	0.0054	0.0023	−0.0002	0.0001
		(3.94)	(5.14)	(3.60)	(−0.43)	(−0.10)	(3.11)	(5.27)	(3.18)	(−0.43)	(0.22)
buy	Wi − Mid	0.0052	0.0038	0.0014	−0.0008	−0.0009	0.0044	0.0035	0.0010	−0.0013	−0.0010
		(3.24)	(4.59)	(2.57)	(−1.83)	(−1.94)	(2.72)	(4.18)	(1.84)	(−2.89)	(−2.10)
sell	Mid − Lo	0.0032	0.0015	0.0012	0.0006	0.0008	0.0022	0.0019	0.0013	0.0011	0.0011
		(2.07)	(2.09)	(2.01)	(1.15)	(1.83)	(1.43)	(2.63)	(2.13)	(2.29)	(2.40)
	DGTW	0.0043	0.0030	0.0018	0.0000	0.0001	0.0023	0.0031	0.0015	−0.0000	0.0002
	[Wi – Lo]	(2.30)	(3.89)	(3.11)	(0.01)	(0.33)	(1.26)	(4.11)	(2.64)	(−0.06)	(0.56)

4.1.1 Industry data

To load the data, we use the function `load_kf_returns` introduce in the `Data` section:

[5]:
```
from skfin.datasets import load_kf_returns
```

[6]:
```
returns_data = load_kf_returns(cache_dir="data", force_reload=True)
```

```
INFO:skfin.datasets:loading from external source
INFO:skfin.datasets:saving in cache directory data/12_Industry_Portfolios
```

Since the Moskowitz-Grinblatt paper was published in August 1999, we will keep the data after 1999 as `out-of-sample` and only use the data before 1999.

[7]:
```
ret = returns_data["Monthly"]["Average_Value_Weighted_Returns"][:"1999"]
```

4.1.2 Backtesting functions

In the next set of helper file, we introduce the main `Bactester` class and the `fit_and_predict` function to run rolling window estimations.

[8]:
```
%%writefile ../skfin/backtesting.py
from dataclasses import dataclass

import numpy as np
import pandas as pd
from joblib import Parallel, delayed
from skfin.mv_estimators import MeanVariance
from sklearn.base import BaseEstimator, clone
from sklearn.model_selection import TimeSeriesSplit
from sklearn.utils.metaestimators import _safe_split


def compute_pnl(h, ret, pred_lag):
    pnl = h.shift(pred_lag).mul(ret)
    if isinstance(h, pd.DataFrame):
        pnl = pnl.sum(axis=1)
    return pnl


def fit_predict(estimator, X, y, train, test, return_estimator=True):
    X_train, y_train = _safe_split(estimator, X, y, train)
    X_test, _ = _safe_split(estimator, X, y, test, train)
    estimator.fit(X_train, y_train)
    if return_estimator:
        return estimator.predict(X_test), estimator
    else:
        return estimator.predict(X_test)


@dataclass
class Backtester:
    estimator: BaseEstimator = MeanVariance()
    max_train_size: int = 36
    test_size: int = 1
    pred_lag: int = 1
    start_date: str = "1945-01-01"
    end_date: str = None
    name: str = None

    def compute_holdings(self, X, y, pre_dispatch="2*n_jobs", n_jobs=1):
        cv = TimeSeriesSplit(
            max_train_size=self.max_train_size,
            test_size=self.test_size,
            n_splits=1 + len(X.loc[self.start_date : self.end_date]) // self.test_size,
        )
        parallel = Parallel(n_jobs=n_jobs, pre_dispatch=pre_dispatch)
        res = parallel(
            delayed(fit_predict)(
                clone(self.estimator), X.values, y.values, train, test, True
            )
```

```
            for train, test in cv.split(X)
        )
        y_pred, estimators = zip(*res)
        idx = X.index[np.concatenate([test for _, test in cv.split(X)])]
        if isinstance(y, pd.DataFrame):
            cols = y.columns
            h = pd.DataFrame(np.concatenate(y_pred), index=idx, columns=cols)
        elif isinstance(y, pd.Series):
            h = pd.Series(np.concatenate(y_pred), index=idx)
        else:
            h = None
        self.h_ = h
        self.estimators_ = estimators
        self.cv_ = cv
        return self

    def compute_pnl(self, ret):
        pnl = compute_pnl(self.h_, ret, self.pred_lag)
        self.pnl_ = pnl.loc[self.start_date : self.end_date]
        if self.name:
            self.pnl_ = self.pnl_.rename(self.name)
        return self

    def train(self, X, y, ret):
        self.compute_holdings(X, y)
        self.compute_pnl(ret)
        return self.pnl_
```

```
Overwriting ../skfin/backtesting.py
```

[9]:
```
from skfin.backtesting import Backtester
from skfin.mv_estimators import MeanVariance
```

4.1.3 Scikit-Learn TimeSeriesSplit

[10]:
```
from sklearn.model_selection import TimeSeriesSplit
```

Given that the data is monthly, we re-estimate the model every month. This is done by choosing the parameter `n_splits` in the class `TimeSeriesSplit` as the number of months.

[11]:
```
start_date = "1945-01-01"
test_size = 1
params = dict(max_train_size=36, test_size=test_size, gap=0)
params["n_splits"] = 1 + len(ret.loc[start_date:]) // test_size

cv = TimeSeriesSplit(**params)
```

More precisely, with `TimeSeriesSplit`:

- the `test` indices are the dates for which the holdings are computed.
- the `train` indices are the date range over which a forecasting model is trained.
- the target will been shifted by -1 and `gap` is set to 0.
- we can estimate batches with `test_size` > 1.
- `n_splits` is determined so that the backtest starts (just) before a certain start date.

[12]:
```
for train, test in cv.split(ret):
    break
```

```
ret.iloc[train].index[-1].strftime("%Y%m%d"), ret.iloc[test].index[0].strftime("%Y%m%d")
```

[12]:
```
('19441101', '19441201')
```

4.2 Empirical results

4.2.1 Cumulative pnl

We first define two transform functions:

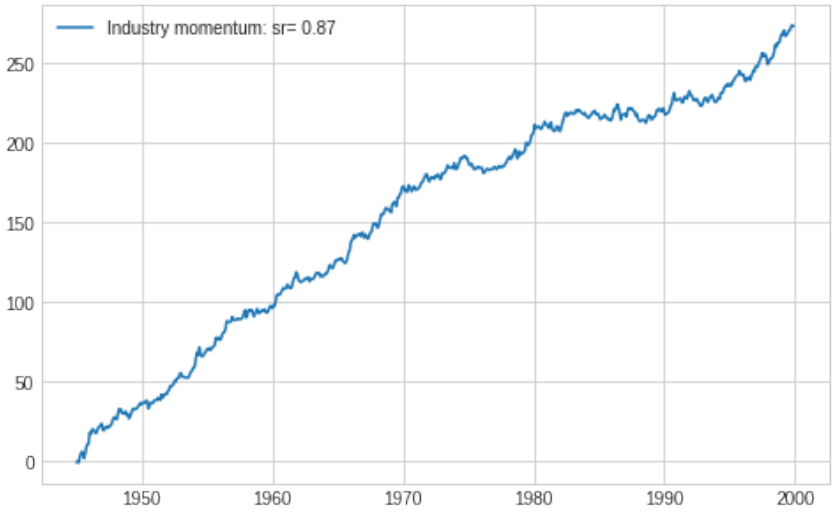
- `transform_X` computes the main feature (or `predictor`) as the average of the trailing 12-month returns;
- `transform_y` is applied on the returns – here when passed to the mean-variance `MeanVariance` class to compute the covariance matrix; but more generally to serve as a target for predictors with machine-learning.

[13]:
```
def transform_X(df, window=12):
    return df.rolling(window=window).mean()


def transform_y(df):
    return df.shift(-1)


X = transform_X(ret)
y = transform_y(ret)
```

[14]:
```
_h = []
for train, test in cv.split(ret):
    m = MeanVariance()
    m.fit(X.values[train], y.values[train])
    _h += [m.predict(X.values[test])]

cols = ret.columns
idx = ret.index[np.concatenate([test for _, test in cv.split(ret)])]
h = pd.DataFrame(np.concatenate(_h), index=idx, columns=cols)
```

Note that the `line` plotting function shows the sharpe ratio "sr" of the strategy.

[15]:
```
pnl = h.shift(1).mul(ret).sum(axis=1)[start_date:].rename("Industry momentum")
line(pnl, cumsum=True, loc="best")
```



We can also use the `Backtester` class and we test that the two approaches yield the same result.

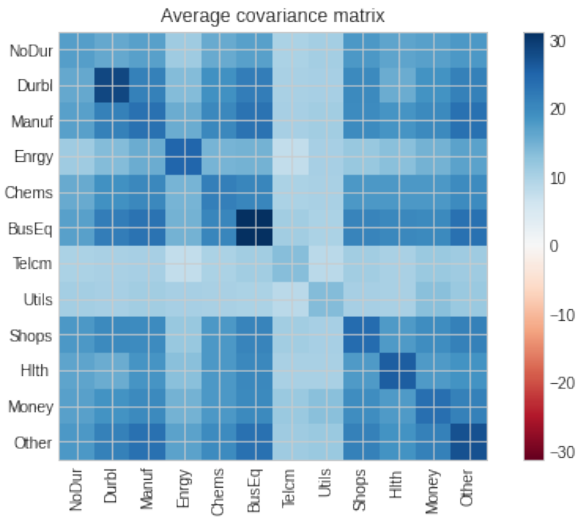
[16]:
```
m = Backtester(estimator=MeanVariance(), name="Industry momentum")
pnl_ = m.train(X, y, ret)
```

[17]:
```
assert np.allclose(h, m.h_)
assert np.allclose(pnl, m.pnl_)
```

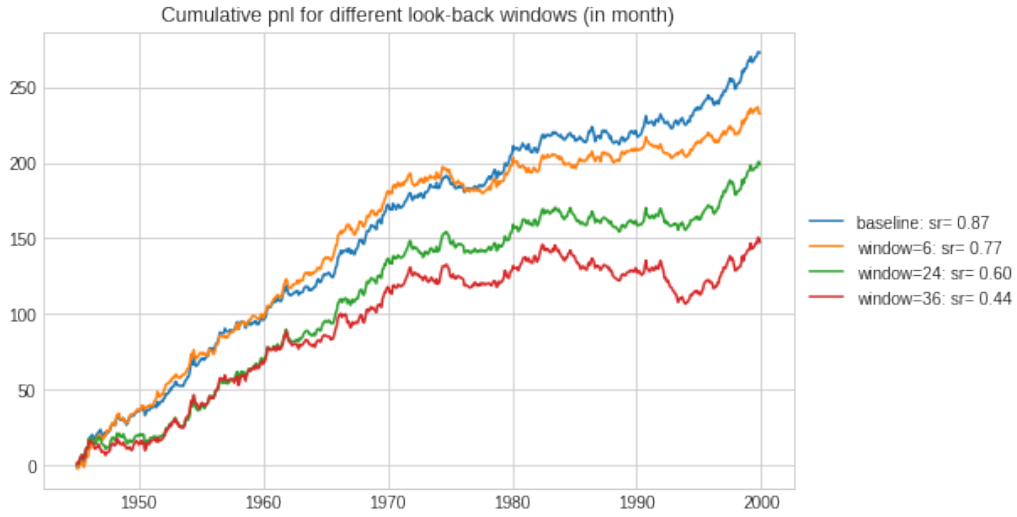
4.2.2 Other backtest statistics

We can also extract information for the estimator – e.g. in this simple case, recover the covariance matrix fitted by the class `MeanVariance()`.

[18]:
```
estimators = m.estimators_
V_mean = pd.DataFrame(
    sum([m.V_ for m in estimators]) / len(estimators),
    index=ret.columns,
    columns=ret.columns,
)
heatmap(V_mean, title="Average covariance matrix")
```



[19]:
```
pnls = []
for window in [6, 12, 24, 36]:
    X_ = ret.rolling(window).mean()
    name = f"window={window}" if window != 12 else "baseline"
    pnls += [Backtester(estimator=MeanVariance(), name=name).train(X_, y, ret)]
title = "Cumulative pnl for different look-back windows (in month)"
line(pd.concat(pnls, axis=1), cumsum=True, start_date="1945", title=title)
```

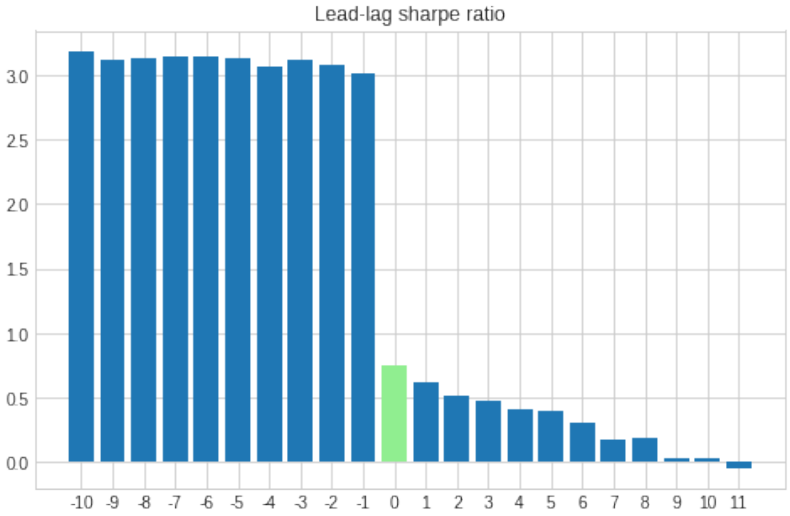


The following graph shows the `lead-lag sharpe ratio` plot for the industry momentum backtest:

- the horizon "0" (green bar) represents the sharpe ratio of the tradable pnl;
- the lagged horizons (with positive coordinates) show the performance when the positions are lagged by several months and illustrates the `alpha decay` of the predictor;
- the lead horizons (with negative coordinates) represent the non-tradable counterfactual performance if the information had been available earlier.

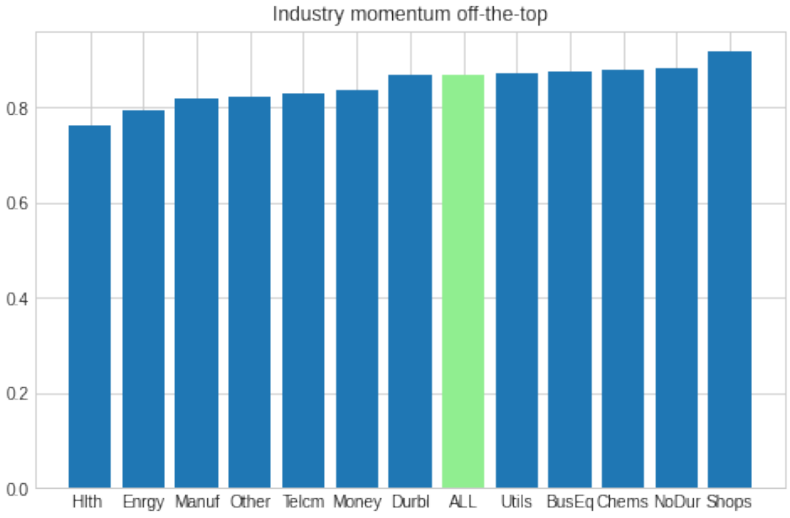
For `trending` predictors (like the industry momentum backtest), the lead sharpe ratios are very high and the alpha decay is quite fast. For `contrarian` predictors, the lead sharpe ratios are negative.

[20]:
```
from skfin.metrics import sharpe_ratio

sr = {i: h.shift(1 + i).mul(ret).sum(axis=1).pipe(sharpe_ratio) for i in range(-10, 12)}
bar(sr, baseline=0, sort=False, title="Lead-lag sharpe ratio")
```



The `off-the-top` approach is to remove an asset from the tradable set and check whether the portfolio sharpe ratio decreases (in which case, this asset is a *contributor*) or increases (in which case, this asset is a *detractor*).

[21]:
```
pnls_ott = {}
for c in ret.columns:
    ret_ = ret.drop(c, axis=1)
    X_ = transform_X(ret_)
    y_ = transform_y(ret_)
    pnls_ott[c] = (
        Backtester(estimator=MeanVariance()).train(X_, y_, ret_).pipe(sharpe_ratio)
    )
pnls_ott["ALL"] = pnl.pipe(sharpe_ratio)
bar(pnls_ott, baseline="ALL", title="Industry momentum off-the-top")
```



Chapter 5

Linear models

In this section, we take advantage of some of `scikit-learn` powerful features such as the `pipeline` to run backtests with some learning. This is an extension of the industry momentum backtests presented in the previous section.

5.1 Ridge / Lasso / Elastic net

Ridge regression: the betas $\langle \beta_1, ..., \beta_p \rangle$ are chosen to minimize

$$\frac{1}{2}\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

The Ridge regression provides more stable and accurate estimates than standard residual sum of squares minimization

Lasso regression: the betas $\langle \beta_1, ..., \beta_p \rangle$ are chosen to minimize

$$\frac{1}{2}\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

The Lasso tends to promote sparse and stable models that can be more easily interpretable.

Elastic net: the betas $\langle \beta_1, ..., \beta_p \rangle$ are chosen to minimize

$$\frac{1}{2}\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} [(1-\alpha)\beta_j^2 + \alpha|\beta_j|]$$

> "The lasso penalty is not very selective in the choice among a set of strong but correlated predictors, and the ridge penalty is inclined to shrink the coefficients of correlated variables towards each other. The compromise in the elastic net could cause the highly correlated features to be averaged while encouraging a parsiminuous model."

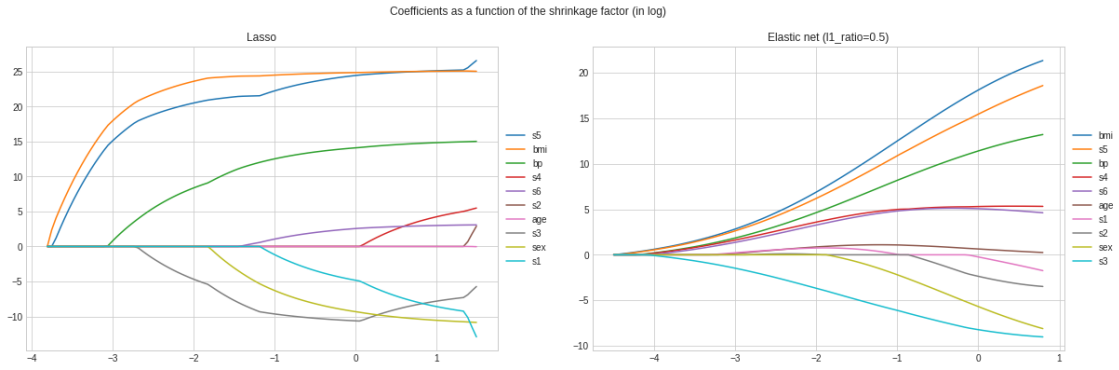
To give an example, we use a diabetes dataset provided by `sklearn`.

[2]:
```
from sklearn.datasets import load_diabetes
from sklearn.linear_model import enet_path, lasso_path
```

[3]:
```
X, y = load_diabetes(return_X_y=True, as_frame=True)
```

[4]:
```
X /= X.std(axis=0)
eps = 5e-3
l1_ratio = 0.5

alphas_lasso, coefs_lasso, _ = lasso_path(X, y, eps=eps)
alphas_enet, coefs_enet, _ = enet_path(X, y, eps=eps, l1_ratio=l1_ratio)
```

[5]:
```
fig, ax = plt.subplots(1, 2, figsize=(20, 6))
fig.suptitle("Coefficients as a function of the shrinkage factor (in log)")
line(
    pd.DataFrame(coefs_lasso.T, -1 * np.log(alphas_lasso), columns=X.columns),
    title="Lasso",
    ax=ax[0],
)
line(
    pd.DataFrame(coefs_enet.T, -1 * np.log(alphas_enet), columns=X.columns),
    title=f"Elastic net (l1_ratio={l1_ratio})",
    ax=ax[1],
)
```



See more discussion in

> Hastie, Trevor, et al. *The elements of statistical learning: data mining, inference, and prediction.* Vol. 2. New York: springer, 2009.

5.2 Revisiting the industry backtest with linear models

In the `Industry momentum` section, we introduced a feature `X` as 12-month trailing past returns and target `y` as the one-month forward return. For N industries (e.g. $N = 12$), `X` and `y` are vectors of size N.

In this section, we are interested in fitting linear models (potentially with regularisation à la Ridge, Lasso or Elastic Net) such that

$$y = BX,$$

where B is a matrix of size $[N \times N]$. (In this equation, we ignore intercept terms which are generally small.) This equation falls into two lines of research:

- in classic Econometrics, this is called a Vector Autoregressive (VAR) model (.e.g see the wikipedia article).

- in Machine-learning, this is often called a `multioutput` model.

The interpretation of this model in the context of industry timing is that it captures cross-industry effect:

- for instance, a increasing trend on oil prices (which is positive for the Energy stocks) might be a negative predictor for sectors that use oil as input (e.g. Manufacturing)

Such VAR/multioutput models will pick up the positive lead-lag correlations across industries and therefore potentially enrich the simple `Industry momentum` strategy that we introduced in the previous section.

5.2.1 Scikit-learn Pipeline and Multi-output

From the documentation, the definition of a `sklearn` pipeline is:

> Sequentially apply a list of transforms and a final estimator. Intermediate steps of the pipeline must be 'transforms', that is, they must implement fit and transform methods. The final estimator only needs to implement fit.

Estimator like linear regressions or `Ridge` regressions have the property to estimate with `multiouput` as the regression above, but are notheless estimators, not transormations. However,

- in the pipelines that we are building, the last step is the `MeanVariance` class that produces the holdings;
- in the `sklearn` pipelines, all the steps except for the last one must be transformations;
- despite formally not having a `transform` function, the `multioutput` linear estimators (such as `Ridge` and `Lasso`) are transformations of a vector `X` of size N into a vector `y` of size N.

In the following module, we extend the estimators that we will be using to have such `transform` property.

[6]:
```
%%writefile ../skfin/estimators.py
from lightgbm.sklearn import LGBMRegressor
from sklearn.base import BaseEstimator, clone
from sklearn.linear_model import LinearRegression, Ridge, RidgeCV
from sklearn.multioutput import MultiOutputRegressor
from sklearn.neural_network import MLPRegressor


class LinearRegression(LinearRegression):
    def transform(self, X):
        return self.predict(X)


class Ridge(Ridge):
    def transform(self, X):
        return self.predict(X)


class RidgeCV(RidgeCV):
    def transform(self, X):
        return self.predict(X)


class MLPRegressor(MLPRegressor):
    def transform(self, X):
        return self.predict(X)
```

```
class MultiOutputRegressor(MultiOutputRegressor):
    def transform(self, X):
        return self.predict(X)


class MultiLGBMRegressor(BaseEstimator):
    """
    Multi-output extension of the lightgbm regressor as a transform class
    get_params and set_params attributes necessary for cloning the class
    """

    def __init__(self, **kwargs):
        if "n_jobs" in kwargs.keys():
            kwargs["n_jobs"] = 1
        else:
            kwargs = {"n_jobs": 1, **kwargs}
        self.m = MultiOutputRegressor(LGBMRegressor(**kwargs))

    def get_params(self, deep=True):
        return self.m.estimator.get_params(deep=deep)

    def set_params(self, **kwargs):
        if "n_jobs" in kwargs.keys():
            kwargs["n_jobs"] = 1
        else:
            kwargs = {"n_jobs": 1, **kwargs}
        return self.m.estimator.set_params(**kwargs)

    def fit(self, X, y):
        return self.m.fit(X, y)

    def transform(self, X):
        return self.m.transform(X)
```

```
Overwriting ../skfin/estimators.py
```

5.2.2 Linear Regression

We first load the data and the main custom functions to run a backtest.

[7]:
```
from skfin.backtesting import Backtester
from skfin.datasets import load_kf_returns
from skfin.mv_estimators import MeanVariance

returns_data = load_kf_returns(cache_dir="data")

ret = returns_data["Monthly"]["Average_Value_Weighted_Returns"][:"1999"]


def transform_X(df, window=12):
    return df.rolling(window=window).mean()


def transform_y(df):
    return df.shift(-1)


features = transform_X(ret)
target = transform_y(ret)
```

```
INFO:skfin.datasets:logging from cache directory: data/12_Industry_Portfolios
```

As a reference, we re-compute the pnl of a simple `Industry momentum` strategy.

[8]:
```
pnls = [
    Backtester(MeanVariance(), name="Industry momentum").train(features, target, ret)
]
```

We now load the functions that are specific to building `sklearn` pipelines.

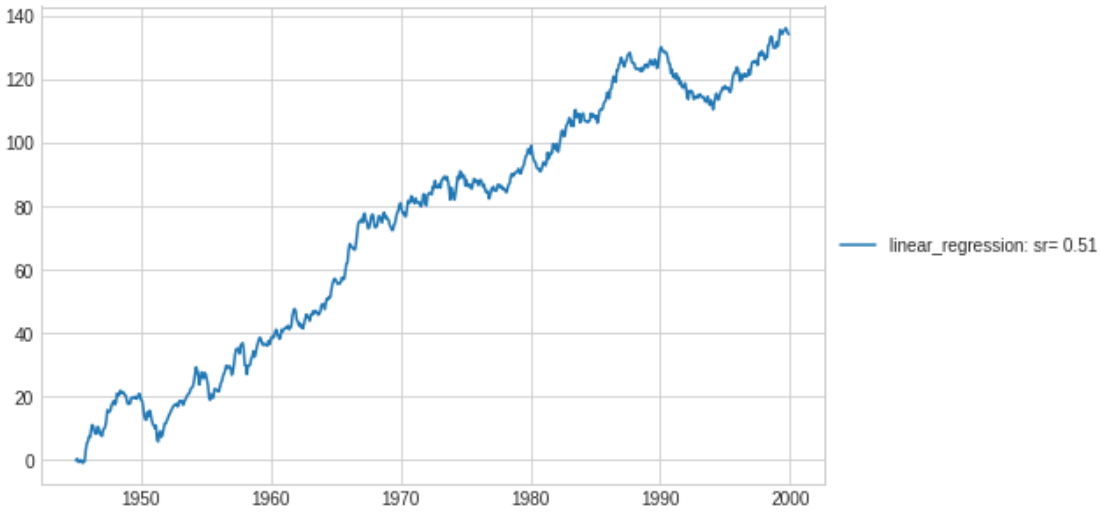
[9]:
```
from skfin.estimators import LinearRegression, Ridge
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
```

In the following simple pipeline, there are two steps:

- the first step is a simple `multioutput` linear regression that produces fitted predictions for each industry
- the second step takes these predictors and scales them as holdings with a mean-variance optimisation (with the `MeanVariance` class introduced earlier.

[10]:
```
estimator = make_pipeline(LinearRegression(), MeanVariance())
```

[11]:
```
m = (
    Backtester(estimator, name="linear_regression")
    .compute_holdings(features, target)
    .compute_pnl(ret)
)
pnls += [m.pnl_]
line(m.pnl_, cumsum=True)
```



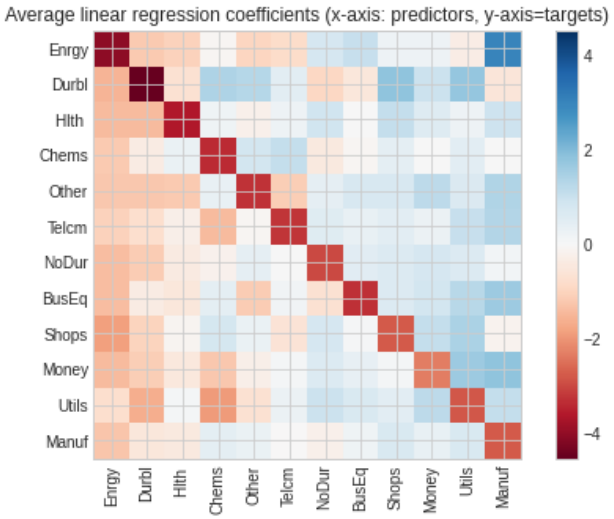
The linear regression fits an intercept and some coefficients.

[12]:
```
ols_ = m.estimators_[0].named_steps["linearregression"]
coef_ = ols_.coef_
intercept_ = ols_.intercept_
vec = ret.mean().values
np.allclose(ols_.predict(vec[None, :]), coef_.dot(vec) + intercept_)
```

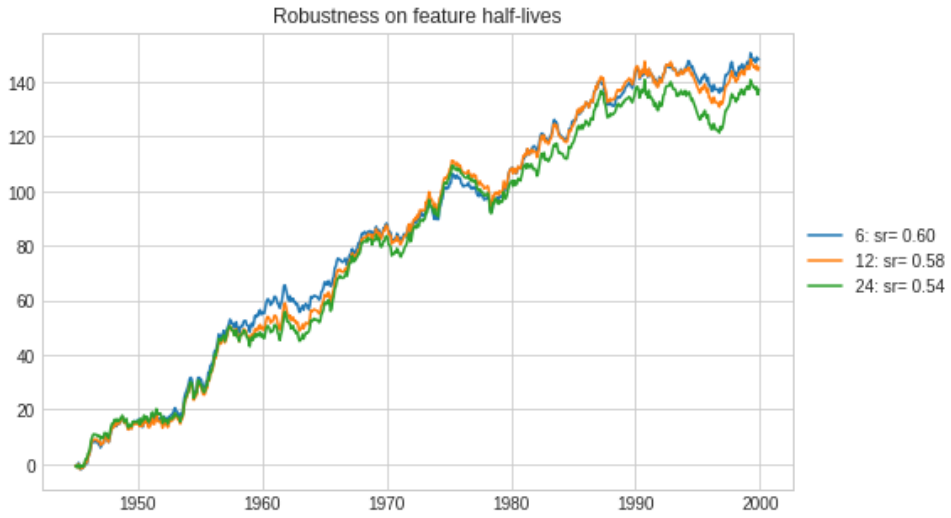
[12]:
```
True
```

[13]:
```
coefs_ = [m.named_steps["linearregression"].coef_ for m in m.estimators_]
coefs_mean = pd.DataFrame(sum(coefs_) / len(coefs_), ret.columns, ret.columns).T
```

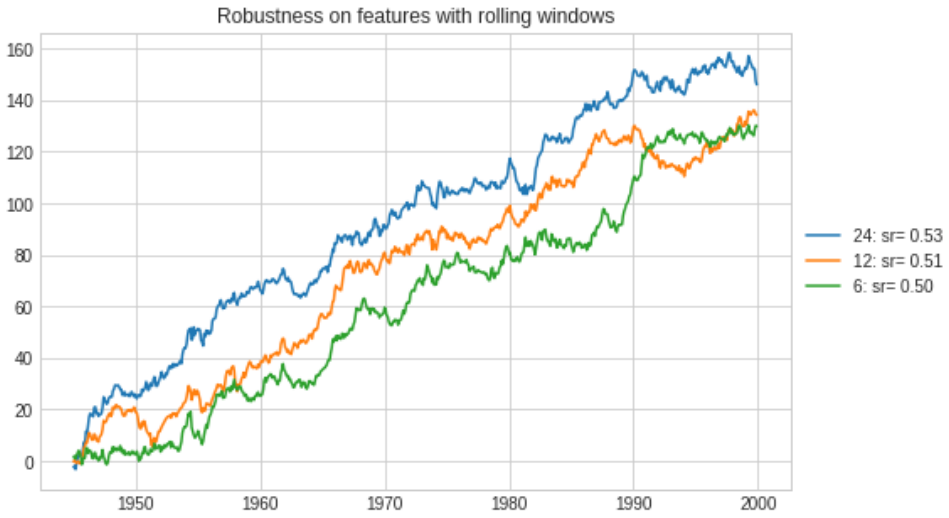
[14]:
```
heatmap(
    coefs_mean.loc[
        coefs_mean.mean(1).sort_values().index, coefs_mean.mean(1).sort_values().index
    ],
    title="Average linear regression coefficients (x-axis: predictors, y-axis=targets)",
)
```



[15]:
```
pnls_ = {}
for hl in [6, 12, 24]:
    features_ = ret.ewm(halflife=hl).mean().fillna(0)
    pnls_[hl] = Backtester(estimator).train(features_, target, ret)
line(pnls_, cumsum=True, title="Robustness on feature half-lives")
```



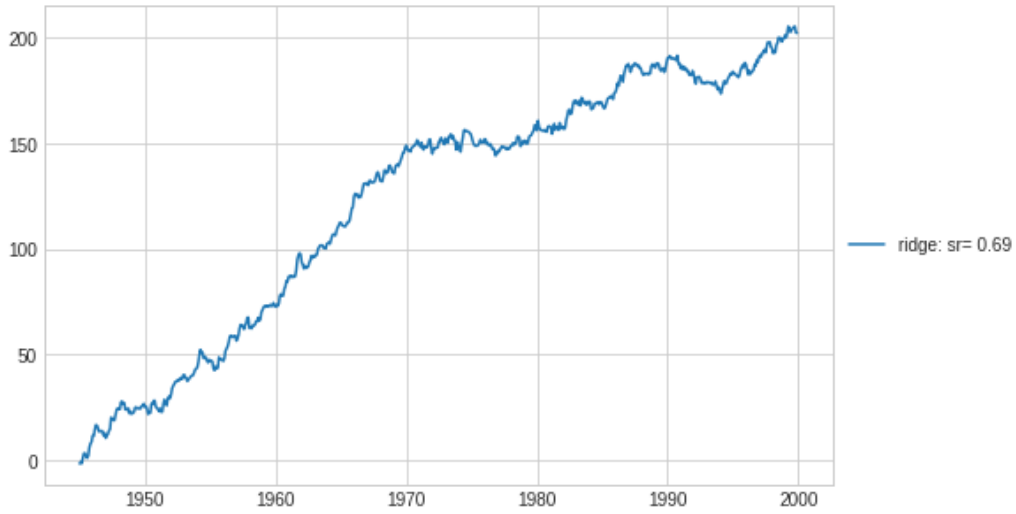
[16]:
```
pnls_ = {}
for hl in [6, 12, 24]:
    features_ = ret.rolling(window=hl).mean().fillna(0)
    pnls_[hl] = Backtester(estimator).train(features_, target, ret)
line(pnls_, cumsum=True, title="Robustness on features with rolling windows")
```



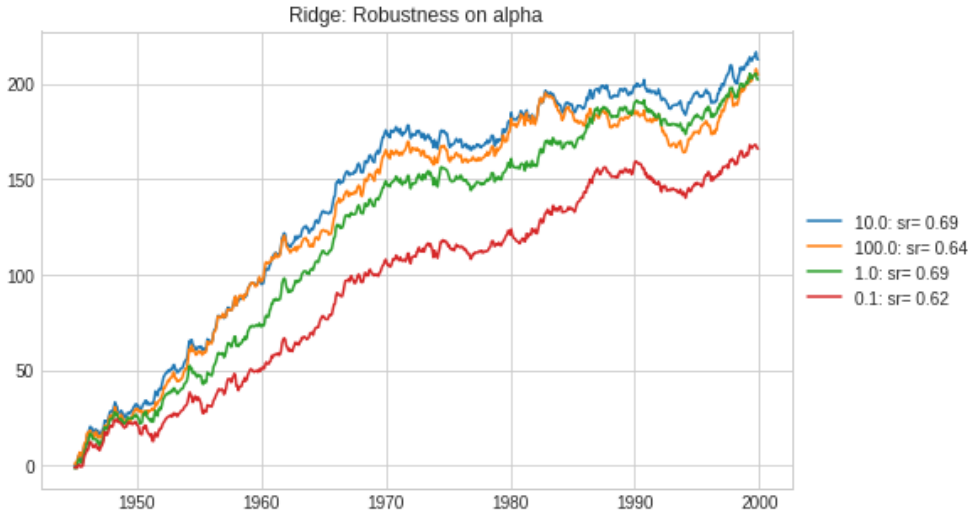
5.2.3 Ridge

Given that the `Ridge` regression applies a constraint across features, each feature needs to be properly rescaled, which is done here with `StandardScalar`.

[17]:
```
estimator = make_pipeline(StandardScaler(with_mean=False), Ridge(), MeanVariance())
m = (
    Backtester(estimator, name="ridge")
    .compute_holdings(features, target)
    .compute_pnl(ret)
)
pnls += [m.pnl_]
line(m.pnl_, cumsum=True)
```



[18]:
```
pnls_ = {}
for alpha in [0.1, 1, 10, 100]:
    estimator_ = make_pipeline(
        StandardScaler(with_mean=False), Ridge(alpha=alpha), MeanVariance()
    )
    pnls_[alpha] = Backtester(estimator_).train(features, target, ret)
line(pnls_, cumsum=True, title="Ridge: Robustness on alpha")
```



5.2.4 Ridge with feature expansion

We can expand the set of features by using polynomial transfomrs with `PolynomialFeatures`.

[19]:
```
PolynomialFeatures(degree=2).fit_transform(ret.iloc[:10]).shape
```

[19]: (10, 91)

Number of new features: intercept, initial features (=12), squared features (12), all cross features of degree 1 (=6*11):

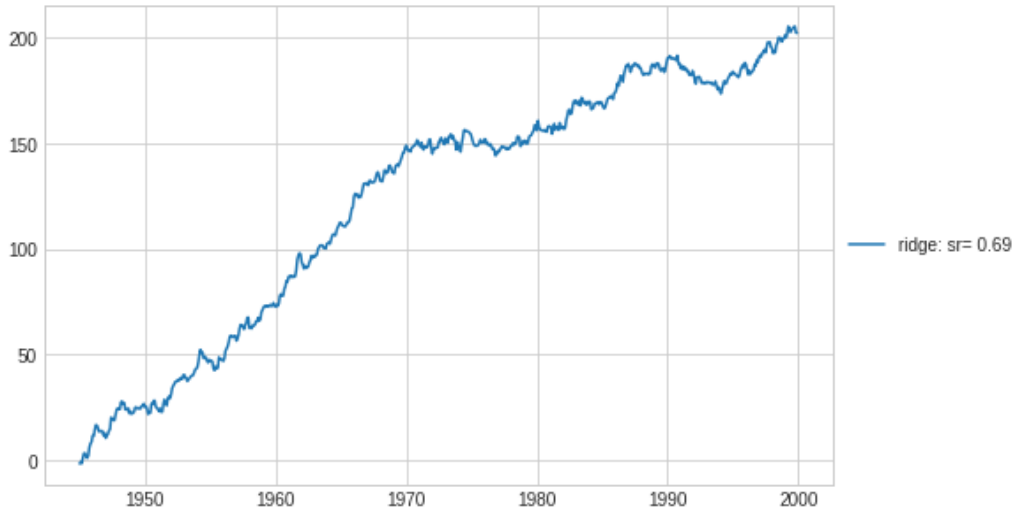
[20]:
```
estimator = make_pipeline(
    StandardScaler(with_mean=False),
    PolynomialFeatures(degree=2),
    Ridge(alpha=100),
    MeanVariance(),
)
```

[21]:
```
print(f"Number of features generated by degree=2: {1+ 12 + 12 + 6 * 11}")
```

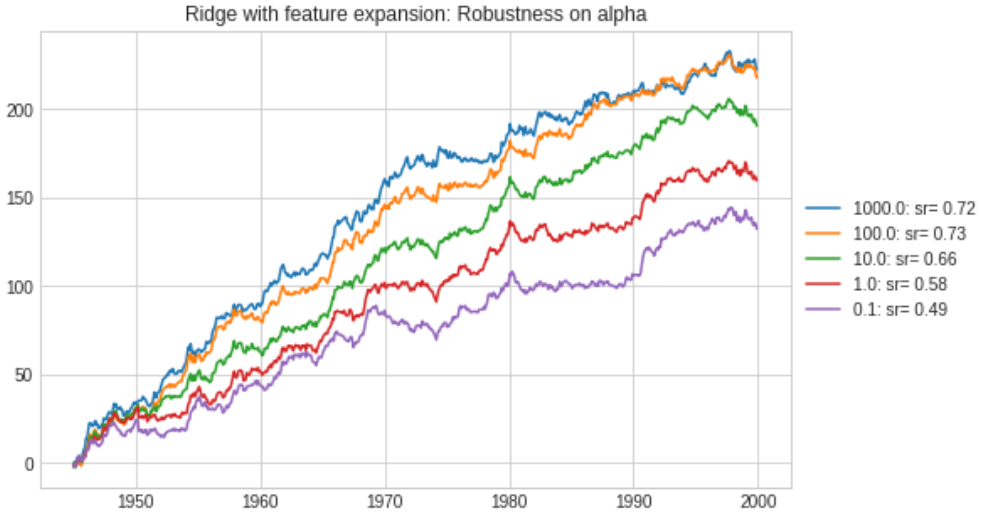
```
Number of features generated by degree=2: 91
```

[22]:
```
pnls += [
    Backtester(estimator, name="ridge_with_feature_expansion").train(
        features_, target, ret
    )
]
```

```
line(m.pnl_, cumsum=True)
```

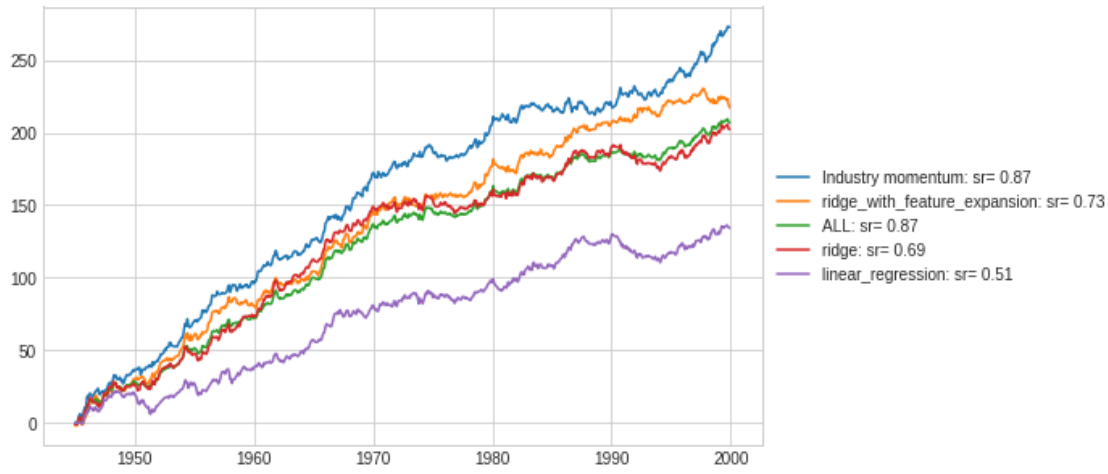


[23]:
```
pnls_ = {}
for alpha in [0.1, 1, 10, 100, 1000]:
    estimator_ = make_pipeline(
        StandardScaler(with_mean=False),
        PolynomialFeatures(degree=2),
        Ridge(alpha=alpha),
        MeanVariance(),
    )
    pnls_[alpha] = Backtester(estimator_).train(features_, target, ret)
line(pnls_, cumsum=True, title="Ridge with feature expansion: Robustness on alpha")
```



Putting all the types of linear predictors together, we can compare the cumulative pnls in the graph below.

[24]:
```
line(pd.concat(pnls, axis=1).assign(ALL=lambda x: x.mean(axis=1)), cumsum=True)
```


250
200
150
100
50
0
1950
1960
1970
1980
1990
2000
Industry momentum: sr= 0.87
ridge_with_feature_expansion: sr= 0.73
ALL: sr= 0.87
ridge: sr= 0.69
linear_regression: sr= 0.51

Chapter 6

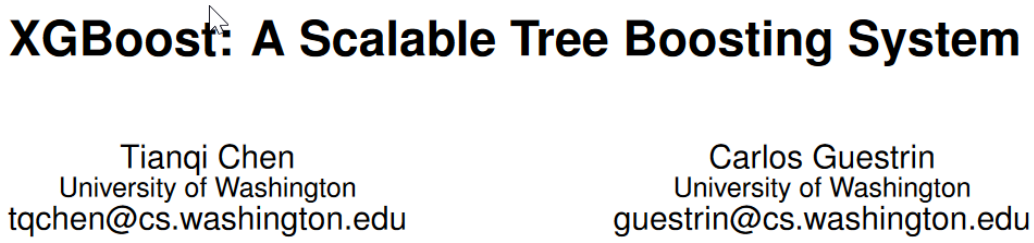
Boosted Trees and Neural nets

In this section, we review two important classes of non-linear forecasting models: boosted trees and the multi-layer perceptron.

6.1 Boosted Trees

We first discuss boosted trees, in particular as described by the companion paper to the package `xgboost`:

> Chen and Guestrin (2016): "XGBoost: A Scalable Tree Boosting System," *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*.



For a dataset $\langle x_n, y_n \rangle$ with N samples ($x_n \in \mathbb{R}^M$), a tree ensemble model uses K additive functions:

$$\hat{y}_n = \phi(x_n) = \sum_k f_k(x_n),$$

where f_k is in the space of regression trees $\mathcal{F} = \{f\}$:

- q: $\mathbb{R}^M \to J$ is a partition;
- $f(x) = w_{q(x)}$ is a constant value on each leaf of the tree.

The objective is to minimize the loss:

$$\mathcal{L}(\phi) = \sum_n l(y_n, \hat{y}_n) + \sum_k \Omega(f_k),$$

where $\Omega(f) = \gamma J + \frac{1}{2}\lambda||w||^2$ is a regularisation term and l is a convex loss function (e.g. mean squared error).

The functions f_k are derived iteratively:

$$\mathcal{L}^k = \sum_n l\left(y_n, \hat{y}_n^{k-1} + f_k(x_n)\right) + \Omega(f_k).$$

With a second-order Taylor expansion, we have

$$\mathcal{L}^k \approx \sum_n \left[l(y_n, \hat{y}_n^{k-1}) + g_n f_k(x_n) + \frac{1}{2} h_n f_k(x_n)^2\right] + \Omega(f_k),$$

where $g_n = \partial_{\hat{y}} l(y_n, \hat{y}_n^{k-1})$ and $h_n = \partial^2_{\hat{y}} l(y_n, \hat{y}_n^{k-1})$. For an instance of leaf j, with $I_j = \{n|q(x_n) = j\}$, we can sum by leaf:

$$\mathcal{L}^k = \sum_{j=1}^{j=J} \left(\sum_{n\in I_j} g_n\right) w_j + \frac{1}{2}\left(\sum_{n\in I_j} h_n + \lambda\right) w_j^2 + \gamma J + constant.$$

For a given $q(x)$, the optimal weight w_j^* of leaf j is

$$w_j^* = -\frac{\sum_{n\in I_j} g_n}{\sum_{n\in I_j} h_n + \lambda}.$$

The corresponding optimal value is then

$$\tilde{\mathcal{L}}^k(q) = -\frac{1}{2}\sum_{j=1}^{j=J} \frac{\left(\sum_{n\in I_j} g_n\right)^2}{\sum_{n\in I_j} h_n + \lambda} + \gamma J + constant.$$

A greedy algorithm that starts from a single leaf and iteratively adds branches to the tree is used to dermine q.

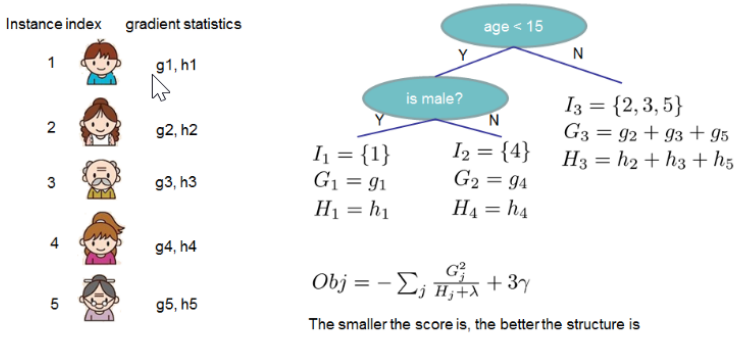


Figure 2: Structure Score Calculation. We only need to sum up the gradient and second order gradient statistics on each leaf, then apply the scoring formula to get the quality score.

In practice, `xgboost` (and `lightgbm`) can be used with custom loss functions – for instance, by providing the gradient and hessian functions.

6.2 Multi-layer perceptron

For the input $x \in \mathbb{M}$, the layer (with hidden size equals to K) of a multi-layer perceptron is given by

$$g(b + Wx)$$

where W is a $[K \times M]$ matrix, b is a scalar and g is an activation function. The output of the last layer has to match the size of the target vector y.

6.3 Predicting industry returns with non-linear models

6.3.1 Lightgbm

[4]:
```
from skfin.estimators import MLPRegressor, MultiLGBMRegressor, Ridge

def transform_X(df, window=12):
    return df.rolling(window=window).mean()


def transform_y(df):
    return df.shift(-1)


features = transform_X(ret)
target = transform_y(ret)
```

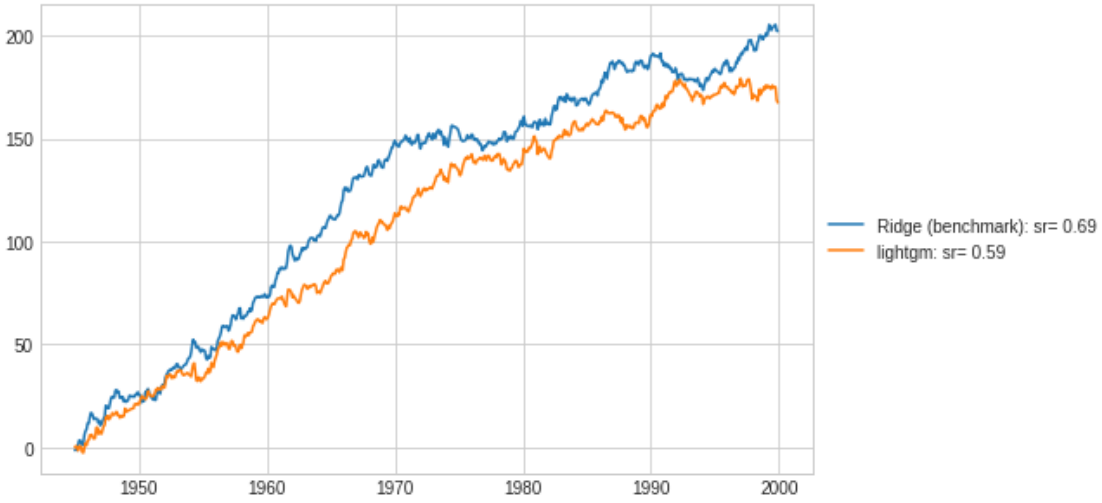
As a benchmark based on estimating the cross-industry effects, we first look at the `Ridge` estimator.

[5]:
```
estimator0 = make_pipeline(StandardScaler(with_mean=False), Ridge(), MeanVariance())
m = (
    Backtester(estimator0, name="Ridge (benchmark)")
    .compute_holdings(features, target)
    .compute_pnl(ret)
)
h0, pnl0, estimators0 = m.h_, m.pnl_, m.estimators_
pnls = [pnl0]
```

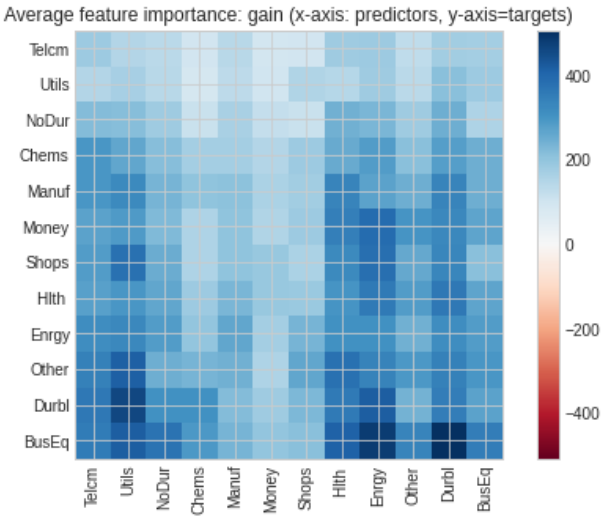
[6]:
```
estimator = make_pipeline(
    MultiLGBMRegressor(min_child_samples=5, n_estimators=25), MeanVariance()
)
```

[7]:
```
%%time
m = (
    Backtester(estimator, name="lightgm")
    .compute_holdings(features, target)
    .compute_pnl(ret)
)
pnls += [m.pnl_]
line(pnls, cumsum=True)
```

```
CPU times: user 2min 3s, sys: 3min 37s, total: 5min 41s
Wall time: 43.1 s
```



[8]:
```
func = lambda x: np.stack(
    [m.booster_.feature_importance(importance_type="gain") for m in x]
)
importance = [func(m_.steps[0][1].m.estimators_) for m_ in m.estimators_]
importance_mean = pd.DataFrame(
    sum(importance) / len(importance), ret.columns, ret.columns
).T

heatmap(
    importance_mean.loc[
        importance_mean.mean().sort_values().index,
        importance_mean.mean().sort_values().index,
    ],
    title="Average feature importance: gain (x-axis: predictors, y-axis=targets)",
)
```



6.3.2 MLPRegressor

We first focus on a single window to understand how the `MLPRegressor` works in `scikit-learn`.

[9]:
```
for train, test in m.cv_.split(ret):
    break
```

[10]:
```
scaler = StandardScaler(with_mean=True)
X_train = scaler.fit_transform(features.iloc[train])
y_train = target.iloc[train]
```

[11]:
```
X_test = scaler.transform(features.iloc[test])
y_test = target.iloc[test]
```

We instantiate a simple `MLP` with 6 neurons. The activation function is a logistic sigmoid.

[12]:
```
m = MLPRegressor(
    hidden_layer_sizes=(6,),
    solver="adam",
    learning_rate_init=0.5,
    alpha=100,
    activation="logistic",
    tol=1e-2,
    n_iter_no_change=25,
    early_stopping=False,
)
m.fit(X_train, y_train)
y_pred = m.predict(X_test)
```

When `early_stopping` is `False`, the optimisation stops based on the in-sample score, while when `early_stopping` is `True`, the decision to stop is based on a random sample (e.g. 10% of the training data).

[13]:
```
print(f"The number of iterations is {m.n_iter_} (out of a maximum of {m.max_iter}).")
```

```
The number of iterations is 78 (out of a maximum of 200).
```

[14]:
```
print(
    f"Number of parameter:\n - first layer: {12 * 6  + 6}\n - second layer: {12 * 6 + 12}\n - ␣
 ↪total number of parameters: {12  * 12 + 6 + 12}"
)
```

```
Number of parameter:
 - first layer: 78
 - second layer: 84
 - total number of parameters: 162
```

[15]:
```
m.coefs_[0].shape, m.coefs_[1].shape
```

[15]:
```
((12, 6), (6, 12))
```

[16]:
```
m.intercepts_[0].shape, m.intercepts_[1].shape
```

[16]:
```
((6,), (12,))
```

The `sigmoid logistic` activation function is also known as `expit` and it is provided by the `scipy` package.

The MLP regressor in this case:

- project the vector of size 12 on a vector of size 6

- a bias vector of size 6 is added
- the activitation function (here the `sigmoid`) regularizes the neurons
- the second layer then projects the vector of size 6 on a vector of size 12 (with a bias of size 12).

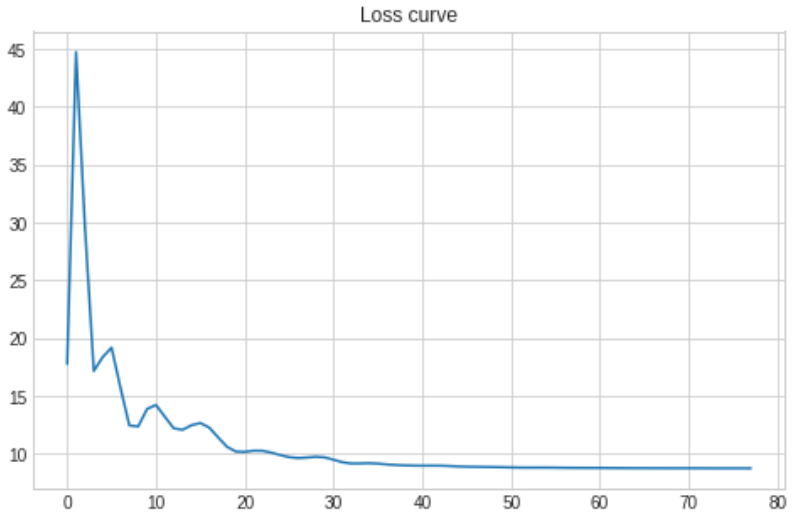
[17]:
```
from scipy.special import expit

y_pred_ = (
    expit(X_test.dot(m.coefs_[0]) + m.intercepts_[0]).dot(m.coefs_[1])
    + m.intercepts_[1]
)
```

[18]:
```
np.allclose(y_pred, y_pred_)
```

[18]:
```
True
```

The `sklearn` package provides a loss curve

[19]:
```
line(pd.Series(m.loss_curve_), legend=False, title="Loss curve")
```



The quadratic loss of is the `squared error`:

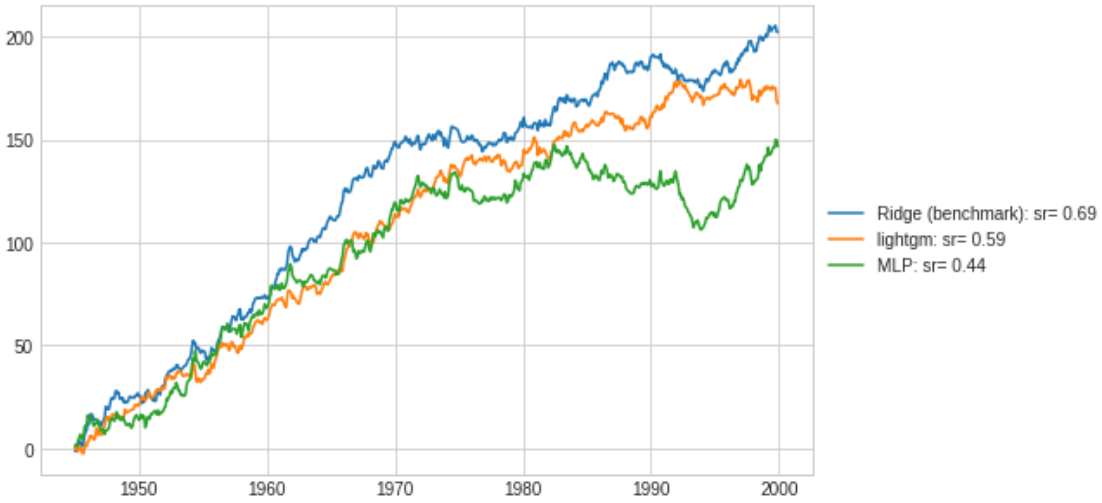
[20]:
```
np.allclose(m.loss_curve_[-1], ((y_train - y_pred) ** 2).mean().mean() / 2, atol=1e-2)
```

[20]:
```
True
```

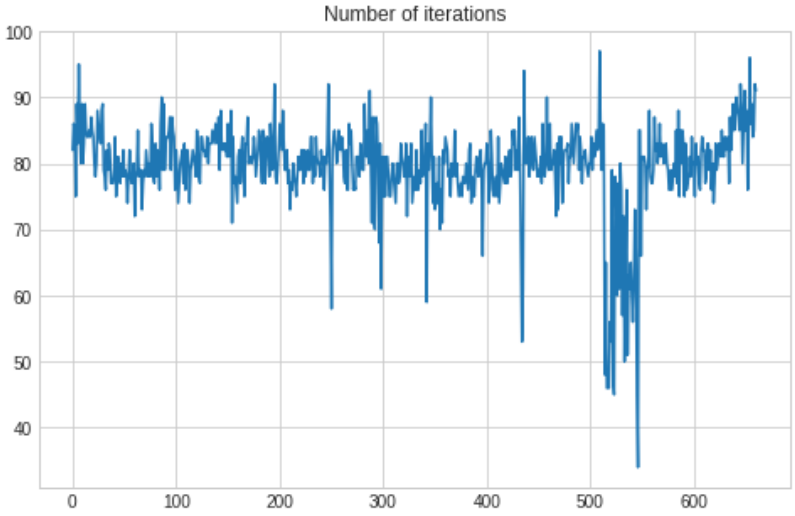
We now look at a backtest using `MLPRegressor`.

[21]:
```
estimator = make_pipeline(
    StandardScaler(with_mean=False),
    MLPRegressor(
        hidden_layer_sizes=(6,),
        learning_rate_init=0.5,
        alpha=100,
        solver="adam",
        activation="logistic",
        tol=1e-2,
        n_iter_no_change=25,
        early_stopping=False,
```

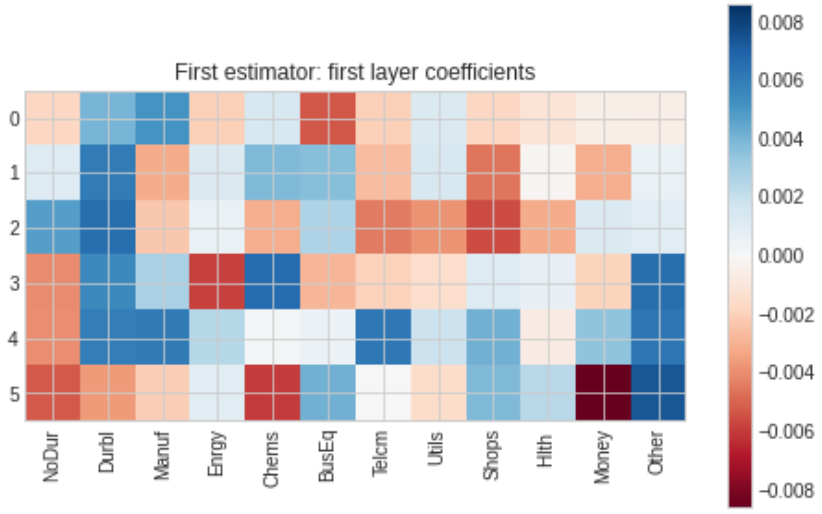
```
    ),
    MeanVariance(),
)
m = (
    Backtester(estimator, name="MLP")
    .compute_holdings(features, target)
    .compute_pnl(ret)
)
pnls += [m.pnl_]

line(pnls, cumsum=True)
```

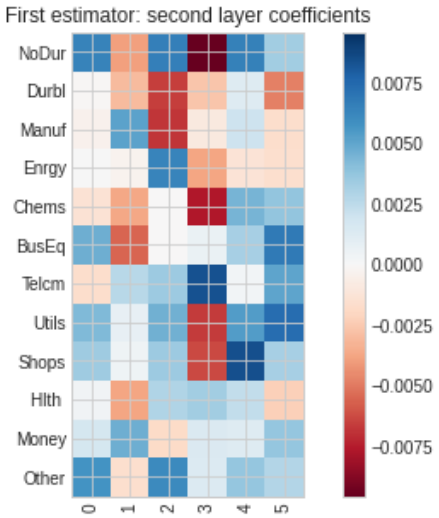


[22]:
```
line(
    pd.Series([m_[1].n_iter_ for m_ in m.estimators_]),
    title="Number of iterations",
    legend=False,
)
```



[23]:
```
for m_ in m.estimators_:
    break

heatmap(
    pd.DataFrame(m_[1].coefs_[0], index=ret.columns),
    title="First estimator: first layer coefficients",
)
heatmap(
    pd.DataFrame(m_[1].coefs_[1], columns=ret.columns),
    title="First estimator: second layer coefficients",
)
```



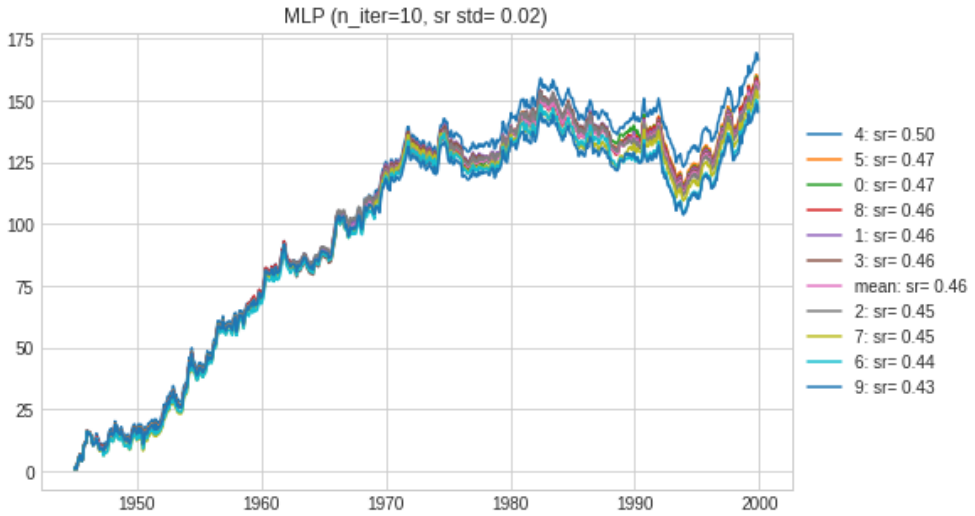


Given the stochasticity of the estimation, we are interested in evaluating the noise associated to a given run. More precisely, we re-run the backtest with exactly the same estimator (and hence the same parameters). In fact, such stochasticity depends on the amount regularisation, and to make this point, we relax it with `alpha=50` (instead of 100).

[24]:
```
%%time
estimator_ = make_pipeline(
    StandardScaler(with_mean=False),
    MLPRegressor(
        hidden_layer_sizes=(6,),
        learning_rate_init=0.5,
        alpha=50,
        n_iter_no_change=25,
        solver="adam",
        tol=1e-2,
        activation="logistic",
    ),
    MeanVariance(),
)
n_iter = 10
pnls_ = {}
for i in range(n_iter):
    pnls_[i] = Backtester(estimator_).train(features, target, ret)

sr_std = np.std([v.pipe(sharpe_ratio) for k, v in pnls_.items()])
```

```
CPU times: user 7min 44s, sys: 13min 22s, total: 21min 6s
Wall time: 2min 40s
```

[25]:
```
title = f"MLP (n_iter={n_iter}, sr std= {sr_std:.2f})"
line(
    pd.concat(pnls_, axis=1).assign(mean=lambda x: x.mean(axis=1)),
    cumsum=True,
    title=title,
)
```



Chapter 7

Hyperparameter optimisation

In this section, we cover the topic of `overfitting`. A folk theorem in asset management is that people are so afraid of overfitting that they tend to (massively) underfit. Or at least, that was the case. Today, better fitting models to extract as much information from a dataset has become a crucial skill.

More precisely, `overfitting` a particular dataset provides a baseline for how well a system can learn (e.g. see the Recipe for Training Neural Nets by Andrej Karpathy).

7.1 Ridge CV

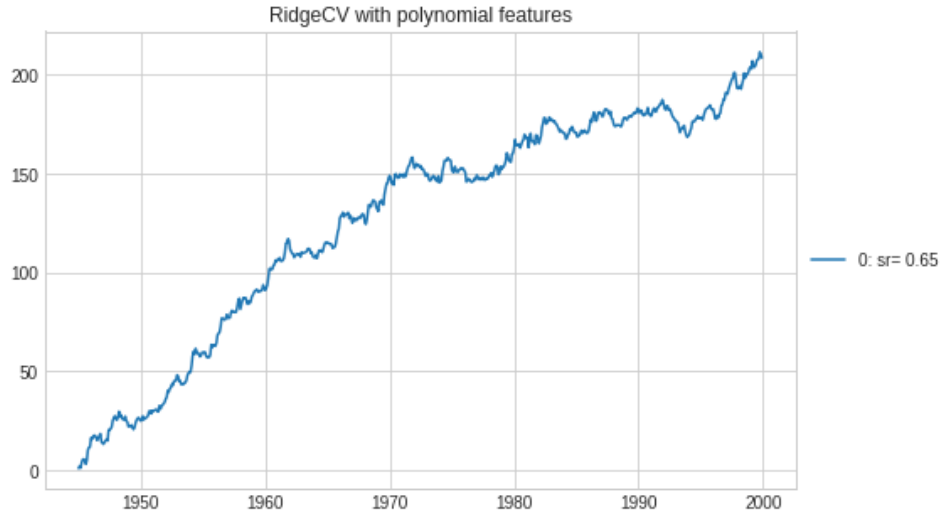
A first strategy is to use estimators that embed some form of cross-validation like `RidgeCV`. K-fold cross validation is described as follows:

Take a model with parameter s (e.g. the Ridge with tuning parameter `alpha`):

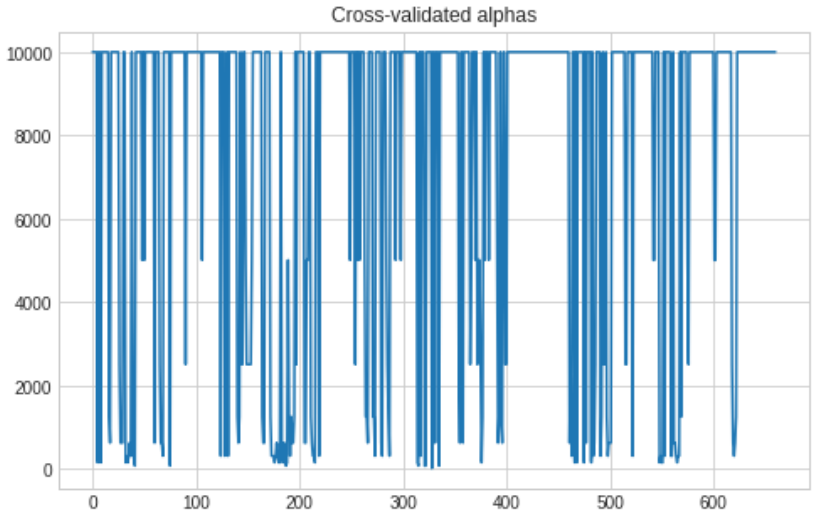
1. divide the data into K roughly equal parts ($K = 5$ or $K = 10$)
2. for each $k \in \{1, 2, ..., K\}$ fit the model with parameter s to the other $K - 1$ parts and compute its error $E_k(s)$ in predicting the k-th part.
3. the overall cross-validation error is then $CV(s) = \frac{1}{K} \sum_{k=1}^{K} E_k(s)$.
4. do this for many values of s and choose the value of s that minimize $CV(s)$

[2]:
```
alphas = np.exp(np.arange(np.log(10), np.log(10001), (np.log(10000) - np.log(10)) / 10))
```

[3]:
```
estimator = make_pipeline(
    StandardScaler(with_mean=False),
    PolynomialFeatures(degree=2),
    RidgeCV(alphas=alphas, cv=5),
    MeanVariance(),
)

m = Backtester(estimator).compute_holdings(features, target).compute_pnl(ret)
line(m.pnl_, cumsum=True, title="RidgeCV with polynomial features")
```



[4]:
```
line(
    pd.Series([m[2].alpha_ for m in m.estimators_]),
    title="Cross-validated alphas",
    legend=False,
)
```



In this example, the fitted alphas over rolling windows are not very stable (probably given the small rolling windows used here).

7.2 Random parameter search for Lightgbm

We first compute a `Lightgbm` benchmark with the fixed baseline parameters used in a previous section.

Some resources on the parameters of `Lightgbm` can be found in its documentation:

- https://lightgbm.readthedocs.io/en/latest/Parameters.html#learning-control-parameters

[5]:
```
estimator = make_pipeline(
    MultiLGBMRegressor(min_child_samples=5, n_estimators=25), MeanVariance()
)

pnl_lgb = {"fixed_params": Backtester(estimator).train(features, target, ret)}
```

We now do a search with random parameters drawn from predetermined distributions:

- the random parameter generators come from the `scipy.stats` module – in particular `randint`, `uniform` and `loguniform`.
- we use the `scikit-learn` function `ParameterSampler` as wrapper.

Setup:

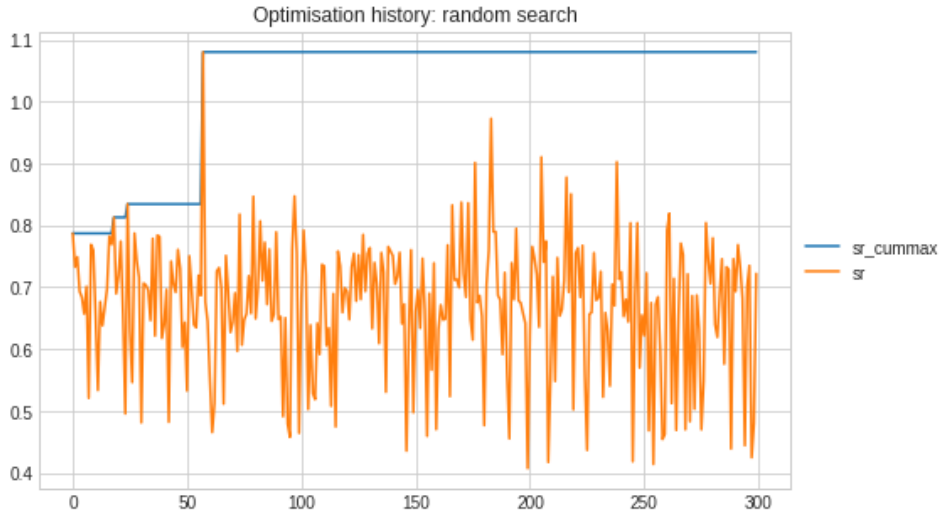
- the objective is to maximize the sharpe ratio over the early period 1945 to 1972 (as the `train` period).
- the evaluation is the performance of the backtest over the 1972-to-2000 period (as the `test` period).

[6]:
```
from scipy.stats import loguniform, randint, uniform
from sklearn.model_selection import ParameterSampler
```

[7]:
```
n_iter = 300
start_date = "1945-01-01"
end_date = "1972-04-01"
param_distributions = {
    "max_depth": randint(3, 10),
    "num_leaves": randint(2, 2**8),
    "n_estimators": randint(5, 50),
    "min_split_gain": uniform(0, 1.0),
    "min_child_samples": randint(1, 5),
    "reg_lambda": loguniform(1e-8, 1.0),
    "reg_alpha": loguniform(1e-8, 1.0),
}
```

[8]:
```
force_recompute = False
cache_dir = Path(os.getcwd()) / "cache"
if not cache_dir.is_dir():
    os.makedirs(cache_dir)

fname = cache_dir / "hpo_lgb.parquet"
if (force_recompute) | (not fname.exists()):
    results_ = {}
    for i, prm in enumerate(
        ParameterSampler(param_distributions=param_distributions, n_iter=n_iter)
    ):
        estimator = make_pipeline(LGBMRegressor(**prm), MeanVariance())
        pnl_ = Backtester(estimator, end_date=end_date).train(features, target, ret)
        prm.update({"sr": pnl_.pipe(sharpe_ratio)})
        results_[i] = pd.Series(prm)
    results = pd.DataFrame.from_dict(results_, orient="index").sort_values("sr")
    results.to_parquet(fname)
else:
    results = pd.read_parquet(fname)
```

[9]:
```
df = results.sort_index()[["sr"]].assign(sr_cummax=lambda x: x.sr.cummax())
line(df, title="Optimisation history: random search")
```



Using the sharpe ratio statistics presented in a previous section, we can compute a standard error around the maximum sharpe ratio:

[10]:
```
sr_max = results.iloc[-1]["sr"] / np.sqrt(12)
sr_std = np.sqrt(12) * np.sqrt((1 + 0.5 * sr_max**2) / len(ret[start_date:end_date]))
sr_range = results["sr"].pipe(lambda x: x.max() - x.min())
print(
    f"The sharpe ratio standard deviation at the maximum sharpe ratio (of {sr_max * np.sqrt(12):.
 ↪2f}) is {sr_std:.2f}"
)

print(f"The range of the sharpe ratios in the random search is {sr_range:.2f}")
```

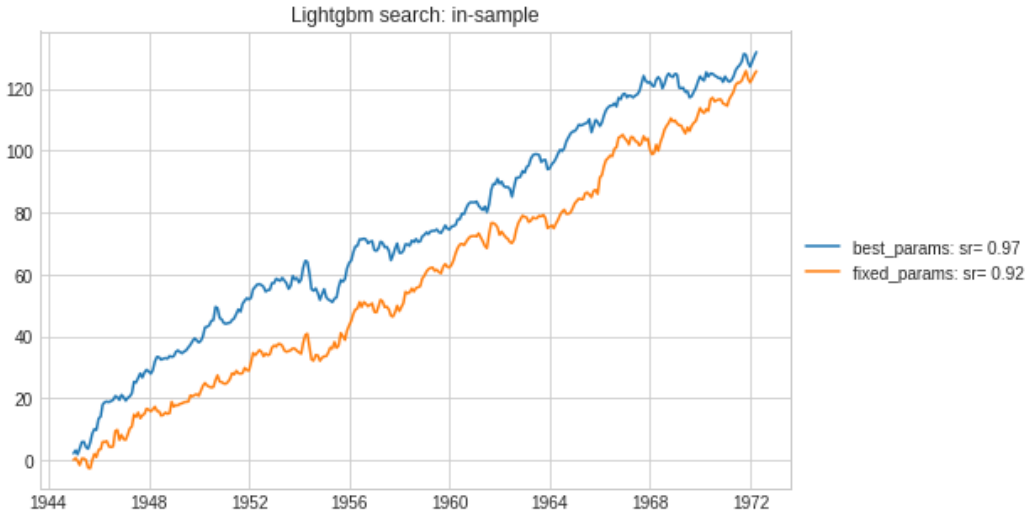
```
The sharpe ratio standard deviation at the maximum sharpe ratio (of 1.08) is
0.20
The range of the sharpe ratios in the random search is 0.67
```

[11]:
```
best_params = results.drop("sr", axis=1).iloc[-1].to_dict()
best_params["num_leaves"] = int(best_params["num_leaves"])
best_params["max_depth"] = int(best_params["max_depth"])
best_params["min_child_samples"] = int(best_params["min_child_samples"])
best_params["n_estimators"] = int(best_params["n_estimators"])
best_params
```

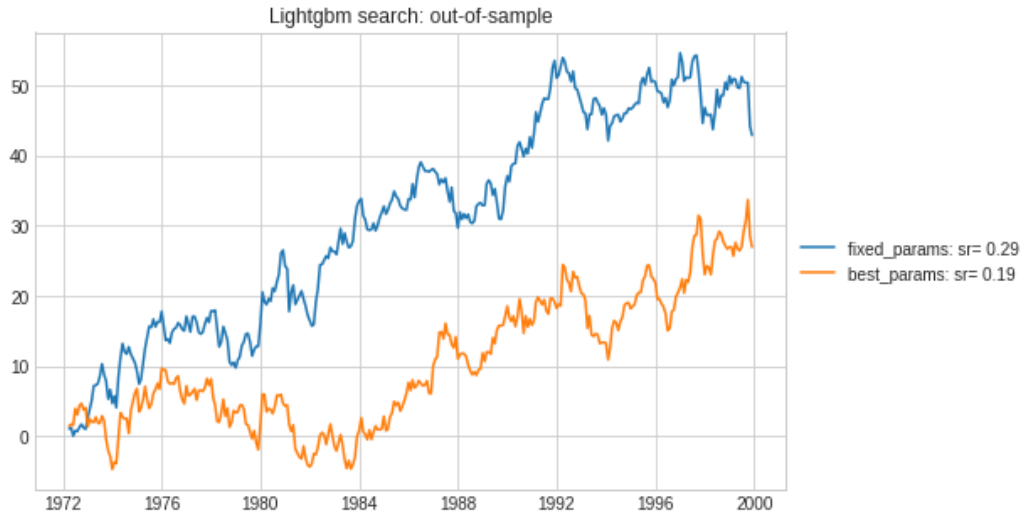
[11]:
```
{'max_depth': 3,
 'min_child_samples': 1,
 'min_split_gain': 0.22454462917437235,
 'n_estimators': 31,
 'num_leaves': 6,
 'reg_alpha': 2.8546321816265037e-05,
 'reg_lambda': 0.0054228446176497695}
```

[12]:
```
estimator = make_pipeline(MultiLGBMRegressor(**best_params), MeanVariance())


pnl_lgb["best_params"] = Backtester(estimator).train(features, target, ret)
```

[13]:
```
line(
    {k: v.loc[start_date:end_date] for k, v in pnl_lgb.items()},
    cumsum=True,
    title="Lightgbm search: in-sample",
)
```



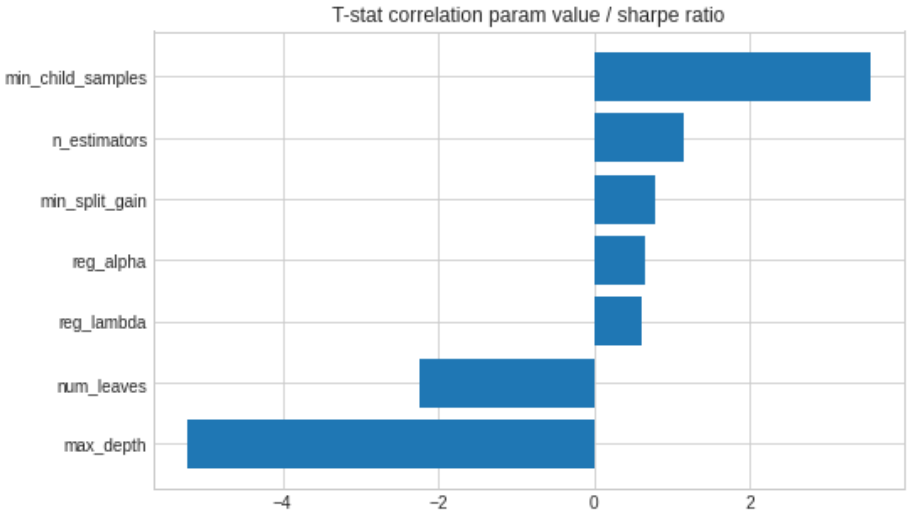
[14]:
```
line(
    {k: v.loc[end_date:] for k, v in pnl_lgb.items()},
    cumsum=True,
    title="Lightgbm search: out-of-sample",
)
```



What are the parameters that are correlated with the sharpe ratio?

[15]:
```
bar(
    results.corr()["sr"].mul(np.sqrt(n_iter)).drop("sr"),
    title="T-stat correlation param value / sharpe ratio",
```

```
    horizontal=True,
)
```



To assess more precisely the impact of parameters on the sharpe ratio, we run a regression.

[16]:
```
from statsmodels import api

m = api.OLS(results["sr"], api.add_constant(results.drop("sr", axis=1))).fit()
```

[17]:
```
m.summary()
```

[17]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:                     sr   R-squared:                       0.152
Model:                            OLS   Adj. R-squared:                  0.132
Method:                 Least Squares   F-statistic:                     7.497
Date:                Thu, 07 Dec 2023   Prob (F-statistic):           2.64e-08
Time:                        09:22:31   Log-Likelihood:                 269.05
No. Observations:                 300   AIC:                            -522.1
Df Residuals:                     292   BIC:                            -492.5
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
=====
                         coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
-----
const                  0.7083      0.029     24.470      0.000       0.651
0.765
max_depth             -0.0159      0.003     -5.482      0.000      -0.022
-0.010
min_child_samples      0.0180      0.005      3.369      0.001       0.007
0.029
min_split_gain         0.0255      0.019      1.321      0.187      -0.012
0.063
n_estimators           0.0007      0.000      1.516      0.131      -0.000
```

```
0.002
num_leaves           -0.0002   7.98e-05     -1.977      0.049      -0.000
-6.77e-07
reg_alpha             0.0276      0.033      0.831      0.407      -0.038
0.093
reg_lambda            0.0137      0.041      0.332      0.740      -0.067
0.095
==============================================================================
Omnibus:                        0.101   Durbin-Watson:                   0.265
Prob(Omnibus):                  0.951   Jarque-Bera (JB):                0.013
Skew:                           0.001   Prob(JB):                        0.993
Kurtosis:                       3.033   Cond. No.                     1.09e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 1.09e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

[ ]:

Chapter 8

Risk

A key ingredient of portfolio construction is the ability to predict portfolio risk (in particular, with a risk-model) to be able to properly size the positions. Of course, an important assumption of the markowitz optimisation is the normality of returns. In this section, we evaluate the non-normality of backtest pnls.

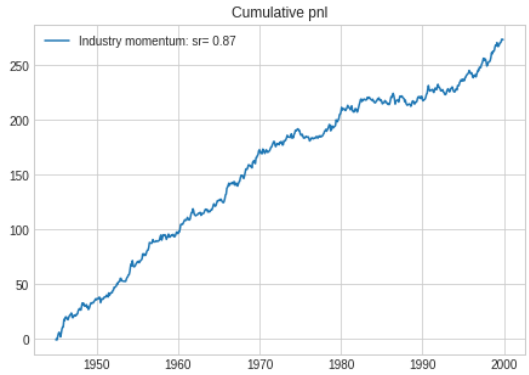
8.1 Risk in the industry momentum backtest

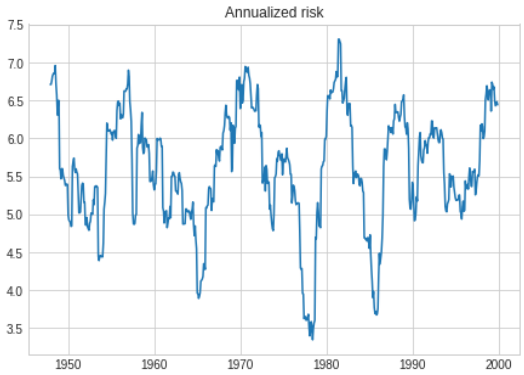
[2]:
```
from skfin.metrics import drawdown, sharpe_ratio
```

We first compute the Industry momentum benchmark.

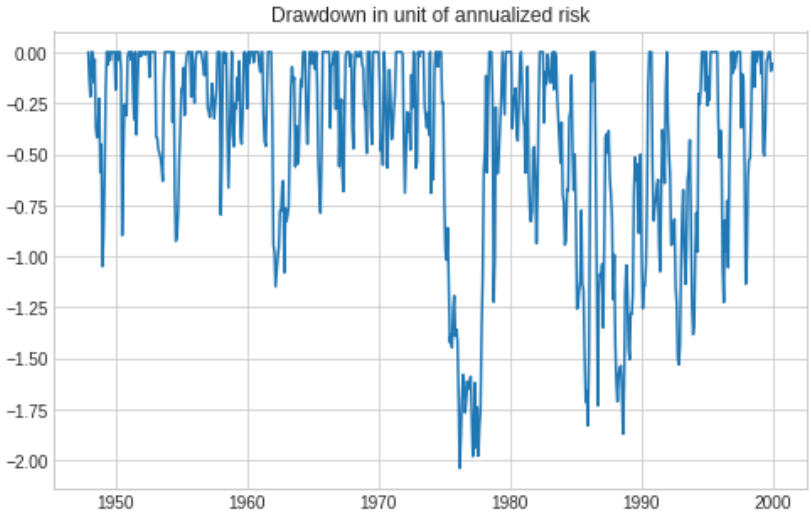
[3]:
```
m = Backtester(MeanVariance()).compute_holdings(features, target).compute_pnl(ret)
h0, pnl0, estimators0 = m.h_, m.pnl_, m.estimators_
```

[4]:
```
fig, ax = plt.subplots(1, 2, figsize=(16, 5))
line(
    pnl0.rename("Industry momentum"),
    cumsum=True,
    loc="best",
    title="Cumulative pnl",
    ax=ax[0],
)
line(
    pnl0.rolling(36).std().mul(np.sqrt(12)),
    title="Annualized risk",
    legend=False,
    ax=ax[1],
)
```



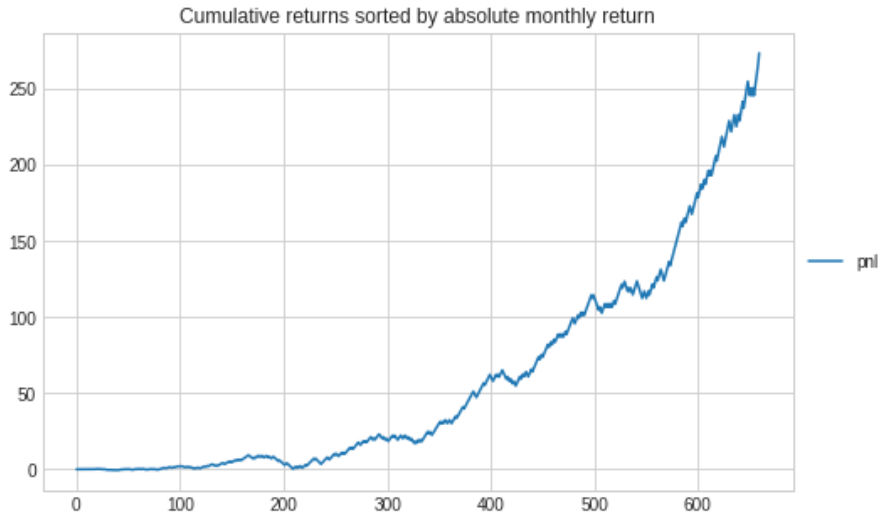


[5]:
```
line(
    pnl0.pipe(drawdown),
    title="Drawdown in unit of annualized risk",
    legend=False,
    figsize=(8, 5),
)
```



The following graph shows that on the period up to 2000, large absolute returns tend to be positive. It turns out that in the following period, the pnl of Momentum becomes left-skewed with large negative returns. For instance, February/March 2009 is a famous example of a Momentum drawdown.

[6]:
```
line(
    pnl0.rename("pnl")
    .to_frame()
    .assign(pnl_abs=lambda x: x.pnl.abs())
    .sort_values("pnl_abs")
    .reset_index(drop=True)["pnl"],
    cumsum=True,
    title="Cumulative returns sorted by absolute monthly return",
    legend_sharpe_ratio=False,
)
```



8.2 Return covariance eigenvalues

The risk-model is defined here as the covariance of returns V. To understand its impact on the backtest, it is important to remember that in the mean-variance optimisation, it is the inverse of the covariance matrix V^{-1} that is used.

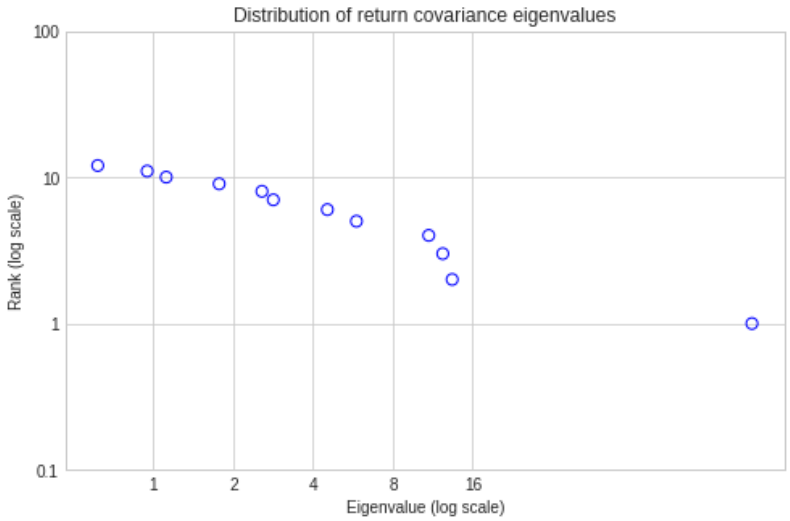
Viewed from the point of view of a singular value decomposition, the smallest eigenvalues of V are not only estimated with noise, but their impact is magnified in V^{-1}, leading to potentially significant noise in the estimate of positions.

[7]:
```
for train, test in m.cv_.split(ret):
    break
```

[8]:
```
U, D, _ = np.linalg.svd(ret.iloc[train].cov())
```

The graph below shows that the largest eigenvalue is two-order of magnitude larger than the smallest one.

[9]:
```
df = pd.Series(D, np.arange(1, 13))
scatter(
    df,
    xscale="log",
    yscale="log",
    xlabel="Eigenvalue (log scale)",
    ylabel="Rank (log scale)",
    xticks=[1, 2, 4, 8, 16],
    yticks=[0.1, 1, 10, 100],
    title="Distribution of return covariance eigenvalues",
)
```



[10]:
```
print(f"The ratio of the largest to the smallest eigenvalue is {D[0]/D[-1]:.1f}")
```

```
The ratio of the largest to the smallest eigenvalue is 293.1
```

8.3 Risk model estimation

The insight of Ledoit and Wolf (2004) is to use a weighted average of two matrices to reduce estimation error

- the empirical covariance matrix V is asymptotically an unbiased estimated – but with a slow convergence for sample samples
- there are biased estimators but with a faster rate of convergence — for instance the diagonal D of V – and on smaller samples, such biased estimators can be more efficient than the unbiased ones
- The covariance matrix used in the portfolio optimisation is

$$V_\omega = \omega D + (1-\omega)V.$$

How to determine ω? Ledoit and Wolf (2004) minimize a norm that applies to matrices (Frobenius norm). For a given portfolio h, the risk-bias is given by

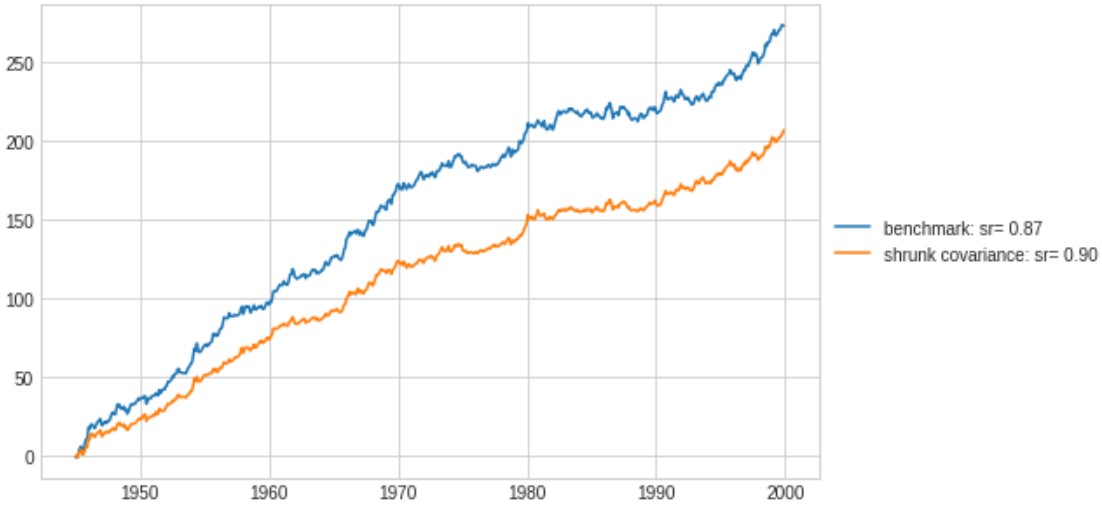
$$\text{RiskBias} = Std\left[\frac{h^T r}{\sqrt{h^T \hat{V}(\omega) h}}\right] - 1,$$

where the variance is evaluated over empirical returns.

[11]:
```
from sklearn.covariance import LedoitWolf, ShrunkCovariance
```

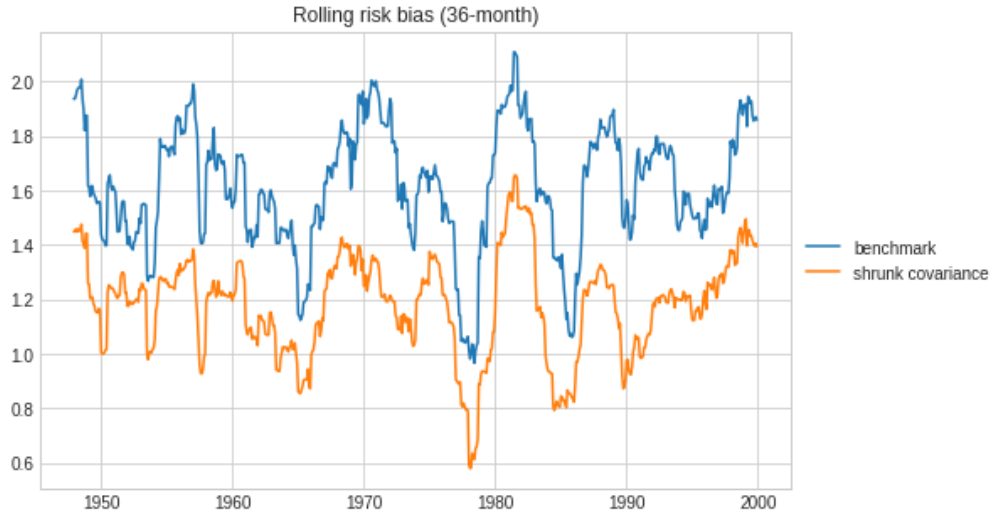
The default value of the `shrinkage` parameter for `ShrunkCovariance` is 0.1. When `shrinkage=0`, there is no shrinkage and when `shrinkage=1`, all the off-diagonal terms are set to zero and the covariance matrix is diagonal.

[12]:
```
transform_V_ = lambda x: ShrunkCovariance(shrinkage=0.1).fit(x).covariance_
m = (
    Backtester(MeanVariance(transform_V=transform_V_))
    .compute_holdings(features, target)
    .compute_pnl(ret)
)
h, pnl, estimators = m.h_, m.pnl_, m.estimators_
line({"benchmark": pnl0, "shrunk covariance": pnl}, cumsum=True)
```



The estimation of risk with the shrunk covariance is much closer to the ex-ante risk (of 1).

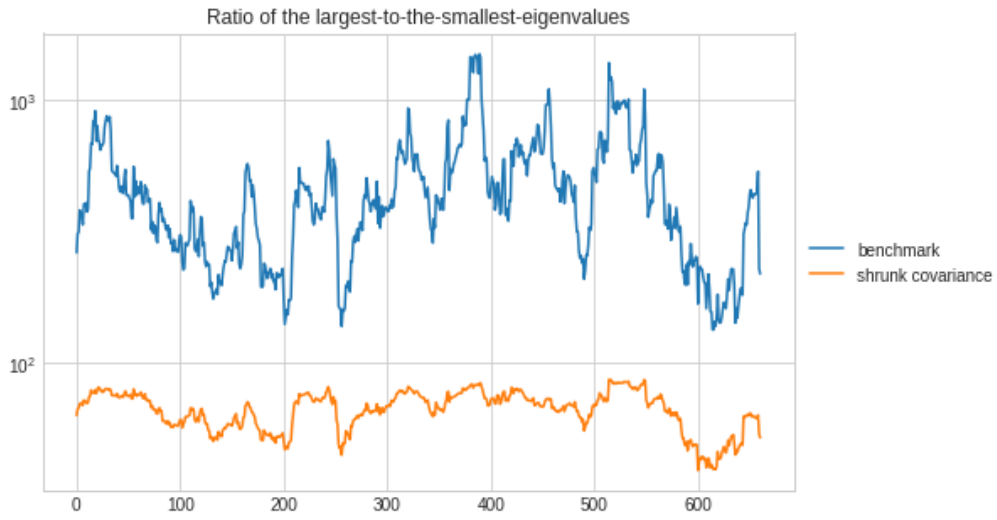
[13]:
```
line(
    {"benchmark": pnl0.rolling(36).std(), "shrunk covariance": pnl.rolling(36).std()},
    title="Rolling risk bias (36-month)",
)
```



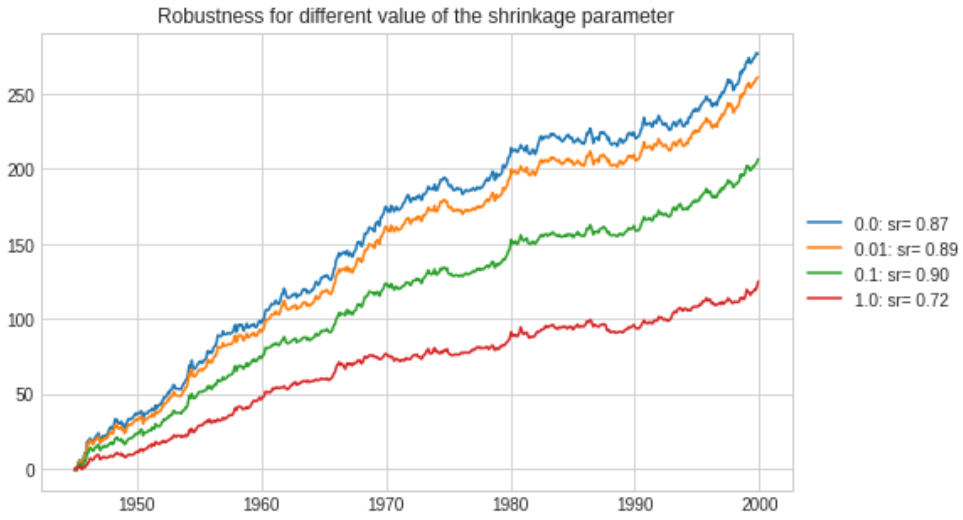
The ratio of the largest to the smallest eigenvalue is an order of magnitude smaller for the backtest

with the shrunk covariance relative to the benchmark.

[14]:
```
get_eigenvalues = lambda estimators: pd.DataFrame(
    [np.linalg.svd(m.V_, compute_uv=False) for m in estimators]
)
ratio_largest_smallest_eigenvalue = lambda x: x.pipe(
    lambda x: x.iloc[:, 0] / x.iloc[:, -1]
)

eigenvalues0 = get_eigenvalues(estimators0)
eigenvalues = get_eigenvalues(estimators)

line(
    {
        "benchmark": eigenvalues0.pipe(ratio_largest_smallest_eigenvalue),
        "shrunk covariance": eigenvalues.pipe(ratio_largest_smallest_eigenvalue),
    },
    yscale="log",
    title="Ratio of the largest-to-the-smallest-eigenvalues",
)
```

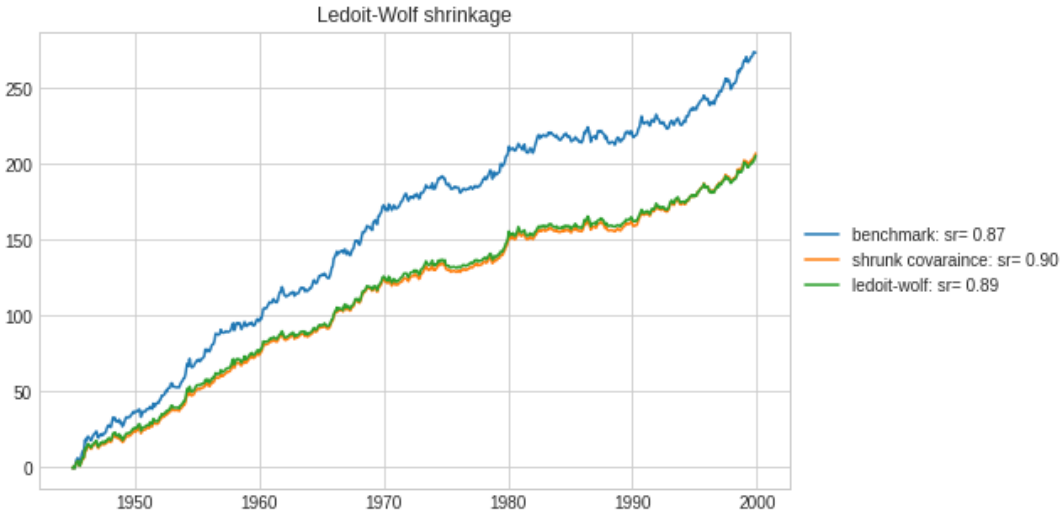


[15]:
```
pnls = {}
for shrinkage in [0, 0.01, 0.1, 1]:
    transform_V_ = lambda x: ShrunkCovariance(shrinkage=shrinkage).fit(x).covariance_
    estimator = MeanVariance(transform_V=transform_V_)
    pnls[shrinkage] = Backtester(estimator).train(features, target, ret)
line(
    pnls, cumsum=True, title="Robustness for different value of the shrinkage parameter"
)
```



A related shrinkage is to use the `LedoitWolf` method to determine the shrinkage and it yield similar performance.

[16]:
```
transform_V_ = lambda x: LedoitWolf().fit(x).covariance_
estimator = MeanVariance(transform_V=transform_V_)
pnl_ = Backtester(estimator).train(features, target, ret)
line(
    {"benchmark": pnl0, "shrunk covaraince": pnl, "ledoit-wolf": pnl_},
    cumsum=True,
    title="Ledoit-Wolf shrinkage",
)
```



The key empirical point is that the sharpe ratio is maximized for a covariance that involves a small amount of shrinkage.

8.4 Non-normality

At the stock-level, non-normality (e.g. skewness or kurtosis) may not be a significant problem: - the portfolio might be short a stock with negative skewness

- part of the stock kurtosis might diversify away at the portfolio level.

But factor-level non-normality is harder to diversify – especially the returns of strategy based on risk-premia (which are generally non-normally distributed):

- by definition, in risk-off environments, these strategies do not pay off and the left tails of the return distribution can be very "thick" leading to "rare disasters"

In this section, we discuss how to document non-normality:

- use some test statistics that involve skewness and kurtosis.
- illustrate some methods using industry momentum.

The statistics of higher moments:

- $skewness = \frac{E[(x-\bar{x})^3]}{\sigma_x^3}$
- $kurtosis = \frac{E[(x-\bar{x})^4]}{\sigma_x^4}$
- Jarque-Bera statistics $JB = \frac{T}{6}\left(skewness^2 + \frac{(kurtosis-3)^2}{4}\right)$
- If the observations $\{x_1, .., x_T\}$ are independant and follow a Normal distribution, then
  1. $skewness = 0$
  2. $kurtosis = 3$
  3. JB follows of Chi-squared distribution with two degrees of freedom.
- The third assertion provides a way to test whether a variable is Normally distributed.

[17]:
```
from scipy.stats import jarque_bera, kurtosis, skew
```

Normally distributed random data:

- we can *not* reject the null hypothesis that the data follows is a Normal distribution!

[18]:
```
(T, N) = ret.shape
```

[19]:
```
x = np.random.normal(0, 1, T)  # create random values based on a normal distribution

print(f"Excess kurtosis of normal distribution (should be 0): {kurtosis(x):.3f}")
print(f"Skewness of normal distribution (should be 0): {skew(x):.3f}")
print(f"Jarque beta of normal distribution (should be 0): {jarque_bera(x)[0]:.3f}")
```

```
Excess kurtosis of normal distribution (should be 0): 0.106
Skewness of normal distribution (should be 0): 0.024
Jarque beta of normal distribution (should be 0): 0.492
```

With `fisher=False`, the function `kurtosis` computes the raw kurtosis (the default is `fisher=True`).

[20]:
```
kurtosis(np.random.normal(0, 1, T), fisher=False)
```

[20]:
```
3.0473405000074054
```

Uniformly distributed random data:

- we can reject at any confidence level the null hypothesis that the data follows a Normal distribution!

[21]:
```
x = np.random.uniform(0, 1, T)  # create random values based on a normal distribution

print(f"Excess kurtosis of normal distribution (should be 0): {kurtosis(x):.3f}")
print(f"Skewness of normal distribution (should be 0): {skew(x):.3f}")
print(f"Jarque beta of normal distribution (should be 0): {jarque_bera(x)[0]:.3f}")
```

```
Excess kurtosis of normal distribution (should be 0): -1.206
Skewness of normal distribution (should be 0): -0.015
Jarque beta of normal distribution (should be 0): 53.505
```

Industry momentum data:

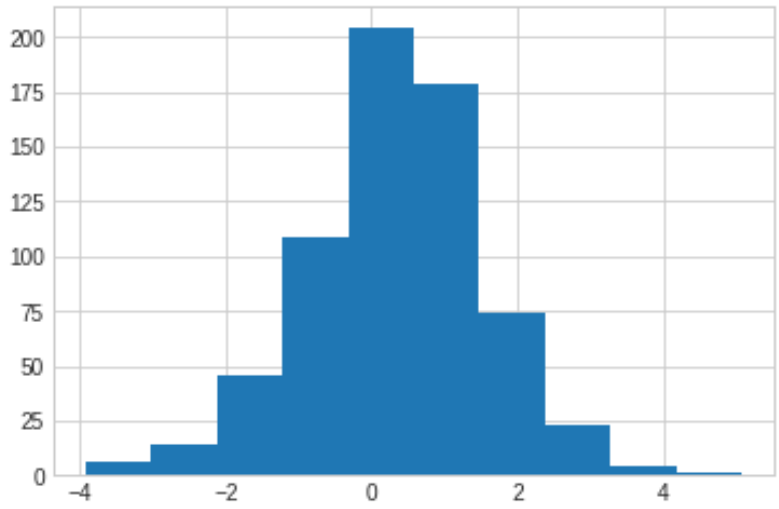
- we can also reject the null hypothesis of a Normal distribution

[22]:
```
x = pnl

print(f"Excess kurtosis of normal distribution (should be 0): {kurtosis(x):.3f}")
print(f"Skewness of normal distribution (should be 0): {skew(x):.3f}")
print(f"Jarque beta of normal distribution (should be 0): {jarque_bera(x)[0]:.3f}")
```

```
Excess kurtosis of normal distribution (should be 0): 0.683
Skewness of normal distribution (should be 0): -0.144
Jarque beta of normal distribution (should be 0): 15.093
```

[23]:
```
print(f"Pnl std={pnl.std():.2f}")
pnl.hist();
```

```
Pnl std=1.21
```



The test in the notebook shows that the normality assumption is easily rejected for the momentum strategy return – there are a lot of observations (T large) and significant deviations.

8.5 The statistics of rolling sharpe ratio

We now focus on 3-year rolling sharpe ratio:

- A 3-year horizon is the natural horizon to evaluate the performance of investment strategies.
- Significant underperformance over 3 years is almost always a show-stopper!
- In particular, we assess the impact of skewness and kurtosis on the rolling sharpe ratio.

Lemma [Lo, 1996]. When the underlying data is iid Normally distributed, the limiting distribution of the estimated monthly sharpe ratio $\hat{S}$ (relative to the true unobserved S) is

$$\sqrt{T}(\hat{S} - S) \to N\left(0, 1 + \frac{IR^2}{2}\right)$$

Proof. For independently Normally distributed returns, we have

$$\sqrt{T}\begin{bmatrix} \hat{\mu} - \mu \\ \hat{\sigma}^2 - \sigma^2 \end{bmatrix} \to N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}; \begin{bmatrix} \sigma^2 & 0 \\ 0 & 2\sigma^4 \end{bmatrix}\right)$$

For $S = g(\mu, \sigma^2) = \frac{\mu}{\sqrt{\sigma^2}}$, the asymptotic variance of $\hat{S}$ is

$$\begin{bmatrix} \frac{\partial g}{\partial \mu} \\ \frac{\partial g}{\partial (\sigma^2)} \end{bmatrix}^T \begin{bmatrix} \sigma^2 & 0 \\ 0 & 2\sigma^4 \end{bmatrix} \begin{bmatrix} \frac{\partial g}{\partial \mu} \\ \frac{\partial g}{\partial (\sigma^2)} \end{bmatrix} = \frac{1}{\sigma^2}\sigma^2 + \left(-\frac{\mu}{2\sigma^3}\right)^2 (2\sigma^4)$$

Corollary. The minimum monthly S_0 for a monthly strategy where the Sharpe ratio is always statistically different from 0 at the 5%-confidence level over a window of T months is given by

$$S_0 - 1.96\sqrt{\frac{1 + \frac{S_0^2}{2}}{T}} = 0 \Leftrightarrow S_0 = \sqrt{\frac{1}{\frac{T}{1.96^2} - 1/2}}$$

- Rule of thumb: for a 36-month horizon, then the monthly $S_0 \approx 0.35$ or after annualization

$$S_0^{\text{annualized}} \approx .35 \times \sqrt{12} = 1.16$$

When the returns are not Normally distributed (*skewness* $\neq$ 0 and (excess) *kurtosis* $\neq$ 0), the Central Limit theorem still ensures the asymptotic normality. (In what follows, `kurtosis` refers to the "excess kurtosis.)

Lemma [Mertens, 2002] The asymptotic distribution of $\hat{S}$ is

$$\sqrt{T}(\hat{S} - S) \to N(0, V_\infty)$$

where $V_\infty = 1 + \frac{S^2}{2} - skewness \times S + \frac{kurtosis \times S^2}{4}$.

Proof. The asymptotic variance of $\begin{bmatrix} \hat{\mu} - \mu \\ \hat{\sigma}^2 - \sigma^2 \end{bmatrix}$ is:

$$E\begin{bmatrix} (r_t - \mu)^2 & (r_t - \mu)[(r_t - \mu)^2 - \sigma^2] \\ (r_t - \mu)[(r_t - \mu)^2 - \sigma^2] & [(r_t - \mu)^2 - \sigma^2]^2 \end{bmatrix} = \begin{bmatrix} \sigma^2 & E[(r_t - \mu)^3] \\ E[(r_t - \mu)^3] & 2E[(r_t - \mu)^4] - \sigma^4 \end{bmatrix}$$

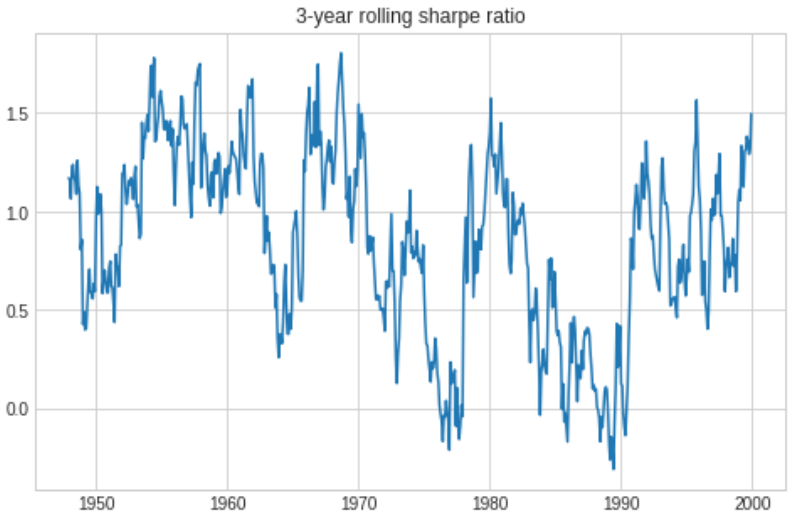
Corollary. The minimum $S_0(skewness, kurtosis)$ for a monthly strategy where the information is always statistically different from 0 at the 5%-confidence level over a window of T months is increasing in *kurtosis* and decreasing in *skewness*.

Proof. The function $f(S, skewness, kurtosis)$

$$= S - 1.96\sqrt{\frac{1 + \frac{S^2}{2} - skewness \times S + \frac{kurtosis \times S^2}{4}}{T}}$$

is increasing in S and *skewness* and decreasing in *kurtosis*, so that S_0 defined by $f(S_0, skewness, kurtosis) = 0$ is decreasing in *skewness* and increasing in *kurtosis*.

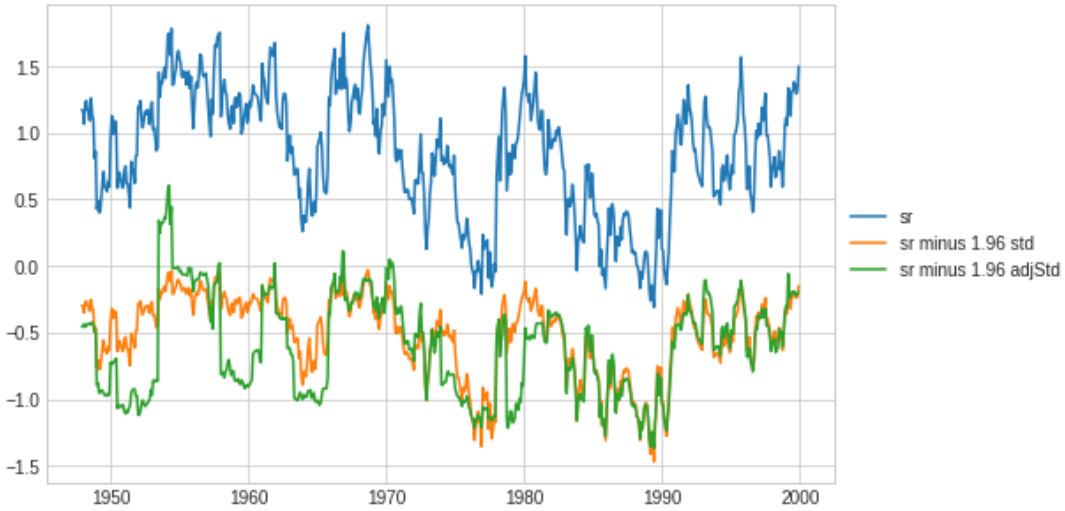
[24]:
```
sr3y = pnl.rolling(36).apply(sharpe_ratio)
line(sr3y, legend=False, title="3-year rolling sharpe ratio", start_date="1945")
```



Below we compute the two lower bounds for the sharpe ratio.

[25]:
```
sr3y_lb = sr3y - 1.96 * np.sqrt((1 + 0.5 * sr3y**2) / 36) * np.sqrt(12)
skew3y = pnl.rolling(36).apply(skew)
kurtosis3y = pnl.rolling(36).apply(kurtosis)
sr3y_lb_bis = sr3y - 1.96 * np.sqrt(
    (1 + 0.5 * sr3y**2 - skew3y * sr3y + 0.25 * kurtosis3y * sr3y**2) / 36
) * np.sqrt(12)
```

[26]:
```
line(
    {"sr": sr3y, "sr minus 1.96 std": sr3y_lb, "sr minus 1.96 adjStd": sr3y_lb_bis},
    start_date="1945",
)
```



Even generic momentum strategies (including industry momentum) have a fairly strong Sharpe ratio…

- … but the performance includes kurtosis/skewness risk premia, with occasional deep underperformance over a 3-year horizon.

Chapter 9

Factors

In this section, we compare the performance of the Industry momentum derived in previous sections with other factors (e.g. Stock Momentum, Value, Size, etc.). In particular, we discuss the Style regression of Sharpe (1992).

9.1 Style analysis

As introduced by Sharpe (1992), Style Analysis is the process of determining what type of investment behaviour an investor or a money manager employs when making investment decisions

Regression to determine the factor exposures $\langle \beta_1, ...\beta_K \rangle$ where:

$$r = \alpha + \beta_1 r_1^{\Phi} + ... + \beta_K r_K^{\Phi} + \epsilon$$

Additional constraints might be added to "regularize" the regression such as non-negative exposures :

$\beta_k \geq 0$ and/or sum equals 1: $\sum_{k=1}^{K} \beta_k = 1$.

Frazzini, Kabiller and Pedersen (2013) state that:

> Berkshire Hathaway has realized a Sharpe ratio of 0.76, higher than any other stock or mutual fund with a history of more than 30 years, and Berkshire has a significant alpha to traditional risk factors."

How did Warren Buffet do it?

- Use a "style analysis" approach applied to equity factors to address this question.

The main regression is:

$$r_t = \alpha + \beta_1 MKT_t + \beta_2 SMB_t + \beta_3 HML_t + \beta_4 UMD_t + \beta_5 BAB_t + \beta QMJ_t + \epsilon_t$$

where the factors are

- r_t : excess return of the Berkshire Hathaway stock
- MKT_t (market): excess market return
- SMB_t (size): small minus big

- HML_t (value): high book-to-market minus low book-to-market
- UMD_t (momentum): up minus down
- BAB_t (betting-against-beta): safe (low beta) minus risky (high beta)
- QMJ_y (quality): quality minus junk

Can we replicate this finding? Fortunately Steve Lihn (on GitHub) already did it.

Data: github.com/slihn/buffetts_alpha_R/archive/master.zip

Table 4

Buffett's Exposures: What Kind of Companies does Berkshire Own?

	Berkshire stock 1976 - 2011		
Alpha	**12.1%** (3.19)	**9.2%** (2.42)	6.3% (1.58)
MKT	**0.84** (11.65)	**0.83** (11.70)	**0.95** (10.98)
SMB	**-0.32** -(3.05)	**-0.32** -(3.13)	-0.15 -(1.15)
HML	**0.63** (5.35)	**0.38** (2.79)	**0.46** (3.28)
UMD	0.06 (0.90)	-0.03 -(0.40)	-0.05 -(0.71)
BAB		**0.37** (3.61)	**0.29** (2.67)
QMJ			**0.43** (2.34)
R2 bar	0.25	0.27	0.28

- The characteristics of the investment of Warren Buffet: high loadings on replicable factors such as beta, size, value and quality – and a negative loading on momentum.
- At least in this replication of the paper's results (with slightly different data), the intercept is no longer statistically significant – it might still be economically significant!

[3]:
```
from skfin.datasets import load_buffets_data

data = load_buffets_data(cache_dir="data").assign(
    excess_return=lambda x: x["BRK.A"] - x["RF"]
)
```

```
INFO:skfin.datasets:logging from cache directory: data/ffdata_brk13f.parquet
```

[4]:
```
from statsmodels import api

m1 = api.OLS(data["excess_return"], api.add_constant(data["MKT"])).fit()
m1.summary()
```

[4]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:          excess_return   R-squared:                       0.443
Model:                            OLS   Adj. R-squared:                  0.440
Method:                 Least Squares   F-statistic:                     141.7
Date:                Thu, 07 Dec 2023   Prob (F-statistic):           2.08e-24
Time:                        09:23:07   Log-Likelihood:                -471.71
No. Observations:                 180   AIC:                             947.4
Df Residuals:                     178   BIC:                             953.8
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.2431      0.250      0.972      0.332      -0.251       0.737
MKT            0.6242      0.052     11.903      0.000       0.521       0.728
==============================================================================
Omnibus:                       55.501   Durbin-Watson:                   2.104
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              251.454
Skew:                          -1.075   Prob(JB):                     2.50e-55
Kurtosis:                       8.377   Cond. No.                         4.79
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
"""
```

[5]:
```
summaries = []
for cols in [
    ["MKT", "SMB", "HML", "UMD"],
    ["MKT", "SMB", "HML", "UMD", "BAB"],
    ["MKT", "SMB", "HML", "UMD", "BAB", "QMJ"],
]:
    m_ = api.OLS(data["excess_return"], api.add_constant(data[cols])).fit()
    summaries += [m_.summary()]
```

[6]:
```
def prettify_table(tbl):
    df = pd.DataFrame(tbl.tables[1].data)
    idx = df.iloc[1:, 0]
    return pd.DataFrame(
        df.iloc[1:, [1, 3]].astype(float).values,
        index=idx.rename(None),
        columns=["coef", "tstat"],
    ).stack()
```

[7]:
```
pd.concat([prettify_table(v) for v in summaries], axis=1).fillna(0).round(2)
```

[7]:

		0	1	2
const	coef	0.28	0.25	0.07
	tstat	1.32	1.21	0.33
MKT	coef	0.69	0.70	0.82
	tstat	14.41	14.31	12.31
SMB	coef	-0.32	-0.31	-0.20
	tstat	-4.28	-4.11	-2.32
HML	coef	0.49	0.45	0.46
	tstat	7.03	5.53	5.83
UMD	coef	-0.12	-0.13	-0.14
	tstat	-3.27	-3.40	-3.62

```
BAB    coef    0.00   0.06   0.02
       tstat   0.00   0.94   0.36
QMJ    coef    0.00   0.00   0.29
       tstat   0.00   0.00   2.61
```

The coefficients are qualitatively close from the results in the paper – with the except of the BAB coefficients not being statistically significant.

9.2 Industry momentum factor exposure

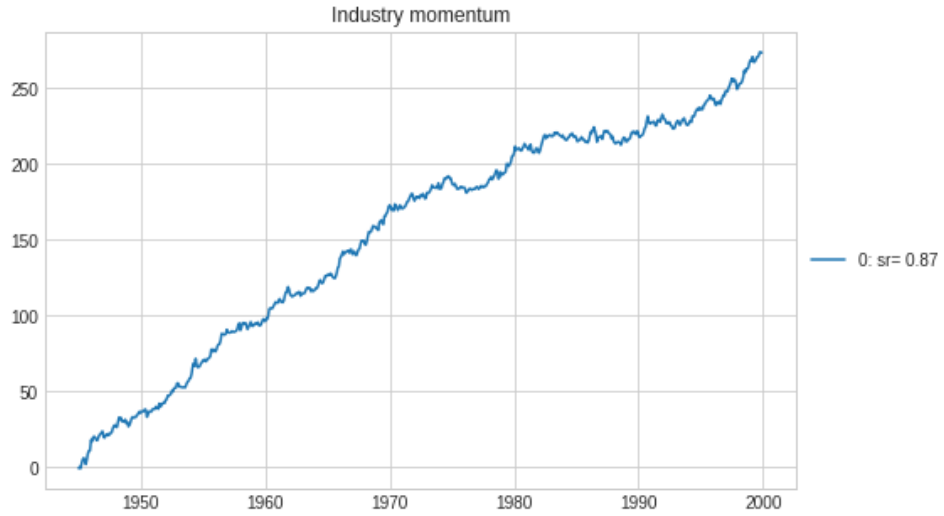
In this section, we go back to the Industry momentum backtest and decompose it on the factors as computed by Ken French.

[8]:
```
returns_data = load_kf_returns(cache_dir="data")
ret = returns_data["Monthly"]["Average_Value_Weighted_Returns"][:"1999"]
```

```
INFO:skfin.datasets:logging from cache directory: data/12_Industry_Portfolios
```

[9]:
```
transform_X = lambda x: x.rolling(12).mean().fillna(0)
transform_y = lambda x: x.shift(-1)
features = transform_X(ret)
target = transform_y(ret)
```

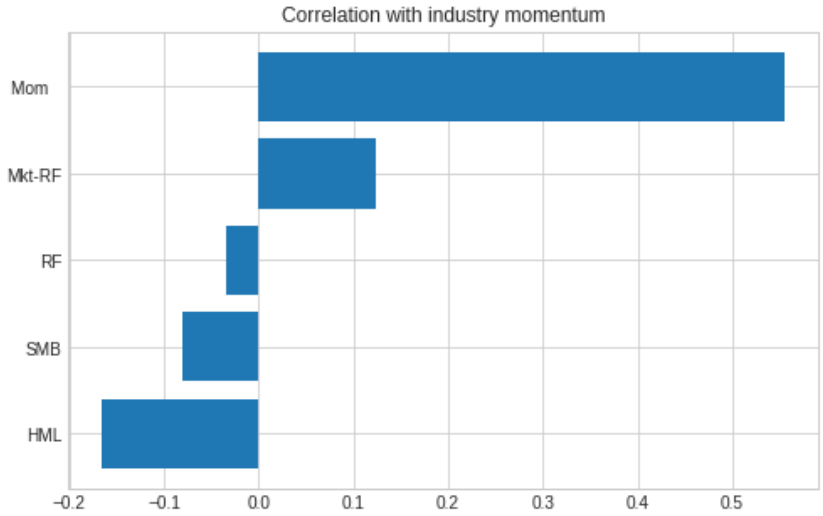
[10]:
```
pnl0 = Backtester(MeanVariance()).train(features, target, ret)
line(pnl0, cumsum=True, title="Industry momentum")
```



[11]:
```
files = ["F-F_Research_Data_Factors", "F-F_Momentum_Factor"]
df = pd.concat([load_kf_returns(c)["Monthly"] for c in files], axis=1)["1945":"1999"]
```

```
INFO:skfin.datasets:logging from cache directory: data/F-F_Research_Data_Factors
INFO:skfin.datasets:logging from cache directory: data/F-F_Momentum_Factor
```

[12]:
```
bar(df.corrwith(pnl0), horizontal=True, title="Correlation with industry momentum")
```



[13]:
```
data = df.join(pnl0.rename("IndustryMom"))
```

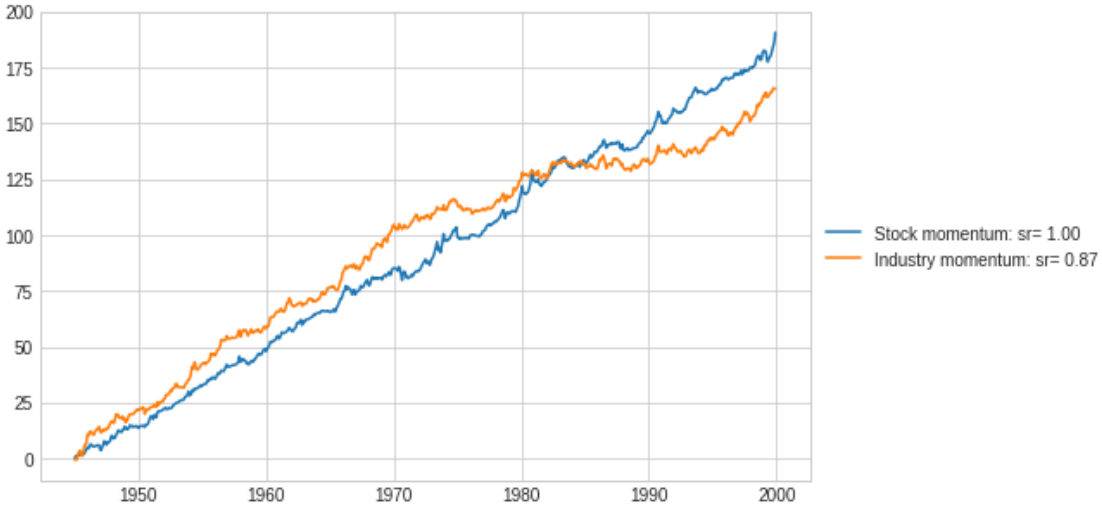
[14]:
```
m = api.OLS(
    data["IndustryMom"], api.add_constant(data.drop("IndustryMom", axis=1))
).fit()
```

[15]:
```
m.summary()
```

[15]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:            IndustryMom   R-squared:                       0.323
Model:                            OLS   Adj. R-squared:                  0.317
Method:                 Least Squares   F-statistic:                     62.29
Date:                Thu, 07 Dec 2023   Prob (F-statistic):           4.00e-53
Time:                        09:23:08   Log-Likelihood:                -1138.0
No. Observations:                 660   AIC:                             2288.
Df Residuals:                     654   BIC:                             2315.
Df Model:                           5
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.2068      0.101      2.047      0.041       0.008       0.405
Mkt-RF         0.0368      0.014      2.706      0.007       0.010       0.064
SMB           -0.0254      0.022     -1.174      0.241      -0.068       0.017
HML           -0.0378      0.023     -1.671      0.095      -0.082       0.007
RF            -0.1547      0.213     -0.726      0.468      -0.573       0.263
Mom            0.2857      0.018     16.186      0.000       0.251       0.320
==============================================================================
Omnibus:                       18.767   Durbin-Watson:                   1.843
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               33.186
Skew:                           0.179   Prob(JB):                     6.22e-08
Kurtosis:                       4.038   Cond. No.                         19.1
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
```

```
specified.
"""
```

[16]:
```
line(
    pd.concat({"Stock momentum": df["Mom   "], "Industry momentum": pnl0}, axis=1).pipe(
        lambda x: x.div(x.std())
    ),
    cumsum=True,
)
```



The main issue with this statistical decomposition is that the estimation is done "full sample". In the next section, we estimate the Momentum loading on rolling windows.

9.3 Residual pnl

To run the rolling estimation decomposition, we use the function `fit_predict` used in previous sections.

[17]:
```
from skfin.backtesting import fit_predict
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import TimeSeriesSplit
```

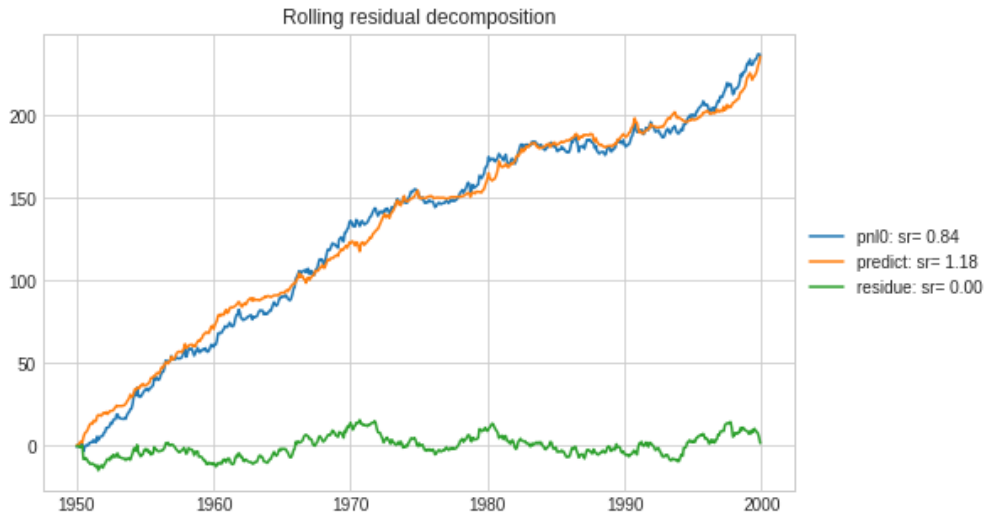
[18]:
```
start_date = "1945-01-01"
max_train_size = 60
test_size = 1
params = dict(max_train_size=max_train_size, test_size=test_size, gap=0)
params["n_splits"] = (len(data) - max_train_size) // test_size
cv_ = TimeSeriesSplit(**params)
```

[19]:
```
pnl_hat, estimator_ = zip(
    *[
        fit_predict(
            estimator=LinearRegression(),
            X=data.drop(["IndustryMom", "RF"], axis=1),
            y=data["IndustryMom"],
            train=train,
            test=test,
```

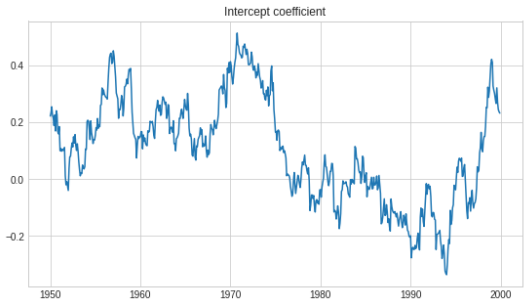
```
            return_estimator=True,
        )
        for train, test in cv_.split(data["IndustryMom"])
    ]
)
```

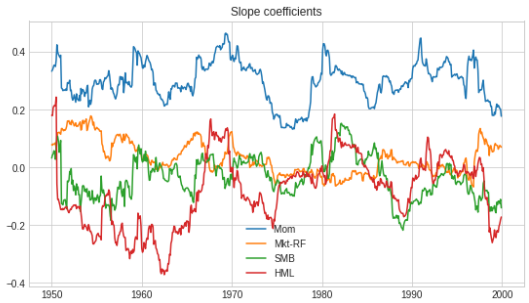
[20]:
```
pnl_hat = pd.Series(
    np.concatenate(pnl_hat), index=data["IndustryMom"].index[max_train_size:]
)
```

[21]:
```
line(
    {
        "pnl0": pnl0[max_train_size:],
        "predict": pnl_hat,
        "residue": pnl0[max_train_size:] - pnl_hat,
    },
    cumsum=True,
    title="Rolling residual decomposition",
)
```



[22]:
```
fig, ax = plt.subplots(1, 2, figsize=(20, 5))
line(
    pd.DataFrame([m.intercept_ for m in estimator_], index=pnl_hat.index),
    title="Intercept coefficient",
    ax=ax[0],
    legend=False,
)

line(
    pd.DataFrame(
        [m.coef_ for m in estimator_],
        columns=data.drop(["IndustryMom", "RF"], axis=1).columns,
        index=pnl_hat.index,
    ),
    title="Slope coefficients",
    loc="best",
    ax=ax[1],
)
```





Over this period, the simple Industry momentum strategy seems to have zero residual relative to other factors.

Chapter 10

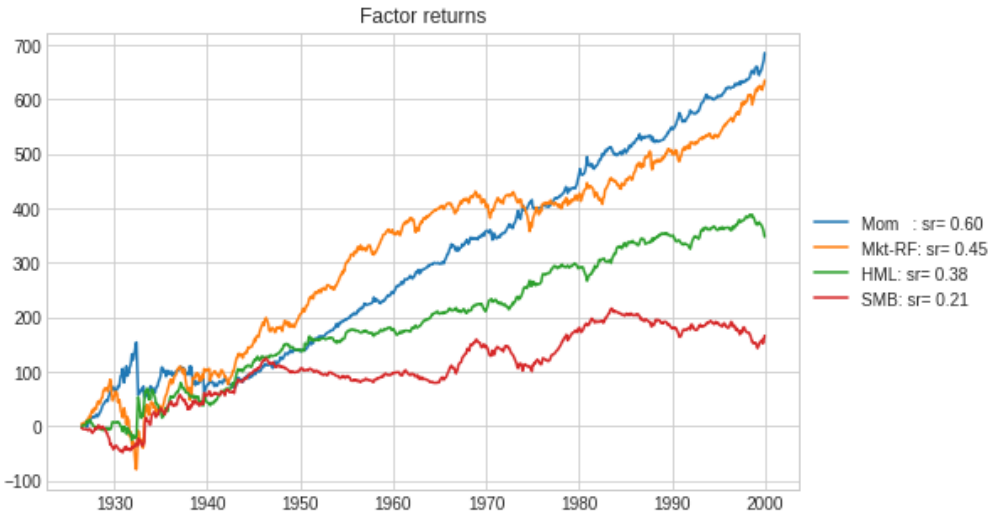
Factor timing

In this section, we discuss a simple Momentum strategy to time the quantitative factors previsouly considers (Stock Momentum, Value, Size and the Market).

[2]:
```
files = ["F-F_Research_Data_Factors", "F-F_Momentum_Factor"]
ret = pd.concat([load_kf_returns(c)["Monthly"] for c in files], axis=1).drop('RF', axis=1)[:
↪'1999']
```

```
INFO:skfin.datasets:logging from cache directory: data/F-F_Research_Data_Factors
INFO:skfin.datasets:logging from cache directory: data/F-F_Momentum_Factor
```

[3]:
```
line(ret, cumsum=True, title='Factor returns')
```



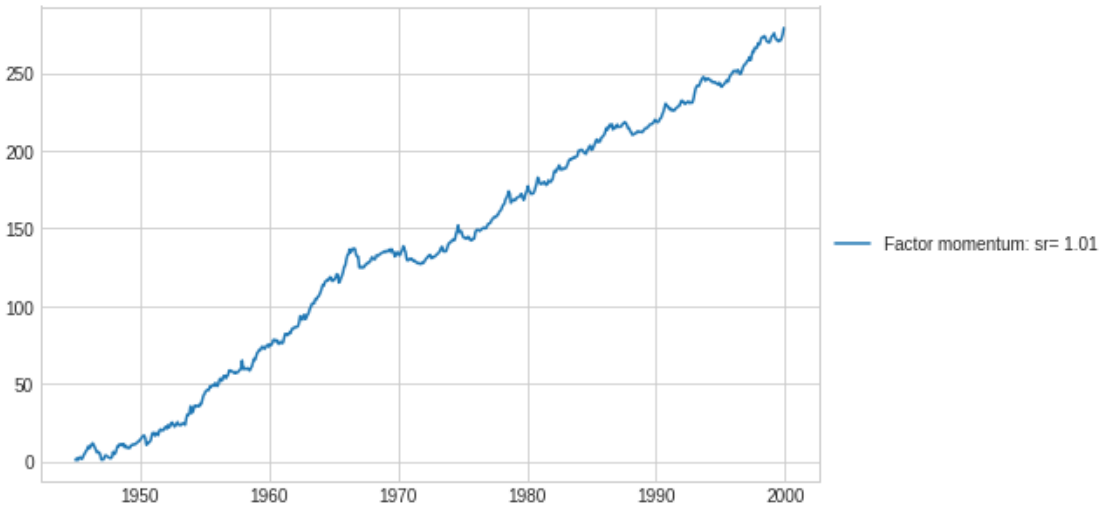
As in the industry Momentum case, we use the previous 12-month returns as the prediction of returns.

[4]:
```
def transform_X(df, window=12):
    return df.rolling(window=window).mean()

def transform_y(df):
    return df.shift(-1)
```

```
X = transform_X(ret)
y = transform_y(ret)
```

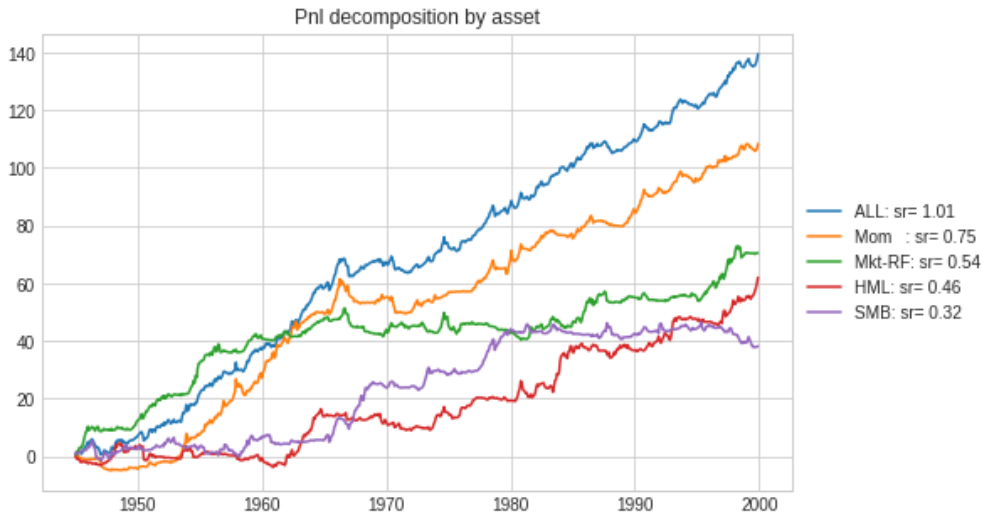
While in the sector rotation case, we imposed a "cash-neutrality constraint" (so that the sum of the positions on a given is zero and the strategy is long-short), this constraint is not necessary in the factor timing case. Below, we set the constraint vector `A` to `None` in the `MeanVariance` estimator.

[5]:
```
m = Backtester(estimator=MeanVariance(A=None), name="Factor momentum")
pnl_ = m.train(X, y, ret)
line(pnl_, cumsum=True)
```



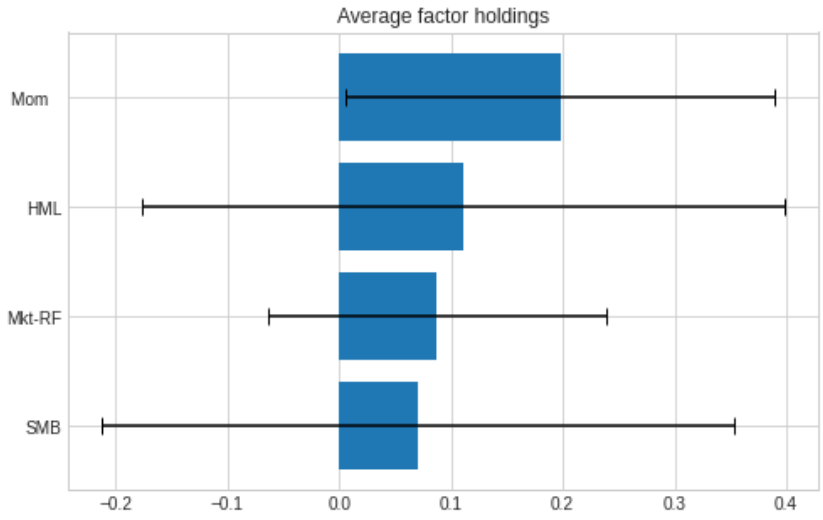
Below we decompose the strategy pnl by the contribution of each asset. (We also divide the total pnl by the square-root of the number of assets so that all the pnl shown below have approximately the same standard devation.)

[6]:
```
line(m.h_.shift(1).mul(ret)\
        .dropna(how='all', axis=0)\
        .assign(ALL = lambda x: x.sum(axis=1).div(np.sqrt(ret.shape[1]))), cumsum=True,
    title='Pnl decomposition by asset')
```



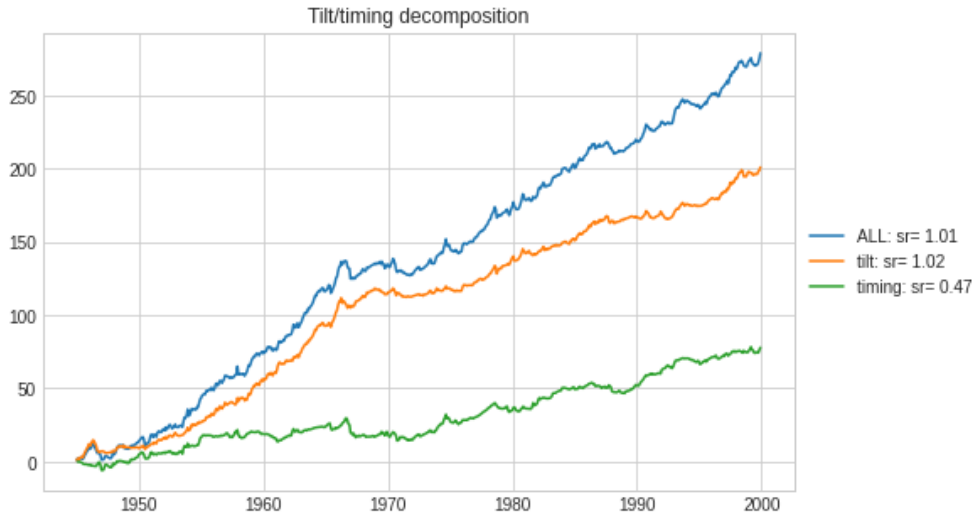
The chart below shows the average positions on the factors is positive, but there is a significant standard deviation on the position.

[7]:
```
bar(m.h_.mean(), m.h_.std(), horizontal=True, title='Average factor holdings')
```



To see whether the performance is coming from factor "tilts", we decompose the pnls into the slow positions (over a one-year halflife) and the faster residual positions (defined as the difference between the positions and slow positions). This tilt/timing decomposition below shows some contribution from timing.

[8]:
```
line(
    {
        "ALL": pnl_,
        "tilt": m.h_.ewm(halflife=12).mean().shift(1).mul(ret).sum(axis=1).replace(0, np.nan).
↪dropna(),
        "timing": m.h_.sub(m.h_.ewm(halflife=12).mean()).shift(1).mul(ret).sum(axis=1).
↪replace(0, np.nan).dropna(),
    },
    cumsum=True, title='Tilt/timing decomposition'
)
```


Tilt/timing decomposition
250
200
150
100
50
0
1950
1960
1970
1980
1990
2000
ALL: sr= 1.01
tilt: sr= 1.02
timing: sr= 0.47

Chapter 11

Ensemble

11.1 Ensemble

Rather than choosing a single estimator (or set of parameters) among many, another stategy is to combine all the possible estimators/parameters. `scikit-learn` allows to do that with classes such as `VotingRegressor`.

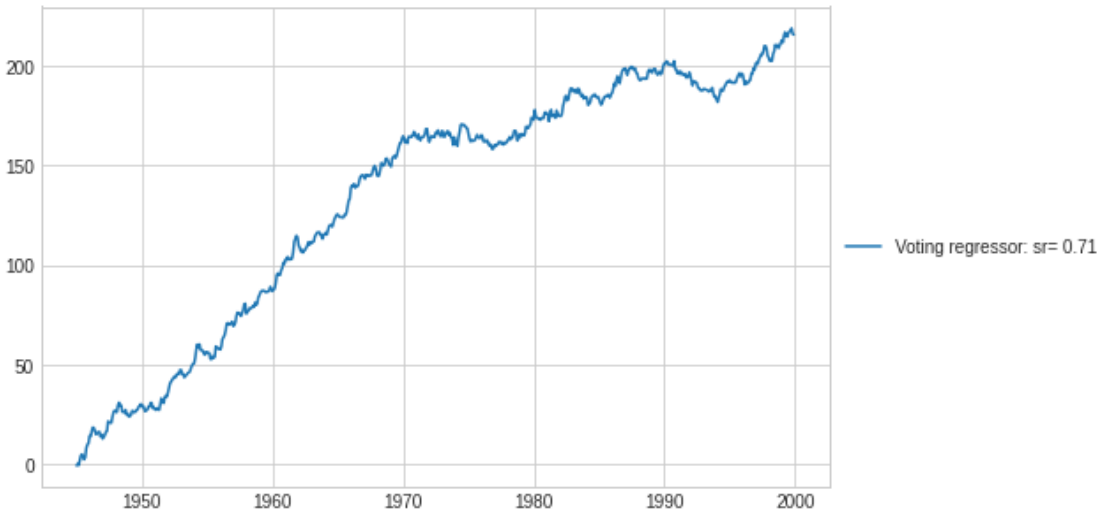
[2]:
```
from skfin.estimators import Ridge
from sklearn.ensemble import VotingRegressor


class VotingRegressor(VotingRegressor):
    def transform(self, X):
        return self.predict(X)
```

[3]:
```
estimators_ = [
    ("ridge1", Ridge(alpha=1)),
    ("ridge2", Ridge(alpha=100)),
]
```

By default, `VotingRegressor` applies equal weights across regressors.

[4]:
```
estimator = make_pipeline(
    StandardScaler(with_mean=False),
    MultiOutputRegressor(VotingRegressor(estimators=estimators_)),
    MeanVariance(),
)
```

[5]:
```
pnl_ = Backtester(estimator, name="Voting regressor").train(features, target, ret)
line(pnl_, cumsum=True)
```



In `scikit-learn`, there is also a `StackingRegressor` but it requires a bit more work to make it work with `MultiOutputRegressor` (and constraints on transform/regressors).

11.2 Rolling ensemble backtest

In this section, we build a custom ensemble method to learn weights on different estimators from pnls.

11.2.1 StackingBacktester

We consider three estimators:

- the simple Industry momentum.
- a strategy that learns cross-industry effect with `Ridge`.
- a strategy that learns cross-industry effect with `Lightgbm`.

[6]:
```
estimators = {
    "momentum": MeanVariance(),
    "ridge": make_pipeline(StandardScaler(with_mean=False), Ridge(), MeanVariance()),
    "lightgbm": make_pipeline(
        MultiOutputRegressor(
            LGBMRegressor(min_child_samples=5, n_estimators=25, n_jobs=1)
        ),
        MeanVariance(),
    ),
}
```

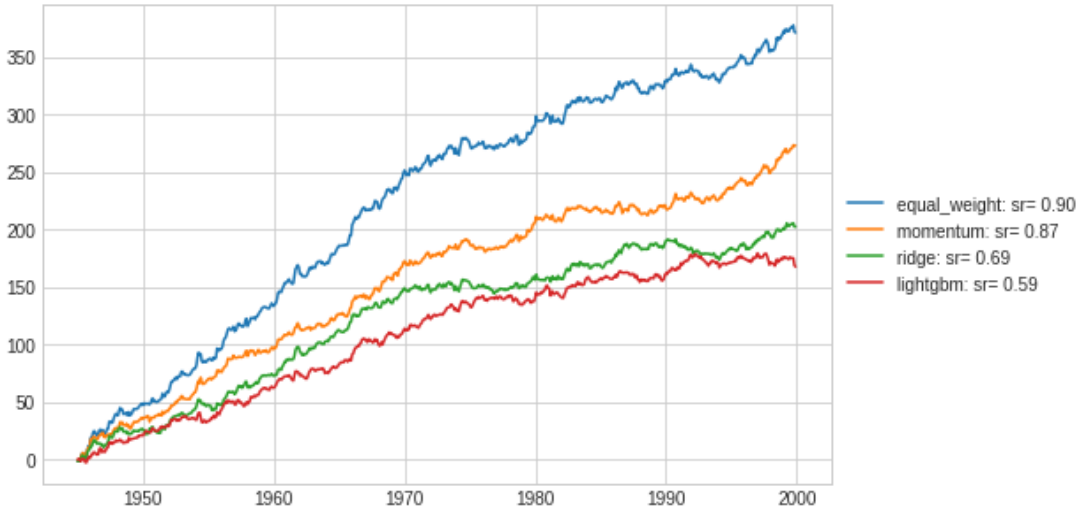
[7]:
```
pnls = pd.concat(
    {k: Backtester(v).train(features, target, ret) for k, v in estimators.items()},
    axis=1,
)
```

To construct an equal-weight strategy, we scale the sum of the pnl by the square-root of the number of strategy:

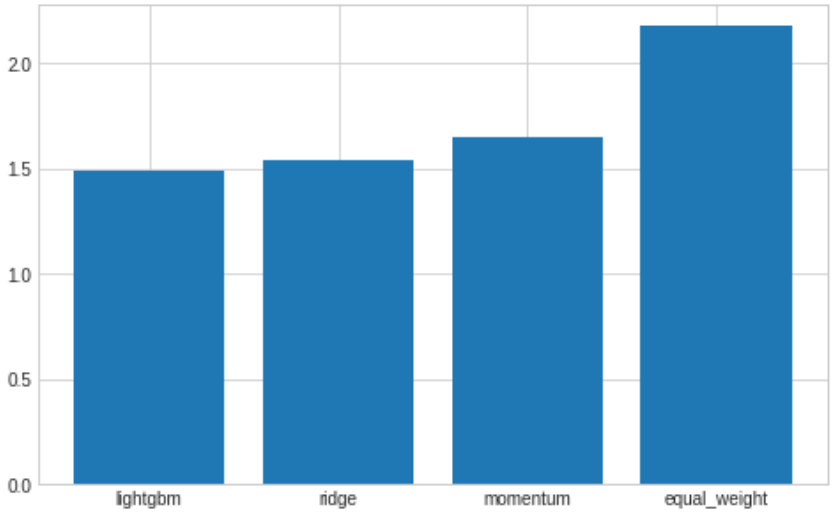
$$pnl_{\text{equal weight}} = \frac{\sum_{n=1}^{n=N} pnl_n}{\sqrt{N}}.$$

In this case, if the individual pnls pnl_n are identically and independently distributed (with the same standard deviation), then $pnl_{\text{equal weight}}$ has the same ex-ante standard deviation.

[8]:
```
pnls_ = pnls.assign(equal_weight=lambda x: x.sum(axis=1).div(np.sqrt(x.shape[1])))
line(pnls_, cumsum=True)
```



[9]:
```
bar(pnls_.std())
```



The average correlation is not particularly high, which explains with some simple ensemble seems to help.

[10]:
```
print(
    f"The average pnl correlation between estimators is {pnls.corr().stack().loc[lambda x: x!=1].
 ↪mean():.2f}"
)
```

```
The average pnl correlation between estimators is 0.47
```

We introduce a `StackingBacktester` with the `sklearn` api.

[11]:
```
%%writefile ../skfin/ensemble.py
from dataclasses import dataclass

import numpy as np
import pandas as pd
from skfin.mv_estimators import Mbj
from sklearn.base import BaseEstimator
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import TimeSeriesSplit


@dataclass
class StackingBacktester:
    estimators: dict
    ret: pd.DataFrame
    max_train_size: int = 36
    test_size: int = 1
    start_date: str = "1945-01-01"
    end_date: str = None
    window: int = 60
    min_periods: int = 60
    final_estimator: BaseEstimator = Mbj()

    def __post_init__(self):
        self.ret = self.ret[: self.end_date]
        self.cv = TimeSeriesSplit(
            max_train_size=self.max_train_size,
            test_size=self.test_size,
            n_splits=1
            + len(self.ret.loc[self.start_date : self.end_date]) // self.test_size,
        )

    def train(self, features, target):
        N_estimators = len(self.estimators)
        cols = self.ret.columns
        idx = self.ret.index[
            np.concatenate([test for _, test in self.cv.split(self.ret)])
        ]

        _h = {k: [] for k in list(self.estimators.keys()) + ["ensemble"]}
        _pnls = {k: [] for k in self.estimators.keys()}
        _coef = []
        for i, (train, test) in enumerate(self.cv.split(self.ret)):
            h_ = {}
            if i > self.min_periods:
                pnl_window = np.stack(
                    [np.array(v[-self.window :]) for k, v in _pnls.items()], axis=1
                )
                coef_ = self.final_estimator.fit(pnl_window).coef_
                _coef += [coef_]
            else:
                _coef += [np.zeros(N_estimators)]
            for k, m in self.estimators.items():
```

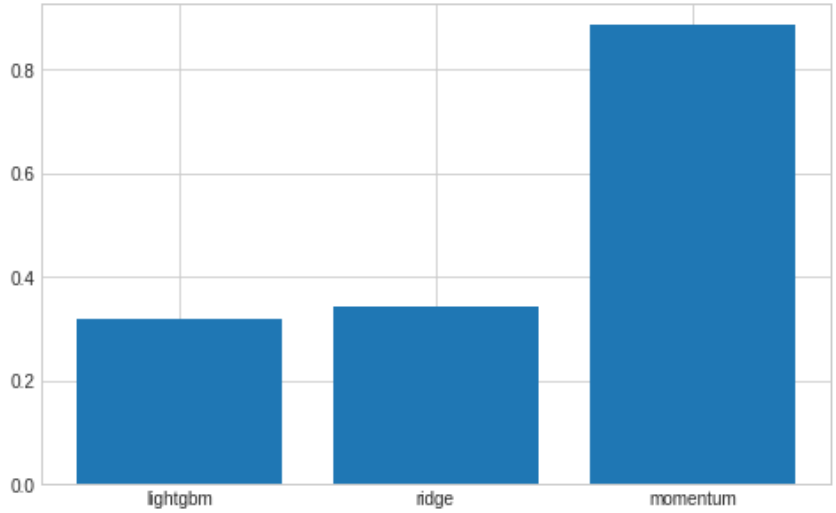
```
                m.fit(features.iloc[train], target.iloc[train])
                h_[k] = m.predict(features.iloc[test])
                _h[k] += [h_[k]]
                if i + 1 < len(idx):
                    _pnls[k] += [self.ret.loc[idx[i + 1]].dot(np.squeeze(h_[k]))]
            if i > self.min_periods:
                h_ensemble = (
                    np.stack([np.squeeze(v) for v in h_.values()], axis=1)
                    .dot(coef_)
                    .reshape(-1, 1)
                )
                V_ = m.named_steps["meanvariance"].V_
                h_ensemble = h_ensemble / np.sqrt(
                    np.diag(h_ensemble.T.dot(V_.dot(h_ensemble)))
                )
            else:
                h_ensemble = np.zeros([len(cols), 1])
            _h["ensemble"] += [h_ensemble.T]

        self.h_ = {
            k: pd.DataFrame(np.concatenate(_h[k]), index=idx, columns=cols)
            for k in _h.keys()
        }
        self.pnls_ = pd.concat(
            {
                k: v.shift(1).mul(self.ret).sum(axis=1)[self.start_date :]
                for k, v in self.h_.items()
            },
            axis=1,
        )
        self.coef_ = pd.DataFrame(
            np.stack(_coef), index=idx, columns=self.estimators.keys()
        )
        return self
```

```
Overwriting ../skfin/ensemble.py
```

[12]:
```
from skfin.mv_estimators import Mbj
```

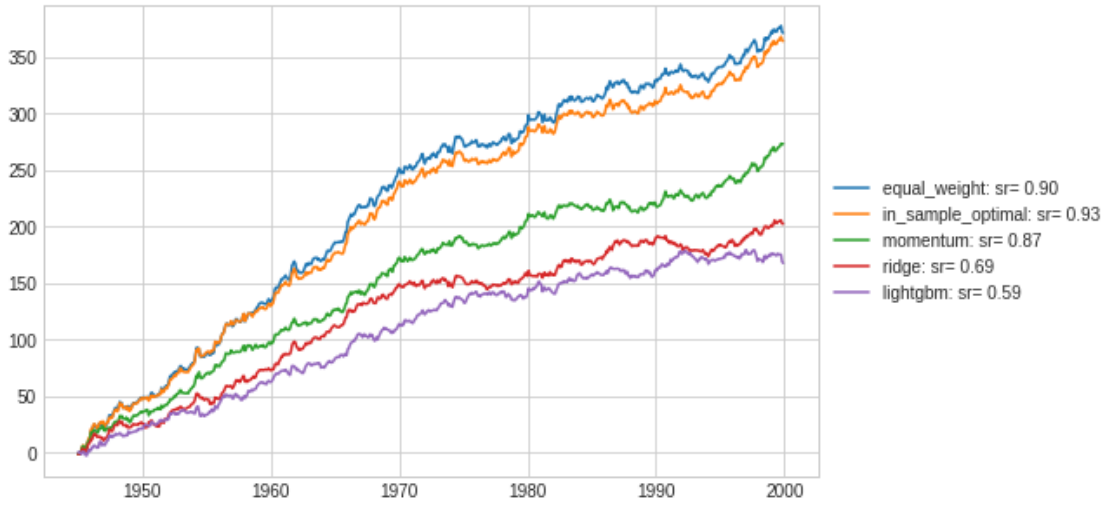
We first use the Britten-Jones (1999) trick (with the `Mbj` estimator) to compute the unconstrained mean-variance weigts over the full sample:

[13]:
```
m = Mbj()
m.fit(pnls)
bar(pd.Series(m.coef_, index=pnls.columns))
```



These in-sample optimal weights improve even more the sharpe ratio – but this is `in-sample`!

[14]:
```
line(pnls_.assign(in_sample_optimal=Mbj().fit_transform(pnls)), cumsum=True)
```

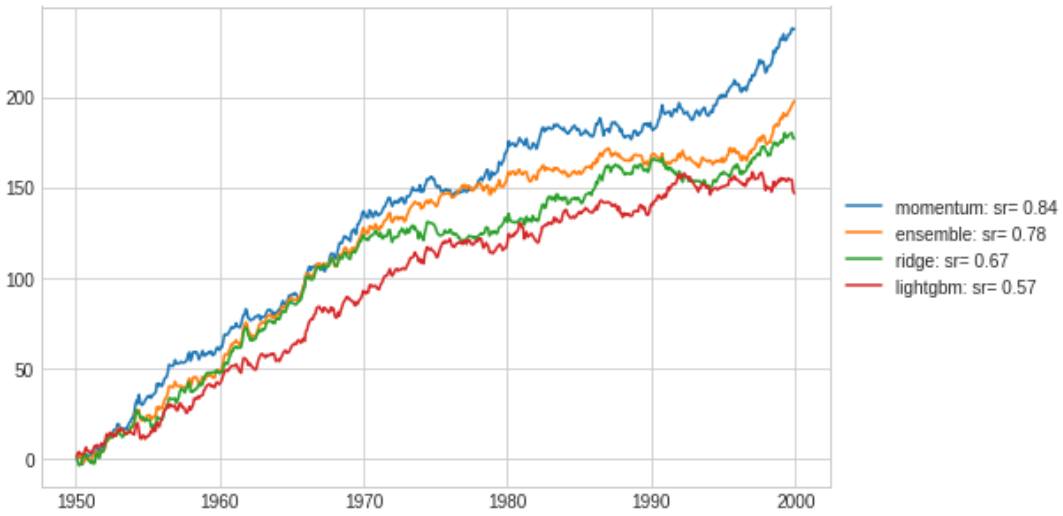


The `StackingBacktester` computes the performance with the learned weights over rolling windows of 60 months (5 years).

[15]:
```
from skfin.ensemble import StackingBacktester
```

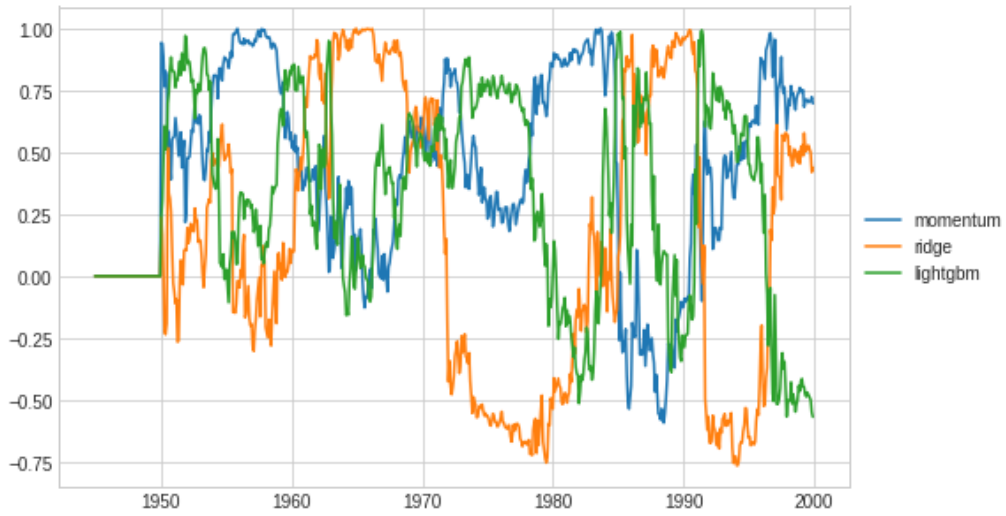
[16]:
```
m = StackingBacktester(estimators=estimators, ret=ret, window=60, min_periods=60).train(
    features, target
)
pnls = pnls.assign(ensemble_mbj=m.pnls_["ensemble"])
```

[17]:
```
line(m.pnls_["1950-02-01":], cumsum=True)
```



To understand why the performance is lower, it is useful to look at the weights – in this case, the weights are often negative.

[18]:
```
line(m.coef_)
```



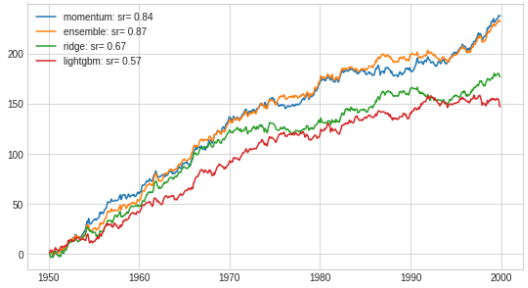
We redo the exercise with a positive-weight constraint.

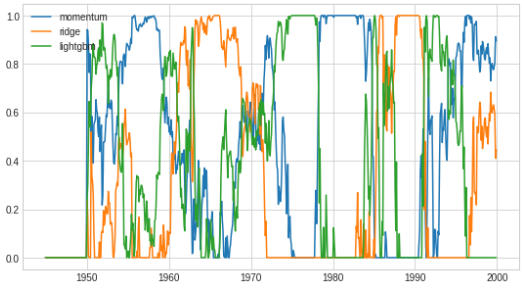
[19]:
```
from skfin.mv_estimators import Mbj
```

[20]:
```
m = StackingBacktester(
    estimators=estimators,
    final_estimator=Mbj(positive=True),
    ret=ret,
    window=60,
    min_periods=60,
)
m.train(features, target)
```

```
pnls["ensemble_mbj_positive"] = m.pnls_["ensemble"]
```

[21]:
```
fig, ax = plt.subplots(1, 2, figsize=(20, 5))
line(m.pnls_["1950-02-01":], cumsum=True, ax=ax[0], loc="best")
line(m.coef_, ax=ax[1], loc="best")
```

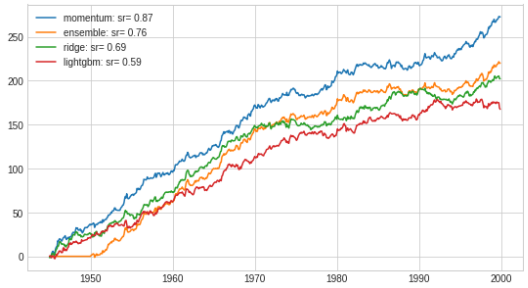


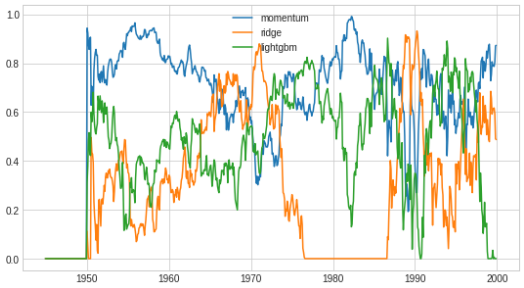


Over longer periods with positive constraints, the performance is closer to the industry momentum.

[22]:
```
m = StackingBacktester(
    estimators=estimators,
    final_estimator=Mbj(positive=True),
    ret=ret,
    window=180,
    min_periods=60,
)

m.train(features, target)
pnls["ensemble_mbj_positive_long_window"] = m.pnls_["ensemble"]
```

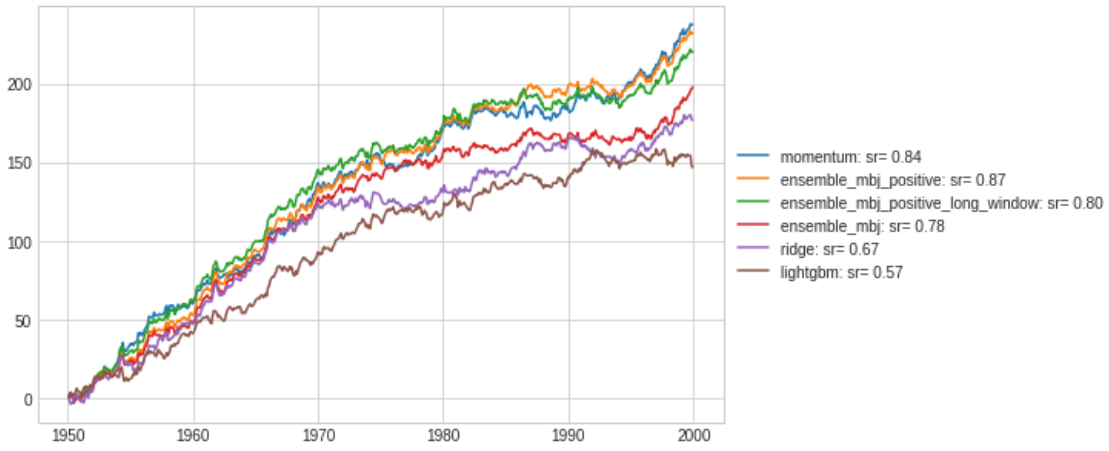
[23]:
```
fig, ax = plt.subplots(1, 2, figsize=(20, 5))
line(m.pnls_, cumsum=True, ax=ax[0], loc="best")
line(m.coef_, ax=ax[1], loc="best")
```





Putting the different ensembles, we compare the pnls in the graph below

[24]:
```
line(pnls["1950-02-01":], cumsum=True)
```


200
150
100
50
0
1950
1960
1970
1980
1990
2000
momentum: sr= 0.84
ensemble_mbj_positive: sr= 0.87
ensemble_mbj_positive_long_window: sr= 0.80
ensemble_mbj: sr= 0.78
ridge: sr= 0.67
lightgbm: sr= 0.57

Chapter 12

Transaction cost

12.1 Order book and trades

In this section, we discuss the costs associated to trading following Section 3.7 in *Engineering Investment Process* by Ielpo, Merhy and Simon (2017). A rule of thumbs is

> "It takes one day of volume to trade one day of volatility."

In other words: *the bigger trade, the higher the cost*. There are three main categories of trade execution:

1. over-the-counter (OTC): the price of the transaction is directly negotiated with another market participant;
2. delegated-execution: the trade is delegated to a broker with a guarantee of execution price (e.g. price at close or volume-weighted average price as VWAP).
3. direct trade and execution on markets.

There is a tradeoff between trading immediately at a known (and potentially high) price and trading later, but at an unknown time/date and for an unknown price. In this context, there are two different types of costs:

- direct/explicit costs: commissions, brokerage fees, settlement/clearing fees, exchange fees, taxes. This also includes shorting costs with easy-to-borrow / hard-to-borrow rates.
- indirect/implicit/impact costs: loosely, these costs capture the idea that the bigger the trade, the higher the cost. But typically, each investor faces impact costs depending on execution choice, execution pattern, latency, procedure speed and horizon. These impact costs are estimated based on (potentially private) past trades.

At any given point in time, some market participants can make public their willingness to trade a quantity at a given price in an *order book*. More precisely, an order book refers to an electronic list of buy and sell orders for a specific security or financial instrument organized by price level.

The bid-ask spread is the difference between the highest price that a buyer is willing to pay for an asset and the lowest price that a seller is willing to accept. More precisely, there are two types of orders:

1. market order: this is an instruction by an investor to a broker to buy or sell an assets at the best available price in the order book.

2. limit order: this is an instruction to a broker to buy or sell only a certain quantity at a certain price.

12.2 Positions and turnover

In previous sections, the positions h_t have been rescaled so that

$$\sqrt{h_t^T V_t h_t} = 1$$

and

$$\sum_t h_{n,t} = 0.$$

More generally the positions are denominated in dollars as:

$$pos_t = \theta \times h_t,$$

where the scaler is a risk target in dollars: $\theta = \sqrt{pos_t^T V_t pos_t}$. Given this risk target, the (two-sided) gross leverage

$$\text{Leverage} = \sum_n |pos_{n,t}|$$

From (rescaled) positions in asset n are denoted $h_{n,t}$, the trades *quantities* are:

$$q_{n,t} = pos_{n,t} - (1 + r_t) pos_{n,t-1}.$$

- (One-sided) turnover

$$\text{TrueTurnover}_t = \sum_n |pos_{n,t} - (1 + r_t) pos_{n,t-1}|/2$$

  where $h_{n,t} - (1 + r_{n,t})h_{n,t-1}$ is the trade. In most of what we do, approximating the small returns away is good enough:

$$\text{Turnover}_t = \sum_n |pos_{n,t} - pos_{n,t-1}|/2$$

- Holding period

$$\text{HP} = \frac{\frac{1}{T}\sum_{t=1}^T \text{Leverage}_t}{\frac{1}{T}\sum_{t=1}^T \text{Turnover}_t}$$

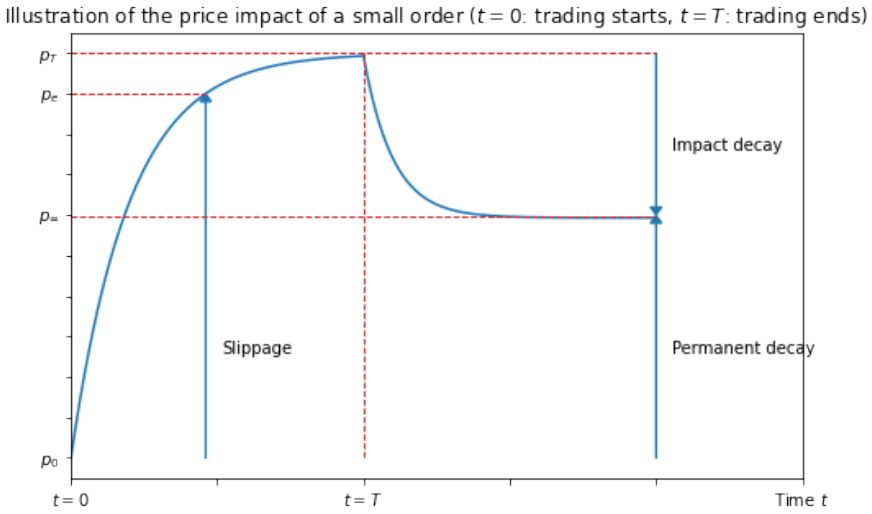
12.3 Impact costs

The figure below is from Ielpo et al. (2017, p. 239):

- p_0: decision price
- p_T: final price
- p_e: average executing price

- p_∞: after the trade, the price reverts to a limit value
- slippage $= \frac{p_e}{p_0} - 1$
- permanent impact $= \frac{p_\infty}{p_0} - 1$

Given that trades are motivated by a view of the market (ie. some alpha opportunity), it is difficult to distinguish alpha et pure impact.



In practice, we define the impact cost as a function I:

$$Q \mapsto I(Q),$$

where Q is the traded quantities. The price impact in dollars of trading the quantity Q is

$$\text{ImpactCost} = Q \times I(Q).$$

The fraction of traded volume is $\delta = \frac{Q_{n,t}}{\text{Liquidity}_{n,t}}$ can be interpreted as time – e.g. how many days (or fraction of a day) to trade $Q_{n,t}$ given the asset liquidity $\text{Liquidity}_{n,t}$. Over that period, the asset volatility scaled in $\sigma_{n,t}\sqrt{\delta}$, so that a simple impact model is:

$$I(Q) = \sigma\sqrt{\frac{Q}{\text{Liquidity}}}.$$

More generally, two parameters $\langle \gamma, \beta \rangle$ are fitted with the functional form:

$$I(Q) = \gamma\sigma\left(\frac{Q}{\text{Liquidity}}\right)^{\beta}.$$

A rule of thumb is as follows:

- for small horizons, $\beta \approx 0$, $I(Q)$ is constant (e.g. bid-ask spread) and the total cost is linear;

- for long horizons, $\beta \approx 1$ and the total impact cost $Q \times I(Q)$ is quadratic.

In analytically more complex (and potentially more realistic) setups, β is taken to be $1/2$ as motivated above.

In what follows, we focus on quadratic impact cost with $\beta = 1$.

12.4 Mean-variance optimisation with quadratic costs

With quadratic cost, the mean-variance optimisation can be solved in closed form. More precisely, the mean-variance utility at time t with transaction cost is

$$h_t^T \alpha - \frac{h_t V h_t}{2\lambda} - \frac{(h_t - h_{t-1})^T \Lambda (h_t - h_{t-1})}{2}$$

Lemma. Given the mean-variance utility with quadratic cost, the optimal portfolio is

$$h_t = \left[\frac{V}{\lambda} + \Lambda\right]^{-1} (\alpha + \Lambda h_{t-1}).$$

Proof. The mean-variance objective is rewritten as

$$h_t^T (\alpha + \Lambda h_{t-1}) - \frac{h_t (V + \lambda\Lambda) h_t}{2\lambda} - \frac{h_{t-1}^T \Lambda h_{t-1}}{2}$$

The optimal portfolio with transaction cost is a weighted average of two holdings:

$$h_t = (I + \lambda V^{-1}\Lambda)^{-1} \lambda V^{-1} \alpha + [I - (I + \lambda V^{-1}\Lambda)^{-1}] h_{t-1}.$$

Interpretation:

- the friction-less mean-variance portfolio $\lambda V^{-1}\alpha$ is the target portfolio
- the optimal portfolio h_t is a weighted-average of this target portfolio and the past holdings h_{t-1}:
- ideally the investor would like to hold the target portfolio but because of transaction cost, it only trades partially towards this target.
- when the transaction costs are lower and the risk-tolerance λ is higher, the investor loads more on the mean-variance target $\lambda V^{-1}\alpha$
- the transaction imply so form of (non-linear) exponential averaging.

12.5 Cost for the Industry Momentum backtest

[3]:
```
from skfin.backtesting import Backtester
from skfin.datasets import load_kf_returns
from skfin.mv_estimators import MeanVariance
from skfin.plot import bar, line
```

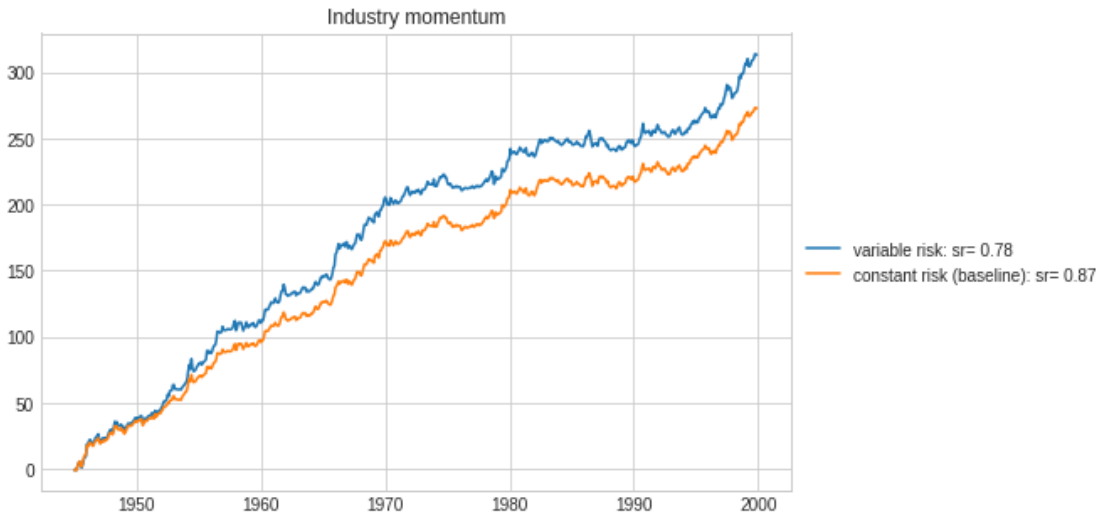
[4]:
```
returns_data = load_kf_returns(cache_dir="data")
ret = returns_data["Monthly"]["Average_Value_Weighted_Returns"][:"1999"]
```

```
INFO:skfin.datasets:logging from cache directory: data/12_Industry_Portfolios
```

[5]:
```
transform_X = lambda x: x.rolling(12).mean()
transform_y = lambda x: x.shift(-1)
features = transform_X(ret)
target = transform_y(ret)
```

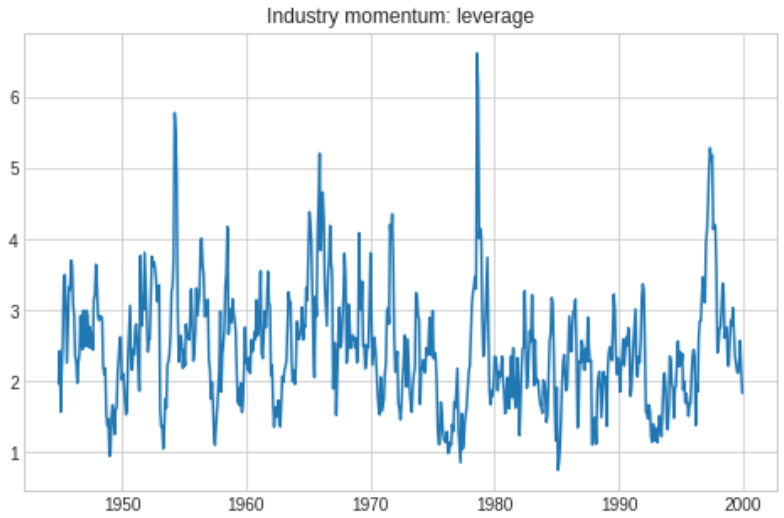
Given that the data is monthly, we re-estimate the model every month and the maximum train window is 12 months.

[6]:
```
estimators_ = {
    "constant risk (baseline)": MeanVariance(constant_risk=True),
    "variable risk": MeanVariance(constant_risk=False),
}
line(
    {k: Backtester(v).train(features, target, ret) for k, v in estimators_.items()},
    cumsum=True,
    title="Industry momentum",
)
```



12.5.1 Leverage and turnover

[7]:
```
m0 = (
    Backtester(MeanVariance(constant_risk=False))
    .compute_holdings(features, target)
    .compute_pnl(ret)
)
line(
    m0.h_.abs().sum(axis=1).rename("leverage"),
    title="Industry momentum: leverage",
    legend=False,
)
```



[8]:
```
def average_holding_period(h):
    h = h.squeeze()
    if isinstance(h, pd.Series):
        return h.abs().mean() / h.diff().abs().div(2).mean()
    else:
        return h.abs().sum(axis=1).mean() / h.diff().abs().sum(axis=1).div(2).mean()
```

[9]:
```
hp = average_holding_period(m0.h_)

print(f"The average holding period is {hp:.2f} months")
```

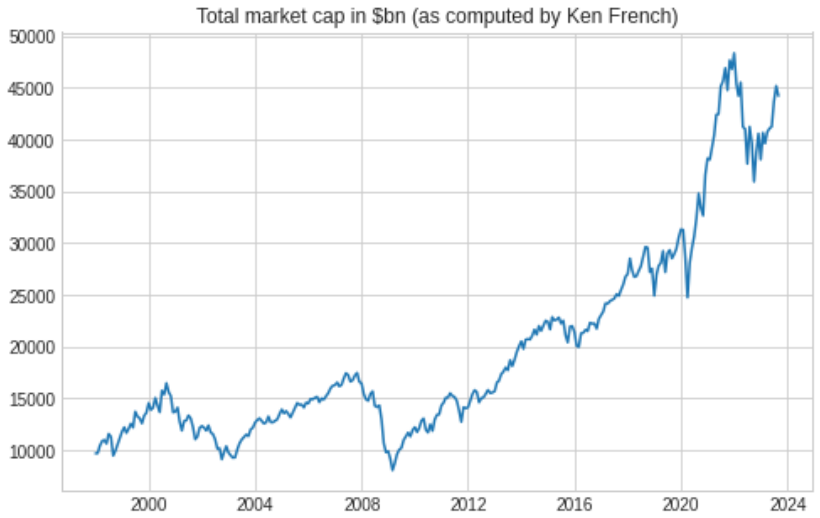
```
The average holding period is 5.55 months
```

12.5.2 Liquidity and market cap

Given that we do not have data on sector liquidity, we use the market capitalisation as a proxy for liquidity.

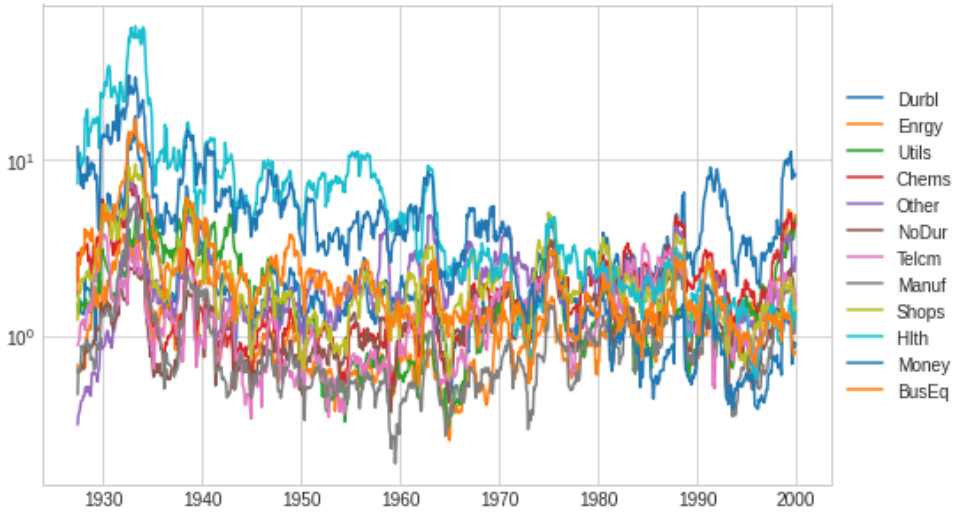
[10]:
```
mcap = returns_data["Monthly"]["Average_Firm_Size"].mul(
    returns_data["Monthly"]["Number_of_Firms_in_Portfolios"]
)
```

[11]:
```
line(
    mcap.loc["1998":].sum(axis=1).div(1e3),
    title="Total market cap in $bn (as computed by Ken French)",
    legend=False,
)
```



[12]:
```
gamma = 1 / 20
mcap_norm = mcap.div(mcap.sum(axis=1), axis=0)
vol_liquidity_factor = 0.5 * gamma * ret.rolling(12).std().div(mcap_norm.loc[ret.index])
```

[13]:
```
line(vol_liquidity_factor, yscale="log")
```



12.5.3 Mark-to-market and backtesting

In this case, Λ is the diagonal of the vector of return standard deviations over market capitalisation (as a proxy for liquity – e.g. traded volume).

[14]:
```
%%writefile ../skfin/backtesting_with_cost.py
import numpy as np
import pandas as pd
from sklearn.base import BaseEstimator, clone
from sklearn.model_selection import TimeSeriesSplit
```

```
def compute_pnl_components(h, ret, vol_liquidity_factor=None):
    ret = ret[h.index[0] : h.index[-1]]
    vol_liquidity_factor = vol_liquidity_factor.loc[h.index[0] : h.index[-1]]

    pnl = h.shift(1).mul(ret).sum(axis=1)
    if vol_liquidity_factor is not None:
        impact_cost = h.diff().pow(2).mul(vol_liquidity_factor).sum(axis=1)
        return {
            "gross": pnl,
            "net = gross - impact cost": pnl.sub(impact_cost),
            "impact cost": -1 * impact_cost,
        }
    else:
        return pnl


def compute_batch_holdings_(
    pred, V, A=None, past_h=None, vol_liquidity_factor=None, lambda_=None
):
    """
    compute markowitz holdings with return prediction "mu" and covariance matrix "V"

    mu: numpy array (shape N * K)
    V: numpy array (N * N)

    """
    if (lambda_ is None) & (vol_liquidity_factor is not None):
        lambda_ = 1
    N, _ = V.shape
    if isinstance(pred, pd.Series) | isinstance(pred, pd.DataFrame):
        pred = pred.values
    if pred.shape == (N,):
        pred = pred[:, None]
    elif pred.shape[1] == N:
        pred = pred.T

    if vol_liquidity_factor is not None:
        invV = np.linalg.inv(V / lambda_ + 2 * np.diag(vol_liquidity_factor))
    else:
        invV = np.linalg.inv(V)
    if A is None:
        M = invV
    else:
        U = invV.dot(A)
        if A.ndim == 1:
            M = invV - np.outer(U, U.T) / U.dot(A)
        else:
            M = invV - U.dot(np.linalg.inv(U.T.dot(A)).dot(U.T))
    if (vol_liquidity_factor is not None) & (past_h is not None):
        h = M.dot(pred + 2 * np.diag(vol_liquidity_factor).dot(past_h.T))
    else:
        h = M.dot(pred)
    return h.T


class MeanVarianceWithCost(BaseEstimator):
    def __init__(self, transform_V=None, A=None):
        if transform_V is None:
            self.transform_V = lambda x: np.cov(x.T)
```

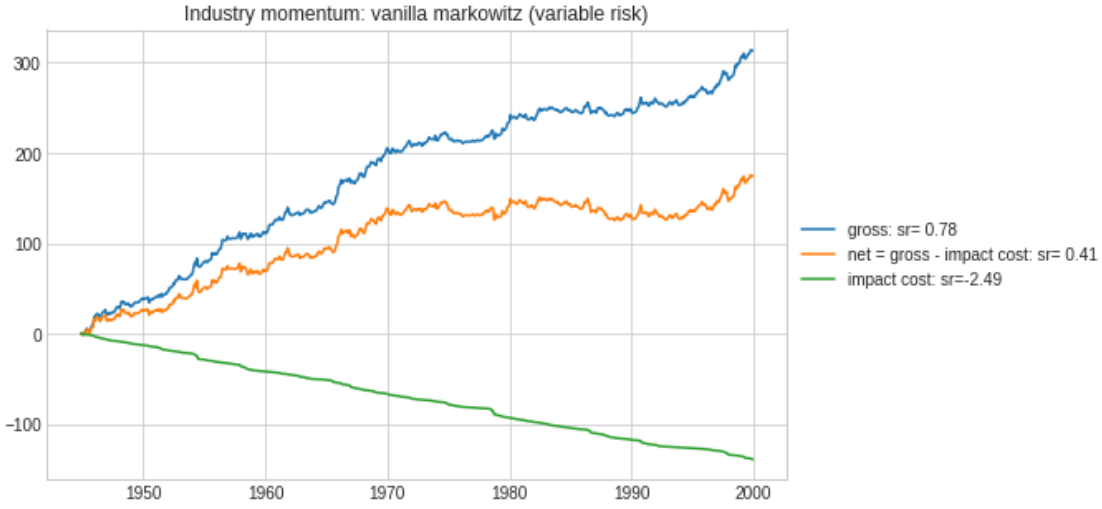
```
        else:
            self.transform_V = transform_V
        self.A = A

    def fit(self, X, y=None):
        self.V_ = self.transform_V(y)

    def predict(self, X, past_h=None, vol_liquidity_factor=None):
        if self.A is None:
            T, N = X.shape
            A = np.ones(N)
        else:
            A = self.A
        h = compute_batch_holdings_(
            X, self.V_, A, past_h=past_h, vol_liquidity_factor=vol_liquidity_factor
        )
        return h


class BacktesterWithCost:
    def __init__(
        self,
        estimator,
        ret,
        vol_liquidity_factor=None,
        max_train_size=36,
        test_size=1,
        start_date="1945-01-01",
        end_date=None,
        h_init=None,
        return_pnl_component=False,
    ):
        self.start_date = start_date
        self.end_date = end_date
        self.estimator = estimator
        self.ret = ret[: self.end_date]
        self.cv = TimeSeriesSplit(
            max_train_size=max_train_size,
            test_size=test_size,
            n_splits=1 + len(ret.loc[start_date:end_date]) // test_size,
        )
        self.h_init = h_init
        self.vol_liquidity_factor = vol_liquidity_factor
        self.return_pnl_component = return_pnl_component

    def train(self, features, target):
        _h = []
        past_h = self.h_init
        for train, test in self.cv.split(self.ret):
            m = clone(self.estimator)
            m.fit(features.iloc[train], target.iloc[train])
            if self.vol_liquidity_factor is None:
                vlf = None
            else:
                vlf = np.squeeze(self.vol_liquidity_factor.values[test])
            current_h = m.predict(
                features.iloc[test], past_h=past_h, vol_liquidity_factor=vlf
            )
            _h += [current_h]
            past_h = current_h
```

```
        cols = self.ret.columns
        idx = self.ret.index[
            np.concatenate([test for _, test in self.cv.split(self.ret)])
        ]
        h_ = pd.DataFrame(np.concatenate(_h), index=idx, columns=cols)

        self.h_ = h_
        if self.return_pnl_component:
            self.pnl_ = compute_pnl_components(
                self.h_, self.ret, vol_liquidity_factor=self.vol_liquidity_factor
            )
        else:
            self.pnl_ = (
                h_.shift(1).mul(self.ret).sum(axis=1)[self.start_date : self.end_date]
            )
        return self
```

```
Overwriting ../skfin/backtesting_with_cost.py
```

[15]:
```
from skfin.backtesting_with_cost import compute_pnl_components
```

[16]:
```
pnlc = compute_pnl_components(m0.h_, ret, vol_liquidity_factor=vol_liquidity_factor)
line(pnlc, cumsum=True, title="Industry momentum: vanilla markowitz (variable risk)")
```



12.5.4 Backtesting with cost

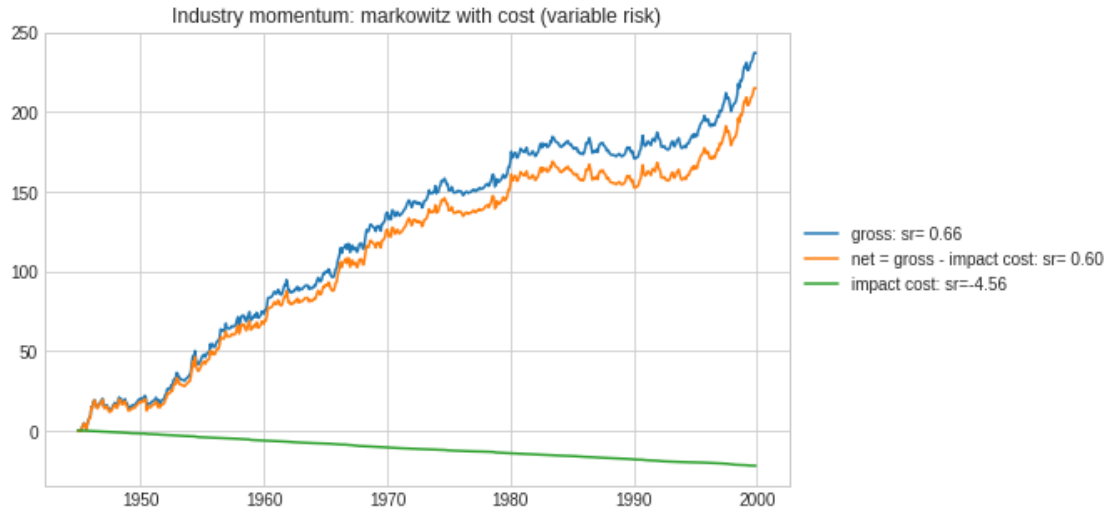
[17]:
```
from skfin.backtesting_with_cost import BacktesterWithCost, MeanVarianceWithCost
```

[18]:
```
m0_ = BacktesterWithCost(MeanVarianceWithCost(), ret, vol_liquidity_factor=None).train(
    features, target
)
m0.pnl_.equals(m0_.pnl_)
```

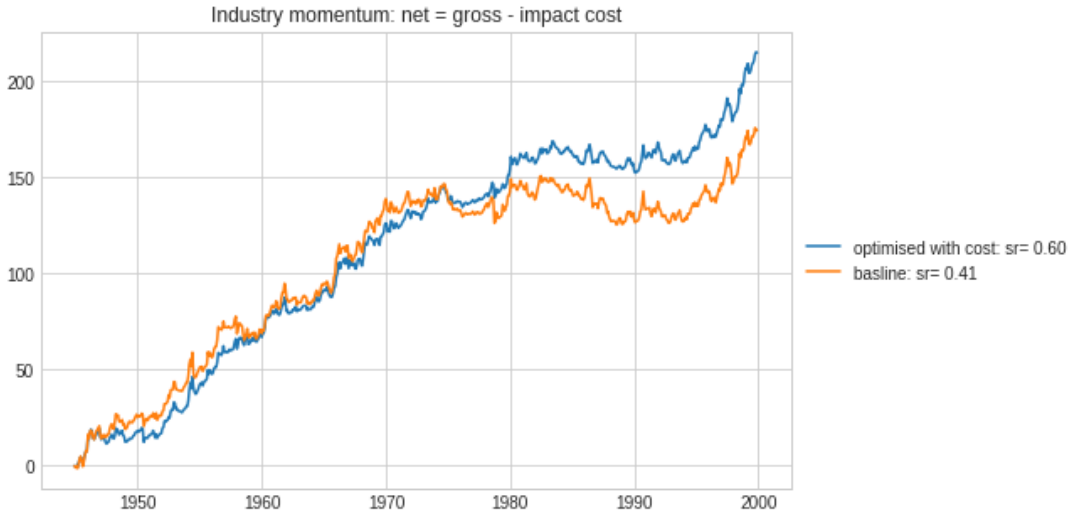
[18]:
```
False
```

[19]:
```
m = BacktesterWithCost(
    MeanVarianceWithCost(), ret, vol_liquidity_factor, return_pnl_component=True
).train(features, target)
```

```
line(
    m.pnl_, cumsum=True, title="Industry momentum: markowitz with cost (variable risk)"
)
```



[20]:
```
c = "net = gross - impact cost"
line(
    {"basline": pnlc[c], "optimised with cost": m.pnl_[c]},
    cumsum=True,
    title=f"Industry momentum: {c}",
)
```



Comparing the average holding periods, we see a sharp increase.

[21]:
```
print(
    f"Average holding period (in month):\n - baseline = {average_holding_period(m0.h_):.1f},\n -␣
 ↪optimised with quadratic cost = {average_holding_period(m.h_):.1f}."
)
```

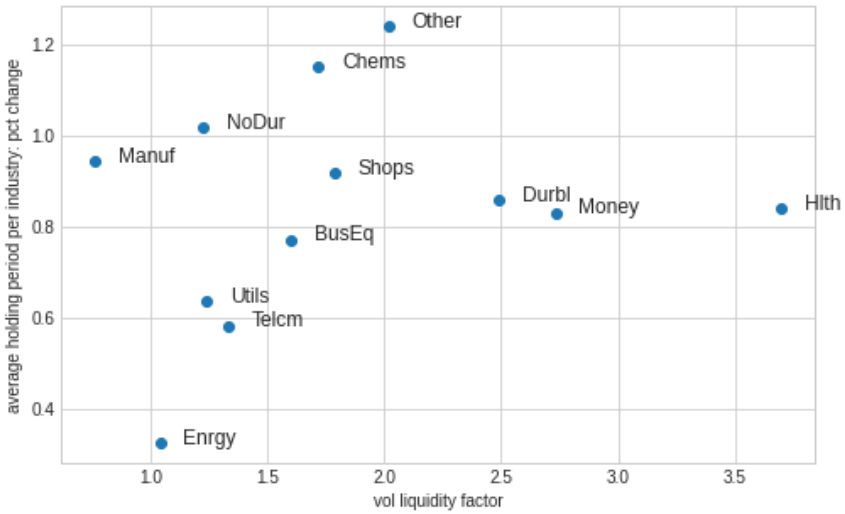
```
Average holding period (in month):
 - baseline = 5.5,
 - optimised with quadratic cost = 10.3.
```

The scatterplot below shows at the industry level:

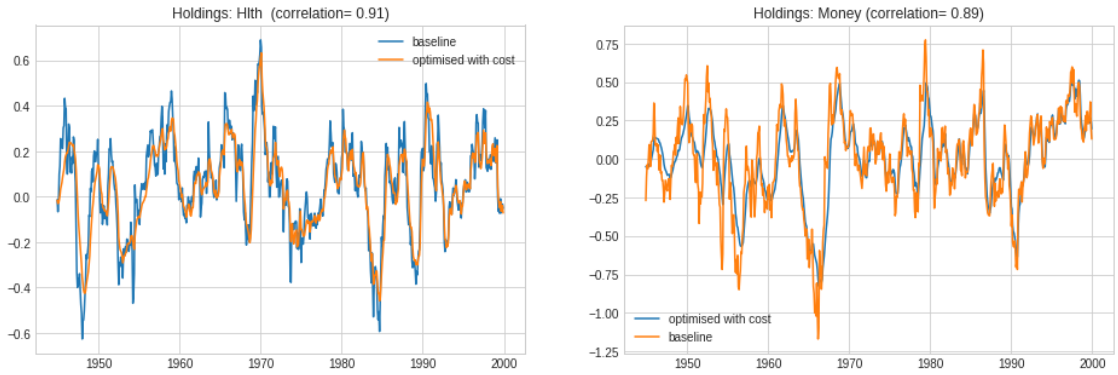
- the average liquidity factor
- the average holding period

Intuitive, the more expensive it is to trade a sector, the longer the holding period should be.

[22]:
```
fig, ax = plt.subplots(1, 1, figsize=(8, 5))

avp = (
    pd.Series({c: average_holding_period(m.h_[c]) for c in m0.h_.columns})
    .div(pd.Series({c: average_holding_period(m0.h_[c]) for c in m0.h_.columns}))
    .sub(1)
)

df = pd.concat(
    {
        "vol liquidity factor": vol_liquidity_factor.loc[m0.h_.index[0] :].mean(),
        "average holding period per industry: pct change": avp,
    },
    axis=1,
)

cols = list(df.columns)
idx = list(df.index)

ax.scatter(df.iloc[:, 0].values, df.iloc[:, 1].values)
ax.set_xlabel(cols[0])
ax.set_ylabel(cols[1])
xshift, yshift, rotation = 0.1, 0, 0
for i, txt in enumerate(idx):
    ax.text(
        df.iloc[i, 0] + xshift,
        df.iloc[i, 1] + yshift,
        txt,
        fontsize=12,
        rotation=rotation,
    )
```



[23]:
```
cols = ["Hlth ", "Money"]
fig, ax = plt.subplots(1, 2, figsize=(16, 5))
for i, c in enumerate(cols):
    df = pd.concat({"baseline": m0.h_[c], "optimised with cost": m.h_[c]}, axis=1)
    line(
        df,
        title=f"Holdings: {c} (correlation={df.corr().iloc[0, 1]: .2f})",
        ax=ax[i],
        loc="best",
    )
```

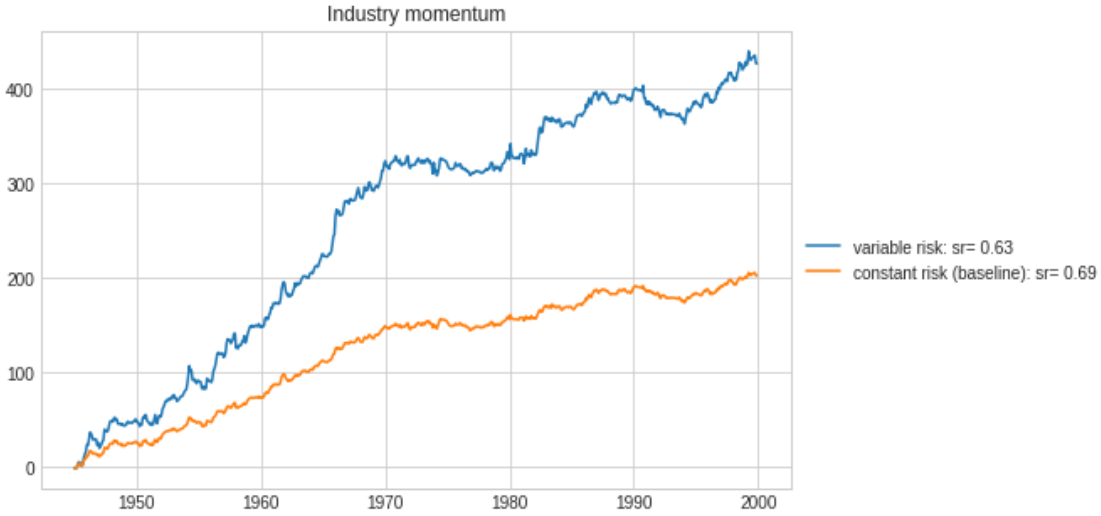


For healthcare, the volatility/liquidity factor seems high on average, but decreases sharply over the period – which explains that the average holding period increases less than expected.

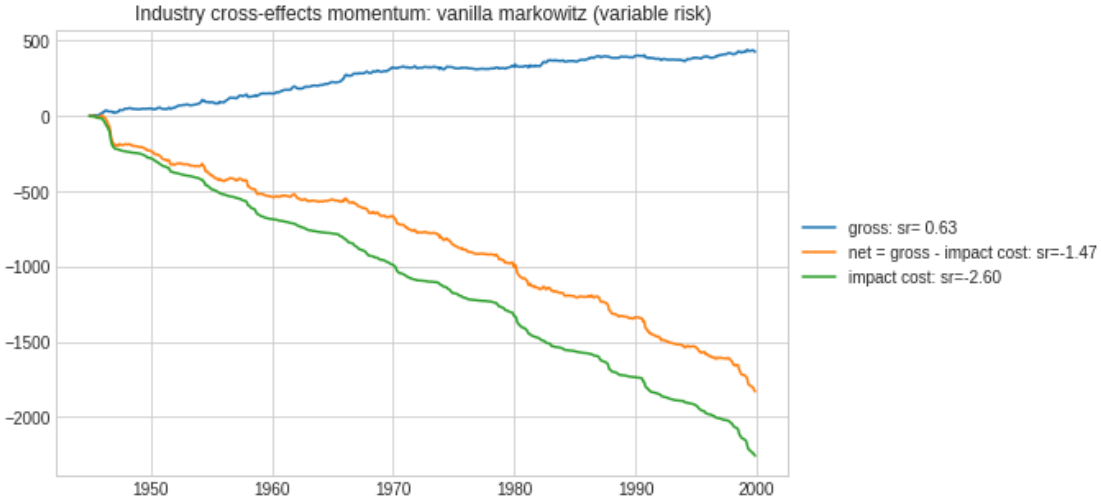
12.5.5 Ridge backtest with cost

[24]:
```
from skfin.estimators import Ridge
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
```

[25]:
```
estimators_ = {
    "constant risk (baseline)": make_pipeline(
        StandardScaler(with_mean=False), Ridge(), MeanVariance(constant_risk=True)
    ),
    "variable risk": make_pipeline(
        StandardScaler(with_mean=False), Ridge(), MeanVariance(constant_risk=False)
    ),
}
line(
    {k: Backtester(v).train(features, target, ret) for k, v in estimators_.items()},
    cumsum=True,
    title="Industry momentum",
)
```

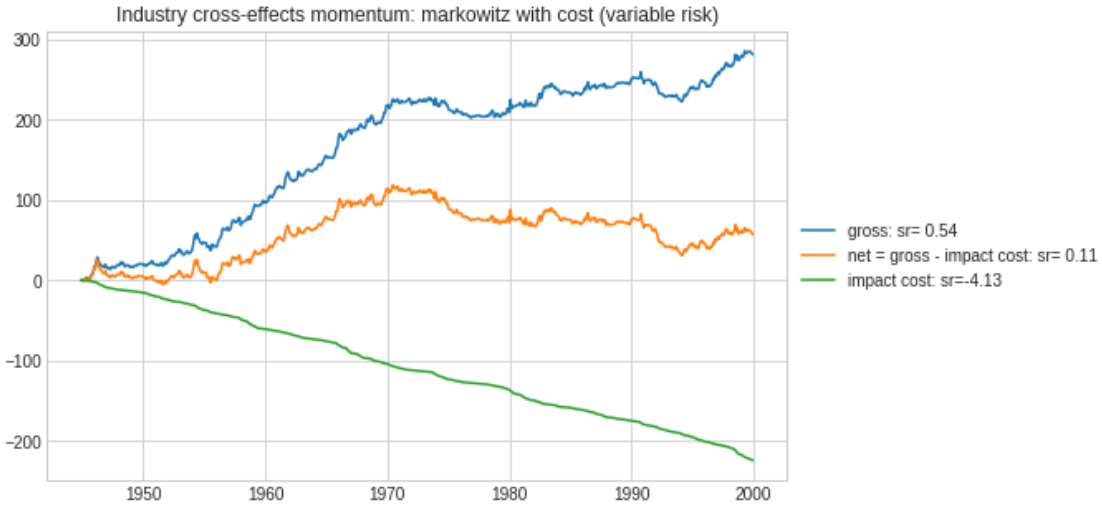


[26]:
```
estimator_ = make_pipeline(
    StandardScaler(with_mean=False), Ridge(), MeanVarianceWithCost()
)
m0_ = BacktesterWithCost(estimator_, ret, vol_liquidity_factor=None).train(
    features, target
)

pnlc = compute_pnl_components(m0_.h_, ret, vol_liquidity_factor=vol_liquidity_factor)
line(
    pnlc,
    cumsum=True,
    title="Industry cross-effects momentum: vanilla markowitz (variable risk)",
)
```



[27]:
```
m_ = BacktesterWithCost(
    estimator_,
    ret,
    vol_liquidity_factor=vol_liquidity_factor,
```

```
    return_pnl_component=True,
).train(features, target)
line(
    m_.pnl_,
    cumsum=True,
    title="Industry cross-effects momentum: markowitz with cost (variable risk)",
)
```



A key difference between the baseline industry momentum strategy and the strategies that lean cross-effects (e.g. learning with a Ridge estimator) is these strategies are faster. In this case, taking cost into account to slow down the strategy is even more important.

[28]:
```
print(
    f"Average holding period (in month):\n - baseline Ridge = {average_holding_period(m0_.h_):.
 ↪1f},\n - optimised with quadratic cost = {average_holding_period(m_.h_):.1f}."
)
```

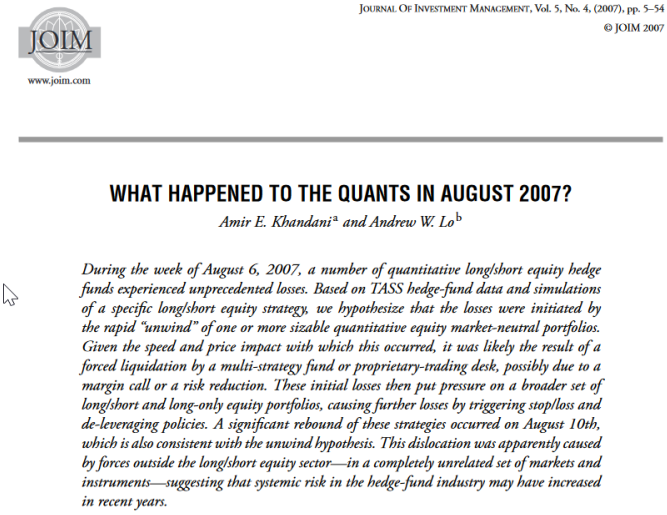
```
Average holding period (in month):
 - baseline Ridge = 2.5,
 - optimised with quadratic cost = 4.5.
```

Chapter 13

Mean-Reversion

In this section, we review the empirical evidence on `liquidity provision` and describe a `mean-reversion` strategy on stock returns.

13.1 Liquidity and autocorrelation of stock returns



JOURNAL OF INVESTMENT MANAGEMENT, Vol. 5, No. 4, (2007), pp. 5–54

JOIM
www.joim.com

WHAT HAPPENED TO THE QUANTS IN AUGUST 2007?

Amir E. Khandani[a] *and Andrew W. Lo*[b]

During the week of August 6, 2007, a number of quantitative long/short equity hedge funds experienced unprecedented losses. Based on TASS hedge-fund data and simulations of a specific long/short equity strategy, we hypothesize that the losses were initiated by the rapid "unwind" of one or more sizable quantitative equity market-neutral portfolios. Given the speed and price impact with which this occurred, it was likely the result of a forced liquidation by a multi-strategy fund or proprietary-trading desk, possibly due to a margin call or a risk reduction. These initial losses then put pressure on a broader set of long/short and long-only equity portfolios, causing further losses by triggering stop/loss and de-leveraging policies. A significant rebound of these strategies occurred on August 10th, which is also consistent with the unwind hypothesis. This dislocation was apparently caused by forces outside the long/short equity sector—in a completely unrelated set of markets and instruments—suggesting that systemic risk in the hedge-fund industry may have increased in recent years.

Hedge funds as provider of liquidity is trend perfected by Long-Term Capital Management (until its fatal crash in 1998...):

> by taking long positions in stocks that have declined and short positions in stocks that have advanced over the previous day, the strategy actively provides liquidity to the marketplace;

This type of strategies generally requires a very high leverage, so anticipating market dislocations is key. Khandani-Lo (2007) argue that hedge funds have overtook traditional market makers as liquidity providers, so much so that when hedge funds retract, the whole market collapses...

Mean-reversion (contrarian) strategy returns: holdings positive in past losers, negative in past winners:

$$h_{n,t} = -1 \times \frac{1}{N} \left(r_{n,t} - r_{\text{Market},t} \right),$$

where $r_{\text{Market},t} = \frac{1}{N} \sum_n r_{n,t}$ is the cross-sectional average. By construction, such a strategy would be cash-neutral:

$$\sum_n h_{n,t} = 0.$$

Khandani and Lo (2007):

- a market-making liquidity provision exhibits extreme losses during the week of August 6, 2007 – suggesting market-wide deleveraging.

  We wish to acknowledge at the outset that the hypotheses advanced in this paper are speculative, tentative, and based solely on indirect evidence."

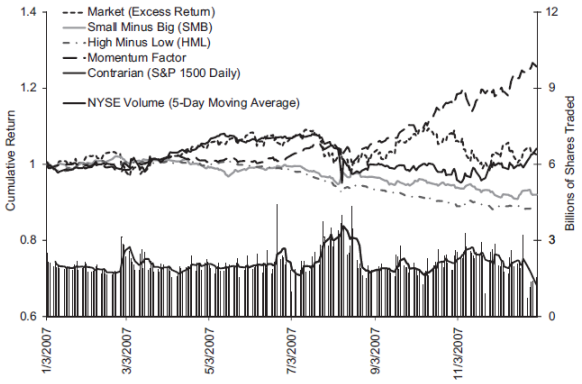


Fig. 1. Factor performance and volume in 2007. The cumulative daily returns for the Market, Small Minus Big (SMB), High Minus Low (HML), Momentum factors as well as the contrarian strategy of Lehmann (1990) and Lo and MacKinlay (1990) for January 3, 2007, to December 31, 2007. Data for Market, SMB, HML, and Momentum factors were obtained from Kenneth French's website (see footnote 8 for details). The contrarian strategy was implemented as in Khandani and Lo (2007) using daily return for stocks listed in the S&P 1500 on January 3, 2007. Volume represents the Daily NYSE Group Volume in NYSE-listed securities obtained from the NYSE Euronext website.

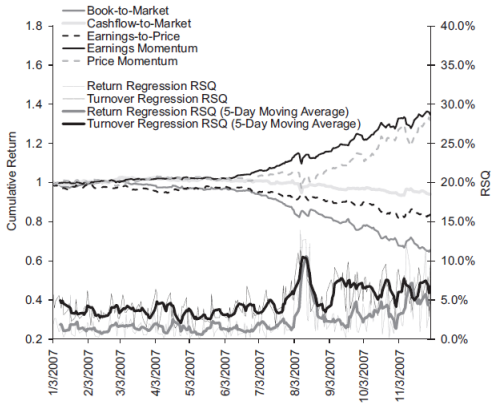


Fig. 2. Cumulative performance and RSQ of cross-sectional regression for quant factors. Cumulative performance of five long/short equity market-neutral portfolios constructed from commonly used equity-valuation factors, from January 3, 2007, to December 31, 2007. Also plotted are daily R^2's and their 5-day moving averages of the following cross-sectional regressions of returns and turnover: $R_{i,t} = \alpha_t + \sum_{f=1}^{5} \beta_{f,t} D_{i,f} + \varepsilon_{i,t}$, and $\text{TO}_{i,t} = \gamma_t + \sum_{f=1}^{5} \delta_{f,t} |D_{i,f} - 5.5| + \eta_{i,f}$, where $R_{i,t}$ is the return for security i on day t, $D_{i,f}$ is the decile ranking of security i according to factor f, and $\text{TO}_{i,t}$, the turnover for security i on day t, is defined as the ratio of the number of shares traded to the shares outstanding.

Table 1
Performance of contrarian strategy at intraday and daily frequency.

Panel B: Daily Contrarian Strategy

Construction Date	Return (%)				
	1 Day	2 Days	3 Days	4 Days	5 Days
8/1/2007	0.14	−1.03	−2.69	−2.57	−0.34
8/2/2007	−0.76	−1.62	−2.57	−2.63	−2.79
8/3/2007	−0.30	−0.57	0.65	0.29	2.04
8/6/2007	**−1.47**	**−1.79**	**−1.75**	**1.24**	**3.44**
8/7/2007	**−2.88**	**−4.49**	**1.38**	**4.00**	**4.52**
8/8/2007	**−3.99**	**3.79**	**7.81**	**8.31**	**8.20**
8/9/2007	**6.85**	**10.12**	**9.83**	**9.47**	**8.96**
8/10/2007	**−1.46**	**−1.71**	**−1.48**	**−1.84**	**−1.49**
8/13/2007	0.19	0.82	3.79	4.61	3.77
8/14/2007	−0.95	−0.83	0.22	0.34	0.56
8/15/2007	−1.34	−0.58	0.31	0.76	1.69
January to July 2007 Sigma	0.36	0.49	0.59	0.66	0.69

Quantitative Equity Funds Hit Hard In August 2007

- specifically, August 7th–9th, and massive reversal on August 10th
- some of the most consistently profitable funds lost too
- seemed to affect only quants
- no real market news

Kandhani-Lo "Unwind hypothesis:"

- many quant funds use similar factor models
- if one fund has to unwind a large position, that hurts other funds
- this causes these other funds to sell as well…

Sequence of event:

- initial losses "initiated by the rapid unwind of one or more sizeable quantitative equity market-neutral portfolios (likely the result of a force liquidation by a multi-strategy fund or proprietary desk, possibly due to a margin call or a risk reduction)"
- "these initial losses put pressure on a broader set of long/short and long-only equity portfolios, causing further losses by triggering stop/loss and deleveraging policies"
- "on Friday, August 10th, sharp reversals in all five strategies erased nearly all of the losses of the previous four days, returning portfolio values back to their levels on the morning of August 6th. Of course, this assumes that portfolio leverage did not change during this tumultuous week, which is an unlikely assumption given the enormous losses during the first few days."

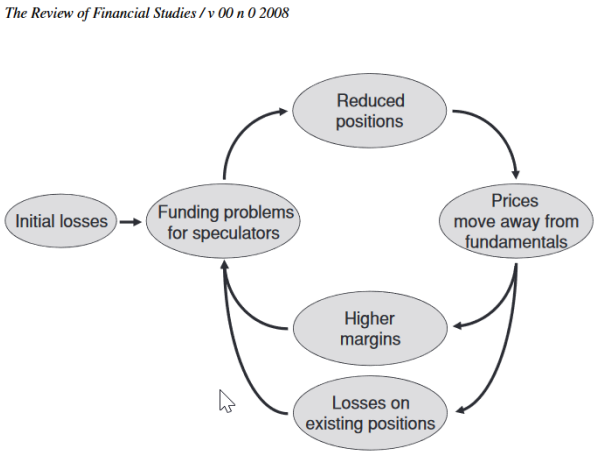


Figure 2
Liquidity spirals
The figure shows the loss spiral and the margin/haircut spiral.

13.2 Sample

[8]:
```
from skfin.datasets import load_sklearn_stock_returns

ret = load_sklearn_stock_returns(cache_dir="data")
```

```
INFO:skfin.datasets:logging from cache directory: data/sklearn_returns.parquet
```

This stock return is found in the `example` folder on `scikit-learn` github repository and contains returns from 2003 to 2008. Interestingly, it includes not only US companies, but also European (e.g. Total, SAP, Novartis, etc) and Japanese (Honda, Sony, Toyota) ones, but also companies that are no longer publically traded (e.g. Dell) or have been split (DuPont).

In particular, removing firms that are no longer publicly traded would create a survival bias in the sample. More generally, it is important to make sure that the firms that would have been available for trading are dynamically included in the backtest sample and removed when there are no longer traded (or not liquid enough to be traded given transaction costs).

13.3 Mean-reversion strategy

We load the same functions as the ones we used for the industry backtest.

[9]:
```
from skfin.backtesting import Backtester
from skfin.mv_estimators import MeanVariance
```

Rather than setting the positions as $-1\times$ returns (as Khandani and Lo do), we feed the $-1\times$ EMA(returns) into the `Backtester` class to get mean-variance positions.

[10]:
```
def xs_score(x, clip=5):
    return (
        x.sub(x.mean(axis=1), axis=0)
        .div(x.std(axis=1), axis=0)
        .clip(lower=-clip, upper=clip)
    )
```

[11]:
```
transform_X = lambda x: -1 * x.pipe(xs_score).ewm(halflife=5).mean().fillna(0)
transform_y = lambda x: x.shift(-1).fillna(0)
features = transform_X(ret)
target = transform_y(ret)
```

In contrast to the industry backtests (which were at the Monthly frequency), these mean-reversion backtests are at the (business) day frequency.

[12]:
```
ret.equals(ret.resample("B").mean())
```

[12]:
```
True
```

The backtest is setup so that the risk-model is computed over 3-months (=63 business days).

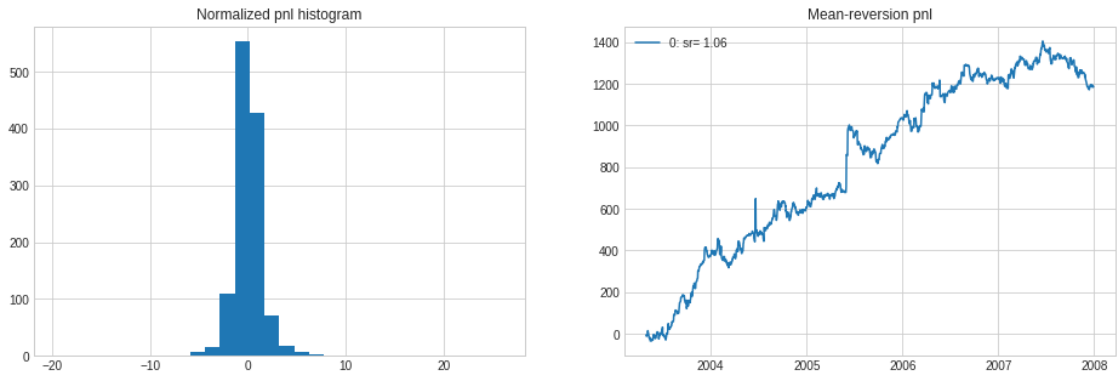
[13]:
```
risk_model_window = 63
```

[14]:
```
m = Backtester(
    MeanVariance(), max_train_size=risk_model_window, start_date="2003-05-01"
)
m.compute_holdings(features, target)
m.compute_pnl(ret)
```

[14]:
```
Backtester(estimator=MeanVariance(transform_V=<function
MeanVariance.__init__.<locals>.<lambda> at 0x7f43402d41f0>), max_train_size=63,
test_size=1, pred_lag=1, start_date='2003-05-01', end_date=None, name=None)
```

[15]:
```
hs = {"no-shrinkage": m.h_}
```

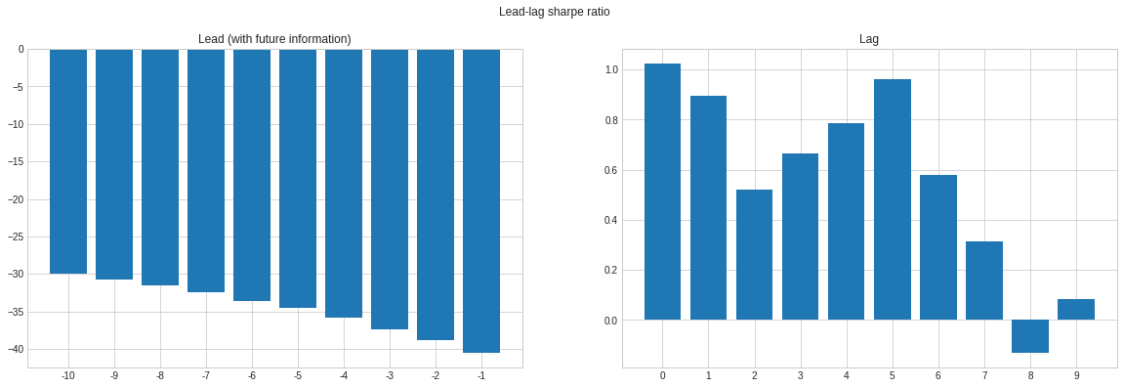
[16]:
```
fig, ax = plt.subplots(1, 2, figsize=(16, 5))
m.pnl_.div(np.sqrt(risk_model_window)).hist(bins=30, ax=ax[0])
ax[0].set_title("Normalized pnl histogram")
line(
    m.pnl_,
    cumsum=True,
    start_date="2003-05-01",
    title="Mean-reversion pnl",
    ax=ax[1],
    loc="best",
)
```



The graph below shows the lead-lag sharpe ratio. The right graph shows the lag: the 0 lag is the tradable lag with a sharpe ratio equal to 1.06 (as in the cumulative pnl graph above). Lagging trading by several business days (up to five) maintain sharpe ratio approximately constant.

The left graph shows the "lead" sharpe ratios – that is, the counterfactual of what the sharpe ratio would have been with future information. Given that the strategy is strong countrarian, this lead sharpe ratio is very negative.

[17]:
```
fig, ax = plt.subplots(1, 2, figsize=(20, 6))
fig.suptitle("Lead-lag sharpe ratio")
bar(
    {
        i: m.h_.shift(1 + i).mul(ret).sum(axis=1).pipe(sharpe_ratio)
        for i in range(-10, 0)
    },
    sort=False,
    title="Lead (with future information)",
    ax=ax[0],
)

bar(
    {
        i: m.h_.shift(1 + i).mul(ret).sum(axis=1).pipe(sharpe_ratio)
        for i in range(0, 10)
    },
    sort=False,
    title="Lag",
    ax=ax[1],
)
```



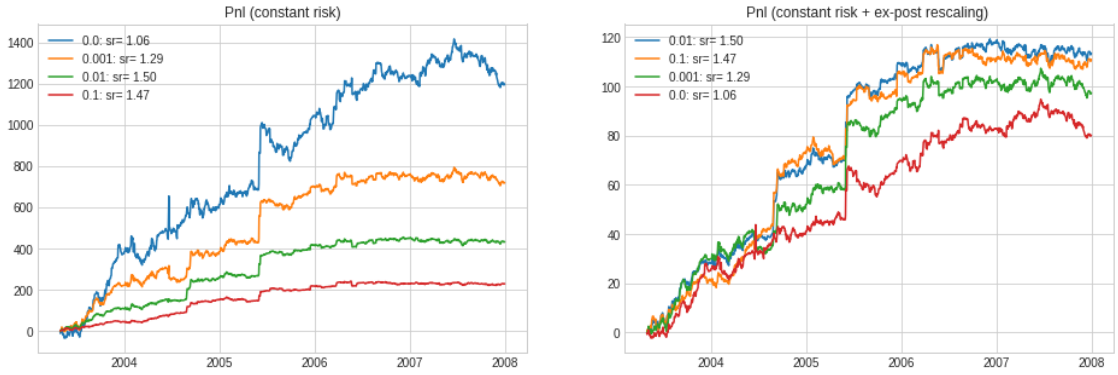
[18]:
```
m_ = Backtester(
    MeanVariance(), max_train_size=risk_model_window, start_date="2003-05-01"
)
pnls_ = {}
for clip in [2, 3, 4, 5]:
    transform_X_ = lambda x: -1 * x.pipe(xs_score, clip=clip).ewm(
        halflife=5
    ).mean().fillna(0)
    features_ = transform_X_(ret)
    pnls_[f"clip={clip}"] = m_.train(features_, target, ret)
line(pnls_, cumsum=True, title="Robustness: return clipping")
```

13.4 Mean-reversion with risk-model shrinkage

Intuitively, risk-model shrinkage is more necessary and useful with more assets (e.g. many stocks versus few industries). We show how the shrinkage of the coveriance matrix helps reducing the over-realisation of risk.

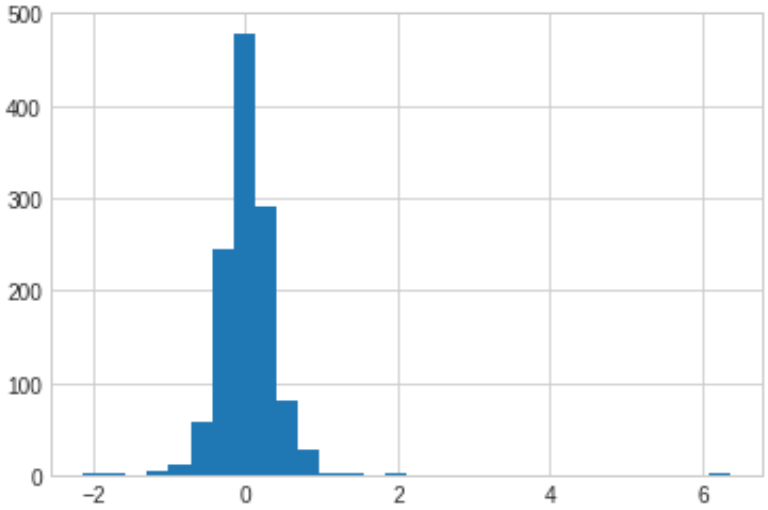
[19]:
```
from sklearn.covariance import ShrunkCovariance

pnls_ = {}
for shrinkage in [0, 0.001, 0.01, 0.1]:
    transform_V_ = lambda x: ShrunkCovariance(shrinkage=shrinkage).fit(x).covariance_
    pnls_[shrinkage] = Backtester(
        MeanVariance(transform_V=transform_V_),
        max_train_size=risk_model_window,
        start_date="2003-05-01",
    ).train(features, target, ret)
```

[20]:
```
fig, ax = plt.subplots(1, 2, figsize=(16, 5))

line(pnls_, cumsum=True, title="Pnl (constant risk)", loc="best", ax=ax[0])
line(
    pd.concat(pnls_, axis=1).pipe(lambda x: x.div(x.std())),
    cumsum=True,
    title="Pnl (constant risk + ex-post rescaling) ",
    loc="best",
    ax=ax[1],
)
```



We see that with no risk-model shrinkage, the over-realisation of risk is very significant. With some shrinkage, there are less pnl outliers – as shown in the histogram below.

[21]:
```
pnls_[0.01].div(np.sqrt(risk_model_window)).hist(bins=30);
```



13.5 Leverage of mean-reversion strategies

In this section, we discuss how we can increase the level of risk per unit of leverage by concentrating the positions.

[22]:
```
shrinkage = 0.01
transform_V_ = lambda x: ShrunkCovariance(shrinkage=shrinkage).fit(x).covariance_
m_ = (
    Backtester(
        MeanVariance(transform_V=transform_V_),
        max_train_size=risk_model_window,
        start_date="2003-05-01",
    )
    .compute_holdings(features, target)
    .compute_pnl(ret)
)
```

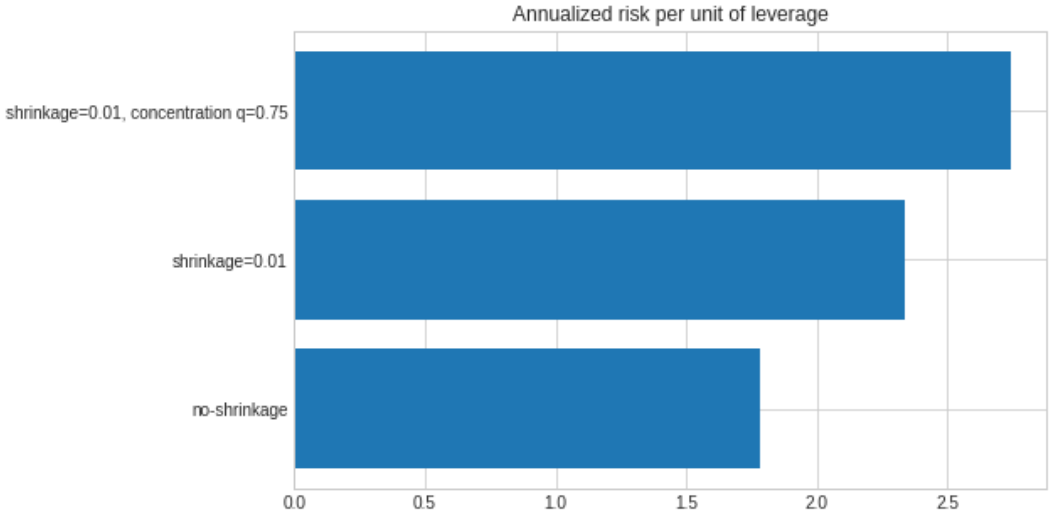
[23]:
```
hs["shrinkage=0.01"] = m_.h_
```

[24]:
```
from skfin.metrics import sharpe_ratio

leverage_scaling = lambda x: x.div(x.abs().sum(axis=1), axis=0)


def risk_per_unit_of_leverage(h, ret, return_sharpe_ratio=False):
    pnl = h.pipe(leverage_scaling).shift(1).mul(ret).sum(axis=1).loc[h.index]
    if return_sharpe_ratio:
        return {"std": pnl.std() * np.sqrt(252), "sharpe_ratio": pnl.pipe(sharpe_ratio)}
    else:
        return pnl.std() * np.sqrt(252)
```

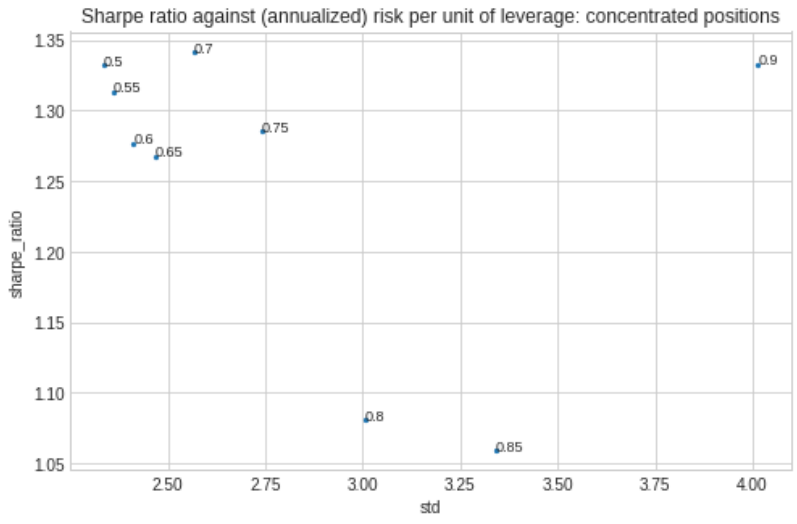
[25]:
```
from sklearn.base import TransformerMixin


def func(x, q):
    return x.where(
        x.ge(x.quantile(q=q, axis=1), axis=0)
        | x.le(x.quantile(q=1 - q, axis=1), axis=0),
        0,
    )


class Concentrate(TransformerMixin):
    def __init__(self, q=0.75):
        self.q = q if q > 0.5 else 1 - q

    def fit(self, X, y=None):
        return self

    def transform(self, X, return_dataframe=False):
        if not isinstance(X, pd.DataFrame):
            X = pd.DataFrame(X)
        df = func(X, q=self.q)
        if return_dataframe:
            return df
        else:
            return df.values
```

[26]:
```
q = 0.75
hs[f"shrinkage=0.01, concentration q={q}"] = m_.h_.pipe(func, q=q)
```

[27]:
```
bar(
    pd.Series({k: risk_per_unit_of_leverage(v, ret) for k, v in hs.items()}),
    horizontal=True,
    title="Annualized risk per unit of leverage",
)
```



[28]:
```
df = pd.DataFrame.from_dict(
    {
        q: m_.h_.pipe(func, q=q).pipe(
            risk_per_unit_of_leverage, ret=ret, return_sharpe_ratio=True
        )
        for q in np.arange(0.5, 0.95, 0.05)
    },
    orient="index",
)


fig, ax = plt.subplots(1, 1, figsize=(8, 5))
ax.scatter(df.iloc[:, 0], df.iloc[:, 1], s=5, c="tab:blue")
ax.set_xlabel(df.columns[0])
ax.set_ylabel(df.columns[1])
ax.set_title(
    "Sharpe ratio against (annualized) risk per unit of leverage: concentrated positions"
)
for i, txt in enumerate(df.index):
    ax.text(df.iloc[i, 0], df.iloc[i, 1], round(txt, 3), fontsize=9)
```



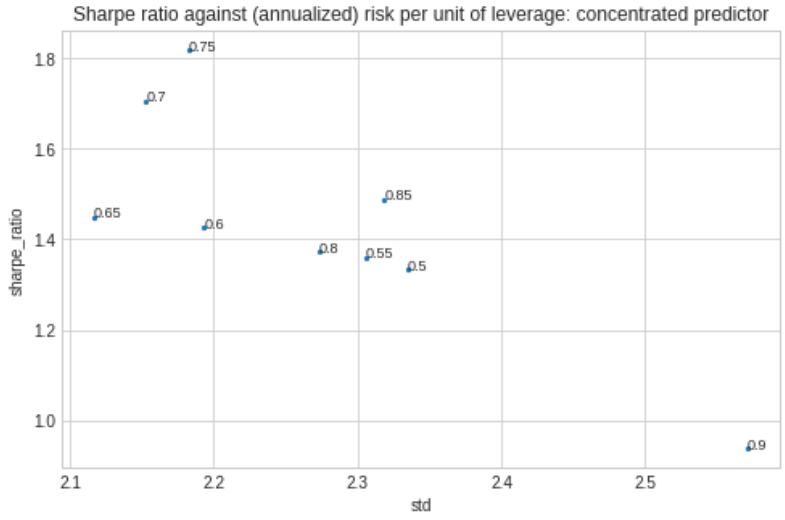
13.6 Concentrating the predictor instead of the positions

Instead of applying the concentration operator on positions, we can also apply on the predictor. The results are similar.

[29]:
```
from sklearn.pipeline import make_pipeline
```

[30]:
```
res = {}
for q in np.arange(0.5, 0.95, 0.05):
    estimator = make_pipeline(Concentrate(q=q), MeanVariance(transform_V=transform_V_))
    m2_ = Backtester(
        estimator, max_train_size=risk_model_window, start_date="2003-05-01"
    ).compute_holdings(features, target)
    res[q] = m2_.h_.pipe(risk_per_unit_of_leverage, ret=ret, return_sharpe_ratio=True)
```

[31]:
```
df = pd.DataFrame.from_dict(res, orient="index")

fig, ax = plt.subplots(1, 1, figsize=(8, 5))
ax.scatter(df.iloc[:, 0], df.iloc[:, 1], s=5, c="tab:blue")
ax.set_xlabel(df.columns[0])
ax.set_ylabel(df.columns[1])
ax.set_title(
    "Sharpe ratio against (annualized) risk per unit of leverage: concentrated predictor"
)
for i, txt in enumerate(df.index):
    ax.text(df.iloc[i, 0], df.iloc[i, 1], round(txt, 3), fontsize=9)
```

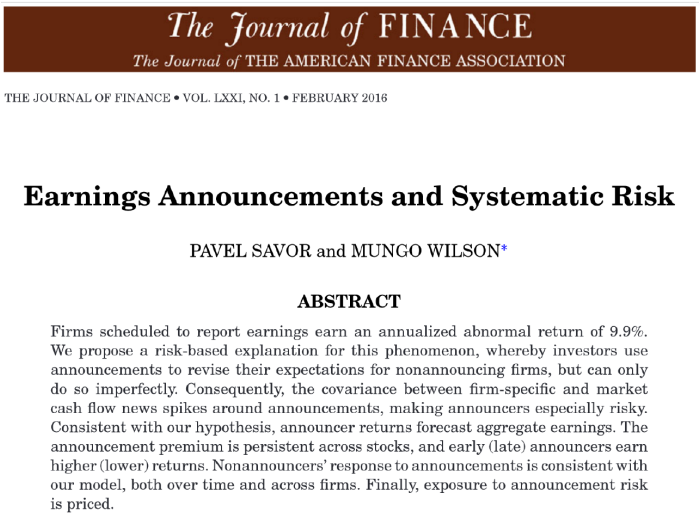


Chapter 14

Corporate event trading

In this section, we discuss the trading around corporate events, specifically earning announcements. For instannce in the US, firms releases their earnings generally on a quarterly frequency and this is a significant piece of information that subsequently gets incorporated into prices.

14.1 Earnings announcement premium



The Journal of FINANCE
The Journal of THE AMERICAN FINANCE ASSOCIATION

THE JOURNAL OF FINANCE • VOL. LXXI, NO. 1 • FEBRUARY 2016

Earnings Announcements and Systematic Risk

PAVEL SAVOR and MUNGO WILSON*

ABSTRACT

Firms scheduled to report earnings earn an annualized abnormal return of 9.9%. We propose a risk-based explanation for this phenomenon, whereby investors use announcements to revise their expectations for nonannouncing firms, but can only do so imperfectly. Consequently, the covariance between firm-specific and market cash flow news spikes around announcements, making announcers especially risky. Consistent with our hypothesis, announcer returns forecast aggregate earnings. The announcement premium is persistent across stocks, and early (late) announcers earn higher (lower) returns. Nonannouncers' response to announcements is consistent with our model, both over time and across firms. Finally, exposure to announcement risk is priced.

Main statements in Savor and Wilson (2016):

1. earnings announcement premium = 9.9% / year
2. announcing firms are "risky" and therefore there should be a compensation for risk
3. more precisely, firm earnings contain news about market cash-flow risk and therefore matter for aggreate risk

We focus on the first point.

Table I
Summary Statistics with Expected Announcement Dates

This table presents summary statistics for calendar-time announcer and nonannouncer portfolios (Panel A) and event-time announcer returns (Panel B). *Ret Ann* is the weekly excess return (value- or equal-weighted) of a portfolio consisting of all firms announcing in a given week, based on expected announcement dates. *Ret Non* is the weekly excess return (value- or equal-weighted) of a portfolio consisting of all nonannouncing firms. *N(Ann)* is the number of firms in the announcer portfolio, and *N(Non)* is the number of firms in the nonannouncer portfolio. *Excess Ret* is an announcing firm's excess return, and *Ab. Ret* is the same firm's return in excess of the market return. The sample is from 1974 to 2012. Returns are expressed in percentage points.

	Mean	Median	Min	Perc. 10	Perc. 90	Max	t-stat	*SD*	Skew.	Kurt.
Panel A: Calendar Time ($N = 2{,}035$)										
Ret Ann (vw)	0.32	0.40	−17.27	−2.94	3.35	17.54	5.00	2.86	−0.11	4.08
Ret Non (vw)	0.13	0.28	−18.77	−2.42	2.52	16.09	2.45	2.35	−0.41	6.79
Ann – Non (vw)	0.19	0.11	−9.10	−1.41	1.88	12.48	5.27	1.62	0.65	8.23
Ret Ann (ew)	0.35	0.40	−18.43	−2.52	3.06	17.34	6.06	2.60	−0.53	6.13
Ret Non (ew)	0.22	0.34	−19.06	−2.13	2.44	15.81	4.37	2.25	−0.84	8.79
Ann – Non (ew)	0.13	0.11	−4.31	−0.97	1.30	12.48	6.06	0.98	0.15	2.43
N(Ann)	308	176	0	55	747	1,509		305	1.6	2.0
N(Non)	6,305	6,393	4,250	4,730	7,970	9,047		1,114	0.2	−0.5
Panel B: Event Time ($N = 626{,}567$)										
Excess Ret	0.26	−0.11	−96.47	−8.50	9.03	473.28	21.73	9.50	3.26	76.60
Ab. Ret	0.15	−0.24	−97.35	−8.26	8.53	473.86	13.14	9.50	3.26	76.60

Table II
Earnings Announcement Premium

This table presents calendar-time abnormal returns for the long-short earnings announcement factor portfolio. Every week all stocks are divided into those that announce earnings and those that do not, based on their expected announcement dates. Portfolio returns equal those of a strategy that buys all announcing stocks and sells short nonannouncing stocks. Alphas are computed using the CAPM, the Fama-French three-factor model, and the Fama-French + momentum model. Returns are expressed in percentage points. t-statistics are given in brackets.

	Excess Ret.	Alpha	Mktrf	SMB	HML	UMD	R^2
Panel A: Value-Weighted Earnings Announcement Premium (%)							
1974–2012	0.19	0.19	0.02				0.12
	[5.27]	[5.18]	[1.54]				
1974–2012	0.19	0.19	0.01	0.06	−0.10		0.90
	[5.27]	[5.40]	[0.52]	[1.93]	[−3.27]		
1974–2012	0.19	0.19	0.01	0.06	−0.08	0.03	1.06
	[5.27]	[5.19]	[0.88]	[1.88]	[−2.66]	[1.82]	
Panel B: Equal-Weighted Earnings Announcement Premium (%)							
1974–2012	0.13	0.12	0.10				5.25
	[6.06]	[5.66]	[10.61]				
1974–2012	0.13	0.12	0.10	0.00	0.03		5.36
	[6.06]	[5.49]	[10.61]	[0.21]	[1.55]		
1974–2012	0.13	0.12	0.10	0.00	0.02	−0.01	5.39
	[6.06]	[5.54]	[10.24]	[0.23]	[1.29]	[−0.77]	
Panel C: Value-Weighted Earnings Announcement Premium (Subsamples) (%)							
1974–1986	0.11	0.10	0.08	0.08	0.06	0.01	1.93
	[2.28]	[2.15]	[3.30]	[1.83]	[1.17]	[0.17]	
1987–1999	0.26	0.24	0.03	0.13	−0.10	0.11	3.63
	[4.92]	[4.44]	[0.94]	[2.86]	[−1.51]	[2.43]	
2000–2012	0.20	0.21	−0.02	0.02	−0.10	0.02	0.84
	[2.48]	[2.59]	[−0.47]	[0.29]	[−1.76]	[0.58]	

14.2 Regulatory filings

The main regulatory filings to the U.S. Securities and Exchange Commission (SEC) are known as the 10-Ks and 10-Qs.

- The 10-K is an annual report required by the U.S. Securities and Exchange Commission (SEC) that gives a summary of the company's financial performance (and includes information such as company history, executive compensation, etc).
- The 10-Q is a quarterly report and contains similar information as the 10-K, but with less details.

The regulatory filings and the earning conference calls take place typically on the same day, so that all the market-moving information is disclosed to the market at the same time.

14.3 Filing dates from 10-Ks/10-Qs

In this section, we use the sample of firms from the daily stock return dataset and match it to the McDonald repository (https://sraf.nd.edu/) used in particular in the Loughran-McDonald paper.

[6]:
```
from skfin.datasets import load_10X_summaries, load_sklearn_stock_returns, mapping_10X
from skfin.plot import bar, line

ret = load_sklearn_stock_returns(cache_dir="data")
```

```
INFO:skfin.datasets:logging from cache directory: data/sklearn_returns.parquet
```

[7]:
```
df = load_10X_summaries()
```

```
INFO:skfin.datasets:logging from cache directory: data/Loughran-
McDonald_10X_Summaries_1993-2021.csv
```

[8]:
```
df.sample(n=5).iloc[:, :10]
```

[8]:

	CIK	FILING_DATE	ACC_NUM	CPR	FORM_TYPE	\
1009736	95574	20170427	0001437749-17-007255	20170331	10-Q	
410663	819577	20021114	0000793631-02-000011	20020930	10QSB	
306474	276750	20010214	0000276750-01-000002	20001231	10QSB	
942561	1498210	20150304	0001014897-15-000057	20140831	10-K	
1055508	879911	20181109	0001615774-18-012396	20180930	10-Q	

	CoName	SIC	FFInd	N_Words	\
1009736	SUPERIOR UNIFORM GROUP INC	2300	10	12432	
410663	AEI REAL ESTATE FUND XVII LIMITED PARTNERSHIP	6500	46	9261	
306474	AMBER RESOURCES CO	1311	30	2402	
942561	RJD Green Inc.	7374	34	8512	
1055508	APPLIED ENERGETICS INC.	3812	36	9758	

	N_Unique_Words
1009736	1478
410663	1308
306474	602
942561	1265
1055508	1145

The mapping of stock tickers to company name is `mapping_10X`:

- given that the name of firms can change (e.g. "Dell computer corp" becoming "Dell inc"), all the possible names need to be tracked.

[9]:
```
random.choices(list(mapping_10X.items()), k=10)
```

[9]:
```
[('COP', 'CONOCOPHILLIPS'),
 ('NAV', 'NAVISTAR INTERNATIONAL CORP'),
 ('IBM', 'INTERNATIONAL BUSINESS MACHINES CORP'),
 ('DELL', ['DELL COMPUTER CORP', 'DELL INC']),
 ('PEP', 'PEPSI BOTTLING GROUP INC'),
 ('HPQ', 'HEWLETT PACKARD CO'),
 ('HPQ', 'HEWLETT PACKARD CO'),
 ('IBM', 'INTERNATIONAL BUSINESS MACHINES CORP'),
 ('MMM', '3M CO'),
 ('F', 'FORD MOTOR CO')]
```

The table below shows the number of regulatory filings over time for the selected firms.

[10]:
```
pd.DataFrame.from_dict(
    {
        k: df.loc[lambda x: x.CoName.isin(v if isinstance(v, list) else [v])]
        .set_index("date")
        .loc[ret.index[0] : ret.index[-1]]
        .groupby(["FORM_TYPE"])["FILING_DATE"]
        .count()
        for k, v in mapping_10X.items()
    },
    orient="index",
).assign(
    **{
        "10_K_Q": lambda x: x["10-K"] + x["10-Q"],
        "restatements": lambda x: x["10-K-A"] + x["10-Q-A"],
    }
).sort_values(
    ["10_K_Q", "restatements"]
).fillna(
    0
).astype(
    int
)
```

[10]:

	10-K	10-Q	10-K-A	10-Q-A	10_K_Q	restatements
NAV	3	9	0	0	12	0
CVC	6	12	6	2	18	8
BA	5	15	1	1	20	2
PEP	5	15	1	1	20	2
F	5	15	1	3	20	4
GE	5	15	2	3	20	5
WFC	5	15	2	3	20	5
JPM	5	15	5	1	20	6
TWX	5	15	2	4	20	6
AIG	5	15	2	5	20	7
XRX	5	15	3	4	20	7
CAT	5	15	7	2	20	9
AAPL	5	15	0	0	20	0
AMZN	5	15	0	0	20	0
AXP	5	15	0	0	20	0
BAC	5	15	0	0	20	0
CL	5	15	0	0	20	0
CMCSA	5	15	0	0	20	0
COP	5	15	0	1	20	0
CSCO	5	15	0	0	20	0
CVS	5	15	0	0	20	0
CVX	5	15	0	0	20	0
DD	5	15	0	1	20	0
DELL	5	15	0	1	20	0
GD	5	15	1	0	20	0
GS	5	15	0	0	20	0
HD	5	15	0	0	20	0
IBM	5	15	0	1	20	0
K	5	15	0	0	20	0
KMB	5	15	0	0	20	0
KO	5	15	0	0	20	0
MAR	5	15	0	0	20	0
MCD	5	15	0	0	20	0
MMM	5	15	0	0	20	0
MSFT	5	15	0	1	20	0
NOC	5	15	0	0	20	0
PFE	5	15	0	0	20	0

```
PG      5   15      0      0      20            0
R       5   15      0      0      20            0
RTN     5   15      1      0      20            0
TXN     5   15      0      0      20            0
VLO     5   15      0      2      20            0
WMT     5   15      1      0      20            0
XOM     5   15      0      0      20            0
YHOO    5   15      0      0      20            0
HPQ     6   15      0      0      21            0
```

14.3.1 mapping checks

[11]:
```
v = mapping_10X["CVC"]
print(v)
df.loc[lambda x: x.CoName.isin(v if isinstance(v, list) else [v])].set_index(
    "date"
).loc[ret.index[0] : ret.index[-1]].loc["2006"].iloc[:, :10]
```

```
CABLEVISION SYSTEMS CORP /NY
```

[11]:

date	CIK	FILING_DATE	ACC_NUM	CPR	FORM_TYPE	\
2006-05-10	1053112	20060510	0001104659-06-032803	20060331	10-Q	
2006-05-10	784681	20060510	0001104659-06-032803	20060331	10-Q	
2006-09-21	1053112	20060921	0001104659-06-062197	20051231	10-K-A	
2006-09-21	784681	20060921	0001104659-06-062197	20051231	10-K-A	
2006-09-21	1053112	20060921	0001104659-06-062201	20060331	10-Q-A	
2006-09-21	784681	20060921	0001104659-06-062201	20060331	10-Q-A	
2006-11-08	1053112	20061108	0001104659-06-072875	20060930	10-Q	
2006-11-08	784681	20061108	0001104659-06-072875	20060930	10-Q	

date	CoName	SIC	FFInd	N_Words	N_Unique_Words
2006-05-10	CABLEVISION SYSTEMS CORP /NY	4841	32	38566	2327
2006-05-10	CABLEVISION SYSTEMS CORP /NY	4841	32	38566	2327
2006-09-21	CABLEVISION SYSTEMS CORP /NY	4841	32	92495	2946
2006-09-21	CABLEVISION SYSTEMS CORP /NY	4841	32	92495	2946
2006-09-21	CABLEVISION SYSTEMS CORP /NY	4841	32	35794	1966
2006-09-21	CABLEVISION SYSTEMS CORP /NY	4841	32	35794	1966
2006-11-08	CABLEVISION SYSTEMS CORP /NY	4841	32	45265	2148
2006-11-08	CABLEVISION SYSTEMS CORP /NY	4841	32	45265	2148

Matching company names is often a time-consuming task: below we use the package `rapidfuzz` to check the candidate matches.

[12]:
```
CoName = list(
    df.assign(CoName=lambda x: x.CoName.str.upper())
    .groupby(["date", "CoName"])["FILING_DATE"]
    .count()
    .loc[ret.index[0] : ret.index[-1]]
    .groupby(level=1)
    .count()
    .index
)

from rapidfuzz import fuzz

pd.Series({c: fuzz.token_set_ratio("CABLEVISION", c) for c in CoName}).sort_values(
    ascending=False
).head(5)
```

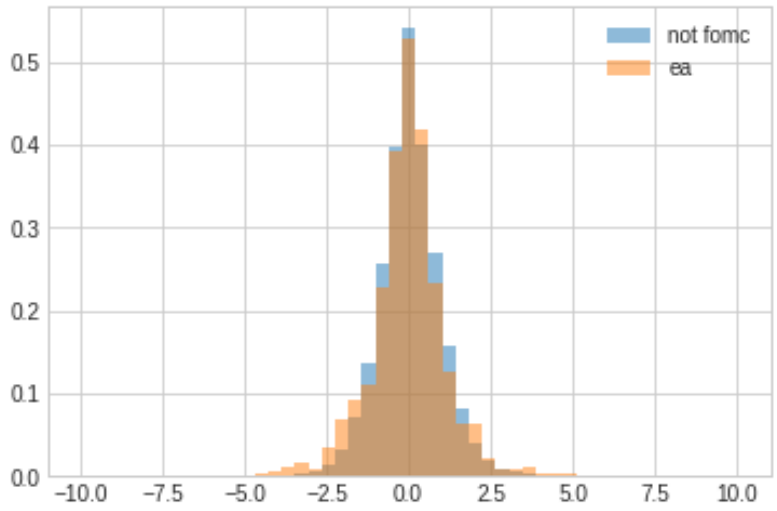
[12]:
```
CABLEVISION SYSTEMS CORP /NY    100.000000
PARKERVISION INC                 66.666667
CAGLES INC                       66.666667
ABLEST INC                       66.666667
LATIN TELEVISION  INC            64.516129
dtype: float64
```

14.4 Stock returns on filing dates

[13]:
```
ret_norm = ret.pipe(lambda x: x.div(x.ewm(halflife=63, min_periods=21).std())).dropna(
    how="all", axis=0
)

mask = (
    pd.concat(
        {
            k: df.loc[lambda x: x.CoName.isin(v if isinstance(v, list) else [v])]
            .set_index("date")
            .loc["2002-01-01" : ret.index[-1]]["FORM_TYPE"]
            for k, v in mapping_10X.items()
        }
    )
    .groupby(level=[1, 0])
    .count()
)

funcs = {
    "ea": lambda x: x.loc[x.FORM_TYPE.notna()].drop("FORM_TYPE", axis=1),
    "not_ea": lambda x: x.loc[x.FORM_TYPE.isna()].drop("FORM_TYPE", axis=1),
}

ret_ea = pd.concat(
    {
        k: ret_norm.stack().rename("ret").to_frame().join(mask).pipe(v).squeeze()
        for k, v in funcs.items()
    },
    axis=1,
)
```

The histogram below shows more extreme returns on filing dates.

[14]:
```
bins = np.linspace(-10, 10, 50)
plt.hist(ret_ea["not_ea"].dropna(), bins, density=True, alpha=0.5, label="not fomc")
plt.hist(ret_ea["ea"].dropna(), bins, density=True, alpha=0.5, label="ea")
plt.legend(loc="upper right")
plt.show()
```



Statistically, this is confirmed with a higher volatility on filing dates. On this sample, the average returns do not seem different.

[15]:
```
pd.concat(
    {"Average volatility": ret_ea.std(), "Average mean": ret_ea.mean()}, axis=1
).round(2)
```

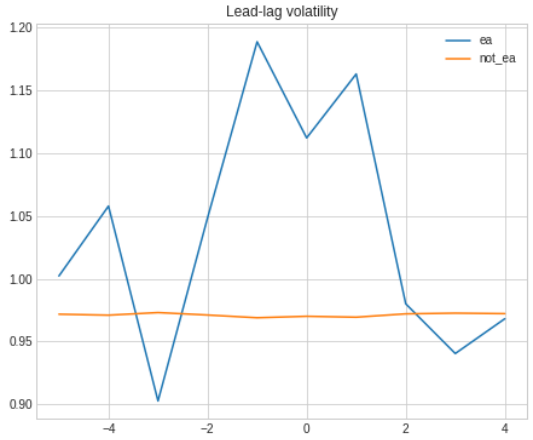
[15]:

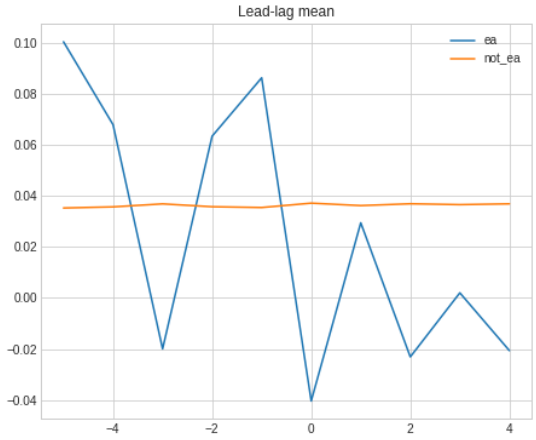
	Average volatility	Average mean
ea	1.11	-0.04
not_ea	0.97	0.04

[16]:
```
ea_std, ea_mean = {}, {}
for i in range(-5, 5):
    mask_ = mask.unstack().reindex(ret.index).shift(i).stack().rename("FORM_TYPE")
    ret_ea_ = pd.concat(
        {
            k: ret_norm.stack().rename("ret").to_frame().join(mask_).pipe(v).squeeze()
            for k, v in funcs.items()
        },
        axis=1,
    )
    ea_std[i] = ret_ea_.std()
    ea_mean[i] = ret_ea_.mean()
```

[17]:
```
fig, ax = plt.subplots(1, 2, figsize=(16, 6))
line(
    pd.DataFrame(ea_std).T,
    title="Lead-lag volatility",
    sort=False,
    ax=ax[0],
    bbox_to_anchor=None,
    loc="best",
)
line(
    pd.DataFrame(ea_mean).T,
    title="Lead-lag mean",
    sort=False,
    ax=ax[1],
    bbox_to_anchor=None,
```

```
    loc="best",
)
```


Lead-lag volatility
ea
not_ea
1.20
1.15
1.10
1.05
1.00
0.95
0.90
−4
−2
0
2
4


Lead-lag mean
ea
not_ea
0.10
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
−4
−2
0
2
4

Chapter 15

Corporate sentiment

In this section, we describe two strategies to measure sentiment in text. A first strategy is based a dictionary of positive and negative words: simple word counts capture the polarity of the document. A second strategy relies on fitting a linear model with target as returns and features as word counts: in this case, the sign and amplitude of the learned coefficients is used to determined the document sentiment.

Two main papers:

- Loughran and McDonald (2011): "When is a Liability not a Liability? Textual Analysis, Dictionaries and 10-Ks," *Journal of Finance*
- Jegadeesh and Wu (2013): "Word Power: A New Approach for Content Analysis," *Journal of Financial Economics*

15.1 Rule-based sentiment

Loughran-McDonalds (2011): textual analysis in finance/accounting to examine the tone and sentiment of corporate 10-K reports. Two statements:

1. a Finance-specific dictionary of negative words matters
2. weighting (e.g. tf.idf weights) matters

Bag of words method: parse the 10-K documents into vectors of words and word counts.

- Dictionaries: http://www3.nd.edu/~mcdonald/Word_Lists.html:
- sentiment negative and positive words
- uncertainty (e.g. approximate, contingency, depend, fluctuate, indefinite, uncertain, and variability)
- litigation (e.g. claimant, deposition, interlocutory, testimony, and tort.)
- modal words are related to levels of confidence: strong modal words (e.g. always, highest, must, and will) and weak model words (e.g. could, depending, might)

The table below shows the most frequent sentiment words in the full 10-K document in the "Management Discussion and Analysis" subsection.

Table III—*Continued*

Panel B: Fin-Neg

Full 10-K Document				MD&A Subsection			
Word in H4N-Inf	Word	% of Total Fin-Neg Word Count	Cumulative %	Word in H4N-Inf	Word	% of Total Fin-Neg Word Count	Cumulative %
√	LOSS	9.73%	9.73%	√	LOSS	9.51%	9.51%
√	LOSSES	5.67%	15.40%	√	LOSSES	7.58%	17.10%
	CLAIMS	3.15%	18.55%	√	IMPAIRMENT	4.71%	21.81%
√	IMPAIRMENT	3.04%	21.59%		RESTRUCTURING	2.93%	24.74%
√	AGAINST	2.58%	24.17%	√	DECLINE	2.89%	27.62%
√	ADVERSE	2.44%	26.61%		CLAIMS	2.71%	30.33%
	RESTATED	2.09%	28.70%	√	ADVERSE	2.44%	32.77%
√	ADVERSELY	1.75%	30.45%	√	AGAINST	2.01%	34.78%
	RESTRUCTURING	1.72%	32.17%	√	ADVERSELY	1.94%	36.72%
	LITIGATION	1.67%	33.83%		LITIGATION	1.67%	38.40%
	DISCONTINUED	1.57%	35.40%		CRITICAL	1.63%	40.03%
	TERMINATION	1.35%	36.75%		DISCONTINUED	1.62%	41.64%
√	DECLINE	1.19%	37.93%	√	*DECLINED*	1.30%	42.94%
√	CLOSING	1.08%	39.01%		TERMINATION	1.06%	44.00%
√	FAILURE	0.97%	39.98%	√	NEGATIVE	0.96%	44.96%
	UNABLE	0.84%	40.82%	√	FAILURE	0.93%	45.89%
√	*DAMAGES*	0.82%	41.64%		UNABLE	0.91%	46.80%
√	DOUBTFUL	0.77%	42.41%	√	CLOSING	0.86%	47.65%
√	LIMITATIONS	0.75%	43.17%		*NONPERFORMING*	0.81%	48.47%
√	FORCE	0.74%	43.91%	√	IMPAIRED	0.81%	49.28%
√	VOLATILITY	0.73%	44.64%	√	VOLATILITY	0.79%	50.07%
	CRITICAL	0.73%	45.37%	√	FORCE	0.75%	50.82%
√	IMPAIRED	0.70%	46.07%	√	*NEGATIVELY*	0.73%	51.56%
	TERMINATED	0.70%	46.77%	√	DOUBTFUL	0.72%	52.27%
√	*COMPLAINT*	0.63%	47.39%	√	*CLOSED*	0.70%	52.97%
√	DEFAULT	0.57%	47.96%	√	DIFFICULT	0.69%	53.66%
√	NEGATIVE	0.51%	48.47%	√	*DECLINES*	0.63%	54.29%
√	*DEFENDANTS*	0.51%	48.99%	√	*EXPOSED*	0.60%	54.89%
√	*PLAINTIFFS*	0.51%	49.49%	√	DEFAULT	0.59%	55.48%
√	DIFFICULT	0.50%	50.00%	√	*DELAYS*	0.56%	56.04%

The following table shows that sentiment words reprenet about 1% of all words. In these sections, there are more words deemed negative (mean=1.39%) than positive (mean=0.75%).

	Full 10-K Document ($N = 50{,}115$)		
Variable	Mean	Median	Standard Deviation
Word Lists			
H4N-Inf (H4N w/ inflections)	3.79%	3.84%	0.76%
Fin-Neg (negative)	1.39%	1.36%	0.55%
Fin-Pos (positive)	0.75%	0.74%	0.21%
Fin-Unc (uncertainty)	1.20%	1.20%	0.32%
Fin-Lit (litigious)	1.10%	0.95%	0.53%
MW-Strong (strong modal words)	0.26%	0.24%	0.11%
MW-Weak (weak modal words)	0.43%	0.39%	0.21%

The following table shows the predictibility of sentiment for 4-day forward value-weighted excess return (1994-2008). More precisely, the `Fin-Neg` (negative) sentiment predicts returns with a t-stat from -2.64 to -3.11 after controlling for risk factors.

	Proportional Weights		tf.idf Weights	
	(1)	(2)	(3)	(4)
Word Lists				
H4N-Inf (Harvard-IV-4-Neg with inflections)	−7.422		−0.003	
	(−1.35)		(−3.16)	
Fin-Neg (negative)		−19.538		−0.003
		(−2.64)		(−3.11)
Control Variables				
Log(size)	0.123	0.127	0.131	0.132
	(2.87)	(2.93)	(2.96)	(2.97)
Log(book-to-market)	0.279	0.280	0.273	0.277
	(3.35)	(3.45)	(3.37)	(3.41)
Log(share turnover)	−0.284	−0.269	−0.254	−0.255
	(−2.46)	(−2.36)	(−2.32)	(−2.31)
Pre_FFAlpha	−2.500	−3.861	−5.319	−6.081
	(−0.06)	(−0.09)	(−0.12)	(−0.14)
Institutional ownership	0.278	0.261	0.254	0.255
	(0.93)	(0.86)	(0.87)	(0.87)
NASDAQ dummy	0.073	0.073	0.083	0.080
	(0.86)	(0.87)	(0.97)	(0.94)
Average R^2	2.44%	2.52%	2.64%	2.63%

The table below shows the same results for several word dictionaries.

		Finance Dictionaries					
Dependent Variable	H4N-Inf	Negative	Positive	Uncertainty	Litigious	Modal Strong	Modal Weak
Panel A: Proportional Weights							
Event period excess return	−7.422	−19.538	−21.696	−42.026	9.705	−149.658	−60.230
	(−1.35)	(−2.64)	(−1.18)	(−4.13)	(1.17)	(−3.82)	(−2.43)
Event period abnormal volume	2.735	6.453	−1.957	2.220	0.057	21.430	4.300
(coefficient /100)	(2.02)	(3.11)	(−0.20)	(0.48)	(0.02)	(1.67)	(0.74)
Postevent return volatility	11.336	34.337	18.803	33.973	−0.299	152.312	59.239
	(8.59)	(12.59)	(3.47)	(8.34)	(−0.23)	(12.32)	(8.58)
Panel B: tf.idf Weights							
Event period excess return	−0.003	−0.003	−0.011	−0.022	−0.001	−0.065	−0.080
	(−3.16)	(−3.11)	(−2.27)	(−4.04)	(−0.62)	(−2.28)	(−3.44)
Event period abnormal volume	0.086	0.098	0.159	0.409	0.135	0.046	0.864
	(4.30)	(4.40)	(1.03)	(2.50)	(2.60)	(0.03)	(1.21)
Postevent return volatility	0.004	0.004	0.014	0.020	0.006	0.073	0.069
	(12.91)	(11.87)	(12.52)	(8.95)	(10.10)	(7.47)	(8.21)

15.2 Learning-based sentiment

Jegadessh and Wu (2013) discuss how to fit word weights to better identify terms that drive returns.

- The idea is to identify sentiment words associated to significant return moves (either positive or negative) when firm file 10Ks with the SEC.

Learning for document d:

$$r_{d,t\to t+3} = a + \sum_{v\in LM} b_v \frac{count_{d,v}}{length_v} + e$$

where the terms v are in the base sentiment vocabulary LM from Loughran and McDonalds.

Out-of-sample forecast:

$$Score_d = \sum_v \left(\frac{b_v - \bar{b}}{\sqrt{Var(b_j)}} \right) \frac{cound_{d,v}}{length_d}$$

$$r_{d,t+5 \to t+w} = \alpha + \beta Score_d + \epsilon$$

where the statistical significance of β is evaluated using Fama-MacBeth statistics.

The table below shows the most impactful words from the regressions.

Table 3
Top five most positive and negative words within frequency quintiles.
This table presents the five positive and words with the largest word power weights within each term frequency quintile. Term frequency of each word is the percentage of 10-Ks in which the word appears. Frequency Quintile 1 contains the quintile of words with the lowest frequency and Frequency Quintile 5 contains the quintile of words with the highest frequency. This table reports the word power weights computed using Eqs. (6) and (7) fitted over the sample period of 1995–2010.

Frequency Quintiles				
1	2	3	4	5
Panel A: Top five most positive words				
ingenuity	influential	exceptional	adequately	favorable
acclaimed	optimistic	proficient	highest	strong
revolutionize	enthusiastic	transparency	progress	gain
courteous	excited	versatile	desirable	efficiency
incredible	regain	compliment	encouraged	opportunity
Panel B: Top five most negative words				
imperil	turbulent	disapprove	unplanned	unresolved
disavow	overestimate	reluctant	illegal	unsuccessful
insubordination	underinsured	uncontrollable	wasteful	discourage
bailout	aggravate	setback	misuse	unauthorized
dismal	unfortunate	turmoil	strain	insufficient

The table below shows taht the words identified in the regressions are not the same as the one with high `tfidf` weights.

Table 4
Comparison of word power weights and *idf* term weights.
This table presents the top and bottom ten positive and negative words based on word power weights and their *idf* term weights. This table reports the words with the largest word power weights based on Eqs. (6) and (7) fitted over the sample period of 1995–2010. The *idf* term weighting scheme assigns weights inversely proportional to document frequency, as described in Eq. (1). The document frequency of each word is the percentage of 10-Ks in which the word appears. Panels A and B present the top and bottom positive and negative words, respectively.

	Most impactful words			Least impactful words	
	WP rank	*idf* rank		WP rank	*idf* Rank
Panel A: Positive words					
ingenuity	1	14	lucrative	123	13
acclaimed	2	7	tremendous	122	35
influential	3	26	worthy	121	22
revolutionize	4	19	happy	120	9
optimistic	5	42	spectacular	119	21
enthusiasm	6	29	beautiful	118	15
excited	7	48	smooth	117	60
courteous	8	20	conducive	116	27
regain	9	39	receptive	115	30
incredible	10	3	proactive	114	38
Panel B: Negative words					
imperil	1	18	dispossess	718	8
disavow	2	22	ridicule	717	2
insubordination	3	20	mischief	716	27
bailout	4	31	derogatory	715	4
dismal	5	10	disorderly	714	3
untruthful	6	39	disassociate	713	35
unwelcome	7	5	immoral	712	23
turbulent	8	140	irreconciliable	711	19
vitiate	9	38	disgrace	710	1
undocumented	10	55	extenuating	709	34

The table shows that the learned sentiment (as the `WP` or word power score) predicts 4-day forward returns – even after controlling for known risk factors.

	Models			
	Combined LM Lexicon		Global Lexicon	
	(1)	(2)	(3)	(4)
Term Weighting Scheme				
WP	0.343	0.192	0.294	0.190
	(2.67)	(3.81)	(2.44)	(3.58)
Control Variables				
Size		-0.018		-0.019
		(-0.21)		(-0.14)
BM		2.631		2.461
		(1.45)		(1.00)
Volatility		-0.312		-0.334
		(-1.75)		(-1.84)
Turnover		-0.117		-0.123
		(-1.56)		(-1.62)
EAD-Ret		0.575		0.546
		(5.80)		(6.38)
Accruals		-0.312		-0.312
		(-1.75)		(-1.62)

15.3 10-Ks

We use the 10-K/10-Q summary file from the McDonalds data repository to test some insights from the Loughran-McDonalds paper. The sentiment metric is: - sentiment = (#positive - #negative)/#words

[10]:
```
from skfin.datasets import load_10X_summaries, load_sklearn_stock_returns, mapping_10X
from skfin.metrics import sharpe_ratio

ret = load_sklearn_stock_returns(cache_dir="data")
```

```
INFO:skfin.datasets:logging from cache directory: data/sklearn_returns.parquet
```

[11]:
```
df = load_10X_summaries()
```

```
INFO:skfin.datasets:logging from cache directory: data/Loughran-
McDonald_10X_Summaries_1993-2021.csv
```

[12]:
```
df.columns
```

[12]:
```
Index(['CIK', 'FILING_DATE', 'ACC_NUM', 'CPR', 'FORM_TYPE', 'CoName', 'SIC',
       'FFInd', 'N_Words', 'N_Unique_Words', 'N_Negative', 'N_Positive',
       'N_Uncertainty', 'N_Litigious', 'N_StrongModal', 'N_WeakModal',
       'N_Constraining', 'N_Negation', 'GrossFileSize', 'NetFileSize',
       'NonTextDocTypeChars', 'HTMLChars', 'XBRLChars', 'XMLChars',
       'N_Exhibits', 'date'],
      dtype='object')
```

[13]:
```
sentiment_func = lambda x: (x.N_Positive - x.N_Negative) / x.N_Words

sent = (
    pd.concat(
        {
            k: df.loc[lambda x: x.CoName.isin(v if isinstance(v, list) else [v])]
            .set_index("date")
            .loc["2002-01-01" : ret.index[-1]]
            .pipe(sentiment_func)
```

```
            for k, v in mapping_10X.items()
        }
    )
    .groupby(level=[0, 1])
    .mean()
)
```

From the sentiment metrics, we follows the steps to contruct a predictor:

- take the sentiment difference from on filing to the previous to measure improvement or deterioration (and remove biases)
- forward-fill the value for one-month (=21 business days) to have an active position over a limited window
- remove the cross-sectional mean (and standardise) so that the predictor have long-short risk-managed positions.

[14]:
```
pred = (
    sent.sort_index(level=[0, 1])
    .groupby(level=0)
    .transform("diff")
    .unstack(level=0)
    .resample("B")
    .last()
    .ffill(limit=21)
    .pipe(lambda x: x.sub(x.mean(axis=1), axis=0).div(x.std(axis=1), axis=0))
    .reindex(ret.index)
)
```

[15]:
```
line(pred.shift(2).mul(ret).sum(axis=1), cumsum=True)
```

Interesting to note that over this period, two firms contribute disportionately to the pnl – Apple and Goldman Sachs.

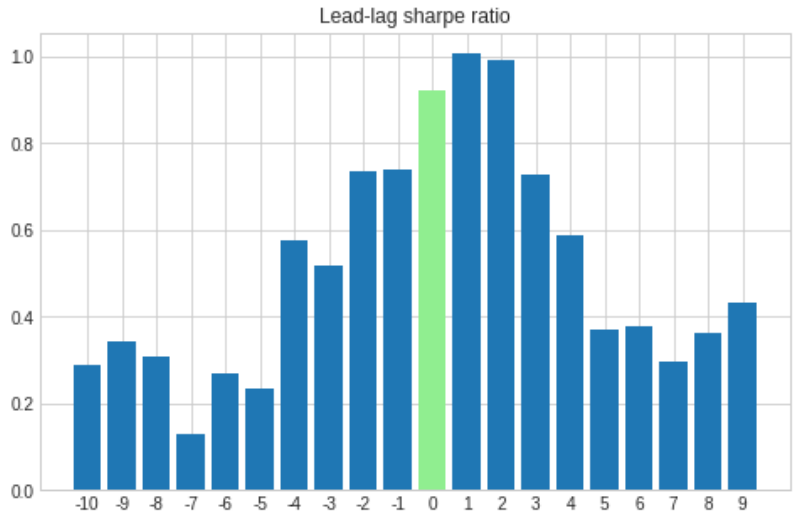
[16]:
```
pred.shift(2).mul(ret).dropna(how="all", axis=1).sum().pipe(
    lambda x: pd.concat([x.nlargest(), x.sort_values(ascending=False).tail(5)])
).rename("Stock-level pnl contribution").to_frame()
```

[16]:

	Stock-level pnl contribution
AAPL	109.887671
GS	97.708109
WFC	27.697014
YHOO	26.637468
DELL	25.705414
TWX	-9.531156
AIG	-10.113330
GD	-21.962501
BAC	-27.134267
MSFT	-35.148272

To assess the alpha decay, the graph below shows the sharpe ratio when the predictor is lagged (or led) by multiple business days.

[17]:
```
bar(
    {
        i: pred.shift(2 + i).mul(ret).sum(axis=1).pipe(sharpe_ratio)
        for i in range(-10, 10)
    },
    sort=False,
    baseline=0,
    title="Lead-lag sharpe ratio",
)
```

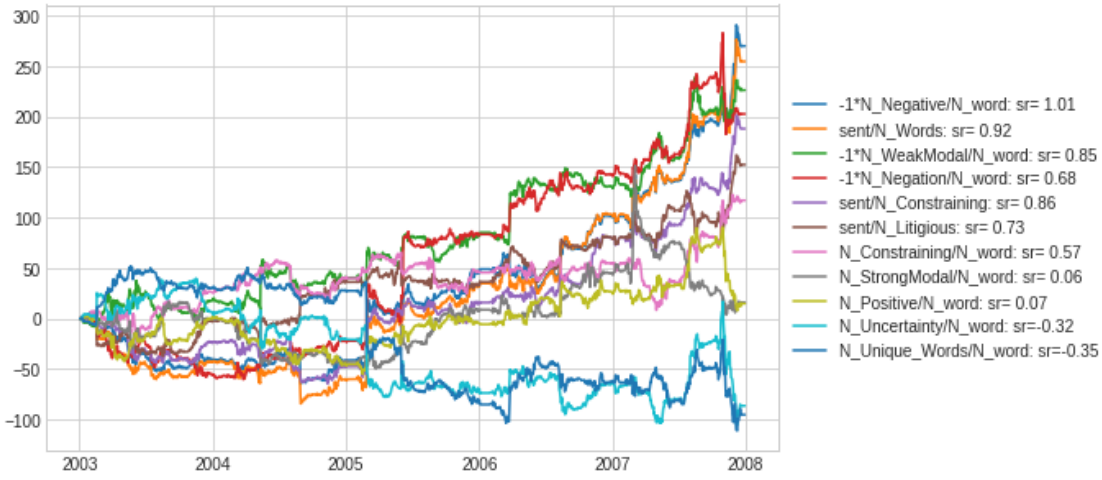


We can test different sentiment construct as shown in the graph below.

[18]:
```
pnls_ = {}
for c in ["N_Litigious", "N_Constraining", "N_Words"]:
    sentiment_func_ = lambda x: (x.N_Positive - x.N_Negative) / x[c]
    sent_ = (
        pd.concat(
            {
                k: df.loc[lambda x: x.CoName.isin(v if isinstance(v, list) else [v])]
                .set_index("date")
                .loc["2002-01-01" : ret.index[-1]]
                .pipe(sentiment_func_)
                for k, v in mapping_10X.items()
            }
```

```
        )
        .groupby(level=[0, 1])
        .mean()
    )

    pred_ = (
        sent_.sort_index(level=[0, 1])
        .groupby(level=0)
        .transform("diff")
        .unstack(level=0)
        .resample("B")
        .last()
        .ffill(limit=21)
        .pipe(lambda x: x.sub(x.mean(axis=1), axis=0).div(x.std(axis=1), axis=0))
        .reindex(ret.index)
    )

    pnls_[f"sent/{c}"] = pred_.shift(2).mul(ret).sum(axis=1)


for c in ["N_Negative", "N_Negation", "N_WeakModal"]:
    sentiment_func_ = lambda x: -1 * x[c] / x.N_Words
    sent_ = (
        pd.concat(
            {
                k: df.loc[lambda x: x.CoName.isin(v if isinstance(v, list) else [v])]
                .set_index("date")
                .loc["2002-01-01" : ret.index[-1]]
                .pipe(sentiment_func_)
                for k, v in mapping_10X.items()
            }
        )
        .groupby(level=[0, 1])
        .mean()
    )

    pred_ = (
        sent_.sort_index(level=[0, 1])
        .groupby(level=0)
        .transform("diff")
        .unstack(level=0)
        .resample("B")
        .last()
        .ffill(limit=21)
        .pipe(lambda x: x.sub(x.mean(axis=1), axis=0).div(x.std(axis=1), axis=0))
        .reindex(ret.index)
    )

    pnls_[f"-1*{c}/N_word"] = pred_.shift(2).mul(ret).sum(axis=1)


for c in [
    "N_Unique_Words",
    "N_Positive",
    "N_Uncertainty",
    "N_StrongModal",
    "N_Constraining",
]:
    sentiment_func_ = lambda x: x[c] / x.N_Words
    sent_ = (
        pd.concat(
```

```
            {
                k: df.loc[lambda x: x.CoName.isin(v if isinstance(v, list) else [v])]
                .set_index("date")
                .loc["2002-01-01" : ret.index[-1]]
                .pipe(sentiment_func_)
                for k, v in mapping_10X.items()
            }
        )
        .groupby(level=[0, 1])
        .mean()
    )

    pred_ = (
        sent_.sort_index(level=[0, 1])
        .groupby(level=0)
        .transform("diff")
        .unstack(level=0)
        .resample("B")
        .last()
        .ffill(limit=21)
        .pipe(lambda x: x.sub(x.mean(axis=1), axis=0).div(x.std(axis=1), axis=0))
        .reindex(ret.index)
    )

    pnls_[f"{c}/N_word"] = pred_.shift(2).mul(ret).sum(axis=1)
line(pnls_, cumsum=True)
```



Chapter 16

Timing backtest with learning

In previous sections, we studied the predictability of industry and stock returns in a long-short "cash-neutral" setting. In this section, we shift to the predictability of a single asset (ie. the "market" as the S&P 500 US index).

16.1 Timing the market

To evaluate the out-of-sample predictability of a variable, Welch-Goyal (2008) compare two regressions:

- conditional regression (based on the predictor)
- unconditional regression (based on a rolling mean)
- the comparison between the two regression provides a test of whether the predictor has any value.

Intuition

- "low" prices relative to dividends forecast higher subsequent returns
- other ratios (earnings, book value, moving average of past prices instead of dividends) should also work
- expected returns vary over the business cycle and higher risk premium required to get people to hold stocks at the bottom of a recession: dividend-price ratios can be interpreted a state-variable capturing business cycle risk

Critical view

- are the in-sample results robust out-of-sample?

Data

- dividend price ratio ("d/p"): difference between the log of dividends and the log of prices
- dividend yield ("d/y"): difference between the log of dividends and the log of lagged prices
- percent equity issuing ("equis"): ratio of equity issuing activity as a fraction of total issuing equity

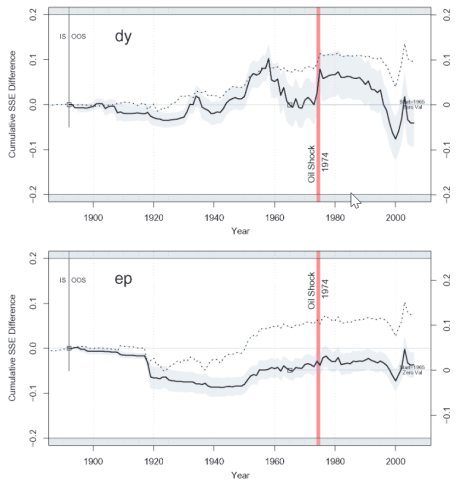


Figure 1
Annual performance of IS insignificant predictors.

Table 1
Forecasts at annual frequency
This table presents statistics on forecast errors in-sample (IS) and out-of-sample (OOS) for log equity premium forecasts at annual frequency (both in the forecasting equation and forecast). Variables are explained in Section 2. Stock returns are price changes, including dividends, of the S&P500. All numbers are in percent per year, except except $\overline{R}^2$ and power which are simple percentages. A star next to IS-$\overline{R}^2$ denotes significance of the in-sample regression as measured by F-statistics (critical values of which are obtained empirically from bootstrapped distributions). The column 'IS for OOS' gives the IS-$\overline{R}^2$ for the OOS period. ΔRMSE is the RMSE (root mean square error) difference between the unconditional forecast and the conditional forecast for the same sample/forecast period. Positive numbers signify superior out-of-sample conditional forecast. The OOS-$\overline{R}^2$ is defined in Equation 6. A star next to OOS-$\overline{R}^2$ is based on significance of MSE-F statistic by McCracken (2004), which tests for equal MSE of the unconditional forecast and the conditional forecast. One-sided critical values of MSE statistics are obtained empirically from bootstrapped distributions, except for caya and all models where they are obtained from McCracken (2004). Critical values for the ms model are not calculated. Power is calculated as the fraction of draws where the simulated ΔRMSE is greater than the empirically calculated 95% critical value. The two numbers under the power column are for all simulations and for those simulations in which the in-sample estimate was significant at the 95% level. Significance levels at 90%, 95%, and 99% are denoted by one, two, and three stars, respectively.

			Full Sample								1927–2005
			Forecasts begin 20 years after sample					Forecasts begin 1965			Sample
			IS	IS for	OOS			IS for	OOS		IS
	Variable	Data	$\overline{R}^2$	OOS $\overline{R}^2$	$\overline{R}^2$	ΔRMSE	Power	OOS $\overline{R}^2$	$\overline{R}^2$	ΔRMSE	Power	$\overline{R}^2$
Full Sample, Not Significant IS												
dfy	Default yield spread	1919–2005	−1.18		−3.29	−0.14		−4.15	−0.12			−1.31
infl	Inflation	1919–2005	−1.00		−4.07	−0.20		−3.56	−0.08			−0.99
svar	Stock variance	1885–2005	−0.76		−27.14	−2.33		−2.44	+0.01			−1.32
d/e	Dividend payout ratio	1872–2005	−0.75		−4.33	−0.31		−4.99	−0.18			−1.24
lty	Long term yield	1919–2005	−0.63		−7.72	−0.47		−12.57	−0.76			−0.94
tms	Term spread	1920–2005	0.16		−2.42	−0.07		−2.96	−0.03			0.89
tbl	Treasury-bill rate	1920–2005	0.34		−3.37	−0.14		−4.90	−0.18			0.15
dfr	Default return spread	1926–2005	0.40		−2.16	−0.03		−2.82	−0.02			0.32
d/p	Dividend price ratio	1872–2005	0.49		−2.06	−0.11		−3.69	−0.09			1.67
d/y	Dividend yield	1872–2005	0.91		−1.93	−0.10		−6.68	−0.31			**2.71***
ltr	Long term return	1926–2005	0.99		−11.79	−0.76		−18.38	−1.18			0.92
e/p	Earning price ratio	1872–2005	1.08		−1.78	−0.08		−1.10	0.11			**3.20***

Welch-Goyal summary: very little predictability and the oil shock 1974 important in explaining results in the literature.

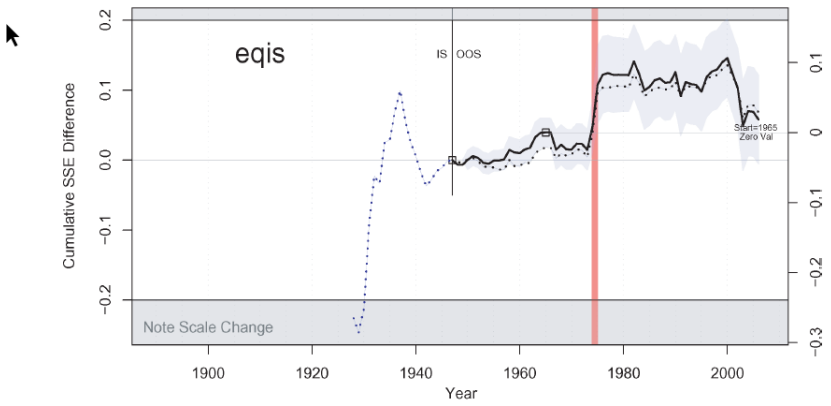


Figure 2
Annual performance of predictors that are not in-sample significant
Explanation: See Figure 1.

Campbell-Thompson: impose "sign-restrictions"

- "in practice, an investor would not use a perverse coefficient but would likely conclude that the coefficient is zero, in effect imposing prior knowledge on the output of the regression" (p. 1516)

Sign restrictions

- set the regression coefficient to zero whenever it has the "wrong" sign (different from the theoretically expected sign estimated over the sample)
- set the forecast equity premium to zero whenever it is negative

Summary: does dividend yield predict returns?

- Yes: dividend yield is a strong predictor in the 1970s and 1980s (in-sample!)
- No: the relationship became weaker in 1990s
- No: the statistical evidence is much weaker when adjusting for fact that regressors highly persistent
- No: dividend yield is also weak predictor out-of-sample –and rarely better than a moving-average.

Ways to improve predictability - Impose restrictions on coefficients (Campbell and Thompson, 2005)

16.2 Data

The data provided by Amit Goyal on the S&P 500 is essentially identical to the one provided by Ken French.

[5]:
```
from skfin.datasets import load_ag_features, load_kf_returns

df = load_ag_features()[:"1999"]
```

```
INFO:skfin.datasets:logging from cache file: data/PredictorData2021.xlsx
```

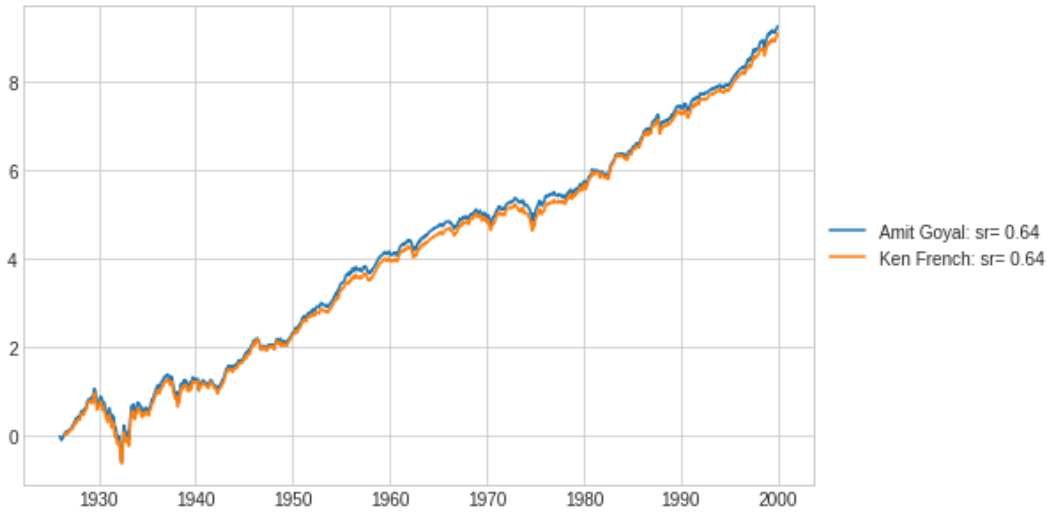
[6]:
```
ret = load_kf_returns(filename="F-F_Research_Data_Factors")["Monthly"][:"1999"]
```

```
INFO:skfin.datasets:logging from cache directory: data/F-F_Research_Data_Factors
```

[7]:
```
corr_ = df[["CRSP_SPvw"]].corrwith(
    ret.assign(Mkt=lambda x: x["Mkt-RF"] + x["RF"])["Mkt"]
)["CRSP_SPvw"]
print(f"Correlation data Ken French/Amit Goyal:{corr_:.2f}")
```

```
Correlation data Ken French/Amit Goyal:0.99
```

[8]:
```
line(
    {
        "Amit Goyal": df["CRSP_SPvw"],
        "Ken French": ret.assign(Mkt=lambda x: x["Mkt-RF"] + x["RF"])["Mkt"] / 100,
    },
    cumsum=True,
)
```



16.3 Timing backtest

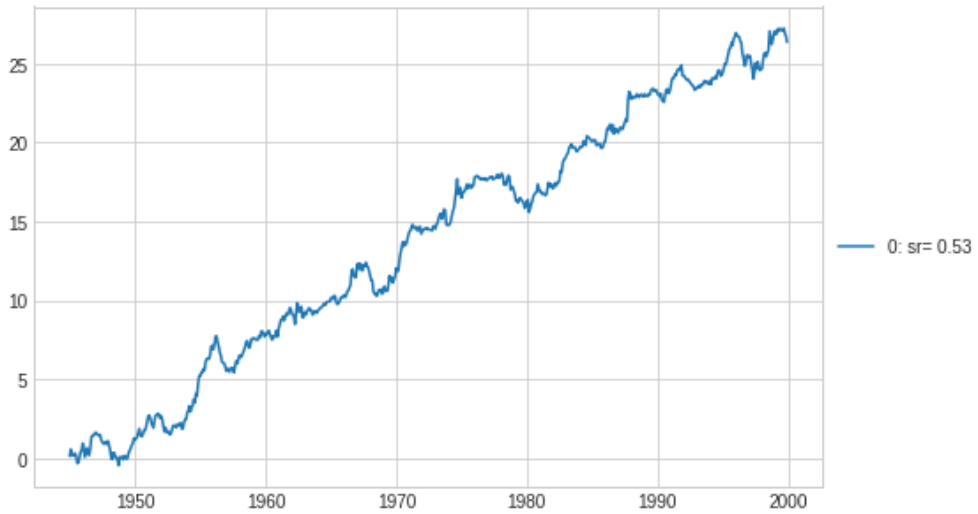
[9]:
```
from skfin.estimators import Ridge, RidgeCV
from skfin.mv_estimators import TimingMeanVariance
from sklearn.model_selection import TimeSeriesSplit
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
```

[10]:
```
start_date = "1945-01-01"
test_size = 1
params = dict(max_train_size=36, test_size=test_size, gap=0)
params["n_splits"] = 1 + len(ret[:"1999"].loc[start_date:]) // test_size

cv = TimeSeriesSplit(**params)
```

[11]:
```
cols = [
    "D12",
    "E12",
    "b/m",
    "tbl",
```

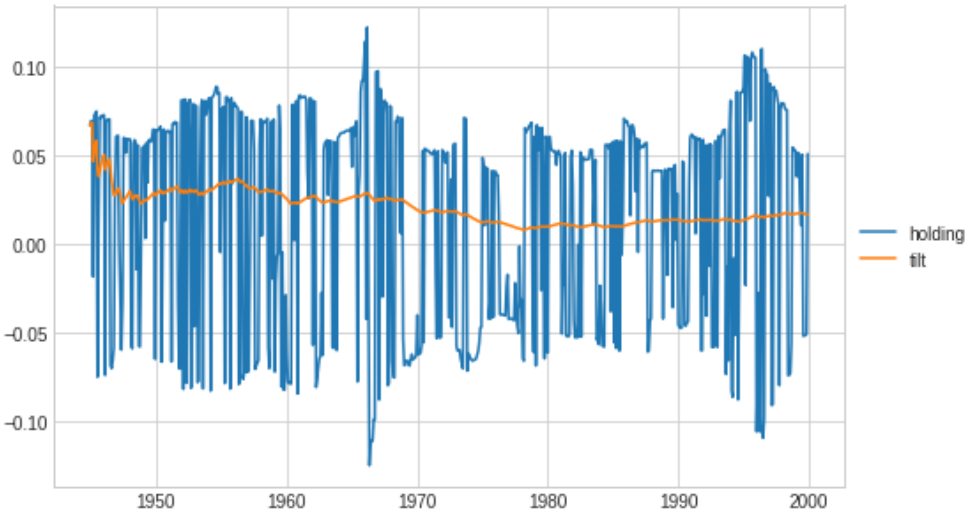
```
    "AAA",
    "BAA",
    "lty",
    "ntis",
    "Rfree",
    "infl",
    "ltr",
    "corpr",
    "svar",
    "csp",
]
ret_ = ret["Mkt-RF"]
target = ret_
features = df.loc[ret.index, cols].fillna(0)
```

[12]:
```
m = make_pipeline(
    StandardScaler(), Ridge(), TimingMeanVariance(a_min=-0.25, a_max=0.25)
)

_h = []
for train, test in cv.split(ret):
    m.fit(features.iloc[train], target.iloc[train])
    _h += [m.predict(features.iloc[test])]

idx = ret.index[np.concatenate([test for _, test in cv.split(ret)])]
h = pd.Series(np.concatenate(_h), index=idx)
pnl = h.shift(1).mul(ret_).dropna()
line(pnl, cumsum=True)
```



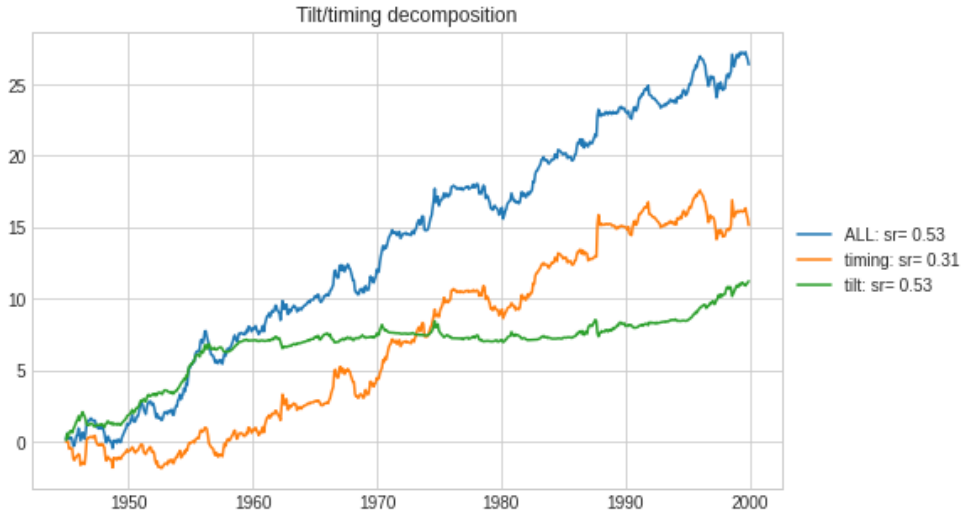
We can plot the holdings and in this case, we see that the positions vary significantly and that there is a significant positive `tilt` (defined as the exponential average over the positions with a 252-day halflife).

[13]:
```
line({"holding": h, "tilt": h.ewm(halflife=252).mean()})
```



Decomposing the pnl attributed to the `tilt` and the `timing` (defined as the difference between the positions and the `tilt`), we see that both contribute – although the `timing` pnl has a lower sharpe ratio.

[14]:
```
line(
    {
        "ALL": pnl,
        "tilt": h.ewm(halflife=12).mean().shift(1).mul(ret_).dropna(),
        "timing": h.sub(h.ewm(halflife=12).mean()).shift(1).mul(ret_).dropna(),
    },
    cumsum=True, title='Tilt/timing decomposition'
)
```



In what follows, we use the `Backtester` clas with the timing pipeline.

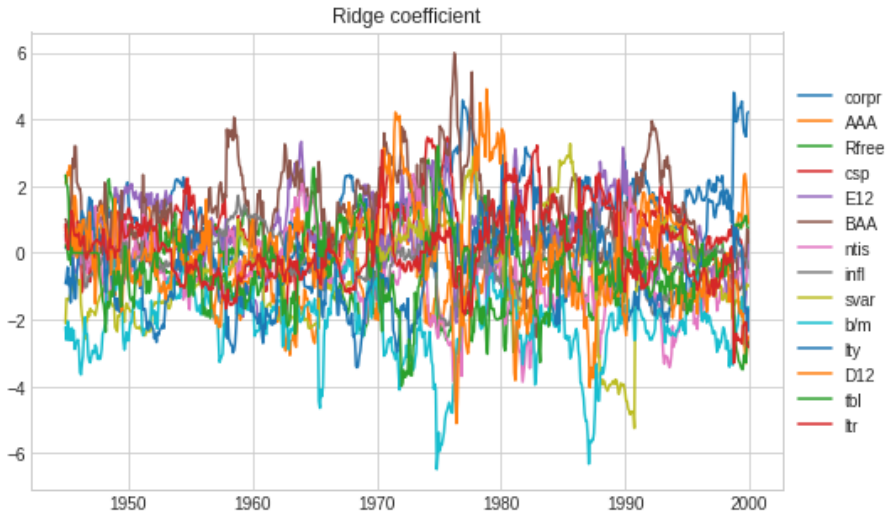
[15]:
```
from skfin.backtesting import Backtester
```

```
estimator = make_pipeline(
    StandardScaler(), Ridge(), TimingMeanVariance(a_min=-0.25, a_max=0.25)
)

m = Backtester(estimator=estimator)
m.compute_holdings(features, target).compute_pnl(ret_)

np.allclose(h, m.h_), np.allclose(pnl, m.pnl_)
```

[15]: (True, True)

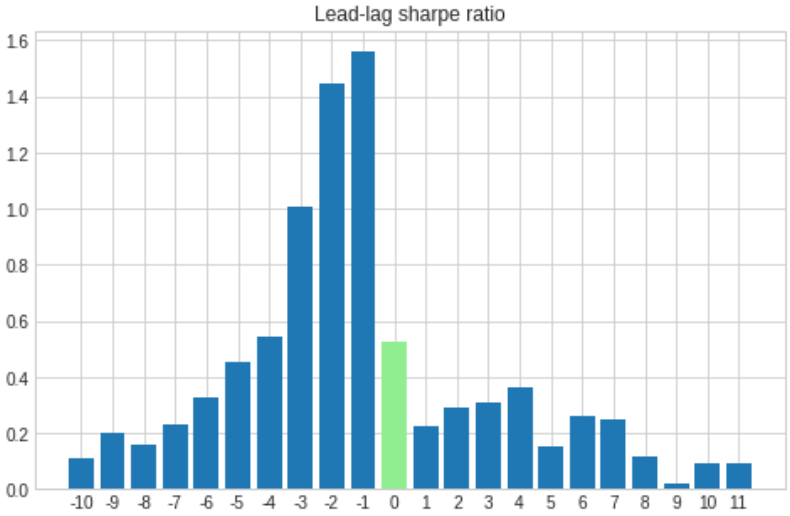
16.4 Other timing backtest statistics

[16]:
```
coef = pd.DataFrame(
    [m_.steps[1][1].coef_ for m_ in m.estimators_], columns=cols, index=m.h_.index
)
line(coef, title="Ridge coefficient")
```

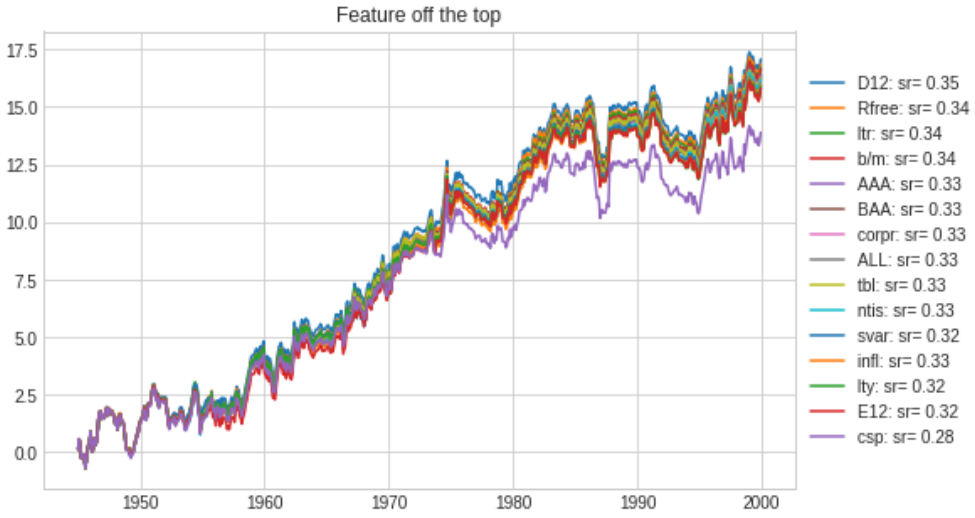


[17]:
```
from skfin.metrics import sharpe_ratio
```

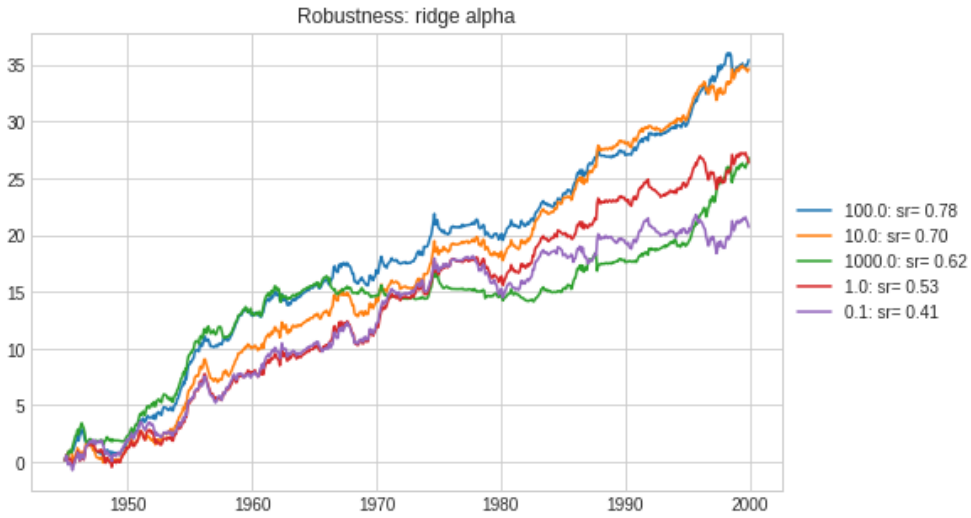
[18]:
```
sr = {i: m.h_.shift(1 + i).mul(ret_).pipe(sharpe_ratio) for i in range(-10, 12)}
bar(sr, baseline=0, sort=False, title="Lead-lag sharpe ratio")
```



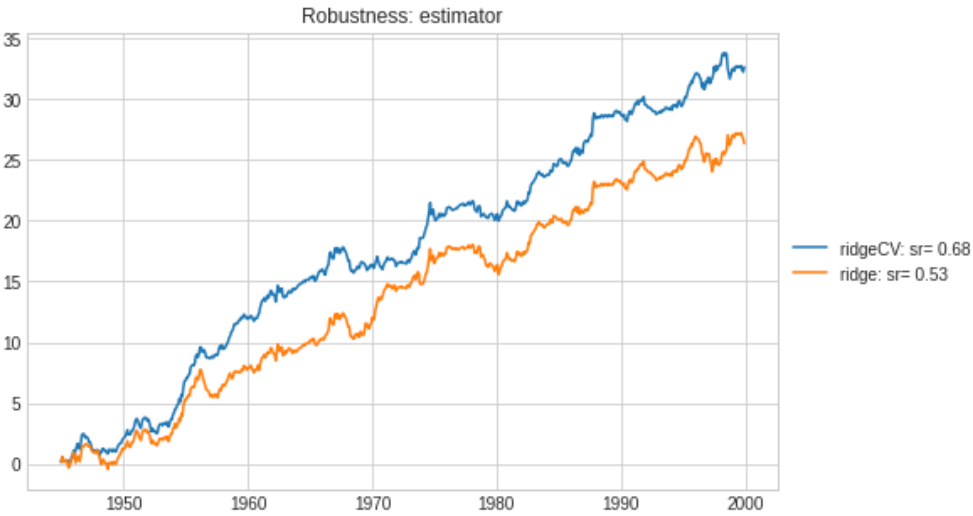
[19]:
```
pnls_ = {}
for c in cols + ["ALL"]:
    features_ = df.loc[ret.index].drop(c, axis=1, errors="ignore").fillna(0)
    pnls_[c] = Backtester(estimator=estimator).train(features_, target, ret=ret_)
line(pnls_, cumsum=True, title="Feature off the top")
```



[20]:
```
pnls_ = {}
for alpha in [0.1, 1, 10, 100, 1000]:
    estimator_ = make_pipeline(
        StandardScaler(),
        Ridge(alpha=alpha),
        TimingMeanVariance(a_min=-0.25, a_max=0.25),
    )
    pnls_[alpha] = Backtester(estimator=estimator_).train(features, target, ret=ret_)
line(pnls_, cumsum=True, title="Robustness: ridge alpha")
```

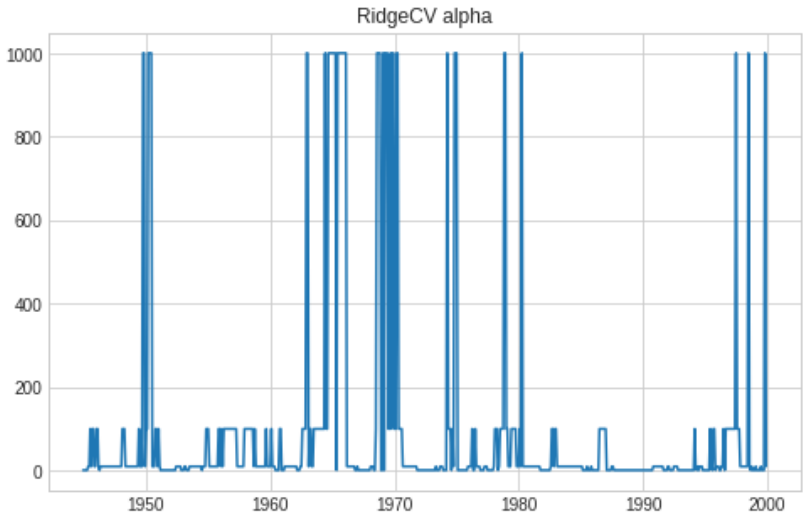


[21]:
```
estimator_ = make_pipeline(
    StandardScaler(),
    RidgeCV(alphas=[1, 10, 100, 1000]),
    TimingMeanVariance(a_min=-0.25, a_max=0.25),
)

m_ = Backtester(estimator=estimator_)
m_.compute_holdings(features, target).compute_pnl(ret_)
line({"ridge": m.pnl_, "ridgeCV": m_.pnl_}, cumsum=True, title="Robustness: estimator")
```



The following graph shows the regularization paramter `alpha` estimated by cross-validation by the `RidgeCV` estimator.

[22]:
```
alpha = pd.Series([m_.steps[1][1].alpha_ for m_ in m_.estimators_], index=m_.h_.index)
line(alpha, legend=False, title="RidgeCV alpha")
```


RidgeCV alpha
1000
800
600
400
200
0
1950
1960
1970
1980
1990
2000

Chapter 17

Macroeconomic event trading

In this section, we describe trading around specific scheduled macroeconomic events.

17.1 Macroeconomic events

JOURNAL OF FINANCIAL AND QUANTITATIVE ANALYSIS Vol. 48, No. 2, Apr. 2013, pp. 343–375

doi:10.1017/S002210901300015X

How Much Do Investors Care About Macroeconomic Risk? Evidence from Scheduled Economic Announcements

Pavel Savor and Mungo Wilson*

Three statements in Savor and Wilson (2013):

- U.S. stock market returns are significantly higher on days when important macroeconomic news is scheduled to be announced.
- Inversely, the returns on T-Bills is significantly lower on these days.
- The announcement surprises do not seem to predict the returns.

These facts are consistent with risk-averse investors demanding higher return for holding event risk.

Data:

- 157 prescheduled CPI announcements from Jan. 1958 to Jan. 1971 and 467 for the PPI from Feb. 1971 to Dec. 2009.
- 621 employment announcements from Jan. 1958 to Dec. 2009.
- FOMC interest rate announcements start in Jan. 1978 and end in Dec. 2009.

Any unscheduled announcements are excluded, leaving us with 279 FOMC observations.

TABLE 1

Summary Statistics for Daily Stock Market Excess Returns

Table 1 presents the distribution of stock market excess returns on announcement days and nonannouncement days. Announcement days are those trading days when CPI/PPI (CPI before 1971 and PPI afterward) numbers, employment numbers, and FOMC interest rate decisions are scheduled for release. The sample covers the period 1958–2009. Market excess returns are computed as the difference between the CRSP value-weighted market return and the risk-free rate. The daily risk-free rate is derived from the 1-month risk-free rate provided by CRSP. *t*-statistics are given in square brackets. All numbers are expressed in basis points, and the numbers in bold are of special interest.

	Panel A. All Obs.			*Panel B. Excl. Outliers (< 1 percentile or > 99 percentile)*		
	Ann.	Nonann.	Diff.	Ann.	Nonann.	Diff.
Mean	11.4 [4.41]	1.1 [1.29]	**10.3 [3.77]**	11.7 [5.39]	1.3 [1.88]	**10.4 [4.55]**
1st percentile	−258.2	−257.7	−1.2	−208.4	−203.2	−5.1
25th percentile	−34.7	−40.8	6.1	−33.3	−39.3	6.0
Median	13.5	4.3	9.2	13.5	4.3	9.2
75th percentile	59.0	45.2	13.7	57.8	44.3	13.5
99th percentile	292.6	242.4	50.3	233.4	191.3	42.1
Std. dev.	98.6	94.6		81.8	75.4	
Skewness	−0.6	−0.6		0.0	−0.2	
Kurtosis	8.5	19.7		1.1	0.8	
N	1,450	11,641		1,420	11,407	

TABLE 2

Regression Analysis: Daily Stock Market Excess Returns

Table 2 presents the results of OLS regressions of daily stock market excess returns on an announcement-day dummy variable and various other controls. Ann. day is a dummy variable equaling 1 if day *t* is an announcement day, and 0 otherwise. Market excess returns (MKTRF) are computed as the difference between the CRSP value-weighted market return and the risk-free rate (expressed in basis points). Monday–Thursday are dummy variables for the corresponding days of the week. *t*-statistics are calculated using Newey-West (1987) standard errors (with 5 lags) and are given in square brackets. The numbers in bold are of special interest.

	Panel A. All Obs.			*Panel B. Excl. Outliers (< 1 percentile or > 99 percentile)*		
Variable	1	2	3	1	2	3
Intercept	1.133 [1.24]	−0.064 [−0.07]	3.448 [2.00]	1.324 [1.71]	0.647 [0.83]	4.670 [3.11]
Ann. day	**10.291 [3.55]**	**10.439 [3.61]**	**7.093 [2.69]**	**10.38 [4.18]**	**10.653 [4.34]**	**8.462 [3.41]**
$MKTRF_{t-1}$		0.081 [4.91]	0.081 [4.93]		0.094 [8.75]	0.094 [8.74]
$(MKTRF_{t-1})^2$		0.0001 [1.61]	0.0001 [1.59]		0.0000 [0.47]	0.0000 [0.45]
Monday			−13.605 [−5.05]			−11.228 [−5.31]
Tuesday			−2.611 [−1.06]			−5.662 [−2.71]
Wednesday			2.151 [0.89]			1.129 [0.56]
Thursday			−2.558 [−1.07]			−3.519 [−1.71]
N	13,091	13,090	13,090	12,827	12,826	12,826
R^2 (%)	0.1	0.9	1.2	0.2	1.4	1.7

The return (resp. standard deviation) is 11.4bps (resp. 98.6 bps) on announcement days and 1.1bps (resp. 94.6 bps) on non-announcement days, so that the annualized Sharpe ratios very different, 1.8 and .18 respectively. The next two figures show similar statements (with inverse sign) for the risk-free asset (= T-Bills) – impacted by precautionary saving demand.

TABLE 3

Summary Statistics for Daily 30-Day T-Bill Returns

Table 3 presents the distribution of daily 30-day T-bill returns on announcement days and nonannouncement days. Announcement days are those trading days when CPI/PPI (CPI before 1971 and PPI afterward) numbers, employment numbers, and FOMC interest rate decisions are scheduled for release. The sample covers the period 1961–2009. The 30-day T-bill returns are defined as the return of the T-bill issue whose length of maturity is closest to 30 days (daily T-bill quotes are obtained from CRSP starting in June 1961). *t*-statistics are given in square brackets. All numbers are expressed in basis points, and the numbers in bold are of special interest.

	Panel A. All Obs.			*Panel B. Excl. Outliers (< 1 percentile or >99 percentile)*		
	Ann.	Nonann.	Diff.	Ann.	Nonann.	Diff.
Mean	1.5 [34.69]	2.3 [94.84]	**−0.7 [−14.49]**	1.5 [51.29]	2.2 [123.03]	**−0.7 [−21.40]**
1st percentile	−0.9	−0.7	−0.2	−0.3	−0.2	−0.2
25th percentile	0.8	1.0	−0.2	0.8	1.0	−0.2
Median	1.3	1.6	−0.3	1.3	1.6	−0.3
75th percentile	1.9	2.9	−1.0	1.9	2.9	−1.0
99th percentile	6.8	10.2	−3.5	5.5	8.5	−3.0
Std. dev.	1.6	2.5		1.1	1.8	
Skewness	8.5	0.2		1.5	1.5	
Kurtosis	163.1	55.3		3.8	2.3	
N	1,370	10,711		1,342	10,495	

TABLE 4

Regression Analysis: Daily 30-Day T-Bill Returns

Table 4 presents the results of OLS regressions of daily 30-day T-bill returns on an announcement-day dummy variable and various other controls. Ann. day is a dummy variable equaling 1 if day *t* is an announcement day, and 0 otherwise. The 30-day T-bill returns (TBILL) are defined as the return of the T-bill issue whose length of maturity is closest to 30 days (expressed in basis points). Monday–Thursday are dummy variables for the corresponding days of the week. *t*-statistics are calculated using Newey-West (1987) standard errors (with 5 lags) and are given in square brackets. The numbers in bold are of special interest.

	Panel A. All Obs.			*Panel B. Excl. Outliers (< 1 percentile or >99 percentile)*		
Variable	1	2	3	1	2	3
Intercept	2.255 [69.77]	1.936 [37.28]	1.057 [13.71]	2.200 [84.54]	1.902 [55.12]	1.092 [24.45]
Ann. day	**−0.726 [−15.42]**	**−0.707 [−15.08]**	**−0.106 [−2.46]**	**−0.724 [−22.70]**	**−0.694 [−22.44]**	**−0.141 [−4.93]**
$TBILL_{t-1}$		0.131 [6.35]	0.214 [6.37]		0.119 [10.6]	0.209 [9.04]
$(TBILL_{t-1})^2$		−0.0053 [−0.32]	−0.0032 [−0.15]		0.0012 [1.62]	0.0017 [0.97]
Monday			2.846 [31.89]			2.647 [57.48]
Tuesday			−0.124 [−1.18]			−0.214 [−2.83]
Wednesday			0.250 [5.01]			0.210 [7.32]
Thursday			0.311 [5.89]			0.280 [10.09]
N	12,080	11,941	11,941	11,836	11,710	11,710
R^2 (%)	0.9	2.6	22.5	1.7	4.5	36.0

	Panel A. Stock Market Excess Returns				*Panel B. T-Bill Returns*			
Varisble	Model Expect.		SPF Expect.		Model Expect.		SPF Expect.	
Ann. day	**7.607 [2.60]**	**8.076 [2.75]**	**11.278 [2.51]**	**9.638 [2.16]**	**−0.150 [−3.45]**	**−0.156 [−3.61]**	**−0.145 [−2.31]**	**−0.144 [−2.28]**
Inflation Surprise	0.015 [0.82]		0.007 [0.37]		−0.000 [−0.23]		0.000 [0.63]	
Unemployment Surprise	0.088 [0.36]		0.151 [0.62]		0.000 [0.11]		0.003 [0.54]	
FED Funds Surprise	−1.472 [−0.77]		−1.347 [−0.74]		−0.039 [−2.16]		−0.039 [−2.47]	
Infl. Sur. × Expansion		0.013 [0.88]		0.012 [0.55]		0.000 [0.33]		0.000 [−0.31]
Unemp. Sur. × Expansion		0.310 [1.28]		−0.167 [−0.86]		−0.002 [−0.56]		−0.001 [−0.12]
FED Sur. × Expansion		−1.915 [−1.20]		−1.869 [−1.19]		−0.052 [−2.82]		−0.049 [−2.76]
Infl. Sur. × Recession		0.018 [0.43]		−0.002 [−0.07]		−0.000 [−0.42]		0.000 [1.09]
Unemp. Sur. × Recession		−0.768 [−1.03]		1.005 [1.61]		0.013 [1.15]		0.011 [1.67]
FED Sur. × Recession		0.161 [0.02]		0.439 [0.07]		0.008 [0.3]		−0.003 [−0.16]
N	12,987	12,987	7,008	7,008	11,841	11,841	6,864	6,864
R^2 (%)	1.2	1.2	0.3	0.4	22.6	22.6	14.7	14.7

Other events from the follow-up paper: Savor and Wilson (2014): "Asset pricing: a Tale of Two Days,"

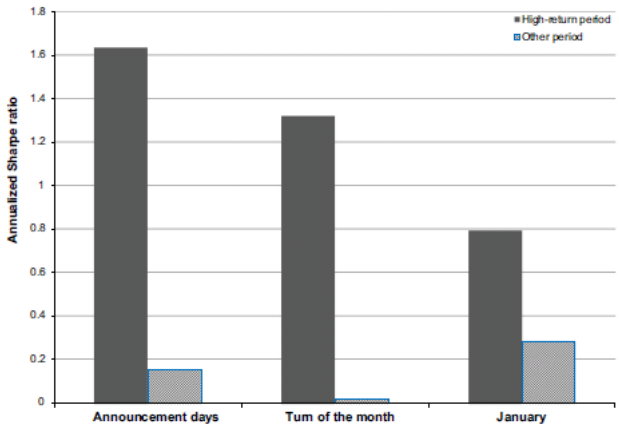


Fig. 9. Annualized market Sharpe ratio across different periods. This figure plots the annualized Sharpe ratio for the value-weighted index of all NYSE, Amex, and Nasdaq stocks separately for announcement days (days on which inflation, employment, or Federal Open Market Committee interest rate decisions are scheduled to be announced) versus all other days; the turn of the month (the last trading day of a month and the first four trading days of the following month) versus all other days; and January versus other months.

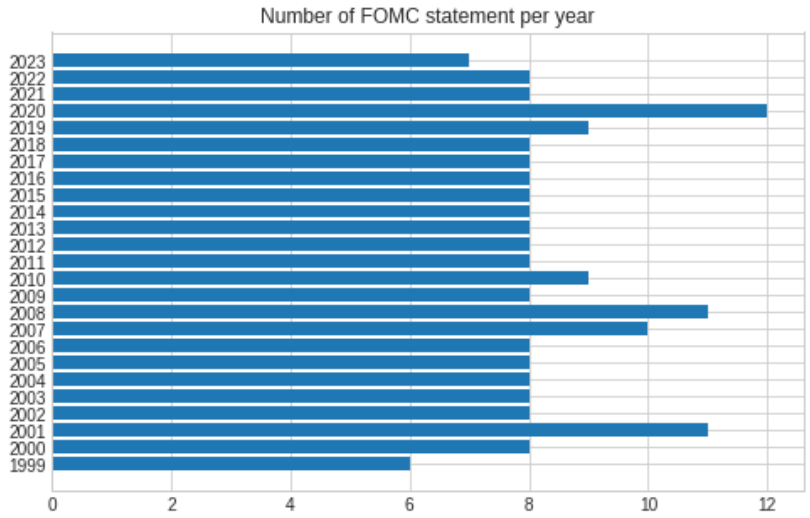
17.2 FOMC dates

[10]:
```
from skfin.datasets import load_fomc_statements
from skfin.plot import bar, line
from skfin.text import show_text
```

[11]:
```
statements = load_fomc_statements(force_reload=False)
```

```
INFO:skfin.datasets:logging from cache file: data/fomc_statements.parquet
```

[12]:
```
bar(
    statements.groupby(lambda x: x.year)["text"].count(),
    sort=False,
    horizontal=True,
    title="Number of FOMC statement per year",
)
```



17.3 Returns on statement days

[13]:
```
import numpy as np
import pandas as pd
from pandas.tseries.offsets import BDay
from skfin.datasets import load_kf_returns

ret = load_kf_returns(filename="F-F_Research_Data_Factors_daily")["Daily"]
```

```
INFO:skfin.datasets:logging from cache directory:
data/F-F_Research_Data_Factors_daily
```

[14]:
```
ret_norm = ret.div(ret.ewm(252).std())[statements.sort_index().index[0] :].asfreq("B")
```

Some FOMC meetings on Sundays or when the markets are closed:

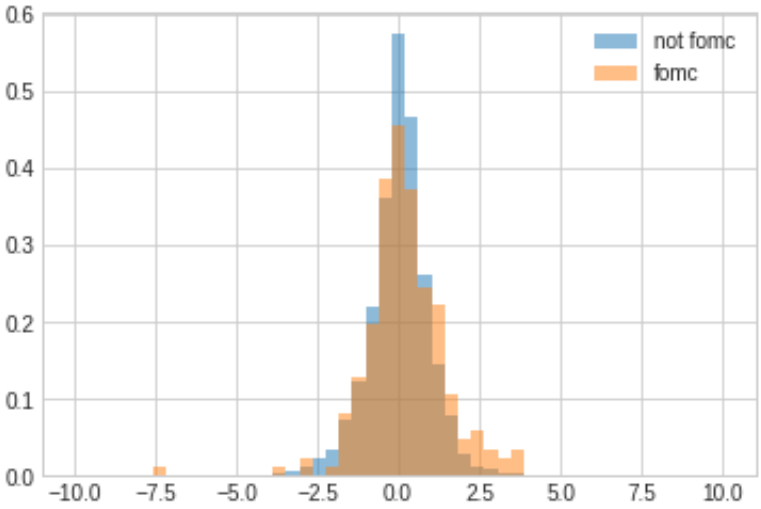
- '2008-01-21': Luther King Jr Day
- '2010-05-09': Sunday
- '2020-03-15': (note the release time!!!)

[15]:
```
special_days = ["2008-01-22", "2010-05-09", "2020-03-15"]
idx0 = pd.to_datetime(pd.Index(special_days))
idx = statements.index.difference(idx0).union(idx0 + BDay(1))
```

[16]:
```
ret_fomc = pd.concat(
    {
        "fomc": ret_norm.loc[ret_norm.index.intersection(idx)],
        "not fomc": ret_norm.loc[ret_norm.index.difference(idx)],
    },
    axis=1,
)
```

[17]:
```
bins = np.linspace(-10, 10, 50)
plt.hist(
    ret_fomc["not fomc"]["Mkt-RF"].dropna(),
    bins,
    density=True,
```

```
    alpha=0.5,
    label="not fomc",
)
plt.hist(
    ret_fomc["fomc"]["Mkt-RF"].dropna(), bins, density=True, alpha=0.5, label="fomc"
)
plt.legend(loc="upper right")
plt.show()
```



[18]:
```
line(
    ret_fomc.xs("Mkt-RF", axis=1, level=1)
    .fillna(0)
    .assign(ALL=lambda x: x.sum(axis=1))["1996":],
    cumsum=True,
    title="Cumulative return from being long the market",
)
```

[19]:
```
ret_fomc.std().unstack().round(2)
```

[19]:
```
           Mkt-RF   SMB   HML    RF
fomc         1.26  1.07  1.25  2.64
not fomc     0.98  0.99  1.00  2.64
```

[20]:
```
from skfin.metrics import sharpe_ratio
```

[21]:
```
ret_fomc.pipe(sharpe_ratio).unstack().round(2)
```

[21]:
```
           Mkt-RF   SMB   HML     RF
fomc         2.74  1.80 -0.38  13.15
not fomc     0.26  0.12 -0.02  12.73
```

[22]:
```
ret_fomc.clip(lower=-3, upper=3).pipe(sharpe_ratio).unstack().round(2)
```

[22]:
```
           Mkt-RF   SMB   HML     RF
fomc         3.21  1.76 -0.55  18.54
not fomc     0.33  0.14 -0.10  17.98
```

[23]:
```
ret_fomc.drop(pd.to_datetime("2020-03-16")).asfreq("B").pipe(
    sharpe_ratio
).unstack().round(2)
```

[23]:
```
           Mkt-RF   SMB   HML     RF
fomc         3.51  1.84 -0.32  13.10
not fomc     0.26  0.12 -0.02  12.73
```

Chapter 18

Text processing

In this section, we introduce several techniques to analyse a corpus of documents. This is done in the context of the statements of the Federal Open Market Committee (FOMC).

18.1 Loading the FOMC statements

[2]:
```
from skfin.datasets import load_fomc_statements
from skfin.text import show_text

statements = load_fomc_statements(force_reload=False)
```

```
INFO:skfin.datasets:logging from cache file: data/fomc_statements.parquet
```

[3]:
```
show_text(statements)
```

```
<IPython.core.display.HTML object>
```

[4]:
```
special_days = ["2008-01-22", "2010-05-09", "2020-03-15"]
```

[5]:
```
show_text(statements.loc[special_days])
```

```
<IPython.core.display.HTML object>
```

18.2 TFIDF vectorization

In order to extract features from text, the simplest way is to count words. In `scikit-learn`, this is done with the function `CountVectorizer`. A slightly more advanced feature is to select words based on a `TFIDF` score, defined as the product of the term frequency (`TF`) and the inverse document frequency (`IDF`). More precisely, the `TFIDF` score trades off: - the terms that are frequent and therefore important in a corpus: - the terms that appear in almost all documents and therefore are not helping to discriminate across documents.

In `TfidfVectorizer`, terms can be filtered additionally with: - a `stop word` list - min and max document frequencies or counts - some token pattern (e.g. that eliminates the short tokens).

[6]:
```
from sklearn.decomposition import NMF, PCA
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
```

[7]:
```
vectorizer = TfidfVectorizer(
    stop_words="english",
    min_df=5,
    max_df=0.8,
    ngram_range=(1, 3),
    token_pattern=r"\b[a-zA-Z]{3,}\b",
)
X = vectorizer.fit_transform(statements["text"].values)
```

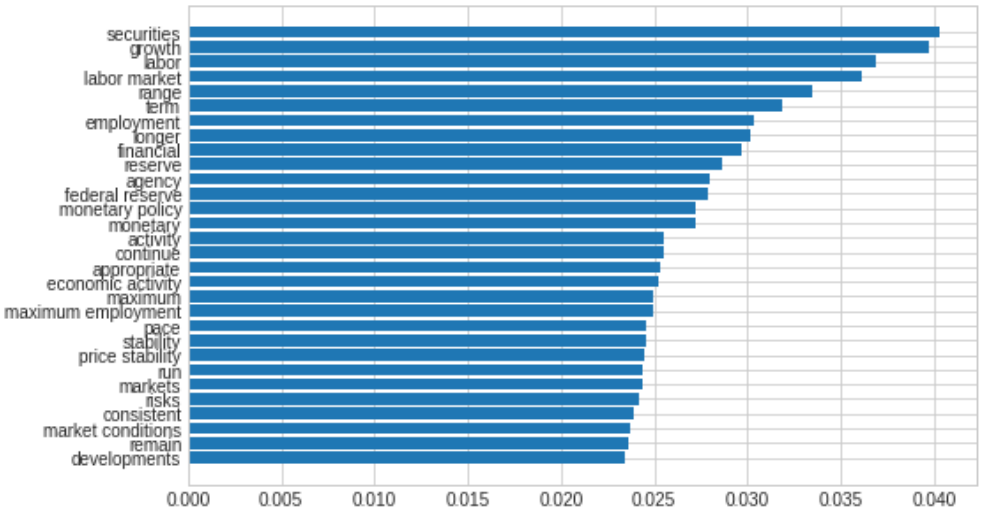
[8]:
```
cols = vectorizer.get_feature_names_out()
print(len(cols))
list(cols)[:10]
```

```
4177
```

[8]:
```
['abate',
 'abating',
 'abroad',
 'accommodation',
 'accommodation balanced',
 'accommodation balanced approach',
 'accommodation economic',
 'accommodation economic activity',
 'accommodation economic growth',
 'accommodation removed']
```

Here are the most frequent tokens

[9]:
```
df = pd.DataFrame(X.toarray(), index=statements["text"].index, columns=cols)
bar(df.mean().sort_values(ascending=False).head(30), horizontal=True)
```

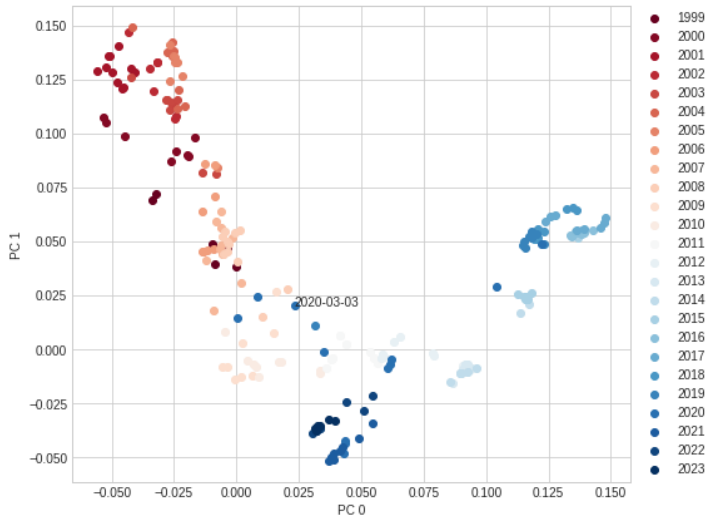


18.3 Principal component exploration

To describe the matrix of tdidf scores, we first perform a simple principal component analysis (PCA) with two modes.

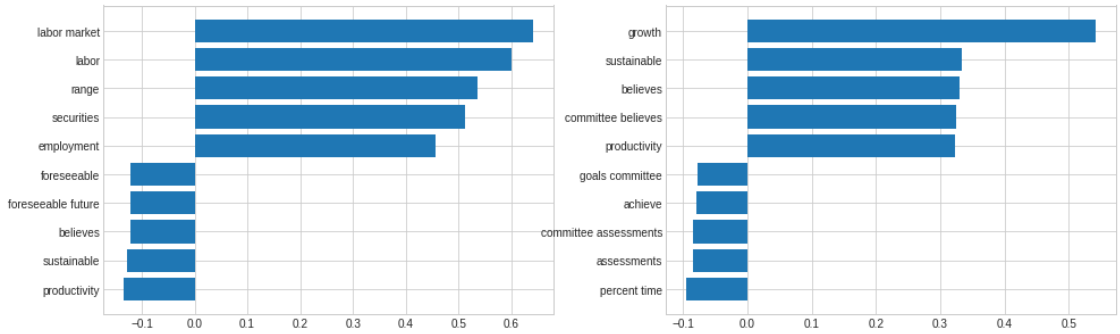
[10]:
```
m = PCA(n_components=2).fit(np.log1p(X.toarray().T))
df = pd.DataFrame(m.components_.T, index=statements.index)
```

[11]:
```
m = PCA(n_components=2).fit(np.log1p(X.toarray().T))
df = pd.DataFrame(m.components_.T, index=statements.index)

fig, ax = plt.subplots(1, 1, figsize=(8, 7))
years = [str(y) for y in df.index.year.unique()]
colors = cm.RdBu(np.linspace(0, 1, len(years)))
for i, y in enumerate(years):
    ax.scatter(x=df.loc[y][0], y=df.loc[y][1], color=colors[i])
ax.legend(years, loc="center left", bbox_to_anchor=(1, 0.5))
ax.set_xlabel("PC 0")
ax.set_ylabel("PC 1")

d = "2020-03-03"
ax.text(x=df.loc[d][0], y=df.loc[d][1], s=d);
```



These two modes can be related to labor market and growth.

[12]:
```
func = lambda x: pd.concat([x.nlargest(), x.sort_values(ascending=False).tail(5)])
W = pd.DataFrame(m.transform(np.log1p(X.toarray().T)), index=cols)

fig, ax = plt.subplots(1, 2, figsize=(16, 5))
plt.subplots_adjust(wspace=0.25)
for i in [0, 1]:
    bar(W[i].pipe(func), horizontal=True, ax=ax[i])
```



18.4 Unsupervised learning: document clustering

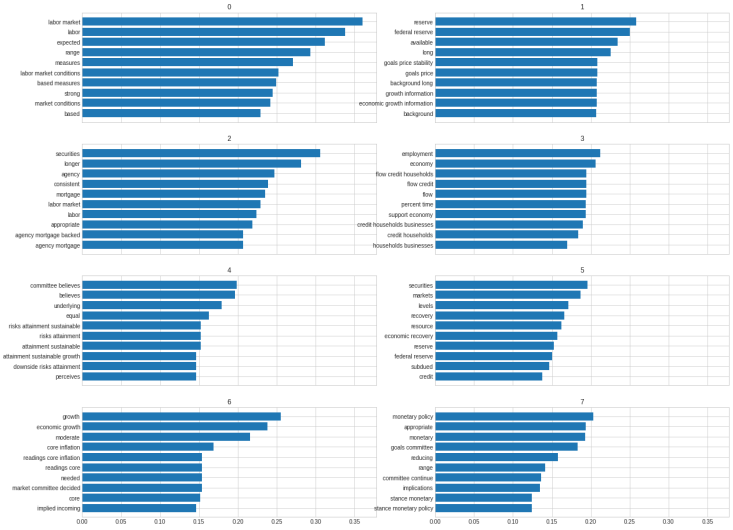
It is often information to group tokens into topics that explain differences across documents. A powerful algorithm is the non-negative matrix factorisation (`NMF`): for a non-negative matrix X (such as the one with tfidf scores), `NMF` finds two other non-negative matrices such that:

$$X \approx WH.$$

The number of topics (called `n_components` in the `scikit-learn` implementation) determines the number of columns in W and the number of rows in H.

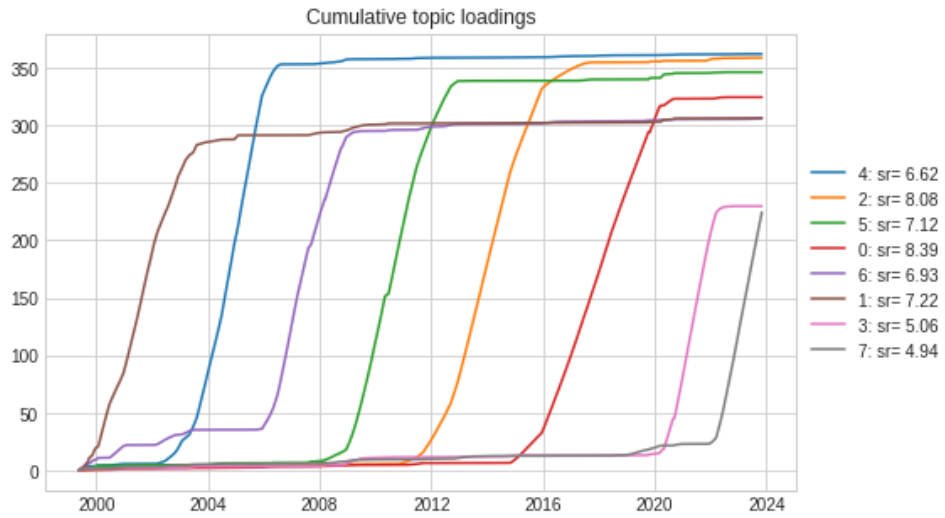
[13]:
```
n_components = 8
m = NMF(
    n_components=n_components,
    init="nndsvd",
    solver="cd",
    beta_loss="frobenius",
    random_state=1,
    alpha_W=0,
    l1_ratio=0,
    max_iter=500,
).fit(X)
```

[14]:
```
fig, ax = plt.subplots(4, 2, figsize=(20, 16), sharex=True)
ax = ax.ravel()
for i in range(8):
    bar(
        pd.Series(m.components_[i, :], cols).sort_values(ascending=False).head(10),
        horizontal=True,
        ax=ax[i],
        title=i,
    )
```



Are these topics interesting? This is a matter of interpretation, but at least, the graph below shows that these topics capture a strong element of time-clustering which makes it a bit less useful.

[15]:
```
W = pd.DataFrame(m.transform(X), index=df.index)
line(W.resample("B").last().ffill(), cumsum=True, title="Cumulative topic loadings")
```



18.5 Supervised learning: TFIDF + Elastic net

In this section, we use the corpus of FOMC statements for supervised learning. More precisely, we match the text of the statements to the decision of the committee to raise rates, decrease rates or do nothing.

In practice, this implemented by using `scikit-learn pipelines` and chaining the `TfidfVectorizer` with a logistic regression.

[16]:
```
import numpy as np
from skfin.datasets import load_fomc_change_date

fomc_change_up, fomc_change_dw = load_fomc_change_date()
```

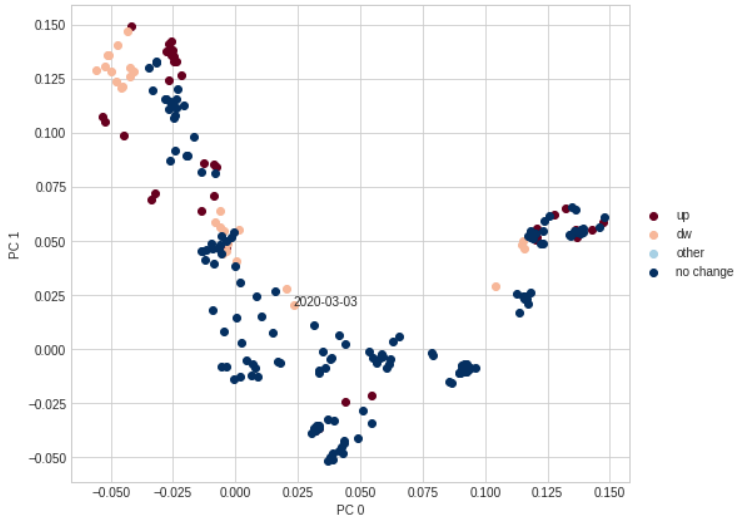
[17]:
```
fomc_change_up, fomc_change_dw
```

[17]:
```
(DatetimeIndex(['1999-06-30', '1999-08-24', '1999-11-16', '2000-02-02',
                '2000-03-21', '2000-05-16', '2004-06-30', '2004-08-10',
                '2004-09-21', '2004-11-10', '2004-12-14', '2005-02-02',
                '2005-03-22', '2005-05-03', '2005-06-30', '2005-08-09',
                '2005-09-20', '2005-11-01', '2005-12-13', '2006-01-31',
                '2006-03-28', '2006-05-10', '2006-06-29', '2015-12-16',
                '2016-12-14', '2017-03-15', '2017-06-14', '2017-12-13',
                '2018-03-21', '2018-06-13', '2018-09-26', '2018-12-19',
                '2022-03-16', '2022-05-04', '2022-06-15', '2022-07-27'],
               dtype='datetime64[ns]', freq=None),
 DatetimeIndex(['2001-01-03', '2001-01-31', '2001-03-20', '2001-04-18',
                '2001-05-15', '2001-06-27', '2001-08-21', '2001-09-17',
                '2001-10-02', '2001-11-06', '2001-12-11', '2002-11-06',
                '2003-06-25', '2007-09-18', '2007-10-31', '2007-12-11',
                '2008-01-22', '2008-01-30', '2008-03-18', '2008-04-30',
                '2008-10-08', '2008-10-29', '2008-12-16', '2019-07-31',
                '2019-09-18', '2019-10-30', '2020-03-03', '2020-03-15'],
               dtype='datetime64[ns]', freq=None))
```

[18]:
```
other = {
    "other_dt_change": ["2003-01-09", "2008-03-16", "2011-06-22"],
    "statements_dt_change_other": ["2007-08-16"],
    "qe1": ["2008-11-25", "2008-12-01", "2008-12-16", "2009-03-18"],
    "qe2": ["2010-11-03"],
    "twist": ["2011-09-21", "2012-06-20"],
    "qe3": ["2012-09-13", "2012-12-12", "2013-12-13"],
    "corona": ["2020-03-20"],
}
```

[19]:
```
dates = {
    "up": fomc_change_up,
    "dw": fomc_change_dw,
    "other": [d for c in other.values() for d in c],
}
dates["no change"] = statements.index.difference([d for c in dates.values() for d in c])
```

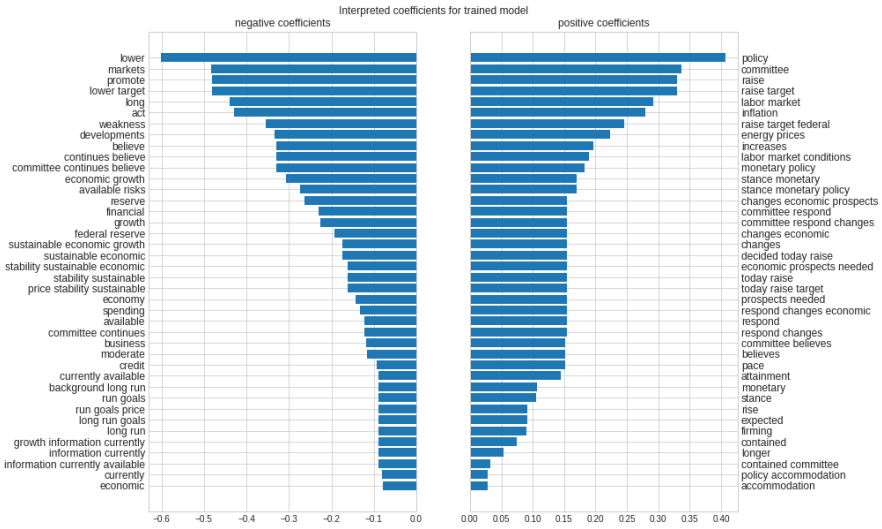
[20]:
```
vectorizer = TfidfVectorizer(
    stop_words="english",
    min_df=5,
    max_df=0.8,
    ngram_range=(1, 3),
    token_pattern=r"\b[a-zA-Z]{3,}\b",
)
X = vectorizer.fit_transform(statements["text"].values)

m = PCA(n_components=2).fit(np.log1p(X.toarray().T))
df = pd.DataFrame(m.components_.T, index=statements.index)

fig, ax = plt.subplots(1, 1, figsize=(8, 7))
colors = cm.RdBu(np.linspace(0, 1, len(dates)))
for i, (k, v) in enumerate(dates.items()):
    ax.scatter(
        x=df.loc[lambda x: x.index.intersection(v)][0],
        y=df.loc[lambda x: x.index.intersection(v)][1],
        color=colors[i],
    )
ax.legend(dates.keys(), loc="center left", bbox_to_anchor=(1, 0.5))
ax.set_xlabel("PC 0")
ax.set_ylabel("PC 1")

d = "2020-03-03"
ax.text(x=df.loc[d][0], y=df.loc[d][1], s=d);
```



[21]:
```
from skfin.text import coefs_plot, show_text
from sklearn.linear_model import ElasticNet, LogisticRegression
```

[22]:
```
est = Pipeline(
    [
        (
            "tfidf",
            TfidfVectorizer(
                vocabulary=None,
                ngram_range=(1, 3),
                max_features=500,
                stop_words="english",
                token_pattern=r"\b[a-zA-Z]{3,}\b",
            ),
        ),
        (
            "reg",
            LogisticRegression(
                C=1, l1_ratio=0.35, penalty="elasticnet", solver="saga", max_iter=500
            ),
        ),
    ]
)
X, y = pd.concat(
    [
        statements.loc[fomc_change_up].assign(change=1),
        statements.loc[fomc_change_dw].assign(change=-1),
    ]
).pipe(lambda df: (df["text"], df["change"]))
est.fit(X, y)
vocab_ = pd.Series(est.named_steps["tfidf"].vocabulary_).sort_values().index
```

[23]:
```
interpret_coef = pd.DataFrame(np.transpose(est.named_steps["reg"].coef_), index=vocab_)
coefs_plot(interpret_coef, title="Interpreted coefficients for trained model")
```

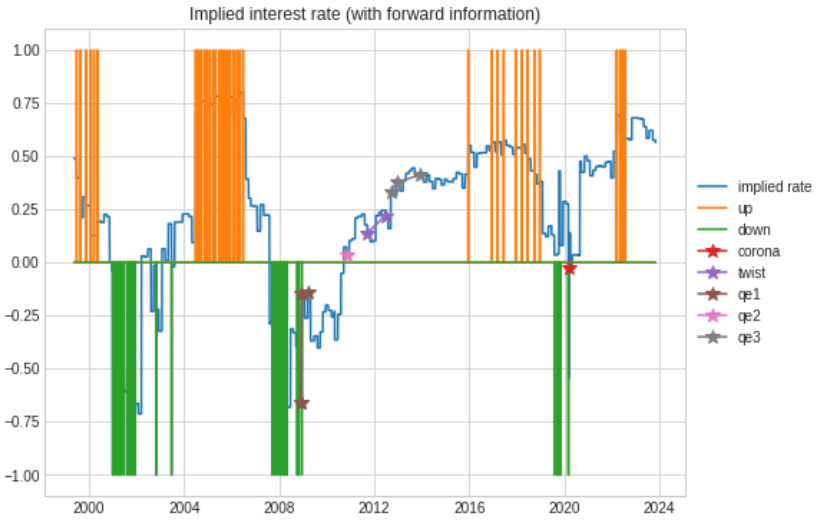


A trick is that using a linear regression (e.g. ElasticNet) instead of a logistic regression is faster and as efficient (even sometimes better)

[24]:
```
est = Pipeline(
    [
        (
            "tfidf",
            TfidfVectorizer(
                vocabulary=None,
                ngram_range=(1, 3),
                max_features=500,
                stop_words="english",
                token_pattern=r"\b[a-zA-Z]{3,}\b",
            ),
        ),
        ("reg", ElasticNet(alpha=0.01)),
    ]
)
X, y = pd.concat(
    [
        statements.loc[fomc_change_up].assign(change=1),
        statements.loc[fomc_change_dw].assign(change=-1),
    ]
).pipe(lambda df: (df["text"], df["change"]))
est.fit(X, y)
vocab_ = pd.Series(est.named_steps["tfidf"].vocabulary_).sort_values().index
```

[25]:
```
interpret_coef = pd.DataFrame(np.transpose(est.named_steps["reg"].coef_), index=vocab_)
coefs_plot(interpret_coef, title="Interpreted coefficients for trained model")
```

[26]:
```
fig, ax = plt.subplots(figsize=(8, 6))
pred_tfidf = (
    pd.Series(est.predict(statements["text"]), index=statements.index)
    .resample("B")
    .last()
    .ffill()
)
line(
    pred_tfidf.rename("implied rate")
    .to_frame()
    .join(
        pd.Series(1, index=fomc_change_up)
        .reindex(pred_tfidf.index)
        .fillna(0)
        .rename("up")
    )
    .join(
        pd.Series(-1, index=fomc_change_dw)
        .reindex(pred_tfidf.index)
        .fillna(0)
        .rename("dw")
    ),
    sort=False,
    ax=ax,
    title="Implied interest rate (with forward information)",
)
cols = ["corona", "twist", "qe1", "qe2", "qe3"]
for c in cols:
    ax.plot(pred_tfidf.loc[other[c]], marker="*", ms=10)
ax.legend(
    ["implied rate", "up", "down"] + cols, loc="center left", bbox_to_anchor=(1, 0.5)
);
```



[27]:
```
lexica = {
    "positive": interpret_coef.squeeze().nlargest(n=10),
    "negative": interpret_coef.squeeze().nsmallest(n=10),
}
```

[28]:
```
idx_ = (
    pd.Series(est.predict(X), index=X.index)
    .sort_values()
    .pipe(lambda x: [x.index[0], x.index[-1]])
)
show_text(statements.loc[idx_], lexica=lexica, n=None)
```

```
<IPython.core.display.HTML object>
```

18.6 UMAP

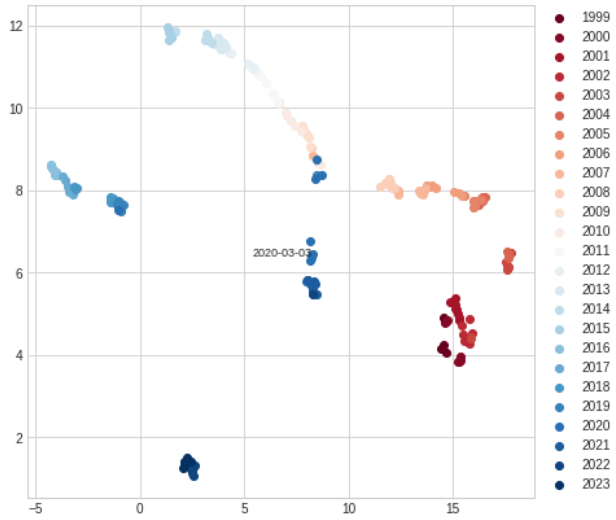
Uniform Manifold Approximation and Projection (UMAP) is a non-linear dimensionality reduction technique. It works by constructing a high-dimensional graph representation of the data and then optimizing a low-dimensional graph to be as structurally similar as possible. UMAP is useful because it effectively preserves both local and global structures in the data, which makes it particularly good for visualizing clusters and relationships in high-dimensional datasets.

- https://umap-learn.readthedocs.io/en/latest/

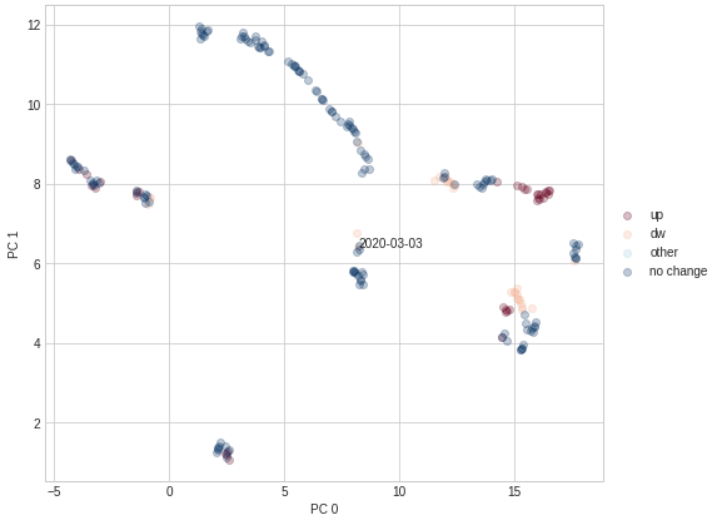
[29]:
```
from umap import UMAP
```

[30]:
```
vectorizer = TfidfVectorizer(
    stop_words="english",
    min_df=5,
    max_df=0.8,
    ngram_range=(1, 3),
    token_pattern=r"\b[a-zA-Z]{3,}\b",
)
X = vectorizer.fit_transform(statements["text"].values)

Xtfm = UMAP().fit_transform(X)
df = pd.DataFrame(Xtfm, index=statements.index)
```

```
classes = df.index.year.unique()

fig, ax = plt.subplots(1, 1, figsize=(7, 7))
colors = cm.RdBu(np.linspace(0, 1, len(classes)))
for i, y in enumerate(classes):
    ax.scatter(df.loc[str(y)][0], df.loc[str(y)][1], color=colors[i])
ax.legend(classes, loc="center left", bbox_to_anchor=(1, 0.5))

d = "2020-03-03"
ax.text(df.loc[d][0], df.loc[d][1], d, fontsize=9, rotation=0, ha="right");
```



[31]:
```
df = pd.DataFrame(Xtfm, index=statements.index)

fig, ax = plt.subplots(1, 1, figsize=(8, 7))
colors = cm.RdBu(np.linspace(0, 1, len(dates)))
for i, (k, v) in enumerate(dates.items()):
    ax.scatter(
        x=df.loc[lambda x: x.index.intersection(v)][0],
        y=df.loc[lambda x: x.index.intersection(v)][1],
        color=colors[i],
        alpha=0.25,
    )
ax.legend(dates.keys(), loc="center left", bbox_to_anchor=(1, 0.5))
ax.set_xlabel("PC 0")
ax.set_ylabel("PC 1")

d = "2020-03-03"
ax.text(x=df.loc[d][0], y=df.loc[d][1], s=d);
```



18.7 Sentence transformer

Sentence Transformers are models fined-tuned from pretrained language models to specifically generate meaningful text representations (as embeddings).

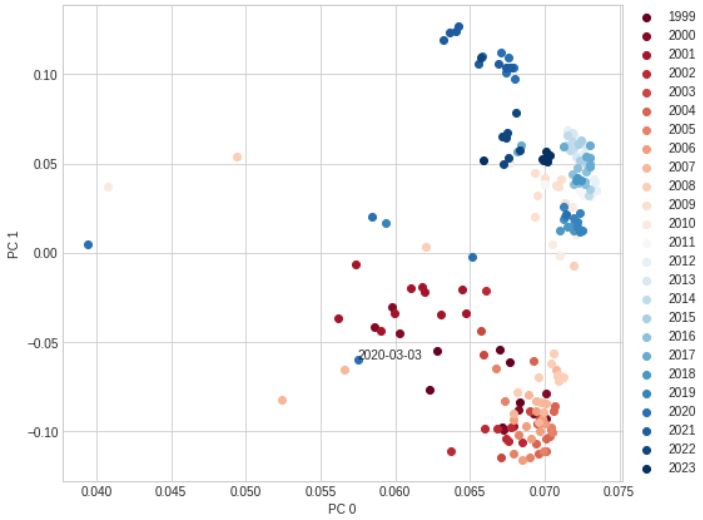
- https://www.sbert.net/

[32]:
```
from sentence_transformers import SentenceTransformer
```

[33]:
```
m = SentenceTransformer("all-distilroberta-v1", device="cpu")
X = m.encode(statements["text"].values, batch_size=2)
```

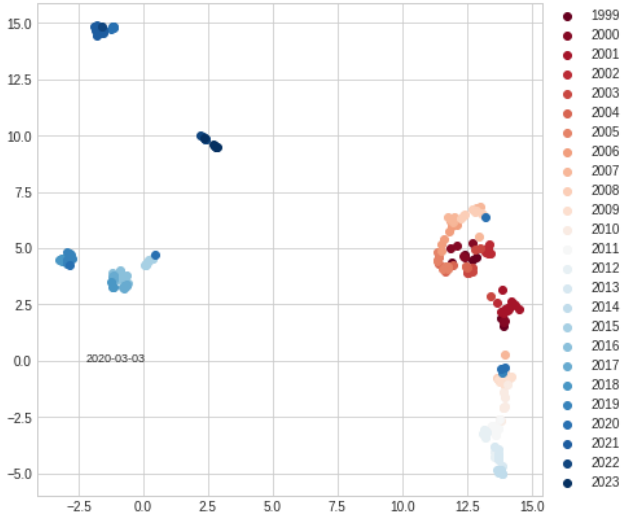
```
INFO:sentence_transformers.SentenceTransformer:Load pretrained
SentenceTransformer: all-distilroberta-v1

Batches:   0%|          | 0/106 [00:00<?, ?it/s]
```

[34]:
```
m = PCA(n_components=2).fit(np.log1p(X.T))
df = pd.DataFrame(m.components_.T, index=statements.index)

fig, ax = plt.subplots(1, 1, figsize=(8, 7))
years = [str(y) for y in df.index.year.unique()]
colors = cm.RdBu(np.linspace(0, 1, len(years)))
for i, y in enumerate(years):
    ax.scatter(x=df.loc[y][0], y=df.loc[y][1], color=colors[i])
ax.legend(years, loc="center left", bbox_to_anchor=(1, 0.5))
ax.set_xlabel("PC 0")
ax.set_ylabel("PC 1")

d = "2020-03-03"
ax.text(x=df.loc[d][0], y=df.loc[d][1], s=d);
```



[35]:
```
Xtfm = UMAP().fit_transform(X)
df_ = pd.DataFrame(Xtfm, index=statements.index)
classes = df.index.year.unique()
```

[36]:
```
fig, ax = plt.subplots(1, 1, figsize=(7, 7))
colors = cm.RdBu(np.linspace(0, 1, len(classes)))
for i, y in enumerate(classes):
    ax.scatter(df_.loc[str(y)][0], df_.loc[str(y)][1], color=colors[i])
ax.legend(classes, loc="center left", bbox_to_anchor=(1, 0.5))

d = "2020-03-03"
ax.text(df.loc[d][0], df.loc[d][1], d, fontsize=9, rotation=0, ha="right");
```



[37]:
```
df = pd.DataFrame(X, index=statements.index)
m = ElasticNet(alpha=0.01)
X_, y_ = pd.concat(
    [df.loc[fomc_change_up].assign(change=1), df.loc[fomc_change_dw].assign(change=-1)]
```

```
).pipe(lambda df: (df.drop("change", axis=1), df["change"]))
m.fit(X_, y_);
```

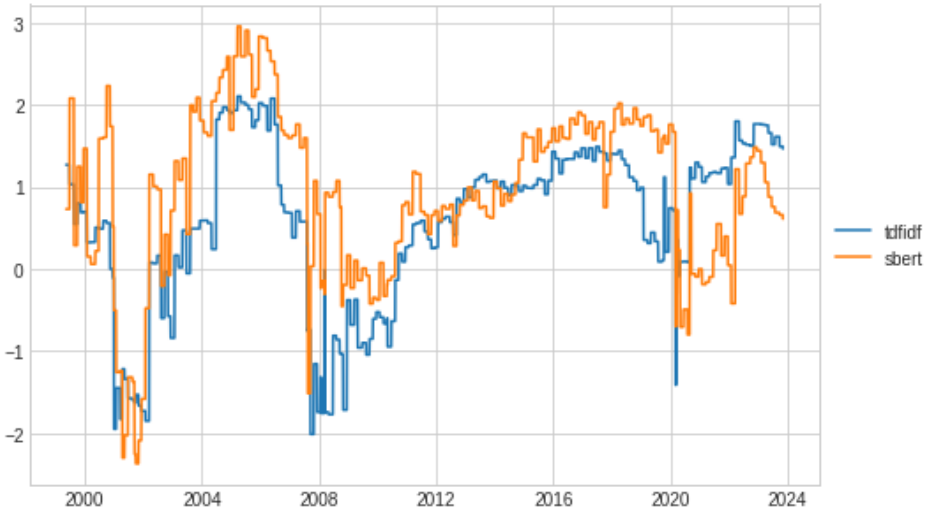
[38]:
```
pred_sbert = (
    pd.Series(m.predict(df), index=statements.index).resample("B").last().ffill()
)
```

[39]:
```
pd.concat({"sbert": pred_sbert, "tdfidf": pred_tfidf}, axis=1).corr()
```

[39]:

	sbert	tdfidf
sbert	1.000000	0.686509
tdfidf	0.686509	1.000000

[40]:
```
line(
    pd.concat({"sbert": pred_sbert, "tdfidf": pred_tfidf}, axis=1).pipe(
        lambda x: x.div(x.std())
    )
)
```



Chapter 19

Sentiment in FOMC statements

In this section, we use rule-based and learning-based methods to measure sentiment in Federal Open Market Committee (FMOC) statements. These two methods have been discussed previously in the context of corporate regulatory filings (10Ks).

19.1 Sentiment in FOMC statements: Loughran-McDonalds dictionary

In this section, we measure sentiment with the Loughran-McDonalds sentiment dictionary in two ways:

- sentiment = (#positive - #negative)/(#positive + #negative)
- sentiment = (#positive - #negative)/(#words)

In the first case, short documents (with few or no sentiment words) might lead to biased estimates.

[2]:
```
import pandas as pd
from skfin.datasets import load_fomc_statements, load_loughran_mcdonald_dictionary
from skfin.plot import line
from skfin.text import coefs_plot, show_text
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
```

[3]:
```
statements = load_fomc_statements()
lm = load_loughran_mcdonald_dictionary()
```

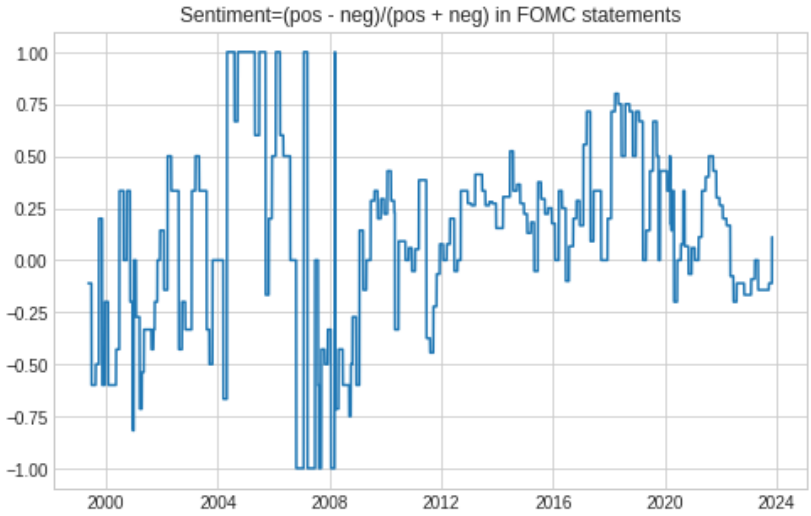
```
INFO:skfin.datasets:logging from cache file: data/fomc_statements.parquet
INFO:skfin.datasets:logging from cache file: data/Loughran-
McDonald_MasterDictionary_1993-2021.csv
```

[4]:
```
X = statements["text"]
```

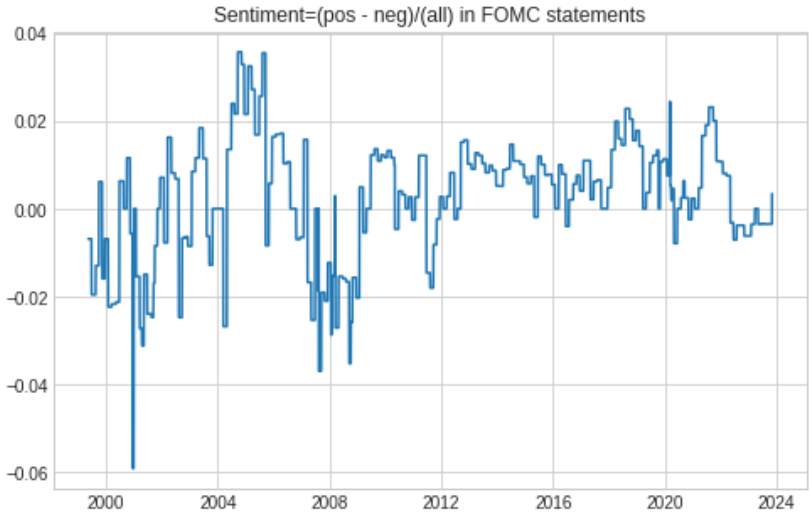
[5]:
```
funcs = {
    "negative": lambda x: x.Negative > 0,
    "positive": lambda x: x.Positive > 0,
    "all": lambda x: x.Word.notna(),
}


def get_total_count(X, lm, func):
    m = CountVectorizer(vocabulary=lm.loc[func].Word.str.lower().values)
    return pd.DataFrame(m.fit_transform(X).toarray(), index=X.index).sum(axis=1)
```

```
lm_counts = pd.concat({k: get_total_count(X, lm, v) for k, v in funcs.items()}, axis=1)
```

[6]:
```
line(
    lm_counts.pipe(lambda x: (x.positive - x.negative) / (x.positive + x.negative))
    .resample("B")
    .last()
    .ffill(),
    legend=False,
    title="Sentiment=(pos - neg)/(pos + neg) in FOMC statements",
)
```



[7]:
```
line(
    lm_counts.pipe(lambda x: (x.positive - x.negative) / x["all"])
    .resample("B")
    .last()
    .ffill(),
    legend=False,
    title="Sentiment=(pos - neg)/(all) in FOMC statements",
)
```



[8]:
```
lm_lexica = {
    "negative": pd.Series(1, lm.loc[lm.Negative > 0].Word.str.lower().values),
    "positive": pd.Series(1, lm.loc[lm.Positive > 0].Word.str.lower().values),
}
show_text(
    statements.loc[["2000-12-19", "2013-12-18", "2014-01-29"]], lexica=lm_lexica, n=None
)
```

```
<IPython.core.display.HTML object>
```

19.2 Sentiment in FOMC statements: supervised learning

Building on previous analyses, we build here a `scikit-learn pipeline` with a `Tfidfvectorizer` and a regularized regression`ElasticNet`. The target is the return of the market on the day of the statement.

[9]:
```
import numpy as np
from pandas.tseries.offsets import BDay
from skfin.datasets import load_kf_returns
from skfin.text import show_text
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import ElasticNet, ElasticNetCV
from sklearn.pipeline import Pipeline
```

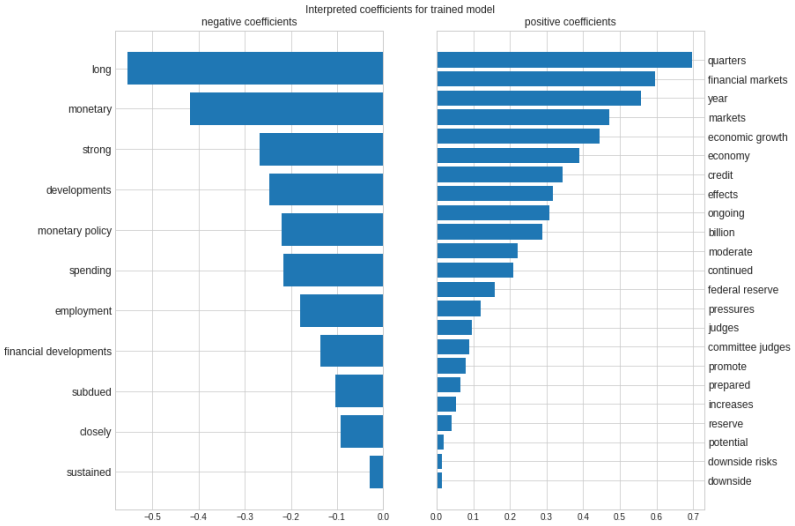
[10]:
```
ret = load_kf_returns(filename="F-F_Research_Data_Factors_daily")["Daily"]
```

```
INFO:skfin.datasets:logging from cache directory:
data/F-F_Research_Data_Factors_daily
```

[11]:
```
special_days = ["2008-01-22", "2010-05-09", "2020-03-15"]
idx0 = pd.to_datetime(pd.Index(special_days))
idx = statements.index.difference(idx0).union(idx0 + BDay(1))
ret_fomc = ret.div(ret.ewm(252).std()).loc[ret.index.intersection(idx)]
```

[12]:
```
est = Pipeline(
    [
        (
            "tfidf",
            TfidfVectorizer(
                vocabulary=None,
                ngram_range=(1, 3),
                max_features=500,
                stop_words="english",
                token_pattern=r"\b[a-zA-Z]{3,}\b",
            ),
        ),
        ("reg", ElasticNet(alpha=0.0075)),
    ]
)
y = ret_fomc["Mkt-RF"].dropna()
X = statements["text"]
idx_ = y.index.intersection(X.index)
X, y = X.loc[idx_], y.loc[idx_]
est.fit(X, y)
vocab_ = pd.Series(est.named_steps["tfidf"].vocabulary_).sort_values().index
interpret_coef = pd.DataFrame(np.transpose(est.named_steps["reg"].coef_), index=vocab_)
```

```
coefs_plot(interpret_coef, title="Interpreted coefficients for trained model")
```



[13]:
```
lexica = {
    "positive": interpret_coef.squeeze().nlargest(n=10),
    "negative": interpret_coef.squeeze().nsmallest(n=10),
}
```

[14]:
```
idx_ = (
    pd.Series(est.predict(X), index=X.index)
    .sort_values()
    .pipe(lambda x: [x.index[0], x.index[-1]])
)
show_text(statements.loc[idx_], lexica=lexica, n=None)
```

```
<IPython.core.display.HTML object>
```

Chapter 20

Conclusion

There are several packages that are natural extensions of the ideas described here:

- `jax`: https://github.com/google/jax
- `cvxpy`: https://github.com/cvxpy/cvxpy
- `optuna`: https://github.com/optuna/optuna or `auto-sklearn`: https://github.com/automl/auto-sklearn

Several recent significant progress were made in Natural Language Processing, Computer Vision, Audio, and Graph Neural Networks.

Moreover, similar analyses could be extented to several asset classes (for instance by leveraging `yfinance`: https://github.com/ranaroussi/yfinance or `pandas-datareader`: https://github.com/pydata/pandas-datareader):

- options
- futures contracts (including commodity futures)
- non-US stocks (e.g. Chinese stocks)
- bitcoin and other crypto currencies

Chapter 21

Appendix: helper functions

Here we create some helper functions that will be used across notebooks using the magic `%%writefile`.

21.1 Data visualisation

Data exploration, in particular based on visualisation, is crucial to modern data science. `Pandas` has a lot of plotting functionalities (e.g. see the graph below), but we will find it usefull to use a custom `plot` set of functions.

[2]:
```
%%writefile ../skfin/plot.py
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from skfin.metrics import sharpe_ratio

plt.style.use("seaborn-whitegrid")


def set_axis(ax=None, figsize=(8, 5), title=None, fig=None):
    if ax is None:
        fig, ax = plt.subplots(1, 1, figsize=figsize)
    if title is not None:
        ax.set_title(title)
    return fig, ax


def line(
    df,
    sort=True,
    figsize=(8, 5),
    ax=None,
    title="",
    cumsum=False,
    loc="center left",
    bbox_to_anchor=(1, 0.5),
    legend_sharpe_ratio=None,
    legend=True,
    yscale=None,
    start_date=None,
):
    df = df.copy()
```

```
    if loc == "best":
        bbox_to_anchor = None
    if isinstance(df, dict) | isinstance(df, list):
        df = pd.concat(df, axis=1)
    if isinstance(df, pd.Series):
        df = df.to_frame()
    if start_date is not None:
        df = df[start_date:]
    if cumsum & (legend_sharpe_ratio is None):
        legend_sharpe_ratio = True
    if legend_sharpe_ratio:
        df.columns = [f"{c}: sr={sharpe_ratio(df[c]): 3.2f}" for c in df.columns]
    if cumsum:
        df = df.cumsum()
    if sort:
        df = df.loc[:, lambda x: x.iloc[-1].sort_values(ascending=False).index]
    fig, ax = set_axis(ax=ax, figsize=figsize, title=title)
    ax.plot(df.index, df.values)
    if legend:
        ax.legend(df.columns, loc=loc, bbox_to_anchor=bbox_to_anchor)
    if yscale == "log":
        ax.set_yscale("log")


def bar(
    df,
    err=None,
    sort=True,
    figsize=(8, 5),
    ax=None,
    title=None,
    horizontal=False,
    baseline=None,
    rotation=0,
):
    if isinstance(df, pd.DataFrame):
        df = df.squeeze()
    if isinstance(df, dict):
        df = pd.Series(df)
    if sort:
        df = df.sort_values()
    if err is not None:
        err = err.loc[df.index]
    labels = df.index
    x = np.arange(len(labels))
    fig, ax = set_axis(ax=ax, figsize=figsize, title=title)
    if horizontal:
        ax.barh(x, df.values, xerr=err, capsize=5)
        ax.set_yticks(x)
        ax.set_yticklabels(labels, rotation=0)
        if baseline in df.index:
            df_ = df.copy()
            df_[df.index != baseline] = 0
            ax.barh(x, df_.values, color="lightgreen")
    else:
        ax.bar(x, df.values, yerr=err, capsize=5)
        ax.set_xticks(x)
        ax.set_xticklabels(labels, rotation=0)
        if baseline in df.index:
            df_ = df.copy()
            df_[df.index != baseline] = 0
```

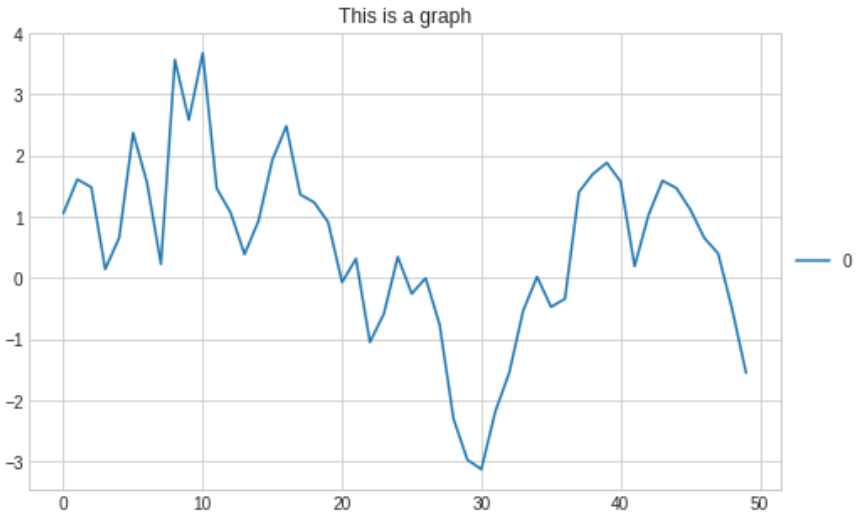
```
            ax.bar(x, df_.values, color="lightgreen")
    ax.set_title(title)


def heatmap(
    df,
    ax=None,
    fig=None,
    figsize=(8, 5),
    title=None,
    vmin=None,
    vmax=None,
    vcompute=True,
    cmap="RdBu",
):
    labels_x = df.index
    x = np.arange(len(labels_x))
    labels_y = df.columns
    y = np.arange(len(labels_y))
    if vcompute:
        vmax = df.abs().max().max()
        vmin = -vmax
    fig, ax = set_axis(ax=ax, figsize=figsize, title=title, fig=fig)
    pos = ax.imshow(
        df.T.values, cmap=cmap, interpolation="nearest", vmax=vmax, vmin=vmin
    )
    ax.set_xticks(x)
    ax.set_yticks(y)
    ax.set_xticklabels(labels_x, rotation=90)
    ax.set_yticklabels(labels_y)
    ax.grid(True)
    fig.colorbar(pos, ax=ax)


def scatter(
    df,
    ax=None,
    xscale=None,
    yscale=None,
    xlabel=None,
    ylabel=None,
    xticks=None,
    yticks=None,
    figsize=(8, 5),
    title=None,
):
    fig, ax = set_axis(ax=ax, figsize=figsize, title=title)
    ax.scatter(df, df.index, facecolors="none", edgecolors="b", s=50)
    if xlabel is not None:
        ax.set_xlabel(xlabel)
    if ylabel is not None:
        ax.set_ylabel(ylabel)
    if xscale is not None:
        ax.set_xscale(xscale)
    if yscale is not None:
        ax.set_yscale(yscale)
    if yticks is not None:
        ax.set_yticks(yticks)
        ax.set_yticklabels(yticks)
    if xticks is not None:
        ax.set_xticks(xticks)
```

```
        ax.set_xticklabels(xticks)
```

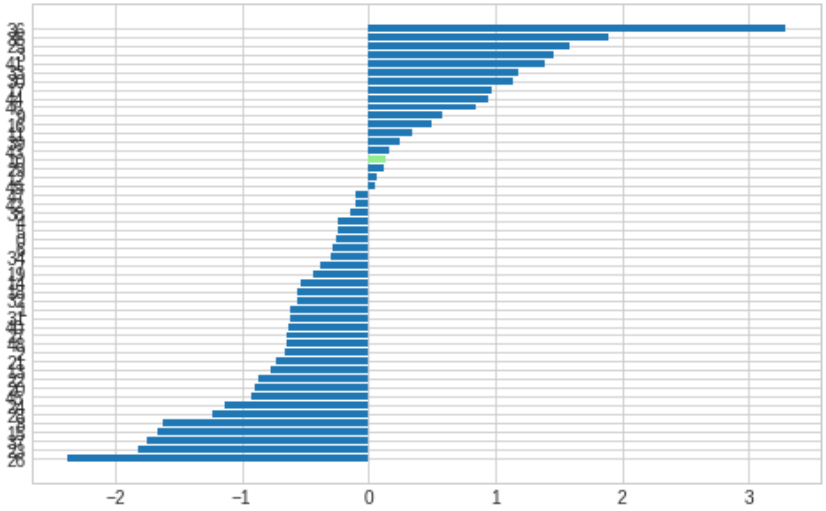
```
Overwriting ../skfin/plot.py
```

[3]:
```
from skfin.plot import bar, heatmap, line
```

[4]:
```
line(
    pd.Series(np.random.normal(size=50)),
    cumsum=True,
    title="This is a graph",
    legend_sharpe_ratio=False,
)
```



[5]:
```
bar(pd.Series(np.random.normal(size=50)), baseline=10, horizontal=True)
```



21.2 Dates and mappings

[6]:
```
%%writefile ../skfin/dataset_mappings.py
symbol_dict = {
    "TOT": "Total",
    "XOM": "Exxon",
    "CVX": "Chevron",
    "COP": "ConocoPhillips",
    "VLO": "Valero Energy",
    "MSFT": "Microsoft",
    "IBM": "IBM",
    "TWX": "Time Warner",
    "CMCSA": "Comcast",
    "CVC": "Cablevision",
    "YHOO": "Yahoo",
    "DELL": "Dell",
    "HPQ": "HP",
    "AMZN": "Amazon",
    "TM": "Toyota",
    "CAJ": "Canon",
    "SNE": "Sony",
    "F": "Ford",
    "HMC": "Honda",
    "NAV": "Navistar",
    "NOC": "Northrop Grumman",
    "BA": "Boeing",
    "KO": "Coca Cola",
    "MMM": "3M",
    "MCD": "McDonald's",
    "PEP": "Pepsi",
    "K": "Kellogg",
    "UN": "Unilever",
    "MAR": "Marriott",
    "PG": "Procter Gamble",
    "CL": "Colgate-Palmolive",
    "GE": "General Electrics",
    "WFC": "Wells Fargo",
    "JPM": "JPMorgan Chase",
    "AIG": "AIG",
    "AXP": "American express",
    "BAC": "Bank of America",
    "GS": "Goldman Sachs",
    "AAPL": "Apple",
    "SAP": "SAP",
    "CSCO": "Cisco",
    "TXN": "Texas Instruments",
    "XRX": "Xerox",
    "WMT": "Wal-Mart",
    "HD": "Home Depot",
    "GSK": "GlaxoSmithKline",
    "PFE": "Pfizer",
    "SNY": "Sanofi-Aventis",
    "NVS": "Novartis",
    "KMB": "Kimberly-Clark",
    "R": "Ryder",
    "GD": "General Dynamics",
    "RTN": "Raytheon",
    "CVS": "CVS",
    "CAT": "Caterpillar",
    "DD": "DuPont de Nemours",
}
```

```
mapping_10X = {
    "AAPL": ["APPLE COMPUTER INC", "APPLE INC"],
    "AIG": "AMERICAN INTERNATIONAL GROUP INC",
    "AMZN": "AMAZON COM INC",
    "AXP": "AMERICAN EXPRESS CO",
    "BA": "BOEING CO",
    "BAC": "BANK OF AMERICA CORP /DE/",
    "CAT": "CATERPILLAR INC",
    "CL": "COLGATE PALMOLIVE CO",
    "CMCSA": "COMCAST CORP",
    "COP": "CONOCOPHILLIPS",
    "CSCO": "CISCO SYSTEMS INC",
    "CVC": "CABLEVISION SYSTEMS CORP /NY",
    "CVS": ["CVS CORP", "CVS/CAREMARK CORP", "CVS CAREMARK CORP"],
    "CVX": ["CHEVRONTEXACO CORP", "CHEVRON CORP"],
    "DD": "DUPONT E I DE NEMOURS & CO",
    "DELL": ["DELL COMPUTER CORP", "DELL INC"],
    "F": "FORD MOTOR CO",
    "GD": "GENERAL DYNAMICS CORP",
    "GE": "GENERAL ELECTRIC CO",
    "GS": "GOLDMAN SACHS GROUP INC/",
    "HD": "HOME DEPOT INC",
    "HPQ": "HEWLETT PACKARD CO",
    "IBM": "INTERNATIONAL BUSINESS MACHINES CORP",
    "JPM": "J P MORGAN CHASE & CO",
    "K": "KELLOGG CO",
    "KMB": "KIMBERLY CLARK CORP",
    "KO": "COCA COLA CO",
    "MAR": "MARRIOTT INTERNATIONAL INC /MD/",
    "MCD": "MCDONALDS CORP",
    "MMM": "3M CO",
    "MSFT": "MICROSOFT CORP",
    "NAV": "NAVISTAR INTERNATIONAL CORP",
    "NOC": "NORTHROP GRUMMAN CORP /DE/",
    "PEP": "PEPSI BOTTLING GROUP INC",
    "PFE": "PFIZER INC",
    "PG": "PROCTER & GAMBLE CO",
    "R": "RYDER SYSTEM INC",
    "RTN": "RAYTHEON CO/",
    "TWX": ["AOL TIME WARNER INC", "TIME WARNER INC"],
    "TXN": "TEXAS INSTRUMENTS INC",
    "VLO": "VALERO ENERGY CORP/TX",
    "WFC": "WELLS FARGO & CO/MN",
    "WMT": "WAL MART STORES INC",
    "XOM": "EXXON MOBIL CORP",
    "XRX": "XEROX CORP",
    "YHOO": "YAHOO INC",
}
```

```
Overwriting ../skfin/dataset_mappings.py
```

[7]:
```
%%writefile ../skfin/dataset_dates.py
import pandas as pd


def load_fomc_change_date(as_datetime=True):
    change_up = [
        "1999-06-30",
        "1999-08-24",
        "1999-11-16",
```

```
        "2000-02-02",
        "2000-03-21",
        "2000-05-16",
        "2004-06-30",
        "2004-08-10",
        "2004-09-21",
        "2004-11-10",
        "2004-12-14",
        "2005-02-02",
        "2005-03-22",
        "2005-05-03",
        "2005-06-30",
        "2005-08-09",
        "2005-09-20",
        "2005-11-01",
        "2005-12-13",
        "2006-01-31",
        "2006-03-28",
        "2006-05-10",
        "2006-06-29",
        "2015-12-16",
        "2016-12-14",
        "2017-03-15",
        "2017-06-14",
        "2017-12-13",
        "2018-03-21",
        "2018-06-13",
        "2018-09-26",
        "2018-12-19",
        "2022-03-16",
        "2022-05-04",
        "2022-06-15",
        "2022-07-27",
    ]

    change_dw = [
        "2001-01-03",
        "2001-01-31",
        "2001-03-20",
        "2001-04-18",
        "2001-05-15",
        "2001-06-27",
        "2001-08-21",
        "2001-09-17",
        "2001-10-02",
        "2001-11-06",
        "2001-12-11",
        "2002-11-06",
        "2003-06-25",
        "2007-09-18",
        "2007-10-31",
        "2007-12-11",
        "2008-01-22",
        "2008-01-30",
        "2008-03-18",
        "2008-04-30",
        "2008-10-08",
        "2008-10-29",
        "2008-12-16",
        "2019-07-31",
        "2019-09-18",
```

```
        "2019-10-30",
        "2020-03-03",
        "2020-03-15",
    ]
    if as_datetime:
        change_up, change_dw = pd.to_datetime(change_up), pd.to_datetime(change_dw)

    return change_up, change_dw
```

```
Overwriting ../skfin/dataset_dates.py
```

21.3 Data utils

[8]:
```
%%writefile ../skfin/data_utils.py
import os
from pathlib import Path

import pandas as pd


def clean_directory_path(cache_dir, default_dir="data"):
    if cache_dir is None:
        cache_dir = Path(os.getcwd()) / default_dir
    if isinstance(cache_dir, str):
        cache_dir = Path(cache_dir)
    if not cache_dir.is_dir():
        os.makedirs(cache_dir)
    return cache_dir


def save_dict(data, output_dir):
    assert isinstance(data, dict)
    if not output_dir.is_dir():
        os.mkdir(output_dir)
    for k, v in data.items():
        if isinstance(v, pd.DataFrame):
            v.to_parquet(output_dir / f"{k}.parquet")
        else:
            save_dict(v, output_dir=output_dir / k)


def load_dict(input_dir):
    data = {}
    for o in os.scandir(input_dir):
        if o.name.endswith(".parquet"):
            k = o.name.replace(".parquet", "")
            data[k] = pd.read_parquet(o)
        elif o.is_dir:
            data[o.name] = load_dict(o)
    return data
```

```
Overwriting ../skfin/data_utils.py
```

Chapter 22

Appendix: helper text visualisation

[2]:
```
%%writefile ../skfin/text.py
import re
import sys

import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

plt.style.use("seaborn-whitegrid")

from IPython.display import HTML, display

pd.options.display.max_colwidth = None

import logging

logging.basicConfig(stream=sys.stdout, level=logging.INFO)
logger = logging.getLogger(__name__)


def show_text(df, lexica=None, text_column="text", n=2):
    if n is not None:
        df = df.sample(n=n)
    df = df.assign(
        **{
            text_column: lambda x: x[text_column]
            .str.replace("$", "\$", regex=False)
            .str.replace("\n", " ", regex=False)
        }
    )
    if lexica is not None:
        df = df.assign(
            **{
                text_column: lambda x: x[text_column].apply(
                    highlight_lexica, lexica=lexica
                )
            }
        )
    display(HTML(df.to_html(escape=False)))


green_text = lambda x: f"<b><font color = green>{x}</font></b>"
red_text = lambda x: f"<b><font color = red>{x}</font></b>"
```

```
def color_text(x, lexica):
    if x.lower() in lexica["positive"]:
        return green_text(x)
    elif x.lower() in lexica["negative"]:
        return red_text(x)
    else:
        return x


def highlight_lexica(string, lexica):
    if isinstance(string, list):
        string = string[0]
    string = string.replace("<br /><br />", "")
    return " ".join([color_text(x, lexica) for x in string.split(" ")])


def coefs_plot(coef, n=40, fontsize=12, rotation=0, title=None, filename=None):
    """
    plot the coefficients from a  tfidf+linear_model pipeline on words (with positive and␣
 ↪negative values)
    """
    fig, ax = plt.subplots(1, 2, figsize=(12, 10))
    df_pos = coef.squeeze().loc[lambda x: x > 0].sort_values().tail(n)
    labels = df_pos.index
    x = np.arange(len(labels))
    ax[1].barh(x, df_pos.values, capsize=5)
    ax[1].set_yticks(x)
    ax[1].set_yticklabels(labels, rotation=rotation, fontsize=fontsize)
    ax[1].yaxis.tick_right()
    ax[1].set_title("positive coefficients")

    df_neg = coef.squeeze().loc[lambda x: x < 0].sort_values(ascending=False).tail(n)
    labels = df_neg.index
    x = np.arange(len(labels))
    ax[0].barh(x, df_neg.values, capsize=5)
    ax[0].set_yticks(x)
    ax[0].set_yticklabels(labels, rotation=rotation, fontsize=fontsize)
    ax[0].set_title("negative coefficients")
    if title is not None:
        fig.suptitle(title, y=0.92)
    if filename is not None:
        plt.savefig(
            str(filename) + ".png", orientation="landscape", bbox_inches="tight"
        )


def error_analysis_plot(data, lexica, n=5):
    error_analysis = (
        data.assign(diff=lambda x: x["label"] - x["pred"])
        .sort_values("diff")
        .loc[:, ["label", "pred", "text"]]
    )
    if n is not None:
        error_analysis = error_analysis.pipe(
            lambda x: pd.concat([x.head(n), x.tail(n)])
        )
    show_text(error_analysis, lexica)
```

```
Overwriting ../skfin/text.py
```

Chapter 23

Appendix: Projet template

We outline a structured approach for presenting research findings. The framework is divided into several key segments:

1. Introduction
2. Dataset overview
3. Analytics and learning strategies
4. Empirical resuts: baseline and robustness
5. Conclusion

The opening segment encompasses four essential elements:

- Contextual Background: What is the larger setting of the study? What makes this area of inquiry compelling? What are the existing gaps or limitations within the current body of research? What are some unanswered yet noteworthy questions?
- Project Contributions: What are the specific advancements made by this study, such as in data acquisition, algorithmic development, parameter adjustments, etc.?
- Summary of the main empirical results: What is the main statistical statement? is it significant (e.g. statistically or economically)?
- Literature and Resource Citations: What are related academic papers? What are the github repositories, expert blogs, or software packages that used in this project?

In the dataset profile, one should consider:

- The origin and composition of data utilized in the study. If the dataset is original, then provide the source code to ensure reproducibility.
- The chronological accuracy of the data points, verifying that the dates reflect the actual availability of information.
- A detailed analysis of descriptive statistics, with an emphasis on discussing the importance of the chosen graphs or metrics.

The analytics and machine learning methodologies section accounts for:

- A detailed explanation of the foundational algorithm.
- A description of the data partitioning strategy for training, validation and test.
- An overview of the parameter selection and optimization process.

To effectively convey the empirical findings, separate the baseline results from the additional robustness tests. Within the primary empirical outcomes portion, include:

- Key statistical evaluations (for instance, if presenting a backtest – provide a pnl graph alongside the Sharpe ratio).
- Insights into what primarily influences the results, such as specific characteristics or assets that significantly impact performance.

The robustness of empirical tests section should detail:

- Evaluation of the stability of the principal finding against variations in hyperparameters or algorithmic modifications.

Finally, the conclusive synthesis should recapitulate the primary findings, consider external elements that may influence the results, and hint at potential directions for further investigative work.

Bibliography

J.-P. Bouchaud, J. Bonart, J. Donier, and M. Gould. *Trades, quotes and prices: financial markets under the microscope*. Cambridge University Press, 2018.

M. Britten-Jones. The sampling error in estimates of mean-variance efficient portfolio weights. *The Journal of Finance*, 54(2):655–671, 1999.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

A. Frazzini, D. Kabiller, and L. H. Pedersen. Buffett's alpha. *Financial Analysts Journal*, 74(4):35–55, 2018.

T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

F. Ielpo, C. Merhy, and G. Simon. *Engineering Investment Process: Making Value Creation Repeatable*. Elsevier, 2017.

M. Isichenko. *Quantitative Portfolio Management: The Art and Science of Statistical Arbitrage*. Wiley, 2022.

S. Jansen. *Machine Learning for Algorithmic Trading: Predictive models to extract signals from market and alternative data for systematic trading strategies with Python*. Packt Publishing Ltd, 2020.

N. Jegadeesh and D. Wu. Word power: A new approach for content analysis. *Journal of financial economics*, 110(3):712–729, 2013.

R. N. Kahn. *The future of investment management*. CFA Institute Research Foundation, 2018.

A. Karpathy. A recipe for training neural networks. 2019.

A. E. Khandani and A. W. Lo. What happened to the quants in august 2007? evidence from factors and transactions data. *Journal of Financial Markets*, 14(1):1–46, 2011.

O. Ledoit and M. Wolf. Honey, i shrunk the sample covariance matrix. *The Journal of Portfolio Management*, 30(4):110–119, 2004.

A. W. Lo. The statistics of sharpe ratios. *Financial analysts journal*, 58(4):36–52, 2002.

A. W. Lo and A. C. MacKinlay. When are contrarian profits due to stock market overreaction? *The review of financial studies*, 3(2):175–205, 1990.

T. Loughran and B. McDonald. When is a liability not a liability? textual analysis, dictionaries, and 10-ks. *The Journal of finance*, 66(1):35–65, 2011.

T. J. Moskowitz and M. Grinblatt. Do industries explain momentum? *The Journal of finance*, 54(4): 1249–1290, 1999.

K. P. Murphy. *Probabilistic machine learning: an introduction*. MIT press, 2022.

L. H. Pedersen. *Efficiently inefficient: how smart money invests and market prices are determined*. Princeton University Press, 2019.

P. Savor and M. Wilson. How much do investors care about macroeconomic risk? evidence from scheduled economic announcements. *Journal of Financial and Quantitative Analysis*, 48(2):343–375, 2013.

W. F. Sharpe. Asset allocation: Management style and performance measurement. *Journal of portfolio Management*, 18(2):7–19, 1992.