03_pca_and_risk_factor_models

September 29, 2021

1 PCA for Algorithmic Trading

PCA is useful for algorithmic trading in several respects. These include the data-driven derivation of risk factors by applying PCA to asset returns, and the construction of uncorrelated portfolios based on the principal components of the correlation matrix of asset returns.

In Chapter 07 - Linear Models, we explored risk factor models used in quantitative finance to capture the main drivers of returns. These models explain differences in returns on assets based on their exposure to systematic risk factors and the rewards associated with these factors.

In particular, we explored the Fama-French approach that specifies factors based on prior knowledge about the empirical behavior of average returns, treats these factors as observable, and then estimates risk model coefficients using linear regression. An alternative approach treats risk factors as latent variables and uses factor analytic techniques like PCA to simultaneously estimate the factors and how the drive returns from historical returns.

In this section, we will review how this method derives factors in a purely statistical or data-driven way with the advantage of not requiring ex-ante knowledge of the behavior of asset returns.

1.1 Imports & Settings

[1]:
```
%matplotlib inline

import warnings
import os
from pathlib import Path
import quandl
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA, IncrementalPCA, FastICA
from sklearn.preprocessing import scale
```

[2]:
```
warnings.filterwarnings('ignore')
sns.set_style('darkgrid')
np.random.seed(42)
```

1.2 Get returns for equities with highest market cap

We will use the Quandl stock price data and select the daily adjusted close prices the 500 stocks with the largest market capitalization and data for the 2010-2018 period. We then compute the daily returns as follows:

[19]:
```
idx = pd.IndexSlice
with pd.HDFStore('../../data/assets.h5') as store:
    stocks = store['us_equities/stocks'].marketcap.nlargest(500)
    returns = (store['quandl/wiki/prices']
               .loc[idx['2010': '2018', stocks.index], 'adj_close']
               .unstack('ticker')
               .pct_change())
```

We obtain 215 stocks and returns for over 2,000 trading days:

[20]:
```
returns.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2072 entries, 2010-01-04 to 2018-03-27
Columns: 215 entries, AAPL to GSBD
dtypes: float64(215)
memory usage: 3.4 MB
```

1.2.1 Winsorize & standardize returns

PCA is sensitive to outliers so we winsorize the data at the 2.5% and 97.5% quantiles, respectively:

[5]:
```
returns = returns.clip(lower=returns.quantile(q=.025),
                       upper=returns.quantile(q=.975),
                       axis=1)
```

1.2.2 Impute missing values

PCA does not permit missing data, so we will remove stocks that do not have data for at least 95% of the time period, and in a second step remove trading days that do not have observations on at least 95% of the remaining stocks.

[6]:
```
returns = returns.dropna(thresh=int(returns.shape[0] * .95), axis=1)
returns = returns.dropna(thresh=int(returns.shape[1] * .95))
returns.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2071 entries, 2010-01-05 to 2018-03-27
Columns: 173 entries, AAPL to PLOW
dtypes: float64(173)
memory usage: 2.7 MB
```

We are left with 173 equity return series covering a similar period.

We impute any remaining missing values using the average return for any given trading day:

[7]:
```
daily_avg = returns.mean(1)
returns = returns.apply(lambda x: x.fillna(daily_avg))
```

1.3 Fit PCA

Now we are ready to fit the principal components model to the asset returns using default parameters to compute all components using the full SVD algorithm:

[8]:
```
cov = np.cov(returns, rowvar=False)
```

[9]:
```
pca = PCA(n_components='mle')
pca.fit(returns)
```

[9]:
```
PCA(copy=True, iterated_power='auto', n_components='mle', random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

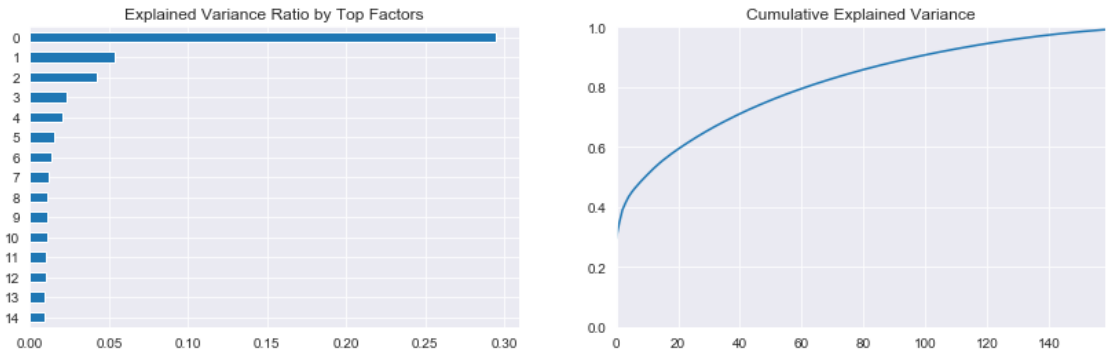
1.3.1 Visualize Explained Variance

We find that the most important factor explains around 30% of the daily return variation. The dominant factor is usually interpreted as 'the market', whereas the remaining factors can be interpreted as industry or style factors in line with our discussion in chapters 5 and 7, depending on the results of closer inspection (see next example).

The plot on the right shows the cumulative explained variance and indicates that around 10 factors explain 60% of the returns of this large cross-section of stocks.

The cumulative plot shows a typical 'elbow' pattern that can help identify a suitable target dimensionality because it indicates that additional components add less explanatory value.

[10]:
```
fig, axes = plt.subplots(ncols=2, figsize=(14,4))
pd.Series(pca.explained_variance_ratio_).iloc[:15].sort_values().plot.
 ↪barh(title='Explained Variance Ratio by Top Factors',ax=axes[0]);
pd.Series(pca.explained_variance_ratio_).cumsum().plot(ylim=(0,1),ax=axes[1],␣
 ↪title='Cumulative Explained Variance');
```



[11]:
```
risk_factors = pd.DataFrame(pca.transform(returns)[:, :2],
                            columns=['Principal Component 1', 'Principal␣
 ↪Component 2'],
                            index=returns.index)
risk_factors.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2071 entries, 2010-01-05 to 2018-03-27
Data columns (total 2 columns):
Principal Component 1    2071 non-null float64
Principal Component 2    2071 non-null float64
dtypes: float64(2)
memory usage: 48.5 KB
```

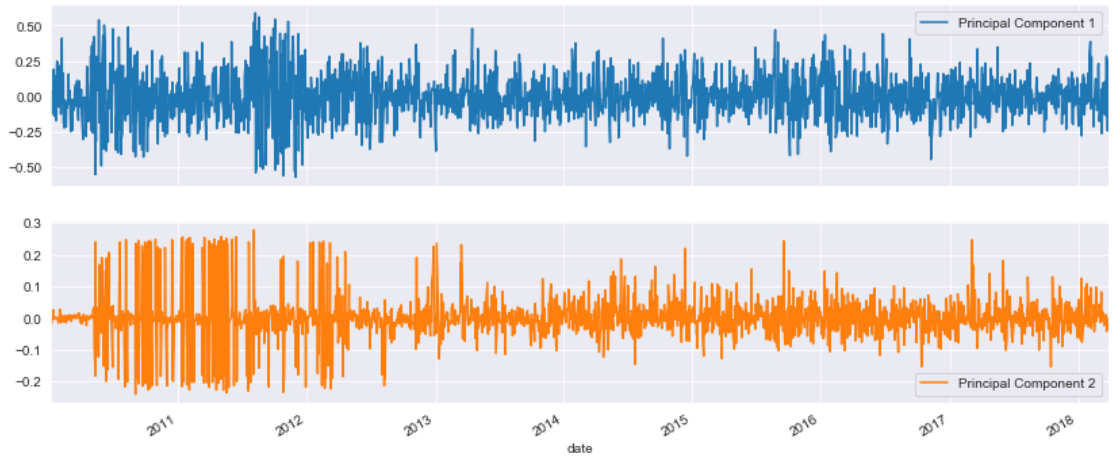
We can select the top two principal components to verify that they are indeed uncorrelated:

[12]:
```
risk_factors['Principal Component 1'].corr(risk_factors['Principal Component␣
 ↪2'])
```

[12]:
```
-1.1270779275205213e-14
```

Moreover, we can plot the time series to highlight how each factor captures different volatility patterns.

[13]:
```
risk_factors.plot(subplots=True, figsize=(14,6));
```



A risk factor model would employ a subset of the principal components as features to predict future returns, similar to our approach in chapter 7.

1.4 Simulation for larger number of stocks

[14]:
```
idx = pd.IndexSlice
with pd.HDFStore('../../data/assets.h5') as store:
    returns = (store['quandl/wiki/prices']
               .loc[idx['2000': '2018', :], 'adj_close']
               .unstack('ticker')
               .pct_change())
```

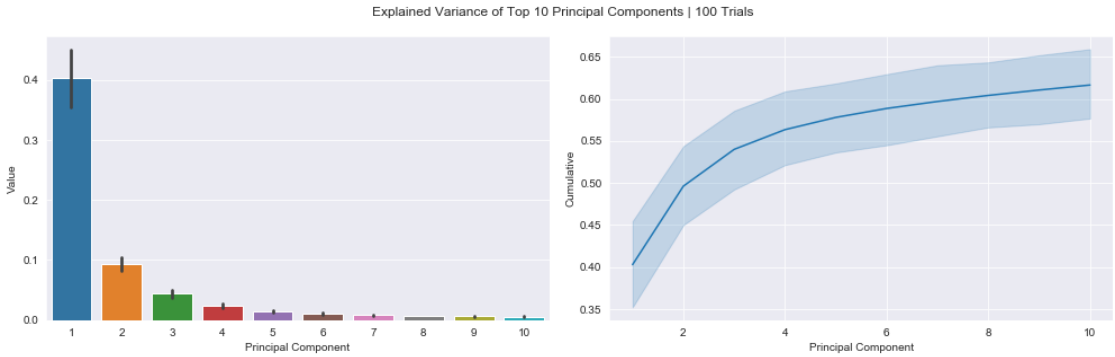
[15]:
```
pca = PCA()
n_trials, n_samples = 100, 500
explained = np.empty(shape=(n_trials, n_samples))
for trial in range(n_trials):
    returns_sample = returns.sample(n=n_samples)
    returns_sample = returns_sample.dropna(thresh=int(returns_sample.shape[0] *␣
 ↪.95), axis=1)
    returns_sample = returns_sample.dropna(thresh=int(returns_sample.shape[1] *␣
 ↪.95))
    daily_avg = returns_sample.mean(1)
    returns_sample = returns_sample.apply(lambda x: x.fillna(daily_avg))
    pca.fit(returns_sample)
    explained[trial, :len(pca.components_)] = pca.explained_variance_ratio_
```

[16]:
```
explained = pd.DataFrame(explained, columns=list(range(1, explained.shape[1] +␣
 ↪1)))
explained.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Columns: 500 entries, 1 to 500
dtypes: float64(500)
memory usage: 390.7 KB
```

[17]:
```
fig, axes =plt.subplots(ncols=2, figsize=(14, 4.5))
pc10 = explained.iloc[:, :10].stack().reset_index()
pc10.columns = ['Trial','Principal Component', 'Value']

pc10['Cumulative'] = pc10.groupby('Trial').Value.transform(np.cumsum)
sns.barplot(x='Principal Component', y='Value', data=pc10, ax=axes[0])
sns.lineplot(x='Principal Component', y='Cumulative', data=pc10, ax=axes[1])
fig.suptitle('Explained Variance of Top 10 Principal Components | 100 Trials')
fig.tight_layout()
fig.subplots_adjust(top=.90);
```



1.5 Eigenportfolios

Another application of PCA involves the covariance matrix of the normalized returns. The principal components of the correlation matrix capture most of the covariation among assets in descending order and are mutually uncorrelated. Moreover, we can use standardized the principal components as portfolio weights.

Let's use the 30 largest stocks with data for the 2010-2018 period to facilitate the exposition:

[22]:
```
idx = pd.IndexSlice
with pd.HDFStore('../../data/assets.h5') as store:
    stocks = store['us_equities/stocks'].marketcap.nlargest(30)
    returns = (store['quandl/wiki/prices']
               .loc[idx['2010': '2018', stocks.index], 'adj_close']
               .unstack('ticker')
               .pct_change())
```

We again winsorize and also normalize the returns:

[23]:
```
normed_returns = scale(returns
                       .clip(lower=returns.quantile(q=.025),
                             upper=returns.quantile(q=.975),
                             axis=1)
                      .apply(lambda x: x.sub(x.mean()).div(x.std())))
```

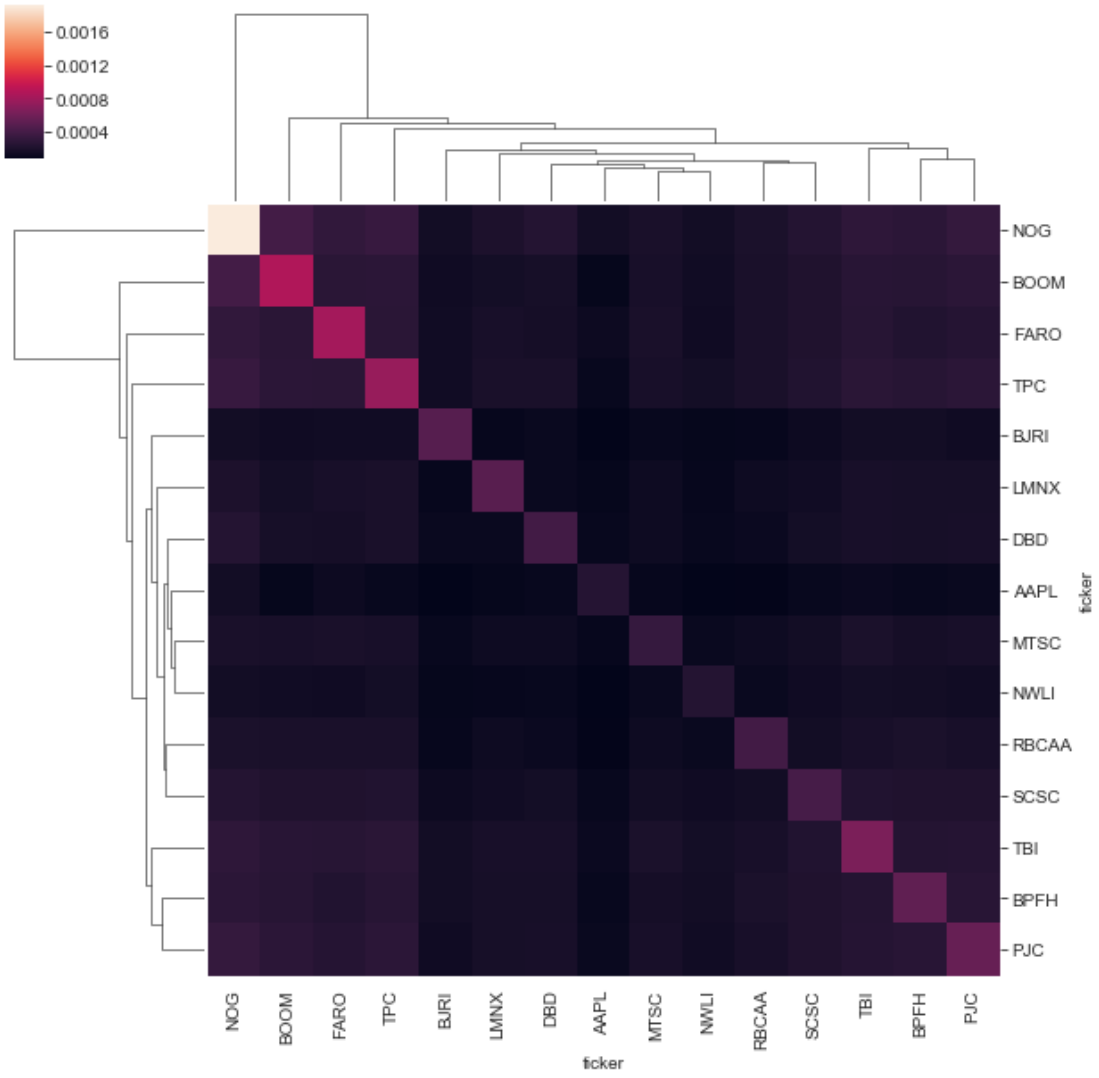
[24]:
```
returns = returns.dropna(thresh=int(returns.shape[0] * .95), axis=1)
returns = returns.dropna(thresh=int(returns.shape[1] * .95))
returns.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2070 entries, 2010-01-05 to 2018-03-27
Data columns (total 15 columns):
AAPL     2070 non-null float64
BJRI     2070 non-null float64
BOOM     2070 non-null float64
```

```
BPFH     2070 non-null float64
DBD      2070 non-null float64
FARO     2070 non-null float64
LMNX     2070 non-null float64
MTSC     2070 non-null float64
NOG      2070 non-null float64
NWLI     2070 non-null float64
PJC      2070 non-null float64
RBCAA    2070 non-null float64
SCSC     2070 non-null float64
TBI      2070 non-null float64
TPC      2070 non-null float64
dtypes: float64(15)
memory usage: 258.8 KB
```

[25]:
```
cov = returns.cov()
```

[26]:
```
sns.clustermap(cov);
```



After dropping assets and trading days as in the previous example, we are left with 23 assets and over 2,000 trading days. We estimate all principal components and find that the two largest explain 57.6% and 12.4% of the covariation, respectively:

[27]:
```
pca = PCA()
pca.fit(cov)
pd.Series(pca.explained_variance_ratio_).to_frame('Explained Variance').head().
↪style.format('{:,.2%}'.format)
```

[27]:
```
<pandas.io.formats.style.Styler at 0x7f03f718e3c8>
```

1.5.1 Create PF weights from principal components

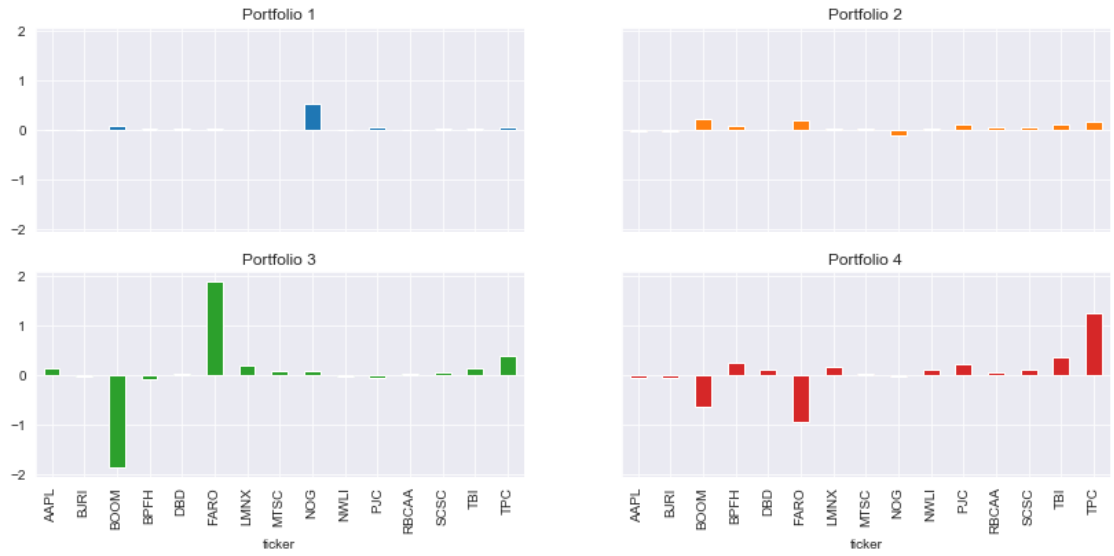
Next, we select and normalize the four largest components so that they sum to 1 and we can use them as weights for portfolios that we can compare to an equal-weighted portfolio formed from all stocks::

[28]:
```
top4 = pd.DataFrame(pca.components_[:4], columns=cov.columns)
eigen_portfolios = top4.div(top4.sum(1), axis=0)
eigen_portfolios.index = [f'Portfolio {i}' for i in range(1, 5)]
```

1.5.2 Eigenportfolio Weights

The weights show distinct emphasis, e.g., portfolio 3 puts large weights on Mastercard and Visa, the two payment processors in the sampel whereas potfolio 2 has more exposure to some technology companies:

[29]:
```
eigen_portfolios.T.plot.bar(subplots=True, layout=(2,2), figsize=(14,6),␣
↪legend=False, sharey=True);
```

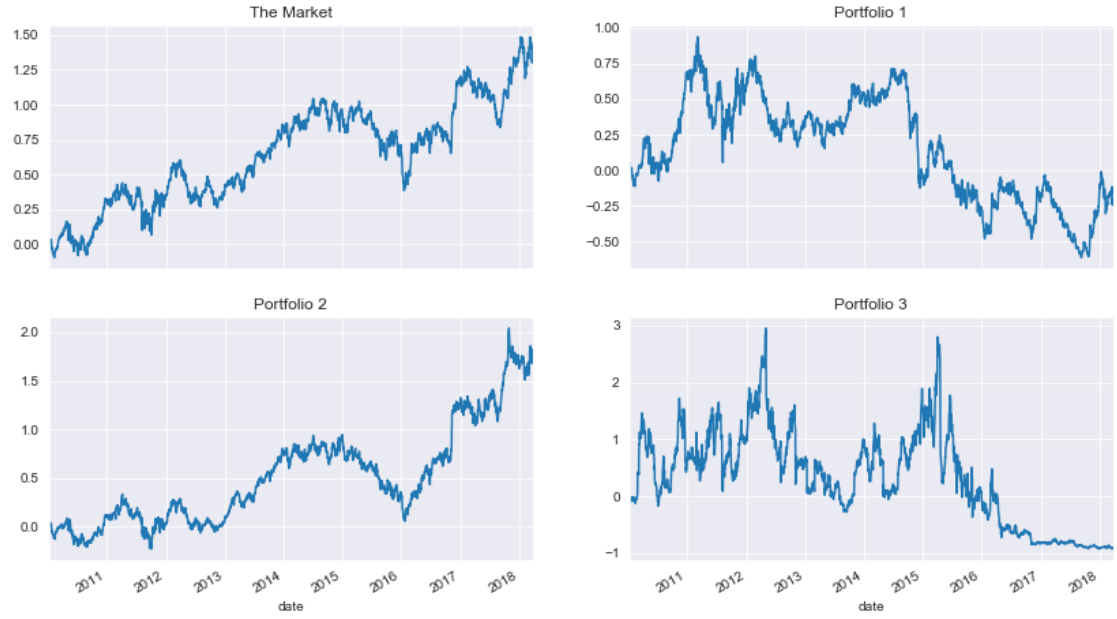


1.5.3 Eigenportfolio Performance

When comparing the performance of each portfolio over the sample period to 'the market' consisting of our small sample, we find that portfolio 1 performs very similarly, whereas the other portfolios capture different return patterns.

[30]:
```
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(14,8), sharex=True)
axes = axes.flatten()
returns.mean(1).add(1).cumprod().sub(1).plot(title='The Market', ax=axes[0]);
for i in range(3):
```

```
    returns.mul(eigen_portfolios.iloc[i]).sum(1).add(1).cumprod().sub(1).
↪plot(title=f'Portfolio {i+1}', ax=axes[i+1]);
```



[ ]: