hrp

September 29, 2021

1 Hierarchical Risk Parity

The key idea of hierarchical risk parity is to use hierarchical clustering on the covariance matrix to be able to group assets with similar correlations together and reduce the number of degrees of freedom by only considering 'similar' assets as substitutes when constructing the portfolio.

1.1 Imports & Settings

[1]:
```
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.cluster.hierarchy import linkage
from scipy.spatial.distance import pdist, squareform
```

[2]:
```
%matplotlib inline
plt.style.use('fivethirtyeight')
np.random.seed(42)
```

1.2 Load Data

[9]:
```
with pd.HDFStore('../../data/assets.h5') as store:
    sp500_stocks = store['sp500/stocks'].index
    prices = store['quandl/wiki/prices'].adj_close.unstack('ticker').
 ↪filter(sp500_stocks)
```

[10]:
```
start = 1988
end = 2017
```

[11]:
```
monthly_returns = prices.loc[f'{start}':f'{end}'].resample('M').last().
 ↪pct_change().dropna(how='all')
monthly_returns = monthly_returns.dropna(axis=1)
monthly_returns.columns.names = ['Ticker']
monthly_returns.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 359 entries, 1988-02-29 to 2017-12-31
```

```
Freq: M
Columns: 214 entries, MMM to XRX
dtypes: float64(214)
memory usage: 603.0 KB
```

1.3 HRP Source

The first step is to compute a distance matrix that represents proximity for correlated assets and meets distance metric requirements. The resulting matrix becomes an input to the scipy hierarchical clustering function that computes the successive clusters using one of several available methods as discussed above.

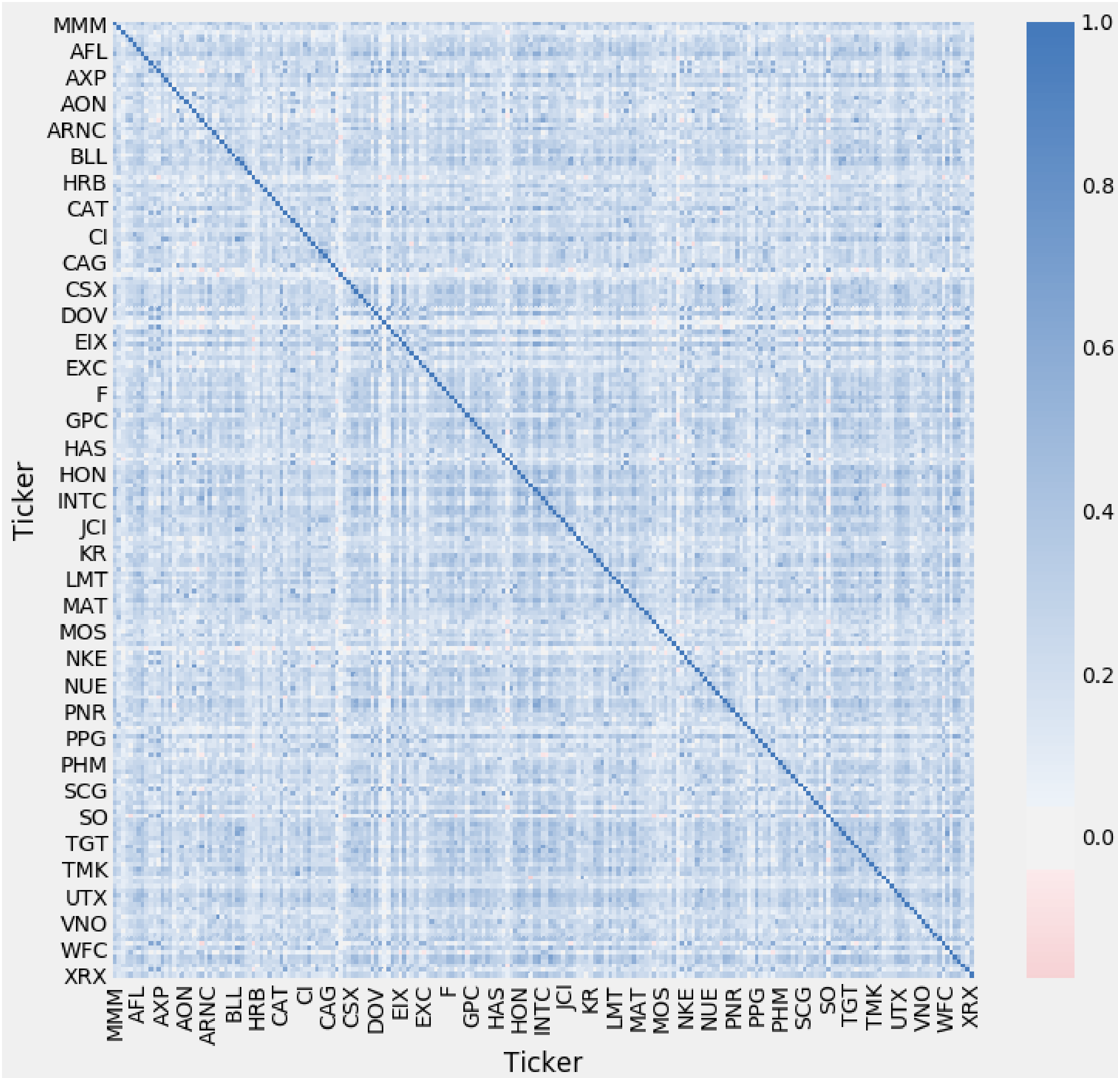
[6]:
```python
def get_inverse_var_pf(cov, **kargs):
    """Compute the inverse-variance portfolio"""
    ivp = 1 / np.diag(cov)
    return ivp / ivp.sum()
```

[7]:
```python
def get_distance_matrix(corr):
    """Compute distance matrix from correlation;
        0 <= d[i,j] <= 1"""
    return np.sqrt((1 - corr) / 2)
```

1.4 Get Correlation Matrix

[8]:
```python
cov = monthly_returns.cov()
corr = monthly_returns.corr()
corr.columns.names=['Ticker']
```

[9]:
```python
cmap = sns.diverging_palette(10, 250, as_cmap=True)
fig, ax = plt.subplots(figsize=(11,10))
sns.heatmap(corr, center = 0, cmap = cmap, ax=ax)
fig.tight_layout()
# fig.savefig('correl_map.png', dpi=300);
```



1.5 Cluster Return Series

[10]:

```
def quasi_diagonalize(link):
    """sort clustered assets by distance"""
    link = link.astype(int)
    sort_idx = pd.Series([link[-1, 0], link[-1, 1]])
    num_items = link[-1, 3]  # idx of original items
    while sort_idx.max() >= num_items:
        sort_idx.index = list(range(0, sort_idx.shape[0] * 2, 2))  # make space
        df0 = sort_idx[sort_idx >= num_items]  # find clusters
        i = df0.index
        j = df0.values - num_items
        sort_idx[i] = link[j, 0]  # item 1
        df0 = pd.Series(link[j, 1], index=i + 1)
```

```
        sort_idx = sort_idx.append(df0)  # item 2
        sort_idx = sort_idx.sort_index()  # re-sort
        sort_idx.index = list(range(sort_idx.shape[0]))  # re-index
    return sort_idx.tolist()
```

[11]:
```
distance_matrix = get_distance_matrix(corr)
linkage_matrix = linkage(squareform(distance_matrix), 'single')
```

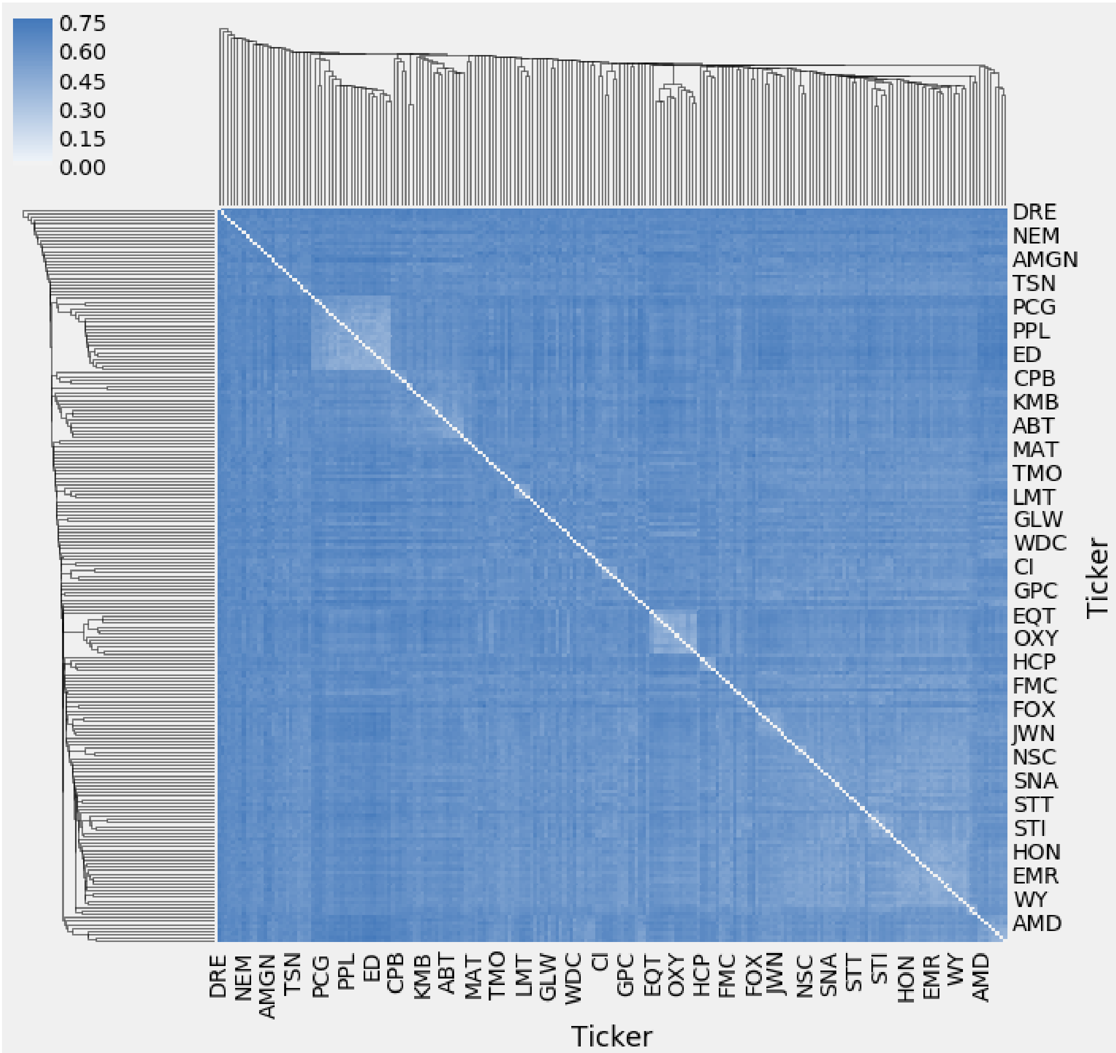
[12]:
```
sorted_idx = quasi_diagonalize(linkage_matrix)
```

1.5.1 Plot Cluster Map

The linkage_matrix can be used as input to the seaborn.clustermap function to visualize the resulting hierarchical clustering. The dendrogram displayed by seaborn shows how individual assets and clusters of assets merged based on their relative distances.

Compared to a seaborn.heatmap of the original correlation matrix above, there is now significantly more structure in the sorted data.

[13]:
```
clustergrid = sns.clustermap(distance_matrix,
                             method='single',
                             row_linkage=linkage_matrix,
                             col_linkage=linkage_matrix,
                             cmap=cmap, center=0);
```

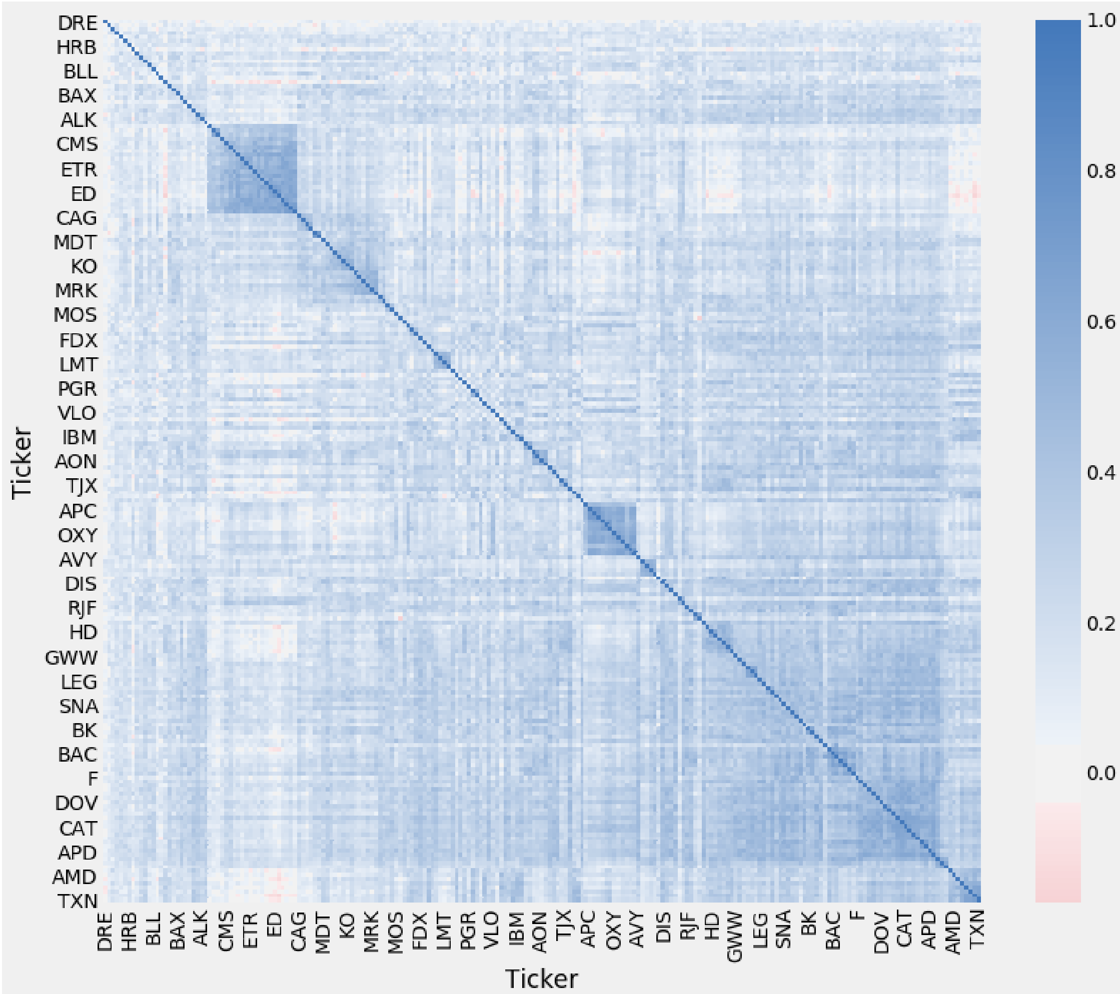


[14]:
```
clustergrid.savefig('clustermap.png', dpi=600)
```

[15]:
```
sorted_idx = clustergrid.dendrogram_row.reordered_ind
```

[16]:
```
sorted_tickers = corr.index[sorted_idx].tolist()
```

[17]:
```
clustered_assets = corr.loc[sorted_tickers, sorted_tickers]  # reorder
fig, ax = plt.subplots(figsize=(12,10))
sns.heatmap(clustered_assets, center = 0, cmap = cmap, ax=ax)
fig.tight_layout()
```



1.6 Compute Allocation

Using the tickers sorted according to the hierarchy induced by the clustering algorithm, HRP now proceeds to compute a top-down inverse-variance allocation that successively adjusts weights depending on the variance of the subclusters further down the tree. To this end, the algorithm uses bisectional search to allocate the variance of a cluster to its elements based on their relative riskiness.

[18]:
```
def get_cluster_var(cov, cluster_items):
    """Compute variance per cluster"""
    cov_ = cov.loc[cluster_items, cluster_items]  # matrix slice
    w_ = get_inverse_var_pf(cov_)
    return (w_ @ cov_ @ w_).item()
```

[19]:
```
def get_hrp_allocation(cov, tickers):
    """Compute top-down HRP weights"""
```

```
    weights = pd.Series(1, index=tickers)
    clusters = [tickers]  # initialize one cluster with all assets

    while len(clusters) > 0:
        # run bisectional search:
        clusters = [c[start:stop] for c in clusters
                    for start, stop in ((0, int(len(c) / 2)),
                                        (int(len(c) / 2), len(c)))
                    if len(c) > 1]
        for i in range(0, len(clusters), 2):  # parse in pairs
            cluster0 = clusters[i]
            cluster1 = clusters[i + 1]

            cluster0_var = get_cluster_var(cov, cluster0)
            cluster1_var = get_cluster_var(cov, cluster1)

            weight_scaler = 1 - cluster0_var / (cluster0_var + cluster1_var)
            weights[cluster0] *= weight_scaler
            weights[cluster1] *= 1 - weight_scaler
    return weights
```

[20]:
```
hrp_allocation = get_hrp_allocation(cov, sorted_tickers)
```

The resulting portfolio allocation produces weights that sum to 1 and reflect the structure present in the correlation matrix.

[21]:
```
hrp_allocation.sort_values().plot.bar(figsize=(15,4));
```