PYTHON

Hierarchical clustering of Exchange-Traded Funds

pmercatoris
(https://quantdare.com/author/pmercatoris/) *13/12/2017*

Clustering has already been discussed in plenty of detail, but today I would like to focus on a relatively simple but extremely modular clustering technique, hierarchical clustering, and how it could be applied to ETFs. We'll also be able to review the Python tools available to help us with this.

Clustering suitability

First of all, ETFs are well suited for clustering, as they are each trying to replicate market returns by following a market's index. We can therefore expect to find clear clusters. The advantage of using hierarchical clustering here, is that it allows us to define the precision of our clustering (number of clusters) after the algorithm has run. This is a clear advantage compared to other unsupervised methods, as it will ensure impartial and equidistant clusters, which is important for good portfolio diversification.

The data used here are the daily series of the past weeks' returns. This ensures stationarity and allows for better series comparison. The prices used are all in US dollars from September 2011 to December 2017, to try and capture different market conditions while keeping a high number of ETFs (790).

How to begin

The first step is to calculate all the pairwise distances between the series. The Scipy package provides an efficient implementation to do this with the pdist function, and includes many distances.

Here I compared all the applicable ones to calculate distances between 2 numerical series. To compare them, I decided to use the cophenetic distance, which is (very briefly) a value ranging from 0 to 1 and allows us to determine how well the pairwise distances between the series compare (correlate) to their cluster's distance. A value closer to 1 would result in better clustering, as the clusters are able to preserve original pairwise distances.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

%matplotlib inline
```

```
from scipy.spatial.distance import squareform, pdist

distances=["euclidean", "sqeuclidean", "cityblock", "cosine", "hamming",
           "chebyshev", "braycurtis", "correlation"]
for distance in distances:
    dist = pdist(returns.T.values, metric=distance)
    print(distance, hcl.cophenet(hcl.ward(dist), dist)[0])
```

```
euclidean 0.445105212991
sqeuclidean 0.636766347254
cityblock 0.449263737373
cosine 0.852746101706
hamming -0.148087351237
chebyshev 0.480135889737
braycurtis 0.486277543793
correlation 0.850386271327
```

Here, the cosine (as well as the correlation) distances worked best. It's then time to apply the agglomerative hierarchical clustering, which is done by the linkage function. There are a few methods for calculating between cluster distances, and I invite you to read further about them in the description of the linkage function. In this case, I will use the ward method, which minimises the overall between-cluster distance using the algorithm Ward variance minimisation, and is often a good default choice.

```
import scipy.cluster.hierarchy as hcl

dist = pdist(returns.T.values, metric="cosine")
Z = hcl.ward(dist)
```

The resulting matrix Z is informing each step of the agglomerative clustering by informing the first two columns of which cluster indices were merged. The third column is the distance between those clusters, and the fourth column is the number of original samples contained in that newly merged cluster. Here are the 3 last merges:

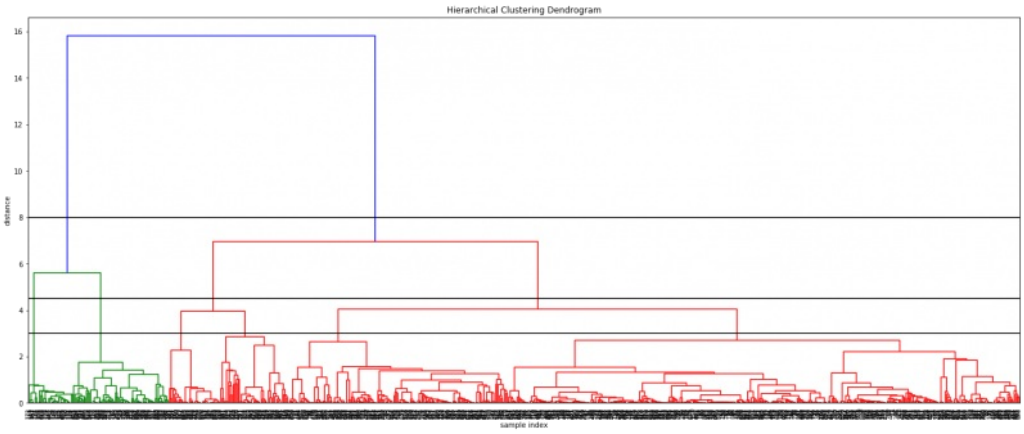
```
print(np.array_str(Z[-3:], precision=1))
```

```
[[ 1522.   1564.      5.6   112. ]
 [ 1574.   1575.      7.    678. ]
 [ 1576.   1577.     15.8   790. ]]
```

Visualising the clusters

A good way to visualise this is with a dendrogram, which shows at which inter-cluster distance each merge occurred. From there, it is possible to select a distance where clusters are clear (indicated with the horizontal black lines).

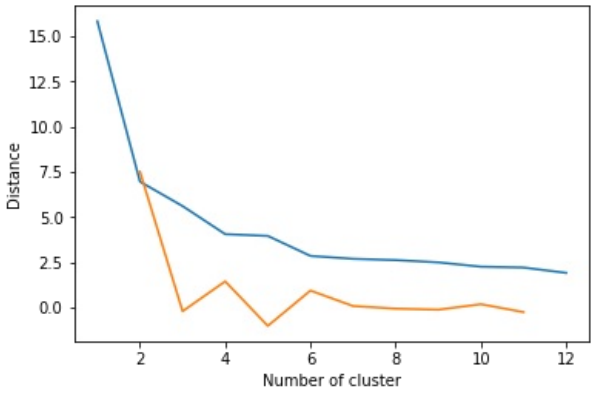
```
plt.figure(figsize=(25, 10))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('sample index')
plt.ylabel('distance')
hcl.dendrogram(
    Z,
    leaf_rotation=90.,
    leaf_font_size=8.
)
plt.axhline(y=8, c='k')
plt.axhline(y=3, c='k')
plt.axhline(y=4.5, c='k')
plt.show()
```



As we can see, the clear number of clusters appear to be 2, 4 and 6 (depending on the desired level of detail).

Another, more automatic, way of selecting the cluster number is to use the Elbow method and pick a number where the decrease of inter-cluster distance is the highest, which seems to occur at 2 clusters. However, this is probably too simplistic and we can also see this occur at 4 and at 6, as shown by the second derivative of those distances (in orange).

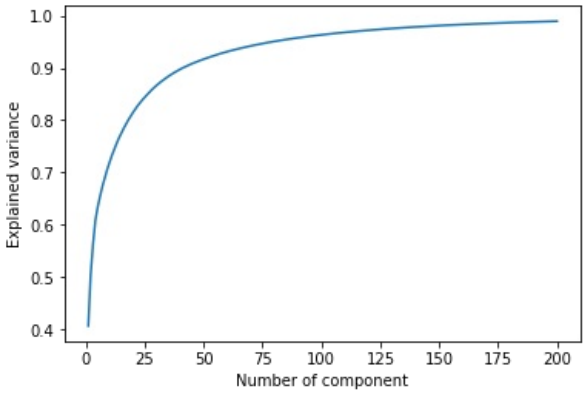
```
last = Z[-12:, 2]
last_rev = last[::-1]
idxs = np.arange(1, len(last) + 1)
plt.plot(idxs, last_rev)

acceleration = np.diff(last, 2)  # 2nd derivative of the distances
acceleration_rev = acceleration[::-1]
plt.plot(idxs[:-2] + 1, acceleration_rev)

plt.ylabel("Distance")
plt.xlabel("Number of cluster")
plt.show()
```



Plotting the results

In order to plot the results, it is necessary to carry out some dimensionality reduction. For this, I have decided to use TSNE as it's particularly efficient to plot over 2 dimensions. However, it's a good idea to first reduce the dimensions to a reasonable number, using PCA, when the number of features is too high. This is certainly the case for time series, where each daily return is considered a dimension.

```
from sklearn.decomposition import PCA
pca = PCA().fit(returns.T)
plt.plot(np.arange(1,len(pca.explained_variance_ratio_)+1,1)[:200],pca.ex

plt.ylabel("Explained variance")
plt.xlabel("Number of component")
plt.show()
```



In order to get a minimum of 95% of the variance explained, it is necessary to use a minimum of 80 components.

```
np.where(pca.explained_variance_ratio_.cumsum()>0.95)[0][0]+1
```

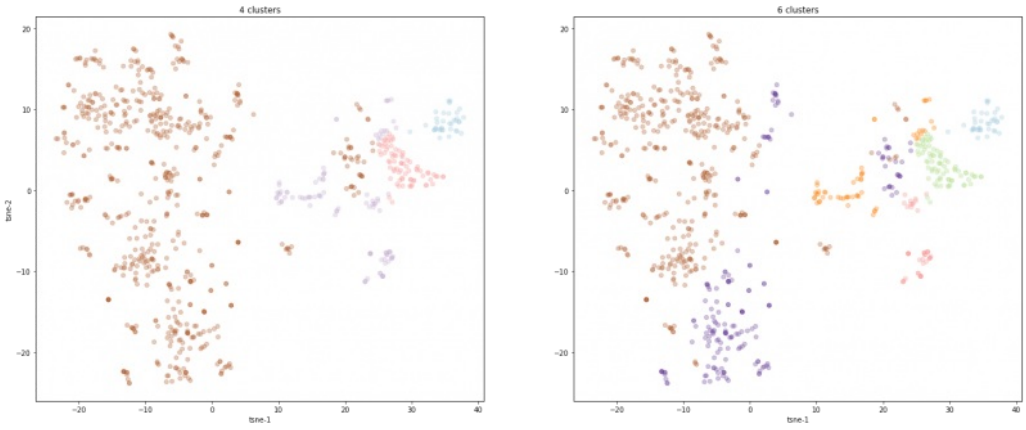
```
80
```

With those reduced dimensions, we can now use the TSNE and reduce it further to 2 dimensions. I highly recommend this read, to see how to fine-tune it (the article has some very nice interactive visualisation).

```
from sklearn.manifold import TSNE
pca = PCA(n_components=80).fit_transform(returns.T)
X2 = TSNE(n_components=2, perplexity=50, n_iter=1000, learning_rate=
```

Finally, the clusters seem to be relatively cohesive when plotted on a two-dimensional space. So assets of each cluster are expected to behave similarly across observed market conditions since 2011. This assumption needs to be taken with a pinch of salt of course, but can help create a diversified portfolio by selecting assets from each cluster.

```
plt.figure(figsize=(25, 10))

k=4
clusters=fcluster(Z, k, criterion='maxclust')

plt.subplot(1,2,1)
ax = plt.gca()
ax.scatter(X2[:,0], X2[:,1], c=clusters, cmap='Paired', alpha = 0.3)
ax.set_title(str(k) + " clusters")
ax.set_xlabel("tsne-1")
ax.set_ylabel("tsne-2")



k=6
clusters=fcluster(Z, k, criterion='maxclust')

plt.subplot(1,2,2)
ax = plt.gca()
ax.scatter(X2[:,0], X2[:,1], c=clusters, cmap='Paired', alpha = 0.3)
ax.set_title(str(k) + " clusters")
ax.set_xlabel("tsne-1")
plt.show()
```




```
from collections import Counter
Counter(clusters)
```

```
Counter({1: 36, 2: 76, 3: 41, 4: 55, 5: 174, 6: 408})
```

RELATED POSTS

(https://quantdare.com/forecasting-sp-500-using-machine-learning/)

(https://quantdare.com/forecasting-sp-500-using-machine-learning/)

(HTTPS://QUANTDARE.COM/CATEGORY/PYTHON/)

Forecasting S&P 500 using Machine Learning

(https://quantdare.com/forecasting-sp-500-using-machine-learning/)

lcampos

(https://quantdare.com/author/lcampos/)

(https://quantdare.com/risk-parity-in-python/)

(https://quantdare.com/risk-parity-in-python/)

(HTTPS://QUANTDARE.COM/CATEGORY/RISK-MANAGEMENT/)

Risk Parity in Python

(https://quantdare.com/risk-parity-in-python/)

fjrodriguez2

(https://quantdare.com/author/fjrodriguez2/)

(https://quantdare.com/world-connections-indexes/)

(https://quantdare.com/world-connections-indexes/)

(HTTPS://QUANTDARE.COM/CATEGORY/PYTHON/)

World Connections using financial indexes

(https://quantdare.com/world-connections-indexes/)

psanchezcri

(https://quantdare.com/author/psanchezcri/)

(https://quantdare.com/calculate-monthly-returns-with-pandas/)

(https://quantdare.com/calculate-monthly-returns-with-pandas/)

(HTTPS://QUANTDARE.COM/CATEGORY/PYTHON/)

Calculate monthly returns...with Pandas

(https://quantdare.com/calculate-monthly-returns-with-pandas/)

mgreco

(https://quantdare.com/author/mgreco/)

ADD A COMMENT

Message