Stock Clustering for Portfolio Diversification

An Unsupervised Learning Approach

Mardan Kydybek ; Théo Maboge

HEC-Liège

January 12, 2026

Research Question & Motivation

Research Question:

- Can unsupervised learning help identify groups of similar stocks in order to improve portfolio diversification?

Motivation (Investor Perspective):

- A real investor aims to diversify risk across different assets.
- A portfolio composed of stocks from different sectors may still be poorly diversified.
- However, stocks from different sectors may still exhibit similar return behavior.
- During market stress, hidden correlations often appear.

Data Description

- Dataset composed of daily closing prices for a set 500 traded stocks.
- Time period: 2023-01-01 – 2026-01-01.
- Data source: financial market data (e.g. Yahoo Finance).
- Focus on stock returns rather than price levels.

Data Cleaning

- Financial price data contain missing values due to non-trading days or data availability.
- We apply a cleaning procedure to ensure consistency across all stocks.

```
# Downloading 'Close' prices only
raw_data = yf.download(TICKERS,
                       start=START_DATE,
                       end=END_DATE)['Close']

# Filtering stocks with more than 5% missing data
missing_threshold = 0.05
data_cleaned = raw_data.loc[:, raw_data.isnull().mean() < missing_threshold]

# Forward fill to handle minor technical gaps
data_cleaned = data_cleaned.ffill()

# Dropping remaining NaNs (typically at the beginning)
data_cleaned = data_cleaned.dropna()
```

Preprocessing: Log-Returns Calculation

- Stock prices are transformed into returns to capture relative price changes.
- Log-returns are preferred due to their time-additivity and better statistical properties.

```
# Calculating Daily Simple Returns
# Formula: (P_t / P_{t-1}) - 1
simple_returns = data_cleaned.pct_change()

# Calculating Daily Logarithmic Returns
# Log-Returns = ln(1 + Simple Return)
returns = np.log(1 + simple_returns)
```

Preprocessing: Standardization

- K-means and PCA require observations as rows and features as columns.
- Standardization ensures each trading day contributes equally to the distance metric.

```python
from sklearn.preprocessing import StandardScaler

# Transposing the data: assets as rows, trading days as columns
data_for_ml = returns.T

# Handling potential numerical artifacts
data_for_ml.replace([np.inf, -np.inf], np.nan, inplace=True)
data_for_ml.fillna(0, inplace=True)

# Initializing and applying the scaler
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data_for_ml)
```

PCA – Analysis of Explained Variance

- We run PCA without limiting the number of components to measure how much variance each component captures.
- We use the cumulative explained variance to choose a target (e.g., 80% or 90%).

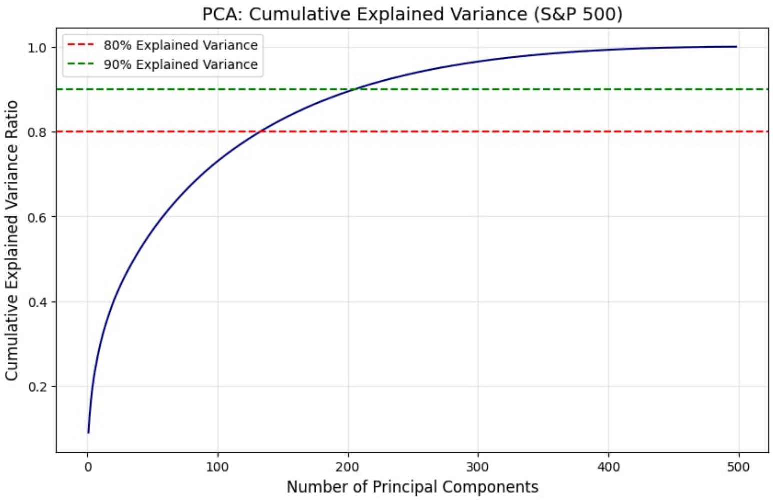
```python
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import numpy as np

# Initial PCA for explained variance analysis
pca_initial = PCA(n_components=None)
pca_initial.fit(scaled_data)

# Cumulative explained variance
cumulative_variance = np.cumsum(pca_initial.explained_variance_ratio_)

# Plot and save figure
```

PCA – Cumulative Explained Variance



PCA – Final Transformation (90% Variance)

- We select the smallest number of components that explains at least 90% of the variance.
- The dataset is reduced from the original space to a lower-dimensional representation.

```
from sklearn.decomposition import PCA
import numpy as np

# Selecting the optimal number of components (90% variance)
n_components_90 = np.argmax(cumulative_variance >= 0.90) + 1

print("PCA ANALYSIS COMPLETE")
print(f"Number of components to explain 90% variance: {n_components_90}")

# Reducing dataset to the optimal number of components
pca_final = PCA(n_components=n_components_90)
pca_features = pca_final.fit_transform(scaled_data)

print(f"Original shape: {scaled_data.shape}")
print(f"Reduced shape (PCA): {pca_features.shape}")
```

Elbow Method – WCSS Calculation

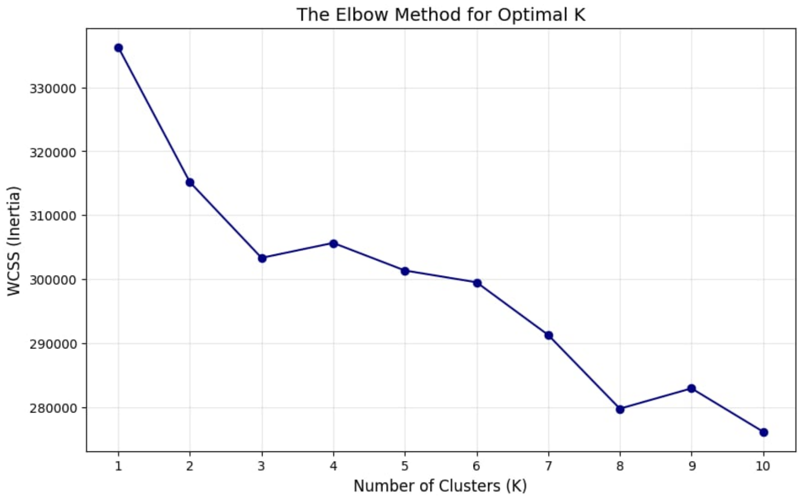
- We test different values of k (number of clusters).
- For each k, we compute the WCSS (inertia), i.e., within-cluster sum of squares.

```python
from sklearn.cluster import KMeans

wcss = []
K_RANGE = range(1, 11)

for k in K_RANGE:
    kmeans = KMeans(n_clusters=k, init='k-means++',
                    random_state=42, n_init='auto')
    kmeans.fit(pca_features)
    wcss.append(kmeans.inertia_)
```

Elbow Plot


The Elbow Method for Optimal K
WCSS (Inertia)
330000
320000
310000
300000
290000
280000
1
2
3
4
5
6
7
8
9
10
Number of Clusters (K)

Conclusion from the Elbow Plot

- Based on the elbow plot, we select the final number of clusters: K_{final}.

```
# Based on the visual analysis of the elbow plot
K_FINAL = 7
```

Final Clustering Model (K-Means)

- We apply the k-means algorithm using the optimal number of clusters ($K = 7$).
- Clustering is performed on the PCA-reduced feature space.

```
from sklearn.cluster import KMeans

# Final number of clusters
K_FINAL = 7

# Training the final k-means model
kmeans_final = KMeans(
    n_clusters=K_FINAL,
    init='k-means++',
    random_state=42,
    n_init='auto'
)

# Assigning cluster labels
cluster_labels = kmeans_final.fit_predict(pca_features)
```

Grouping and Cluster Sizes

- Each stock is mapped to a cluster label.
- Cluster sizes provide insight into market structure.

```
import pandas as pd

# Creating a DataFrame with cluster assignments
clustered_stocks = pd.DataFrame({
    'Ticker': final_tickers,
    'Cluster': cluster_labels
})
```

```
Final mapping complete. Shape: (498, 2)

Cluster
0       1
1      10
2       3
3     180
4     182
5      62
6      60
```

Portfolio Interpretation

- Clusters group stocks with similar return dynamics, not necessarily from the same sector.
- Stocks within the same cluster tend to be more correlated.
- Diversification can be improved by selecting stocks across different clusters.
- Cluster-based diversification can complement traditional portfolio construction methods.
- Interpretation:
  - **Large clusters**: market core, high exposure to common market factors.
  - **Medium clusters**: sectoral or factor-driven behavior.
  - **Small clusters**: niche or idiosyncratic risk profiles.

Supervised Learning: Cluster Prediction

- Objective: predict cluster membership for new stocks.
- Features: PCA-reduced returns.
- Target: cluster labels from k-means.

```
# Train-test split
X = pca_features
y = cluster_labels
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Logistic Regression
lr_model = LogisticRegression(max_iter=1000)
lr_model.fit(X_train, y_train)
lr_acc = accuracy_score(y_test, lr_model.predict(X_test))

# Random Forest
rf_model = RandomForestClassifier(n_estimators=100)
rf_model.fit(X_train, y_train)
rf_acc = accuracy_score(y_test, rf_model.predict(X_test))
```

Supervised Learning Results

- Supervised models can successfully predict cluster membership.
- Logistic Regression performs slightly better in this setting.

```
Logistic Regression Accuracy: 0.85
Random Forest Accuracy: 0.80
```

Conclusion

- We use unsupervised learning to identify groups of stocks with similar return behavior.
- The clustering results highlight hidden correlations beyond traditional sector classifications.
- Cluster-based selection can improve portfolio diversification and risk management.
- **Key takeaway:**
  - Unsupervised learning provides a data-driven complement to traditional portfolio construction.