Data Structures and Algorithms

Final Project: Stock Market Clustering

1 Introduction

In this assignment, you will develop an algorithmic approach to analysing the similarity of different stocks via cluster analysis.

The general purpose of cluster analysis is dividing a set of objects into groups that are "similar" to each other. It is a widespread tool used for exploratory data analysis in diverse fields in both science and business. For example, in marketing analytics, cluster analysis is employed to group consumers into segments based on their characteristics or *features*, such as age, post code, purchase history, etc. These features are somehow aggregated to compare the similarity between consumers. Based on this similarity, a clustering algorithm then divides the consumers into segments.

You will apply this idea on stock market data to identify groups of stocks that perform similarly over time. There are many reasons for grouping stocks together, such as analysing trading strategies, risk management, or simply presenting stock market information. Publicly traded companies are often grouped together by simple features such as the industry they operate in (eg tech companies or pharma companies), but here you'll take a data-driven approach, grouping together stocks that perform similarly over time.

Cluster analysis is an umbrella term for many different algorithmic approaches. Here you'll develop one that's based on the concept of *greedy* algorithm design, specified below. You'll also have the opportunity to explore other approaches using Python libraries.

Instructions. Complete the project in your assigned study group following the instructions below. The project is due on Monday 18/10 at 4pm.

You have two options for delivering the project.

1. A single stand-alone Jupyter Notebook, which contains your analysis and discussion of the problem, as well all the code required to re-run your analysis. This is the preferred approach, and submitting a Notebook will have a positive impact on your grade.

2. A doc/PDF report of your analysis and discussion, plus a separate Python file containing all the code required to re-run your analysis. The document should not exceed four pages in length, and should preferably be significantly shorter.

There is a tradeoff associated with this choice: While the Notebook is a very useful tool to master going forward, it has a small learning curve to it. However, it is easy to find many online tutorials for writing Jupyter Notebooks.

The project will be graded on (1) the correctness of your code, (2) the quality and depth of your analysis, and (3) the clarity of your writing and the presentation of your code. 80% of the grade will be based on the "main" part of the project, while the remaining 20% relate to the "extra" part, as specified below.

Notice that the project has several parts that can be worked on simultaneously. After reading through all the instructions carefully, you may want to modularize the development and divide some of the work within your group.

2 The main part

Data. In the homework folder, you will find the following `csv` files:

- `SP_500_firms.csv`. This file contains all firms currently included in the S&P 500 index.
- `SP_500_close_2015.csv`. This file contains daily stock price data of the firms listed in the previous file for 2015 (without some firms for which data was not available for the entire year).

The price data, normalised to start at 1 for each firm, is presented in the figure below.

Analysis approach. We want to analyse which stocks perform similarly over time, and then gather them into groups based on this information. In the terms of the introductory discussion in the beginning, our features are the daily stock prices, and we seek to cluster the stocks based on similarity in these features.

But what does it mean when we say "performing similarly"? We don't want to look at just how much each stock gained over the year: Even if the yearly returns of two stocks are the same, they might have performed very differently during the year. For example, suppose one stock went steadily up, and another one down during the entire year. But then in the end, the lower-performing

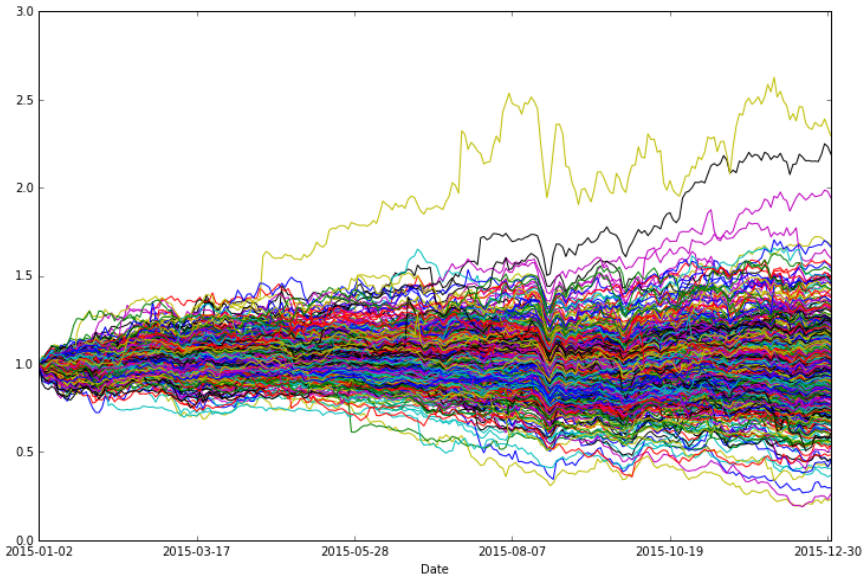


Figure 1: The S&P 500 companies' prices in 2015.

stock jumps up in one day. Clearly we can't conclude that the stocks have been performing similarly throughout.

Instead, a common way to analyse performance similarity is looking at daily stock price movements: if two stocks tend to have very similar movements on day-to-day basis, we can infer that they are similar. The day-to-day movement of a stock price p_t is typically measured as its *percentage change* compared to the previous price:

$$x_t = \frac{p_t - p_{t-1}}{p_{t-1}}. \tag{1}$$

This is called the daily return of the stock. Notice that the return may be either positive or negative.

To calculate the similarity between stock price movements, we often measure the *correlation* between the returns of the two prices. Correlation describes a statistical relationship between the two returns: if they prices move very similarly, their returns' correlation is close to one; if they tend to make exactly the opposite movements (ie one price moves up, the other one down), the correlation is close to -1. If there is no clear statistical relationship between the movements of two stock prices, the correlation in their returns is close to zero.

For a sample of stock price returns x, y with observations for n days, the correlation between x

and y can be calculated as:

$$r_{xy} = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{n s_x s_y} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\,\sqrt{n\sum y_i^2 - (\sum y_i)^2}}. \tag{2}$$

Here $\bar{x}$ refers to the average value of x over the n observations, and s_x to its standard deviation.

Thus, based on time series (eg over a year) of stock returns we can calculate a single correlation value for each pair of stocks, for example MSFT (Microsoft) and AAPL (Apple). This would give us a score for the similarity between the two stocks in this time period.

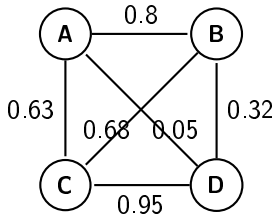
How can we use this similarity information for clustering? Suppose we have access to all correlations between stock returns in S&P 500. We can think of this as a *graph* as follows. The nodes of the graph are the stocks (eg MSFT and AAPL). The edges between them are the correlations, which we have between each stock, where the value of the correlation is the edge weight. Notice that since we can calculate all such correlations, this is a *dense* graph, where all possible edges exist.

We thus have a graph representing pairwise "similarity" scores in correlations, and we want to divide the graph into clusters. There are many possible ways to do this, but here we'll use a *greedy* algorithm design. The algorithm is as follows:

1. Sort the edges in the graph by their weight (ie the correlation)
2. Create a single-node set from each node in the graph
3. Repeat k times:
   (a) Pick the highest-weight edge (this is a "greedy" choice)
   (b) Merge the sets containing the source and the destination of the edge
   (c) Repeat from (a) with the next-highest weight edge
4. Return the remaining sets

What does the algorithm do? Essentially, it first initializes a graph with no connections. Then in the main loop, it runs through the k highest-weighted edges, and adds connections at those edges. The result is "groups" determined by the highest correlations between the stock returns. These are your stock clusters.

For example, in the correlation graph below, the algorithm would initialize four sets of one node each. It would then first connect C and D, resulting in just three sets. Then it would connect A and B, resulting in two sets of two nodes each.



Task details. Your task is to implement the above analysis in Python. This is the "main" part of the project. The "extra" part is described further below. The main project can be divided into the following parts:

1. **Stock returns.** Calculate the daily returns for all stocks in the data. Discuss the returns over the year: what companies experienced the maximum and minimum daily returns - can you find the reasons for these? Which companies performed overall best and worst over the year? Which companies exhibited most and least volatility, as measured by the standard deviation of their returns over the year?

2. **Correlations.** Calculate the correlations between all stocks in the data using returns. Provide a convenient way for a user to print out two companies' full names and a correlation between their returns using your data. Also provide a convenient way to print out the top and bottom correlated companies for any given company. Use this to comment on the following companies in the tech sector: Amazon, Microsoft, Facebook, Apple, and Google — which companies are they most closely related to in terms of highest correlations? Would you have expected the results you see?

3. **Clustering algorithm.** Implement the above algorithm in Python. Discuss the results for different values of k. Do some detective work: what is the greedy algorithm called? In what other graph problem is it often used? How are the problems related? (Hint: the algorithm is mentioned on the Wikipedia page for greedy algorithms.) Do the resulting clusters "make sense"? (You may need to search online what the companies do.) Verify that the stocks in your clusters perform similarly by plotting the returns and the (normalised) stock prices for some of the clusters.

3 Suggestions and guidelines for your Python code

Please read this section carefully before you start coding.

- **Open problem.** This is a fairly open problem, intended to roughly correspond to real-world problem solving using Python. That is, you'll be working on a new problem AND using a few new methods. You can choose to follow the hints below or take a different approach of your own. And even though your eventual final submission may not be that long in terms of lines of code, be prepared for quite some work in figuring things out — so start early.

- **Loading the data & analysis tools.** You have two options, which involves a tradeoff. You can either do all the analysis using built-in standard Python data structures like lists and dictionaries, or you can use the `pandas` library for data analysis. With `pandas` you'll need to learn some new commands to deal with data. But `pandas` comes with many useful built-in functions eg for calculating correlations, which will save you lots of looping. For loading the data into Python, the Python file on the Hub provides instructions for both approaches. It may even be a good idea to diversify such that some people start doing the analysis with `pandas` and others without...

- **Using Python libraries.** If you do decide to use Python libraries such as `pandas/numpy`, and find a function that calculates something (eg correlations) directly, please demonstrate your understanding of the concept by also using Python to calculate some correlations directly from the definition, and making sure you get the same result. For the algorithm part, please do not use any readymade Python code or libraries.

- **Focus first on getting something running.** Then you can think about whether it can be made cleaner etc. For the algorithm part, you may find it helpful to define a "toy" version of the problem to test your solution.

- **Once you have something running.** It is usually helpful to write code snippets as functions and then call them. Similarly to the homeworks, try to divide your code into small parts which you can wrap into functions for easy reuse.

- **Presenting results/data.** You may sometimes decide that it's easier to present your results in a graph rather than just words — you can use the `matplotlib` Python library for creating graphs.

- **You're working as a group.** You may divide tasks but do keep everyone on board on all aspects of the project. Make sure to comment your code and clearly communicate what your code does – this makes it easier for others to use it.

- **Hints for using `pandas/numpy/matplotlib`.** The Python file provided has a bunch of useful commands, as do your tutorials. For anything not covered, search online or read through the excellent official tutorial: http://pandas.pydata.org/pandas-docs/version/0.18.1/tutorials.html.

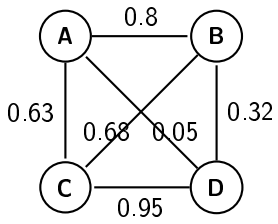
- **Hints for implementing the algorithm.** Even if you're using `pandas` for the previous analysis, it may be easier to move to standard Python data structures for this part.

  - To be able to sort the edges, you'll need to represent them in eg a list. One way to do this is to create a list where each element is a tuple `(weight,source,destination)`, and using `sorted()` on the list. You may need to do some looping to get to this list representation...
  - You'll need to think about how to model the sets of nodes in the algorithm, and how to merge them. With every edge, you'll need to figure out the sets in which the source node and the destination nodes are. One way to do it is to simply use Python sets and loop through all of them until you find the correct set. You'll then need to merge these sets, ie add all items of one set to the other and then delete the set. For large datasets, this kind of looping becomes slightly cumbersome, so you may prefer the approach below instead.
  - An alternative, and preferred, approach to modelling and merging sets would work as follows:
    * Model the sets as dictionaries of linked nodes. Initialize a dictionary called `nodePointers` where the keys are each node, with the corresponding value the node itself. That is, initially each node points to itself. When you connect sets through an edge, you update these pointers.
    * More specifically, when considering an edge `(weight,source,destination)`, look at the source node. If it does not point to itself, look at what node it points to and repeat until you reach a node that points to itself. Do the same for the destination node. Then make one of these 'bottom' nodes point to the other. Notice that several

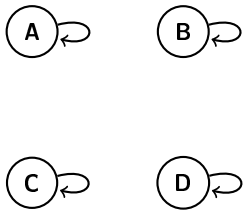
nodes can point to the same node in this way. This procedure performs set merging: all nodes from which you can follow links to the same bottom node are in the same set. See the figure below for examples.

* But how do you recover the sets from this procedure? There are several ways to do this, but you could for example store information on which nodes are at the "starting" end of the resulting trees, or you could store links in both directions for each node. In either case, you'd then need to write a loop to recover the sets.

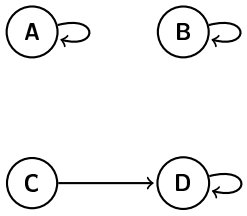
Consider the situation in the example above, repeated here for convenience.



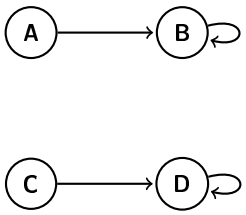
We start with four nodes that point to themselves, ie the dictionary `{'A':'A','B':'B','C':'C','D':'D'}`.



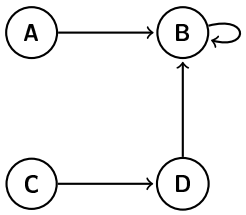
We first look at the edge between C and D. Now one of them will point to the other: in the dictionary, `{'A':'A','B':'B','C':'D','D':'D'}`.



Next we consider the edge between A and B: `{'A':'B','B':'B','C':'D','D':'D'}`..



Next we consider the edge between C and B. We find the bottom nodes of both chains of links by going through the dictionary. That is, if we start from C, we find D, then that refers to itself so it is at the bottom. We then make one point to the other. `{'A':'B','B':'B','C':'D','D':'B'}`.



Now this is just a single set. But how do you recover the set? One way to do this would be to store some other information during the algorithm, such as the 'starting nodes' to which no node links, ie A and C above. For example, every node could start out labeled as a 'starting node', but cease being one if another node points to it. Then you could write some kind of a separate loop that finds everything it can find starting from all remaining 'starting nodes' in order to create sets. How would such a loop create a single set out of the above nodes?

In implementing the above steps, it may be useful to write helper functions — for example a function to find the bottom node.

4 The extra part

This part is worth up to 20% of your grade. Depending on your interests, you may work on either subsection below, or both. You might go deeper into one question than another, but for an outstanding grade, you should have at least some discussion on both.

4.1 In-depth analysis

The project is "open" in the sense that you can probably think of further interesting questions to look into based on returns, correlations, and clusters. This is not required but being creative and

going further than the above questions will make your work stand out. You can explore one or several of the ideas below, or come up with questions of your own.

Depending on your interests, you might look at different things. For example, when researching the algorithm, you might be interested in its complexity, and how to improve your implementation's efficiency. There is scope to go quite a bit deeper here.

If you're more interested in the financial applications of clustering, there are also opportunities to think about further steps. For example, some people claim that you can derive trading strategies based on clustering — that often one of the stocks in a cluster is a leader and the others follow that price. If this is true, you could track the price of the leader stock and then trade the other stocks in the cluster based on changes in the leader's price. Do you think this would make sense? Do you have an idea on how to identify a leader stock?

You might also want to repeat the analysis for different time periods. You would be able to do this by looking at the code for the second homework to figure out how to read data from Yahoo Finance using pandas, and going through the process for all companies in the csv file for another time period. Perhaps you could explore for example how correlations between companies have changed over time, or how clusters found by your algorithm change over time.

4.2 Exploring other clustering methods

You've used just one approach to clustering, and arguably not the best one. Research clustering algorithms and libraries to apply them in Python. Discuss some other algorithms that could be used, and how they differ from the one you've implemented. Look at the Python library `scikit-learn`. How would you apply the clustering algorithms provided by the library to stock price data? Would you need to develop new metrics other than correlations? If you want to go even further, try running some of these other clustering algorithms on your data, and report the results. Start from here: http://scikit-learn.org/stable/modules/clustering.html#clustering — you'll find a stock market example there too. For future reference, you may also find other interesting machine-learning tools for both stock market analysis or other analytics purposes.