HierarchicalClusteringAlgorithm

November 12, 2018

1 Clustering Challenge: Partitioning Two-Thousand Stocks by their return series

Attached .csv file contains return series of 2,000 stocks(columns) for 3,000 days(rows). Please make 10 partition groups of 2,000 stocks satisfying following conditions.

1. Each partition group has 200 stocks.
2. Minimize inter-group return correlation and maximize intra-group return correlation. Objective function will be CARS/TICS.
3. Intra-group return correlation TICS is defined as below. For each partition group with label k, 200x200 correlation matrix IC(k) can be calculated. ICS(k) is a summation of all elements in IC(k). TICS = ICS(1) + ICS(2) + ... + ICS(10)
4. Inter-group return correlation CARS is defines as below. AR(k) is a return series obtained by averaging over 200 stocks' return series. Then AR(k) is a vector with 3,000 elements(3,000 days). CAR is 10x10 correlation matrix of [AR(1), AR(2), ... AR(10)] and CARS is a summation of all elements in CAR matrix.

1.0.1 Intuition:

For normal Machine Learning clustering algorithms, such as K-Means algorithm, the absolute co-ordinates of points are needed. The correlation problem can be seen as a clustering problem in which only relative distances of points are provided.Without absolute co-ordinates, normal clustering algorithms are not applicable in solving this problem. Also, directly importing existing Machine Learning Python packages is not applicable, and we need to re-write a class based on our unique requirements.

Hence, I re-wrote Hierarchical Clustering algorithm to solve the problem.

2 Challenge 1: No requirement of same sizes of clusters.

2.0.1 1.1.1 Description

You are required to use Python 2.7 or C++ as your programming language. You mayor may not automate all the process which the question is asking you to solve. If you think a certain part takes longer time foryou to code, you may skipthat part and manually compete the task. Task is mainly on data science problem.

Please download daily price data of about 500 stocks in S&P 500 to calculate daily return series of 500 stock in S&P500. Now you can calculate correlation between 500 stocksfrom return series. This will be written as 500 x500 matrix with ones along the diagonal.

Let's define Corr(A,B) as the correlation coefficient between A and B derived from their return series. Then, build clusters satisfying the following conditions: Correlation between any pair of stock in the same cluster must be higher than pair of stocks from different cluster. For example, A and B are from one cluster and C are foranother cluster; Corr(A,B) >= Corr(A,C) and Corr(A,B) >= Corr(B,C)

Then, you are now supposed to find clusters defined above to optimize the following objective functions.

2.0.2 1.1.2 Requirements:

a. Summation over Correlations between stocks in the same cluster is maximized.This sum of correlation is sum of all the elements in correlation matrix.I.e.ifyou constructed acluster-with100 stocks.You can calculate 100 x 100 correlation matrix and sum up all the elements to get "sum of correlation". In other word, stocksin the same cluster must be highly correlated.

b. The number of clusters is not restricted as long as you have more than four clusters. But standard deviation of the numbers of clusters is to be minimized. For example, clusters of 100 stocks, 100 stocks, 100 stocks and 200 stocks are better than those of 300 stocks, 160 stocks, 35 stocks and 5 stocks.

c. You can calculate the average of returns of stocks in the same cluster. If you have four clusters, you will have four average returns, from which 4x4 correlation matrix can be calculated. Summation over Correlation between average returns of clusters is minimized. In other word, clusters must not be correlated in termsof average return.

2.0.3 1.2 Data, Code and Answer

503 stocksfrom S&P 500 have available daily prices fromYahoo! Finance and Quandl databases.

Intra-group return correlation TICS is defined as below. For each partition group with label k, 200x200 correlation matrix IC(k) can be calculated. ICS(k) is a summation of all elements in IC(k). TICS = ICS(1) + ICS(2) + ... + ICS(10).

Inter-group return correlation CARS is defines as below. AR(k)is a return series obtained by averaging over 200 stocks'return series.ThenAR(k)is a vector with 3,000 elements(3,000 days). CAR is 10x10 correlation matrix of [AR(1),AR(2), ... AR(10)] and CARS is a summation of all elements in CAR matrix.

To find clusters with maximum TICS, maximum CARS and similar sizes, the results are as below.

1.2.1 Code for downloading stock returns from Yahoo! Finance, Quandl and IEX datasets.

```
In [1]: import os
        import fix_yahoo_finance as yf
        import pandas as pd
        import pandas_datareader.data as web
        from datetime import datetime,date
        import numpy as np
```

```
        #Download daily prices from Yahoo! Finance
        def yahooFinanceDownload(ticker,filepath,start_date,end_date):
            prices = yf.download(ticker, start=start_date, end=end_date)
            if len(prices):
                out_filename = filepath + ticker + '.csv'
                prices.to_csv(out_filename)

        def quandlDownload(ticker,filepath):
            start = datetime(2000,1,1)
            end = date.today()
            try:
                prices = web.DataReader(ticker, 'quandl', start, end)
            except:
                pass
            else:
                out_filename = filepath + ticker + '.csv'
                prices.to_csv(out_filename)

        def collectStockName(filepath):
            stock_list=[]
            for root, dirs, files in os.walk(filepath):
                if files:
                    for f in files:
                        if 'csv' in f:
                            stock_list.append(f.split('.csv')[0])
            return stock_list

        def getDownloadList(ticker_list,filepath):
            download_list = collectStockName(filepath)
            unsuccess_list = [ticker for ticker in ticker_list if ticker not in download_list]
            return download_list, unsuccess_list
```

2.0.4 1.2.2 Algorithm

In [2]:
```
        class Node(object):
            def __init__(self,index_list,position=None,count=1):
                self.count = count
                self.index_list = index_list
                self.position = position

        class EqualSizedClustering(object):
            def __init__(self,corr_matrix,k,node_size=None):
                self.k = k
                self.corr_matrix = corr_matrix
                self._len = np.shape(self.corr_matrix)[0]
                if node_size is None:
                    self.node_size = int(self._len/self.k)
                else:
```

```
            self.node_size = node_size
        self.nodes = []

        self.all_index = [i for i in range(self._len)]
        self.df = None

    def initial_fit(self):
        nodes = []
        for i in self.all_index:
            nodes.append(Node(index_list=[i],position=i,count=1))
        position = -1
        distances = {}

        new_nodes = []

        while True:
            max_dist = 0
            nodes_len = len(nodes)

            for i in range(nodes_len-1):
                for j in range(i+1,nodes_len):
                    node_a,node_b = nodes[i],nodes[j]
                    position_pair = (node_a.position,node_b.position)
                    if position_pair not in distances:
                        distances[position_pair] = self.node_distance(node_a,node_b)
                    d = distances[position_pair]
                    if max_dist < d:
                        max_dist = d
                        max_i = i
                        max_j = j
                        max_node_a = node_a
                        max_node_b = node_b

            node_a = max_node_a
            node_b = max_node_b

            new_index_list = node_a.index_list + node_b.index_list
            new_count = node_a.count + node_b.count
            new_node = Node(index_list=new_index_list,count=new_count,position=positio
            if len(nodes) > 2:
                nodes.pop(max_j)
                nodes.pop(max_i)
            else:
                node_a = nodes[0]
                node_b = nodes[1]
                new_index_list = node_a.index_list + node_b.index_list
                new_count = node_a.count + node_b.count
                new_node = Node(index_list=new_index_list,count=new_count,position=posi
```

```
                new_nodes.append(new_node)
                break

            if new_node.count >= self.node_size:
                new_nodes.append(new_node)
            else:
                nodes.append(new_node)
            position -= 1

            if len(nodes)+len(new_nodes) == self.k :
                if len(nodes) > 0:
                    new_nodes.extend(nodes)
                break


        self.nodes = new_nodes

    def adjust_size(self):
        exact_size = [node for node in self.nodes if node.count == self.node_size]
        large_size = [node for node in self.nodes if node.count > self.node_size]
        small_size = [node for node in self.nodes if node.count < self.node_size]
        from_large_to_small = []
        if large_size != []:
            for node in large_size:
                n = node.count - self.node_size
                remove_index = self.remove_smallest_index(node,n,
                                                          distance=False,all_index=sel
                #remove_index = self.remove_smallest_index(node,n)
                from_large_to_small.extend(remove_index)
                for index in remove_index:
                    node.index_list.remove(index)
                exact_size.append(node)
            for node in small_size:
                n = self.node_size - node.count
                temp_all_index = [index for index in self.all_index if index not in fr
                add_index = self.add_largest_index(node,from_large_to_small,n,distance

                #add_index = self.add_largest_index(node,from_large_to_small,n)
                node.index_list.extend(add_index)
                exact_size.append(node)
                for index in add_index:
                    from_large_to_small.remove(index)
        self.nodes = exact_size


    def setDataDf(self,df):
        self.df = df
```

```
def ICS(self,node):
    node_corr_matrix = self.corr_matrix[node.index_list][:,node.index_list]
    return node_corr_matrix.sum()

def TICS(self):
    TICS = 0
    for node in self.nodes:
        TICS += self.ICS(node)
    return TICS

def AR(self,node):
    temp = self.df[node.index_list]
    return temp.mean(axis=1)

def CARS(self):
    output = pd.DataFrame(index=self.df.index)
    id = 0
    for node in self.nodes:
        id += 1
        output[id] = self.AR(node)
    temp = np.array(output.corr())
    cars = temp.sum()
    return cars

def node_distance(self,node_a,node_b):
    temp = self.corr_matrix[node_b.index_list][:,node_a.index_list]
    return temp.mean()


def output_results(self,output_filename):
    label = 0
    index_list,label_list = [],[]
    for node in self.nodes:
        label += 1
        for index in node.index_list:
            index_list.append(index)
            label_list.append(label)
    output = pd.DataFrame({'Stock':pd.Series(index_list),'Label':pd.Series(label_li
    output.to_csv(output_filename)

def current_target_value(self):
    cars = self.CARS()
    tics = self.TICS()
    value = cars/tics
    return cars,tics,value

def remove_smallest_index(self,node,remove_num,distance=True,all_index=None):
    index_list = node.index_list
```

```
        remove_index = []
        contributions = np.zeros(len(index_list))
        for pos in range(len(index_list)):
            index = index_list[pos]
            temp_node = Node(index_list=[index])
            if distance:
                contributions[pos]= self.node_distance(node,temp_node)
            else:
                contributions[pos] = self.relative_contributions(node,index,all_index)
        for select_time in range(remove_num):
            pos = np.where(contributions==contributions.min())[0][0]
            remove_index.append(index_list[pos])
            contributions[pos] = 1000
        return remove_index

    def deleteIndex(self,del_index,index_list):
        new_list = []

        for index in index_list:
            if index != del_index:
                new_list.append(index)
        return new_list

    def relative_contributions(self,node,index,all_index):
        index_list = self.deleteIndex(index,node.index_list)
        return self.corr_matrix[[index]][:,index_list].sum()/self.corr_matrix[[index]]

    def add_largest_index(self,node,add_index_candidate,add_num,distance=True,all_index
        add_index = []
        contributions = np.zeros(len(add_index_candidate))
        for pos in range(len(add_index_candidate)):
            index = add_index_candidate[pos]
            temp_node = Node(index_list=[index])
            if distance:
                contributions[pos]= self.node_distance(node,temp_node)
            else:
                contributions[pos] = self.relative_contributions(node,index,all_index)
        for select_time in range(add_num):
            pos = np.where(contributions==contributions.max())[0][0]
            add_index.append(add_index_candidate[pos])
            contributions[pos] = -1000
        return add_index
```

2.0.5 1.2.3 Answers

To find clusters with maximum TICS, maximum CARS and similar sizes, the results are as below.

```
In [13]: def test_c201():
             corr = pd.read_csv('D:/c102/c102_corr.csv')
             corr_matrix = np.array(corr)
             eqc = EqualSizedClustering(corr_matrix=corr_matrix,k=4,node_size=125)
             eqc.initial_fit()

             df = pd.read_csv('D:/c102/c102_data.csv')
             df = df.set_index('Date')

             df.columns = [i for i in range(503)]

             eqc.setDataDf(df)
             current_value = eqc.current_target_value()
             output = pd.DataFrame()
             output.loc['Result', 'CARS'] = current_value[0]
             output.loc['Result', 'TICS'] = current_value[1]
             output.loc['Result', 'Target Value'] = current_value[2]
             print(output)

In [14]: if __name__== '__main__':
             test_c201()

             CARS          TICS  Target Value
Result  14.455305  32423.822044      0.000446
```

3 Challenge 2: Clusters must be with equal sizes

Attached .csv file contains return series of 2,000 stocks(columns) for 3,000 days(rows). Please make 10 partition groups of 2,000 stocks satisfying following conditions. 1. Each partition group has 200 stocks. 2. Minimize inter-group return correlation and maximize intra-group return correlation. Objective function will be CARS/TICS. 3. Intra-group return correlation TICS is defined as below. For each partition group with label k, 200x200 correlation matrix IC(k) can be calculated. ICS(k) is a summation of all elements in IC(k). TICS = ICS(1) + ICS(2) + ... + ICS(10) 4. Inter-group return correlation CARS is defines as below. AR(k) is a return series obtained by averaging over 200 stocks' return series. Then AR(k) is a vector with 3,000 elements(3,000 days). CAR is 10x10 correlation matrix of [AR(1), AR(2), ... AR(10)] and CARS is a summation of all elements in CAR matrix.

```
In [18]: def test_HFC():
             df = pd.read_csv('D:/54_hfc_20170614_comp.csv')
             df.columns = [i for i in range(2000)]
             corr = df.corr()
             corr_matrix = np.array(corr)
             eqc = EqualSizedClustering(corr_matrix=corr_matrix,k=10)
             eqc.initial_fit()
             eqc.setDataDf(df)
             current_value = eqc.current_target_value()
```

```
            output = pd.DataFrame()
            output.loc['Not Equal Sized Clusters', 'CARS'] = current_value[0]
            output.loc['Not Equal Sized Clusters', 'TICS'] = current_value[1]
            output.loc['Not Equal Sized Clusters', 'Target Value'] = current_value[2]
            eqc.adjust_size()
            value = eqc.current_target_value()
            output.loc['Equal Sized Clusters', 'CARS'] =value[0]
            output.loc['Equal Sized Clusters', 'TICS'] =value[1]
            output.loc['Equal Sized Clusters', 'Target Value'] = value[2]
            print(output)

In [23]: if __name__== '__main__':
            # This takes arount 40 minutes to get the results. Please be patient.\
            test_HFC()
```

```
                              CARS           TICS  Target Value
Not Equal Sized Clusters  67.048950  151963.456286      0.000441
Equal Sized Clusters      80.936464  135266.081639      0.000598
```