Discovering higher-order correlations between timeseries

Leguijt, L.

Award date:
2020

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Database Systems Group

Discovering higher-order correlations between timeseries

Lars Leguijt

Assessment committee:
dr. Odysseas Papapetrou (Supervisor)
dr. Nikolay Yakovets
dr.ir. Joaquin Vanschoren

Final version

Eindhoven, January 2020

Abstract

We study the problem of discovering higher-order correlations between timeseries. A higher-order correlation is a correlation between sets of timeseries. higher-order correlations can express higher-order interrelationships that are not captured by their pairwise relationships. There exist various correlation measures that can express these higher-order relationships, each with their own way of combining the variables.

Through a survey of the existing body of research on higher-order correlation discovery, we identify a number of these measures and their aggregation functions. We propose a general definition for the problem of higher-order correlation maximization that admits different measures of correlation C and their aggregation functions f. In addition, we propose CorrelationMiner, an application that runs on top of the Apache Spark[1] distributed computing framework.

It allows one to specify both distributed as well as local "solvers" that solve the higher-order correlation maximization problem for different measures C. We offer a number of local solvers for solving the higher-order correlation maximization problem when C is the Pearson correlation coefficient. Moreover, we implement a distributed solver that distributes the exhaustive solution to the maximization problem over a cluster of k worker nodes. Finally, we evaluate the performance of the developed solutions through a series of experiments using a dataset consisting of timeseries representing stock price values for a number of companies.

Acknowledgement

This thesis project has been carried out at Eindhoven University of Technology within the Database Group of the Department of Mathematics and Computer Science. I would like to thank my supervisor dr.Odysseas Papapetrou for his guidance and support while working on this project.

In addition, I would like to thank Hamid Shahrivari for his helpful advice regarding the development of the Apache Spark application that was part of this project. Finally, I would like to thank dr.Nikolay Yakovets and dr.ir.Joaquin Vanschoren for serving on my assessment committee.

Contents

Chapter 1

Introduction

Over the years, the amount of available timeseries data originating from sensor devices, climate data [2], genomics [3] and other sources has steadily increased. This increase in timeseries data also increased the need for extracting meaningful information from this data within a reasonable time frame. Correlation analysis is a statistical method of evaluating the strength of association between variables. It allows one to determine if there exist relationships between the variables under consideration.

Consider a set of timeseries $S = \{T_1, .., T_n\}$ with each timeseries $T_i \in \mathbb{R}^d$. We assume each $T_i \in S$ to be standardized, i.e. to have zero mean and unit variance. To determine the correlation between any two timeseries $T_i, T_j \in S$, we can use various measures of correlation. Perhaps the most well known of these, the Pearson correlation coefficient, can be used to express the pairwise correlation between $T_i, T_j \in S$ as $\rho(T_i, T_j)$ where $-1 \leq \rho(T_i, T_j) \leq 1$. We can consider a third timeseries $T_k \in S$ and compute its pairwise correlations with T_i and T_j as respectively $\rho(T_i, T_k)$ and $\rho(T_j, T_k)$. Continuing, we can compute the pairwise correlations of all pairs of timeseries in S and obtain a $n \cdot n$ *correlation* matrix Σ for which each entry (i, j) with $1 \leq i, j \leq n$ corresponds to the correlation coefficient $\rho(T_i, T_j)$ of timeseries pair (T_i, T_j). Note that Σ is usually used to refer to the *covariance* matrix. However, since we assume S to consist of standardized vectors the covariance matrix and the correlation matrix are equivalent.

As it turns out, the Pearson correlation measure has a property that we can use to express correlation beyond two timeseries. Specifically, using the property of bilinearity of covariance, the Pearson correlation measure can express the pairwise correlation between linear combinations of the timeseries in S. Thus, by linearly combining (subsets of) vectors in S, we are able to express pairwise correlation between arbitrarily sized subsets $X, Y \subseteq S$. For example, we could take the sum of our earlier timeseries vectors T_j, T_k and compute their combined correlation with T_i as $\rho(T_i, T_j + T_k)$. As we shall see in chapter 4, this specific relationship has been formalized into the notion of a tripole [2].

While there is currently no single agreed-upon term to describe this generalization of pairwise correlations to sets of variables, we deem the term *higher-order correlation* a suitable one [4]. This since it is able to capture higher-order interrelationships between sets $X, Y \subset S$ that are not apparent from the examination of Σ. To see this, consider the example in figure 1.1 taken from [2]. There, a graphical representation of three timeseries T_1, T_2, T_3 is depicted with pairwise correlations $\rho(T_0, T_1) = 0.42, \rho(T_0, T_2) = 0.26$ while $\rho(T_0, T_1 + T_2) = 0.83$. Thus, the higher-order correlation between sets $X = \{T_0\}, Y = \{T_1, T_2\}$ is significantly higher than the individual pairwise correlations, indicating a more complex relationship not explained by the pairwise correlations.

We note that the Pearson correlation measure is not the only measure that can express higher-order correlations. For example, the information theoretic measure of mutual information has also been generalized to a multivariate form called multi information or total correlation, which can express correlation for a set of random variables. In fact, as we shall see in chapter 4, the existing works on higher-order correlation discovery consider various generalizations of pairwise correlation measures as their measure of correlation [2] [5] [3] [6]. In addition, the input to measures that

can express higher-order correlations is also not limited to timeseries or real-valued vectors. They can be defined for arbitrary sets of input variables $X, Y \subset S$ given that there exists a correlation measure that operates on these sets.

Like pairwise correlations, higher-order correlations can be expressed using various (generalizations of) correlation measures and are not limited to a specific type of input. However, the cost of exhaustively computing all higher-order correlations in a given input dataset S is much larger than that of computing all pairwise correlations. As an example, let us consider again the Pearson correlation measure. The correlation matrix Σ will consist of $O(n^2)$ correlation coefficients. Since higher-order (Pearson) correlations are defined on a pair of sets of variables $X, Y \subset S$ we can instead consider computing all pairs of these subsets. Since the order of the variables is of no importance to the correlation coefficient, and we want to have that $X \cap Y = \emptyset$, we would thus compute all unordered pairs of disjoint subsets $X, Y \subset S$. This means that we will need to evaluate $O(3^n)$ pairs of subsets, which is a staggering amount and intractable except for small n.

Even when we restrict the cardinality of the subsets $X, Y \subset S$ in such a way that we consider no more than three vectors in total, there are still $O(n^3)$ possible triples to evaluate. This means that for a moderate amount of input variables, say $n = 100$, there are already $100^3 = 1000000$ (1 million) combinations to evaluate. The Pearson correlation measure expects a pairwise input, but even for correlation measures that do not expect a pairwise input the search space is still prohibitively large. Consider the multi-information measure for example, which expects a single subset of timeseries $X \subseteq S$. In this case, exhaustively computing all higher-order correlations requires evaluating all possible subsets of S, i.e. its powerset $\mathcal{P}(S)$. This means we would still have to evaluate $O(2^n)$ possible combinations. In other words, in both cases, the search space grows exponentially with n. This is problematic, as many real-world datasets for which we would want to discover higher-order correlations contain many variables.

The discovery of higher-order correlations is currently limited to a number of works that each consider a specific correlation measure and that develop their own definitions and constraints on this measure in order to develop a scalable solution for discovering higher-order correlations [2] [5] [3] [6]. To the best of our knowledge, there currently exists no overarching problem definition that unifies the existing work on higher-order correlations. Such a definition is useful as basic optimizations that are common to more than one correlation measure would not have to be reinvented. In our view, such a general problem definition is the first step to enabling the wider adoption of higher-order correlations.

In this thesis, we focus on the problem of discovering higher-order correlations between timeseries. Specifically, we define the general problem of higher-order correlation maximization and develop solutions for the case where the correlation measure is the Pearson correlation. In doing so, we make the following contributions.

- We propose a general problem definition for the problem of maximizing higher-order correlations. The definition supports correlation measures C and aggregation functions f.
- We develop an application CorrelationMiner that runs on top of the distributed computing framework Apache Spark [1] and offers a way of distributing the exhaustive solution to the aforementioned maximization problem.
- We develop various solvers for CorrelationMiner that focus on maximizing higher-order correlations when the correlation measure C is the Pearson correlation coefficient.

1.1 Applications

To motivate our study of higher-order correlations, let us look at some applications. As it turns out, higher-order correlations show up in many real-life datasets.

1.1.1 Stock market

One of the example applications for the pairwise correlations is that of diversification of a stock portfolio by reducing the correlations between stocks. A portfolio that contains stocks with low pairwise correlations is desirable because the prices of the stocks are less likely to move together, thereby achieving proper diversification of the portfolio. In other words, when one stock price drops, the overall portfolio value does not decrease significantly because the prices of the other stocks are less likely to fall along with it. When looking to add new stocks to our portfolio, the pairwise correlations between the stocks to add and the stocks already in the portfolio can provide us with a metric on the impact on the diversification of the portfolio.

However, considering just the pairwise correlations between the stocks in a portfolio may not capture certain relationships that exist between sets of stocks, such as the relationship between a portfolio of stocks and a set of stocks that one is looking to add to the portfolio. Therefore, we may want to consider the stock portfolio as a whole when looking for new investment opportunities. That is, when considering the correlation between stocks, we do not just compute pairwise correlations between the individual stocks and potential stocks to add to our portfolio. We also take into account the correlation between the set of stocks we already have in our portfolio and the set of stocks that we are looking to add to it. In doing this, we may find that the resulting higher-order correlation differs significantly from the individual pairwise correlations, impacting the diversification of the portfolio in a way that the pairwise correlations did not express.

1.1.2 Climate phenomena

Another application area of higher-order correlations between sets is climate data. In [5], the authors propose the notion of a multipole, which is a linear combination of variables that they compute a form of correlation between. They note that these multipoles cannot be found by merely looking at pairwise correlations, and propose a solution for efficiently finding these multipoles. Applying this method of finding multipoles to a dataset consisting of climate data yields multipoles that express novel climate phenomena that had not been discovered before.

1.1.3 Traffic patterns

The work on multipoles [5] and earlier work by the same authors [2] mention how their method may be applied to traffic patterns as measured by highway sensors. Again, they show how the multipole approach may find relationships not captured by other measures.

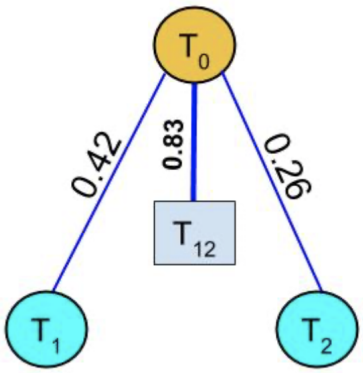


Figure 1.1: A higher-order correlation in the form of a tripole [2].

Chapter 2

Preliminaries

Before defining the problem itself, it is helpful to go over some basic concepts and definitions that will provide some background for the rest of this thesis. We make the distinction between concepts that are directly used in later chapters and ones that are merely applicable to the work, such as the various similarity measures. The former will be treated more extensively in this chapter, while the latter will be mentioned briefly with reference to a more substantial treatment.

2.1 Timeseries

In this thesis, we consider the problem of higher-order correlation maximization in the context of timeseries data. Formally, we define the problem context as

Problem context 2.1. Consider a set of n standardized timeseries $S = \{T_1, .., T_n\}$ observed over d consecutive timestamps, with $T_i \in \mathbb{R}^d$ for all $1 \leq i \leq n$ and $d \in \mathbb{N}^+$.

Here, standardized means that any timeseries $T_i \in S$ has mean $\mu = 0$ and variance $\sigma^2 = 1$.

We assume an offline setting. In other words, we assume that all d values correspond to timestamps in the past. Alternatively, we could consider the problem in an online (streaming) setting, in which new values (dimensions) arrive at every timepoint t. In this setting, d is increasing with every timepoint. In that case, we may consider working with slices of the timeseries using sliding windows. Formally, we denote the values of a timeseries $T_i \in S$ as $T_i = x_1, .., x_d$. Then, a sliding window S_i with $1 \leq i \leq n$ is defined as consisting of a subset of the timeseries values $x_{k-W}, ..x_k \in T_i$ with $W + 1 \leq k \leq d$. At every update t_{k+1}, S_i is updated to consist of the values $x_{k-W+1}, .., x_{k+1}$.

Note that the problem context is non limiting in the sense that the solutions we propose will work for high-dimensional vectors in general. The problem context of timeseries is based on the dataset that we consider in this thesis. This dataset consists of a set of timeseries representing stock price values of companies.

2.2 Similarity measures, distance metrics and correlation measures

In this section, we consider various measures of similarity, distance metrics and correlation measures. While only a subset of these measures will return in later chapters, we note that these measures may all be valid choices for C that we will define as part of problem 3.1. For all these measures, it holds that they can be used to compare different kinds of data in different problem contexts. The choice of the most appropriate measure is dependent on the problem context and type of input data, but as long as it can be generalized to operate on more than two data objects, then the definition of problem 3.1 applies.

2.2.1 Cosine similarity

Cosine similarity measures the cosine of the angle between two nonzero vectors $X, Y \in \mathbb{R}^d$ where d equals the dimensionality of the vectors. For vectors that have the same orientation, the cosine similarity is 1. For orthogonal vectors, the cosine similarity is 0. While for orthogonal vectors, the cosine similarity is -1. The cosine similarity is especially useful when dealing with sparse vectors, as only nonzero dimensions have to be taken into account. Formally, the cosine similarity between two nonzero vectors X and Y can be expressed as follows:

$$X \cdot Y = \|X\|\|Y\| cos\theta \tag{2.1}$$

Note that cosine similarity and the Pearson correlation coefficient are equivalent when X, Y are standardized, i.e. when they have mean zero and unit variance.

2.2.2 Distance metrics

2.2.2.1 Manhattan distance

The Manhattan distance expresses the distance between two points measured along axes at right angles. Given n points in space, we have that the Manhattan distance is defined as $(\sum_{i=1}^{n} |x_i - y_i|^p)^{1/p}$.

2.2.2.2 Euclidean distance

The Euclidean distance is a well known distance measure that expresses the straight line distance between two points p_1, p_2. Given n points or vectors in (Euclidean) space, we have that the Euclidean distance is defined as $\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$.

2.2.3 Correlation

We are interested in computing and tracking the correlation between timeseries. As mentioned earlier, this is useful to a number of disciplines. We want to determine whether two (sets of) variables or timeseries have a mutual relationship. Correlation is a way of expressing the strength of this mutual relationship.

2.2.3.1 Pearson's correlation

Perhaps the most well-known correlation measure is Pearson's correlation coefficient, which measures the strength of a linear relationship between two normally distributed variables X and Y

$$\rho = \frac{\text{cov}(X, Y)}{\sigma_x \sigma_y} \tag{2.2}$$

Where we have that $cov_{x,y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n-1}$. In this way, the covariance describes the joint variability of the variables X and Y.

Using Pearson's correlation coefficient to express the correlation between timeseries $T_i, T_j \in S$ as in our problem context 2.1, where x_i, y_i such that $1 \leq i \leq m$ denote values from T_i, T_j respectively.

$$\rho(T_i, T_j) = \frac{\sum_{i=1}^{m}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{m}(x_i - \overline{x})^2 (y_i - \overline{y})^2}} \tag{2.3}$$

Through a property called bilinearity of covariance, the Pearson correlation measure can be extended to relating a pair of linear combinations of random variables. This property hence allows

one to generalize the Pearson correlation coefficient to express correlation between sets of variables instead of merely individual variables.

The bilinearity of covariance property states that

$$\langle au + bv, w\rangle = a\langle u, w\rangle + b\langle v, w\rangle \tag{2.4}$$

and on the RHS:

$$\langle u, av + bw\rangle = a\langle u, v\rangle + b\langle u, w\rangle \tag{2.5}$$

Now, let $X_1, ..., X_n$ be a set of n (different) random variables such that $L = a_1X_1 + a_2X_2 + ... + a_kX_k$ and $R = b_1X_1 + b_2X_2 + ... + b_lX_l$ with $k, l \subseteq \{1, n\}$ form linear combinations of the variables. Here, a and b are scalars. By bilinearity of covariance, we obtain:

$$Cov(L, R) = \sum_{i=1}^{k}\sum_{j=1}^{l} a_ib_jCov(X_i, X_j) \tag{2.6}$$

We use the definitions of $X_1, ..., X_n$, L and R here again. First, we note that

$$Corr(L, R) = \frac{Cov(L, R)}{\sqrt{Var(L) \cdot Var(R)}} = \frac{\sum_{i=1}^{k}\sum_{j=1}^{l} a_ib_jCov(X_i, X_j)}{\sqrt{Var(L) \cdot Var(R)}} \tag{2.7}$$

where Var denotes the variance between the variables. By definition of variance, we have that

$$Var(aX + bY) = a^2Var(X) + b^2 \cdot Var(Y) + 2abCov(X, Y) \tag{2.8}$$

and for linear combinations (like L and R) it follows that

$$Var(\sum_{i=1}^{N} a_iX_i) = \sum_{i=1}^{N} a_i^2Var(X_i) + 2 \cdot \sum_{1\leq i\leq j\leq N} a_ia_jCov(X_i, X_j) \tag{2.9}$$

Again, a is a scalar here.

Thus, $Corr(L, R)$ can be reduced to our earlier definition while expressing $Var(L), Var(R)$ using the definition of variance for linear combinations that we just defined. The resulting equation is left out to save space.

2.2.3.2 Spearman's correlation

Other measures of correlation have also been proposed, such as Spearman's rank correlation coefficient. This measure determines how well the relationship between two variables can be described using a monotonic function, as opposed to a linear function with Pearson's. It is applicable in situations where the datapoints cannot be assumed to be normally distributed.

2.2.3.3 Distance correlation

Acknowledging the importance of establishing independence between two variables, [7] proposed the distance correlation measure, which measures dependence between two random vectors that do not necessarily have the same number of dimensions. A limitation of Pearson's correlation measure is that two variables X, Y having $\rho(X, Y) = 0$ does not imply that the variables are independent. On the contrary, the distance correlation between X, Y being zero does mean that the variables are independent. As opposed to Pearson and Spearman's correlation measures, the distance correlation is also able to express nonlinear association between the variables. For an in depth treatment of the distance correlation measure, we refer the reader to [7].

2.2.4 Information theory measures

2.2.4.1 Mutual information

A measure of dependence that originated from the field of information theory is that of mutual information [8]. This measure expresses the mutual dependence between two random variables X, Y by measuring how much having information about X reduces uncertainty about Y and vice-versa. Independence of X and Y in this case implies that one cannot obtain information about either variable by knowledge of the other. Specifically, it measures the dependence expressed in the joint distribution of X and Y relative to the joint distribution of X and Y when one assumes their independence.

2.2.4.2 Multi-information

The mutual information is able to express the pairwise dependence of variables X and Y. However, it being a pairwise measure of dependence, it cannot express the dependence beyond two random variables. To extend beyond pairwise relationships, various multivariate mutual information measures have been proposed. One of these measures, the multi information or total correlation, generalizes the mutual information to a set of variables. For a detailed treatment of the multi-information measure, we refer the reader to [9] in particular.

2.3 Linear regression

Linear regression can be used to find the linear relationship between a target (dependent) variable and one or more predictor (independent) variables. Given problem context 2.1 and a single dependent variable, a linear regression can find the linear combination that maximizes the Pearson correlation with the dependent variable. The most basic form of linear regression is linear regression with a single variable, often referred to as simple linear regression. Given a *dependent* variable Y, it aims to find a linear function that, as best as possible, predicts the values of Y as a function of the values of the *independent* variable X. The extension of the independent variable to a set of independent variables turns the problem into a multiple linear regression problem. Finally, regressing on a set of dependent variables instead of a single dependent variable can be achieved using multivariate linear regression.

For the simple variant, the procedure aims to fit a line $y = \alpha X + b + \epsilon$. Here, a denotes the slope of the line, b denotes the y-intercept and ϵ is the error term, also called the residuals of the model. The most common formulation of fitting a linear regression is as a least squares problem. In this formulation, the goal is to minimize the squared loss of the residuals. Given k datapoints, this means that for every datapoint (x_i, y_i) with $1 \leq i \leq k$ each residual can be defined as $r_i = y_i - f(x_i, \alpha)$. In other words, r_i describes the deviation between the point y_i on y and the actual value x_i of the independent variable X. The sum of squared residuals is then defined as $S = \sum_{i=1}^{k} r_i^2$, which is the quantity to be minimized. For multiple linear regression, with n independent variables, the line to fit becomes $Y = \alpha X_1 + .. + \alpha X_n + b + \epsilon$. In the case of multivariate (multiple) linear regression, Y becomes a matrix of dependent variables.

Observe that there exists an inverse relationship between the pairwise correlation of X and Y $\rho_{X,Y}$ and the sum of squared residuals S. Recall that $\rho_{X,Y}$ will express the strength of the linear relationship between X and Y. Hence, any increase in $\rho_{X,Y}$ will result in a decrease in S due to the deviations between the datapoints and (X, Y) and the line $aX + b$ being smaller.

2.3.1 Regularization

In multiple linear regression, the goal is to obtain a model that best explains the linear relationship between the dependent variable(s) Y and possible independent variables $X_1, .., X_n$. However, when working with a large dataset consisting of many variables, a regression model containing many of the n independent variables will be less useful to the researcher due to the model becoming

harder to interpret. To increase interpretability of the model, regularization approaches have been proposed. With regularized regression, the number of variables in the solution is penalized, thereby increasing interpretability of the model. Regularization is often done by penalizing the regression coefficients in some form. One popular regularization strategy is the LASSO, in which the L_1-norm of the regression coefficients is penalized. By adding a penalization term for either of these norms in the optimization objective of the least squares procedure, sparsity is induced, resulting in a model that contains less variables. Other strategies include ridge regression[10] and the elastic net[11] where the L_2 norm or a combination of the L_1 and L_2 norm are penalized respectively.

2.3.2 Variable selection

When conducting regression analysis on a dataset, one often wants to know which subset of features or variables best explain the dependent variable. In other words, which variables does one need to select to get the best fit. This process, known as variable or feature selection in statistics and machine learning, aims to extract the relevant variables from the dataset that best predict the dependent variable. By selecting only the relevant variables, the interpretability of the model increases and the fitting of the regression is faster. This is especially helpful in datasets containing many variables, since there having every variable contribute to the regression model will make it unexplainable and thus no longer useful to the researcher. To select the best subset, the most obvious optimal solution is to try all different subsets of variables for the model and select the best one. However, the number of subsets to consider grows exponentially with the number of variables in the dataset. It is thus computationally infeasible for datasets with many variables, although recent approximate models have made considerable progress in terms of the number of variables they support [12].

Other methods for selecting the relevant subsets have been proposed. They can be broadly categorized into embedded methods, wrapper methods and filter methods. Embedded methods select variables during model construction, they include regularization techniques such as the LASSO. Wrapper methods search the space of possible subsets using a search algorithm and then train a predictive model on each subset. They then use the error rate of the subset on the model, obtaining a score to judge its relevance. Filter methods are similar to the wrapper methods in that they come up with a score for each subset, but instead apply a computationally less expensive measure to score the subset. The trade off is that this makes the resulting subset less tuned to the model since the score is not directly derived from its error rate on the model. Examples of search strategies employed by filter and wrapper methods include greedy approaches such as forward selection and backward elimination and least angle regression. Measures that are suitable for filter methods include correlation measures such as Pearson correlation and the information theoretic measure of mutual information.

2.4 Canonical correlation analysis

We discussed the use of (multiple) linear regression to maximize the Pearson correlation between a single dependent variable and a linear combination of one or more independent variables. When considering more than one dependent variable, i.e. a set of variables, the technique of canonical correlation analysis (CCA) can be used. Proposed by Hotelling in 1936 [13], CCA makes it possible to maximize the Pearson correlation between sets of input variables X, Y. Given two column vectors of the variables X, Y, CCA finds linear combinations of the variables in X and Y such that they are maximally correlated with each other. It does so starting from the cross-covariance matrix $\Sigma_{X,Y} = cov(X, Y)$ with $n \cdot m$ entries with each entry (i, j) being the $cov(x_i, y_j)$. CCA finds up to $min(n, m)$ pairs of linear combinations, the so called canonical variates. These pairs of canonical variables are vectors $a \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ such that $a^T X$ and $b^T Y$ maximize $\rho = corr(a^T X, b^T X)$. The pairs are reported in order of their correlation coefficient, with the pair having the highest correlation coefficient reported first. Moreover, for every one of the subsequent

pairs it holds that it is uncorrelated with the previous pair.

Thus, the parameter to optimize for in CCA is $\rho = \frac{a^T \Sigma_{XY} b}{\sqrt{a^T \Sigma_{XX} a}\sqrt{b^T \Sigma_{YY} b}}$. where Σ_{XX}, Σ_{YY} denote the covariance matrices of input vectors X and Y respectively.

For an in depth treatment on CCA, we refer the reader to [14].

2.4.1 Sparse CCA

Like in regression analysis, when the number of variables n in the dataset is large, the linear combinations found using regular canonical correlation analysis lack interpretability. In an effort to increase the interpretability and usefulness of the correlations found by CCA, sparse CCA approaches have been proposed. With sparse CCA, the goal is to constrain the number of variables in the linear combinations that CCA finds. Here too, the regularization approaches used in linear regression models are of use. Notably the LASSO, which uses the $L1$-norm, is used in quite a few works to obtain sparse linear combinations with CCA. Various sparse CCA methods have been proposed as part of [15] [11] [16].

2.5 Linear programming

Linear programming is an optimization technique for a system of linear constraints and a linear objective function. The linear objective function is minimized or maximized subject to various linear constraints.

$$\begin{aligned} \text{maximize} \quad & c_1 x_1 + \cdots + c_n x_n \\ & a_{11} x_1 + \cdots + a_{1n} x_n \leq b_1 \\ \text{subject to} \quad & \vdots \\ & a_{m1} x_1 + \cdots + a_{mn} x_n \leq b_m \end{aligned} \tag{2.10}$$

There are two efficient methods for solving linear programming problems, the simplex method and the interior point method. Various mathematical optimization toolkits exist that offer efficient linear programming solvers [17] [18].

2.6 Practical preliminaries

2.6.1 Apache Spark

We propose an application CorrelationMiner, which we introduce in chapter 5, that runs on top of Apache Spark, a distributed computing engine. For more information about Apache Spark, we refer the reader to [1].

2.6.2 Gurobi

As part of the application CorrelationMiner, we develop two linear programs and implement them using the Gurobi optimization toolkit. For more information about Gurobi, we refer the reader to [17].

Chapter 3

Problem definition

We define the higher-order correlation maximization problem.

Definition 3.1. higher-order correlation maximization Consider a set of n standardized timeseries $S = \{T_1, .., T_n\}$ observed over d consecutive timestamps, with $T_i \in \mathbb{R}^d$ for all $1 \leq i \leq n$ and $d \in \mathbb{N}^+$. Let C denote a correlation measure and let f denote a suitable aggregation function. We want to find sets $X, Y \subset S$ with $X \cap Y = \emptyset$ such that $\max \arg_{X,Y} C(f(X, Y))$. We overload the notations of C and f such that they support different measures of correlation and corresponding aggregation functions.

As mentioned earlier, examples of correlation measures C are the Pearson correlation coefficient and the multi-information. For the former, a suitable aggregation function f would be one that produces linear combinations X', Y' of sets X, Y. As a simple example, we may define $f(X, Y) = (\sum_{T_i \in X} \frac{T_i}{|X|}, \sum_{T_i \in Y} \frac{T_i}{|Y|})$ that computes the component-wise average of all timeseries in X (respectively Y) such that we end up with a pair of timeseries corresponding to the timeseries in X (respectively Y). When C is the multi-information measure, an example of an aggregation function can be simply $f(X, Y) = X \cup Y$, as the multi-information measure expects just a single set of variables for its input.

Definition 3.1 describes the basic, unconstrained version of the problem. We can make the problem more interesting by allowing different constraints to be applied. One fairly simple constraint is to bound the cardinality of the sets X, Y. For example, we may define an upper bound on $|X|$ and $|Y|$. In many cases, such constraints are desireable, as including too many vectors may hinder interpretability. Constraining the problem in this way may significantly improve scalability, as it can restrict the search space. Bounding the cardinality of the sets X, Y for example allows solutions to skip evaluation of subsets of which their cardinality is outside these bounds.

3.1 Maximizing Pearson correlation

The above definition defines the problem of correlation maximization such that it admits any correlation measure C and accompanying aggregation function f. For the remainder of this thesis, we will look at correlation maximization given a specific correlation measure: the Pearson correlation coefficient. The reason for focusing on this measure is that it is well understood, as opposed to some of the multivariate information theoretic measures like multi-information. In addition, the technical contribution, in the form of an application on top of the Apache Spark platform, already has good support for computing the Pearson correlation coefficient over large vectors. Given the focus on Pearson correlation, we will henceforth use 'correlation' as a shorthand for the Pearson correlation coefficient in the subsequent solution and results chapters.

Thus, shifting our focus to the problem of maximizing Pearson correlation, we define the following

Definition 3.2. Pearson correlation maximization Consider a set of n standardized timeseries $S = \{T_1, .., T_n\}$ observed over d consecutive timestamps, with $T_i \in \mathbb{R}^d$ for all $1 \leq i \leq n$ and $d \in \mathbb{N}^+$. We want to find sets $X, Y \subset S$ with $X \cap Y = \emptyset$ such that $\max \arg_{X,Y} C(f(X, Y))$. Here, f is defined as the aggregation function $f(X, Y) = (X', Y')$ producing linear combinations $X' = a_1 T_1 + .. + a_l T_l, Y' = a_1 T_1 + .. + a_m T_m$.

Now, observe that the correlation between the linear combinations is determined by two factors: the vectors participating in the combinations and their coefficients (or weights). Thus, when looking to maximize $\rho(X', Y')$, we should pick these two in a smart way. One way of finding linear relationships between variables is linear regression. For simple linear regression between variables X^*, Y^*, we know that the regression coefficient as determined by the regression procedure will be equivalent to the Pearson correlation $\rho(X^*, Y^*)$ if X^* and Y^* are standardized.

For multiple linear regression, one starts from a dependent variable and a set of independent variables. Then, the regression procedure finds multiple coefficients for the independent variables that maximize the linear relationship with the dependent variable. Hence, regression may be a valuable tool for determination of the coefficients as finding the weighted linear combination that best predicts the dependent variable aligns with our goal of maximizing the correlation.

It should be noted that our starting point differs from a traditional regression analysis in that we do not know a priori what the timeseries participating in the regression should be. The problem of determining the set of independent variables that best fit the dependent variable is not uncommon to regression analysis. To determine this set, variable selection techniques such as forward selection or L_1 regularization may be used to determine the set of independent variables that best fit the dependent variable.

However, not only do we not know the set of independent variables $Y \subset S$, we also do not know the dependent variable. In fact, we may not deal with a single dependent variable but with a set of dependent variables $X \subset S$. In that case, we should construct the dependent variable to be some linear combination of the set of timeseries that we are looking to maximize correlation with. Thus, using regression techniques in an attempt to solve problem 3.2 will require us to first construct our dependent variable X' as a linear combination of the timeseries in X, but we do not know X.

There exists a variation of problem 3.2 that bears significant interest itself. Therefore, we will define it and treat it separately.

Definition 3.3. Problem variation Consider a set of n standardized timeseries $S = \{T_1, .., T_n\}$ observed over d consecutive timestamps, with $T_i \in \mathbb{R}^d$ for all $1 \leq i \leq n$ and $d \in \mathbb{N}^+$. We are given a set $X \subset S$ and we want to find a set $Y \subset S$ with $X \cap Y = \emptyset$ such that $\max \arg_{X,Y} C(f(X, Y))$. As in problem 3.2, f is defined as the aggregation function $(X, Y) = (X', Y')$ producing linear combinations $X' = a_1 T_1 + .. + a_l T_l, Y' = a_1 T_1 + .. + a_m T_m$.

Before considering possible solutions to problem 3.3, we first state a possible practical application of this problem in the domain of our running example dataset.

Example 3.1. Portfolio diversification Imagine owning portfolio of stocks consisting of l stocks $P = T_1, .., T_l$ in certain proportions $a_1, .., a_l$ such that we can form the linear combination $P' = a_1 T_1 + .. + a_l T_l$. Then, when looking to expand P with $m > 1$ stocks $P^* = \{T_1, .., T_m\} \subset S \setminus P$, we may want to consider the correlations of the existing l stocks with stocks that we may want to add to P.

The reason for considering these correlations is that we want these to be low so as to keep P properly diversified. Considering the fact that the individual pairwise correlations of the m stocks in P^* with the l stocks in P may not tell the full story, we can compute the higher-order correlation between P and P^*. In case we find that there exists a significant higher-order correlation $\rho(P, P^*)$, we may want to refrain from adding the m stocks in P^* to our existing portfolio P.

Example 3.1 maps to problem 3.3 with $X = P, Y = S \setminus P$. Let us now look at a possible solution to problem 3.3.

First, consider the case where we are trying to maximize correlation for a single $T_i \in S$ forming our dependent variable, in other words, when $X = T_i$. Then, any choice of aggregation function f that produces $X' = T_i$ will solve the problem given that Y' is picked correctly. For the set of independent variables, we define $Y = S \setminus T_i$. Determining Y' can then be done using the LASSO for example, where we set λ such that the resulting Y' found by the LASSO procedure has the desired cardinality.

Now, for the case where $|X| \geq 2$, X' becomes more complicated to determine as we have to pick its coefficients such that $\rho(X', Y')$ is maximized. Simply making X' consist of equiweighted timeseries by setting $a_i = \frac{1}{l}$ will not necessarily achieve this. Neither will the ordinary least squares regression and its regularized variants (such as the LASSO) construct X', they can only help determine the best Y'. Instead of these techniques, we can use a technique called the canonical correlation analysis.

3.1.1 Using canonical correlation analysis

With canonical correlation analysis, the input usually consists of two datasets X, Y. For example, X and Y could represent different neuroimaging datasets between which one wants to determine the similarity [19]. For our purposes, we can define $X = \{T_1, .., T_l\}, Y = S \setminus X$. Then, the CCA procedure will find $k = \min(|X|, |Y|)$ mutually orthogonal linear combinations X'_i, Y'_i with $1 \leq i \leq k$ for which the Pearson correlation coefficient is maximized. Of these linear combinations, the so-called canonical variates, the first pair X'_1, Y'_1 will have the highest Pearson correlation coefficient. Given that the goal of problems 3.2, 3.3 is to maximize the Pearson correlation coefficient, we will always use the first canonical vector when using CCA in this thesis.

While CCA helps us solve problem 3.3, it should be noted that the regular CCA procedure will include all vectors $T_i \in Y$ in the linear combination. Even though problem 3.3 is unconstrained, including every vector of Y in the solution does not make its result very practical. Reducing the cardinality of Y', i.e. making it a sparse linear combination, would benefit interpretability. For this purpose, we may use a regularized variant of CCA called sparse CCA. In addition, we can use a different method for inducing sparsity in the regular CCA solution that we will explore as part of chapter 5. In that chapter, we will also explore solutions to the more general problem 3.2 and integrate them into an application CorrelationMiner.

Chapter 4

Related work

4.1 Pairwise correlations

Over the years, a number of works have looked into the problem of correlation analysis. The problem of finding pairwise correlations is well studied. In the basic problem, we are interested in finding the pairs of timeseries with high correlations. Solutions generally aim at two settings: offline queries and the online setting, where time series arrive as a stream of values. Moreover, variations on the basic problem exist, such as considering only correlations above a threshold τ, the top-k correlations or lag correlations. Due to the high dimensionality of the time series, proposed solutions often resort to dimensionality reduction techniques, indexing schemes and approximation of the correlation coefficient.

An early work on finding correlations between time series in an offline setting is HierarchyScan [20]. HierarchyScan locates one-dimensional subsequences within a collection of sequences of arbitrary length. It finds correlations between pairs of sequences, pruning pairs that have their correlation coefficient below a predefined threshold. The correlation coefficient itself is approximated, without the guarantee of preventing false negatives. Another well known work on pairwise correlations is StatStream [21], which proposes a solution for finding correlations between time series in an online setting. It considers tracking pairwise correlations among all pairs of streams, using a sliding window over the time series as its temporal span. Dimensionality reduction is applied by adapting the Discrete Fourier Transform (DFT) to an incremental version, keeping the first few DFT coefficients to represent the stream. Using these coefficients, correlation between streams can be approximated with small error. A grid structure is proposed in which the grid cells store hash values of a subset of DFT coefficients representing a stream. A property of this index scheme is that for any given stream, a limited number of cells adjacent to its cell in the grid need to be examined to determine its correlated pairs. This greatly reduces the numbers of pairs to examine.

One interesting type of correlation is a lag correlation, which is a correlation between unaligned time series. The StatStream work [21] does not handle lag correlations. An example of a work that considers lag correlations between streaming timeseries is [22]. The work proposes the BRAID system to deal with lag correlations, noting the motivating example of an increase in house sales some months after a decrease in interest rates.

4.2 higher-order correlations

Even though there are many applications of pairwise correlation analysis and numerous proposed solutions to the problem and its variations, pairwise correlations have their limitations. As described in the problem definition, often a correlation between a combination of variables may exist that is not captured by the correlations between pairs of the combinations variables. Existing literature on finding these higher-order correlations is a lot sparser compared to the literature on

pairwise correlations. This chapter aims to present a survey of works that have considered this problem in some form. We will treat a number of works that we deem most relevant to ours in more detail and attempt to characterize them in terms of the problem definition of problem 3.1.

4.2.1 Tripoles

Starting with [2], an attempt to generalize correlations beyond pairs of variables is made by introducing the notion of tripoles. A tripole captures a relationship pattern between three interacting time series, that is not captured by looking at just pairwise relationships. The authors provide an example that we use in chapter 3 in which three time series T_0, T_1 and T_2 are to form the tripole. It is stated that the sum of the two time series T_1 and T_2 may provide much higher correlation with T_0 than do the individual pairwise correlations (T_0, T_1) and (T_0, T_2).

Looking at this tripole from the perspective of a graph, the series T_0 forms the root node and has edges with the nodes representing the series T_1 and T_2, which are leaf nodes. The edges are weighted, with their weights representing the pairwise correlation between the nodes they connect. "Interesting" tripoles are tripoles for which the correlation between T_0 and the sum of their timeseries, T_{1+2} is much stronger than the individual pairwise correlations T_0 has with either T_1 or T_2. T_0 bears similarity to the dependent variable in regression models.

In terms of our proposed problem definition, we note that the concept of a tripole is an instantiation of it. Specifically, it can be seen as a constrained instantiation with X consisting of a single timeseries T_0, and Y consisting of 2 timeseries T_1, T_2. The constraint here is an upper bound on the cardinality of sets X, Y, with $|X| = 1, |Y| = 2$. The aggregation function f can simply be defined as $f(X, Y) = (\sum_{T \in X} T, \sum_{T \in Y} T)$ as every timeseries contributes equally to the correlation. The correlation measure C is the Pearson correlation coefficient. The authors reiterate the computational complexity of the brute-force solution, which is $\binom{N}{3}$ in this case due to only considering triples as a consequence of the cardinality constraints on sets X, Y. They note the importance of formalizing the concept of a tripole as a first step to studying them and take special care to devise measure that can identify "interesting" tripoles.

In many ways, the tripoles paper is very relevant to problem 3.2 that is the focus of this thesis. It uses the same measure of correlation and strengthens our thesis that the concept of higher-order correlations is useful by identifying new phenomena in real datasets. However, the work differs from this one in that a tripole is a relatively narrow definition compared to problem 3.2. It defines a very simple aggregation function f that does not support weighing differently the contributions of the timeseries in Y. Moreover, by restricting the cardinality of X and Y many possible higher-order correlations involving more than a triple of timeseries can not be considered.

It is not immediately clear whether the definitions they introduce for tripoles will generalize to be applicable to problem 3.2. However, their relatively severe restrictions on the problem definition, such as the cardinality constraint on sets $X, Y \subset S$, does allow their solution to be significantly more scalable than the brute force one. The measures they introduce, such as the notion of jump to distinguish interesting tripoles from non-interesting ones partially motivates our use of the maximum pairwise correlation when using CCA to tackle the problem variation defined as problem 3.3.

4.2.2 Multipoles

A follow up work by the same authors [5] introduces the notion of multipoles. A multipole is a class of linear relationships between more than two time series (variables) subject to the criteria that the variables together show strong linear dependence and that each individual variable has a strong contribution to this linear dependence. Observe that these criteria express that a multipole captures a relationship that is not observed from just the pairwise linear dependencies of the time series, as excluding any variable from the multipole would significantly weaken it. Hence, since a pairwise relationship between any two time series in the multipole necessarily excludes the other time series of the multipole, the linear dependence would not be strong.

To formalize the multipole, the authors introduce a number of supporting definitions. They start from a set S consisting of k standardized time series observed over T consecutive timestamps. They then define a matrix X of size $T \cdot k$ from which they compute the covariance matrix Σ of size $k \times k$. Continuing, a normalized linear combination is defined as a linear combination of which the L2-norm is 1 and hence a normalized linear combination of S is defined as $Z_s = Xl$ where l is a vector of time series values. They then define Z_S^* to be the least variant normalized linear combination (LVNLC), that is, the normalized linear combination with least variance across the T observations.

As explained by the authors, the variance of the LVNLC of the set S can be used as an inverse indicator of the strength of its linear dependence. This allows them to express the linear dependence of a set of vectors S as $\sigma_S = 1 - var(Z_S^*)$. Defining the linear dependence in this way gives it the convenient properties that $\sigma_S \in [0, 1]$ and that it holds that any superset of S always has equal or higher linear dependence than S. Another important observation stated by the authors is that the linear dependence σ_S is directly related to the least eigenvalue λ_{min} of the covariance matrix Σ: $\sigma_S = 1 - \lambda_{min}$. Linear dependence is able to express the strength of the relationship among the variables in S. However, it is not yet sufficient to determine if S is a multipole as it does not yet satisfy the criterium that each variable in S should have a strong contribution to the linear dependence. For this, the authors introduce a notion of linear gain that expresses the gain in σ_S with respect to one of its (proper) subsets S'. The linear gain is defined as $\Delta_{\sigma_S} = \sigma_S - max_{S' \subset S}(\sigma_{S'})$ and setting a sufficiently high threshold on this measure thus allows for excluding irrelevant variables from any potential multipole set S. Moreover, having a high threshold on the linear gain also avoids the problem of multicollinearity among the variables of S. With these supporting definitions, the authors propose the following definition of a multipole:

A multipole refers to the set S of variables with $|S| \geq 2$ such that $\sigma_S \geq \sigma$ and $\Delta_{\sigma_S} \geq \delta$, where σ and δ are user-specified thresholds. In terms of problem 3.1, the concept of a multipole can be seen as a correlation measure C in itself. A suitable aggregation function f would be $f(X, Y) = X \cup Y$.

The authors explicitly contrast the multipole against their earlier work on tripoles. In a multipole, there is no special role for any timeseries, like there is with T_0 in tripoles. Therefore, a tripole does not generalize to a multipole. This special role that the tripoles concept [2] has can be related to the dependent and independent sides in regression analysis, which the authors also note. We use a similar observation to motivate the consideration of regression approaches to the development of solutions for problem 3.2. There are a number of theorems defined over multipoles that facilitate the efficient discovery of them. For example, the work defines a multipole S to be maximal when none of its supersets are in the set Q of multipoles. The problem of multipole discovery can then be formulated as finding the set of P of all maximal multipoles in a given set of time series, taking into account the user-defined thresholds δ and σ.

Noting the combinatorial explosion of the search space for discovering multipoles in a brute-force manner, the authors explore ways to prune the search space in order to more efficiently discover the multipoles set P. They relate the linear gain Δ_{σ_S} to the strength of the pairwise correlations between the members of S in order to identify a set of candidate subsets more likely to qualify as multipole relationships. The discovery of maximal multipoles is then done by a clique-enumeration approach on a correlation graph induced on the time series dataset, where the graph is constructed such that promising candidates form cliques in the graph. The authors evaluate their solution on two separate datasets: one containing time series of sea level pressure (SLP) data and one containing fMRI data.

4.2.3 NIFS

In [3], the authors propose a method for finding correlations in binary data. While they do not share our problem context 2.1, the work is still very relevant. They, like ours and other works, note the limitations in considering just pairwise correlations. Specifically, they note the importance of finding correlations between more than two features (variables) in biology, their application domain. The work considers the problem of finding higher-order correlations in feature subspaces,

i.e. in subsets of the total set of variables. As opposed to looking at the smallest eigenvalues of the covariance matrix, as done in [5], they use the multi-information as their measure of correlation C. The aggregation function f can the same as for the measure of linear dependence used in [5], namely $f(X, Y) = X \cup (Y)$ as both measures operate on a single subset of variables.

In particular, the feature subsets of interest are introduced as Non-Redundant Interacting Feature Subsets (NIFS). A NIFS is a subset of the features $S = \{X_1, X_2, ..., X_n\}$ if and only if it satisfies the following two criteria: (1) S is a strongly correlated feature subset, that is, the multi information $C(S)$ is such that $C(S) \geq \beta$ while (2) all proper subsets $S' \subset S$ are weakly correlated, that is, $C(S') \leq \alpha$. Here, β and α are user-defined thresholds. The goal is to find all NIFSs in a binary dataset with the given thresholds β and α such that $0 < \alpha < \beta$. Evaluating if any feature subset X is an NIFS is expensive due to requiring the evaluation of the $2^{|X|}$ subsets of X to check if X satisfies the second criterium. However, they have a useful property that can be used to prune the search space. Namely, there is the property that if X is an NIFS, then any of its supersets cannot also be NIFSs due to the subsets of the NIFS needing to be weakly correlated feature subsets. If one views the search space as a tree, this implies that once an NIFS is found, its subtree in the search space can be pruned because none of its members can be an NIFS.

The remaining candidate feature subsets, i.e. the ones that have not been pruned using the previous property, still need to be evaluated in order to determine whether they qualify as an NIFS. This requires computing the multi-information of these candidate subsets, which can be expensive. Using existing inequalities in information theory, the authors derive bounds on the pairwise correlation between the pairs of variables of a candidate feature subset. Here, the pairwise correlation is based on the mutual information measure, the information theoretic equivalent of a pairwise similarity measure. These bounds provide a way to estimate the multi information of a candidate subset. In addition, since the child nodes X' of a candidate subset X in the search tree differ by just a single variable, it is useful to estimate the multi information of X' using the multi information of X and the addition or removal of a single variable X'_i. To this end, the authors derive bounds on this change in multi information that express the change in terms of the Hamming distance between X'_i and the variables in X.

Lastly, the authors show that it is possible to use mutual information to prune pairs from becoming candidate feature subsets during the enumeration of these candidate feature subsets. Observe that the definitions allowing for the pruning of the search space of this work are very similar to the ones proposed in [5], albeit that they are proposed for different correlation measures. The works consider a different problem context and do not acknowledge each other. In our view, the development of similar pruning constraints like these reiterates the importance of the general problem definition introduced in problem 3.1.

4.2.4 Other work

While the above papers have direct relevance to the problem of discovering higher-order correlations in that they can be cast as an instance of problem 3.1,

- In [6], higher-order correlations are studied in the context of categorical data. Like [3], the correlation measure C is the multi-information. One interesting contribution of this work is that they derive a reliable estimator for the total correlation (multi information) measure. Emphasizing the hardness of the problem and the combinatorial explosion of the search space, the authors propose algorithms aimed at coping with this inherent hardness of the problem. In particular, they propose both a branch-and-bound based solution as well as a greedy algorithm. In both solutions, they employ their estimator to efficiently estimate the total correlation of a given subset. For the branch-and-bound solution, they develop bounding functions that can be used to prune the search space.

- There is the field of structure learning, that aims to learn a graphical model of the stochastic dependencies in a given variable set. This graphical model, or Markov network, represents a graph where the nodes represent variables. Any pair of nodes connected by an edge are

conditionally dependent. We refer to [23] and [24] for a more thorough treatment on structure learning.

- The work of [25] defines a problem that is somewhat similar to our problem definition 3.3 in that it defines the problem of maximizing correlation for a generic correlation measure by determining transformations of a set of random variables X_i such that the correlation measure $CORR$ is maximized.
- There exist various works that consider higher-order correlations in the context of the firing of neurons in the brain. The higher-order effects are observed with the synchronous firing of a group of neurons and the works argue that these effects cannot adequately be captured by pairwise correlations [4] [26] [27].

Chapter 5

CorrelationMiner

We propose an application CorrelationMiner that provides an extensible API facilitating the mining of higher-order correlations. It supports a variety of correlation measures in accordance with problem definition 3.1. In this chapter, we first motivate its development and identify its requirements. We then present an overview of its architecture and detail a number of implementations that we have developed in our quest to develop effective solutions to problem 3.2.

5.1 Requirements

Earlier in this thesis, we mentioned the lack of a general problem definition that captures the existing work [2] [5] [3] [6] on higher-order correlation discovery. We noted how this may prevent the wider adoption of higher-order correlations and lead to the reinvention of problem solutions that are not specific to any correlation measure in particular. We then introduced the problem of higher order correlation maximization in problem 3.1. We view this definition as a good first step to handle the aforementioned issues.

CorrelationMiner goes a step further, by implementing a scalable infrastructure and a set of basic solutions that address various instances of problem 3.1, as well as an extensible API for adding more definitions of aggregation and correlation measures and more solutions in a common environment. Having a common infrastructure for all these solutions that handles the basic engineering challenges out of the box (e.g., data management, scaling out) drastically reduces the effort of the researcher to develop new solutions for adaptations of problem 3.1.

As an example, consider the exhaustive solution. Having one good implementation of the exhaustive solution may prevent the need for reinventing it in future works that deal with different correlation measures. In that case, this implementation should allow researchers to plug in their own correlation measure and aggregation function. However, we should also recognize the fact that some optimizations, while very effective in pruning the search space, do not generalize well beyond their specific correlation measure. As an example, consider the bounding of the multi-information value by use of the mutual information as proposed in [3]. Hence the application should also allow researchers to implement solutions tailored to a specific measure C. Finally, while pruning constraints like the bounding of a correlation measure may prove to be most impactful in reducing the search space, we can consider other optimizations that improve scalability. Taking again the exhaustive solution as an example, we may for instance distribute the work of evaluating the subsets of timeseries $X, Y \subset S$ over multiple computers so that the evaluation can be parallelized.

Summarizing, for any application to be effective in offering a general solution to problem 3.1, it should satisfy the following requirements:

- For problem solutions that are independent of the correlation measure, any implementation in the application should offer an extensible API that allows researchers to plug in their own correlation measure C and aggregation function f

- For problem solutions that contain optimizations dependent on the correlation measure, the application should allow researchers to provide their own "solver" component which implements such a measure-specific problem solution
- The application should offer support for the problem solutions to make use of general tactics that improve scalability

5.2 Architecture

Based on the above requirements, we have developed CorrelationMiner in Scala. Its architecture, depicted in figure 5.1, consists of two major components: measures and solvers. The first component, measures, allows researchers to plug in their own correlation measures and aggregation functions according to an interface that matches that of the general problem definition proposed as part of problem 3.1. The second component, solvers, allow researchers to specify their own problem solution implementations in case they (a) contain optimizations specific to their choice of correlation measures and (b) are not already available.

To allow the solvers to make use of general scalability tactics, in particular that of parallelization, we make the distinction between so-called distributed solvers and local solvers. The former type of solver runs on top of a cluster of k worker nodes to parallelize the problem solution implemented by the solver. The latter runs in a centralized fashion, i.e. on a single node, and may optionally be called by one of the distributed solvers. Distributing the work of the distributed solvers is the responsibility of Apache Spark [1], a distributed computing framework on top of which we implemented the CorrelationMiner application.

5.2.1 Higher-order correlation interface

We implemented the general problem definition of problem 3.1 as an interface available in the CorrelationMiner application.

```
trait HOCorrelation {
    def C(ts: AbstractAggregateTimeSeries): Double
    def f(ts: AbstractMultipleTimeSeries): AbstractAggregateTimeSeries
}
```

Listing 5.1: The higher-order correlation interface

```
abstract class AbstractAggregateTimeSeries

case class AggregateTimeSeries(x: TimeSeries) extends
    AbstractAggregateTimeSeries
case class AggregateTimeSeriesPair(x: TimeSeries, y: TimeSeries) extends
    AbstractAggregateTimeSeries
```

Listing 5.2: The AbstractAggregateTimeSeries interface

```
import breeze.linalg.DenseVector

abstract class AbstractMultipleTimeSeries

case class TimeSeries(id: String, values: DenseVector[Double])
case class MultipleTimeSeries(x: Array[TimeSeries]) extends
    AbstractMultipleTimeSeries
case class MultipleTimeSeriesPair(x: Array[TimeSeries], y: Array[TimeSeries])
    extends AbstractMultipleTimeSeries
```

Listing 5.3: The AbstractMultipleTimeSeries interface

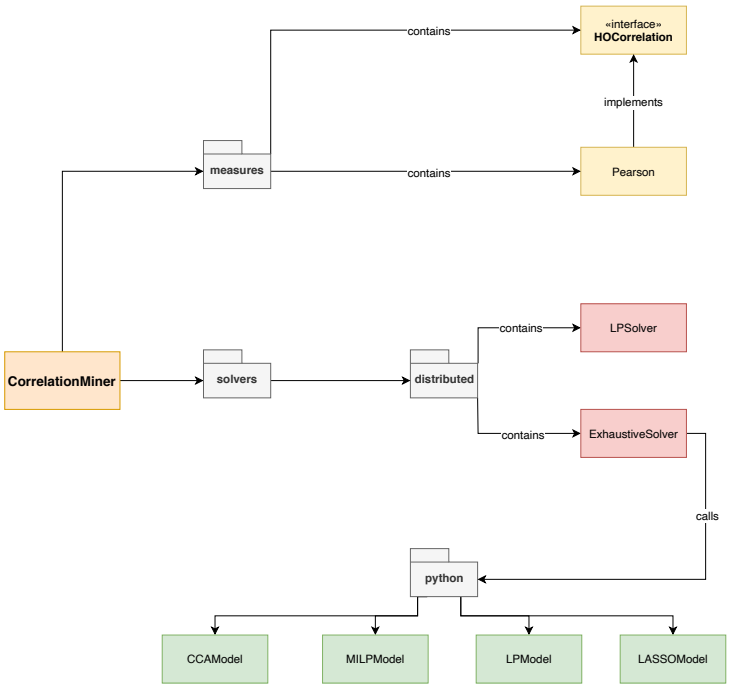


Figure 5.1: An overview of the Spark application architecture

The interface defines two methods C and f that can be implemented by a class provided by the user. In this way, the overloading of C and f are supported by the application, facilitating a plug-in component architecture that admits many different measures of similarity and correlation. By default, the interface requires the implementation of at least one aggregation function named f, but more can be defined. The interfaces **AbstractAggregateTimeSeries** and **AbstractMultipleTimeSeries** are defined in appendix A and B.

Pearson We provide one implementation of **HOCorrelation**, a class **Pearson** that provides an implementation the Pearson correlation coefficient for C. In addition, it implements an aggregation function f. The aggregation functions range from a simple implementation such as the average function, which assigns $a_i = 1/k$ as the coefficient for each vector, to implementations that call the solvers that we will cover next.

5.2.2 Distributed solvers

We provide two distributed solvers. The first consists of a partial implementation of the linear programming solution that we will detail later. The second offers a distributed exhaustive solution to the higher order correlation maximization problem and accepts any instance of the higher order correlation interface detailed earlier.

5.2.2.1 Linear programming

Various Spark implementations for linear programming exist[28]. Two of these, the Spark TFOCS library and an implementation of Mehrotra's interior-point method, run in a distributed fashion. The latter should be better suited to handle large-scale linear programs[29]. The linear program described in subsection 5.2.3.4 can be ported to use [29]. Currently, we offer only a partial, unfinished implementation available as **LPSolver** as the solver API is not as flexible as that of the Gurobi solver that we used for the initial local solver implementation. While our running example dataset easily fits entirely into the memory of a single worker node in the cluster, a distributed linear programming solver is especially useful for datasets for which this does not apply.

5.2.2.2 Exhaustive solution

Recall the complexities of the exhaustive solution to problem 3.1. For some choices of C, the exhaustive solution requires us to evaluate all $O(2^n)$ subsets of S, i.e. it's powerset $\mathcal{P}(S)$. For other choices of C, particularly for pairwise correlation measures, the exhaustive solution requires evaluating all $O(3^n)$ disjoint pairs of subsets of S. Most works that present enumerative approaches for their choice of correlation measure have developed pruning constraints that can significantly reduce the number of subsets to evaluate [5] [3]. While reducing the number of subsets to evaluate is very important to developing a scalable solution, there are other ways to scale the exhaustive solution. In particular, we can distribute the subset evaluation over different worker nodes, so that they can be evaluated in parallel. We have implemented two distributed enumeration strategies *UnconstrainedEnumeration* and *ConstrainedPairwiseEnumeration*, that run on top of an Apache Spark cluster consisting of a master node and zero or more worker nodes. Both are made available as part of the **ExhaustiveSolver** class.

Unconstrained enumeration strategy First, we consider the *UnconstrainedEnumeration* strategy. In this strategy, we exhaustively evaluate all $O(2^n)$ subsets of S. Here, $k \geq 1$ denotes the number of worker nodes in the cluster. Now, let us define $X_j \subset S$ for all $1 \leq j \leq 2^n - 1$ to denote the subsets of S. Obviously, we cannot form all subsets prior to distributing them, we would quickly ran out of memory. Alternatively, we may distribute the ranges of subsets that a worker node is responsible for evaluating and leave the subset enumeration itself to the workers. If we denote the i^{th} worker node by $i \in \{1, .., k\}$, a naive strategy would have each worker be responsible for a subset of timeseries $\{T_i, .., T_{i+\frac{n}{k}}\} \in S$. In that case, any subset X_j with cardinality $|X_j| > k$ would be evaluated more than once. Instead, we can obtain a more efficient partitioning by observing that the subsets can be represented using a binary number of n bits[30].

Definition 5.1. Binary representation of subsets Let $S = \{T_1, .., T_n\}$ be a set of n real-valued, standardized timeseries with $T_i \in \mathbb{R}^d$ for all $1 \leq i \leq n$. Moreover, let $\mathcal{P}(S)$ denote the powerset of S, i.e. the set containing all subsets of S. Then we can represent each subset of S by a binary sequence $\{0, 1\}^n$ using the bijection $f : \{0, 1\}^n \to \mathcal{P}(S)$ mapping the binary sequence to $\mathcal{P}(S)$.

Using this binary mapping, we can evenly distribute the workload over the k workers by denoting each worker with an integer $i = \{1, .., k\}$ and having every i^{th} worker start from a subset represented by the binary equivalent of i. That is, i represents the first subset to evaluate after which the worker continues enumerating and evaluating subsets, stepping by k subsets on each iteration. Since a subset processed by any worker i can contain any of the n timeseries of S, we

need to replicate S to every one of the k workers in the cluster. Hence, we construct an RDD of k tuples $< i, S >$. This RDD is then partitioned k times such that each of its k records is sent to the corresponding worker i.

This results in the following initialization step executed on the master node

Algorithm 1: InitializeUnconstrainedEnumeration(k)

Result: An RDD workerStarts of k tuples $\langle i, S \rangle$
$workerStarts \leftarrow \{\}$ **while** $i < k$ **do**
  add $\langle i, S \rangle$ to workerStarts;
  $i \leftarrow i + 1$;
end
return *workerStarts*

After this step, each worker runs algorithm 2. The *UnconstrainedEnumeration* strategy does not directly support pairwise higher-order correlation measures of which the aggregation functions expects a pair of disjoint sets $X, Y \subset S$ as input. An easy adaptation of the strategy supporting pairwise measures would be to, for every $X_j \subset S$, evaluate the correlation measure C for all $\binom{|X_j|}{2}$ possible pairs of X_j. Of course, this adaptation is inefficient as we would be repeating evaluations for any subsets or supersets of X_j.

Algorithm 2: UnconstrainedEnumeration(k, i, S, C, f)

Result: A list of correlation coefficients dependent on the correlation measure C
$n \leftarrow |S|$;
$j \leftarrow i$;
$results \leftarrow \{\}$;
while $j < 2^n - 1$ **do**
  $X_j \leftarrow$ getSubsetOfTimeSeriesForNBitBinaryString(j);
  $results \leftarrow results + \{C(f(X_j))\}$;
  $j \leftarrow j + k$;
end
return results

Constrained pairwise enumeration strategy Instead, we offer a second enumeration strategy *ConstrainedPairwiseEnumeration* that directly supports pairwise measures by evaluating disjoint pairs of subsets of S. However, unlike the previous strategy, it adds a cardinality constraint on each of the subsets in the pair. Constraining the cardinality of the subsets is done for two reasons. First, restricting the cardinality of the subsets is often desirable to keep the resulting aggregate vectors interpretable. Second, in terms of scalability, there is a significant reduction in the number of subsets to evaluate. To see why, let us abuse notation a bit and consider m pairs of subsets $\langle X, Y \rangle$ to evaluate with $X, Y \subset S, X \cap Y = \emptyset$, then we have an upper bound $m \leq \binom{n}{|X|} \times n^{|Y|}$. Since we have the requirement that $X \cap Y = \emptyset$, m becomes significantly lower than this upper bound.

To implement this strategy, we define $p = \max\{|X|, |Y|\}, q = \min\{|X|, |Y|\}$ and make use of a routine described in [31]. Starting from a given binary string B_1 of which p bits are set to 1, this routine gives us the lexicographically next bit permutation that again has p bits set to 1. In this way, we can obtain the next binary string B_2 that represents the next subset to evaluate, since p bits being set to 1 means we are iterating over the subsets of cardinality p. Abbreviating this routine as *nextPBitBinaryString*, we obtain the following modified initialization step for this

strategy

Algorithm 3: InitializeConstrainedEnumeration(k)

Result: An RDD workerStarts of k tuples $\langle startingSubset, S\rangle$
$workerStarts \leftarrow \{\}$;
$nextSubset \leftarrow$ first binary string with p bits set to 1;
while $i < k$ **do**
  workerStarts(i) $\leftarrow \langle$nextPBitBinaryString(nextSubset), $S\rangle$;
end
return $workerStarts$

After the initialization step, the constrained pairwise enumeration strategy starts and iterates over all n-bit binary strings for which p bits are set to 1. Effectively, we thus only consider subsets $X_l \subset S$ with $1 \leq l \leq \binom{n}{p}$. Continuing, for each of these subsets X_l, we then iterate over all subsets $Y_r \subset S$ with $1 \leq r \leq n^q$. In each of those iterations, we evaluate $C(f(X_l, Y_r))$ iff $X_l \cap Y_r = \emptyset$. Finally, after having evaluated all pairs, we collect the results to the master node.

Algorithm 4: ConstrainedPairwiseEnumeration($k, xSize, ySize, i, startingSubset, S, C, f$)

Result: A list of correlation coefficients dependent on the correlation measure C
$n \leftarrow |S|$;
$p \leftarrow \max\{xSize, ySize\}$;
$q \leftarrow \min\{xSize, ySize\}$;
$l \leftarrow 1$;
$nextSubset \leftarrow startingSubset$;
$results \leftarrow \{\}$;
while $l + i \leq \binom{n}{p}$ **do**
  $X_l \leftarrow$ getSubsetOfTimeSeriesForNBitBinaryString(j);
  for $r \leftarrow 0$ **to** $\binom{n}{q-1}$ **do**
    $Y_r \leftarrow$ getSubsetOfTimeSeriesForNBitBinaryString(r);
    if $X_l \cap Y_r \leftarrow \emptyset$ **then**
      $results \leftarrow results + \{C(f(X_l, Y_r))\}$;
    end
  end
  $l \leftarrow l + k$;
  if $l < \binom{n}{p}$ **then**
    for $j \leftarrow 0$ **to** k **do**
      $nextSubset \leftarrow$ nextPBitBinaryString(nextSubset)
    end
  end
end
return results

To illustrate the division of the subsets over the cluster, example 5.1 considers running the unconstrained enumeration strategy for $n = 4$ timeseries and $k = 3$ workers.

Example 5.1. Worker $i = 1$
i: 1, binary representation: 1, subset: $\{T_0\}$
$i + k$: 4, binary representation: 100, subset: $\{T_2\}$
$i + k$: 7, binary representation: 111, subset: $\{T_0, T_1, T_2\}$
$i + k$: 10, binary representation: 1010, subset: $\{T_1, T_3\}$
$i + k$: 13, binary representation: 1101, subset: $\{T_0, T_2, T_3\}$

Worker $i = 2$
i: 2, binary representation: 10, subset: $\{T_1\}$
$i + k$: 5, binary representation: 101, subset: $\{T_0, T_2\}$
$i + k$: 8, binary representation: 1000, subset: $\{T_3\}$

$i + k$: 11, binary representation: 1011, subset: $\{T_0, T_1, T_3\}$
$i + k$: 14, binary representation: 1110, subset: $\{T_1, T_2, T_3\}$

Worker $i = 3$
i: 3, binary representation: 11, subset:$\{T_0, T_1\}$
$i + k$: 6, binary representation: 110, subset: $\{T_1, T_2\}$
$i + k$: 9, binary representation: 1001, subset: $\{T_0, T_3\}$
$i + k$: 12, binary representation: 1100, subset: $\{T_2, T_3\}$
$i + k$: 15, binary representation: 1111, subset: $\{T_0, T_1, T_2, T_3\}$

Discussion By distributing subset evaluation over a cluster of k worker nodes, both the unconstrained and the constrained enumeration strategies achieve a factor k speed up over their centralized counterparts. However, due to the complexity of $O(2^n)$ for the unconstrained enumeration strategy, adding 2 workers to the cluster would grant us only one more timeseries to evaluate than the same strategy running centralized. While this does not seem to make the unconstrained enumeration strategy provide an added benefit, we argue that it does provide a good starting point for further optimizations. In particular, when we assume a reduced search space as a result of applying pruning constraints such as those proposed in [5] [3], the factor k speed up can become more significant. Beyond the limitations on the strategies originating from the fundamental computational complexity of problem 3.1, there is also a practical limitation to consider for the implementation of the constrained pairwise enumeration strategy. Specifically, the routine *nextPBitBinaryString* that we use to iterate over the binary strings for which p bits are set to 1 works only for binary numbers encoded using the long datatype. This means that we can only use the strategy up to $n = 64$. However, there is nothing preventing the implementation from being adapted to work with actual binary character strings (using the String datatype) in which case the limitation disappears.

The constrained pairwise enumeration strategy is already useful as an exhaustive solution for the cardinality constrained version of problem 3.2. It currently uses the aggregation function f implemented as part of the **Pearson** class. That implementation simply averages the timeseries for every input pair of subsets $X, Y \in S$ to produce the linear combinations X', Y' that C evaluates to $\rho(X', Y')$. A more interesting aggregation function could utilize the local solvers that we will detail next. Alternatively, the local solvers could be called on the entire set S by implementing a new solver that passes the (RDD of) S to the local solver implementations. Since the local solvers are implemented in Python, both approaches would have to make use of Spark's **pipe()** operation to pipe the RDD of S to the Python process running the local solvers.

Another point of consideration for both strategies is that of maximization. Currently, we do not take into account the maximization of the higher-order correlation values found for whatever measure C is chosen to evaluate the subsets. This is worth noting as strictly speaking we have defined problem 3.1 as a maximization problem. We can consider maximization by simply sorting the *results* lists that we collect from the workers and select only the subsets with the largest higher-order correlation values. Here, we could consider an approach similar to [3], in that we do not report "redundant" subsets of $X \subset S$ whenever there exists a subset $X' \subset S$ for which it holds that $X \subset X'$ and $C(X) \geq C(X')$.

5.2.3 Local solvers

We provide a number of local solvers that run in a centralized fashion. They can also be called by one of the distributed solvers, such as the distributed exhaustive solver we just discussed.

5.2.3.1 CCA (problem variation)

When defining problem variation 3.3, we identified canonical correlation analysis (CCA) as a possible solution to the problem. In particular, we noted that the first canonical pair X'_1, Y'_1 that

CCA produces will have the highest Pearson correlation coefficient $\rho(X'_1, Y'_1)$. In the context of our running example dataset, we identified a motivating application in example 3.1. Using CCA, we let $X = P$ and let the first canonical variate pair (X'_1, Y'_1) guide us in our diversification strategy for P. However, X'_1 will not necessarily match our current linear combination of stocks P'. Therefore, any higher-order correlation it finds will not always hold for the linear combination that we hold the stocks in. To resolve this, we can enforce the linear combination we hold the stocks in by pre-aggregating P' into a single timeseries $T_{P'}$ that forms our input set X for the CCA. In this way, CCA maximizes $\rho(P', P^*)$.

We mentioned a limitation of CCA in its regular formulation in that it does not produce sparse linear combinations. This poses a problem to our setup of the problem for example 3.1, as there we want to only add a limited number of stocks to our portfolio P. As discussed in the previous chapter, various works have proposed sparse variants of CCA [15] [16] [11]. However, many of these works rely on the diagonalization of the covariance matrix. As a consequence, they hinge on the assumption that the variables in the dataset are independent. Since our stocks timeseries instead have significant collinearity, one existing implementation for sparse CCA [32] cannot readily be used as it is based on the penalized matrix decomposition of [15] that also makes this assumption. In addition, it is our understanding that many interesting datasets also contain collinear variables making the mentioned aforementioned implementations of sparse CCA a bad fit for these cases.

Therefore, we have instead used regular CCA and obtained sparse solutions through a different approach. To determine $X, Y \subset S$, we proceed as follows: we initialize $X = \{T_1, .., T_l\}, Y = S \setminus X$ where the stocks represented by timeseries $T_1, .., T_l$ are picked at random (since in this case we have no preference for our starting portfolio) and we then run CCA. Now, CCA will produce linear combinations X'_1, Y'_1 of which we will pick $P' = X'_1 = a_1T_1 + .. + a_lT_l$ to form our initial portfolio. This linear combination will form an aggregate timeseries $T_{P'}$. For Y'_1, we then sort coefficients $a_{l+1}, .., a_n$ along with their timeseries $T_{l+1}, .., T_n$ in descending order. From this sorted set of coefficients of Y'_1, we take the top k coefficients $a_1, .., a_k$ and their timeseries $T_1, .., T_k$ such that $k \leq n - l$, obtaining a linear combination $K' = a_1T_1 + .. + a_kT_k$. We form an aggregate timeseries $T_{K'} = a_1T_1 + .. + a_kT_k$ of and then compute the Pearson correlation $\rho(P', K') = \rho(T_{P'}, T_{K'})$. Depending on how large we pick k, this will decrease the correlation coefficient with P'. In other words, we would have that $\rho(T_{P'}, T_{K'}) \leq \rho(X'_1, Y'_1)$. To again maximize the correlation, we then re-run CCA with $X = T_{P'}, Y = \{T_1, .., T_k\}$, i.e. where Y consists of just the vectors for which their coefficients $a_1, .., a_k$ were part of the top k coefficients. If we denote this last run's linear combinations as X^*, Y^*, then we have that $X^* = T_{P'}$ and $Y^* = a_1T_1 + .. + a_kT_k$ is a linear combination such that $\rho(X^*, Y^*)$ is maximized.

We will experimentally verify the above approach to obtaining sparse vectors for which the correlation is maximized.

Discussion The strategy we just described for regular CCA achieves its goal of obtaining sparse linear combinations X', Y'. However, we only experimentally verify the approach, judging its performance by looking at the Pearson correlation coefficient $\rho(X', Y')$ relative to the sparsity we obtain by picking a suitable value for k. Hence, we feel that a complete verification of this approach to obtaining sparse solutions would also evaluate its soundness from a statistical perspective. Since this thesis focuses on efficiently, i.e. scalably, maximizing the higher-order correlation between timeseries, we consider this verification outside the scope.

We should also note that regular CCA too can have issues with multicollinearity [33]. Therefore, trying to "circumvent" the use of sparse CCA using this approach may not actually help us if we are dealing with significantly collinear timeseries in S.

5.2.3.2 LASSO

In chapter 3, we explained how the LASSO may be used as a way of determining a sparse linear combination Y' that maximizes the Pearson correlation coefficient with a pre-specified linear combination X' forming the dependent variable. Therefore, the LASSO regression procedure can not directly be used to solve problem 3.2. To overcome this issue, we can exhaustively construct pairs of sets $X_i = \{T_i\}, Y_i = S \setminus X$ with $1 \leq i \leq n$ and form the n linear combinations $X'_i = a_iT_i$

by using a simple aggregation f such that $f(X_i) = X'_i$. Using a LASSO regression of X'_i on Y_i, we obtain $Y'_i = a_1T_1 + .. + a_kT_k$ with $k \leq n$. We can extend this exhaustive approach to $O(n^2)$ linear combinations by having X_i consist of all pairs of $T_i \in S$ and similarly for $O(n^3)$ linear combinations with X_i consisting of all triples etc.

To this end, we have implemented a local solver **LASSOModel** that uses a simple average aggregation function f producing a linear combination $X' = a_1T_1 + .. + a_lT_l$ by setting $a_i = \frac{1}{l}$ for every timeseries in the input set $X \subset S$ that f operates on to produce X'.

Discussion The most obvious downside to this solver is that its complexity is $O(n^{|X|})$, i.e. it is exponential in the cardinality of the set of dependent variables X. Therefore, it will be hard to scale the model to a large amount of timeseries depending on the desired number of timeseries in X. For $|X| = 3$, evaluating all possible timeseries for $n = 1000$ timeseries would already require the LASSO procedure to be ran 1.000.000 times. A second issue is that by opting for a simple average as our aggregation function f, we do not optimize the coefficients of X'.

On a more positive note, the LASSO procedure is a well-studied technique to obtaining sparse solutions in linear regression models. In addition, an efficient distributed implementation is available on Apache Spark and hence developing a distributed solver utilizing this implementation seems very feasible. In this case, a distributed solver version of this model would support datasets where the vectors are too large to be processed in a centralized fashion. That being said, such large vectors would increase the cost of repeatedly running the LASSO procedure, which would be problematic when considering larger values of $|X|$.

One interesting remark by the authors of [2] when discussing work related to their work on tripoles is that regularized regression techniques like the LASSO can not be considered to be directly suited to their problem formulation. It is true that without the notion of jump it is not easy to identify tripoles from the results of a regression on the timeseries in S. At the same time, the cardinality constraints that their tripoles concept impose upon the sets X, Y do make the LASSO a possible computationally viable candidate for extending their restricted aggregation function f into a more flexible one that supports differently weighing the timeseries in Y. With their constraint that $|X| = 1$ and $|Y| = 2$, we could imagine running the LASSO solver for each $T_i \in S$. In that case, we would thus invoke the LASSO solver for a total of n times. With a sufficiently high value for the penalty parameter λ we could satisfy the constraint that $|Y| = 2$. Then, using their notion of jump, the interesting tripoles could be identified from this set of n potential tripoles.

5.2.3.3 Mixed integer linear program

The techniques for the local solvers we have considered so far all assume that X is known, which is not the case in problem 3.2. Therefore, we have considered models that do not make this limiting assumption.

First, consider linear combinations X', Y' that we aim to maximize $\rho(X', Y')$ for. Observe that if

$$\begin{aligned} & X' = Y' \\ \iff & \sum_{i=1}^{n} a_i T_i = \sum_{i=1}^{n} a'_i T_i \end{aligned} \tag{5.1}$$

then $|\rho(X', Y')| = 1$. Hence, if we would develop a solution with the goal of reducing the difference between X' and Y', we should be able to maximize the correlation.

However, we need a number of constraints to exclude invalid solutions that do not conform to the definition of problem 3.2. One obviously invalid solution would be setting all $a_i, a'_i = 0$. In addition, there is the constraint that $X \cap Y = \emptyset$, meaning that in 5.1 we cannot have that $a_i = 1, a'_i = 1$ when. Finally, even if we encode these constraints into our solution, it would be unlikely to find two linear combinations of different timeseries that are exactly equal to each other. But this should not be necessary, as we are mainly looking to maximize $\rho(X', Y')$, thus we do not need a perfect correlation. Thus, we can allow X', Y' some "slack" by adding error terms that

capture the difference between X' and Y'. Such error terms should be vectors themselves, thus we declare two residual vectors $r, r' \in \mathbb{R}^d$.

Finally, we can state the above problem as an optimization problem, specifically a linear program where the objective is to minimize the sum of the values of residual vectors r, r'. To prevent the case where all coefficients are set to zero, we add two constraints that state that the sums of the coefficients a_i, a'_i for respectively X', Y' should be at least 1. We then obtain the following linear program

$$\begin{aligned} \text{minimize } & \sum_{i=1}^{d}(r_i + r'_i) \\ \text{subject to } & \sum_{i=1}^{n} a_i T_i + r = \sum_{i=1}^{n} a'_i T_i + r' \\ & \sum_{i=1}^{n} a_i \geq 1 \\ & \sum_{i=1}^{j} a'_i \geq 1 \\ & b_i, b'_i \in \{0, 1\} \end{aligned} \tag{5.2}$$

The above is not yet complete, as the current formulation does not have a constraint that guarantees that any $T_i \in X', Y'$, i.e. that T_i occurs in both linear combinations. To prevent this, we add a constraint that guarantees that $a_i > 0$ if and only if $a'_i = 0$ and vice-versa. In fact, we found that requiring a_i to be positive while $a'_i = 0$ and vice-versa gave the best results. To express this in the linear program, we introduce binary variables $b_i, b'_i \in \{0, 1\}$ for every pair a_i, a'_i and obtain

$$\begin{aligned} \text{minimize } & \sum_{i=1}^{d}(r_i + r'_i) \\ \text{subject to } & \sum_{i=1}^{n} b_i a_i T_i + r = \sum_{i=1}^{n} b'_i a'_i T_i + r' \\ & a_i, a'_i \geq 0.01 \text{ for all } 1 \leq i \leq n \\ & \sum_{i=1}^{n} a_i \geq 1 \\ & \sum_{i=1}^{n} a'_i \geq 1 \\ & b_i + b'_i = 1 \text{ for all } 1 \leq i \leq n \\ & b_i, b'_i \in \{0, 1\} \end{aligned} \tag{5.3}$$

The introduction of the binary variables turns the program into a mixed-integer linear program. Integer programming is known to be NP-hard. It is not possible to constrain the coefficients without the use of integer variables due to the convexity of the feasible region of a linear program. To see why, consider the following example.

Example 5.2. Consider two feasible solutions, one where $a_1 > 0, a_2 = 0, a'_1 = 0, a'_2 > 0$ and another where $a_1 = 0, a_2 > 0, a'_1 > 0, a'_2 = 0$. By convexity of the feasible region, we have that $a_1 > 0, a'_1 > 0, a_2 > 0, a'_2 > 0$ would also be a feasible solution. The latter solution would violate our constraint that $X \cap Y = \emptyset$ as $T_1, T_2 \in X', Y'$.

The mixed-integer linear program has been implemented as a centralized solution **MILP-Model** using Gurobi. By default, the solver will search for the optimal solution, i.e. the solution

where the sum of the residual vectors r, r' is minimal. To obtain multiple higher-order correlations we may extract multiple sub-optimal solutions that the solver retains.

Discussion The mixed-integer linear program seems to give fairly reasonable results given its constraints, and has the added benefit that multiple sub-optimal solutions can be extracted from the solver so as to obtain multiple solutions. However, a major problem is the use of binary variables to enforce the constraint that $X \cap Y = \emptyset$. This means that the scalability of the model to larger values of n is greatly reduced due to the NP-hardness of integer programming, as its running time may become exponential in n.

In terms of sparsity of the solutions, a problematic constraint is that we require the model to set $a_i >= 0.01$ whenever $a'_i = 0$ and vice versa. While we found that this makes the model assign reasonable weights to the included timeseries, there is a downside in that it will by default include all timeseries as part of the solution. One option is to try and penalize $\sum_{i=1}^{n} |a_i|, \sum_{i=1}^{n} |a'_i|$, which is what the LASSO regression procedure uses to achieve sparsity. However, this does not make sense for our model as we explicitly require that $\sum i = 1^n a_i \geq 1, \sum_{i=1}^{n} a'_i \geq 1$ to make sure that the model will not consider setting all coefficients to zero to be a valid solution.

Instead, we may induce sparsity by taking the top k coefficients from the model as we did for the CCA solution to the problem variation. In addition, to further maximize the correlation, we re-run the optimization for just the timeseries that had their coefficient be part of the top k. This is again similar to our approach with CCA.

5.2.3.4 Linear program

Due to the NP-hardness of integer programming, the mixed-integer linear programming solution may be exponential in the worst case. Since the problematic constraint will require binary variables in its current form, we may instead reformulate the linear program such that the binary variables can be avoided.

We reformulate the constraint

$$\sum_{i=1}^{n}\sum_{j=1}^{d} b_i a_i T_i + r_j = \sum_{i=1}^{n}\sum_{j=1}^{d} b'_i a'_i T_{ij} + r'_j \tag{5.4}$$

as

$$\sum_{i=1}^{n}\sum_{j=1}^{d} a_i T_{ij} - r_j = 0 \tag{5.5}$$

and obtain the following linear program

$$\begin{aligned} \text{minimize } & \sum_{i=1}^{d} r_i \\ \text{subject to } & \sum_{i=1}^{n}\sum_{j=1}^{d} a_i T_{ij} - r_j = 0 \\ & \sum_{i=1}^{n} a_i = 1 \\ & r_i \geq 0 \text{ for all } 1 \leq i \leq d \end{aligned} \tag{5.6}$$

In contrast to the mixed integer linear program, the linear program will report a single optimal solution for its input.

Discussion The obvious benefit to the linear program formulation without the binary integer variables is that it scales a lot better. However, this comes at the cost of accuracy as we are circumventing the need for these variables by constructing the linear combination Y' from the negative coefficients $a_i < 0$. Another problem is with the subtraction of the residual vector. Since

standardized timeseries may contain negative values, for some of the d dimensions it may happen that all values are negative.

In that case, the model will become infeasible because subtracting r_i in that case will make it impossible to satisfy the constraint $\sum_{i=1}^{n} a_i T_i - r = 0$ as we cannot make the value negative for the residual vector in that dimension. We can handle these cases by either skipping those dimensions entirely, which is not desirable, or we can make an exception for these dimensions and instead decide to change the constraint to $\sum_{i=1}^{n} a_i T_i + r = 0$ instead. We have chosen for the latter. In addition, the constraint that $\sum_{i=1}^{n} a_i = 1$ will favour positively valued coefficients a_i over negative coefficients. This means that running the linear program with small values of n will cause $Y' = 0$ in some cases due to there being no $a_i < 0$.

For obtaining sparse solutions we can apply the same reasoning as with the mixed-integer linear program. Obtaining multiple solutions is not immediately possible for the linear program, as it reports a global optimum. However, we may combine the top k coefficient selection to get a sparse solution with the re-running of the linear program with starting set $S \setminus S'$ where $S' = X^* \cup Y^*$ where X^*, Y^* consist of the timeseries $T_i \in X, T_i in Y$ for which their a_i is part of the top k coefficients. This will allow us to obtain multiple solutions for the linear program.

Chapter 6

Experimental evaluation

We have evaluated the centralized and distributed solvers of the CorrelationMiner introduced in chapter 5 in terms of their quality and performance on a subset of the stock timeseries dataset that forms the running example dataset of this thesis. The results of this evaluation are presented in this chapter, following the structure of chapter 5 in that the results are presented per solver. Before presenting the results, we first elaborate on the preprocessing steps necessary to derive the running example dataset that the evaluation has been conducted on. Moreover, as part of the initial exploration of the dataset, we introduce the notion of an interesting higher-order correlation and show a number of interesting higher-order correlations we found in the dataset.

6.1 Data preparation

The original dataset consists of timeseries of stock price values for stocks in the stock market spanning a period of two years. They were extracted through an API gratuitously provided to us by Spring Techno. Using their API, stock prices and their timestamps were queried and saved for the period of January, 2018 up to and including March, 2019. This resulted in 15 files per stock, one for each month. The structure of a file for a stock in a certain month consists of all available timestamps and corresponding prices for that stock in that month. For the preparation of a convenient, clean dataset that can be used in the CorrelationMiner application, we developed a Java application that reads in these stock files and applies the preprocessing steps detailed next.

The application produces, for every stock, a CSV file containing the columns $id, time, value$ and d rows with each row consisting of respectively the name (id), timepoint and value for that dimension. Here, d is the number of dimensions each stock should have and is specified by the user. However, working with the clean dataset in the form of multiple CSV files is not convenient for use in the CorrelationMiner application nor the Python implementation of the centralized solvers. Therefore, we use an additional Python module that is responsible for the final preprocessing steps consisting of the standardization of the timeseries and the production of a single output file consisting of preprocessed timeseries. To make the data preparation process fully transparent, the relevant implementation has been made available as part of a Java class **StocksReader** and a Python module **data_utility** available with the CorrelationMiner application. The stocks timeseries dataset itself is proprietary and is thus not included with this thesis.

6.1.1 Preprocessing

6.1.1.1 Combining monthly timeseries input files

As a first preprocessing step, we combine for each stock the 15 monthly timeseries that we read from the input files into a single (in-memory) timeseries per stock. For some stocks, in certain months, there were the occassional invalid timestamps or price values extracted from the API. This resulted in malformed lines being written to the corresponding monthly files for these stocks.

Since these cannot be used, they are immediately excluded during concatenation. This may result in missing dimensions for the stocks for which this occurs. This is not a problem however, as we shall correct this during the time-alignment step.

6.1.1.2 Time alignment

In an initial exploration of the data extracted from the API, we found that the timestamps provided by the API were not always equal when comparing the different stocks. Often, the amount of minutes between timestamps differed between stocks. Due to differences in stock exchanges, it could also happen that one stock's first reported timestamp differed hours from another stock's first reported timestamp. To find correlations between time series, it is important that time series are comparable and hence that they are time-aligned.

Thus, we apply a best effort time-alignment step to construct time-aligned timeseries. In this additional preprocessing step, we start with a user-specified parameter $amountOfWorkdays$ corresponding to the amount of workdays that the timeseries should span. Then, based on another user parameter $amountOfMinutesPerTimepoint$, the amount of minutes per timepoint is used to compute the possible timepoints within the timeframe of the workdays. This workday timeframe is based on a 9-5 business hour schedule. These timepoints form so-called anchor timepoints, together making up a pivot timeseries to which every individual time series will be aligned as best as possible.

Continuining the alignment procedure, every timeseries is scanned linearly and, for every anchor timepoint at_i defined by the pivot timeseries, the timepoint $t_i \in T$ closest to at_i is picked greedily. The resulting time series has a dimensionality equal to that of the reference time series and hence the desired minute granularity. Once all time series are processed by this best effort alignment procedure, another pass over the time series is done as a verification step. The reason for doing an additional verification step is to make sure that the timepoints we picked are close enough to the amount of minutes per timepoint the user specified. Therefore, in the verification step, we compute the minute delta $\Delta(t_{i+1}, t_i)$ between the timepoints we picked for the timeseries and check whether $\Delta(t_{i+1}, t_i) \leq 1.75 \times amountOfMinutesPerTimepoint$. If it exceeds this threshold, we increment a counter. Finally, for every time series, we check if the value of this counter is within 5% of the total amount of timepoints (dimensions) we determined. If this is not the case, we discard the timeseries from being considered for S.

6.1.1.3 Standardization

Recall that for the higher-order correlation maximization problem, we assume the input timeseries to be standardized. Thus, as one of the final preprocessing steps, we standardize the timeseries such that each stock timeseries has centered mean $\mu = 0$ with unit variance $\sigma^2 = 1$.

6.1.1.4 Combining CSV files

The last preprocessing step consists of the reading of each of the CSV files produced by the **StocksReader** Java application and combining them into a single Apache Parquet file **stocks.parquet** using the Python module **data_utility**.

6.1.2 The running example dataset

We experimented with multiple time periods within the full 2-year time period of the original stocks dataset. We found that a continuous chunk of 458 time-aligned stock timeseries could be obtained for a 120 workday period, between January and June 2018. Hence we end up with our running example dataset S consisting of 458 stock timeseries with each stock consisting of $d = 11520$ dimensions, where each dimension represents about 5 minutes.

To get an idea of the degree to which we deal with multicollinearity in this dataset, we computed the average of the $O(n^2)$ pairwise correlations for all pairs of stocks $(T_i, T_j) \in S$.

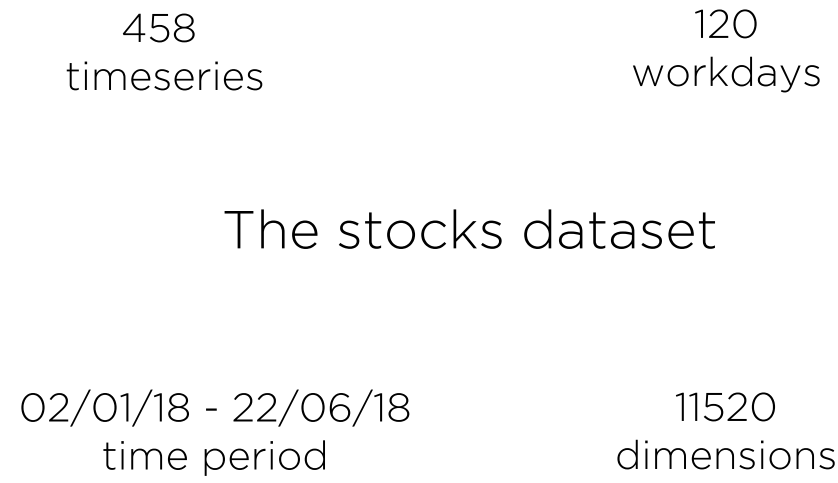


Figure 6.1: An overview of the running example dataset S

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{n}|\rho(T_i, T_j)| = 0.42 \tag{6.1}$$

6.2 Interesting higher-order correlations

In chapter 3, we introduced canonical correlation analysis (CCA) as a technique to find two linear combinations such that the correlation between them is maximized. In the running example dataset, using CCA, we have identified a number of higher-order correlations we deem interesting. The unconstrained maximization of higher-order correlations does not have a notion of "interestingness". Therefore, let us first define what makes an higher-order correlation interesting.

Definition 6.1. Interesting higher-order correlation Let $\rho X, Y$ be a higher-order correlation between We define an interesting higher-order correlation to be a higher-order correlation for which it holds that

(i) $|\rho(X', Y')| \geq \tau$

(ii) $|\rho(X', Y')| - |\max \rho(T_i, T_j)| \geq \epsilon$

with $T_i \in X$, $T_j \in Y$ and where $0 \leq \tau, \epsilon \leq 1$ are user-defined thresholds.

In other words, we deem a higher-order correlation interesting when there is a significant difference between the value of the higher-order correlation and the value of the maximum pairwise correlation of any of the timeseries participating in linear combinations X', Y'. It is similar to the concept of jump defined for a tripole in [2], although our definition is a bit more simplistic. To discover the interesting higher-order correlations, we construct the sets X, Y according to the procedure described in section 5.2.3.1 of the previous chapter and we run CCA. We repeat the CCA procedure m times for randomly picked sets $X_1, .., X_p$ consisting of $|X| = 2$ stocks. We end up with a list of size $p \times m$ from which we prune all results that do not conform to (the thresholds of) definition 6.1. Note that this list of interesting higher-order correlations is not exhaustive

and hence there may exist higher-order correlations that are even "more interesting" according to definition 6.1, in that the difference between their correlation $\rho(X', Y')$ and maximum pairwise correlation is larger.

The following table shows a number of interesting higher-order correlation we found in the stocks dataset with $|X| = 2$ and $k \in \{2, 3\}$ for the number of timeseries we retained from each linear combination Y'.

Table 6.1: Interesting higher-order correlations in the stocks dataset ($\tau = 0.7$, $\epsilon = 0.5$)

$\rho(X', Y')$	$\max \rho(T_i, T_j)$	X'	Y'
0.93	0.35	$0.75 Paris\_AI + 0.67 London\_TPK$	$0.75 Xetra\_660200 + 0.64 Paris\_OR + -0.16 Amsterdam\_ASM$
0.83	0.28	$0.76 Kopenhagen\_SYDB + -0.65 Mailand\_TIT$	$0.78 Amsterdam\_PHIA + -0.59 London\_CRH + -0.22 London\_SMIN$
0.79	0.25	$0.96 London\_INVP + 0.28 Oslo\_NAS$	$-0.72 Madrid\_REP + 0.7 London\_MRW$
0.74	0.21	$0.79 Amsterdam\_ASML + 0.61 London\_III$	$0.71 London\_HSBA + -0.7 Kopenhagen\_MAERSK$
0.79	0.28	$0.93 Paris\_GLE + -0.37 Xetra\_590900$	$-0.71 Xetra\_723610 + 0.71 Xetra\_577330$

6.3 Experimental evaluation

We have experimentally evaluated a number of the solvers that were implemented as part of the CorrelationMiner application introduced in the previous chapter. In addition, we have verified our solution to problem variation 3.3 using the CCA solver. In this section, we elaborate upon the setup of these experiments and discuss their results.

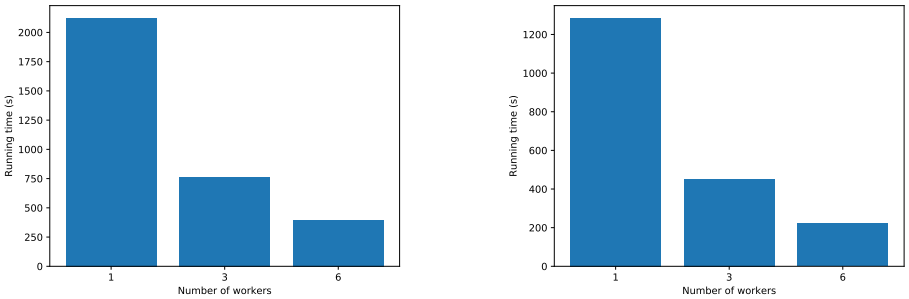
6.3.1 Distribution of the exhaustive solution

We introduced a distributed solver that distributes the work of evaluating the subsets in the exhaustive solution to problem 3.1 over a cluster of k worker nodes using two strategies. We have evaluated the performance of the unconstrained enumeration as well as the constrained pairwise enumeration strategies. The goal of the experiment was to verify that both strategies indeed achieve a factor k speed up in the exhaustive evaluation of the subsets of our set of timeseries S. To this end, we have setup 3 Apache Spark clusters on AWS EMR[34] with respectively $k = 1$, $k = 3$ and $k = 6$ workers per cluster. The instance type of all instances in each cluster is $c3.2xlarge$. The first cluster contains of just a single master node, while the other two clusters consist of respectively a master node and k worker nodes.

For the *UnconstrainedEnumeration* strategy, we set $n = 22$ stocks, which evaluates to $2^n - 1 = 2^{22} - 1 = 4194303$ subsets to be evaluated. For the *ConstrainedPairwiseEnumeration* strategy, we start from $n = 64$ stocks, which is currently the maximum due to an implementation detail of the solver, as discussed in section 5.2.2.2. We then configure it to evaluate all pairs $X, Y \subset S$ for which $|X| = 2, |Y| = 2$. This means we need to evaluate at most $\binom{n}{|X|} \times n^{|Y|} = \binom{64}{2} \times 64^2 = 8257536$ pairs. However, we exclude the pairs for which it holds that $X \cap Y \neq \emptyset$, hence we only evaluate 3812256 pairs.

6.3.2 Problem variation (CCA)

For problem variation 3.3, we verify the approach to obtaining sparse solutions described in subsection 5.2.3.1 on the stocks dataset. The verification steps consists of comparing X^* against p random sets of stocks $\phi = \{Y_1, .., Y_p\}$ and running CCA on each of the p sets. Note that the number of stocks in each set $Y_i \in \phi$ is equivalent to k, the number of stocks part of linear combination Y^*. We then end up with a linear combination Y'_j for each $Y_j \in \phi$ and compare visually whether $\rho_{(X^*, Y'_j)} \leq \rho_{(X^*, Y^*)}$. The reasoning here is that if we can use CCA to determine a sparse linear combination Y^* for which correlation is maximized with X^*, then $\rho_{(X^*, Y'_j)}$ should be significantly higher than the correlation of X^* with linear combinations of k randomly selected stocks. We have collected results for a total of 4 randomly picked stock portfolios P'_1, P'_2, P'_3, P'_4. For each portfolio



(a) Unconstrained enumeration, $n = 22$

(b) Constrained pairwise enumeration, $n = 64, |X| = 2, |Y| = 2$

Figure 6.2: Scalability of the distributed subset enumeration strategies

P_i' with $1 \leq i \leq 4$, we set $p = 50$ and obtain sets $\phi_1, \phi_2, \phi_3, \phi_4$. Moreover, the following holds for the portfolios:

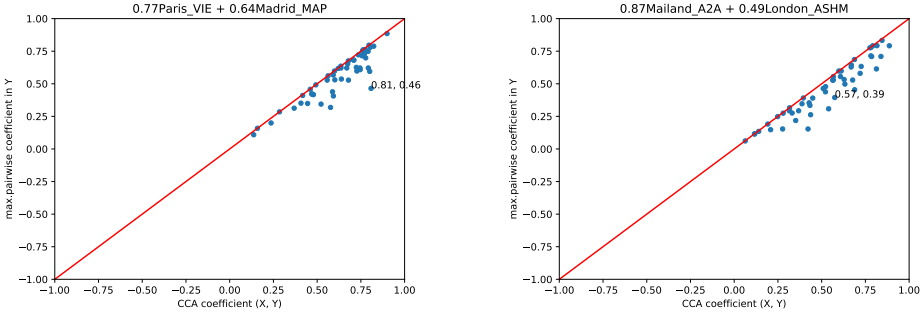
- Portfolio P_1' is a linear combination of 2 stocks compared to linear combinations of $k = 2$ stocks
- Portfolio P_2' is a linear combination of 2 stocks compared to linear combinations of $k = 2$ stocks
- Portfolio P_3' is a linear combination of 2 stocks compared to linear combinations of $k = 1$ stocks
- Portfolio P_4' is a linear combination of 3 stocks compared to linear combinations of $k = 2$ stocks

The results of the verification for these portfolios are depicted in the scatterplots of figures 6.3 6.4. On the x-axis, we plot $\rho_{(P_i', Y_j')}$. On the y-axis, we plot $\max \rho(P_i', T_l)$ with $T_l \in Y_j$, i.e. it represents the maximum pairwise correlation of the portfolio with any of the stocks $T_l \in Y_j$. The first dot in the plot is labeled and its x-value denotes $\rho_{(P_i', Y_i')}$ while its y-value denotes the maximum pairwise correlation between the portfolio P_i' and any of the stocks in part of the linear combination Y_i'. Here, we use Y_i' to denote the linear combination of k stocks that we obtained for P_i' using the approach to extracting sparse solutions using CCA. The reason for plotting the maximum pairwise correlations of the portfolio with any of the stocks in Y_j' is to see if any stock in particular is a major contributor to the correlation with the portfolio.

From the above figures, it can be seen that in quite a number of cases the random linear combinations actually end up having a higher correlation with the portfolio than does the original linear combination that we initially determined using CCA. However, in all of these cases, a single timeseries participating in the random linear combination seems to be a major contributor to the higher correlation coefficient. This since the maximum pairwise correlation of our portfolio with any individual timeseries in the original linear combination found by CCA.

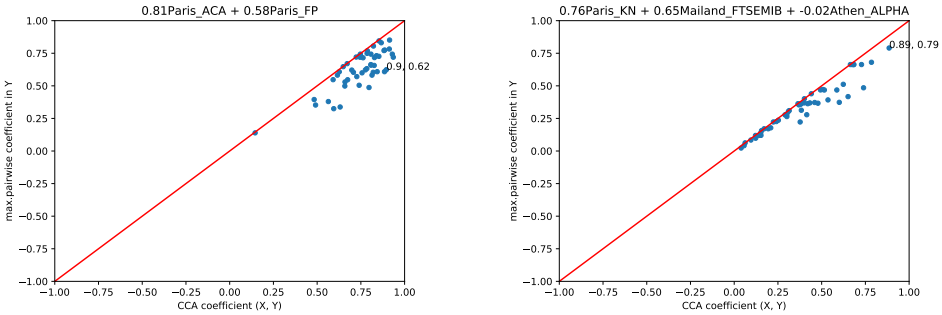
6.3.3 Correlation maximization performance of the linear programs

We have presented a mixed-integer linear program as well as a linear program that aim to solve problem 3.2. To evaluate these models, we evaluated their performance on maximizing the Pearson correlation coefficient for a series of k random stocks against CCA. Since CCA requires us to specify the input sets $X, Y \subset S$, we ran CCA on all the possible $\binom{k}{2}$ ways to form X, Y from the k stocks and then picked the model with the highest CCA coefficient. We then plotted its results to that



(a) Portfolio P_1' (b) Portfolio P_2'

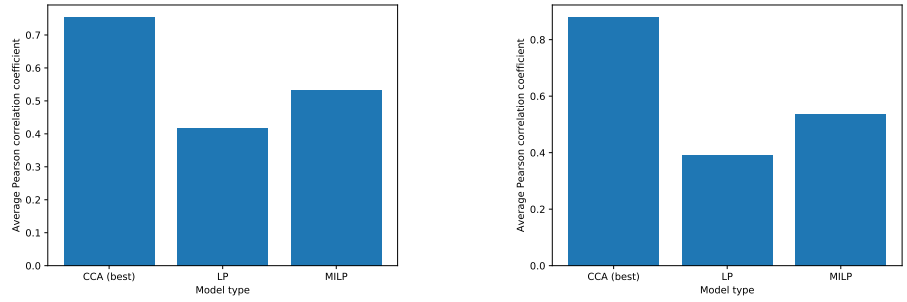
Figure 6.3: Verification of 2 stock portfolios P_1', P_2' against sets of randomly picked stocks



(a) Portfolio P_3' (b) Portfolio P_4'

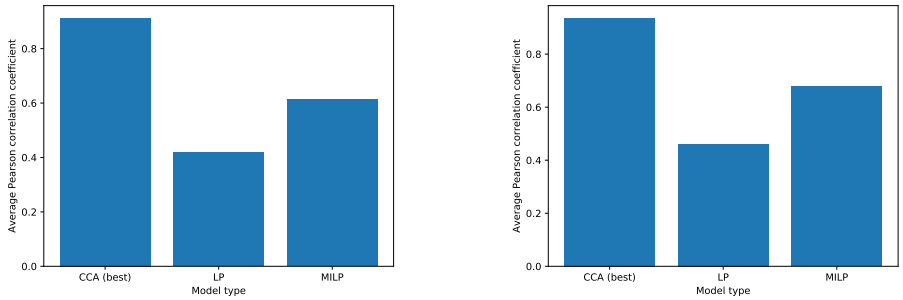
Figure 6.4: Verification of 2 stock portfolios P_3', P_4' against sets of randomly picked stocks

of the mixed-integer linear program and linear program on the same set of k stocks. Since the mixed-integer linear program has a very long running time, we set $d = 1000$ for this experiment, repeating the experiment for $r = 50$ iterations. We ran the experiment for $k = 3, 4, 5, 6$. The results are presented in figures 6.5 and 6.6.



(a) $k = 3$ stocks (b) $k = 4$ stocks

Figure 6.5: Comparison of correlation maximization performance of the mixed-integer linear program, linear program and the best CCA model for $k = 3$ and $k = 4$ stocks



(a) $k = 5$ stocks

(b) $k = 6$ stocks

Figure 6.6: Comparison of correlation maximization performance of the mixed-integer linear program, linear program and the best CCA model for $k = 5$ and $k = 6$ stocks

6.3.4 Multiple sparse solutions of the linear program

While discussing the linear programs as part of sections 5.2.3.3 and 5.2.3.4, we briefly explained how we could obtain multiple sparse solutions for both linear programs. We have evaluated the strategy by iterating through the available sparse solutions of size $k = 3$ for the linear program and inspecting the linear combinations X', Y' they produce as well as their correlation coefficient $\rho(X', Y')$.

For the linear program, the results of the first run of the optimization are included in table 6.2 For the mixed-integer linear program, the running time prohibited us from obtaining meaningful solutions for this experiment.

6.3.5 Mixed integer linear program running time

The mixed-integer linear program we formulated has worst-case exponential running time due to the use of integer (binary) variables. We confirmed this by executing the model on an increasing number of stocks $n = 5, .., 20$ and plotting the running time.

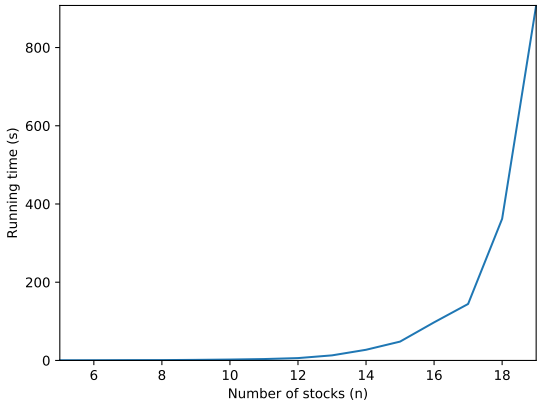


Figure 6.7: MILP running time

Table 6.2: Multiple sparse solutions for the linear program, $k = 3$

$\rho(X', Y')$	X'	Y'
−0.92	$0.06 Mailand\_ENI + 0.049 Mailand\_UCG + 0.046 Amsterdam\_BOKA$	$0.167 Mailand\_FTSEMIB + 0.029 Amsterdam\_RDSA + 0.024 London\_BT\_A$
−0.71	$0.083 Paris\_RMS + 0.045 Paris\_AF + 0.042 Stockholm\_SWMA$	$0.027 London\_SVT + 0.026 Amsterdam\_WKL + 0.025 Amsterdam\_FUR$
−0.73	$0.056 Xetra\_LED400 + 0.052 Paris\_SW + 0.048 Mailand\_MONC$	$0.031 London\_SMDS + 0.026 Kopenhagen\_DANSKE + 0.025 Amsterdam\_SBMO$
−0.28	$0.047 Paris\_ILD + 0.043 Oslo\_PGS + 0.043 Helsinki\_STERV$	$0.034 Amsterdam\_REN + 0.031 London\_VOD + 0.028 Xetra\_623100$
−0.37	$0.053 Paris\_DSY + 0.043 Madrid\_ACS + 0.04 London\_HTG$	$0.034 London\_STJ + 0.03 Paris\_AI + 0.028 London\_ULVR$

6.4 Discussion

We were provided an API that allows various stocks and their price movement through time to be queried. Using this API, we extracted a timeseries dataset that we then applied a number of preprocessing steps to in order to extract a time-aligned dataset that we could use as our input set S of timeseries. The time-alignment has been applied in a best-effort manner and we have chosen the threshold to be on the conservative side to avoid including misaligned time-series. It may still occur that misaligned time-series are included in S however, but given the quality of the original input dataset of timeseries this is hard to prevent.

Using this set S, we then evaluated the distributed and local solvers introduced with the CorrelationMiner application in chapter 5. The exhaustive enumeration strategies of the distributed solver were confirmed to achieve their factor k speed up, although this speed up is relatively minor in light of the fundamental complexities of the exhaustive solutions. We used CCA to discover interesting higher-order correlations as well as to tackle problem 3.3.

The results showed that in various cases the original linear combination determined by CCA to be the best for the portfolio had a lower correlation with the portfolio than randomly chosen linear combinations of stocks. This may indicate that our method of inducing sparsity in the CCA solution is flawed, or it may simply mean that evaluating random linear combinations of k stocks is not a good approach to verify this method. The latter is especially worth pondering when considering the fact that correlations can be spurious. Although it is worth noting that the maximum pairwise correlation of any of the timeseries participating in the original linear combination with the portfolio was in all cases lower than the equivalent for the random linear combinations. In that sense, we could argue that we can ignore these points because the pairwise relationship of one of their timeseries already explains most of the relationship, i.e. there is no "higher-order" effect. This would be somewhat similar to how the work of [2] uses the notion of jump to distinguish between interesting tripoles and non-interesting ones. However, we should be careful with this line of reasoning as the portfolio itself is already a linear combination of timeseries. One possible way of improving this experiment would thus be to consider the maximum pairwise correlation between all individual timeseries making up the portfolio and the individual timeseries of the randomly determined linear combination.

In any case, since we do not have a better way of inducing sparsity into the linear combinations produced by the considered problem solutions, we again used this strategy for obtaining sparse solutions from the linear programming solutions we proposed. Comparing their performance to CCA in terms of their ability to maximize correlation, we may conclude that they perform significantly worse than the best CCA model for random linear combinations of stocks of size k. Of course there is the caveat that CCA has the limiting assumption of needing $X, Y \subset S$ to be specified, which is why we had to compute all $\binom{k}{2}$ partitionings and run CCA for each of them. Of the two linear programming based solutions, the mixed-integer linear program fared best in terms of its ability to maximize correlation. However, its running time being exponential in n means it cannot really be used for larger datasets.

We suspect that the reason for the models not performing as well as CCA is partially due to the way their constraints are formulated. We say partially because the CCA procedure also has a fundamentally different optimization that it is solving than our linear programs. For the linear program, the reformulation that allows us to avoid the binary variables will also cause issues such as it assigning all timeseries to X'. Overall, requiring some constraint that states that the coefficients in the linear program should be positive is necessary to make sure that setting all

coefficients to 0 does not become a valid solution.

Chapter 7

Conclusions

7.1 Conclusions

We studied the problem of discovering higher-order correlations within the context of a set of timeseries S and developed a general problem definition (3.1) that defines the problem of higher-order correlation maximization. It covers many of the correlation measures C and aggregation functions f covered in our survey of existing research on the topic [2] [5] [3] [6]. Of these measures, we focused on one in particular: the Pearson correlation coefficient. The Pearson correlation coefficient can be generalized to operate on sets of timeseries $X, Y \subset S$ by using the property of bilinearity of covariance to compute the correlation coefficient between linear combinations of timeseries $X' = \sum_{i=1}^{l} a_i T_i, Y' = \sum_{j=1}^{m} a_j T_j$.

We developed several ways to solve the problem of higher-order correlation maximization for Pearson correlation. These have been integrated into an application CorrelationMiner that runs on top of Apache Spark. One way to solve the problem of higher-order correlation maximization is by exhaustively evaluating C for all possible subsets of timeseries $X \subset S$ or pairs of subsets $X, Y \subset S$. The problem is that the complexity of these exhaustive solutions are $O(2^n)$ and $O(3^n)$ respectively. In an attempt to scale the exhaustive solution, we have developed a distributed solver for the CorrelationMiner application that distributes the work of evaluating the subsets over a cluster of k worker nodes. We have shown that the solver manages to speed up the solution by a factor k. Due to the exponential complexity of the exhaustive solution, a factor k speed up quickly becomes insignificant as n grows. However, when used in combination with a suitable pruning constraint, the distributed solver strategies add further scalability to the solution. Moreover, it can be extended to support multiple implementations of correlation measures C and their corresponding aggregation functions f in accordance with the general definition proposed in problem 3.1.

In addition to this distributed solver, we developed 4 centralized solutions. The first uses canonical correlation analysis (CCA) and can be used to solve the problem variation where $X \subset S$ is known by specifying the second input set $Y = S \setminus X$. As regular CCA by default does not produce sparse linear combinations X', Y'. Recognizing that existing sparse CCA implementations would not be useable due to the presence of multicollinearity in our stocks dataset S, we implemented a strategy where we extract the top k coefficients of the linear combination Y' that CCA produces. As this causes some loss in terms of the correlation coefficient $\rho(X', Y')$, we re-run the CCA procedure on just the top k timeseries in Y' such that the CCA procedure will find a new linear combination Y^* for which the correlation with X' is again maximized. We used our running example dataset S consisting of stock timeseries to experimentally verify the approach by comparing $\rho(X', Y^*)$ against the correlation of X^* with a number of random linear combinations $Y'_1, .., Y'_5$ for 4 different linear combinations X^*. It turned out that some of the random linear combinations actually have a higher correlation with X^* than did Y^*. However, in all of these cases the maximum pairwise correlation of any of the timeseries in the random linear combinations

was higher than the maximum pairwise correlation of any of the timeseries in Y^*. Hence, we could reasonably conclude that these caused the random linear combinations to have a higher correlation than the initially determined Y^*. In general, we have to be careful with lending any credibility to higher-order correlations obtained from randomly forming linear combinations of stocks in S and correlating them using CCA as the correlations may likely be spurious.

While CCA can be used in case X is known, it cannot be used to solve the general maximization problem for which X, Y are both unknown. To this end, we have developed two linear programming solutions that aim to solve the general maximization problem. Of these two, one of the formulations turned out to require binary integer variables to express the constraint that $X \cap Y = \emptyset$. This is problematic, because integer programming is known to be NP-hard. As expected, a performance evaluation of the mixed-integer linear program shows that it scales exponentially in n, the number of timeseries that we put into the model. As an alternative, we reformulated the linear program into one that does not need binary variables. While the linear program without integer variables is much faster, it is also less accurate than the mixed-integer linear program. Due to the reformulation, it also suffers from a problem where it may assign all stocks to a single linear combination X' in case the number of input stocks is small enough. This is unavoidable due to the way we constrain the linear program.

The linear programming solutions, like regular CCA, do not produce sparse linear combinations. Hence, we again used the approach of picking the top k coefficients and re-running the models on just the top k timeseries. Re-running the linear program on just the top k timeseries did not produce meaningfully better solutions for the mixed-integer linear program than just keeping the default weight distribution. For the linear program, we often ran into the issue of it assigning all k coefficients to X'. While the mixed-integer linear program supports multiple solutions directly by saving the suboptimal solutions it finds, the regular linear program only finds a global optimum. We can make the linear program generate multiple solutions by re-running the model and excluding the timeseries that it reported as part of the linear combinations X', Y' that formed its global optimum.

Finally, we evaluated the ability of the linear programs to maximize the correlation between X' and Y' by running them on random sets consisting of k stocks. While the mixed-integer linear program performed better than the linaer program, both models obtained significantly lower correlation coefficients than the best CCA model. Overall, we conclude that the presented solutions achieve their goal of maximizing the Pearson correlation by finding the optimal linear combinations X', Y' reasonably well. For the CCA solution, we achieve the best performance in terms of the maximization of the correlation coefficient, but of course CCA assumes that the input sets X, Y are provided. In contrast, the linear programming solutions are able to find linear combinations X', Y' with just S as their input, but their ability to maximize the correlation coefficient is limited.

7.2 Future recommendations

In future work, we may consider the following possible research directions as well as improvements to be made to the CorrelationMiner application.

- Currently, the local solvers are implemented in Python. A relatively easy way to extend the functionality of the CorrelationMiner application would be to call the local solvers through Spark's pipe operator for RDDs.

- There is currently a limitation of $n = 64$ for the $ConstrainedPairwiseEnumeration$ strategy of the distributed solver **ExhaustiveSolver** implemented in the CorrelationMiner application. This because it uses a bit trick to loop over the lexicographically next bit permutations that represent subsets of the same cardinality. It should be relatively straightforward to change this implementation to use bit strings so that we are no longer bound by the 64 bit length of the long datatype that we currently use in that implementation.

- In [3], the authors make use of the Hamming's distance to show that the multi-information of a set X' is bounded by a function of the Hamming distance between a single newly added feature and the features in X. Currently, the *UnconstrainedEnumeration* strategy iterates over all $O(2^n)$ possible bit strings representing the subsets of S. It would be interesting to use a different order of enumerating the subsets.

  In particular, we may consider using Gray codes to enumerate the subsets in an order that mimics the transition from X to X'. If we could somehow combine this with the bounds on the multi-information and pruning constraints defined in [3], then we may be able to facilitate the pruning of subsets on the different workers. In that case, the scalability of the exhaustive solution would be greatly improved.

Bibliography

[1] "Apache Spark^{TM} - Unified Analytics Engine for Big Data." ii, 2, 9, 19

[2] S. Agrawal, G. Atluri, A. Karpatne, W. Haltom, S. Liess, S. Chatterjee, and V. Kumar, "Tripoles: A new class of relationships in time series data," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, vol. Part F1296, pp. 697–706, Association for Computing Machinery, aug 2017. 1, 2, 3, 14, 15, 18, 26, 32, 37, 39

[3] X. Zhang, F. Pan, W. Wang, and A. Nobel, "Mining Non-Redundant High Order Correlations in Binary Data," *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, vol. 1, no. 1, p. 1178, 2008. 1, 2, 15, 16, 18, 21, 24, 39, 41

[4] S. M. Bohte, H. Spekreijse, and P. R. Roelfsema, "The effects of pair-wise and higher-order correlations on the firing rate of a postsynaptic neuron," *Neural Computation*, vol. 12, no. 1, pp. 153–179, 2000. 1, 17

[5] S. Agrawal, M. Steinbach, D. Boley, S. Chatterjee, G. Atluri, A. The Dang, S. Liess, and V. Kumar, "Mining Novel Multivariate Relationships in Time Series Data Using Correlation Networks," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, may 2019. 1, 2, 3, 14, 16, 18, 21, 24, 39

[6] P. Mandros, M. Boley, and J. Vreeken, "Discovering Reliable Correlations in Categorical Data," in *2019 IEEE International Conference on Data Mining (ICDM)*, pp. 1252–1257, IEEE, nov 2019. 1, 2, 16, 18, 39

[7] G. J. Székely and M. L. Rizzo, "Brownian distance covariance," *Annals of Applied Statistics*, vol. 3, pp. 1236–1265, dec 2009. 6

[8] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Hoboken, NJ, USA: John Wiley & Sons, Inc., sep 2005. 7

[9] S. Watanabe, "Information Theoretical Analysis of Multivariate Correlation," *IBM Journal of Research and Development*, vol. 4, pp. 66–82, apr 2010. 7

[10] A. E. Hoerl and R. W. Kennard, "Ridge Regression: Biased Estimation for Nonorthogonal Problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970. 8

[11] S. Waaijenborg and A. H. Zwinderman, "Sparse canonical correlation analysis for identifying, connecting and completing gene-expression networks.," *BMC bioinformatics*, vol. 10, p. 315, sep 2009. 8, 9, 25

[12] D. Bertsimas, A. King, and R. Mazumder, "Best Subset Selection via a Modern Optimization Lens," jul 2015. 8

[13] H. Hotelling, "Relations Between Two Sets of Variates," *Biometrika*, vol. 28, p. 321, dec 1936. 8

[14] M. Borga and M. Borga, "Canonical correlation a tutorial," 1999. 9

[15] D. M. Witten, R. Tibshirani, and T. Hastie, "A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis," *Biostatistics*, vol. 10, pp. 515–534, jul 2009. 9, 25

[16] E. Parkhomenko, D. Tritchler, and J. Beyene, "Sparse Canonical Correlation Analysis with Application to Genomic Data Integration," *Statistical Applications in Genetics and Molecular Biology*, vol. 8, pp. 1–34, jan 2009. 9, 25

[17] "Gurobi Optimizer - Gurobi." 9

[18] "CPLEX Optimizer — IBM." 9

[19] M. J. Rosa, M. A. Mehta, E. M. Pich, C. Risterucci, F. Zelaya, A. A. T. S. Reinders, S. C. R. Williams, P. Dazzan, O. M. Doyle, and A. F. Marquand, "Estimating multivariate similarity between neuroimaging datasets with sparse canonical correlation analysis: an application to perfusion imaging.," *Frontiers in neuroscience*, vol. 9, p. 366, 2015. 12

[20] C.-s. Li, P. S. Yu, and V. Castelli, "HierarchyScan : A Hierarchical Algorithm for Similarity Search in Databases Consisting of Long Sequences," tech. rep., 1999. 13

[21] Y. Zhu and D. Shasha, "StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time," *VLDB '02: Proceedings of the 28th International Conference on Very Large Databases*, pp. 358–369, jan 2002. 13

[22] Y. Sakurai, "BRAID: Stream Mining through Group Lag Correlations," tech. rep., 2005. 13

[23] J. Friedman, T. Hastie, and R. Tibshirani, "Sparse inverse covariance estimation with the graphical lasso," *Biostatistics*, vol. 9, pp. 432–441, jul 2008. 17

[24] G. Lugosi, J. Truszkowski, V. Velona, and P. Zwiernik, "Structure learning in graphical models by covariance queries," jun 2019. 17

[25] H. V. Nguyen, E. M. Uller, J. Vreeken, P. Efros, and K. Böhm, "Multivariate Maximal Correlation Analysis," tech. rep. 17

[26] J. Trousdale, Y. Hu, E. Shea-Brown, and K. Josić, "A generative spike train model with time-structured higher order correlations," *Frontiers in Computational Neuroscience*, vol. 7, 2013. 17

[27] B. Staude, S. Grün, and S. Rotter, "Higher-Order Correlations and Cumulants," in *Analysis of Parallel Spike Trains*, pp. 253–280, Springer US, 2010. 17

[28] DataBricks, "databricks/spark-tfocs: A Spark port of TFOCS: Templates for First-Order Conic Solvers (cvxr.com/tfocs)." 21

[29] E. Mohyedin Kermani, "Distributed linear programming with Apache Spark," 2016. 21

[30] S. S. Skiena, *The algorithm design manual*. Springer, 2008. 21

[31] S. E. Anderson, "Bit Twiddling Hacks." 22

[32] "CCA: Perform sparse canonical correlation analysis using the... in PMA: Penalized Multivariate Analysis." 25

[33] B. M. Alves, A. Cargnelutti Filho, and C. Burin, "Multicollinearity in canonical correlation analysis in maize," *Genetics and Molecular Research*, vol. 16, mar 2017. 25

[34] Amazon, "Amazon EMR - Amazon Web Services." 33

Appendix A

AbstractAggregateTimeSeries interface

```
abstract class AbstractAggregateTimeSeries

case class AggregateTimeSeries(x: TimeSeries) extends
    AbstractAggregateTimeSeries
case class AggregateTimeSeriesPair(x: TimeSeries, y: TimeSeries) extends
    AbstractAggregateTimeSeries
```

Listing A.1: The AbstractAggregateTimeSeries interface

Appendix B

AbstractMultipleTimeSeries interface

```
import breeze.linalg.DenseVector

abstract class AbstractMultipleTimeSeries

case class TimeSeries(id: String, values: DenseVector[Double])
case class MultipleTimeSeries(x: Array[TimeSeries]) extends
    AbstractMultipleTimeSeries
case class MultipleTimeSeriesPair(x: Array[TimeSeries], y: Array[TimeSeries])
    extends AbstractMultipleTimeSeries
```

Listing B.1: The AbstractMultipleTimeSeries interface