A High-Dimensional Visualization System with Applications in Portfolio Management

Amy Tian

Advisor: Professor Han Liu

Submitted in partial fulfillment

of the requirements for the degree of

Bachelor of Science in Engineering

Department of Operations Research and Financial Engineering

Princeton University

June 2017

I hereby declare that I am the sole author of this thesis.

I authorize Princeton University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

_____

Amy Tian

I further authorize Princeton University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

_____

Amy Tian

Abstract

Often in modern multivariate analysis, data analysts rely solely on statistical estimators to explore the data. We are interested in using the notion of visual dependence to verify numerical tests of dependence (specifically, we focus on correlation metrics) and applying the results in portfolio selection, a setting that involves high-dimensional data sets. High-dimensional visualization is problematic because the number of pairwise plots to sort through increases quadratically as the number of variables increase. We present a visualization system that actively learns the user's concept of visual correlation in the data, applies the fitted classifier to unlabeled data to form the visual graph $\hat{G} = (V, E)$, and outputs the difference between $\hat{G}$ and some given numerical graph $\hat{G}^{\text{num}} = (V, E^{\text{num}})$. Specifically, we focus on the active learning and graph comparison components of the visualization system. We perform a simulation study with parameters that mimic the intended qualities of the system in order to select the best active learning method to use in the visualization system for the financial application. We compile various graph summarization metrics to compute the difference between two graphs (e.g. $\hat{G}$ and $\hat{G}^{\text{num}}$), and propose and verify a procedure for selecting $\hat{G}^*$, the numerical correlation graph most similar to some base graph (the visual graph $\hat{G}$ in this case), given the differences. Furthermore, we propose a simple but effective stock selection procedure that, given a correlation graph, selects a "buy and hold" portfolio of k stocks which are as uncorrelated with each other as possible, a proxy for independence. Numerical correlation graphs $\hat{G}^{i,\text{num}}$ are formed from stock price data with correlation metric $i \in I$ (the set of all correlation metrics), the data is fed into the visualization system to create $\hat{G}$, stocks are selected for portfolios, and yearly returns are compiled. The results indicate that the portfolio corresponding to $\hat{G}^*$, the numerical correlation graph which most resembles the visual correlation graph $\hat{G}$, is the top performer.

Acknowledgements

First, I would like to thank my advisor, Professor Han Liu, for his guidance and expertise.

I would like to thank Kevin for his patience, feedback, and for being a friendly Munchlax.

I would like to thank Tony for emergency kittens and delicious weekend noms.

I would like to thank my sister, Elizabeth, for being a magical friend and thesis fairy. I wish you the best of luck in your own thesis endeavors two years down the road. I would like to thank my younger brothers who are always fun to be around.

Finally, last but not least, I would like to thank my parents for their encouragement, support, and unconditional love.

To my family.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem statement

More than 2.5 quintillion bytes of data are produced daily as the field of data analysis continues to grow. "Statistical thinking and methodology" has become the framework for disciplines such as education, agriculture, economics, biology, medicine, astronomy, geology, and physics [6], but there is still a lack of accountability and consistency in the field. What is striking in the current practice of data analysis is the lack of progress on this particular subject beyond the development of numerical methods. The rapid increase in computing resources has led to the proliferation of high-dimensional data sets, which require much more work to efficiently understand patterns in the data and verify numerical estimators. In fact, the "physical limitations of display devices and our [human's] visual system prevent the direct display and instantaneous recognition of structures with higher dimensions than *two or three*" [13] (emphasis mine). One solution is to manually plot all variables against each other, but this becomes computationally tedious and unfeasible to sort through when there are even a few hundred variables. This problem gets even more complicated when considering *interaction terms* (various transformations or combinations

of explanatory variables). As such, the problem with current visual high-dimensional data analysis is that there are too many potential plots to sort through manually, which increase quadratically with the number of variables. Furthermore, although methods for dimension reduction have been developed [13], it is still unclear how the analyst can easily check the resulting model to ensure that the variables which were culled in the dimension reduction process are actually undesirable.

In computer science, a framework for "clean code" has been extensively documented and is the accepted industry standard for writing, interacting with, and thinking about code. But in empirical data analysis with large data sets, analysts blindly depend on estimators and hypothesis tests to explore the data and have no justification of their analysis aside from asymptotic, mathematical guarantees. Furthermore, since each estimator inherently performs well or poorly under different settings, data analysts are unable to differentiate between the properties intrinsic to the data set and the spurious properties the estimators added. Little, a Professor of Biostatics at the University of Michigan, notes that "developing good statistical solutions to real applied problems, based on good science rather than 'cookbookery,' is far from easy" [11]. This lack of agreement and "cookbook" mentality of data analysis has far-reaching consequences. It is simple to run the data through a list of many estimators and cherry-pick the most "interesting" result. Similarly, an analyst can remove "undesirable" data points without justification or unknowingly fit egregiously incorrect models. Regardless of whether these situations are performed maliciously or with good intentions, the art of data analysis is unclear without standards. The lack of clear-cut guidelines makes it difficult for analysts to discern the "truth" from the data and avoid the aforementioned pitfalls while simultaneously making it difficult for consumers of the resulting analysis to evaluate how trustworthy it is. These difficulties arise due to the difficulty in visualizing high-dimensional data. Plotting is one of the most consistent and universally interpretable "sanity checks" for numerical re-

sults; without it, effective and accountable analysis is difficult to achieve. Specifically, we are interested in the problem of high-dimensional visualization because notions of visual dependence are extremely useful in verifying numerical tests of dependence, allowing for improved analysis in fields such as finance.

Consider the following scenario with two different bivariate data sets. The problem is if x contains explanatory power of y. Common numerical analysis techniques yield the results summarized in Table 1.1.

Table 1.1: Simple numerical analysis in the bivariate case. The results suggest that the data are uncorrelated. For Dataset 1, refer to Appendix A.1. For Dataset 2, refer to Appendix A.2

Dependency test	**Dataset 1**	**Dataset 2**
Linear regression	$y = 0.461 + 0.008x$	$y = -0.131 - 0.2699x$
p-values	(2e-16) (0.911)	(0.488) (0.190)
Conclusion	Insignificant	Insignificant
ANOVA p-value	0.9109	0.1896
Conclusion	Insignificant	Insignificant
Shapiro p-value	0.5795	0.1632
Conclusion	Normally-distributed residuals	Normally-distributed residuals
Pearson's correlation	-0.1886	0.0113
p-values	0.1896	0.9109
Conclusion	Uncorrelated	Uncorrelated

Supposing that an analyst must rely on numerical tests alone, the reasonable conclusion to reach would be that x and y are uncorrelated. Given the power to plot quickly and efficiently, however, an analyst would quickly discover that the data exhibits a strong dependency (Figure 1.1). There are certainly many more ways to numerically analyze the data, and in retrospect, it can be argued that an analyst might have tried an estimator that captured the dependency properly. Even then, without the ability to plot, the other numerical results (which were strongly uncorrelated) cast doubt on the lone metric that might indicate correlation.

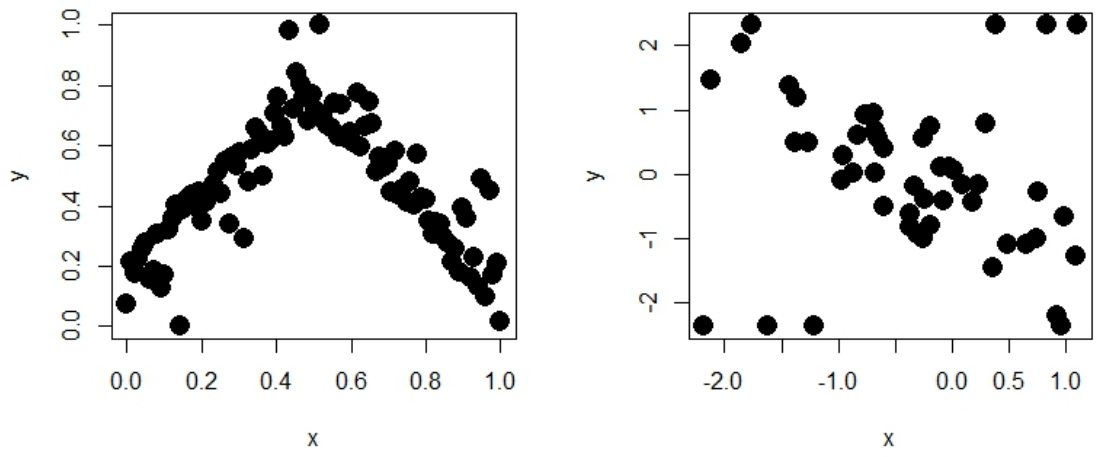


Figure 1.1: Simple visual analysis in the bivariate case. The data exhibits a strong visual dependency but fails common numerical tests of dependence (Table 1.1). *Left*: Dataset 1 (Appendix A.1), *Right*: Dataset 2 (Appendix A.2)

It is interesting to note that Dataset 2 (Figure 1.1, right) is clearly linear yet common tests of linear correlation (linear regression, Pearson's correlation) are not significantly different from zero (Table 1.1). Indeed, the data used in these examples was purposefully constructed to be dependent but bypass common tests for dependency. However, if it is possible to construct bivariate data sets in that evade common numerical methods, it is believable that it is even easier to construct analogous data sets in higher dimensions. Hence, no such standards of "clean analysis" currently exist in data science despite its importance in financial decisions, judicial evidence, government policy, and scientific discovery. Verification of numerical tests of dependence is especially important in finance as equity markets are large and involve billions of dollars. For instance, portfolio selection often involves determining the relationship among as many stocks as possible in order to thoroughly construct the best possible portfolio.

1.2 Correlation graphs

Independence graphs are one way to discover the dependency structure among different stocks whose prices may be represented as random variables following some distribution. The importance of independence among assets is explained later in Section 1.3.

Let $G = (V, E)$ be an undirected graph formed from the data set X with d observations of n variables. G has vertices $|V| = n$ and edges $E_{i,j} \in \{0, 1\}$. $E_{i,j} = 1$ when there is an edge between V_i and V_j, and 0 otherwise. Edges are determined with the following heuristic:

$E_{i,j} = 0$ for $i \neq j$ if and only if $X_i \perp X_j$ (the two variables are independent)

Unfortunately, independence graphs are difficult to construct in practice since independence is determined by the true joint distributions of all n variables in X, which is difficult to determine. However, the independence graph G may be estimated by the sample correlation graph $\hat{G} = (V, E)$. While it is true that non-correlation does not always imply independence, it is a useful estimate for *some* form of independence between two variables. It is important to be careful about the terminology usage. We use "correlation" to refer to *any form* of pair-wise dependence.

There are two main methods to estimate the dependence between two random variables; the first and most common is **numerical correlation**. By "numerical correlation", we do not mean the traditional mathematical interpretation of "Pearson's correlation" but, instead, anything which involves explicitly computing an estimate of the correlation e.g. Pearson's, Spearman's, and various other correlation metrics which are described later in this section. Let $\hat{G}^{\text{num}}$ be the numerical correlation graph of X. $\hat{G}^{\text{num}}$ can be drawn from a matrix P where $P_{i,j} = p(X_i, X_j)$ (the p-value of some sample correlation coefficient computed from the variables X_i and X_j) with

the following heuristic:

$E_{i,j} = 1$ for $i \neq j$ if $P_{i,j} \leq p$ where p is the p-value for the desired confidence level

---

Alternatively and more simplistically, the numerical correlation graph can be drawn from a correlation matrix $\hat{\Sigma}$ where $\hat{\Sigma}_{i,j} = \hat{\rho}(X_i, X_j)$ (some sample correlation coefficient computed from X_i and X_j) with the following heuristic:

$E_{i,j} = 1$ for $i \neq j$ if $|\hat{\Sigma}_{i,j}| \geq \lambda$ where $0 \leq \lambda \leq 1$ is the desired threshold value for a "significant" correlation (e.g. $\lambda = 0.75$)

We implement this latter approach when drawing correlation graphs for the financial application in Chapter 5.

The second method is **visual correlation**. By "visual correlation", we mean "pairs of variables that marginally appear dependent". Subsequently, the visual graph $\hat{G}^{\text{vis}}$ of X can be drawn with the following heuristic:

$E_{i,j} = 1$ for $i \neq j$ if the pairwise scatter plot of (X_i, X_j) contains a distinct trend

---

Visual correlation is, admittedly, subjective by nature and, as a consequence, slightly vague. Traditionally, numerical correlation is used over visual correlation due to its objectivity. However, visual correlation is still necessary in conjunction with numerical correlation. Consider if the joint distributions of the variables changed over time or if there were outliers as in the example of Section 1.1. While the dependency of the variables could not be captured by standard correlation metrics, we have a notion of the visual correlation between them. But because visual interpretations are subjective and dependent on how the data is plotted, numerical correlation is still useful. The two views on estimating the population correlation do not have to be

mutually exclusive and actually supplement each other's faults instead. To capture both qualities, a visual correlation graph may be utilized to check the suitability of various numerical correlation metrics that can be applied to the data. What follows is an overview of common numerical methods to estimate the correlation between two random variables. It will become clear that the numerical correlation metrics below are sensitive in some way or another, further emphasizing the need for both methodologies.

1.2.1 Pearson's correlation

Pearson's correlation measures the linear dependence among two random variables X and Y. In a population, the Pearson correlation is given by:

$$\rho_{X,Y} = \frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{\mathrm{E}(XY) - \mathrm{E}(X)\mathrm{E}(Y)}{\sqrt{\mathrm{E}(X^2) - \mathrm{E}(X)^2}\sqrt{\mathrm{E}(Y^2) - \mathrm{E}(Y)^2}}$$

As alluded earlier, the quantity above is typically unknown in practice since the expectation requires knowing the true joint distribution of X and Y. Instead, the population correlation is typically estimated by the sample correlation. Given n observations of X and Y, the sample expectation is given by the formula $\mathrm{E}(X) = \frac{1}{n}\sum_{i=1}^{n} x_i$. By substituting into the formulation above and multiplying by n^2/n^2, we can estimate the sample Pearson correlation with the following:

$$\hat{\rho}_{x,y}^{\mathrm{Pear}} = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\sqrt{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}\sqrt{n\sum_{i=1}^{n} y_i^2 - \left(\sum_{i=1}^{n} y_i\right)^2}}$$

such that $-1 \leq \hat{\rho}_{x,y}^{\mathrm{Pear}} \leq 1$. With perfect positive and negative linear dependence respectively, $\hat{\rho}_{x,y}^{\mathrm{Pear}} = \pm 1$. It is important to note that $\hat{\rho}_{x,y}^{\mathrm{Pear}} = 0$ does not necessarily indicate independence, though it is an indication of *linear* independence.

1.2.2 Spearman's correlation

Spearman's correlation is more broad than Pearson's; it measures monotonic dependence among two random variables X and Y. Monotonic functions are either strictly increasing or decreasing; while linear functions are monotonic, monotonic functions are not necessarily linear. Subsequently, Spearman's correlation may also capture non-linear dependencies. Spearman's correlation is computed by calculating the Pearson's correlation among "ranked variables". Each sample observation x_i of X is ranked from 1 to n based on its position relative to $x_j, j \in \{1, ..., n\} \backslash i$. The ranking is also computed for all observations of Y. Then the difference of a sample (x_i, y_i) is defined as $d_i = x_i - y_i$, and the sample Spearman's correlation is computed as:

$$\hat{\rho}_{x,y}^{\text{Spear}} = 1 - \frac{6 \sum_{i=1}^{n} d_i^2}{n(n^2 - 1)}$$

such that $-1 \leq \hat{\rho}_{x,y}^{\text{Spear}} \leq 1$. With perfect increasing and decreasing monotonic dependence respectively, $\hat{\rho}_{x,y}^{\text{Spear}} = \pm 1$. Again, it is important to note that $\hat{\rho}_{x,y}^{\text{Spear}} = 0$ does not necessarily indicate independence, though it is an indication of *monotonic* independence.

1.2.3 Kendall's tau

Similar to Spearman's correlation, Kendall's tau is another method of identifying monotonic dependence among two random X and Y as it also computes correlation among ranked variables. However, it does not utilize the difference among a single sample. Instead, it compares pairs of samples among each other. For $i \neq j$, (x_i, y_i) and (x_j, y_j) are defined as "concordant" if the ranks of both elements agree i.e. $x_i > x_j$ and $y_i > y_j$ or $x_i < x_j$ and $y_i < y_j$. Pairs are defined as "discordant" if the ranks of both elements disagree i.e. $x_i > x_j$ and $y_i < y_j$ or $x_i < x_j$ and $y_i > y_j$. In the case where ranks of either element are equal, the pair is ignored. Let c denote the number

of concordant pairs and d denote the number of discordant pairs. Then the sample Kendall's tau is computed as:

$$\hat{\rho}_{x,y}^{\text{Kend}} = \frac{c-d}{n(n-1)/2}$$

such that $-1 \leq \hat{\rho}_{x,y}^{\text{Kend}} \leq 1$. Kendall's tau is less sensitive to errors in the data as its correlation is based on pairs of samples rather than deviations within a single sample, though the resulting coefficients have the same interpretations. As with Spearman's correlation, $\hat{\rho}_{x,y}^{\text{Kend}} = \pm 1$ with perfect increasing and decreasing monotonic dependence respectively. Furthermore, $\hat{\rho}_{x,y}^{\text{Kend}} = 0$ does not necessarily indicate independence, though it is an indication of *monotonic* independence.

1.2.4 Distance correlation

While the aforementioned correlation metrics are well-known and commonly used, they are constrained to monotonic functions. Distance correlation was first proposed in 2007 as a way to further test for non-monotonic dependence between X and Y [19]. The distance correlation is a function of the distance covariance and distance variance of X and Y. The $n \times n$ matrices a and b denote the distance matrices of X and Y, respectively. The elements $a_{k,l}$ and $b_{k,l}$ are respectively defined as $||X_k - X_l||$ and $||Y_k - Y_l||$ for all $k, l = 1, 2, ..., n$ where $||z||$ is the Euclidean norm $\sqrt{z_1^2 + ... + z_n^2}$. Then the sample distance covariance and sample distance are given by:

- $\text{dCov}(X,Y) = \sqrt{\frac{1}{n^2}\sum_{k=1}^{n}\sum_{l=1}^{n} A_{k,l}B_{k,l}}$
- $\text{dVar}(X) = \text{dCov}(X,Y)$
- $\text{dVar}(Y) = \text{dCov}(Y,Y)$

where $A_{k,l} = a_{k,l} - \bar{a}_{k,} - \bar{a}_{,l} + \bar{a}$ and $\bar{a}_{k,}$ is the kth row mean of a, $\bar{a}_{,l}$ is the lth column mean of a, and $\bar{a}$ is grand mean of a. Similarly, $B_{k,l} = b_{k,l} - \bar{b}_{k,} - \bar{b}_{,l} + \bar{b}$ and $\bar{b}_{k,}$ is

the kth row mean of b, $\bar{b}_{l}$ is the lth column mean of b, and $\bar{b}$ is grand mean of b. The sample distance correlation is then computed as:

$$\hat{\rho}_{x,y}^{\text{Dist}} = \frac{\text{dCov}(X,Y)}{\sqrt{\text{dVar}(X)\text{dVar}(Y)}}$$

such that $0 \leq \hat{\rho}_{x,y}^{(\text{Dist})} \leq 1$. As before, $\hat{\rho}_{x,y}^{(\text{Dist})} = 1$ indicates perfect dependence. Like the other correlation metrics, $\hat{\rho}_{x,y}^{\text{Dist}} = 0$ implies *some* form of independence. However, the interpretation of $\hat{\rho}_{x,y}^{\text{Dist}} = 0$ is interesting in the sense that its corresponding population correlation metric $\rho_{x,y}^{\text{Dist}}$ has two important properties not seen in the other population metrics [19]:

1. X and Y may be of different dimensions.
2. $\rho_{x,y}^{\text{Dist}} = 0$ if and only if X and Y are independent.

1.3 Portfolio management

An important application of correlation graphs is in modeling the dependencies among financial equities. Determining the relationship among various stocks is especially useful when managing portfolios. One such methodology is the "buy and hold" tactic where an investor selects a portfolio of stocks and never rebalances. The idea is to select diversified stocks with low correlation and positive drift such that losses are offset by gains, and the portfolio gains on average. This strategy is especially useful when transaction costs are high as fees are prohibitive to rebalancing gains.

In a world with low transaction costs, however, there is more to be gained (less to be lost, alternatively) by frequently rebalancing the portfolio rather than holding. The concept for rebalancing is similar to that of "buy and hold", though there are some key differences. Another component for successful rebalancing gains is for the stock returns to be relatively independent [12]. Thus, when one stock goes down,

the others do not fall with it; with perfect independence, rebalancing gains become a function of the volatility of the stocks rather than of stock returns. Independence is further important because frequent asset trading may move the market and lead to unexpected price swings. With independent stocks, the price risk associated with rebalancing is minimized. In summary, rebalancing gains are best suited to markets with low correlation, independent returns, and high volatility [12]. For simplicity, we focus on the former problem of porfolio selection where we buy a portfolio and hold for the remainder of time. A detailed application may be found in Chapter 5 where a stock selection methodology for correlation graphs is also proposed.

Again, we acknowledge that while non-correlation does not necessitate independence, it is still a useful proxy for independence. It must further be noted that numerical correlation alone cannot capture the full complexity of these financial applications; in fact, cases such as that of Figure 1.1 (right) reflect the limitations of correlation coefficients and reinforce the importance of visualization in aiding with the selection of numerical metrics. An unassuming analyst might select the stocks (variables) from Table 1.1, Dataset 2, because their numerical correlation is not significantly different from zero. But, it turns out that the stocks are highly correlated, and this is only clear when looking for visual correlation. However, numerical correlation is still an important component of portfolio management in theory and in practice. As such, an important goal of this paper is to not only develop a solution to the problem of high-dimensional visualization described in Section 1.1 but to also consider the reason we are interested in visualization in the first place. As noted earlier, plotting is an incredibly useful tool to check numerical methods. Thus, we will also develop a method to quickly check the output from numerical methods with an analyst's own concepts of visual dependence, allowing the analyst to select the numerical method most suited to their application (numerical correlation graphs and portfolio selection, in this case). What follows is a roadmap of the high-dimensional

visualization solution that is developed further in this paper with Chapters 2, 3, and 4.

1.4 Summary

In this work, we tackle the problem of sorting through $n(n-1)/2$ pairwise scatter plots by developing a sophisticated visualization system (abbreviated VS) to intelligently and automatically explore the data. Specifically, we focus on two aspects of the VS: (1) efficiently automating the procedure of sorting through plots by learning the user's interests, and (2) the procedure of comparing the numerical and visual output (in the form of correlation graphs). This allows the system to find visually interesting relationships that the numerical model may have missed and/or toss out relationships which turn out to be uninteresting. Furthermore, this allows future analysts to combine visual feedback with the numerical feedback from estimators to make better decisions during data analysis and provide clear justification of their decisions.

A broad overview of the VS and its framework may be found in Chapter 2. Chapter 3 focuses on the active learning stage of the VS, which answers the problem of high-dimensional visualization raised in Section 1.1. The chapter details and simulates various active learning methods to be used in the financial application in Chapter 5. Chapter 4 focuses on the application of the visual output in relation to the financial application of Chapter 5. Specifically, Chapter 4 is concerned with the VS output in the context of graphs, and it discusses methods to quantify the differences between numerical and visual graphs. Furthermore, the chapter proposes a procedure to find the numerical graph most similar to (least different from) the visual graph. Though our application is concerned with correlation graphs, the methods discussed in Chapter 4 may apply to any other sort of graph (e.g. a graphical model). Again, Chapter 5

contains the financial application where the visual correlation graph output from the VS is used to aid in the selection of the most suitable numerical correlation graph for portfolio selection with our particular data set. The chapter further proposes a simple but effective portfolio selection procedure given the numerical correlation graph. Finally, Chapter 6 recaps the work and presents further extensions.

Chapter 2

Visualization System

Regardless of whether high dimensional data visualization methods are computationally heavy or interaction heavy, user interactivity is a critical component of the analysis; it is simply a question of what degree [13]. It is not enough to simply have user interaction, however. Given n variables, there are a total of $\binom{n}{2} = n(n-1)/2$ possible scatter plots of the data. This blows up quadratically as n increases, which is infeasible for the analyst to sort through in any reasonable amount of time. Subsequently, automation is another necessary element in the task of visualizing high dimensional datasets.

We develop a system that first learns what visual patterns the data analyst finds promising, querying the user where the decision boundary is ambiguous. It then automatically iterates through all possible pairwise scatter plots and returns an adjacency matrix that captures the classification of each pair. Classifications are binary where the label 1 indicates "visually correlated" and the label 0 indicates "not visually correlated". The VS may then return the top "visually correlated" scatter plots for the user, perform line-up tests to refine the classifier (see Section 6.2.5), and/or compare the visual graph with some numerical graphs (or a single numerical graph) of the user's choice. This allows analysts to compare and contrast visual feedback with nu-

merical algorithms for improved model selection. Figure 2.1 is a visual summarization of the system.

Figure 2.1: Broad overview of the visualization system.

Section 2.1 describes characteristics of an interesting scatter plot; incorporating these ideas in the VS facilitates user accuracy in stage 1. Sections 2.2 and 2.3 provide a brief overview of stage 1 and 2, respectively, while Section 2.4 describes the focus of the rest of the work which respond to the problems and application posed in Chapter 1.

2.1 Scatterplot characterization

2.1.1 Characteristics of a "good" plot

The simplest scatter plot is one without any transformations on the data. This, however, may not be the best way to ascertain independence for the user. This notion is illustrated in Figure 2.2. The left plot appears to be independent as it's a cluster of points near the origin, but it's not entirely clear due to the multitude of stray points outside of $y \in (-2, 2)$ and $x \in (-1.5, 1.5)$. By looking at the outliers, it could also be argued that there is some dependency. However, applying the CDF in both directions creates a plot distributed on (0,1). This transformation is non-destructive and preserves dependency in the data if it exists. The data is clearly independent as the points appear to be uniformly distributed within the box.

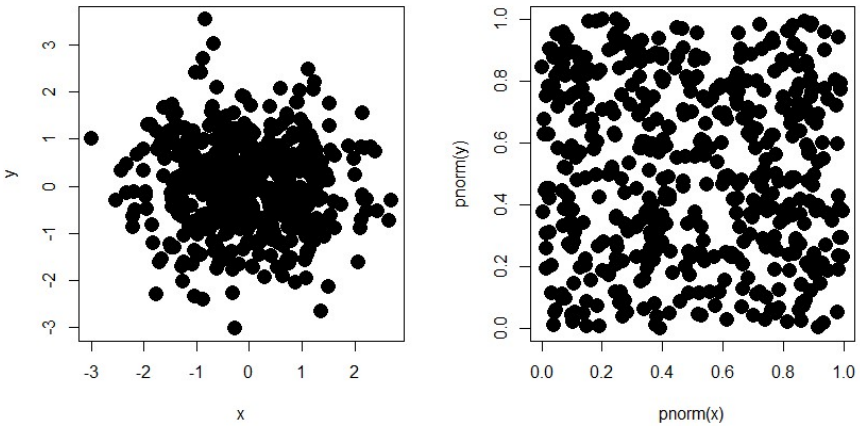


Figure 2.2: A plot of y against x with no transformation (left) and after the CDF is applied in both directions (right). The code for this example may be found in Appendix A.3

Restricting the scatter plot to a unit box allows analyst's visual systems to focus on locations where there is low spatial frequency, which is ideal for detecting dependence [8]. The effects of this concept can be progressively observed from left to right in Figure 2.3 below.

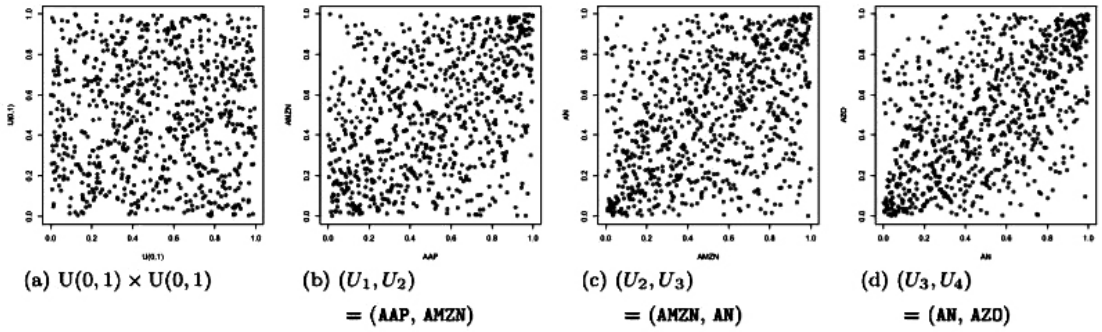


Figure 2.3: *(a):* Transformed scatter plots of independent $U(0,1)$ random variables. *(b,c,d):* Transformed scatter plots of the pseudo-observation pairs $(U_j, U_{j+1}), j \in \{1,2,3\}$. The variables are clearly more correlated as j increases; this trend can easily be observed due to the nature of the unit box. Ticker abbreviations: AAP = Advanced Auto Parts (U_1), AMZN = Amazon.com Inc. (U_2), AN = AutoNation Inc. (U_3), AZO = AutoZone Inc. (U_4). Figure from Hofert and Oldford 2016 with slight modifications [8]

2.1.2 Feature extraction from plot

In order for the active learning classifier to properly understand and classify all $\binom{n}{2}$ scatter plots in stage 1 and 2 (Sections 2.2 and 2.3), characteristic features must be extracted from each pairwise scatter plot. The more useful criteria there are, the more sophisticated the classification will be.

Numerical features

Our goal is to quantify various features of a scatter plot for the computer, and that does include numerical features. The following features are currently implemented in the VS:

- **Correlation coefficients and their p-values:** Examples include Pearson's, Spearman's, Kendall's, and distance correlation. See Section 1.2 for more details on the various types of coefficients.
- **Kullback-Leibler divergence criterion:**
- **Chi-square test of independence and its p-value:**

Visual features

What is more challenging is to find a way to quantify the visual features of scatter plots. This may be done by looking for concentration of points in various spaces of the plot domain. The following features are currently implemented in the VS:

- **Middle box criterion:** The percentage of points near the center of the plot
- **LR criterion:** The percentage of points that lie above and below the linear regression line
- **Clustering criterion:** The percentage quantile of the ratio between the largest and next-largest distance
- **Visual trend criterion:** This is computed as max(PosTrendCriterion, NegTrendCriterion). The positive trend criterion (PosTrendCriterion) is computed from the percentage of points in the bottom left and upper right while the negative trend criterion (NegTrendCriterion) is computed from the percentage

of points in the upper left and bottom right. A higher visual trend criterion value suggests a greater visual trend.

2.2 Active learning (stage 1)

The main goal of stage 1 is to learn the user's interests. This requires the system to select ("query") data for the analyst (the "oracle") to label ("classify"). The VS data for the active learning component is composed of all observations of characteristic features (as described in Section 2.1.2)for all $\binom{n}{2}$ pairwise scatter plots of the *actual* data set. This is **stage 1**. The learner may then utilize a classification model (discriminant analysis, naive Bayes, decision tree(s), logistic regression, etc.) that trains on the labeled data to "learn" user interests. The user's interests are encoded in a classifier (some instance of the classification model) that is applied to automatically label the rest of the data. Here again, it is important to be careful about the terminology usage. For more on the semantic differences between "classification model" and "classifier" in this body of work, see Figure 2.4. This is **stage 2** (Section 2.3). It is important to make the stage 1 process as efficient as possible to avoid redundancy for the end user. There are various methods that may be used for querying in stage 1 as described by Dasgupta [5]:

- **Supervised learner**: This learner queries a random subset of all unlabeled data. It ignores the rest of the data when refining the classifier.
- **Semisupervised learner**: Similar to a supervised learner, a semisupervised learner queries a single, random subset of all unlabeled data but proceeds to utilize the remaining unlabeled data to better inform the final classifier.
- **Active learner**: An active learner selects its queries in a non-random, intelligent manner.

It has been shown that when a learning algorithm is allowed to choose its next query, it performs better with less training; as such, we choose to utilize active learning to select the plots to be queried by the oracle in stage 1 [18]. This section is primarily

focused on the active learner's role in the overall system; Chapter 3 goes into detail on different active learning methodologies.

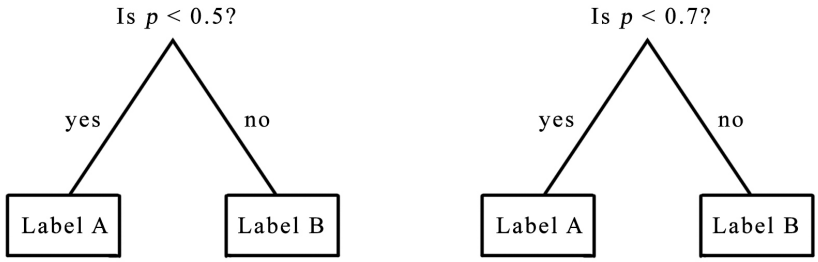


Figure 2.4: Although both figures on the left and right are slightly different classifiers, they are both instances of (extremely simple) decision trees, a type of classification model. Other models include discriminant analysis, naive Bayes, random forest, logistic regression, etc. See Section 2.3.1 for more details on trees.

2.2.1 Initialization of active learner

It is problematic to start from scratch; how does the system determine the best first point of ambiguity when it knows nothing (the hypothesis space is everything)? A classic method is to simply select k random data instances for the user to label. As initialization is not the focus of this work, the VS currently utilizes this methodology. To compensate for a potentially poor initialization, we allow the stage 1 algorithm to have a larger budget of user queries in the application of Chapter 5.

Alternatively, suppose the user has a numerical graph that they believe to be a good representation of the data, and they would like the visualization system to provide a visual check (as opposed to selecting from multiple numerical graphs as in our application in this work). An edge drawn in the numerical graph may be considered a starting point for an edge drawn in the visual graph; the system may exploit this idea and initialize stage 1 by fitting a classifier that agrees with the given numerical graph as much as possible. Doing so greatly narrows the hypothesis space and makes it easier to determine points of ambiguity. However, to reconcile with

the fact that the user wishes to check the numerical model and may not necessarily believe it is a good representation of fit, the learner should first check whether the initial classifier is a proper fit; this may be achieved with line-up tests, which are described in Section 6.2.5. As the user proceeds to label various pairwise scatter plots (which are queried by the active learner) as "visually correlated" or not, the learner better understands the users interest, and the main classifier continues to evolve and improve.

2.2.2 Query selection

Post-initialization, the active learner cleverly queries vital plots so that the system can best learn the user's interests. The system first determines which features it is uncertain about classifying and then returns a plot matching those characteristics to the user. The "oracle" responses allow the system to utilize its classification model of choice to build a better classifier in stage 2. **It is important to distinguish between the active learner, which selects the next queries from the pool of unlabeled data and *may use its own classification model(s) to aid in query selection*, and the VS, which uses a *single classification model* to fit both the initialized and actively-selected queries in order to build a classifier to label the remaining plots in stage 2 (Section 2.3)**. Various active learning (selection) algorithms include uncertainty sampling, query by committee, query by bagging, and min-max clustering, which are all described more thoroughly in Chapter 3.

2.3 Automated plot generation (Stage 2)

As a reminder, in stage 2, the visualization system uses the labeled scatter plots to build a classifier and automatically label the rest of the plots.

2.3.1 Decision tree classification of user interests

Given labels for the initialized and actively-selected queries, there are several classification models that the visualization system can use to create a final fitted model of user interests. One such model is a *decision tree*. A decision tree is composed of nodes (which correspond to classification labels) and branches (which correspond to decision boundaries). A tree is constructed at each node by sampling all M possible vertical and horizontal splits in the sample space and selecting the split which minimizes the *Gini criterion* [3]. A mapping of the sample space to a tree is shown in Figure 2.5. The Gini criterion measures the homogeneity of the nodes in each side of the proposed split and is computed as follows (for a vertical split) [3]:

$$G = N_{\text{left}} \sum_{k \in \{0,1\}} p_{k,\text{left}}(1 - p_{k,\text{left}}) + N_{\text{right}} \sum_{k \in \{0,1\}} p_{k,\text{right}}(1 - p_{k,\text{right}})$$

where N_s is the number of nodes in side s of a split ($s = \{\text{left,right}\}$ in a vertical split, and $s = \{\text{top,bottom}\}$ in a horizontal split) and $p_{k,s}$ is the fraction of class label k on side s of the split. Note that $k \in \{0, 1\}$ in binary classification, which is used in the VS. Decision trees are a more sophisticated classification method than simple linear regression and retain interpretability. The root node is, naturally, the most important as it corresponds to a split that optimizes the Gini criterion while the terminal nodes can be thought of as the data's homogeneous clusters. However, decision trees are also unstable; perturbing a single data point may change the entire tree. Furthermore, due to the nature of vertical and horizontal splitting, a single tree cannot fully capture diagonal (or non-vertical/horizontal) splits in the decision boundary.

A *random forest* provides solutions to the problems that a single decision tree faces. A forest is composed of many decision trees "grown" from random partitions of the labeled set (the training set). Furthermore, each decision tree is constructed

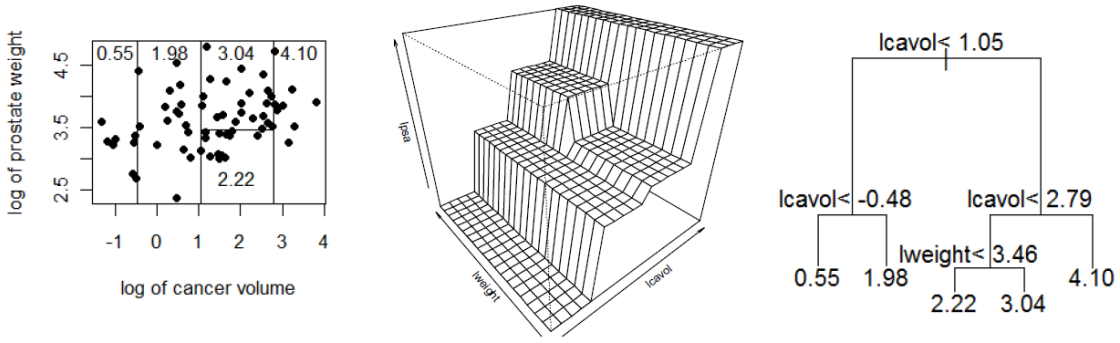


Figure 2.5: Mapping the sample space (*left*) to a decision tree (*right*). Images from Cutler 2010 [3].

by finding the best split among $m \in M$ splits [3]. This allows each tree to specialize on a subset of the data, creating a more informative aggregate. A simplistic example of a forest with 2 trees may be found in Figure 2.4. Each tree in the forest has a vote of weight one on the label of each unlabeled data point, and the forest is aggregated by majority vote, which makes the resulting decision boundaries more stable (less variant) on average. On the other hand, a forest with many trees is difficult to visualize. However, there are methods to simplify a forest into a single tree for the purposes of visualization (rather than for usage in classification, as the issue with stability would then remain). One such method for single tree approximate is presented by Zhou and Hooker [23]. Utilization of such a method to present the resulting random forest classifier to the user aids in user interpretability of the system output while maintaining the robustness of a forest in practice. As such, the VS sets the choice of classification model to random forest by default but allows the user to specify their preferred classification model if they wish to do so.

2.3.2 User interaction with active learning output

The system has now learned which of the unlabeled plots may be of interest to the user. The final learned classifier is used to fit the rest of the $\binom{n}{2}$ unlabeled scatterplots,

and a visual graph $\hat{G} = (V, E)$ may be built from these labels with the heuristic presented in Section 1.2:

$E_{i,j} = 1$ (draw an edge) for $i \neq j$ if the pairwise scatter plot (X_i, X_j) contains a distinct trend

---

Furthermore, the VS may provide a visualization of the resulting classifier boundaries such as the decision tree itself (as discussed earlier). The active learning output may also be visualized as a heat map. A classic heat map represents the qualities of each pair of variables as colors from a bivariate spectrum. Heat maps are difficult to interpret as they are one-dimensional; each end of the color spectrum represents minimum and maximum values respectively. It is unclear what the maximum or minimum values are because they depend on the domain of the whatever criterion the heat map is plotting; as such, the minimum may not necessarily be negative while the maximum may not necessarily be positive. Furthermore, the subtle variations in hue between colors make it difficult to compare the relative ranking of different variable pairs with similar colors. Buja *et al.* propose an alternate, clearer method of visualizing the heat map, termed the "association navigator" [2]. The association navigator is two-dimensional; the size of each field corresponds to the criterion's value while the color itself indicates if the criterion is positive or negative [2]. As such, the association navigator only utilizes two colors rather than a spectrum of colors, making it simpler to distinguish between the two. By simplifying the color scheme and adding the dimension of size, the association navigator makes it easier to interpret and compare different pairs of data at a glance. The stark difference between a traditional heat map and the association navigator (applied to the same dataset) may be seen in Figure 2.6. The VS accommodates for both options, allowing the analyst method is preferred. With these visualizations of the active learning output, the user may better understand his/her own interests.

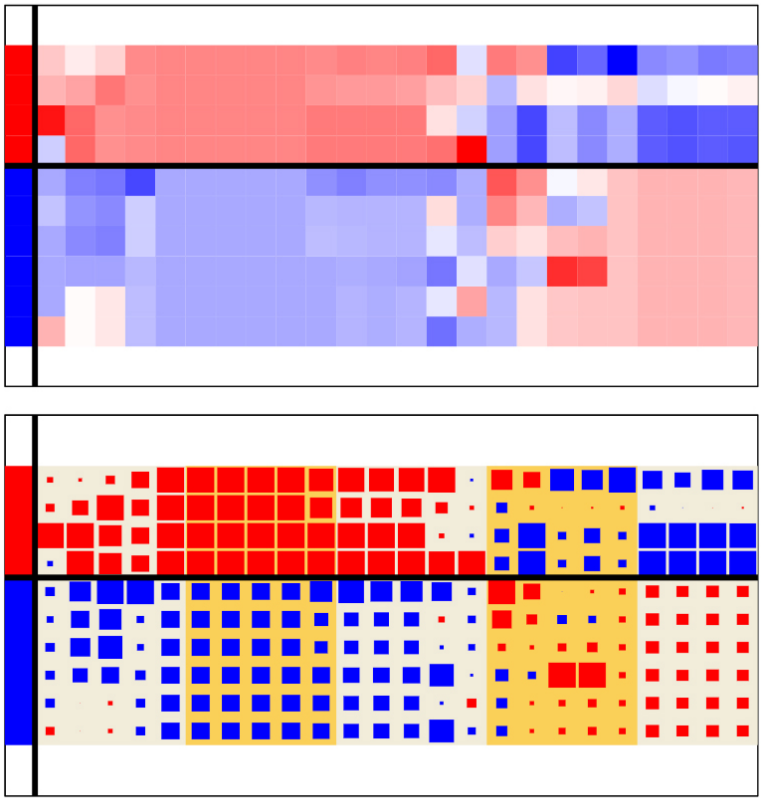


Figure 2.6: *Top:* A traditional heat map. *Bottom:* The association navigator on the same set of data. Each row corresponds to a labeled pairwise scatter plot, and each column corresponds to a different plot characteristic.

2.3.3 System output

There are three options at the end of stage 2. Two of them are concrete outputs that may be easily used in a final report on the data analysis findings, and the third is a refinement of the active learning component in stage 1.

- **Automatic plot generation:** The VS compiles a selection of the most interesting and non-interesting plots along with their associated transformation variables.
- **Graph comparison:** The VS accepts a numerical graph such as $\hat{G}^{\text{num}}$, a correlation graph generated from one of the numerical correlation coefficients described in Section 1.2, and the VS measures the difference between $\hat{G}^{\text{num}}$ and the visual graph $\hat{G}$ (the active learning output). This is especially useful for determining which numerical graph is "closest" to the visual graph, allowing the

user to select a more appropriate numerical method for their analysis. Details on graph comparison methods and a similarity selection strategy can be found in Chapter 4.

- **Line-up test:** In the event that the classifier is not a satisfactory representation of the analyst's interests, the VS may utilize line-up tests to help determine where to query from further. Once the stage 1 component is refined and the classifier has been refit, the concrete outputs mentioned above may then be generated if desired. For more details on this methodology, see Section 6.2.5.

Because this work focuses on numerical and visual correlation graphs and their financial application, graph comparison is the most useful output from the visualization system. As such, it is one of our primary VS focuses. The next section provides a roadmap for the rest of this work, which goes into detail on two aspects of the VS.

2.4 Specific focus: active learning and graph comparison

The active learning component is the bread and butter of the visualization system and forms stage 1 of the system. It solves for the tedious nature of classifying $\binom{n}{2} = n(n-1)/2$ pairwise scatter plots in high-dimensional visualization. The next issue to consider is the usage of the visual results. This is a question of the VS output, and graph comparison is a solution to this problem that is both useful in selecting the numerical method most similar to the visual graph (which we are interested in conjunction with numerical correlation graphs for portfolio selection) and informative to the analyst. Furthermore, in instances where the analyst is using the VS to check their numerical models, large differences encourage the user to investigate whether their selected method was truly appropriate for the dataset at hand. Subsequently, we primarily focus on active learning methods in Chapter 3 and graph comparison methods in Chapter 4. These methods are added to the current iteration of the VS, which is used (with healthcare stock data) in conjugation with numerical correlation

graphs to perform stock selection in Chapter 5. The current iteration of the VS is under development in the `graphicalModelEDA` package by `linnylin92` [10].

It should be noted that the VS is a large project which is still under construction, and there is much room for refinement beyond the work that we do in this paper, which develops two of its major components. Future extensions of the VS may be found in Section 6.2; there are several ideas on improving various aspects of the system which are not the focus on this work.

Chapter 3

Active learning

Active learning is a subset of machine learning wherein an algorithm is "trained" on labeled data instances in order to learn from and make predictions on the unlabeled data. Consider a large set of unlabeled data X, each with a hidden label from a finite set Y that can be queried from some human "oracle"; we would like to learn a good classifier of the data, some mapping $h : X \to Y$ from the set $\mathcal{H}$ without making too many queries [5]. Active learning is the process of intelligently selecting the queries (a constrained resource) to learn as much as possible. When a learning algorithm is allowed to select its next query, it performs better with less training [18]. To put it more concretely, the error (a measurement of the difference between the predicted labels and the "true" labels) converges to zero faster. This property is especially desirable when labeled data are difficult, time-consuming, or computationally expensive to obtain. Stage 1 of the VS (Section 2.2) is one such classification task where $Y \in \{\text{visual correlation, no visual correlation}\}$. Classification tasks are both tedious and redundant, which necessitates intelligent selection of queries [18]. What follows is an active learning literature review in Section 3.1, an overview of active learning methods and their algorithms in Section 3.2, and a simulation study in Section 3.3

to determine the method that is best-suited for usage with the VS in Chapter 5's financial application.

3.1 Literature review

It is important to consider the main framework in which the learning algorithm selects a query from unlabeled data. There are three different scenarios in which the learner may request queries as described by Settles [18]:

1. **Membership query synthesis**: The learner may select a query from any unlabeled samples in X.
2. **Stream-based selective sampling**: An unlabeled sample is randomly selected from X, and the learner decides whether to query or not.
3. **Pool-based sampling**: k unlabeled samples are randomly selected from X, and the learner picks one to query.

While the methods above may apply to many different active learning environments, the specific situation for the VS is *pool-based selective sampling*. Because it is computationally expensive to extract features from every single $\binom{n}{2}$ plot (see Section 2.1.2), membership query synthesis is not an option. While stream-based selective sampling may work, it has a similar problem because there are no constraints on the number of samples that the algorithm may choose to discard.

The next problem, then, is to determine the informativeness of unlabeled instances that are presented by the situations above. This is the crux of active learning as it allows for intelligent selection of queries. Dasgupta identifies two different approaches to active learning that are fundamental to the process of query selection [5]:

1. **Efficient search through hypothesis space $\mathcal{H}$**: The idea is to select a query that shrinks $\mathcal{H}_t$, the set of all possible classifiers at time t that explain the labeled data, as much as possible.
2. **Exploiting cluster structure in data**: The idea is to cluster the data and select queries based on cluster structure (i.e. query from each cluster). While

clusters may be split over time if they are discovered to be non-homogenous, a learner may leverage situations where clusters are fairly homogeneous in order to classify points by propagating a node's label to its neighbors.

Uncertainty sampling, query by committee, and query by bagging are active learning algorithms that are a form of the aforementioned efficient search through the hypothesis space. These algorithms may be found in Sections 3.2.1 to 3.2.3 respectively. We also present a clustering partitioning algorithm in Section 3.2.4 that seeks to exploit the clustering structure in the data.

3.2 Overview of active learning methods

In this section, we present algorithms for specific active learning methods and a reference to its implementation in the appendix. The `activelearning` package by `ramhiser` [16] has adapted much of the methods reviewed in Settles' work [18]. As such, most of the implementation code has been adapted and reworked (with substantial components written from scratch) from the `activelearning` package, which is too general for our purposes.

3.2.1 Uncertainty sampling

In uncertainty sampling, the active learner selects the query q that it is most uncertain on how to label; in other words, the algorithm queries the label that has the highest posterior probability [9]. With binary classification as is the case with the VS (either "visually correlated" or "not visually correlated"), this reduces to the case of querying the instance whose posterior probability of being "visually correlated" is closest to 0.5 [9]. While the uncertainty sampling algorithm presented in Algorithm 1 follows this methodology and may subsequently only be used with classification models that are able to encompass posterior probability computations, there has been much work

done to expand uncertainty sampling to non-probabilistic classifiers such as decision trees and nearest-neighbor [18].

Uncertainty sampling is simple but not without problem. Given a multi-label classification problem (labels $k > 2$), uncertainty sampling only considers the information about the most probable label i and ignores the other possible labels $j \in \{1, ..., k\} \backslash i$. "Margin sampling" and "entropy" are variants that try to solve for these problems, but both reduce to the scheme of querying from the sample with a posterior probability closest to 0.5 when $k = 2$ (binary classification) [18].

We have developed an algorithm for uncertainty sampling based on the literature review (see Appendix A.4 for code):

Algorithm 1 Uncertainty sampling (as described by Settles [18])

1: **procedure** (X is a $n \times d$ matrix of d observations of all n variables, y is an n-length vector of labels for each variable in X (y_i=N/A when X_i, has no label))
2: $tout \leftarrow \text{train}(X^{\text{labeled}}, y^{\text{labeled}}, \text{classification model})$
3: $p \leftarrow \text{predict}(tout, X^{\text{unlabeled}}, \text{posterior prob.} = \text{TRUE})$
4: **loop from** $i = 1$ **to** $\text{len}(p)$:
5: $p_i \leftarrow |p_i - 0.5|$
6: **return** $\text{where}(p == \min(p))$

By searching for the most "uncertain" point, uncertainty sampling is able to further refine the classifier as the oracle must label the point either "visually correlated" or "not visually correlated". This can be viewed as a search within the hypothesis space $\mathcal{H}$ that contains multiple classifiers, instances of a single classification model.

3.2.2 Query by committee

Query by committee is a clearer case of efficient search through the hypothesis space. In query by committee, a "committee" of classification models are trained on the current labeled instances. Each model represents a competing hypothesis, and its prediction for an unlabeled query candidate q is a "vote" of weight one on q; the

most disagreeable candidate is then queried [18]. Settles reviews various methods for initial committee selection by the active learner (for both probabilistic and non-probabilistic models), though our implementation of QBC (Appendix A.5) allows the user to specify the committee members [18].

The basic algorithm is as follows:

Algorithm 2 Query by committee (as described by Settles [18])

1: **procedure** (X is a $n \times d$ matrix of d observations of all n variables, y is an n-length vector of labels for each variable in X (y_i=N/A when $X_{i,}$ has no label). Let C be the vector of all committee members i.e. classification models)
2: **loop from** $i = 1$ **to** len(C):
3: $tout_i \leftarrow \text{train}(X^{\text{labeled}}, y^{\text{labeled}}, C_i)$
4: $p_{i,} \leftarrow \text{predict}(tout_i, X^{\text{unlabeled}})$
5: $d \leftarrow \text{disagreement}(p)$
6: **return** where($d == \max(d)$)

While it is simpler to maintain the same committee throughout, it would be more informative to prune the committee as the algorithm proceeds; it may simply be the case that a model is ill-suited for the problem at hand and consistently returns predictions that skew the voting procedure. Subsequently, a model is removed from the committee if its predictions are consistently out-of-line with whatever the "true" label of q_t (the next query point decided and then queried at time t) turns out to be. It is important to note that the "true" label is not known until *after* the algorithm has returned the index of q_t. In the VS, this index corresponds to the next pair-wise scatter plot to show the user and query. Subsequently, the committee from t is pruned at time $t+1$ at the start of the algorithm using the label retrieved from time t. The pruning function should only be run after a good number of iterations (where each iteration results in a query) have passed to allow the error ratios to converge (Otherwise, there is no room for learning). In our implementation, we begin using the pruning algorithm after $iter/2$ queries where $iter$ is the query budget (the maximum number of queries allowed). Furthermore, the query by committee methodology

described by Settles only uses the committee to select q; thus, the committee is independent of the final classification model (in the VS, this is a random forest) once the budget has been used up [18]. However, there may be merit in maintaining the final pruned committee as its own classification model after the budget has been used up. As a classification model, the pruned committee may label unlabeled instances via majority vote. Further details on implementation and a performance comparison may be found in the simulation study (Section 3.3). The revised algorithm is as follows (see Appendix A.5 for code):

Algorithm 3 Query by committee (revised framework)

procedure (X is a $n \times d$ matrix of d observations of all n variables, y is an n-length vector of labels for each variable in X (y_i=N/A when $X_{i,}$ has no label). Let C be the vector of all committee members i.e. classification models, E be the average error ratio of the respective committee members (initialized to 0), $0 < \epsilon < 1$ be some threshold for the error ratio, *iter* be the total active learning budget, and t be the current iteration of QBC starting at $t = 1$)
    function QBC
        loop from $i = 1$ **to** len(C):
            $tout_i \leftarrow \text{train}(X^{\text{labeled}}, y^{\text{labeled}}, C_i)$
            $p_{i,} \leftarrow \text{predict}(tout_i, X^{\text{unlabeled}})$
        $d \leftarrow \text{disagreement}(p)$
        return $j = \text{where}(d == \max(d))$
    function ORACLE
        return l, the label of $X_{j,}$
    $y_j \leftarrow l$
    function PRUNE (When iteration $i \in [1, iter] > iter/2$, run PRUNE)
        $prune = [\ ]$ (empty vector)
        loop from $i = 1$ **to** len(C):
            If $(p_{i,j} == y_j)$ **then** $iv = 0$ **Else** $iv = 1$
            $E_i = E_i + \frac{iv - E_i}{t}$
            If $E_i > \epsilon$ **then** $prune$.append(i)
        $t++$
        return $prune$
    loop from $i = 1$ **to** len($prune$):
        Delete E_{prune_i}, C_{prune_i}, $p_{prune_i,}$, and $tout_{prune_i}$

Note that both Algorithm 2 and 3 contain a generic disagreement function, which measures the disagreement among the committee members. Let Y be the set of all possible labels. There are two main methods of measuring disagreement described by Settles [18]:

1. **Vote entropy**: The disagreement for query candidate q is computed

$$d_{\mathrm{VE}}(q) = -\sum_{y \in Y} \frac{V(y|q)}{\mathrm{len}(C)} \log \frac{V(y|q)}{\mathrm{len}(C)}$$

   where $0 \leq V(y|q) \leq \mathrm{len}(C)$ is the number of votes that label y received for query candidate q. The interested reader may refer to Dagan and Engelson [4] for further details.

2. **Kullback-Leibler divergence**: The disagreement for query candidate q is computed

$$d_{\mathrm{KL}}(q) = \frac{1}{\mathrm{len}(C)} \sum_{i=1}^{\mathrm{len}(C)} \sum_{y \in Y} \mathrm{P}_{C_i}(y|q) \log \frac{\mathrm{P}_{C_i}(y|q)}{\mathrm{P}_C(y|q)}$$

   Recall that C is the vector of committee members. We can interpret $\mathrm{P}_C(y|q) = \frac{1}{\mathrm{len}(C)} \sum_{k=1}^{\mathrm{len}(C)} \mathrm{P}_{C_k}(y|q)$ as the probability that label y will have the most votes for q, and $\mathrm{P}_{C_k}(y|q)$ as the probability that committee member k votes y for q. The interested reader may refer to McCallum and Nigam [14] for further details.

An implementation of vote entropy disagreement can be found in Appendix A.6. This function was imported from the `activelearning` package developed by `ramhiser` [16] and simply calls the `entropy` package in `R`.

3.2.3 Query by bagging

Query by bagging and boosting was proposed as a way to improve the performance of a single classifier by forming a committee of classifiers trained on random (weighted, in the case of Boosting) subsets of the labeled data [1]. Bagging uniformly samples k subsets from the labeled data to form a committee of k different classifiers, which are trained with the same classification model (e.g. random forest) [1]. The unlabeled data with the most disagreement among the committee members is then selected as

the next oracle query (see Section 3.2.2 for details on disagreement measures). The algorithm is as follows (see Appendix A.7 for code):

Algorithm 4 Query by bagging (as described by Abe and Mamitsuka [1])

procedure (X is a $n \times d$ matrix of d observations of all n variables, y is an n-length vector of labels for each variable in X (y_i=N/A when $X_{i,}$ has no label). num_class is the desired number of committee members (subsets to sample). Let $r \in (0, 1)$ such that $r * \text{len}(X^{\text{labeled}})$ is the number of points to randomly sample for each subset of the labeled set.)
    loop from $i = 1$ **to** num_class:
        $idx \leftarrow \text{unif\_sample}(\text{labeled}, r * \text{len}(X^{\text{labeled}}))$
        $tout_i \leftarrow \text{train}(X_{idx,}, y_{idx}, \text{classification model})$
        $p_{i,} \leftarrow \text{predict}(tout_i, X^{\text{unlabeled}})$
    $d \leftarrow \text{disagreement}(p)$
    return $\text{where}(d == \max(d))$

Since each iteration of the query selection is a new process of random committee training, there is no starting committee. Instead, what the user specifies is the classification model that each subset will be trained on. Subsequently, there is no need to (1) maintain error ratios to prune a starting committee or to (2) use majority vote with a pruned committee for the final fitted model over a random forest, which is the classification model for stage 2 (Section 2.3).

3.2.4 Min-max clustering

The goal of the Min-Max Approach is to query points that are far from each other [21]. This can be done by selecting a query that maximizes the minimal distance of each unlabeled query from the labeled set [21]. In other words,

$$\max_{q \in X^{\text{unlabeled}}} \left(\min_{k \in X^{\text{labeled}}} \left(\text{distance}(q, k) \right) \right)$$

Given natural clustering in the data, the active learner would be able to sample from each cluster with this methodology. If each cluster is fairly homogeneous, the

final fitted classifier will be a good representation of the data set's "true" labels (Section 3.1). As such, min-max clustering is able to exploit the clustering structure of data but may perform more poorly (converge more slowly i.e. require more queries to get to a reasonable error level) in data sets that do not naturally form clusters. Naturally, there are two important considerations:

- **Initialization:** How does the active learner find the clusters in the first place? Selecting points near the centers of the clusters, which are the densest regions, allows the algorithm to converge faster [21].
- **Query selection:** How does the active learner quantify the distance between data points? Euclidean distance is a common metric of the straight-line distance between points in the Euclidean space.

Vu *et al.* proposed the creation of a k-nearest neighbors graph to measure the local density of each data point; the most dense points may be selected to initialize the active learner [21]. Since the VS will be initialized randomly for reasons described in Section 2.2.1, we do not concern ourselves too much with min-max clustering initialization; the interested reader may refer to [21] for more of the mathematical details and a demonstration of viability. Instead, we present the simple algorithm for the actual active learning (querying) process (see Appendix A.8 for code):

Algorithm 5 Min-max clustering (as described by Vu *et al.* [21])

procedure (X is a $n \times d$ matrix of d observations of all n variables, y is an n-length vector of labels for each variable in X (y_i=N/A when $X_{i,}$ has no label))
    loop from $i = 1$ **to** len($y^{\text{unlabeled}}$):
        min $\leftarrow \infty$
        loop from $j = 1$ **to** len(y^{labeled}):
            $d \leftarrow \text{distance}(X_{i,}^{\text{unlabeled}}, X_{j,}^{\text{labeled}})$
            If (min $> d$) : min $\leftarrow d$
        $q_i \leftarrow$ min
    return where($q ==$ max (q))

3.3 Simulation study

We utilize a simulation study in order to determine the method that is best-suited for usage with the VS in Chapter 5's equity application. Recall that the VS data is composed of all observations of characteristic features (as described in Section 2.1.2) for all $\binom{n}{2}$ pairwise scatter plots of the *actual* data set. To recap, these are key features of the visualization system that should be (and are) captured in the simulations:

- **Initialization (Section 2.2.1):** Initializing the active learner begins with a random selection of points (pairwise scatter plots) that are presented to the oracle for classification. Each simulation is initialized with 10 randomly selected data points.
- **Pool-based sampling (Section 3.1):** After initialization, k unlabeled samples are randomly selected from X, and the active learner picks one to query. Each simulation iteration (of the AL algorithm) is presented $k = 15$ unlabeled points to query from.
- **Random forest (Section 2.3.1):** The VS's overall classification model (for use in stage 2 when stage 1 initialization and querying are complete) is a random forest. Each active learning method in the simulations optimize for the final random forest classification model by utilizing random forests in their selection process (excluding QBC due to the nature of the algorithm). Instead, the QBC simulations have been run with a committee of classification models that includes random forest (See Section 3.3.3 for the full list). Furthermore, the QBC simulations have been run with both (1) majority vote and (2) random forest as the overarching stage 2 classification model, as well as (1) with pruning and (2) without pruning.
- **Binary classification:** The classification of user interests have two possible labels/levels: "visually correlated" and "not visually correlated". The simulations also use data with two levels of classification (Section 3.3.1).

Finally, it should be noted that each active learning algorithm is given a budget of 50 queries (50 progressive iterations of a single trial). While the VS used in the financial application of Chapter 5 will not have such a large budget, its implementation in the simulations allows for proper observation of the performance of each active learning method.

3.3.1 Data

The data is taken from the MNIST database of handwritten digits (`http://yann.lecun.com/exdb/mnist/`). All data have already been classified into digits $(0, 1, ..., 9)$ and may be visualized in the form of a 28×28 pixel array; the hue of each pixel is represented by a value from 0 (light, white space) to 255 (dark, pure black). Each image has been transformed into a single 784-length vector by "unfurling" each row and adding it to the last column of the row above it. For ease of use and computational efficiency, the data has been further compressed to a 196-length vector (14×14 pixel image). In order to maintain the condition of binary classification, two out of the ten digits were selected. The digits 7 and 9 were selected as they are visually similar, making it more difficult for an active learning algorithm to correctly parse the data with few queries (as opposed to 1 and 0, which are visually different). The MNIST training set contains 60,000 total data points while the testing set contains 10,000 total data points. For the sake of computational efficiency, the simulator selects 250 random samples (125 of each digit) from the training set to compose the final data set. The final data set may be visualized in Figure 3.1. The functions for working with the MNIST data are adapted from file `gist:39760` [15] and may be found in Appendix A.9.1.

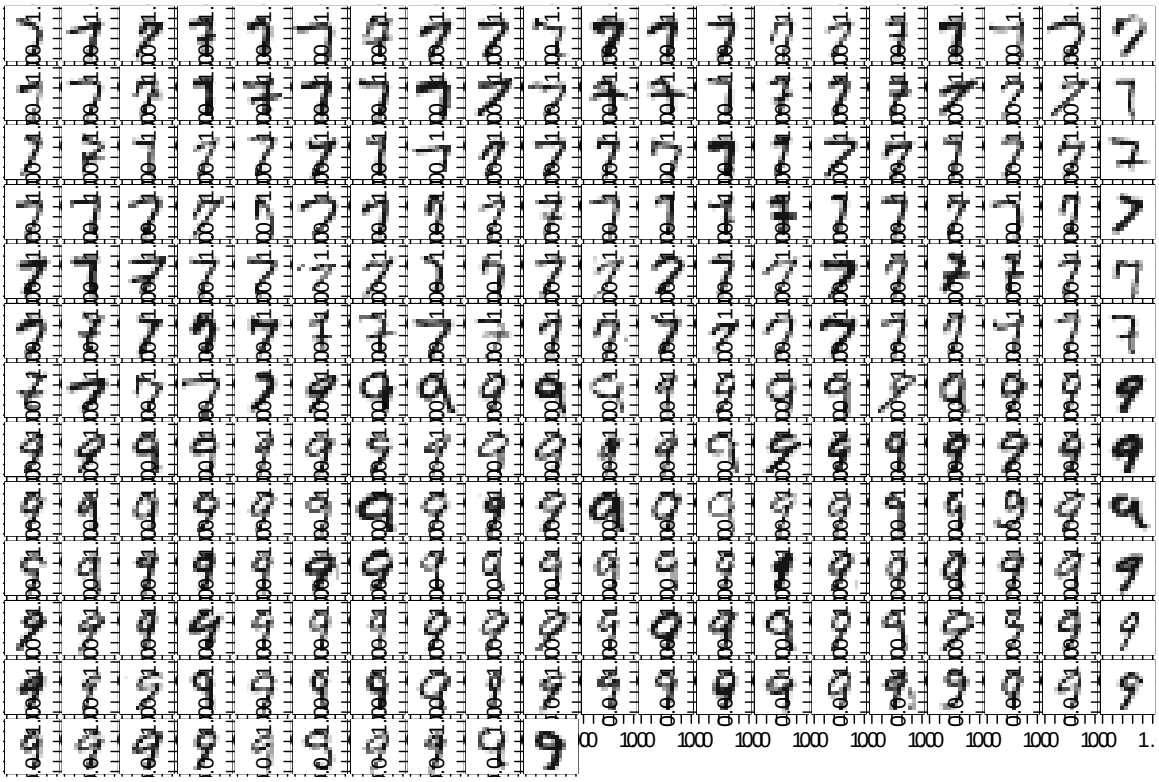


Figure 3.1: MNIST data used in the active learning simulations. The digits 7 and 9 are visually similar, making the simulations more realistic as it is harder for the classification model to achieve a "perfect fit" (given only the initial labeled set) without the use of querying.

3.3.2 Evaluation

The performance of a learner at iteration (query) j of trial i is encapsulated in its error ratio $\epsilon_{i,j}$. Since there are 25 trials for each active learning method, $i \in \{1, ..., 25\}$, and since there are 50 iterations per trial, $j \in \{1, ..., 50\}$. Given the current labeled set, the **main classification model** is trained, and the entire data set is predicted given the resulting classifier. The predictions for iteration j of trial i are stored in an n-length vector $p^{i,j}$. We know the true labels ahead of time (thanks to the MNIST data set), and they are stored in an n-length vector y'. Recall that there are $n = 250$

variables in our data set. The predictions are compared against the true labels and the error ratio of iteration j of trial i is given by:

$$\epsilon_{i,j} = \frac{1}{n}\left(\sum_{z=1}^{n} \mathbf{1}_{p_z^{i,j} \neq y'_z}\right)$$

where $\mathbf{1}_{p_z^{i,j} \neq y'_z}$ is an indicator variable that returns 1 if $p_z \neq y'_z$ and 0 otherwise. Together, these error ratios form ϵ, a 25×50 matrix of error ratios. Each active learning algorithm's error ratios ϵ are averaged *over each iteration* so that the performance of the algorithms may be compared across each iteration. This helps offset the randomness of the initialization and pooling scheme. The final, averaged error ratios are contained in a 50-length vector E where the average error ratio of iteration j is given by:

$$E_j = \frac{1}{s}\left(\sum_{i=1}^{s} \epsilon_{i,j}\right)$$

The final algorithm for computing $\epsilon_{i,}$, a single trial i's vector of error ratios, is summarized as follows (This procedure is encapsulated by the simulation engine code in Appendix A.9.2):

Algorithm 6 Computing $\epsilon_{i,}$, a single trial i's vector of error ratios

procedure (X is a $n \times d$ matrix of d observations of all n variables, y is an n-length vector of labels for each variable in X (y_i=N/A when $X_{i,}$ has no label). y' is an n-length vector of **true** labels for each variable in X. *iter* is the maximum number of queries allowed per trial)
    loop from $i = 1$ **to** *iter*:
        $idx \leftarrow$ active_learning_method$(X, y, ...)$
        QUERY $X_{idx,}$
        $tout \leftarrow \text{train}(X^{\text{labeled}}, y^{\text{labeled}}, \text{classification model})$
        $p \leftarrow \text{predict}(tout, X)$
        $res_i \leftarrow \frac{\text{length}(\text{which}(p \neq y'))}{\text{length}(y')}$
    return res

3.3.3 Summary of methods

Table 3.1 contains a summary of the active learning methods used in the simulation with details such as tuning parameter values. As a point of comparison, random sampling is used as the control because it is the most simplistic base case in which active learning is not present.

Table 3.1: A summary of the active learning methods tested in the simulation. *Note:* The "classification model" column is the main classification model that is used to fit the error and final classifier, not the classification model(s) used in the active learning methods (which is listed under *classifier* or *committee* in the "parameters" column). The "classification model" column in the simulation is akin to the main classification model in stage 2 of the VS.

AL method	**Simulations**	**Classification model***	**Parameters**
Random sampling (CONTROL)	25 trials, 50 iterations, 15 "pooled" points each	Random forest	*Classifier*: None
Uncertainty sampling 3.2.1	25 trials, 50 iterations, 15 "pooled" points each	Random forest	*Classifier*: Random forest
Query by committee 3.2.2	25 trials, 50 iterations, 15 "pooled" points each	Random forest	*Committee*: RF, NB, SVM, PLS *Disagreement*: Vote entropy *C_Pruning*: T, ϵ: 0.5
		Majority committee vote	*Committee*: RF, NB, SVM, PLS *Disagreement*: Vote entropy *C_Pruning*: T, ϵ: 0.5
(QBC CONTROL)		Random forest	*Committee*: RF, NB, SVM, PLS *Disagreement*: Vote entropy *C_Pruning*: F, ϵ: N/A
		Majority committee vote	*Committee*: RF, NB, SVM, PLS *Disagreement*: Vote entropy *C_Pruning*: F, ϵ: N/A

(Continued on next page)

Table 3.1: (continued)

AL method	**Simulations**	**Main classification model**	**Parameters**
Query by bagging 3.2.3	25 trials, 50 iterations, 15 "pooled" points each	Random forest	*Classifier*: Random forest *Disagreement*: Vote entropy *num_class*: 5, r: 0.75
Min-max clustering 3.2.4	25 trials, 50 iterations, 15 "pooled" points each	Random forest	*Classifier*: None *Distance*: Euclidean

QBC was tested with different parameters for the overall/main classification model (random forest vs majority committee vote) and committee pruning (yes or no). This was done in order to see the effectiveness of the addendums proposed in Section 3.2.2 (Algorithm 3) and, ultimately, select the best QBC method in terms of *algorithm structure* for comparison with the other methods. Naturally, the control for comparing the QBC methods is the method with a random forest classification model and no committee pruning (This turns into the QBC methodology as detailed in Algorithm 2). An initial committee was selected from classification models that worked with the data. For example, consider discriminant analysis, which utilizes matrix inversion. The nature of the MNIST data is that many columns are 0 (white space). This is evident from the plots in Figure 3.1. These columns of 0 makes the matrix degenerate, so discriminant analysis cannot function without some form of data cleaning. To avoid further tampering with the data set (as the data has already been compressed), LDA, DDA, FDA, etc. were not included in the initial committee. A brief overview of each classification model used in the initial committee for QBC implementation is as follows:

- **Random forest:** A random forest is a collection of decision trees which are grown from independent draws of the training set. A more detailed description can be found in Section 2.3.1.

- **Naive Bayes:** The naive Bayes classification model applies Bayes' Theorem $\left(P(y|x) = \frac{P(x|y)P(y)}{P(x)}\right)$ to compute the posterior probability of label y given characteristic feature x in order to classify the data. It is important to note that the algorithm assumes independence among the features that characterize each label, which may not necessarily be the case.
- **Support vector machine:** In the support vector machine classification model, each data point in X is mapped to a space in $\mathbb{R}^2$ where the various labels are split by a line that makes sure they are as far apart as possible from one another. As new labeled points are queried, the mapping updates. SVMs can also perform nonbinary and nonlinear classification through the usage of a kernel.
- **Partial least squares:** In partial least squares, a *principle component analysis* is first performed on the data set X. Principle component analysis takes the n-length set of variables in X (which may or may not be correlated) and converts them into some set X' of linearly uncorrelated "principle components". Principle components are not necessarily individual variables; they may instead be a combination of variables (e.g. X_iX_j, $X_i^2X_j$, etc.), thereby accounting for potential dependence in the data. PLS then selects the principle components which best explain the observed response in Y (the labels), performs a linear regression on the selected components, and uses the results in order to classify the data.

The call to each active learning function is controlled by the active learning engine code found in Appendix A.9.3. By implementing the active learning call in this way, the main active learning functions are hidden from the end user so that they cannot call the functions directly, which may lead to bypassing checks and/or improperly calling functions.

3.3.4 Results

The full simulator which loads the data (using functions in Appendix A.9.1), calls the simulation engine 25 times for each active learning method (the simulation engine, Appendix A.9.2, calls the active learning engine, Appendix A.9.3), and plots the averaged error ratios may be found in Appendix A.9.4.

Figure 3.2 is a summarization of the performance of query by committee with parameters specified in Table 3.1. With the main classification model, majority committee vote outperforms random forest on average in the first half (25 iterations).

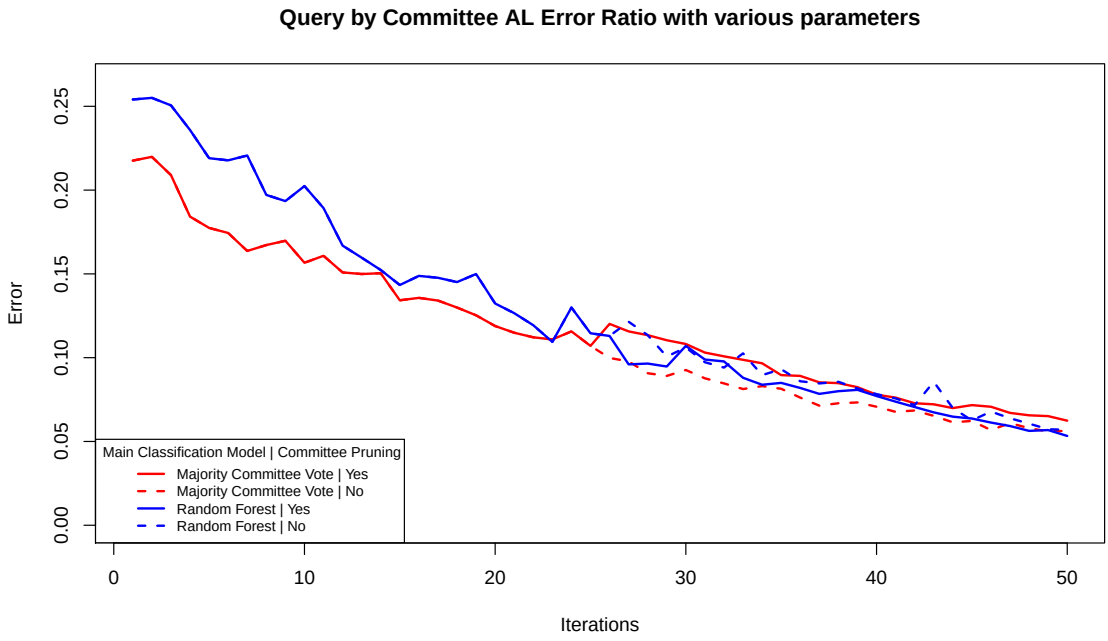


Figure 3.2: Active learning simulation results for query by committee with varying parameters. Each of the 25 trials were consistently seeded with its own value, so the values before committee pruning would/would not have occurred (at $iter/2 = 25$) are the same.

What is interesting to note in iteration 25 onwards is that the error ratio for majority committee vote (with committee pruning) spikes up *after pruning has occurred*. This is interesting since the idea was to prune committee who performed poorly; this indicates that the insight of *all* committee members, despite performance (or with more room for error than 0.5), are valuable. Naturally, this phenomenon cannot be observed in the random forest (with committee pruning) results because the classification model is a random forest; since committee members are not weighing in on the final predicted labels, the members themselves hold less weight. Although majority committee voting (without committee pruning) continues to maintain its lead past 25 iterations, the error ratios converge as the number of iterations gets closer to 50. In fact, near iteration 50, it appears that both random forest methods (with and without committee pruning) start converging faster and just barely overtake majority commit-

tee vote (without committee pruning) at the end. In other words, QBC with majority voting (without committee pruning) starts off better but improves more slowly while QBC with random forest (both with and without committee pruning) starts off worse but improves more rapidly. Each method has its own trade-offs, so it becomes all the more important to keep the end goal, the visualization system, in mind. Recall that an important criterion for active learning performance is performance given less query points. The user should not have to label more than fifty plots in order for the system to learn his/her interests; not only is it tedious for the user, but this tedium may cause the user to perform the tasks with less seriousness or cause the user's concentration to slip. Subsequently, we select **QBC with majority committee vote and no committee pruning** for comparison with the other algorithm types.

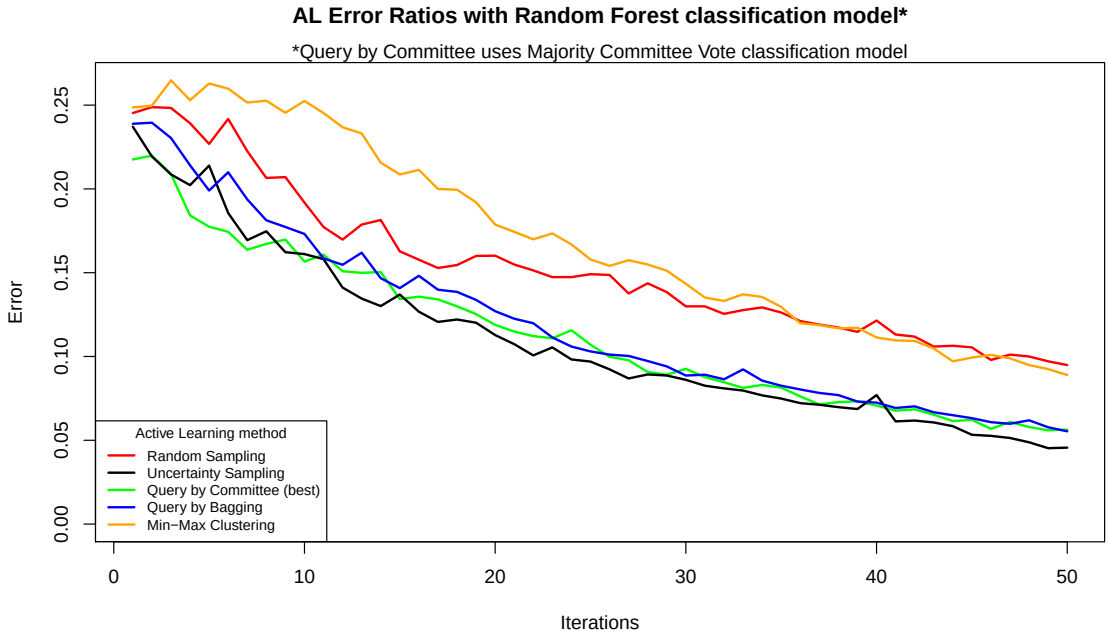


Figure 3.3: Aggregated active learning simulation results. Parameters for each method may be found in Table 3.1.

Figure 3.3 summarizes the performance of the selected QBC method against the other active learning methods and random sampling as a control (a baseline for com-

parisons). Random sampling actually performs better than one might expect, though it is the first to fall off (converge more slowly). This may be, in part, due to the constraints of pool-based sampling. In each iteration, random sampling was also constrained to 15 random points to query from. Thus, each active learning method cannot optimize over the entire sample space.

What is especially interesting is that random sampling performs better than min-max clustering for the longest time, but min-max clustering converges more rapidly and eventually overtakes random sampling. This may be attributed to the nature of the active learner's initialization. With clustering algorithms, initialization is extremely important; by starting within a clusters, the min-max methodology allows the learner to sample more informative labels without having to search through the entire space for clusters. In the simulator, recall that each active learning method is initialized randomly with the same set of data and starting points. We suspect that if we had initialized the min-max clustering method with the k-nearest neighbors graph as suggested by Vu *et al.* [21], the min-max clustering algorithm would have performed better.

Query by bagging, overall, performs worse than query by committee. In fact, it performs similarly to QBC with a main random forest classification model, whose performance can be seen in Figure 3.2. This isn't surprising because the bagging algorithm itself uses a committee of 5 random forest classifiers and also uses a random forest for the main classification model.

There appears to be another "tie" between query by committee and uncertainty sampling. QBC starts off better, but the faster convergence in uncertainty sampling eventually outperforms QBC around $iter = 10$. Unlike the case of QBC with a random forest versus majority committee vote classification model (Figure 3.2), uncertaintly sampling overtakes QBC much earlier. Thus, out of all methods described in Table 3.1, **uncertainty sampling** most closely embodies the qualities that char-

acterize a strong active learner: intelligent selection of queries that allow the main classifier to learn interests more quickly. It would seem natural to select $iter = 10$ for the financial application in Chapter 5, but recall in Section 2.2.1 that we would like to compensate for the randomness in the initialization by providing a larger query budget. As such, the VS stage 1 budget is set to $iter = 15$ for the financial application.

Extensions

Naturally, there are many other parameters that could have been tested but were not in the interest of time. For example, the classification model used in the active learning method to aid in query selection does not have to be a random forest (This applies to uncertainty sampling and query by bagging); perhaps another would have worked better despite the intuition that, if the active learning methods optimize over the same classification model that the main program (the VS, in this case) will use, the learner will perform better. As noted earlier, initialization is important for min-max clustering, and the performance of the learner could also be tested with different initialization schemes. Furthermore, there are other disagreement methods that could be implemented aside from vote entropy, one of which was described earlier in Section 3.2.2 (This applies to query by committee and query by bagging). The tuning parameters of ϵ in QBC and r in QBB could also be perturbed to see the effect on the error ratio. Perturbing ϵ may also provide insight into why QBC suddenly performs worse the moment a committee is pruned. Finally, different initial committees could be fed into the QBC method, and different starting points for committee pruning (e.g. $iter/3$ or $iter/4$ instead of $iter/2$) could also be tested.

Chapter 4

Graph comparison

Graph comparison may be reduced to a problem of measuring the difference between two graphs. A primary goal of the visualization system is to allow the user to better verify their numerical results with visual intuition. As such, there are two main goals for the graph comparison output: (1) allow the user to gain some intuition about the qualities of the visual graph in order to better understand their perceptions of a "visual trend" in the data, and (2) provide a metric that quantifies the difference between the visual graph and numeric graph (supplied by the user). With the knowledge of both in hand, the user may select the numerical method which is "closest" to their visual intuition and utilize the results in alternate applications (as in the stock selection in Chapter 5). It should be noted that we operate under the assumption that the graphs are labeled. This avoids the sticky issue of isomorphism and more naturally lends itself to the visualization system's goals. As the VS is concerned with practical application in data analytics, each variable in a dataset should already correspond to some label (e.g. a stock). What follows is broad overview of two graph comparison methods in Section 4.1 with more details on the graph summarization comparison method in Section 4.2. Finally, we propose a method to select the graph $\hat{G}^i, i \in \{1, ..., k\}$ out of k given graphs where $\hat{G}^i$ is most similar to the "base" graph $\hat{G}$ and perform a

simulation study to demonstrate the method's viability in Section 4.3. Note that we do not use the notation $\hat{G}^{i,\text{num}}$ for the k graphs which we wish to compare to $\hat{G}$ because the methods discussed in this section may be applied to any graphs, not just numerical and visual correlation graphs.

4.1 Graph summarization

There are two ways of thinking about graph comparison. Let $\hat{G}^1 = (V^1, E^1)$ and $\hat{G}^2 = (V^2, E^2)$ each be some labeled graph.

1. **Graph summarization:** Compute some vector v^1 on $\hat{G}^1$ that "summarizes" various properties that it has. Similarly, compute v^2 on $\hat{G}^2$ to capture the same properties that v^1 does. Finally, compute $f(v^1, v^2)$ where f is some distance function (e.g. Euclidean distance). A high value indicates strong dissimilarity while a low value indicates similarity.
2. **Graph distance:** Create a valid distance function f on the graphs $\hat{G}^1, \hat{G}^2$ directly rather than a vector of metrics (v^1, v^2). Compute $f(\hat{G}^1, \hat{G}^2)$. Again, a high value indicates strong dissimilarity while a low value indicates similarity.

We specifically focus on the graph summarization comparison because it fulfills both desired criteria for the visualization system; an understanding of each summarization method allows the user to gain intuition on the qualities of the graph, and the difference between two graph summarization methods can be quantified easily. To be more specific, we seek to better understand the qualities of existing graph summarization methods and their associated advantages and disadvantages. Understanding these metrics is important due to the complex nature of graphs. It is natural to ask why we cannot just plot the graphs instead (which are typically in the form of either an adjacency matrix of list of edges) and examine the result for visual patterns in order to understand the qualities of the graph. This is unfeasible for several reasons (see Figure 4.1).

- **Arrangement:** Different arrangements of nodes on the visual plane may have a profound effect on the interpretability of the graph; from the start, it is unclear

what the best arrangement might be, and it is unfeasible for an analyst to try all possible arrangements especially as the number of variables increase.

- **Density:** The more nodes there are, the more edges there may be, and the more dense a graph may become, making it incredibly difficult to interpret. Furthermore, it makes it more difficult to find anomalies among the variable's relationships (represented by edges). A non-existent edge in a dense graph is just as important as an existent edge in a sparse graph because each indicates a relationship which isn't quite like all the others.

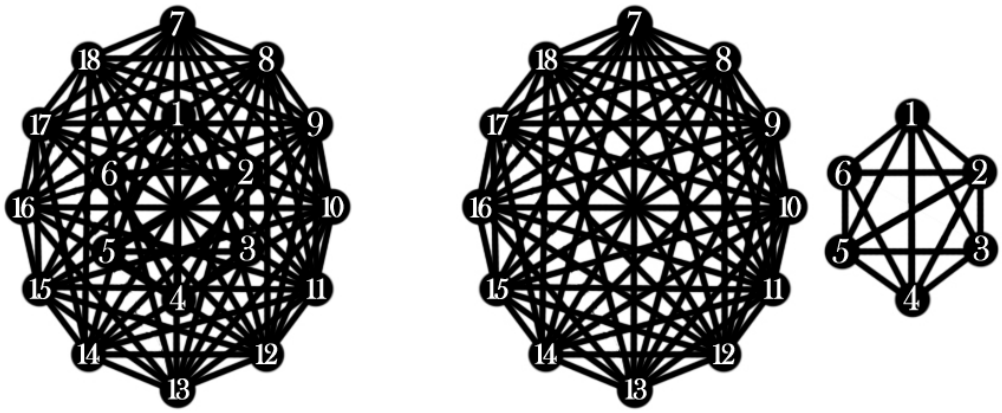


Figure 4.1: Suppose we have a graph $G = (V, E)$ with clusters C^1, C^2 and $|V| = 18$. Let C^1 be composed of nodes $(V_1, ..., V_6)$, and C^2 be composed of nodes $(V_7, ..., V_{18})$. *Left:* C^1, the smaller cluster, has been placed inside C^2, the larger cluster. It is difficult to discern any meaningful patterns; each node appears to be connected to every other node as the density of the cluster C^2 makes it almost impossible to see that of C^1. *Right:* The nodes have been rearranged in a manner that clearly distinguishes between C^1 and C^2. At a glance, it is evident that C^1 is missing $E_{3,6}$. It is difficult to tell, however, that C^2 is even missing an edge ($E_{13,17}$) in the first place! C^1 is more sparse, which makes it easier to visually perceive anomalies in the graph.

Graph summarization acts as a proxy for visually exploring the graph itself. Each metric, which is associated with a different characteristic of the graph, can be parsed and later combined to "reconstruct" or better understand the qualities of each graph, subsequently allowing the user to better understand *exactly where* the visual and numeric graphs differ (should the final graph comparison metric suggest that the two

graphs are highly different). Graph distance functions skip that critical step by going through the graphs directly.

To put it in another way, graph summarization has the *exact opposite problem* that is present in data analytics as it is practiced today (discussed in Chapter 1, this is the motivation behind the VS and this work). The visualization system allows the user to use the visual qualities of pairwise scatter plots to better understand numerical qualities of the data, but graph summarization allows the user to use numerical qualities to better understand the visual qualities of a graph.

With an understanding of the properties of various graph summarization methods, the graph summarization difference metric may be more informative. In the VS, the graph summarization difference computes a list of graph summarization metrics $(m^{\text{num}}, m^{\text{vis}})$ for each graph in order to provide a single vector d that quantifies the difference between them with some difference function (e.g. Euclidean distance, L2, etc.). As such, a natural byproduct of the computation is $(m^{\text{num}}, m^{\text{vis}})$, which (armed with an understanding of the values) provides insight on the qualities of the visual graph and numerical graph *on an individual level* and where exactly their differences and similarities lie.

4.2 Overview of graph summarization methods

There are many different ways to summarize a graph. However, akin to how a correlation coefficient cannot sufficiently capture all aspects of dependency on its own, there is no single summarization method that can summarize every quality of a graph on its own. As such, this section discusses various graph summarization methods, the qualities that they measure, and their drawbacks. Furthermore, it is useful to normalize the returned summarization difference in order to understand the value scale properly, so a normalization method is proposed for each metric difference. In sec-

tion 4.3, we propose a method to select the most similar graph pair given a vector of differences where each difference corresponding to one of the six summarization methods listed below.

Most methods listed below are implemented in the `igraph` package in `R`. As such, we do not implement most methods ourselves as we did in Chapter 3. The code for all methods may be found in Appendix A.10. It should be noted that there are various distance functions that may be used. Euclidean distance, L1, and L2 are currently implemented (The code for the distance functions may also found in Appendix A.10).

For all methods below, let $\hat{G} = (V, E)$ be some labeled graph.

4.2.1 Centrality

The centrality of a graph $\hat{G}$ measures the importance of its nodes. Let c be a vector of length $|V|$ where each element c_i is the "importance" of node i. Then the centrality of $\hat{G}$ is given by:

$$\mathcal{C} = \sum_{i=1}^{|V|} \max_{\forall j \in \{1,\ldots,|V|\}} c_j - c_i$$

The importance of each node may be measured by utilizing metrics such as:

- **Degree:** The degree of a node is the number of edges the node is part of. In other words, it is the number of direct paths from the node to all other nodes. The higher the degree, the more important the node is.
- **Closeness:** The closeness of a node is the average shortest path length (the path which traverses the smallest number of edges) between the node and all other nodes. The higher the closeness, the closer the node is to all other nodes (in terms of path length), the more important the node is.
- **Betweenness:** The betweenness of a node is the number of times the node is part of the shortest path between two other nodes. Direct paths are excluded. The higher the betweenness, the more important the node is as it provides a critical cost-saving link (path length) for the path between many other nodes.

The normalized centrality of a graph $\hat{G}$ may be computed as $\frac{Cen(\hat{G})}{\max Cen(\hat{G})}$ where $\max Cen(\hat{G})$ is the maximum centrality that graph $\hat{G}$ may have (for whatever metric

is specified). Taking the difference between two normalized scalars will also yield a normalized scalar. Degree centrality is implemented in `centr_degree`, closeness centrality is implemented in `centr_clo`, and betweenness centrality is implemented in `centr_betw` from the `igraph` package. By default, each function normalizes the returned centrality coefficient.

Now consider a graph whose edges are rotated as in Figure 4.2; the centrality of the overall graph will not change as the edges are in the same relative positions, but the graph itself is clearly different. The computed centrality difference (with distance function L2 and for all centrality types) between the graphs pictured in Figure 4.2 is 0, which suggests they are exactly the same. This is due to the "global" nature of the centrality metric, which ignores the individual label of the nodes. Thus, when edges are rotated and nodes change, the graph "looks" exactly the same. Should centrality be examined at node-level and not at a graph-level, this issue would be accounted for, though normalization is not as trivial. It is more difficult to efficiently parse node-level metrics especially in high dimensions, so this paper does not detail or implement this metric.

4.2.2 Assortativity

Assortativity measures the level of resemblance among the connected nodes of a graph $\hat{G}$ based on vertex category labels. For graphs with uncategorized nodes (as in the examples of this chapter and the application in Chapter 5), it is common to use $\deg(V_i) - 1$ as the category label for node V_i instead. For all edges $E_{i,j}$ that exist in $\hat{G}$, the degree of all associated nodes V_i and V_j are collected in two distinct vectors v^1, v^2. For instance, if $|V| = 4$ and $\hat{G}$ contained the set of edges $\{(1,2),(1,3),(2,4)\}$, then $v^1 =< 2,2,2 >$ and $v^2 =< 2,1,1 >$. Then the assortativity metric of a graph $\hat{G}$ is computed:

$$\mathcal{A} = \hat{\rho}^{\text{Pear}}(v^1, v^2)$$

where $\hat{\rho}^{\text{Pear}}$ is the Pearson correlation as defined in Section 1.2. As such, the assortativity of $\hat{G}$ is summarized in a single value $-1 \leq \mathcal{A} \leq 1$ where the interpretation is the same as that of the Pearson correlation coefficient. For instance, $\mathcal{A} = 1$ when nodes with high/low degrees in $\hat{G}$ are mostly connected to other nodes with high/low degrees in $\hat{G}$, respectively. In other words, if $\mathcal{A}$ is large, then connected vertices tend to have the same qualities, and when $\mathcal{A}$ is small, then connected vertices tend to have opposite qualities. Since assortativity involves a correlation computation, the result is already normalized. Again, taking the difference between two normalized scalars will also yield a normalized scalar. Assortativity is called with the `assortativity_degree` function from the `igraph` package.

Similarly, the assortativity difference metric is susceptible to rotated edges as shown in Figure 4.2. Assortativity is another "global" metric that measures the correlation among the degree of nodes but does not care about the exact nodes themselves. As such, the metric can be potentially misleading in situations where the actual nodes matter. In fact, the computed difference (with distance function L2) between the graphs in Figure 4.2 is 0.0032, which suggests the graphs are very similar when they are (visually) different.

4.2.3 Community

The community of a graph $\hat{G}$ is given by $\mathcal{S}$, a set of its dense subgraphs. Subgraphs may be detected with any of the following algorithms:

- **Random walks:** By implementing a random walk within a graph to determine the various subgraphs (or, in other words, to "describe" the community), this procedure leverages the natural idea that random walks within a graph will tend to stay in the same subgraph. This occurs because a subgraph is often densely connected within itself but sparsely connected to other subgraphs, so there is an increased probability of proceeding via an edge that is within the same subgraph. This approach to building a community is called with the `cluster_walktrap` function from the `igraph` package.

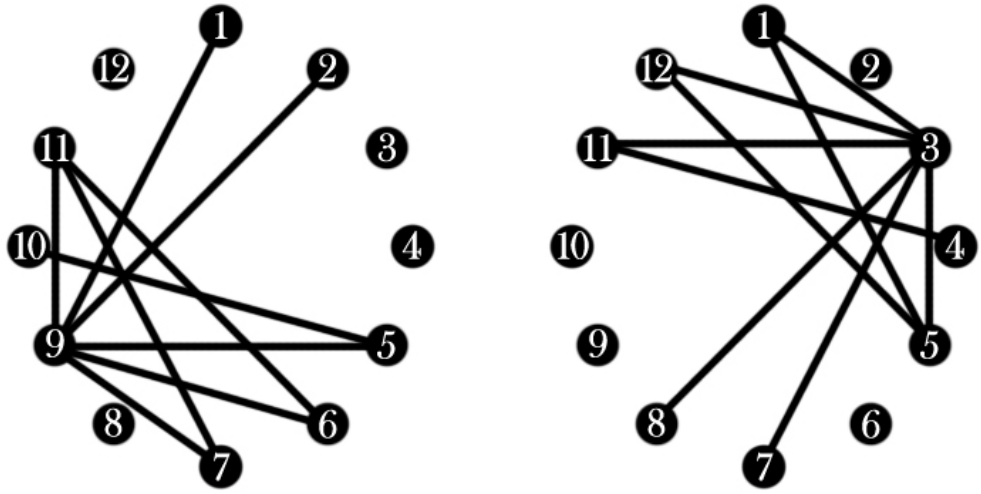


Figure 4.2: A graph pair with rotated edges. These labeled graphs are clearly different from each other. Graph summarization differences (L2 difference function):

Centrality (degree)	Centrality (closeness)	Centrality (betweenness)
0	0	0

Community (random walk)	Community (infomap)	Community (betweenness)
0.9677	0.9677	0.9677

Assortativity	Distance matrix	Edge connectivity	Edge density histogram
0	0.3557	0	0

- **Infomap:** An infomap utilizes random walks to build a community by minimizing the expected length that it takes a random walk to "describe" the community. This approach to building a community is called with the `cluster_infomap` function from the `igraph` package.

- **Betweenness:** As described earlier, the betweenness of a node is the number of times the node is part of the shortest path between two other nodes (where direct paths are excluded). The algorithm sequentially removes the node with the highest betweenness and places it on a tree, recalculating betweenness of the existing graph every time a removal is performed. This algorithm once again takes advantage of the natural idea that subgraphs tend to be dense among themselves but with few edges connecting them to other subgraphs. As such, edges which connect separate subgraphs have high betweenness because all shortest paths from one subgraph to another must pass through them. This approach to building a community is called with the `cluster_edge_betweenness` function from the `igraph` package.

There is no way of normalizing $\mathcal{S}$ because it is not a numeric value. What is important to note, however, is that comparison among graphs $\hat{G}^1 = (V^1, E^2)$ and $\hat{G}^2 = (V^2, E^2)$ for the community graph difference computation (which *does* return a numeric value) must compare $\mathcal{S}^1$ and $\mathcal{S}^2$ with a method such as the Jaccard distance. Consider sets labeled $\mathcal{S}^1 = (1, 1, 2, 2, 3, 3)$ and $\mathcal{S}^2 = (4, 4, 5, 5, 6, 6)$. These are the exact same sets, but they are labeled with different values; thus, a traditional distance metric (e.g. Euclidean distance) would not capture the difference properly. The Jaccard distance compares the actual sets rather than their literal values, and it is already normalized between 0 and 1. The Jaccard distance is obtained by subtracting the Jaccard similarity from 1. The Jaccard similarity is computed with the `cluster_similarity` function in the `clusteval` package.

Where community has trouble is when a graph pair has many sparse clusters. Consider the simplified example found in Figure 4.3. While one graph has two physical clusters and the other has three, they are not all that visually different because only one edge has been removed. The community difference values are 0, 0.2727, and 0.2727 for random walk, infomap, and betweenness respectively. Interestingly, the random walk community difference is 0; this is likely because it is a *single random walk* which leaves more room for errors and may not fully explore the clustering structure especially if clusters are completely isolated as is the case in both graphs of the set illustrated in Figure 4.3. The values of 0.2727, on the other hand, are extremely significant when considering that the "best case" scenario where the split communities (6,7,8,9) and (10,11,12) from Figure 4.3 (right) are *perfectly recognized by the subgraph detection algorithm* yields a Jaccard difference of 0.3870968.

4.2.4 Distance matrix

The distance matrix M of a graph $\hat{G}$ is a square matrix that contains the shortest path length between all variable pairs. Each element $M_{i,j}$ contains the shortest path

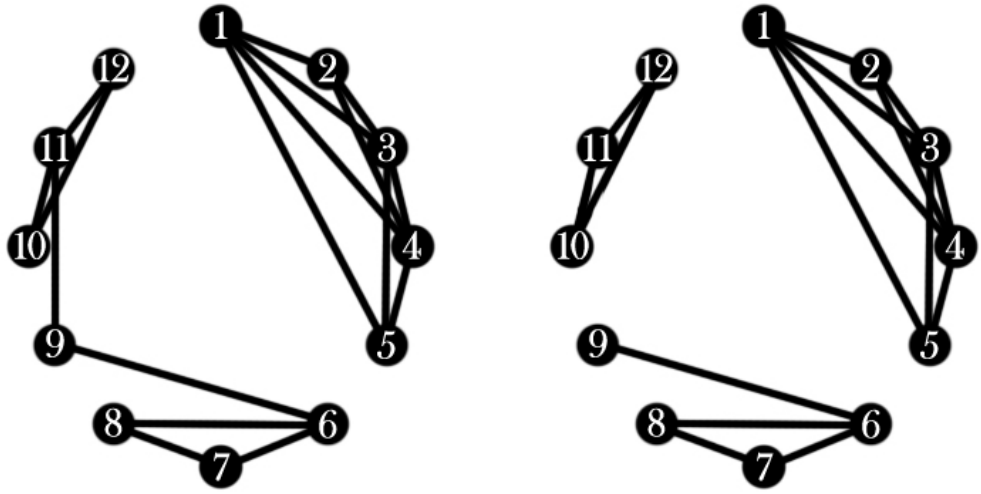


Figure 4.3: A graph pair with two and three clusters, respectively. Only one edge has been removed, so the graphs do not appear that extremely different. Graph summarization differences (L2 difference function):

Centrality (degree)	Centrality (closeness)	Centrality (betweenness)
0.0002	0.0002	0.0102

Community (random walk)	Community (infomap)	Community (betweenness)
0	0.2727	0.2727

Assortativity	Distance matrix	Edge connectivity	Edge density histogram
0.0075	0.2233	0	0.0139

distance between node i to node j. If no path exists, it is conventional to set the value to infinity. However, since our purpose is to compare two matrices, setting a value of infinity doesn't make sense. Alternatively, the value may be set to zero since we define the shortest path distance in Section 4.2.1 to be the path which traverses the smallest number of edges; thus, a *direct path* between nodes has path length 1. Subsequently, the shortest path length can never be 0, which allows the existence of a "non-existent path" to be distinguished from the "length of an existing path". It is important to note that the matrix itself is repeated; the upper and lower triangles are the same. Furthermore, the entries across the diagonal must be 0. Thus, the matrix can be unfurled into a vector m with the values of all $\binom{|V|}{2}$ unique pairs. It is less

trivial to normalize the distance matrix difference metric. It would be incorrect, for instance, to normalize by dividing the raw distance by $z = \text{dist}\big(|V| - 1, 0\big) * \text{len}(m)$ because it is impossible for a graph to be constructed such that every node's *shortest path* to all other nodes is exactly length $|V| - 1$ (the maximum possible degree) or for a graph to be constructed such that the value is even close to z. The simplest method would be to treat the difference between an empty graph (a graph with no edges) and a graph with a single Hamiltonian path (a path that visits each vertex once) as the upper bound. This yields a bound that is actually slightly above 1, but it is close enough for the purposes of interpretation. The distance matrix method is called with the `distances` function from the `igraph` package. `distances` returns infinity when a path doesn't exist, so the output is adjusted accordingly.

The distance-matrix is more granular. Because the metric relies on the shortest paths between all unique pairs of nodes, it is a node-level metric. As such, edge rotation is not a problem, but clustering becomes an issue when there are few clusters. Consider Figure 4.4; the removal of edge (4,5) forces the shortest path between pairs with a node from each cluster to drop to 0 and has a strong effect on the distance matrix distance since many pairs are now disconnected. The actual distance matrix difference for the graphs pictured in Figure 4.4 is 0.7708, which suggests that the graphs are very different when they are actually quite similar visually (only one edge has changed, so the mind perceives the same clustering structure). Although Figure 4.3 is similar to Figure 4.4 in that both involve cluster manipulation, the larger number of clusters in Figure 4.3 ensures that disconnecting two clusters will not change the distance matrix difference as much; less of the graph is interconnected, so fewer pairs will have a distance of 0. In fact, the distance matrix difference of Figure 4.3 happens to be 0.2233, which is much lower than the distance matrix difference of Figure 4.4.

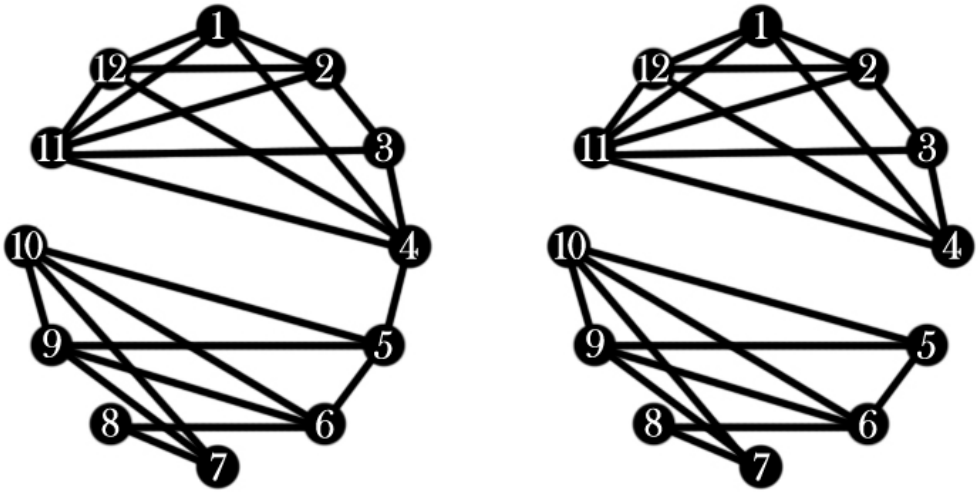


Figure 4.4: A graph pair with one and two clusters, respectively. Only one edge has been removed, so the graphs do not appear that extremely different. Graph summarization differences (L2 difference function):

Centrality (degree)	Centrality (closeness)	Centrality (betweenness)
0.0002	0.0899	0.1933

Community (random walk)	Community (infomap)	Community (betweenness)
0	0	0

Assortativity	Distance matrix	Edge connectivity	Edge density histogram
0.0032	0.7708	0.25	0.0139

4.2.5 Edge connectivity

The edge connectivity of a graph $\hat{G}$ is given by $\mathcal{E}$, the number of edges that need to be deleted in order to disconnect $\hat{G}$ such that there exists two nodes V_i, V_j in $\hat{G}$ which do not have a path between them. As the metric is upper bounded by the minimum degree of any node within a graph, the metric may be normalized by computing $\frac{\mathcal{E}}{\min_i \deg(V_i)}$ for all possible nodes $i \in \{1, ..., |V|\}$. Then, the difference normalization will naturally follow. The edge connectivity of a graph is computed with the `edge_connectivity` function from the `igraph` package.

It is natural to see where this metric could fall through the cracks as it also "global" and ignores the labels of each node. Consider a graph whose edges are rotated as in

Figure 4.2; the number of edges needed to disconnect the graph won't change as the edges are in the same relative positions, but the graph itself is clearly different. In fact, the computed edge connectivity difference (with distance function L2) between the graphs pictured in Figure 4.2 is 0, which suggests they are the exact same.

4.2.6 Edge density histogram

The edge density histogram captures the number of nodes in a graph $\hat{G}$ that have a specified number of degrees. Naturally, a larger number of high-degree nodes signals that $\hat{G}$ is dense, and a larger number of low-degree nodes signals that $\hat{G}$ is sparse. For $\hat{G}$, the edge density is summarized in a vector v of length b, the number of desired "bins" (e.g. bin 1 is $1-5$ degrees, bin 2 is $6-10$ degrees, etc.). Then the value v_i is the number of nodes which correspond to bin i. This metric may be normalized by computing $\frac{v_i}{|V|}$ for all $i \in \{1, ..., b\}$ since there are a total of $|V|$ nodes; thus, each bin is upper bounded by $|V|$. Normalizing the graph-specific metric v then allows the difference computation to be normalized. This summarization method is not implemented in the `igraph` package.

The edge density histogram difference metric is also susceptible to rotated edges as illustrated in Figure 4.2 because it aggregates nodes into bins, which discards the labels of each individual node; the number of high-degree and low-degree nodes will not change even if the edges are rotated, but the graph is still clearly different. In fact, the computed edge density histogram difference (with distance function L2) between the exact graphs in Figure 4.2 is 0, which suggests that the graphs are exactly the same.

4.3 Similarity selection

Given a set of k graphs $\hat{G}^i$ for $i \in \{1, ..., k\}$, we would like to select the graph $\hat{G}^*$ which is most similar to the "base" graph $\hat{G}$. Each of the k given graphs are associated with a vector d^i of their difference to $\hat{G}$. Each element d^i_m for $m \in \{1, ..., 10\}$ corresponds to one of the ten summarization methods listed earlier in Section 4.2 (different flavors of centrality and community are included). We propose the following methodology to select the most similar graph pair:

1. Compare each difference metric d^i_m against $d^{i'}_m$ for all $m \in \{1, ..., 6\}$ and $i, i' \in \{1, ..., k\}$ where $i \neq i'$.
2. Select the index i which has the most number of smallest differences. In other words, find

$$\max_{i \in \{1,...,k\}} \sum_{m=1}^{10} \mathbf{1}_{d^i_m > d^{i'}_m} \text{ where } i \neq i' \in \{1, ..., k\}$$

Aside from returning the most similar graph pair, the proposed similarity selection method further eliminates the need to normalize the resulting difference metrics (which will not be on the same scale within the vector d^i), as it compares each metric to its own metric rather than using d^i to compute a scalar "difference" between graphs $\hat{G}^i$ and $\hat{G}$. In the next section, we demonstrate the viability of this strategy with a simulation study. This method is later applied in Chapter 5 to compare the visual correlation graph (formed from the VS) with the various numerical correlation graphs (formed from the correlation coefficients described in Section 1.2). The similarity selection method is coded in Appendix A.11.

4.3.1 Simulation study

We now perform a simulation study to demonstrate the viability of the proposed selection strategy. Let $G = (V, E)$ be a base graph with $|V| = 20$ nodes and $|E| = 50$ edges. The idea is to generate two random graphs G^1, G^2 from G such that $\text{diff}(G, G^1) << \text{diff}(G, G^2)$ or vice versa. In other words, one graph should clearly be

similar to G while the other is very dissimilar to G. This is achieved by randomly swapping $\frac{|E|}{5} = 10$ and $|E| = 50$ edges from G to build G^1 and G^2, respectively. The random swapping algorithm is as follows:

Algorithm 7 Random edge swapping

procedure ($G = (V, E)$ is the "base" graph. Generate a random graph $G^i = (V, E^i)$ by randomly swapping e edges of G. Let p_k be the probability of randomly selecting node V_k.)
    $G^i \leftarrow G$
    loop from $f = 1$ **to** e:
        Select an edge uniformly at random and delete from G^i
        Select V_i uniformly at random
        Select V_j at random where $p_k = \frac{\deg(V_k)}{\sum_{x=1}^{|V|} \deg(V_x)}$ $\forall\ k \in \{1, ..., |V|\}$
        while ($V_i == V_j$ **or** edge (V_i, V_j) exists):
            Reselect V_j at random with the PDF given earlier*
        Add edge (V_i, V_j) to G^i
    return where$(p == \min(p))$

*Selecting the second node in this manner ensures that the assortativity coefficient continues to evolve as the swapping proceeds.

Naturally, G^1 is more similar to G since less edges are being swapped than in G^2. The differences between the pair (G, G^i) for $i \in \{1, 2\}$ are computed with the summarization methods discussed in Section 4.2, and the most similar graph is selected with the similarity selection method described in Section 4.3. The results are recorded for 1000 trials, and an average is computed. The code for the simulation study may be found in Appendix A.12.

4.3.2 Results

The results are summarized in Table 4.1.

Table 4.1: A summary of the similarity selection simulation results as described in Section 4.3.1. The table summarizes the probability of selecting graphs G^1 and G^2 with the given number of swaps. (C) is an abbreviation for (CONTROL)

Distance	**Number of swaps**	**Prob**(G^1)	**Prob**(G^2)
Euclidean	$G^1 : 10, G^2 : 10$ (C)	0.54	0.46
	$G^1 : 10, G^2 : 50$	0.885	0.115
L1	$G^1 : 10, G^2 : 10$ (C)	0.532	0.468
	$G^1 : 10, G^2 : 50$	0.886	0.114
L2	$G^1 : 10, G^2 : 10$ (C)	0.541	0.459
	$G^1 : 10, G^2 : 50$	0.885	0.115

Though the control is slightly skewed towards G^1, it is still clear that, on average, the proposed similarity selection method is more likely to properly select G^1, the actual "most similar graph". To visualize the differences between 10 vs 10 and 10 vs 50 swaps, please review Figures 4.5 and 4.6, which plot G, G^1, and G^2 from the final simulation trial. Because the control and the regular simulations were run separately and because each trial is seeded with its trial number, G and G^1 (which has 10 swaps in each simulation) are the same in Figures 4.5 and 4.6; only G^2 differs.

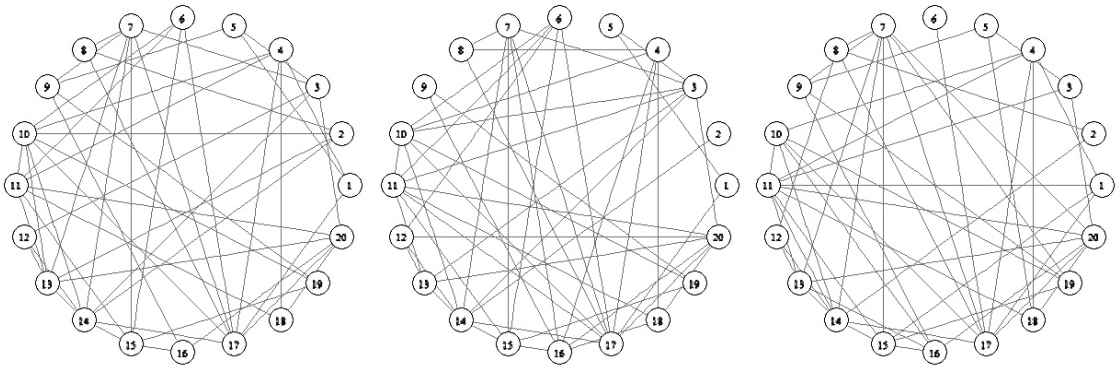


Figure 4.5: *Left:* Base graph $G = (V, E)$ with $|V| = 20$ nodes and $|E| = 50$ edges. *Middle:* Random graph G^1 built from G by swapping 10 edges. *Right:* Random graph G^2 built from G by swapping 10 edges. Distance: L2. Seed: 1000

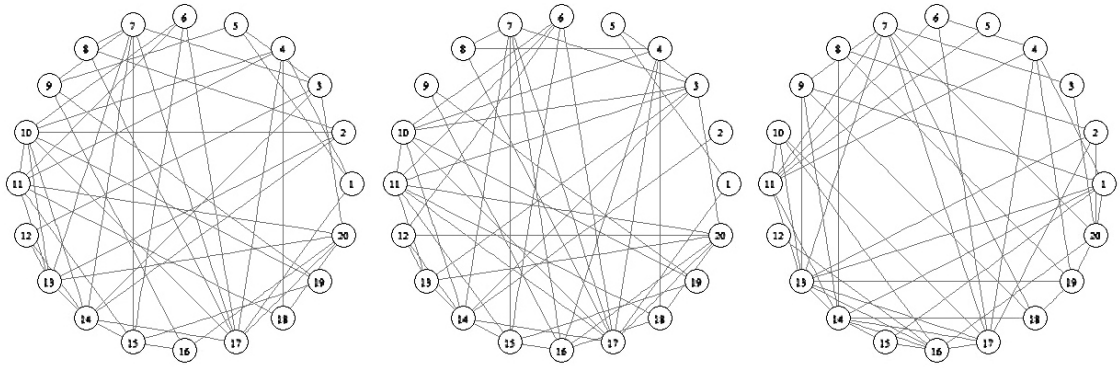


Figure 4.6: *Left:* Base graph $G = (V, E)$ with $|V| = 20$ nodes and $|E| = 50$ edges.*Middle:* Random graph G^1 built from G by swapping 10 edges. *Right:* Random graph G^2 built from G by swapping 50 edges. Distance: L2. Seed: 1000

Extensions

Although the normalization methods proposed in Section 4.2 have bounded the individual graph summarization metrics between 0 and 1, these metrics are not necessarily uniformly distributed between 0 and 1. An alternative method of normalization utilizes the Erdos-Renyi model (which generates a random graph by creating every edge with the same probability) to normalize the *overall difference metric* between two existing graphs $G^1 = (V, E^1)$ and $G^2 = (V, E^2)$. While this does not actually yield a scalar difference metric, the result provides insight on the significance of the resulting difference metric. Let $d = \text{diff}(G^1, G^2)$. Then the procedure is as follows:

Algorithm 8 Alternative graph difference normalization procedure

1: **procedure** ($G^1 = (V, E^1)$ and $G^2 = (V, E^2)$. Let $d = \text{diff}(G^1, G^2)$. n is the desired number of trials e.g. $n = 1000$.)
2: **loop from** $i = 1$ **to** n:
3: Generate an Erdos-Renyi graph $\hat{G}^1$ with $|V|$ nodes and $|E^1|$ edges.
4: Generate an Erdos-Renyi graph $\hat{G}^2$ with $|V|$ nodes and $|E^2|$ edges.
5: $t_i \leftarrow \text{dist}(\hat{G}^1, \hat{G}^2)$
6: Compare d to the empirical distribution $\{t_1, ..., t_n\}$*

*Compare d to the empirical distribution $\{t_1, ..., t_n\}$ by computing a p-value

$$\frac{\#\{t_i : t_i \geq d\}}{n}$$

for an upper tail distribution and a p-value

$$\frac{\#\{t_i : t_i \leq d\}}{n}$$

for a lower tail distribution. The p-values may be computed using the similarity selection methodology presented in Section 4.3. A lower p-value suggests that d is more significant (i.e. the computed difference metric is reliable) because it indicates that d has a low probability of being an "extreme" event.

Chapter 5

Financial application

In this chapter, the visualization system is used to help select a "buy and hold" portfolio of healthcare stocks. Section 5.1 presents the data, Section 5.2 presents a simple stock selection methodology given a correlation graph $G = (V, E)$ (as defined in Section 1.2), Section 5.3 outlines the application procedure, and Section 5.4 presents the results.

5.1 Data

The data set contains 4110 price observations of 43 stocks from the healthcare industry from 1998 to 2014. The data is cleaned to get rid of the dependency on time and yearly seasonality so that each observation may better simulate an independent draw from some distribution. This is done with the `sstl` function from the `Rsafd` package; after fitting the seasonal and trend components, the residuals are used as the new dataset. A list of tickers and their associated firms may be found in Table 5.1.

Table 5.1: Healthcare firms and tickers.

Ticker	**Firm**
ABT	ABBOTT LABORATORIES
ABBV	ABBVIE INC
ACT	ACTAVIS PLC
AET	AETNA INC
A	AGILENT TECHNOLOGIES INC
ALXN	ALEXION PHARMACEUTICALS INC
AGN	ALLERGAN INC
ABC	AMERISOURCEBERGEN CORP
AMGN	AMGEN INC
BAX	BAXTER INTERNATIONAL INC
BDX	BECTON DICKINSON AND CO
BIIB	BIOGEN IDEC INC
BSX	BOSTON SCIENTIFIC CORP
BMY	BRISTOL-MYERS SQUIBB CO
CAH	CARDINAL HEALTH INC
CFN	CAREFUSION CORP
CELG	CELGENE CORP
CERN	CERNER CORP
CI	CIGNA CORP
COV	COVIDIEN PLC
BCR	CR BARD INC
DVA	DAVITA HEALTHCARE PARTNERS I
XRAY	DENTSPLY INTERNATIONAL INC
EW	EDWARDS LIFESCIENCES CORP
LLY	ELI LILLY & CO
ESRX	EXPRESS SCRIPTS HOLDING CO
FRX	FOREST LABORATORIES INC
GILD	GILEAD SCIENCES INC
HSP	HOSPIRA INC
HUM	HUMANA INC
ISRG	INTUITIVE SURGICAL INC
JNJ	JOHNSON & JOHNSON
LH	LABORATORY CRP OF AMER HLDGS
MCK	MCKESSON CORP
MDT	MEDTRONIC INC
MRK	MERCK & CO. INC.
MYL	MYLAN INC
PDCO	PATTERSON COS INC
PKI	PERKINELMER INC
PRGO	PERRIGO CO PLC

(Continued on next page)

Table 5.1: (continued)

Ticker	Firm
PFE	PFIZER INC
DGX	QUEST DIAGNOSTICS INC
REGN	REGENERON PHARMACEUTICALS
STJ	ST JUDE MEDICAL INC
SYK	STRYKER CORP
THC	TENET HEALTHCARE CORP
TMO	THERMO FISHER SCIENTIFIC INC
UNH	UNITEDHEALTH GROUP INC
VAR	VARIAN MEDICAL SYSTEMS INC
VRTX	VERTEX PHARMACEUTICALS INC
WAT	WATERS CORP
WLP	WELLPOINT INC
ZMH	ZIMMER HOLDINGS INC
ZTS	ZOETIS INC

The data has been split into testing and training sets of equal length (2050 observations each) between 1998 and 2014. The idea is that we are standing in the middle (at 3/13/2006 to be precise) and wish to select a healthcare portfolio on that day. Although the future is uncertain, we do have the historical price performance of each stock and may also extract qualities such as drift, volatility, returns, etc. from the data. As such, correlation graphs (see Section 1.2) may be computed from the data and used to inform the portfolio selection. Once the portfolio has been selected and purchased, the "buy and hold" model is implemented (no rebalancing occurs), and we observe the yearly returns until 2014. The next section details the stock selection procedure given a correlation graph.

5.2 Stock selection methodology

Given a correlation graph $G = (V, E)$, we wish to select $k < |V|$ stocks (represented as nodes in G) such that each stock is as independent as possible of all other stocks in the set. Two random variables X and Y are independent if and only if their joint

cumulative distribution function may be written as $F_{X,Y}(x,y) = F_X(x)F_Y(y)$. With this formal definition, Santos *et al.* notes that "we say two random variables X and Y are dependent if they are not independent" so the problem may be transformed into that of "how to measure and detect dependence from the observation of the two random variables"[17]. In general, it is incorrect to say that "uncorrelated" equates "independence", but correlation is still a useful tool to inform a portfolio stock selection procedure (see Section 1.3). Thus, the procedure must select k stocks such that each stock is as *uncorrelated* as possible with all other stocks in the set.

In a correlation graph, an existent edge indicates some form of correlation (visual or numerical) while a non-existent edge indicates an uncorrelated relationship and is a proxy for independence (see Section 1.2). Subsequently, we might consider the following simple selection strategy that finds the set(s) of k stocks with *a minimum number of edges* between all pairs in the set:

Algorithm 9 Simple stock selection strategy

procedure (k is the number of stocks to select and A is the adjacency matrix of the correlation graph $G = (V, E)$)
    $z \leftarrow \text{Comb}(|V|, k)$*
    $min \leftarrow \infty$
    $index \leftarrow 0$
    loop from $i = 1$ **to** $\text{len}(z) = \binom{|V|}{k}$:
        $c \leftarrow 0$ (the number of connections)
        loop from $j = 1$ **to** $k - 1$:
            loop from $l = j + 1$ **to** k:
                $c \leftarrow c + A_{z_{j,i}, z_{l,i}}$
        If $c < min$ **then**
            $min \leftarrow c$
            $index \leftarrow i$
        Else If $c == min$ **then**
            Randomly determine if replacing min and $index$ or not
    return $z_{,index}$

*Comb$(|V|, k)$ is a function that returns a matrix of all possible combinations of k values selected from $|V|$ e.g. the vector $z_{,i} = < z_{1,i} = 1, z_{2,i} = 5, z_{3,i} = 7 >$ is a combination for $|V| = 10, k = 3$.

This algorithm becomes computationally difficult as $|V|$ increases and does not account for other qualities of the data; as discussed in Section 1.3, stocks with positive drift and high volatility are shown to improve the performance of the portfolio over time in the "buy and hold" model, or improve gains in the case of rebalancing. These two criteria may be used to minimize the space of stocks to select from. Drift over time is simplified as the average price difference between any two consecutive points in time. We let p_d and p_v denote the threshold ratios (with respect to averages) for drift and volatility respectively. These values will be dependent on the size of the data set as we wish to retroactively select p_d and p_v such that the number of viable stocks is reduced to a reasonable amount (such as 20 - 30) to reduce the computational burden but not reduced so much that the portfolios all end up being the same (which would be the case if the remaining number of stocks was close to k). The adjusted selection strategy may be found in Algorithm 10, and the corresponding code may be found in Appendix A.13.

Although the revised selection procedure is still simplistic, it is certainly better than randomly selecting stocks and actually holds up in performance against a portfolio fully invested in the S&P 500 (This result may be seen in Section 5.4). Furthermore, we will see that this procedure also fulfills the purpose of demonstrating the viability of the VS when applied in a setting such as finance.

It is also important to make a note here about the creation of the numerical correlation graphs for the eventual purpose of portfolio selection. Because the goal is to find pairs *without* edges, it is more useful to have a graph with *more* edges so that there are less pairs *without* edges; as such, pairs *without* edges may better represent a more "pure" idea of non-correlation, making them more informative about the relationship between the corresponding stocks. Furthermore, having less pairs *without* edges would make them simpler to sort through and interpret. As V_i and

V_j have *no edge* when $\text{Corr}(i, j)$ is below some threshold t, we set a low threshold of $t = 0.15$ for creating the numerical correlation graphs.

Algorithm 10 Simple stock selection strategy (adjusted)

procedure (k is the number of stocks to select and A is the adjacency matrix of correlation graph $G = (V, E)$. Let D be the vector of sample average price differences over time for each stock in A. Let V be the vector of sample average standard deviations for the price of each stock in A.)
    function DRIFT
        $rmv = [\,]$
        loop from $i = 1$ to $\text{len}(A)$:
            If $D_i \leq \text{Avg}(D) * p_d$ **then** rmv.append(i)
        Delete all indices in rmv from the rows and columns of A
    function VOLATILITY
        $rmv = [\,]$
        loop from $i = 1$ to $\text{len}(A)$:
            If $V_i \leq \text{Avg}(V) * p_v$ **then** rmv.append(i)
        Delete all indices in rmv from the rows and columns of A
    function SELECTION
        $z \leftarrow \text{Comb}(|V|, k)$*
        $min \leftarrow \infty$
        $index \leftarrow 0$
        loop from $i = 1$ **to** $\text{len}(z) = \binom{|V|}{k}$:
            $c \leftarrow 0$ (the number of connections)
            loop from $j = 1$ **to** $k - 1$:
                loop from $l = j + 1$ **to** k:
                    $c \leftarrow c + A_{z_{j,i}, z_{l,i}}$
            If $c < min$ **then**
                $min \leftarrow c$
                $index \leftarrow i$
            Else If $c == min$ **then**
                Randomly determine if replacing min and $index$ or not
        return $z_{,index}$

*$\text{Comb}(|V|, k)$ is a function that returns a matrix of all possible combinations of k values selected from $|V|$ e.g. the vector $z_{,i} = < z_{1,i} = 1, z_{2,i} = 5, z_{3,i} = 7 >$ is a combination for $|V| = 10, k = 3$.

5.3 Procedure

Now the issue is that there are many numerical correlation graphs which may be computed with different correlation coefficients; given the large data set, it is unclear which metric might be the best one. This is where the visualization system comes in and ties everything together. By learning the user's interests and automatically labeling all pairwise scatter plots as "visually correlated" or "not visually correlated", the VS outputs a visual correlation graph that may be used as a "sanity check" against the various numerical correlation graphs and provide a guide for selecting the most visually intuitive numerical correlation graph.

Let X be a $n \times d$ data matrix where there are d observations of n variables (Specifically, $n = 43$ and $d = 2050$ in the healthcare data set). Recall that the data used for active learning classification by the VS is actually composed of all observations of characteristic features (as described in Section 2.1.2) for all $\binom{n}{2} = n(n-1)/2 = 903$ pairwise scatter plots of the n variables from the actual data set X. The final procedure is as follows:

1. With threshold $t = 0.15$, create four different numerical correlation graphs $\hat{G}^{i,\text{num}} = (V, E)$ where $|V| = n$ for all $i \in \{1, ..., 4\}$, one for each correlation coefficient described in Section 1.2 (Pearson's, Spearman's, Kendall's, and distance correlation).

2. Run the VS on the same data set. Stage 1 of the system utilizes the active learning algorithm selected in Chapter 3 (uncertainty sampling with a budget of $iter = 15$) to learn what is "visually correlated" and "not visually correlated." Stage 2 of the system uses the resulting classifier (built from a random forest classification model) to iterate through all unlabeled pairwise scatter plots and returns a visual graph $\hat{G} = (V, E)$ built from the following heuristic:

$E_{i,j} = 1$ for $i \neq j$ if the pairwise scatter plot of (X_i, X_j) is "visually correlated" (as determined by the learned stage 2 labels)

---

3. Utilize the graph comparison methodology described in Chapter 4 to find the difference of each pair $(\hat{G}, \hat{G}^{i,\text{num}})$ for all $i \in \{1, ..., 4\}$. Select numerical graph $\hat{G}^*$ such that $\text{diff}(\hat{G}, \hat{G}^*) \leq \text{diff}(\hat{G}, \hat{G}^{i,\text{num}})$ for all $i \in \{1, ..., 4\}$. The numerical

correlation graph $\hat{G}^*$ is then the graph which best matches the visual interpretation of the relationships among the data set's variables.

4. Using the stock selection procedure described in Section 5.2, create P^i, a portfolio of $k = 5$ stocks, from the numerical correlation graph $\hat{G}^{i,\text{num}}$ for all $i \in \{1, ..., 4\}$ (including portfolio P^* from the numerical graph selected as $\hat{G}^*$). Each selected stock holds equal weight in the portfolio.

5. Observe the "buy and hold" performance of each portfolio until 2014.

It might also be sufficient to simply select P^* from $\hat{G}^*$, record the yearly returns, and then be done. However, because the purpose of this chapter is to demonstrate the usage and viability of the VS in aiding with the selection of an appropriate numerical model (given a data set and some application), it is necessary to compare the results of all options to each other. It is natural to expect that the portfolio P^* will perform the best as that is the purpose of the visualization system, after all. By comparing the results of P^* against the results of the portfolios created from the other numerical correlation graphs, we may determine just how well the system captures the user's intuition of visual dependence (which is likely to be far more complex than a single metric can capture). The final code for the application procedure may be found in Appendix A.14.

5.4 Results

The specific queries given by the VS and the corresponding oracle labels (the oracle, in this case, is the author of this work) may be seen in Figures 5.1 and 5.2. Recall that the active learner in stage 1 was given a budget of 15 queries. The queried plots are not labeled during stage 1 in order to prevent potential bias from clouding the oracle's responses. After stage 2, the user may plot and view the corresponding variable names should he/she wish to do so.

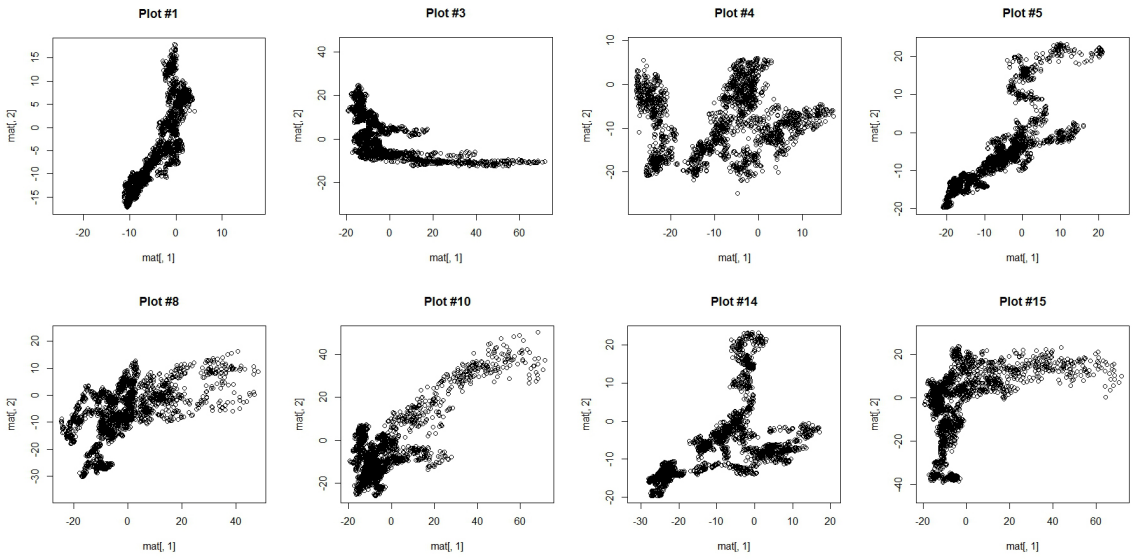


Figure 5.1: VS queries which were labeled "visually correlated".

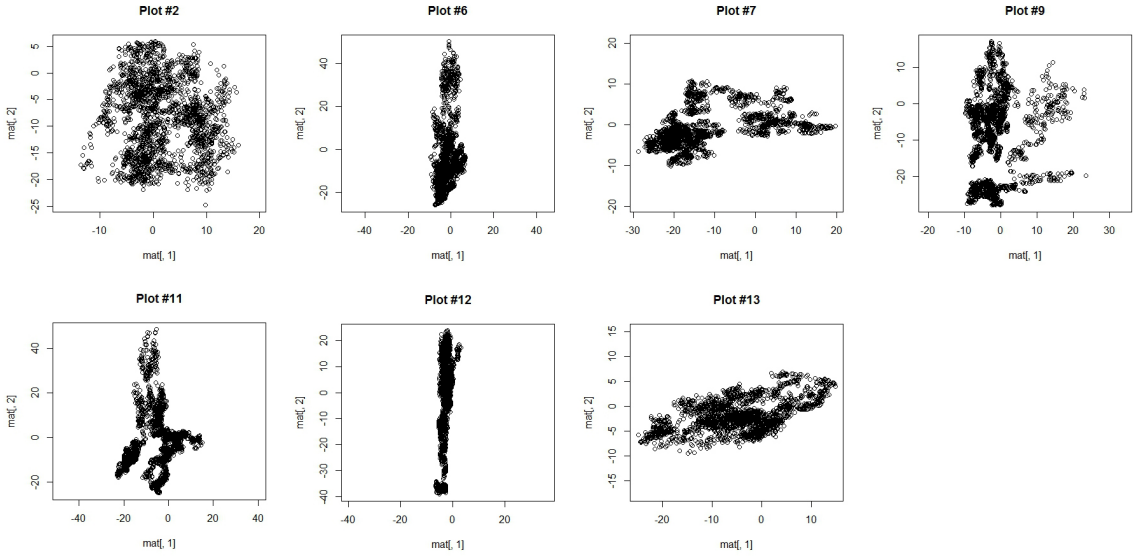


Figure 5.2: VS queries which were labeled "not visually correlated".

It is also interesting to examine the heat maps (which are normalized) that correspond to the 15 queries; these heat maps may be found in Figure 5.3, which also contains a list of associated criterion. A quick glance at the heat maps reveal that the middle criteria are all zero. It actually turns out that these criteria are associated

with various Pearson correlation tests. The zero-value heat map entries indicate that the associated p-values are all extremely low (the correlation is significantly different from zero). In particular, this indicates that the criterion 9, 10, 11, 12, 13, and 14 are all significant. Consider the actual variable pairs associated with these criteria. The first row corresponds to the first queried plot which received a label of "not visually interesting" (It is the top left plot depicted in Figure 5.2). Specifically, the pair has a Pearson correlation coefficient of -0.069 and p-value of 0.001728. The Pearson correlation coefficient is actually rather close to 0 (uncorrelated) despite the p-value suggesting otherwise. When looking at the metric numerically as the heat map does, the natural response would be to draw an edge as in the p-value heuristic given in Section 1.2. However, visually examining the pair's scatter plot (and even just looking at the correlation coefficient without the p-value) makes it clear that the pair is uncorrelated. How did this happen? Because there are 2050 observations of each stock, the variance, and subsequently the p-value, is low. If the correlation graph was constructed solely based on the Pearson correlation coefficient p-value without consideration of other factors, the result would have been less than ideal. This is another scenario that demonstrates the usefulness of the VS.

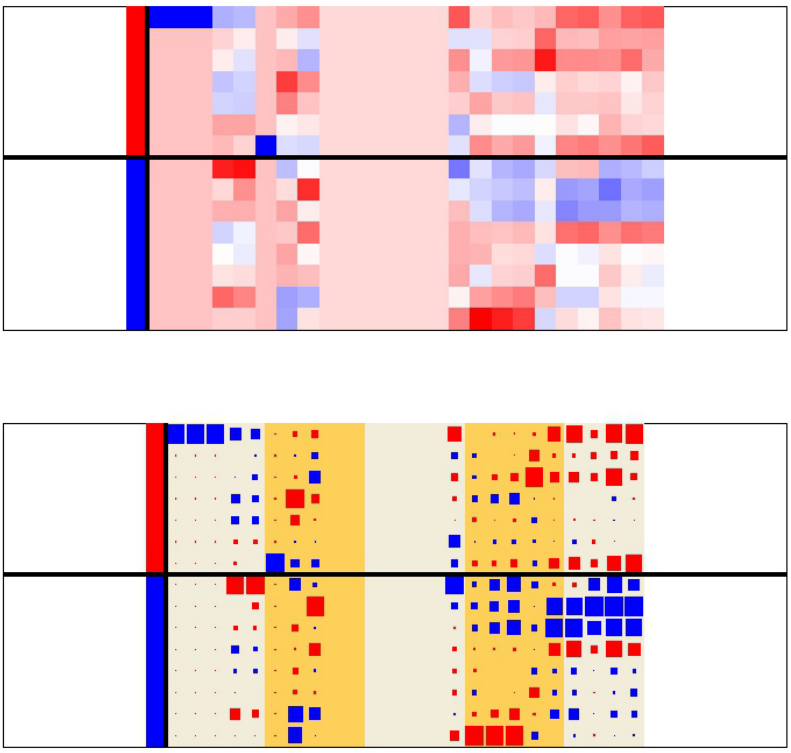


Figure 5.3: *Top:* Standard heat map of the queries and given labels (normalized). *Bottom:* Revised heat map (the association navigator) as proposed by Buja *et al.* [2] (also normalized). Each row corresponds to a pairwise scatter plot queried by the VS in stage 1. Each scatter plot is ordered by the oracle's responses; red indicates "not visually correlated" and blue indicates "visually correlated" (as dictated by the oracle). Each column corresponds to a different criterion of the associated pairwise scatterplot (see Section 2.1). Ordered criterion list (left to right, up to down):

"criterion_correlation_pearson_pval"	"criterion_correlation_kendall_pval"	"criterion_correlation_spearman_pval"
"criterion_bottomLeftTri"	"criterion_negativeTrend"	"criterion_chiSquare_pval"
"criterion_middleBox"	"criterion_upperRightTri"	"criterion_energy_pval"
"criterion_energy_test"	"criterion_correlation_pearson_test"	"criterion_correlation_spearman_test"
"criterion_chiSquare_test"	"criterion_correlation_kendall_test"	"criterion_cluster"
"criterion_upperLeftTri"	"criterion_positiveTrend"	"criterion_trend"
"criterion_bottomRightTri"	"criterion_correlation_kendall"	"criterion_correlation_spearman"
"criterion_KLdiv_diagonalGaussian"	"criterion_correlation_pearson"	"criterion_energy"

Figure 5.4 is a predicted probability histogram. The probability of being "visually interesting" or not is computed for 100 random pairwise scatter plots. Each probability is computed using the final classifier output from stages 1 and 2 of the VS. Most of the pairs fall within two bins (low and high probability). This can be observed as the two upper and lower hills in the histogram, indicating that the final classifier determined by the VS is quite confident in its labeling of most of the scatter plots.

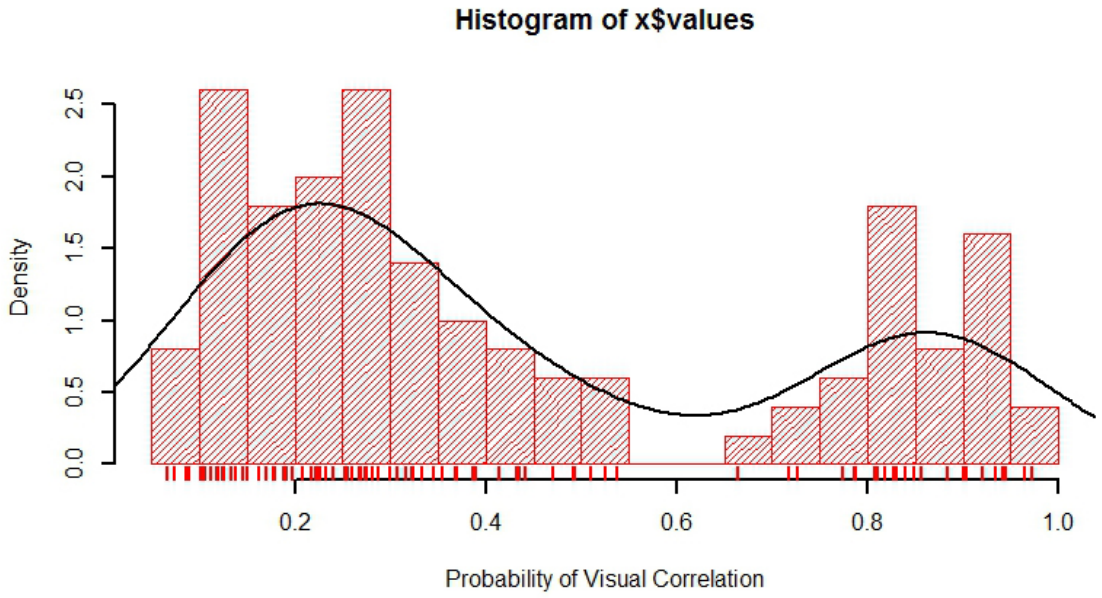


Figure 5.4: The histogram shows the predicted probabilities for 100 pairwise scatter plots. The probability corresponds to the probability of a plot being "visually correlated" or "not visually correlated".

The final resulting visual correlation graph is shown in Figure 5.5.

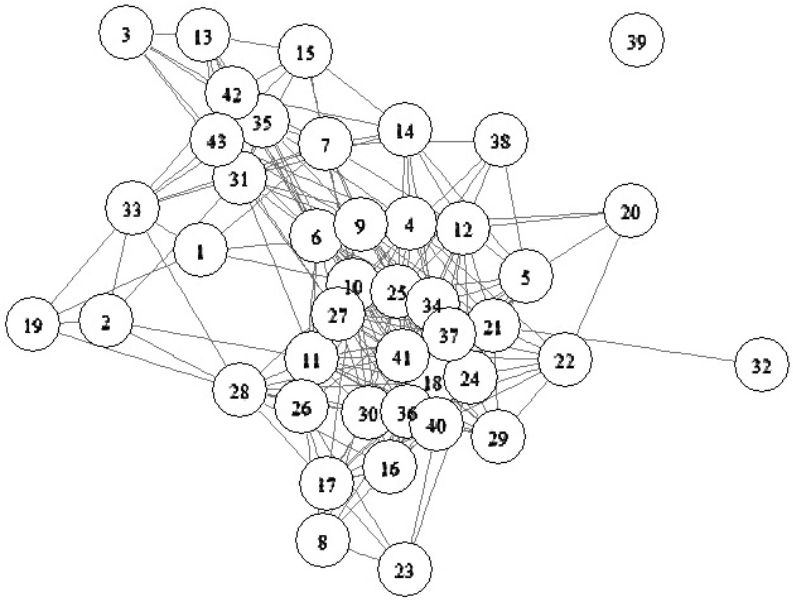


Figure 5.5: Visual correlation graph output from the visualization system.

The graph comparison procedure determined that the **distance correlation graph** is closest to the visual correlation graph. The computed graph differences may be found in Table 5.2. The resulting portfolios selected by the procedure in Section 5.2 for each numerical correlation graph are summarized in Table 5.3.

Finally, the the "buy and hold" performance (yearly returns) and cumulative sum of returns are plotted for the S&P 500 (the control) and all portfolios P^i for $i \in \{1, ..., 4\}$ may be found in Figure 5.6. Each portfolio P^i was created from its corresponding numerical correlation graph $\hat{G}^{i,\text{num}}$ and includes P^*, which is created from $\hat{G}^* = \hat{G}^{4,\text{num}}$ (which corresponds to the distance correlation graph). The associated data may be found in Table 5.3. Note that the returns for 2006 and 2014 are low because the data begins on 3/14/2006 and ends on 5/13/2004; as such, the returns for 2006 and 2014 do not reflect a full year's worth of data. Our focus is mostly

diverted to the full years in between and especially the portfolio performance during the financial crisis.

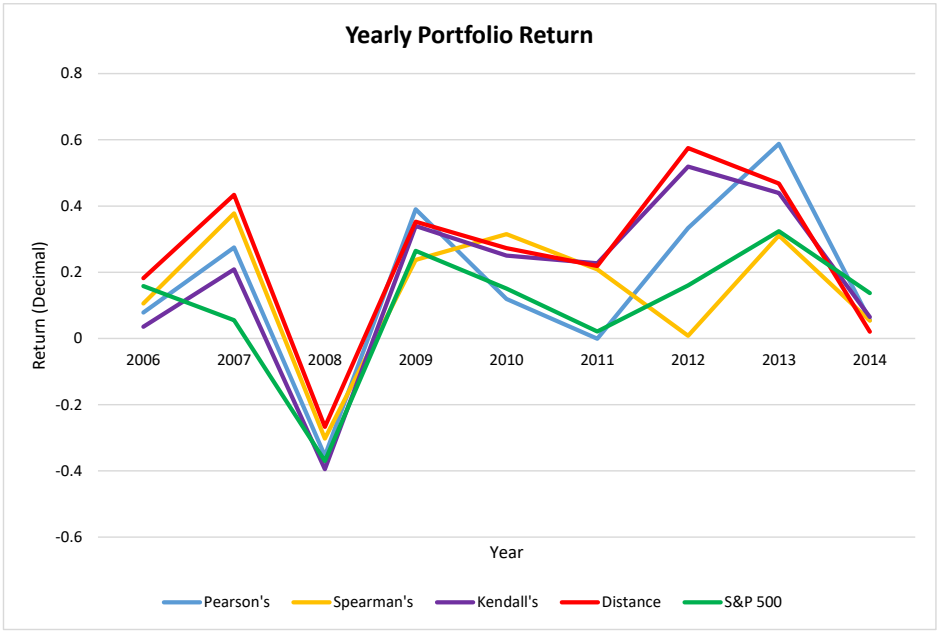
The portfolio P^*, constructed from the numerical distance correlation graph $\hat{G}^*$ that is most similar to the visual correlation graph $\hat{G}$, performs the strongest by far. P^* consistently posts the highest returns year after year. Though it doesn't seem to strongly outperform on a yearly basis, the differences really add up as can be seen in cumulative sum of yearly returns. Further consider the financial crisis; P^* has the least negative returns of 2008. This indicates that the visual graph is capturing the independence and dependence among various stocks properly, which translates into the selection of $\hat{G}^*$ and, subsequently, P^*, a portfolio that is well balanced and captures the spirit of the "buy and hold" model described in Section 1.3. As noted earlier in Section 5.2, the proposed stock selection strategy is simple, but every selected portfolio has outperformed the S&P 500, suggesting that an even more sophisticated selection procedure would yield even more fruitful returns.

Extensions

It should also be noted that, while a rebalancing method performs better in practice [12], the "buy and hold" strategy is better-suited for observing portfolio performances over time with less external influences, providing a more concrete basis of comparison for $\hat{G}^*$. However, the procedure described in Section 5.3 can certainly be repeated yearly (once a year's worth of new data has been collected) to determine the direction in which to rebalance the portfolio. Furthermore, the weights of the stocks in the portfolio can certainly be optimized; currently, each stock holds equal weight within its portfolio. All of these ideas are potential extensions that may improve upon portfolio management techniques that this application does not utilize.

Table 5.2: Computed differences between visual and numerical correlation graphs. See Section 4.2 for details on each graph summarization metric.

Graph Type	**Centrality (degree)**	**Centrality (closeness)**	**Centrality (betweenness)**	**Assortativity**	**Distance matrix**
Pearson's	0.0034	0.0119	0.0032	0.061	0.0313
Spearman's	0.0052	0.0079	0.0032	0.0773	0.029
Kendall's	0.0002	0.019	0.0024	0.0559	0.0253
Distance	0.0018	0.0276	0.0012	0.0465	0.0206
	Community (random walk)	**Community (infomap)**	**Community (betweenness)**	**Edge connectivity**	**Edge density histogram**
Pearson's	0.7602	0.5039	0.6488	1	0.6014
Spearman's	0.8478	0.5039	0.6948	1	0.5625
Kendall's	1	0.5039	0.7284	1	0.4413
Distance	0.253	0.5039	0.5437	1	0.2455



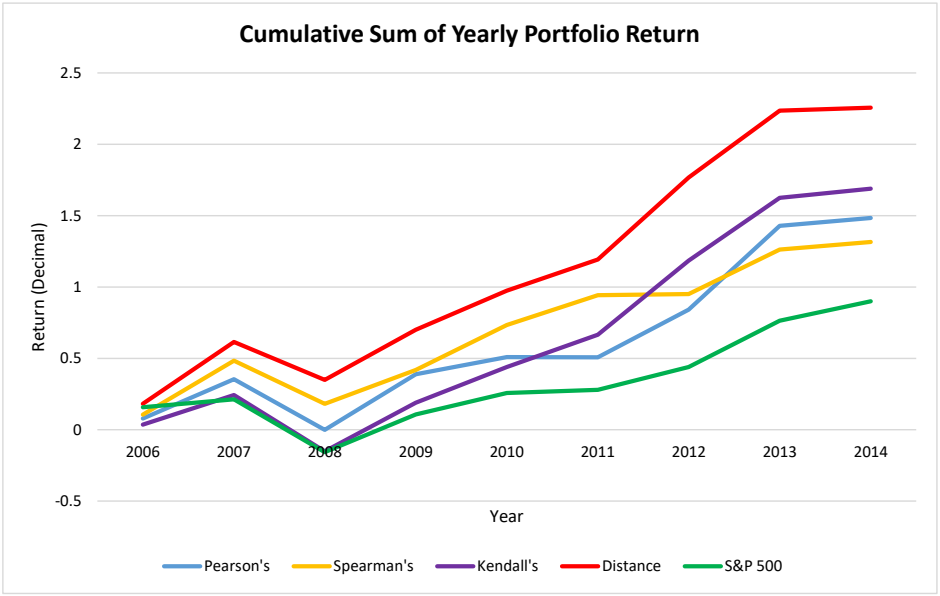


Figure 5.6: Yearly and cumulative performance of the selected portfolios and the S&P 500. Note that the "yearly" returns for 2006 and 2014 are low because the data begins on 3/14/2006 and ends on 5/13/2004; as such, those returns do not reflect a full year of data. The associated data may be found in Table 5.3.

Table 5.3: Yearly returns (in decimal) of the portfolios selected from the numerical correlation graphs.

Graph Type	**Portfolio**	**2006**	**2007**	**2008**	**2009**	**2010**	**2011**	**2012**	**2013**	**2014**
S&P 500 (CONTROL)	**Total**	0.158	0.055	-0.37	0.265	0.151	0.021	0.16	0.324	0.137
Pearson's	CI	0.041	0.226	-0.686	1.099	0.041	0.147	0.274	0.637	0.011
	GILD	0.056	0.417	0.111	-0.154	-0.162	0.129	0.795	1.045	0.069
	SYK	0.15	0.362	-0.46	0.267	0.079	-0.061	0.121	0.393	0.073
	TMO	0.276	0.274	-0.409	0.4	0.161	-0.188	0.432	0.758	0.057
	VAR	-0.131	0.096	-0.328	0.337	0.479	-0.031	0.046	0.106	0.064
	Total	0.078	0.275	-0.354	0.39	0.119	-0.001	0.333	0.588	0.055
Spearman's	HUM	0.122	0.362	-0.505	0.177	0.247	0.615	-0.206	0.523	0.184
	LH	0.291	0.028	-0.147	0.162	0.175	-0.022	0.008	0.055	0.092
	PRGO	0.096	1.041	-0.072	0.243	0.597	0.542	0.073	0.479	-0.144
	SYK	0.15	0.362	-0.46	0.267	0.079	-0.061	0.121	0.393	0.073
	VAR	-0.131	0.096	-0.328	0.337	0.479	-0.031	0.046	0.106	0.064
	Total	0.106	0.378	-0.302	0.237	0.315	0.208	0.008	0.311	0.054
Kendall's	MCK	-0.039	0.297	-0.404	0.63	0.138	0.118	0.256	0.676	0.117
	REGN	0.231	0.203	-0.24	0.317	0.358	0.688	2.086	0.609	0.03
	SYK	0.15	0.362	-0.46	0.267	0.079	-0.061	0.121	0.393	0.073
	UNH	-0.035	0.084	-0.543	0.148	0.199	0.422	0.086	0.41	0.04
	VAR	-0.131	0.096	-0.328	0.337	0.479	-0.031	0.046	0.106	0.064
	Total	0.035	0.209	-0.395	0.34	0.25	0.227	0.519	0.439	0.065

(Continued on next page)

Table 5.3: (continued)

Graph Type	**Portfolio**	**2006**	**2007**	**2008**	**2009**	**2010**	**2011**	**2012**	**2013**	**2014**
Distance	JNJ	0.137	0.036	-0.078	0.113	-0.006	0.099	0.108	0.346	0.111
(Most similar to	PRGO	0.096	1.041	-0.072	0.243	0.597	0.542	0.073	0.479	-0.144
visual graph)	REGN	0.231	0.203	-0.24	0.317	0.358	0.688	2.086	0.609	0.03
	TMO	0.276	0.274	-0.409	0.4	0.161	-0.188	0.432	0.758	0.057
	WAT	0.169	0.615	-0.536	0.691	0.254	-0.047	0.177	0.148	0.048
	Total	0.182	0.434	-0.267	0.353	0.273	0.219	0.575	0.468	0.021

Chapter 6

Conclusion and extensions

6.1 Conclusion

The proliferation of high-dimensional datasets has led analysts to rely on unverifiable numerical estimators. It would be useful to be able to construct high-dimensional visual correlation graphs as a way to verify numerical tests of dependence. Correlation graphs are important in portfolio selection as it is desirable to select stocks that are as uncorrelated as possible. However, high-dimensional visualization is problematic because there are too many potential plots to sort through manually; to be specific, there are $\binom{n}{2} = n(n-1)/2$ plots where n is the number of variables e.g. stocks that we are interested in. As a general solution to these issues, the visualization system actively learns user preferences regarding their notion of visual correlation, applies the fitted classifier to unlabeled data, provides visualization tools for the active learning output, and outputs the difference among the visual graph G and some given numerical graph G^{num} (Chapter 2).

In this work, we focused specifically on the active learning and graph comparison components of the VS. We reviewed the two main approaches to active learning (efficient search through hypothesis space $\mathcal{H}$ and exploiting clustering tendencies in

data) and provided algorithms for different active learning methods in Chapter 3. These methods include uncertainty sampling, query by committee (for which we proposed a revised framework), query by bagging, and min-max clustering. A simulation study with parameters that mimicked the intended qualities of the VS indicated that uncertainty sampling would be the ideal active learning method to be implemented in stage 1 of the VC for usage in the financial application of Chapter 5. Similarly, we discussed various graph summarization difference metrics, proposed a method to compare the distance metrics of various graph pairs in order to select the most similar pair, and demonstrated the viability of the proposed procedure in Chapter 4). The graph summarization difference and similarity selection are output functions of the VS that may be applied to any graphs and, subsequently, find a use in various other applications.

Our financial data contains the daily prices of 43 different healthcare stocks that were treated to get rid of time-dependency. The data is split in half with the first half (1998 to 2006) used as "training" input and the second half (2006 to 2014) used as "testing" output. Various correlation coefficients were applied to the training set to create $G^{i,\text{num}}$ for all $i \in \{1, ..., 4\}$, the numerical correlation graphs for Pearson's, Spearman's, Kendall's, and distance correlation coefficient respectively. We then ran the VS on the training set to determine the visual graph G. After computing the difference between each graph pair $(G^{i,\text{num}}, G)$, the similarity selection procedure presented in Section 4.3 was used to find G^*, the numerical graph $G^{i,\text{num}}$ that most closely matches the visual graph G. The algorithm selected $G^{4,\text{num}}$, the distance correlation graph, as G^*. Then the stock selection methodology described in Section 5.2 was used to select a portfolio P^i of $k = 5$ stocks for each corresponding correlation graph $G^{i,\text{num}}$. The testing set was used to compute yearly returns for each portfolio in a simulation of the "buy and hold" strategy. $P^* = P^4$, the distance correlation portfolio, is the consistent top performer, demonstrating the reliability and usefulness of the VS

in a concrete application. Furthermore, as can be seen in the results in Section 5.4, the entire process is transparent, and results are reproducible given the recorded user responses to the VS queries. Transparency and reproducibility are other important purposes of the VS which can certainly be explored further.

The visualization system presented in this work is an important step towards streamlining the future of clean analysis. It provides a systematic way for confirming and/or suggesting dependencies among variables that match the user's concept of visual dependence and produces an explicit decision tree that allow others to understand and replicate the data analysis process. The VS both alleviates the problems associated with high-dimensional data analysis and allows the user to quickly identify where the numerical model may have fallen short of the "true" relationship between variables. Nevertheless, there are several places to develop further work in order to refine the system and improve our concept of "clean analysis."

6.2 Further extensions

This section details further extensions on improving other aspects of the visualization system. For ideas on improving the specific work done in this paper, see the final "extensions" section in Chapter 3 for active learning, Chapter 4 for graph comparison, and Chapter 5 for portfolio management.

6.2.1 Estimator selection

Estimator selection involves actively fitting the "best" numerical model as opposed to passively "checking" a numerical model that's been given. For instance, rather than requiring a numerical model in order to compute the graph distance as in Chapter 4, the VS would select a numerical model without requiring the analyst to fit one beforehand. This problem is more difficult to solve in an objective manner, but such a

function would be extremely useful in contributing to the idea of an accountable and reproducible "clean analysis" procedure.

6.2.2 Outlier removal

Outliers are unavoidable in raw data and can skew results quite a bit. This can be seen in the analysis of Dataset 2 in Table 1.1 and Figure 1.1. The question to answer here would be: when can the system remove outliers, and what criteria should it use to make this decision? The procedure should be both systematic and transparent so that the analyst and future readers may understand what has(n't) been removed and for what reason. Data given to the VS should be locked once analysis is complete so as to prevent analysts from maliciously removing or adding outliers themselves to favorably skew the results if the output is not to their liking, thereby increasing accountability.

6.2.3 Graphical models and improved stock selection

A single numerical correlation coefficient can be thought of as a regression of the response variable against only one observed variable; it is a "local" property because it compares the behavior of only two random variables. Correlations tend to fall short of the desired result due to the property of transitivity.

Transitivity states that if X is correlated to Y, and Y to Z, then X is also correlated to Z. Although correlation is not always transitive, in situations where the correlations of various pairs are close to 0 or 1, the data will often strongly exhibit properties of transitivity [20]. Let us assume that the universe consists of Apple, Google, and Silicone (a manufacturer providing chips to both Apple and Google) stock. Suppose an analyst wants to model Google stock. Apple stock moves with Silicone stock as they depend on them for their chips. Similarly, Google stock (unbeknownst to the analyst) also moves with Silicone stock but not with Apple stock.

Without considering the way the stocks are connected to the other observed variable (Silicone stock), the correlation between observed prices of Google and Apple stock will be erroneously positive .

On the other hand, a single link in a graphical model can be thought of as a regression of the response variable against all variables in the space. The general idea is that a graphical model has a "global" property because it takes all of the other variables into account. Let $G = (V, E)$ be an undirected graph formed from the data set X with $|V| = n$ and edges $E_{i,j} \in \{0, 1\}$. $E_{i,j} = 1$ when there is an edge between V_i and V_j, and 0 otherwise. $E_{i,j} = 0$ if and only if $X_i \perp X_j$ given X_k where $k \in \{1, ..., d\}$ $\backslash\{i, j\}$. In other words, do not draw an edge if

$$\mathrm{P}(X_i, X_j|X_k) = \mathrm{P}(X_i|X_k)\mathrm{P}(X_j|X_k)$$

The resulting graph is known as a graphical model. The drawback is the difficulty in empirically computing conditional distributions and the problems associated with fitting distributions to real data. However, the conditional independence of graphical models is more of a global property than correlation is because every pair of variables is conditioned on all other variables. Returning to the universe of Google, Apple, and Silicone stocks, conditioning Google on Silicone and Apple on Silicone would indicate that Google and Apple are uncorrelated.

While correlation graphs and graphical models each have their own niche to fill, graphical model rebalancing has been shown to strongly outperform traditional portfolio management methods such as the "buy and hold" method employed in Chapter 5 [12]. It is important, therefore, for the VS to support both models. The system would need to be modified to allow for plotting X_i against X_j conditional on X_k for all $k \in \{1, ..., d\}\backslash\{i, j\}$. One method to consider would be to control for the behavior

of all the other explanatory variables while making the scatter plot of (X_i, X_j) in a procedure similar to that of regression.

6.2.4 Ordering of queried plots

As people interact with graphs, they maintain a "mental map" of the graph; when users label a new graph, they remember the previous plots that they have labeled [7]. The importance of the mental map in influencing user perception of the data depends on various factors such as the user preferences and tasks that they must complete [7]. Suppose that several active learning queries (scatter plots) are selected at once. Given a gradation of graphs (consecutive graphs that are similar to one another), users are less able to distinguish differences than if given consecutive graphs from different ends of the spectrum [7]. To put it concisely, the scatter plot display itself (discussed in Section 2.1.1) is not the only thing that matters. The ordering of the scatter plots also matter, and it is best to show the plots in an order that allows users to distinguish the differences among graphs that they have already seen. By improving user perception of the data, careful scatter plot ordering advances the accuracy of user responses. If user responses are to be thought of as observations of the user's true preferences, then the ordering of plots is another as a way to fine-tune the precision of the classifier.

6.2.5 Line-up tests

One of the pitfalls of data visualization is "apophenia," a phenomenon where the user sees patterns in random noise. Part of the reason this happens is due to the vagueness of defining "independence" on a non-uniform domain and range (see Section 2.1.1). Wickham *et al.* propose a line-up protocol that is similar to the Rorscach test in which subjects are asked to interpret abstract blots of ink [22]. In the line-up test, users are asked to identify the real data from a set of k plots where $k - 1$ plots are synthetically generated [22]. Identification of the raw data against the synthetic data

is an indicator that the current fitted model is a poor fit of the user's preferences. Suppose that $k = 5$ as in Figure 6.1. In the context of the classification problem, the line-up test may generate four "not visually correlated" plots (following the proposed decision tree) and one "visually correlated" plot before asking the user if he/she can identify the "visually correlated" plot. If the user is able to consistently identify the "visually correlated" plot, it is an indication that the current decision tree is a close fit of the user's preferences.

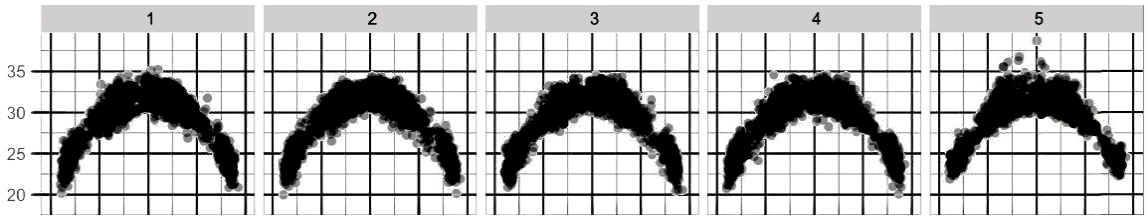


Figure 6.1: A line-up test for $k = 5$. Consistently identifying the raw data against the synthetic data indicates that the fitted model may not be good enough. Figure from Wickham *et al.* 2010 with slight modifications [22]

6.2.6 Edge-weighted graphs

Edge-weighted graphs stray from the binary classification of correlation graphs (described in Section 1.2) and graphical models (described in Section 6.2.3). In an edge-weighted graph, edges are weighted depending on the type of conditional dependence or correlation. Negative and positive values are assigned 1 and -1, respectively. As before, a value of 0 (no edge) implies that variables are uncorrelated (for correlation graphs) or conditionally independent (for graphical models). The active learning becomes more difficult as the number of classification labels changes from two to three. Subsequently, active learning methods may need to be more generalized as in the case of uncertainty sampling (Section 3.2.1).

6.2.7 Rejection classification

While interacting with the data visually, the user's own concept of "visual correlation" *in the specific data set* may evolve over time; at the beginning, the user has no idea what the data looks like and where to set the bar for their own standards of dependence. There are several ways to take this into consideration.

- The visualization system could assign a weight to the analyst's responses by trial number where the last few plots are more valuable than the first few. However, this may destabilize cases where the user's preferences don't end up changing.
- The system can include an alternative option that allows the user to refuse to label a plot when it's too close to their decision boundary. This is welcome for the user who is not forced into making a decision he/she is uncertain about, but it is problematic for the learner as it causes the hypothesis space to remain unchanged rather than shrink. The point of the active learning in stage 1 (Section 2.2, Chapter 3) is to have the user provide labels on plots the learner is uncertain about in order to better understand user preferences. By allowing for this option, the active learner may require more queries and/or return a poorly-defined tree.
- Aside from the current responses of "yes" and "no", the VS could include a third option that permits the user to "recycle" the plot. This method allows the user to return to the plot later after learning more about what the rest of the data looks like and understanding his/her own preferences better, and it also ensures that the active learner will eventually receive a label for all its queries. This method could potentially de-balance an ordering procedure (described in Section 6.2.4), but the VS can strategically account for both by inserting the recycled plot between two plots it is relatively different from with the constraint that the insertion location is after the current plot.

Appendix A

Implementation

A.1 Code for Figure 1.1, left

```
# Generate a reproducible dataset and scale to [0,1]
set.seed(10)
x <- seq(0, 1, length.out = 100)
y <- rnorm(100)
y <- (y-min(y))/(max(y)-min(y))

# Sort the noise
y <- sort(y)
y <- y[c(seq(1,99,length.out=50), seq(100,2,length.out=50))]

# Local swapping
for(i in 4:96){
        y[(i-3):(i+3)] <- y[sample((i-3):(i+3))]
}

idx <- sample(1:100)
x <- x[idx]; y <- y[idx]

###################### Numerical feedback

## Fit linear regression
fitlm <- lm(y ~ x)
anova(fitlm)

## See if any coefficients are significant
summary(fitlm)

## See if residuals are normally-distributed
shapiro.test(fitlm$residuals)

## Correlation is not significantly different from zero
```

```
cor.test(x, y)

##################### Visual feedback
plot(x, y, pch = 16, cex = 2)
```

A.2 Code for Figure 1.1, right

```
## Generate a reproducible dataset
set.seed(10)
n <- 50
x <- sort(rnorm(n))
sd.vec <- c(seq(1, 1.5, length.out = 50), seq(1.5, 1, length.out =
   50))
y <- -x + 0.5*rnorm(n, sd = sd.vec)
y <- scale(y)

y[c(1,5,10)] <- min(y)
y[c(n-10, n-5, n)] <- max(y)

##################### Numerical feedback

## Fit linear regression
fitlm <- lm(y ~ x)
anova(fitlm)

## See if any coefficients are significant
summary(fitlm)

## See if residuals are normally-distributed
shapiro.test(fitlm$residuals)

## Correlation is not significantly different from zero
cor.test(x, y)

##################### Visual feedback
plot(x,y, pch = 16, cex = 2)
```

A.3 Code for Figure 2.2

```
# Generate the dataset
set.seed(10)
n <- 500
x <- rnorm(n)
y <- rnorm(n)

# Plot data and apply CDF
par(mfrow=c(1,2))
plot(x,y, pch = 16, cex = 2)
plot(pnorm(x),pnorm(y),pch = 16, cex = 2)
```

A.4 Uncertainty sampling

Refer to Algorithm 1.

```
#' Uncertainty Sampling with bivariate labels
#'
#' @param X the full data matrix, n x d, including all unlabeled
    data
#' @param y a factor vector with 2 levels and NAs for unlabeled data
#' @param unlabel_index_c is a vector of n pre-selected (pooled)
    indices
#' @param classifier the classifier name
#' @param ... additional parameters for the active learning method
#'
#' @return an index to query
#' @export

uncertainty_sample <- function(X, y, unlabel_index_c, classifier,
        isR = FALSE, tout = NULL, ...){

        if (length(classifier) > 1 || missing(classifier) || is.null
           (classifier) ||
        is.na(classifier)) {
                stop("A␣single␣classifier␣is␣required␣for␣
                   uncertainty␣sampling")
        }
        if (isR & is.null(tout)) {
                stop("Re-feed␣classifier_method␣return␣to␣next␣
                   uncertainty_sample␣call")
        }

        # Check that the classifier is compatible with uncertainty
           sampling
        c <- try(caret::modelLookup(classifier))
        if (!any(c$probModel)) {
                stop(classifier,"␣must␣return␣posterior␣
                   probabilities")
        }

        # Split X and y to retrieve labeled and unlabeled pairs
        unlabel_index <- which(is.na(y))
        x_lab <- X[-unlabel_index,]
        y_lab <- y[-unlabel_index]
        x_ulab <- X[unlabel_index_c,]

        if (!isR) {
                tout <- caret::train(x_lab,y_lab,classifier)
                p <- as.matrix(stats::predict(tout, newdata=x_ulab,
                   type="prob"))
        } else {
                # Reuse the trained classifier from the classifier_
                   method call
```

```
                # Of course, this only works since classifier = "rf
                    ", and the
                # classifier_method function also uses "rf"
                p <- as.matrix(stats::predict(tout, newdata=x_ulab,
                    type="prob"))
        }

        # Return corresponding X index of posterior closest to 0.5
        p <- apply(p, 1, function(x) abs(x[1]-0.5))
        index <- unlabel_index_c[which(p == min(p))]
        if (length(index) > 1) index <- sample(index,1)
        index
}
```

A.5 Query by committee

Refer to Algorithm 3. This implementation contains the functions for query selection and pruning. These functions are called by the main simulation engine (Appendix A.9.2) for the QBC method. The simulation engine is coded in such as way that it acts as the skeleton of the entire algorithm in Section 3.2.2.

A.5.1 Query selection

```
#' Query by Committee
#'
#' @param X the full data matrix, n x d, including all unlabeled
    data
#' @param y a factor vector with 2 levels and NAs for unlabeled data
#' @param unlabel_index_c is a vector of n pre-selected (pooled)
    indices
#' @param committee the list of committee classifiers
#' @param dis is the disagreement measure between committee
    classifications
#' @param isMajority is if overall classifier Majority Vote or
    Random Forest
#' @param tout is a list of trained classifiers from Majority Vote
    computation
#' @param ... additional parameters for the active learning method
#'
#' @return a list with: an index to query AND committee predictions
#' @export

qbc_sample <- function(X, y, unlabel_index_c, committee,
        dis = "vote_entropy", isMajority = FALSE, tout= NULL, ...){
```

```
        if (missing(committee)||is.null(committee)) stop("A␣
            committee␣is␣required")
        if (isMajority & is.null(tout)) {
                stop("Re-feed␣the␣majority␣vote␣return␣to␣the␣next␣
                    QBC_sample␣call")
        }

        unlabel_index <- which(is.na(y))
        x_lab <- X[-unlabel_index,]
        y_lab <- y[-unlabel_index]
        x_ulab <- X[unlabel_index_c,]
        p <- vector("list",length(committee))

        if (!isMajority) {
                for (i in 1:length(committee)) {
                        tout <- caret::train(x_lab,y_lab,committee[i
                            ])
                        p[[i]] <- predict(tout, newdata=x_ulab)
                }
        } else {
                # Reuse the trained classifiers from the majority
                    vote call
                for (i in 1:length(committee)) {
                        p[[i]] <- predict(tout[[i]], newdata=x_ulab)
                }
        }

        # Compute disagreement (functions from the activelearning
            package)
        d <- switch(dis,
                vote_entropy=vote_entropy(p),
                post_entropy=post_entropy(p),
                kullback=kullback(p)
                )

        index <- unlabel_index_c[which(d == max(d))]
        if (length(index) > 1) index <- sample(index,1)
        # Gather each committee's prediction
        pre <- rep(0,length(committee))
        for (i in 1:length(committee)) {
                # Predict function returns a factor
                pre[i] <-
                as.numeric(as.character(p[[i]][which(unlabel_index_c
                    ==index)]))
        }

        list(index, pre)
}
```

A.5.2 Committee pruning

```
#' Query by Committee (committee pruning function)
#'
#' @param X the full data matrix, n x d, including all unlabeled
    data
#' @param y a factor vector with 2 levels and NAs for unlabeled data
#' @param index is the classification of X[index,] which was queried
#' @param committee_pred is the list of committee predictions for
    index
#' @param k is the current iteration number that the AL_engine is on
#' @param pt is the pruning threshold (any error value above it is
    pruned)
#' @param err in (0 best,1 worst) is the committee's error-to-
    iteration ratio
#' @param is_prune is TRUE when pruning is desired, FALSE when not
#' @param ... additional parameters for the active learning method
#'
#' @return a list with: updated error AND indices to delete from the
     committee
#' @export

qbc_prune <- function(X,y,index,committee_pred,k,pt = 0.5,err,is_
    prune,...){

        if (missing(err) || is.null(err) || is.na(err)) {
                stop("Committee error ratio is required for QBC
                    pruning")
        }
        prune <- vector() # Do not know how long prune will be until
             the end
        # Do not prune if committee size is 1 or if it's the first
            round
        if (length(committee_pred) == 1 | k == 1) {
                list(err, prune)
        } else {
                # Update error value
                for (i in 1:length(committee_pred)) {
                        if (committee_pred[i] == y[index]) iv <- 0
                            else iv <- 1
                        err[i] <- err[i] + (iv - err[i])/k
                        if (err[i] > pt & is_prune) {
                                prune <- c(prune,i)
                        }
                }
                list(err, prune)
        }
}
```

A.6 Vote entropy

Refer to Section 3.2.2.

```
#' Disagreement method (from activelearning package)
```

```
#' @importFrom itertools2 izip
#' @importFrom entropy entropy

vote_entropy <- function(x, type='class', entropy_method='ML'){

        it <- do.call(itertools2::izip, x)
        disagreement <- sapply(it, function(obs) {
                entropy::entropy(table(unlist(obs)), method=entropy_
                   method)
        })
        disagreement
}
```

A.7 Query by bagging

Refer to Algorithm 4.

```
#' Query by Bagging
#'
#' @param X the full data matrix, n x d, including all unlabeled
   data
#' @param y a factor vector with 2 levels and NAs for unlabeled data
#' @param unlabel_index_c is a vector of n pre-selected (pooled)
   indices
#' @param classifier the name of a classification model
#' @param dis is the disagreement measure between committee
   classifications
#' @param num_class is the number of desired committee members
#' @param r in (0,1). r*(labeled set) = training set for each num_
   class round
#' @param ... additional parameters for the active learning method
#'
#' @return an index to query
#' @export

qbb_sample <- function(X, y, unlabel_index_c, classifier,
        dis = "vote_entropy", num_class, r, ...){

        if(r<=0 || r>=1) stop("r must be in (0,1)")

        x_ulab <- X[unlabel_index_c,]

        # Randomly sample from the labeled set to create a
           classifier
        label_index <- which(!is.na(y))
        committee <- vector("list",num_class)
        for (i in 1:num_class) {
                idx <- sample(label_index,round(length(label_index)*
                   r,0))
```

```
                committee[[i]] <- caret::train(X[idx,],y[idx],
                    classifier)
        }

        # Utilize the resulting classifiers as a committee
        p <- vector("list",length(committee))
        for (i in 1:length(committee)) {
                p[[i]] <- stats::predict(committee[[i]], x_ulab)
        }

        # Compute disagreement (functions from the activelearning
            package)
        d <- switch(dis,
                vote_entropy=vote_entropy(p),
                post_entropy=post_entropy(p),
                kullback=kullback(p)
                )

        index <- unlabel_index_c[which(d == max(d))]
        if (length(index) > 1) index <- sample(index,1)
        index
}
```

A.8 Min-max clustering

Refer to Algorithm 5.

```
#' Min-Max Clustering
#'
#' @param X the full data matrix, n x d, including all unlabeled
    data
#' @param y a factor vector with 2 levels and NAs for unlabeled data
#' @param unlabel_index_c is a vector of n pre-selected (pooled)
    indices
#' @param dis is the distance measure between data
#' @param ... additional parameters for the active learning method
#'
#' @return an index to query
#' @export

cluster_sample <- function(X,y,unlabel_index_c,dis="euclidean",...){

        label_index <- which(!is.na(y))
        x_lab <- X[label_index,]
        y_lab <- y[label_index]
        x_ulab <- X[unlabel_index_c,]
        y_ulab <- y[unlabel_index_c]

        # Select the point furthest from the labeled set
        q <- rep(0,length(y_ulab))
```

```
        for (i in 1:length(y_ulab)) {
                min <- Inf
                for (j in 1:length(y_lab)) {
                        temp <- cs_distance(X[unlabel_index_c[i],],X
                           [label_index[j],],dis)
                        if (min > temp) min <- temp
                }
                q[i] <- min
        }
        index <- unlabel_index_c[which(q==max(q))]
        if (length(index) > 1) index <- sample(index,1)
        index
}

# Main distance engine
cs_distance <- function(a,b,dis = "euclidean"){
        d <- switch(dis,
                euclidean=cs_euclidean_distance(a,b)
                )
}

# Euclidean Distance
cs_euclidean_distance <- function(a,b) {
        sqrt( sum( mapply( function(x,y) (x-y)^2, a, b)))
}
```

A.9 AL simulation study

Refer to Section 3.3.

A.9.1 MNIST data

```
# Load the MNIST dataset
load_mnist <- function() {
        load_image_file <- function(filename) {
                ret = list()
                f = file(filename,'rb')
                readBin(f,'integer',n=1,size=4,endian='big')
                ret$n = readBin(f,'integer',n=1,size=4,endian='big')
                nrow = readBin(f,'integer',n=1,size=4,endian='big')
                ncol = readBin(f,'integer',n=1,size=4,endian='big')
                x = readBin(f,'integer',n=ret$n*nrow*ncol,size=1,
                   signed=F)
                ret$x = matrix(x, ncol=nrow*ncol, byrow=T)
                close(f)
                ret
        }
```

```
        load_label_file <- function(filename) {
                f = file(filename,'rb')
                readBin(f,'integer',n=1,size=4,endian='big')
                n = readBin(f,'integer',n=1,size=4,endian='big')
                y = readBin(f,'integer',n=n,size=1,signed=F)
                close(f)
                y
        }
        train <<- load_image_file('mnist/train-images-idx3-ubyte')
        train$y <<- load_label_file('mnist/train-labels-idx1-ubyte')
}

# Plot a single digit
show_digitsmall <- function(arr196, col=gray(12:1/12), ...) {
        image(matrix(arr196, nrow=14)[,14:1], col=col, ...)
}

# Compress the MNIST dataset from 28x28 to 14x14
compressImg <- function(full){
        compressFour <- function(j){
                pixelvec = rep(NA,4)
                pixelvec[1] = full[2*j-1+floor((j-1)/14)*28];
                pixelvec[2] = full[2*j+floor((j-1)/14)*28];
                pixelvec[3] = full[2*j-1+28+floor((j-1)/14)*28];
                pixelvec[4] = full[2*j+28+floor((j-1)/14)*28];
                return(mean(pixelvec))
        }

        compress = unlist(lapply(1:196,compressFour))
        return(compress)
}

# Plot a multitude of digits
plotTable <- function(numRow,numCol,vec.labels,mat.images){
        vec.uniq = unique(vec.labels)
        par(mfrow=c(numRow,numCol),pty="s",mar = c(0.1,0.1,0.1,0.1))
        for(i in 1:length(vec.uniq)){
                tmpidx = which(vec.labels==vec.uniq[i])
                for(j in 1:length(which(vec.labels==vec.uniq[i]))){
                        show_digitsmall(mat.images[tmpidx[j],],asp=
                           TRUE)
                }
        }
}
```

A.9.2 Simulation engine

Although the function name is `AL_engine`, this code corresponds to the simulation engine (all the `main` simulation file names are preceded by a `AL_`). The actual active learning engine may be found in Appendix A.9.3.

```
AL_engine <- function(X, y, y_unlabeled, al_method,
classifier_method, return_method, iter, n, ...){

        stopifnot(nrow(X) == length(y), is.matrix(X), is.factor(y),
                length(levels(y)) == 2)
        idx <- which(is.na(y_unlabeled))
        stopifnot(length(idx) > 0, all(y[-idx] == y_unlabeled[-idx])
           ,
          length(y)==length(y_unlabeled),is.factor(y_unlabeled))

        res <- rep(0,iter)

        ### SET THE COMMITTEE HERE
        cm <- c("rf","nb","pls","svmRadialWeights")
        err<- rep(0,length(cm))

        for(i in 1:iter){
                # If QBC, the procedure is a little different....
                if (al_method == "qbc") {
                        if (i != 1 &
                        as.character(substitute(classifier_method))
                           =="qbc_majority") {
                                # QBC Majority method re-trains
                                   committee after the oracle
                                # Save computation time by passing
                                   those results to QBC algo
                                next_sample <- active_learning(X=X,
                                        y=y_unlabeled,
                                        almethod=al_method,n=n,
                                        committee = cm,
                                        isMajority = TRUE,
                                        tout = tout, ...)
                        } else {
                        next_sample <- active_learning(X=X,
                                y=y_unlabeled, almethod=al_method,
                                n=n, committee = cm, ...)
                }
                y_unlabeled[next_sample[[1]]] <- y[next_sample[[1]]]

                # Update error and prune committee
                if (i > iter/2) {
                        prune <- active_learning(X=X, y=y_unlabeled,
                                almethod="qbc_prune", n = 0,
                                index=next_sample[[1]],
                                committee_pred=next_sample[[2]],
                                k = i, err = err, is_prune = TRUE,
                                ...)
                        err <- prune[[1]]
                        # check if there's stuff to prune
                        if (length(prune[[2]] != 0)) {
                                cm <- cm[-unlist(prune[[2]])]
                                err <- err[-unlist(prune[[2]])]
                        }
```

```
                }
                else {
                        prune <- active_learning(X=X, y=y_unlabeled,
                                almethod="qbc_prune", n = 0,
                                index=next_sample[[1]],
                                committee_pred=next_sample[[2]],
                                k = i, err = err, is_prune = FALSE,
                                ...)
                        err <- prune[[1]]
                }

                # Compute residual error
                idx <- which(!is.na(y_unlabeled))
                tout <- classifier_method(X[idx,], y_unlabeled[idx],
                    committee = cm)
                res[i] <- return_method(tout, X, y, committee = cm)
        }
        # Everything else (not QBC)
        else {
        if (i != 1 & al_method == "us") {
                # classifier_method re-trains random forest after
                    the oracle
                # Save computation time by passing those results to
                    US algo
                # Of course, this only works since classifier = "rf
                    ", and the
                # classifier_method function also uses "rf"
                next_sample <- active_learning(X,
                        y_unlabeled, al_method, n,
                        isR = TRUE, tout = tout, ...)
        } else {
                next_sample <- active_learning(X, y_unlabeled, al_
                   method, n, ...)
        }
                y_unlabeled[next_sample] <- y[next_sample]

                # Compute residual error
                idx <- which(!is.na(y_unlabeled))
                tout <- classifier_method(X[idx,], y_unlabeled[idx])
                res[i] <- return_method(tout, X, y)
                }
        }
        res
}
```

The AL engine without committee pruning (`AL_engine_noprune`) is the exact same as `AL_engine` with the committee pruning section removed. `AL_engine_noprune` is called by the two QBC methods which do not utilize committee pruning (see Table 3.1 where *C_Pruning* = F). For the sake of completeness, the code is also included below.

```
AL_engine_noprune <- function(X, y, y_unlabeled, al_method,
classifier_method, return_method, iter, n, ...){

        stopifnot(nrow(X) == length(y), is.matrix(X), is.factor(y),
        length(levels(y)) == 2)
        idx <- which(is.na(y_unlabeled))
        stopifnot(length(idx) > 0, all(y[-idx] == y_unlabeled[-idx])
            ,
        length(y)==length(y_unlabeled),is.factor(y_unlabeled))

        res <- rep(0,iter)

        ### SET THE COMMITTEE HERE
        cm <- c("rf","nb","pls","svmRadialWeights")
        err<- rep(0,length(cm))

        for(i in 1:iter){
                # If QBC, the procedure is a little different....
                if (al_method == "qbc") {
                        if (i != 1 &
                        as.character(substitute(classifier_method))
                           =="qbc_majority") {
                                # QBC Majority method re-trains
                                   committee after the oracle
                                # Save computation time by passing
                                   those results to QBC algo
                                next_sample <- active_learning(X=X,
                                        y=y_unlabeled,
                                        almethod=al_method,n=n,
                                        committee = cm,
                                        isMajority = TRUE,
                                        tout = tout, ...)
                        } else {
                        next_sample <- active_learning(X=X,
                                y=y_unlabeled, almethod=al_method,
                                n=n, committee = cm, ...)
                }
                y_unlabeled[next_sample[[1]]] <- y[next_sample[[1]]]

                # Compute residual error
                idx <- which(!is.na(y_unlabeled))
                tout <- classifier_method(X[idx,], y_unlabeled[idx],
                    committee = cm)
                res[i] <- return_method(tout, X, y, committee = cm)
        }
        # Everything else (not QBC, not US)
        else {
                if (i != 1 & al_method == "us") {
                        # classifier_method re-trains random forest
                           after the oracle
                        # Save computation time by passing those
                           results to US algo
                        # Of course, this only works since
                           classifier = "rf", and the
```

```
                        # classifier_method function also uses "rf"
                        next_sample <- active_learning(X,
                                y_unlabeled, al_method, n,
                                isR = TRUE, tout = tout, ...)
                } else {
                next_sample <- active_learning(X, y_unlabeled, al_
                   method, n, ...)
                }
                y_unlabeled[next_sample] <- y[next_sample]

                # Compute residual error
                idx <- which(!is.na(y_unlabeled))
                tout <- classifier_method(X[idx,], y_unlabeled[idx])
                res[i] <- return_method(tout, X, y)
                }
        }
        res
}
```

A.9.3 AL algorithm engine

```
#' Main active learning engine
#'
#' The missing labels in y are denoted by NA.
#' This method takes X as a matrix of all the data
#'
#' @param X the full data matrix, n x d, including all unlabeled
   data
#' @param y a factor vector with 2 levels and NAs for unlabeled data
#' @param almethod the AL method name
#' @param n is the number of unlabeled points to be "pooled"
#' @param ... additional parameters for the active learning method
#'
#' @return an index corresponding to the row of X to learn the label
    of next
#' @export

active_learning <- function(X, y, almethod = "us", n, ...){

        stopifnot(nrow(X) == length(y), is.matrix(X), any(is.na(y)),
        is.factor(y), length(levels(y)) == 2)

        if (n == 0) {
                unlabel_index_c <- which(is.na(y))
        } else unlabel_index_c <- sample(which(is.na(y)), n)

        switch(almethod,
                us=uncertainty_sample(X,y,unlabel_index_c, ...),
                rs=random_sample(unlabel_index_c, ns = 1, ...),
                qbc=qbc_sample(X,y,unlabel_index_c, ...),
```

```
                qbb=qbb_sample(X,y,unlabel_index_c,...),
                qbc_prune=qbc_prune(X=X, y=y, ...),
                cluster=cluster_sample(X,y,unlabel_index_c, ...)
                )
}
```

A.9.4 Simulator (main)

Finally, the main simulator is what calls the simulation engine 25 times (trials) for each active learning method with the parameters described in Table 3.1 and collects the results.

```
setwd("---simulation␣file␣path---")
# NOTE: AL_header.R loads the simulation R package
# External installation via devtools::install_github("amytian789/
    thesis-al")
# Local installation via devtools::load_all("---package path---")
source("main/AL_header.R")
source("main/AL_engine.R")
source("main/AL_engine_noprune.R")
source("main/AL_data.R")




################################### Overall classifier and return
    methods

# MAIN class.model that will train on the data once AL selection is
    completed
# X and y contain labeled points
classifier_method <- function(X, y, ...) {
        caret::train(X,y,method="rf")
}

# MAIN prediction method for the data once AL selection is completed
# X contains all points to predict
classifier_predict <- function(classifier, X, ...) {
        stats::predict(classifier, X)
}

# Majority Committee Vote classification model
# X and y contain labeled points
qbc_majority <- function(X, y, committee, ...) {
        tout <- vector("list",length(committee))
        for (i in 1:length(committee)){
                tout[[i]] <- caret::train(X,y,committee[i])
        }
        tout
}
```

```
# Generic error ratio
# X contain all points. y are known labels (unknown to the learning
    algorithm)
return_method <- function(classifier, X, y, ...) {
        p <- stats::predict(classifier, X)
        length(which(p != y))/length(y)
}

# QBC error ratio
# X contain all points. y are known labels (unknown to the learning
    algorithm)
qbc_m_return <- function(tout, X, y, committee, ...) {
        p <- vector("list",length(committee))
        for (i in 1:length(committee)) {
                p[[i]] <- predict(tout[[i]],newdata=X)
        }
        # Aggregate prediction
        ap <- rep(0,length(y))
        for (i in 1:length(y)){
                temp <- as.numeric(as.character(p[[1]][i]))
                for (j in 1:length(committee)){
                        temp <- c(temp,as.numeric(as.character(p[[j
                            ]][i])))
                }
                # error checking if a value doesn't appear at all
                if (is.na(as.numeric(sort(table(temp),decreasing=
                    TRUE)[2]))) {
                        ap[i] <- as.numeric(names(sort(table(temp),
                            decreasing=TRUE)[1]))
                } else {
                        # pick one at random if there is a tie
                        if (as.numeric(sort(table(temp),decreasing=
                            TRUE)[1]) ==
                        as.numeric(sort(table(temp),decreasing=TRUE)
                            [2])){
                                temp <- c(0,1)
                                ap[i] <- sample(temp,1)
                        } else {
                                # Otherwise, insert the first one
                                ap[i] <- as.numeric(names(sort(table
                                    (temp),decreasing=TRUE)[1]))
                        }
                }
        }
        length(which(ap != y))/length(y)
}



################################## Set up the data

load_mnist()
```

```
names(train)

# Randomly select the dataset. a and b are the labels which we want
    to compare
# (a,b in [0,10]. We are only interested bivariate classification)
a <- 7
b <- 9
n <- 250 # desired dataset size
init <- 10 # desired number of points to initialize with
set.seed(10)
idx <- c(sample(which(train$y == a),n/2),sample(which(train$y == b),
    n/2))
X <- train$x[idx,]
X <- t(apply(X,1,compressImg)) # compress from 28x28 to 14x14 pixels
y <- as.factor(train$y[idx]) # y contains the "true" labels. y is
    never seen by
the AL algorithms

# Randomly select the initial points given to the AL algorithms
y_unlabeled <- y
set.seed(10)
y_unlabeled[sample(1:n,n-init)] <- NA

# Visual representation of the data
plotTable(13,20,y,X)

rm(train)



####################################

s <- 15 # Number of random unlabeled points to "pool"
        # n = 0 indicates that the "pool" should sample from all
            data points
k <- 25 # Number of simulations to run
iter <- 50  # Number of AL algorithm iterations (the "budget")



# Classifier performance given all data is labeled
# pred <- classifier_method(X,y)
# perf_results <- rep(return_method(pred,X,y),iter)
# This has been shown to yield perfect results, so it is commented
    out

# Uncertainty Sampling
us_results <- matrix(0,nrow=k,ncol=iter)
for (i in 1:k){
        set.seed(i)
        us_results[i,] <- AL_engine(X=X, y=y,
                y_unlabeled=y_unlabeled, al_method = "us",
```

```
                classifier_method = classifier_method,
                return_method = return_method,
                iter = iter, n = s, classifier = "rf")
        print(c("Trial␣",i,"complete"))
}

# Query by Committee with "Majority Committee Vote" model
qbc_majority_results <- matrix(0,nrow=k,ncol=iter)
for (i in 1:k){
        set.seed(i)
        ### To change the committee, you must set it in the AL_
            engine
        qbc_majority_results[i,] <- AL_engine(X=X, y=y,
                y_unlabeled=y_unlabeled, al_method = "qbc",
                classifier_method = qbc_majority,
                return_method = qbc_m_return,
                iter = iter, n = s, dis = "vote_entropy", pt = 0.5)
        print(c("Trial␣",i,"complete"))
}

# Query by Committee with "Majority Committee Vote" model
# no pruning
qbc_majority_noprune_results <- matrix(0,nrow=k,ncol=iter)
for (i in 1:k){
        set.seed(i)
        ### To change the committee, you must set it in the AL_
            engine_noprune
        qbc_majority_noprune_results[i,] <- AL_engine_noprune(X=X,
                y=y, y_unlabeled=y_unlabeled, al_method = "qbc",
                classifier_method = qbc_majority,
                return_method = qbc_m_return,
                iter = iter, n = s, dis = "vote_entropy", pt = 0.5)
        print(c("Trial␣",i,"complete"))
}

# Query by Committee with overall "Random Forest" model
qbc_rf_results <- matrix(0,nrow=k,ncol=iter)
for (i in 1:k){
        set.seed(i)
        ### To change the committee, you must set it in the AL_
            engine
        qbc_rf_results[i,] <- AL_engine(X=X, y=y,
                y_unlabeled=y_unlabeled, al_method = "qbc",
                classifier_method = classifier_method,
                return_method = return_method,
                iter = iter, n = s, dis = "vote_entropy", pt = 0.5)
        print(c("Trial␣",i,"complete"))
}

# Query by Committee with overall "Random Forest" model
# no pruning
qbc_rf_noprune_results <- matrix(0,nrow=k,ncol=iter)
for (i in 1:k){
        set.seed(i)
```

```
        ### To change the committee, you must set it in the AL_
            engine_noprune
        qbc_rf_noprune_results[i,] <- AL_engine_noprune(X=X, y=y,
                y_unlabeled=y_unlabeled, al_method = "qbc",
                classifier_method = classifier_method,
                return_method = return_method,
                iter = iter, n = s, dis = "vote_entropy", pt = 0.5)
        print(c("Trial␣",i,"complete"))
}

# Query by Bagging
qbb_results <- matrix(0,nrow=k,ncol=iter)
for (i in 1:k){
        set.seed(i)
        qbb_results[i,] <- AL_engine(X=X, y=y,
                y_unlabeled=y_unlabeled, al_method = "qbb",
                classifier_method = classifier_method,
                return_method = return_method,
                iter = iter,n = s,classifier="rf",
                dis = "vote_entropy", num_class=5, r=0.75)
        print(c("Trial␣",i,"complete"))
}

# Min-Max Clustering
cluster_results <- matrix(0,nrow=k,ncol=iter)
for (i in 1:k){
        set.seed(i)
        cluster_results[i,] <- AL_engine(X=X, y=y,
                y_unlabeled=y_unlabeled, al_method = "cluster",
                classifier_method = classifier_method,
                return_method = return_method,
                iter = iter,n = s, dis = "euclidean")
        print(c("Trial␣",i,"complete"))
}

# Random Sampling
random_results <- matrix(0,nrow=k,ncol=iter)
for (i in 1:k){
        set.seed(i)
        random_results[i,] <- AL_engine(X, y, y_unlabeled,
                al_method = "rs", classifier_method, return_method,
                iter, s)
        print(c("Trial␣",i,"complete"))
}

# Average
us_vec <- apply(us_results,2,mean)
random_vec <- apply(random_results,2,mean)
qbc_majority_vec <- apply(qbc_majority_results,2,mean)
qbc_majority_noprune_vec <- apply(qbc_majority_noprune_results,2,
    mean)
qbc_rf_vec <- apply(qbc_rf_results,2,mean)
qbc_rf_noprune_vec <- apply(qbc_rf_noprune_results,2,mean)
qbb_vec <- apply(qbb_results,2,mean)
```

```
cluster_vec <- apply(cluster_results,2,mean)

# Select best QBC output (with pruning)
if (length(which(qbc_majority_vec < qbc_rf_vec)) >
length(which(qbc_majority_vec > qbc_rf_vec))){
        qbc_prune_vec <- qbc_majority_vec
} else if (length(which(qbc_majority_vec < qbc_rf_vec)) <
length(which(qbc_majority_vec > qbc_rf_vec))){
        qbc_prune_vec <- qbc_rf_vec
} else{
        # select one at random
        set.seed(1)
        rr <- sample(c(0,1),1)
        if (rr == 0) qbc_prune_vec <- qbc_majority_vec
        else qbc_prune_vec <- qbc_rf_vec
}
# Select best QBC output (with no pruning)
if (length(which(qbc_majority_noprune_vec < qbc_rf_noprune_vec)) >
length(which(qbc_majority_noprune_vec > qbc_rf_noprune_vec))){
        qbc_noprune_vec <- qbc_majority_noprune_vec
} else if (length(which(qbc_majority_noprune_vec < qbc_rf_noprune_
   vec)) <
length(which(qbc_majority_noprune_vec > qbc_rf_noprune_vec))){
        qbc_noprune_vec <- qbc_rf_noprune_vec
} else{
        # select one at random
        set.seed(2)
        rr <- sample(c(0,1),1)
        if (rr == 0) qbc_noprune_vec <- qbc_majority_noprune_vec
        else qbc_noprune_vec <- qbc_rf_noprune_vec
}
# Select best overall QBC output
if (length(which(qbc_prune_vec < qbc_noprune_vec)) >
length(which(qbc_prune_vec > qbc_noprune_vec))){
        qbc_vec <- qbc_prune_vec
} else if (length(which(qbc_prune_vec < qbc_noprune_vec)) <
length(which(qbc_prune_vec > qbc_noprune_vec))){
        qbc_vec <- qbc_noprune_vec
} else{
        # select one at random
        set.seed(3)
        rr <- sample(c(0,1),1)
        if (rr == 0) qbc_vec <- qbc_prune_vec
        else qbc_vec <- qbc_noprune_vec
}


################################### Plot the results

date <- Sys.Date()
pdf(file=paste0("results/results_", date, ".PDF"),
height = 6, width = 10)

### Plot all AL performance
```

```
ymax <- max(c(us_vec, random_vec, qbc_vec,cluster_vec))
graphics::plot(1:iter, qbc_vec, ylim=c(0,ymax), lwd=2, type="l",
        main="AL Error Ratios with Random Forest classification
           model*",
        xlab="Iterations", ylab="Error", col = "green")
mtext("*Query by Committee uses Majority Committee Vote
   classification model")
graphics::lines(1:iter, random_vec, lwd = 2, col = "red")
graphics::lines(1:iter, us_vec, lwd = 2, col = "black")
graphics::lines(1:iter, qbb_vec, lwd = 2, col = "blue")
graphics::lines(1:iter, cluster_vec, lwd = 2, col = "orange")
graphics::legend(x="bottomleft",lwd=2,cex = 0.75,
        title="Active Learning method",
        legend = c("Random Sampling","Uncertainty Sampling",
        "Query by Committee (best)","Query by Bagging","Min-Max
           Clustering"),
        col=c("red","black","green","blue","orange"))

### Plot QBC performance
graphics::plot(1:iter,qbc_majority_vec,ylim=c(0,ymax),lwd=2,type="l"
        ,main="Query by Committee AL Error Ratio with various
           parameters",
        xlab="Iterations", ylab="Error", col = "red")
graphics::lines(1:iter, qbc_majority_noprune_vec, lwd = 2, lty = 2,
   col = "red")
graphics::lines(1:iter, qbc_rf_vec, lwd = 2, col = "blue")
graphics::lines(1:iter,qbc_rf_noprune_vec, lwd = 2, lty = 2, col = "
   blue")
graphics::legend(x="bottomleft",lwd=2,cex = 0.75,
        title="Main Classifier | Committee Pruning?",
        legend=c("Majority Committee Vote | Yes", "Majority
           Committee Vote | No",
        "Random Forest | Yes", "Random Forest | No"),
        col=c("red","red","blue","blue"), lty=c(1,2,1,2))

graphics.off()
save.image(file = paste0("results/results_", date, ".RData"))
```

A.10 Graph summarization engine

Given graphs $\hat{G}^1 = (V^1, E^1)$ and $\hat{G}^2 = (V^2, E^2)$, the graph summarization engine computes the vector $d^{1,2}$ which contains the difference between $\hat{G}^1$ and $\hat{G}^2$ with the various summarization metrics described in Section 4.2. After the graph summarization engine code, the distance engine function is also included. The distance engine is called by the graph comparison engine in order to compute the distance between the

two results (one from each graph) for a summarization method. Distance methods available are Euclidean distance, L1, L2, and Jaccard distance (for use with community only).

```
# g1 and g2 are igraphs
# gSumm is a vector of strings corresponding to graph summarization
    methods
        ### cen_deg = Centrality (degree)
        ### cen_clo = Centrality (closeness)
        ### cen_bet = Centrality (betweenness)
        ### ast = Assortativity
        ### com_rw = Community (random walk)
        ### com_im = Community (infomap)
        ### com_bet = Community (betweenness)
        ### dis = distance matrix
        ### eco = Edge connectivity
        ### edh = Edge density histogram
# distf is the desired difference function

GC_engine <- function(g1, g2, gSumm, distf = "euclidean", ...){

        stopifnot(igraph::is_igraph(g1), igraph::is_igraph(g2),
                igraph::gorder(g1) == igraph::gorder(g2),
                !is.null(gSumm))

        diff <- rep(0,length(gSumm))
        names(diff) <- rep("",length(gSumm))
        i <- 1

        # Centrality (degree)
        if ("cen_deg" %in% gSumm){
                a <- igraph::centr_degree(g1)$centralization
                b <- igraph::centr_degree(g2)$centralization

                diff[i] <- dist_engine(a,b,distf)
                names(diff)[i] <- "cen_deg"
                i <- i + 1
        }

        # Centrality (closeness)
        if ("cen_clo" %in% gSumm){
                a <- igraph::centr_clo(g1)$centralization
                b <- igraph::centr_clo(g2)$centralization

                diff[i] <- dist_engine(a,b,distf)
                names(diff)[i] <- "cen_clo"
                i <- i + 1
        }

        # Centrality (betweenness)
        if ("cen_bet" %in% gSumm){
```

```
        a <- igraph::centr_betw(g1)$centralization
        b <- igraph::centr_betw(g2)$centralization

        diff[i] <- dist_engine(a,b,distf)
        names(diff)[i] <- "cen_bet"
        i <- i + 1
}

# Assortativity
if ("ast" %in% gSumm){
        a <- igraph::assortativity_degree(g1)
        b <- igraph::assortativity_degree(g2)

        if (is.nan(a)) a <- 0
        if (is.nan(b)) b <- 0

        diff[i] <- dist_engine(a,b,distf)
        names(diff)[i] <- "ast"
        i <- i + 1
}

# Community (random walk)
if ("com_rw" %in% gSumm){
        a <- igraph::membership(igraph::cluster_walktrap(
                g1,steps=igraph::gorder(g1)/2))
        b <- igraph::membership(igraph::cluster_walktrap(
                g2,steps=igraph::gorder(g2)/2))

        diff[i] <- dist_engine(a,b,dist="jaccard")
        names(diff)[i] <- "com_rw"
        i <- i + 1
}

# Community (infomap)
if ("com_im" %in% gSumm){
        a <- igraph::membership(igraph::cluster_infomap(g1))
        b <- igraph::membership(igraph::cluster_infomap(g2))

        diff[i] <- dist_engine(a,b,dist="jaccard")
        names(diff)[i] <- "com_im"
        i <- i + 1
}

# Community (betweenness)
if ("com_bet" %in% gSumm){
        a <- igraph::membership(igraph::cluster_edge_
           betweenness(g1))
        b <- igraph::membership(igraph::cluster_edge_
           betweenness(g2))

        diff[i] <- dist_engine(a,b,dist="jaccard")
        names(diff)[i] <- "com_bet"
        i <- i + 1
}
```

```
# Distance matrix
if ("dis" %in% gSumm){
    a <- distances(g1)
    b <- distances(g2)

    # change from matrix -> vector for distance computation
       (by column)
    # keep only 1 side of the matrix (both sides are the
       same)
    # don't keep the values in the middle (since it's the
       same node)
    a[a == Inf] <- 0
    a <- a[upper.tri(a)]
    b[b == Inf] <- 0
    b <- b[upper.tri(b)]

    diff[i] <- dist_engine(a,b,distf) / (dist_engine(
            rep(0,igraph::gorder(g1)-1),
            seq(1,igraph::gorder(g1)-1,by=1),distf)
    names(diff)[i] <- "dis"
    i <- i + 1
}

# Edge connectivity
if ("eco" %in% gSumm){
        if (min(igraph::degree(g1)) != 0){
        a <- igraph::edge_connectivity(g1) / min(igraph::
           degree(g1))
        } else a <- 0
        if (min(igraph::degree(g2)) != 0){
        b <- igraph::edge_connectivity(g2) / min(igraph::
           degree(g2))
        } else b <- 0

        diff[i] <- dist_engine(a,b,distf)
        names(diff)[i] <- "eco"
        i <- i + 1
}

# Edge density histogram
if ("edh" %in% gSumm){
        # histograms should be on the same scale.
        # Use Freedman-Diaconis rule to determine bin width
        dd <- max(igraph::degree(g1),igraph::degree(g2))
        bw <- 2 * stats::IQR(igraph::degree(g1)) / igraph::
           gorder(g1)^(1/3)
        a <- graphics::hist(igraph::degree(g1),plot=FALSE,
                breaks=seq(0,dd+bw,by=bw))$counts / igraph::
                   gorder(g1)
        b <- graphics::hist(igraph::degree(g2),plot=FALSE,
                breaks=seq(0,dd+bw,by=bw))$counts / igraph::
                   gorder(g2)
```

```
                diff[i] <- dist_engine(a,b,distf)
                names(diff)[i] <- "edh"
                i <- i + 1
        }

        diff
}

######## Engine to call various distance computation methods
dist_engine <- function(a,b,dist = "euclidean", ...){
        switch(dist,
        euclidean=dist_euc(a,b),
        l1=dist_l1(a,b),
        l2=dist_l2(a,b),
        jaccard=dist_jac(a,b)
        )
}

# Euclidean distance
dist_euc <- function(a,b){
        sqrt( sum( mapply( function(x,y) (x-y)^2, a, b)))
}

# L1: Sum of absolute differences
dist_l1 <- function(a,b){
        sum ( mapply ( function(x,y) abs(x-y), a, b))
}

# L2: Sum of squared differences
dist_l2 <- function(a,b){
        sum ( mapply ( function(x,y) (x-y)^2, a, b))
}

# Jaccard distance
dist_jac <- function(a,b){
        1-clusteval::cluster_similarity(a,b,similarity="jaccard")
}
```

A.11 Similarity selection

Given a base graph $\hat{G}$ and a set of graphs, the similarity selection method selects the graph $\hat{G}^*$ that is most similar to $\hat{G}$. The method is described in Section 4.3.

```
# Search for the "most similar" graph pair
# (characterized by having the lowest differences across the board)
#
# gc is a list of the differences among various graph pairs
# base is the index with which to start the search
```

```
GC_selection <- function(gc, base = 1){

        stopifnot(!is.null(gc))

        idx <- base
        old_idx <- 0
        while (old_idx != idx){
                old_idx <- idx
                idx <- best_c(gc,idx)
                #print(paste(old_idx,idx))
        }
        idx
}

# Select the next idx candidate given current best candidate
best_c <- function(gc, cand){

        for (i in 1:length(gc)){
                if (i != cand){
                        if (sum(gc[[cand]] < gc[[i]]) < sum(gc[[cand
                           ]] > gc[[i]])){
                                return( i )
                        } else if (sum(gc[[cand]] < gc[[i]]) > sum(
                           gc[[cand]] > gc[[i]])){
                                # do nothing since the current
                                    candidate is better
                        } else{
                                return( sample(c(cand,i),1) )
                        }
                }
        }
        return( cand )
}
```

A.12 GC simulation study

The graph comparison simulation study, which is used to demonstrate the viability of the proposed similarity selection model, is described in Section 4.3.1.

```
setwd("---simulation␣file␣path---")
source("GC_engine.R")
source("dist_engine.R")
source("GC_selection.R")

library(igraph)
library(clusteval)
```

```
################################## Random sample given PDF

randomdraw <- function(n, prob){
        if(round(sum(prob),1) != 1 | length(which(prob<0)) > 0)
        stop("Probability must be between 0 and 1")

        runningsum <- 0
        s <- runif(n)
                for (i in 1:n){
                        for (j in 1:length(prob)){
                                runningsum <- runningsum + prob[j]
                                if (s[i] < runningsum){
                                s[i] <- j
                                break
                        }
                }
        }
        s
}

################################## Run simulations

# Create base graph
set.seed(10)
ss <- 50 # desired sample size (# of edges)
bg <- igraph::sample_gnm(20, ss)
bg_e <- igraph::as_edgelist(bg)

# Setting parameters
gSumm <- c("cen_deg","cen_clo","cen_bet","ast",
        "com_rw","com_im","com_bet","dis","eco","edh")
distf <- "l2"

msg <- rep(0,1000)
for (i in 1:1000) {
        set.seed(i)

        # g1: randomly swap 20% edges.
        # Weight of each node is proportional to its degree
        g1 <- bg
        for (j in 1:(ss/5)) {
                idx <- sample(igraph::as_edgelist(g1),1)
                g1 <- igraph::delete_edges(g1,idx)
                prob <- igraph::degree(g1) / sum(igraph::degree(g1))
                nva <- sample(seq(1,length(prob),1),1)
                nvb <- randomdraw(1, prob)
                # make sure edges are not repeated
                while (g1[nva,nvb] == 1 | nva == nvb) nvb <-
                   randomdraw(1,prob)
                g1 <- igraph::add_edges(g1, c(nva,nvb))
        }

        # g2: randomly swap 100% edges.
        # Weight of each node is proportional to its degree
```

```
        g2 <- bg
        for (j in 1:(ss)) {
                idx <- sample(igraph::as_edgelist(g2),1)
                g2 <- igraph::delete_edges(g2,idx)
                prob <- igraph::degree(g2) / sum(igraph::degree(g2))
                nva <- sample(seq(1,length(prob),1),1)
                nvb <- randomdraw(1, prob)
                # make sure edges are not repeated
                while (g2[nva,nvb] == 1 | nva == nvb) nvb <-
                   randomdraw(1,prob)
                g2 <- igraph::add_edges(g2, c(nva,nvb))
        }

        # Compute difference between (bg,g1), (bg,g2)
        gc <- vector("list",2)
        gc[[1]] <- GC_engine(g1=bg,g2=g1,gSumm=gSumm,distf=distf)
        gc[[2]] <- GC_engine(g1=bg,g2=g2,gSumm=gSumm,distf=distf)

        # Select the most similar graph pair
        msg[i] <- GC_selection(gc = gc, base = 1)
}
p1 <- length(which(msg == 1)) / length(msg)
p2 <- length(which(msg == 2)) / length(msg)
cat("Prob. of selecting (bg,g1):",p1,"\nProb. of selecting (bg,g2):"
    , p2)

################################### Plot simulation graphs

bg$layout <- layout_in_circle # base graph
g1$layout <- layout_in_circle # from trial 1000 (seed = 1000)
g2$layout <- layout_in_circle # from trial 1000 (seed = 1000)

par(mfrow=c(1,3))
plot(bg)
plot(g1)
plot(g2)
```

A.13 Stock selection

Given a correlation graph $G = (V, E)$, the program selects $k < |V|$ stocks (nodes) such that they are as uncorrelated with each other as possible. See Section 5.2 and Algorithm 10.

```
# g_num is the numerical graph as an adjacency matrix
# k is the number of stocks to select
# dd is a vector of sample average price differences over time
# pd is the "drift" threshold ratio
```

```
# vv is a vector of sample average standard deviation
# pv is the "volatility" threshold ratio
#
# Returns a vector of selected stock tickers

stock_selection <- function(g_num, k, dd, pd = 0, vv, pv = 0){

        stopifnot(ncol(g_num) > k, pd >= 0, pv >= 0)

        # A) Drift
        rmv <- c()
        avgret <- sum(dd) / ncol(dd)
        for (i in 1:ncol(g_num)){
                if(dd[i] <= avgret * pd){
                        rmv <- c(rmv, i)
                }
        }

        # B) Volatility
        avgvol <- sum(vv) / ncol(vv)
        for (i in 1:ncol(g_num)){
                if(vv[i] <= avgvol * pv){
                        if(!(i %in% rmv)){
                                rmv <- c(rmv, i)
                        }
                }
        }
        g_num <- g_num[-rmv,-rmv]

        #print(g_num)

        # C) Main selection
        z <- combn(ncol(g_num),k)
        min <- Inf
        idx <- 0
        for (i in 1:ncol(z)){
                c <- 0 # number of connections
                for (j in 1:(k-1)){
                        for (l in (j+1):k){
                                c <- c + g_num[z[j,i],z[l,i]]
                        }
                        if (c < min){
                                min <- c
                                idx <- i
                        } else if (c == min){
                                if (runif(1,0,1) > 0.5){
                                        min <- c
                                        idx <- i
                                }
                        } else{
                                # do nothing
                        }
                }
        }
```

```
        idx_s <- c()
        for (i in 1:length(z[,idx])){
                idx_s <- c(idx_s,colnames(g_num)[z[i,idx]])
        }
        idx_s
}
```

A.14 Final application

Section 5.3 details the application procedure that is implemented below.

```
setwd("---application␣file␣path---")
source("stock_selection.R")

library(Rsafd)
library(timeDate)
library(energy)
library(igraph)
library(clusteval)




################################### Read in data.

rd <- read.csv(file="HealthcareETF_prices.csv",sep=",",header=TRUE)
date <- as.timeDate(rd[,1])
data <- rd[,-1]

# avg drift and vol (computed in Excel)
dd <- read.csv(file="HealthcareETF_drift.csv",sep=",",header=TRUE)
vv <- read.csv(file="HealthcareETF_vol.csv",sep=",",header=TRUE)

# Remove seasonal and trend components of data
for (i in 1:ncol(data)){
        temp.ts <- Rsafd::timeSeries(data=data[,i],positions=date)
        # 261 is the avg number of workdays in a year
        temp.stl <- Rsafd::sstl(temp.ts, FREQ=261)
        data[,i] <- temp.stl$rem@data
}

# Split data into testing and training sets
# Assume we are standing in the middle of the 1998 and 2014
data_train <- as.matrix(data[1:(nrow(data)/2),])
data_test <- as.matrix(data[(nrow(data)/2+1):nrow(data),])
```

```
################################## Create numerical graphs

# correlation threshold
t <- 0.15

# Pearson's correlation graph
m_pear <- matrix(0, ncol(data_train), ncol(data_train))
for (i in 1:(ncol(data_train)-1)){
        for (j in (i+1):ncol(data_train)){
                m_pear[i,j] <- stats::cor.test(data_train[,i],
                        data_train[,j],method="pearson")$estimate
        }
}
g_pear <- igraph::make_empty_graph(n = ncol(data_train), directed =
   FALSE)
for (i in 1:(ncol(data_train)-1)){
        for (j in (i+1):ncol(data_train)){
                if (abs(m_pear[i,j]) > t){
                        g_pear <- igraph::add_edges(g_pear, c(i,j))
                }
        }
}

# Spearman's correlation graph
m_spea <- matrix(0, ncol(data_train), ncol(data_train))
for (i in 1:(ncol(data_train)-1)){
        for (j in (i+1):ncol(data_train)){
                m_spea[i,j] <- stats::cor.test(data_train[,i],
                        data_train[,j],method="spearman")$estimate
        }
}
g_spea <- igraph::make_empty_graph(n = ncol(data_train), directed =
   FALSE)
for (i in 1:(ncol(data_train)-1)){
        for (j in (i+1):ncol(data_train)){
                if (abs(m_spea[i,j]) > t){
                        g_spea <- igraph::add_edges(g_spea, c(i,j))
                }
        }
}

# Kendall's correlation graph
m_ktau <- matrix(0, ncol(data_train), ncol(data_train))
for (i in 1:(ncol(data_train)-1)){
        for (j in (i+1):ncol(data_train)){
                m_ktau[i,j] <- stats::cor.test(data_train[,i],
                        data_train[,j],method="kendall")$estimate
        }
}
g_ktau <- igraph::make_empty_graph(n = ncol(data_train), directed =
   FALSE)
for (i in 1:(ncol(data_train)-1)){
        for (j in (i+1):ncol(data_train)){
                if (abs(m_ktau[i,j]) > t){
```

```
                        g_ktau <- igraph::add_edges(g_ktau, c(i,j))
                }
        }
}

# Distance correlation graph
m_dist <- matrix(0, ncol(data_train), ncol(data_train))
for (i in 1:(ncol(data_train)-1)){
        for (j in (i+1):ncol(data_train)){
                m_dist[i,j] <- energy::dcor.ttest(data_train[,i],
                        data_train[,j])$estimate
        }
}
g_dist <- igraph::make_empty_graph(n = ncol(data_train), directed =
   FALSE)
for (i in 1:(ncol(data_train)-1)){
        for (j in (i+1):ncol(data_train)){
                if (abs(m_dist[i,j]) > t){
                        g_dist <- igraph::add_edges(g_dist, c(i,j))
                }
        }
}

date <- Sys.Date()
save.image(file = paste0("corrgraphs_", date, ".RData"))



################################## Visualization system

devtools::install_github("linnylin92/graphicalModelEDA",
        ref = "kevin",subdir = "graphicalModelEDA", force = T)
library(graphicalModelEDA)

# Run the VS
control_obj <- graphicalModelEDA::visualizer_control()
control_obj$parameter_list$initial_set <- 10 # faster initialization
res <- graphicalModelEDA::visualizer_system(data_train, trials = 15,
        visualizer_control = control_obj, asp = T)

# Visual graph
g_visg <- graphicalModelEDA:::visual_graph(res, data_train)
plot(g_visg)

# Heat map
plot(res$results)

# Association navigator
plot(res$results, aes.list = aes.buja())

# Evaluate 100 random pairs and their associated plots
search_res <- graphicalModelEDA::automated_search(res,data_train)
# Histogram of predicted probabilities of correlation
```

```
plot(search_res)
# Ordered predicted probabilities w/ the scatterplot pairs
cbind(search_res$pairs[order(search_res$values),], sort(search_res$
   values))
par(mfrow=c(2,3))
# Plot the top 5 most "interesting" pairs
graphicalModelEDA::selective_plot(search_res, dat,
        selection = function(vec){ifelse(vec >= sort(vec,
        decreasing = TRUE)[5], TRUE, FALSE)})

date <- Sys.Date()
save.image(file = paste0("visgraph_", date, ".RData"))



################################### Graph comparison

setwd("---GC␣file␣path---")
source("GC_engine.R")
source("dist_engine.R")
source("GC_selection.R")
setwd("---application␣file␣path---")

# Setting parameters
gSumm <- c("cen_deg","cen_clo","cen_bet","ast",
        "com_rw","com_im","com_bet","dis","eco","edh")
distf <- "l2"

# Compute difference between all graph pairs (g_visg is the "base")
gc <- vector("list",4)
gc[[1]] <- GC_engine(g1=g_visg,g2=g_pear,gSumm=gSumm,distf=distf)
gc[[2]] <- GC_engine(g1=g_visg,g2=g_spea,gSumm=gSumm,distf=distf)
gc[[3]] <- GC_engine(g1=g_visg,g2=g_ktau,gSumm=gSumm,distf=distf)
gc[[4]] <- GC_engine(g1=g_visg,g2=g_dist,gSumm=gSumm,distf=distf)

# Select the most similar graph pair
idx <- GC_selection(gc = gc, base = 3)

if (idx == 1){
        print("Pearson's")
} else if (idx == 2){
        print("Spearman's")
} else if (idx == 3){
        print("Kendall's")
} else{
        print("Distance")
}



################################### Stock selection
```

```
# Set parameters
k <- 5 # number of stocks to select
pd <- 2/3 # drift threshold ratio
pv <- 2/3 # vol threshold ratio

# Pearson's
set.seed(10)
a_pear <- igraph::as_adjacency_matrix(g_pear)
colnames(a_pear) <- colnames(data)
s_pear <- stock_selection(a_pear, k, dd, pd, vv, pv)
s_pear

# Spearman's
set.seed(10)
a_spea <- igraph::as_adjacency_matrix(g_spea)
colnames(a_spea) <- colnames(data)
s_spea <- stock_selection(a_spea, k, dd, pd, vv, pv)
s_spea

# Kendall's
set.seed(10)
a_ktau <- igraph::as_adjacency_matrix(g_ktau)
colnames(a_ktau) <- colnames(data)
s_ktau <- stock_selection(a_ktau, k, dd, pd, vv, pv)
s_ktau

# Distance
set.seed(10)
a_dist <- igraph::as_adjacency_matrix(g_dist)
colnames(a_dist) <- colnames(data)
s_dist <- stock_selection(a_dist, k, dd, pd, vv, pv)
s_dist

# Visual
set.seed(10)
a_visg <- igraph::as_adjacency_matrix(g_visg)
colnames(a_visg) <- colnames(data)
s_visg <- stock_selection(a_visg, k, dd, pd, vv, pv)
s_visg

date <- Sys.Date()
save.image(file = paste0("portfolios_", date, ".RData"))




################################## Compare future performance

# ON EXCEL: Use testing set to do this
```

Bibliography

[1] N. Abe and H. Mamitsuka. Query learning strategies using boosting and bagging. *Proceedings of the International Conference on Machine Learning (ICML)*, pages 1–9, 1998.

[2] A. Buja, A. Krieger, and E. George. A Visualization Tool for Mining Large Correlation Tables: The Association Navigator. `http://stat.wharton.upenn.edu/~buja/PAPERS/Buja-et-al-Association-Navigator.pdf`, 2016.

[3] A. Cutler. Random forests for regression and classification. `http://www.math.usu.edu/adele/RandomForests/Ovronnaz.pdf`, 2010.

[4] I. Dagan and S. Engelson. Committee-based sampling for training probabilistic classifiers. *Proceedings of the International Conference on Machine Learning*, pages 150–157, 1995.

[5] Dasgupta, S. Two faces of active learning. `http://cseweb.ucsd.edu/~dasgupta/papers/twoface.pdf`, 2011.

[6] B. Efron. Why Isn't' Everyone a Bayesian? *The American Statistician*, pages 1–5, February 1986.

[7] P. Federico and W. Oldford. Evaluation of two interaction techniques for visualization of dynamic graphs. *arXiv preprint arXiv:1608.08936*, pages 1–15, August 2016.

[8] M. Hofert and W. Oldford. Visualizing Dependence in High-Dimensional Data: An Application to S&P 500 Constituent Data. *arXiv preprint arXiv:1609.09429*, pages 1–33, September 2016.

[9] D. Lewis and W. Gale. A sequential algorithm for training text classifiers. *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–12, 1994.

[10] K. Lin. Eda for graphical models. `https://github.com/linnylin92/graphicalModelEDA/tree/kevin`, 2017. Github repository.

[11] R. J. Little. In Praise of Simplicity not Mathematistry! Ten Simple Powerful Ideas for the Statistical Scientist. *Journal of the American Statistical Association*, pages 359–369, July 2013.

[12] H. Liu, J. Mulvey, and T. Zhao. A semiparametric graphical modelling approach for large-scale equity selection. *Quantitative Finance*, pages 1053–1067, 2016.

[13] S. Liu, D. Maljovec, B. Wang, P. T. Bremer, and V. Pascucci. Visualizing high-dimensional data: Advances in the past decade. *IEEE Transactions on Visualization and Computer Graphics*, pages 1249–1268, December 2016.

[14] A. McCallum and K. Nigam. Employing EM in pool-based active learning for text classification. *Proceedings of the International Conference on Machine Learning*, pages 359–367, 1998.

[15] B. O'Connor. load the MNIST data set in R. `https://gist.github.com/brendano/39760`, 2008. Github repository.

[16] J. Ramey. Active learning in r. `https://github.com/ramhiser/activelearning`, 2015. Github repository.

[17] S. Santos, D. Y. Takahashi, A. Nakata, and A. Fujita. A comparative study of statistical methods used to identify dependencies between gene expression signals. *Briefings in Bioinformatics*, pages 1–13, August 2013.

[18] Settles, B. Active Learning Literature Survey. `http://burrsettles.com/pub/settles.activelearning.pdf`, 2010.

[19] G. Szekely, M. Rizzo, and N. Bakirov. Measuring and testing independence by correlation of distances. *The Annals of Statistics*, pages 2769–2794, March 2007.

[20] Tao, T. When is correlation transitive? `https://terrytao.wordpress.com/2014/06/05/when-is-correlation-transitive/`, 2014.

[21] V. Vu, N. Labroche, and B. Bouchon-Meunier. Active Learning for Semi-Supervised K-Means Clustering. *Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on*, pages 12–15, October 2010.

[22] H. Wickham, D. Cook, H. Hofmann, and A. Buja. Graphical inference for infovis. *IEEE Transactions on Visualization and Computer Graphics*, pages 973–979, December 2010.

[23] Y. Zhou and G. Hooker. Interpreting Models via Single Tree Approximation. *arXiv prepint arXiv:1610.09036v1*, pages 1–15, October 2016.