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

Abstract

Often in modern multivariate analyses, data analysts rely solely on statistical estimators to explore the data. We are interested in visualizing high-dimensional correlation graphs as a way to verify numerical tests of dependence, which have far-reaching financial implications. High-dimensional visualization is problematic because (1) the number of plots to sort through increases quadratically as the number of variables increase, (2) it is tedious to verify numerical results with visual results and vice versa. We present a visualization tool that actively learns user preferences, applies the fitted classifier to unlabeled data, and outputs the difference among the numerical graph $G^{\text{num}} = (V, E)$ and the visual graph $G = (V, E)$. As a specific response to the aforementioned problems, we focus on the active learning and graph comparison components of the visualizer system. Both employ simulation studies in order to select the best procedure for use in the selection of healthcare stocks for a portfolio. The data is run through the visualization system, which utilizes the $- - --$ active learning algorithm and $- - --$ graph comparison method to select the correlation graph G^{num} closest to the system's output G. The portfolios are then put through a "buy and hold" strategy. Yearly returns are compiled, and the portfolio corresponding the selected correlation graph G^{num} is one of the top performers (??). The visualization tool provides a simple and intuitive way to improve predictive and portfolio management methods in the financial industry. Moreover (and arguably more importantly), it increases standardization in the data analysis process, thereby increasing accountability in an industry where ambiguity can mean a global financial crisis.

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 datasets, which are more tedious 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" [12]. One solution is to manually plot each explanatory variable against the response variable, 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).

Although methods for dimension reduction have been developed [12], 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. Thus, the problem with current high-dimensional visual analysis is two-fold: (1) there are too many potential plots to sort through manually, and (2) it is tedious to verify numerical results with visual results and vice versa.

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 datasets, analysts blindly depend on estimators and hypothesis tests to explore the data and have no justification of their analyses 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 dataset 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" [10]. This lack of agreement and "cookbook" mentality of data analyses 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 all 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 analyses to evaluate how trustworthy it is. This mentality arises due to the difficulty in visualizing high-dimensional data; plotting is one of the most consistent and universally interpretable “sanity checks” for numerical results.

Consider the following scenario with two different univariate datasets (Appendix A.1 and A.2). 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: Numerical analysis in the univariate 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 previous numerical results (which were strongly uncorrelated) cast doubt on the sole correlated estimator.

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. See Sec-

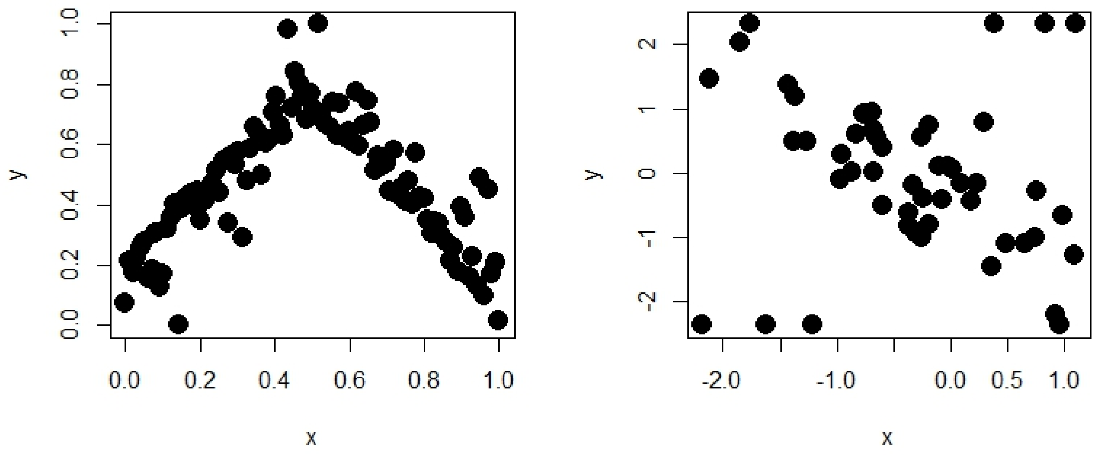

Figure 1.1: Visual analysis in the univariate case. The data exhibits a strong visual dependency but fails common numerical tests of dependence (Table 1.1). *Left*: Dataset 1 (Appendix A.1), *Right*: Dataset 2 (Appendix A.2)

tion 1.2) are not significantly different from 0 (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 datasets in one-dimension that evade commonly-used numerical methods, it is believable that it is even easier to construct analogous datasets 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 methods is especially important in finance as equity markets are large and involve billions of dollars; portfolio selection often involves determining the relationship among as many stocks as possible in order to be thorough and achieve the best possible portfolio.

1.2 Correlation graphs

Correlation graphs are one way to discover the dependency structure among different stocks whose returns may be represented as random variables following some distribution. Let $G^{\text{num}} = (V, E)$ be an undirected graph with vertices $V_1, ..., V_d$ (a d-dimensional distribution) and edges $E_{i,j} \in \{0, 1\}$. We set $E_{i,j} = 1$ when there is an edge between V_i and V_j, and 0 otherwise. An edge is drawn between V_i and V_j iff the two random variables are correlated. This graph can be drawn from a correlation matrix $\sum$ where $\sum_{i,j} = corr(V_1, V_2)$ with the following heuristic:

If $\sum_{i,j} > p$, draw edge $E_{i,j}$ where p is the p-value for the desired confidence level

---

We differentiate between "visual correlation" (which can be thought more of as "pairs of variables that marginally appear dependent") and the more common mathematical interpretation of "correlation". More specifically, we would like to visually understand what a correlation graph looks like and compare it to correlation graphs constructed with the traditional interpretation of correlation. Two variables that have a correlation coefficient near 0 may not necessarily be uncorrelated. A scatter plot can reveal this by showing outliers or patterns in the data that the analyst wasn't expecting; this was seen in Figure 1.1 (right). But in order to confirm that the coefficient used applies to each potential relationship, the analyst must plot all possible sets of data, which we have already established as computationally infeasible and tedious to sort through in high dimensions. We will present a solution in Section 1.4. What follows is an overview of common numerical methods to estimate the correlation between two random variables.

1.2.1 Pearson's correlation

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

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

The formulation above is not as useful in practice as datasets are regarded as samples of a population. Given n observations, the sample expectation is given by the formula $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$. By substituting into the above and multiplying by n^2/n^2, we can estimate the Pearson's correlation with the following:

$$\rho_{x,y} = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\sqrt{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}\sqrt{n\sum_{i=1}^{n} y_i^2 - \left(\sum_{i=1}^{n} y_i\right)^2}}$$

such that $-1 \leq \rho \leq 1$. With perfect positive and negative linear dependence respectively, $\rho = \pm 1$. It is important to note that $\rho = 0$ 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 correlations may also capture non-linear dependencies. Spearman's correlation is computed by computing 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. We then define the difference of a sample (x_i, y_i) as $d_i = x_i - y_i$ and compute Spearman's correlation as

$$\rho_{x,y} = 1 - \frac{6\sum\limits_{i=1}^{n} d_i^2}{n(n^2-1)}$$

such that $-1 \leq \rho \leq 1$. With perfect increasing and decreasing monotonic dependence respectively, $\rho = \pm 1$. Again, it is important to note that $\rho = 0$ 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 $c =$ the number of concordant pairs and $d =$ the number of discordant pairs. Then Kendall's tau is computed as

$$\tau_{x,y} = \frac{c-d}{n(n-1)/2}$$

such that $-1 \leq \tau \leq 1$. Kendall's tau is less sensitive to errors in the data as its correlation is based on sample pairs rather than deviations within an observation, though the resulting values tend to result in the same interpretations. As with Spearman's correlation, $\tau = \pm 1$ with perfect increasing and decreasing monotonic dependence respectively. Furthermore, $\tau = 0$ 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-monotone dependence between X and Y [18]. The distance correlation is a function of the distance covariance and distance variance of X and Y. We define the $n \times n$ matrices a and b as 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 we define the distance covariance and distance variance as

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

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 distance correlation is then defined as

$$\mathcal{R}_{X,Y} = \frac{\text{dCov}(X, Y)}{\sqrt{\text{dVar}(X)\text{dVar}(Y)}}$$

such that $0 \leq \mathcal{R} \leq 1$. As before, $\mathcal{R} = 1$ indicates perfect dependence. However, the interpretation of $\mathcal{R} = 0$ is different, which is one of its two important properties [18]:

1. X and Y may be of different dimensions.
2. $\mathcal{R} = 0$ 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 stock picking is similar to that of "buy and hold", though there are some key difference. Another component for successful rebalancing gains is for the stock returns to be relatively independent [11]. 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. This 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 [11].

Another financial application is the creation of a predictive stock model. Predictive models are a form of "model selection" where dependence rather than independence is important. Model selection addresses the problem of determining which explanatory variables (commonly seen as columns of the matrix X) are informative to explain the variation in the response variable (commonly seen as the vector Y). This is also related to the concept of "sparsity," a statistical term referring to the fact that many coefficients of a fitted model should be 0. Model selection with sparsity aids in the interpretability of the model since there are fewer variables for data analysts to understand. This is another way to view rebalancing; the predictive model, if it is to be believed, can signal price swings to come which open up arbitrage opportunities. Furthermore, stocks which are uncorrelated are those with insignificant coefficients in the resulting model; this "negative space" point of view is partly utilized in the stock selection methodology for correlation graphs proposed in Section 5.2.

It must be noted that 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. An unassuming analyst might select the data as their correlation is not significantly different from zero (Table 1.1, Dataset 2), but it turns out that they have selected a stocks which we can see are highly correlated! However, it is still an important component of portfolio management in theory and in practice. Regardless, correlation graphs and their financial applications are not solutions to the two problems proposed in Section 1.1. Rather, correlation graphs and their financial applications are a concrete application of whatever the proposed solution is (A detailed application may be found in Chapter 5, and a stock selection methodology for correlation graphs is proposed in Section 5.2). What follows is a roadmap of the high-dimensional visualization solution that is developed further in this paper.

1.4 Summary

In this work, we tackle the problem by developing a sophisticated visualization system (abbreviated VS) to explore the data and numerical model in a different way. Specifically, we focus on two aspects of the VS which address the two problems raised in Section 1.1: (1) the procedure of sorting through plots is efficiently automated by learning the user's interests, and (2) the procedure of comparing the numerical and visual output is also automated. 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 then focuses on the active learning stage of the VS, which is part of the first solution to the problems 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 second solution to the problems raised in Section 1.1. The chapter is concerned with the VS output, which quantifies the differences between numerical and visual correlation graphs for the user. Subsequently, the chapter details various graph comparison methods and ultimately selects one for usage in the financial application in Chapter 5. Finally, Chapter 6 recaps the work and presents future extensions of the VS.

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 high dimensional visualization analysis; it is simply a question of what degree [12]. It is not enough to simply have user interaction, however. Given d variables, there are a total of $\binom{d}{2}$ possible scatter plots of the data. This blows up almost quadratically as d 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 thousands of possible plots and returns an adjacency matrix that captures the classifications of variable pairs. The VS may then return several plots for the user, perform line-up tests to refine the tree 6.1.5, or compare the visual graph with a numerical graph of the user's choice. This allows users to compare and contrast visual feedback with numerical 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 plot; mindfulness and incorporation of these ideas in the VS facilitates improves user accuracy in stage 1 . Sections 2.2 and 2.3 provide a brief overview of stage 1 and stage 2 respectively (see Figure 2.1) while Section 2.4 describes the focus of the rest of the work which respond to the problems posed in Section 1.1.

2.1 Scatterplot characterization

2.1.1 Characteristics of a "good" plot

The simplest scatterplot is the response against the observed variables. 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.

Restricting the 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].

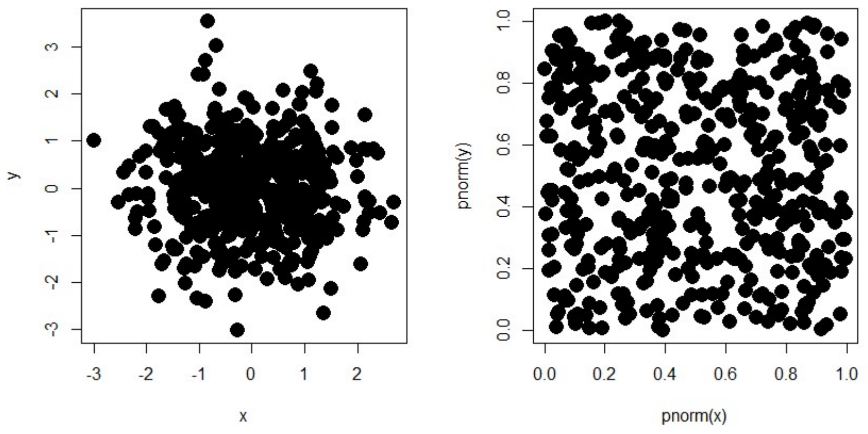

Figure 2.2: A plot of y against x with no transformation (left) and after the CDF is applied in both directions (right). The code for this example may be found in Appendix A.3

The effects of this can be progressively observed by looking from the left to the right in Figure 2.3 below.

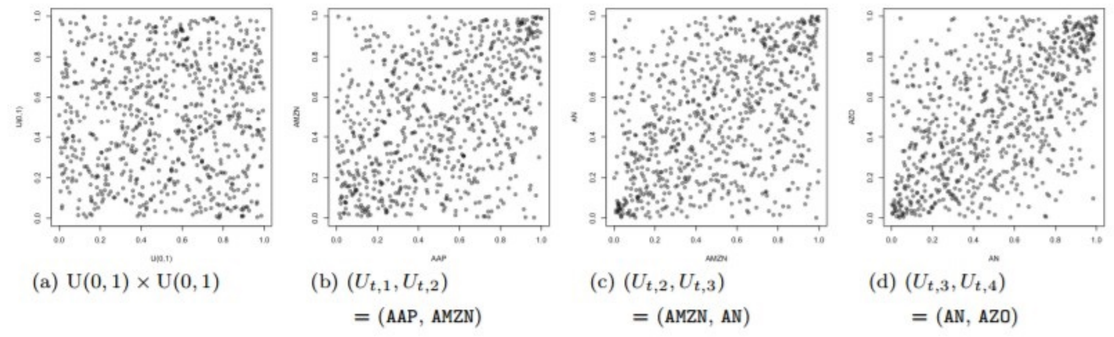

Figure 2.3: Scatterplots of (a) independent $U(0, 1)$ random variables and (b,c,d) the pseudo-observation pairs $(U_{t,j}, U_{t,j+1}), j \in \{1, 2, 3\}$. Ticker abbreviations: AAP = Advanced Auto Parts, AMZN = Amazon.com Inc, AN = AutoNation Inc., AZO = AutoZone Inc. Figure from Hofert and Oldford 2016 [8]

2.1.2 Feature extraction from plot

In order for the active learning classifier to properly understand and classify all $\binom{d}{2}$ plots (Sections 2.2 2.3), the features must be extracted from each 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:** See Section 1.2 for more details on the various types of quantifiers
- **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:** A higher value suggests a greater visual trend. This is computed by max(PosTrendCriterion, NegTrendCriterion) which is computed from the percentage of points in the upper left and bottom right, and bottome left and upper right, respectively

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 (which are n characterizations of $\binom{d}{2}$ plots in the case of the VS as described in Section 2.1.2) for the analyst (the "oracle") to label (classify). 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 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 (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). As such, it is important to make the 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 [5]:

- **Supervised learner**: This learner queries a single, 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 to reduce the hypothesis space $\mathcal{H}$ of all possible classifiers that may explain the data.

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 [17]. Chapter 3 goes into detail on different active learning methodologies as this section is primarily focused on its role in the system.

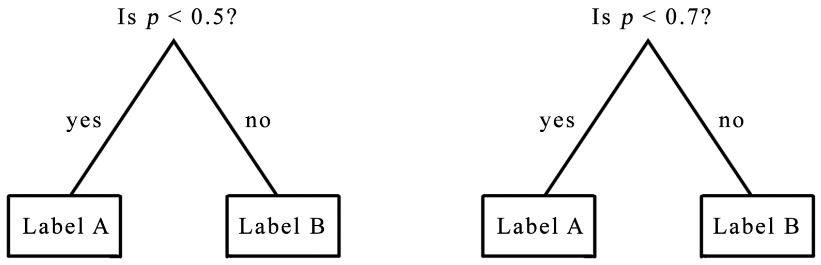

Figure 2.4: Although both figures on the left and right are slightly different classifiers, they are both instances of (extremely simple) decision trees, a type of classification model. Other models include discriminant analysis, naive bayes, random forest, logistic regression, etc. See Section 2.3.1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

Alternatively, we can exploit the fact that the user is already providing a numerical model that they believe to be a good representation of the data which they would like the visualization system to check visually. Given this data, the system may build a classifier that utilizes the various properties of the plots to determine whether one is interesting or uninteresting. 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 must check whether the initial classifier is a proper fit (This may be achieved with line-up tests, which are described briefly in Section 6.1.5). As the user then proceeds to label various conditional plots (queried by the active learner) as "interesting" or "not interesting," the learner better understands the users interest, and the final 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 users interests. The system first determines which features it is uncertain about classifying and then returns a plot matching those characteristics to the user. This allows the system to utilize its classification model of choice to build a better classifier more efficiently. **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 initialization and actively selected queries in order to build a model to classify the user's interests and 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, all of which are described more thoroughly in Chapter 3.

2.3 Automated plot generation (Stage 2)

2.3.1 Decision tree classification of user interests

Given initialization and actively selected labeled data, what classification models can the visualization system use to create a final fitted model of user interests? 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.* 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)

$$G = N_{left} \sum_{k=1}^{K} p_{k,left}(1 - p_{k,left}) + N_{right} \sum_{k=1}^{K} p_{k,right}(1 - p_{k,right})$$

where N_s is the number of nodes in side s of a split ($s = \{left, right\}$ in a vertical split, and $s = \{top, bottom\}$ in a horizontal split) and $p_{k,s}$ is the fraction of class label k on side s of the split. Decision trees are a more sophisticated classification method than simple linear regression and retains interpretability. The root node is, naturally, 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,

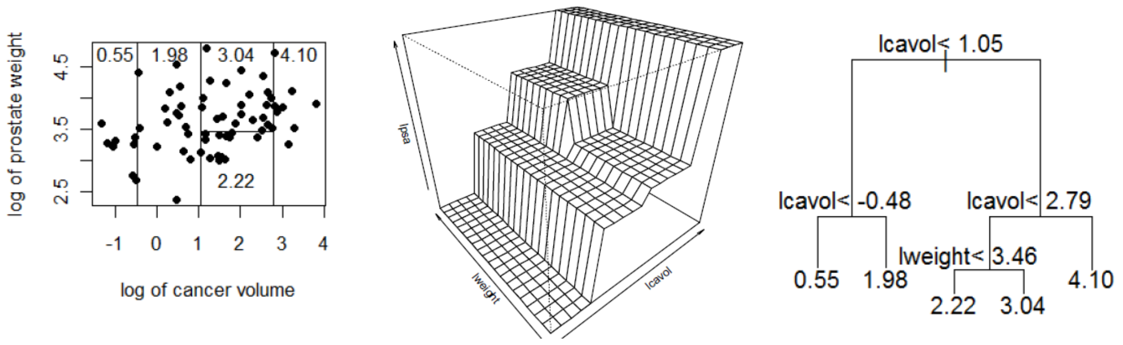

Figure 2.5: Mapping the sample space (left) to a decision tree (right). Images from Cutler [3].

A *random forest* provides solutions to the problems that a single decision tree faces. In a random forest, a forest is composed of many trees "grown" from random partitions of the labeled set (the training set). Furthermore, each decision tree is constructed by finding the best split among $m \in M$ splits. 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 for each unlabeled data, 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 [22]. Utilization of such a methodology aids in user interpretability of the system output while maintaining the robustness of a forest. 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 $\binom{d}{2}$ unlabeled scatterplots, and a visual graph $G = (V, E)$ may be built from these labels with the following heuristic where $i, j \in \{1, ..., d\}$:

If label$(i, j) > 0$ (i.e. "interesting" instead of "uninteresting"), draw edge $E_{i,j}$

Furthermore, the VS may provide a visualization of the resulting classifier itself 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 each pair of variables as colors from a bivariate spectrum. Heat maps are difficult to interpret as it is one-dimensional; each end of the color spectrum represents minimum and maximum values respectively. It is unclear what the max or the min is as it depends on the domain of the whatever 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 relative ranking among different pairs for colors that are similar. 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 color corresponds to its value while the color represents a positive or negative value [2]. As such, the association navigator only utilizes two colors rather than a spectrum of colors, making it simple to distinguish between the two. By simplifying the color scheme and adding the dimension of size, the association navigator makes it much easier to interpret and compare different pairs of data at a glance. The stark difference between a traditional heatmap and the association navigator when applied to the same dataset may be seen in Figure 2.6. The VS provides both options, allowing the analyst to select whichever is easier to for them to interpret. With these visualizations of the active learning output, the user should be able to understand his/her own interests.

Figure 2.6: Heatmap versus association navigator example.

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 an analyst's final report, 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 numeric graph (For example, a correlation graph G^{num} generated with numerical correlation coefficients as described in Section 1.2) and measures the difference between the numeric graph G^{num}

and visual graph 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 numerical graph selection strategy may 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. For more details on this methodology, see Section 6.1.5.

With the focus on various correlation graphs in this work, 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: AL and GC

The active learning component is the bread and butter of the visualization system and forms the first stage of the system. Furthermore, it solves for the tedious nature of classifying $\binom{d}{2}$ plots in high-dimensional visualization , which was first brought up in Section 1.1. The second issue with high-dimensional visualization is the verification of the numerical results against the visual result. This is a question of the VS output, and graph comparison is a solution to this problem that is both informative to the analyst and useful in selecting the most relevant numerical method where large differences encourage the analyst to investigate whether their selected numerical method was truly appropriate for the dataset at hand. As such, the graph comparison output is most useful (as opposed to plot generation and line-up tests) in the case of numerical correlation graphs, which we are interested in for their financial applications. As such, the primary focus of this work that follows are active learning methods (Chapter 3) and graph comparison methods (Chapter 4). These methods are applied to the current iteration of the VS, which is then utilized in conjugation with numerical correlation graphs to perform stock selection.

It should be noted that the VS is a large project and may certainly be further refined despite the work that we do to refine two major components of the system. Ideas for future extensions of the VS may be found in Section 6.1. There are several recommendations for improvements that can be made to different 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 data. Consider a large set of unlabeled data X 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 choose its next query, it performs better with less training [17]. 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 $Y \in \{\text{interesting, non-interesting}\}$. Classification and filtering tasks are both tedious and redundant, which necessitates intelligent selection of queries [17]. 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 equity application.

3.1 Literature review

It is important to consider the situation in which the learning algorithm receives the points in which it selects a query from. There are three different scenarios in which the learner may request queries as described by Settles [17]:

1. **Membership query synthesis**: The learner may select a query from an 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 $\binom{d}{2}$ plot (Section 2.1.2), membership query synthesis is not an option. While stream-based selective sampling may work, it has a similar problem as membership query synthesis because there are no constrains 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 methodologies 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 drivers 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 the labels to its neighbors.

Uncertainty sampling and query by committee 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 cluster 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 the implementation code, which can be found in the appendix. The `activelearning` package by `ramhiser` [15] has adapted much of the methods reviewed in Settles' work [17]. As such, most of the implementation code has been adapted and reworked (with the inclusion of substantial parts 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 labels such as the VS system (either "interesting" or "non-interesting"), this reduces to the case of querying the instance whose posterior probability of being "interesting" is closest to 0.5 [9]. While the algorithm presented later 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 [17].

Uncertainty sampling is simple but not without problem. Given that the number of possible classification labels is $k > 2$, 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 above (selecting q with a posterior probability closest to 0.5) when $k = 2$ [17].

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 [17])

```
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:     $tout \leftarrow$ train($X^{labeled}, y^{labeled}$, classifier)
3:     $p \leftarrow$ predict($tout, X^{unlabeled}$, posterior prob. $= TRUE$)
4:     loop from $i = 1$ to len($p$):
5:         $p_i \leftarrow |p_i - 0.5|$
6:     return 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 "interesting" or "non-interesting". 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 hypotheses, and its prediction for an unlabeled query candidate q is a "vote" of weight one on q; the most disagreeable candidate is then selected [17]. Settles reviews various methods for committee selection for both probabilistic and non-probabilistic models, though the implementation of the algorithm (Appendix A.5) allows the user to specify the committee members [17].

We may write the basic algorithm as follows:

Algorithm 2 Query by committee (as described by Settles [17])

```
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)
2:     loop from $i = 1$ to len($C$):
3:         $tout_i \leftarrow$ train($X^{labeled}, y^{labeled}, C_i$)
4:         $p_{i,} \leftarrow$ predict($tout_i, X^{unlabeled}$)
5:     $d \leftarrow$ disagreement($p$)
6:     return where($d == \max(d)$)
```

While it is simpler to constantly 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 it is 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 return the index of q_t. Subsequently, the committee from t is pruned at time $t + 1$ at the start of the algorithm using the retrieved label from time t. It should be noted that 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 implement the pruning algorithm after $iter/2$ queries where $iter$ is the maximum number of queries allowed. Furthermore, the query by committee methodology described by Settles is only focused on selection of q independent of the final classification model once the rest of the training data is selected [17]. However, there is merit in maintaining the final pruned committee as its own classification model after the querying is complete and the training data is selected. This may be achieved by selecting labels on unlabeled instances via majority vote. Further details on implementation and performance may be seen in the simulation study (Section 3.3). The revised selection algorithm is as follows (see Appendix A.5 for code):

Algorithm 3 Query by committee (revised framework)

```
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, $E$ be the error ratio of the respective committee members (initialized to 0), $0 < \epsilon < 1$ be some threshold for the error ratio, and $t$ be the current iteration of QBC starting at $t = 1$)
2:     function QBC
3:         loop from $i = 1$ to len($C$):
4:             $tout_i \leftarrow \text{train}(X^{labeled}, y^{labeled}, C_i)$
5:             $p_{i,} \leftarrow \text{predict}(tout_i, X^{unlabeled})$
6:         $d \leftarrow \text{disagreement}(p)$
7:         return $j = \text{where}(d == \max(d))$
8:     function ORACLE
9:         return $l$, the label of $X_{j,}$
10:    $y_j \leftarrow l$
11:    function PRUNE (Let $iter$ be the total active learning budget. When iteration $i \in [1, iter] > iter/2$, run PRUNE )
12:        $prune = [\ ]$ (empty vector)
13:        loop from $i = 1$ to len($C$):
14:            If ($p_{i,j} == y_j$) then $iv = 0$ Else $iv = 1$
15:            $E_i = E_i + \frac{iv - E_i}{t}$
16:            If $E_i > \epsilon$ then $prune$.append($i$)
17:        $t++$
18:        return $prune$
19:    loop from $i = 1$ to len($prune$):
20:        Delete $E_{prune_i}$, $C_{prune_i}$, $p_{prune_i,}$, and $tout_{prune_i}$
```

The algorithm contains a generic disagreement function, so it is important to note that there are different measures of disagreement among the committee members. There are two main methods of disagreement measurement described by Settles [17]:

1. **Vote entropy**: The disagreement for variable d is computed

$$x^*_{VE} = \arg\max_x - \sum_{j \in \text{all possible labels}} \frac{V(y_j)}{\text{len}(C)} \log \frac{V(y_j)}{\text{len}(C)}$$

   where $V(y_j)$ is the number of votes each possible label y_i for d received. The interested reader may refer to [4] for further details on this methodology.

2. **Kullback-Leibler divergence**: The disagreement is computed

$$x^*_{KL} = \arg\max_x \frac{1}{\text{len}(C)} \sum_{i \in 1}^{\text{len}(C)} \sum_{j \in \text{all possible labels}} \text{P}_{C_i}(y_j|x) \log \frac{\text{P}_{C_i}(y_j|x)}{\text{P}_C(y_j|x)}$$

   Since C was defined as the committee, we can interpret $\text{P}_C(y_j|x) = \frac{1}{\text{len}(C)} \sum_{k=1}^{\text{len}(C)} \text{P}_{C_k}(y_j|x)$ as the probability that the label with the most votes will be y_j where $\text{P}_{C_k}(y_j|x)$ is the probability that committee member k votes y_j for variable d.The interested reader may refer to [13] for further details on this methodology.

An implementation of vote entropy disagreement can be found in Appendix A.6. This function was imported from the `activelearning` package developed by `ramhiser` [15] 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 of committee of classifiers trained on random (weighted, in the case of Boosting) samples of the labeled data [1]. Bagging, quite simply, uniformly samples k training sets from the labeled data to form a committee of k classifiers trained on the same classification model [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). Thus, the algorithm is as follows (see Appendix A.7 for code):

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

```
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. Let $r \in (0,1)$ such that $r * \text{len}(X^{labeled})$ rounded is the number of points to randomly sample from the labeled set.)
2:     loop from $i = 1$ to $num\_class$:
3:         $idx \leftarrow \text{unif\_sample}(labeled, r * \text{len}(X^{labeled}))$
4:         $tout_i \leftarrow \text{train}(X_{idx,}, y_{idx}, \text{classifier})$
5:         $p_{i,} \leftarrow \text{predict}(tout_i, X^{unlabeled})$
6:     $d \leftarrow \text{disagreement}(p)$
7:     return where($d == \max(d)$)
```

Since each iteration of the active learner is a new process of random committee training and selection, there is no need to (1) maintain error ratios to prune the starting committee or to (2) use majority vote for the final fitted model over a random forest, which the VS utilizes 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 [20]. This can be done by selecting query that maximizes the minimal distance of each unlabeled query from the labeled set [20]. In other words,

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

Given natural clustering in the data, the active learned would be able to select a healthy sample from each cluster with this methodology with the hope that each cluster is fairly homogeneous (Section 3.1). As such, the methodology 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 datasets 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? This is important because selecting points near the centers of the clusters, the densest regions, allows the algorithm to converge faster [20].
- **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.* has 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. Since the VS will be initialized randomly, we do not concern ourselves too much with min-max clustering initialization; the interested reader may refer to [20] 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.* [20])

```
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:     loop from $i = 1$ to len($y^{unlabeled}$):
3:         min $\leftarrow \infty$
4:         loop from $j = 1$ to len($y^{labeled}$):
5:             $d \leftarrow$ distance($X^{unlabeled}_{i,}, X^{labeled}_{j,}$)
6:             If (min $> d$) : min $\leftarrow d$
7:         $q_i \leftarrow$ min
8:     return where($q ==$ max$(q)$)
```

3.3 Simulations

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. 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 k data that is 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 when 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 classification, as well as (1) with pruning and (2) without pruning.
- **Bivariate classification:** The classification of user interests have two possible labels/levels: "interesting" and "not interesting". 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).

3.3.1 Data

The data is taken from the MNIST database of handwritten digits (`http://yann.lecun.com/exdb/mnist/`). The digits $(0, 1, ..., 9)$ have already been classified 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 computation efficiency, the data has been further compressed to a 196-length vector (14×14 pixel image). In order to maintain the condition of bivariate classification, two out of ten digits were selected. The digits 7 and 9 were selected as they are similar visually, 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 very different on the visual plane). 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 make up the final data. The final dataset may be visualized in Figure 3.1. The functions for working with the MNIST data were adapted from file `gist:39760` [14] and may be found in Appendix A.9.1.

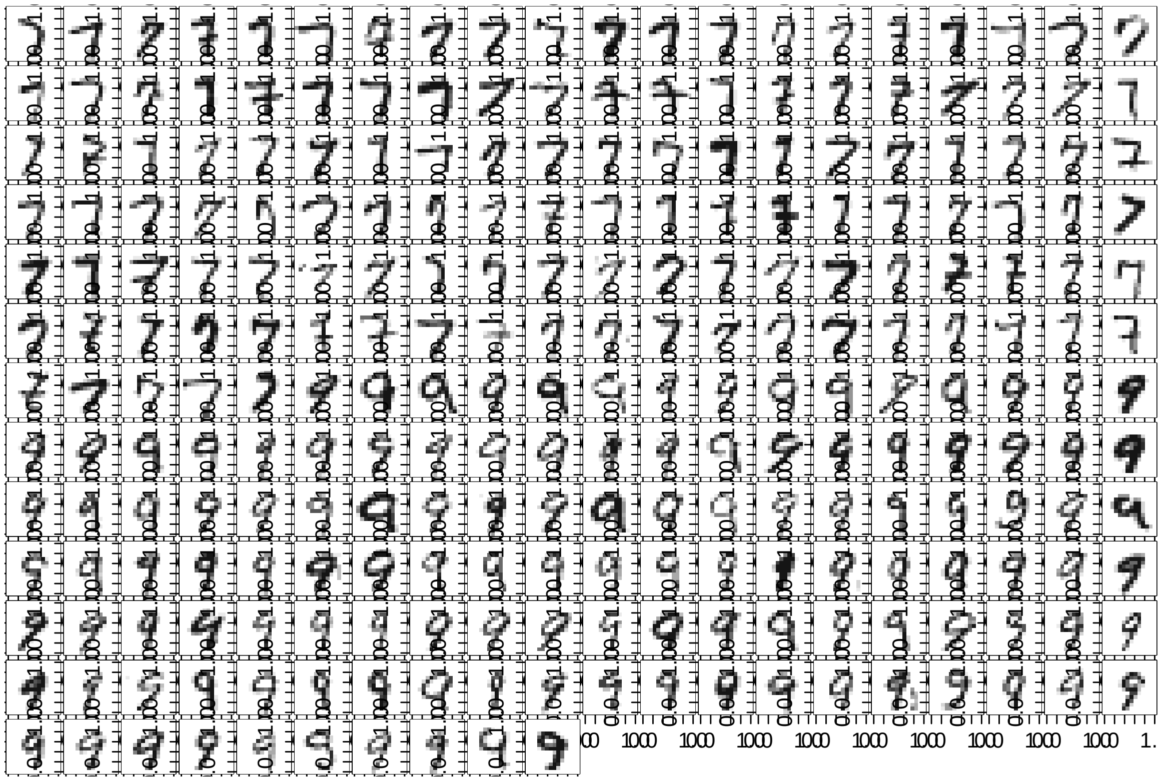

Figure 3.1: MNIST data used in the simulations. The digits 7 and 9 are similar visually, making the simulations more realistic as it is harder for the classification model to achieve a "perfect fit" (given only the initial labeled) without the use of querying.

3.3.2 Evaluation

The performance of a learner at any given point in time (at any iteration i of simulation number $k \in [1, 25]$) is encapsulated in its error ratio ϵ. That is, given the current labeled set, the **main classifier (Random Forest model)** is trained, and the entire dataset is predicted given the current classifier. The predictions are stored in an n-length vector p. We know the true labels ahead of time (thanks to the MNIST dataset), and they are stored in an n-length vector y'. The predictions are compared against the true labels and the error ratio of iteration i is given by

$$\epsilon_i = \frac{1}{250}\left(\sum_{z=1}^{250} \mathbf{1}_{p_z \neq y'_z}\right)$$

where $\mathbf{1}_{p_z \neq y'_z}$ is an indicator variable that is 1 when $p_z \neq y'_z$ and 0 otherwise. These 50 error ratios then form one vector ϵ_s, the result of trial s. To select the best active learning algorithm *on average*, each algorithm's error ratios are averaged over 25 total trials. This helps offset the randomness of the initialization and pooling scheme. Then the final error ratio of an algorithm is given by

$$\epsilon = \frac{1}{25}\left(\sum_{i=1}^{25} \epsilon_s\right)$$

The final framework for computing ϵ_s, a single trial'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 ϵ_s, a single trial's vector of error ratios

```
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). $y'$ is an $n$-length vector of true labels for each variable in $X$. $iter$ is the maximum number of queries allowed per trial)
2:     loop from $i = 1$ to $iter$:
3:         $idx \leftarrow$ active_learning_method($X, y, ...$)
4:         QUERY $X_{idx,}$
5:         $tout \leftarrow$ train($X^{labeled}, y^{labeled}$, randomforest)
6:         $p \leftarrow$ predict($tout, X$)
7:         $res_i \leftarrow \frac{\text{length(which}(p \neq y'))}{\text{length}(y')}$
8:     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 tuning parameter values. Random sampling is used as the active learning method's control as a point of comparison; 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 is the main classification model that is used to fit the error, not the classification model(s) used in the active learning methods (those are parameters). The classification model in the simulator is akin to the main classification model in 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
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

(Continued on next page)

Table 3.1: (continued)

AL method	**Simulations**	**Main classification model**	**Parameters**
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 proposed addendums to the QBC method (Section 3.2.2) 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 naive QBC methodology as detailed in Algorithm 2). A general committee was randomly selected which worked with the data. For example, any form of discriminant analysis, which utilizes matrix inversion, does not work without data cleaning. 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. To avoid further tampering with the dataset (as the data has already been compressed), LDA, DDA, FDA, etc. were not included in the starting committee. A brief overview of each classification model used in the starting 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 uses Bayes' Theorem $\left(P(A|B) = \frac{P(B|A)P(A)}{P(B)}\right)$ 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 SVM, each data point in X is mapped to a space in $\mathbb{R}^2$, linearly demarcating each category with a line that is as distinct as possible (i.e. the two categories are as far apart as possible). 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 PLS, a *principle component analysis* is first performed on the data X. Principle component analysis takes a n-length set of variables X (which may or may not be correlated) and converts it 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 AL engine code in Appendix A.9.3. By implementing the active learning call in this way, the main AL 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 (Appendix A.9.2) 25 times for each active learning method (the simulation engine calls the active learning engine, Appendix A.9.3), and averages then plots the results may be found in Appendix A.9.4.

Figure 3.2 is a summarization of the performance of query by committee given different parameters. With query by committee, majority committee vote outperforms random forest on average in the first half (25 iterations) as the main classification model. What is interesting to note in iteration 25 onwards is that the error ratio for both majority committee vote 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

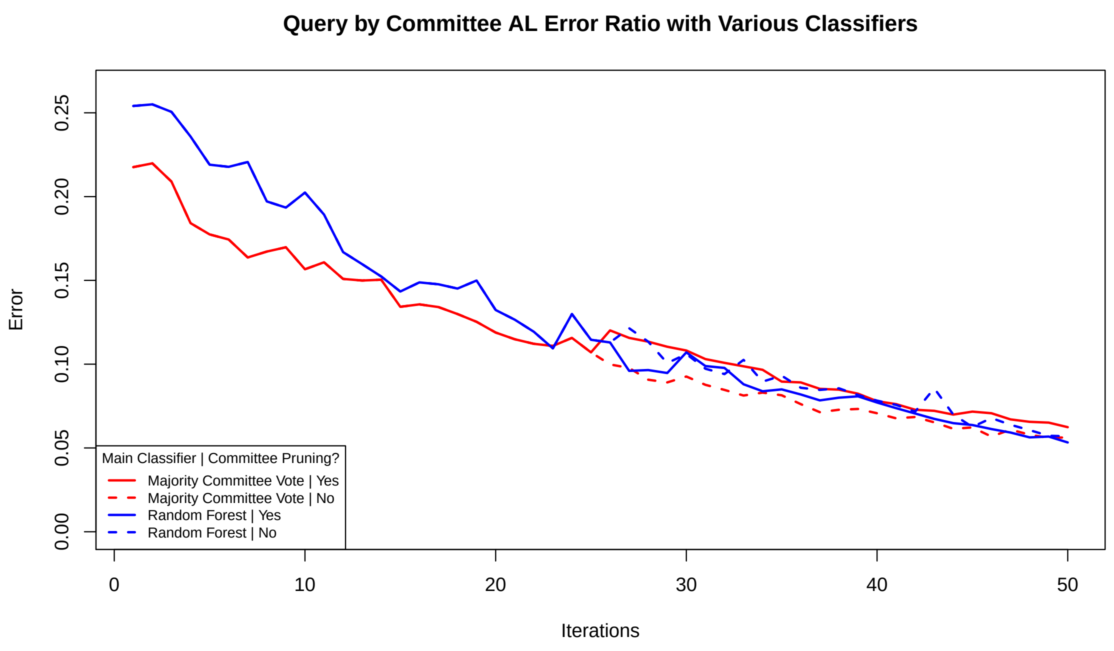

Figure 3.2: Active learning simulation results for Query by Committee with varying parameters. Each of the 25 trials were consistently seeded with its own value, so the values before committee pruning would/would not have occurred (at $iter/2 = 25$) are the same.

than 0.5), are valuable. Naturally, this phenomenon cannot be observed in the random forest results because the classification model is a random forest; since committee members are not weighing in on the predicted labels, the members themselves hold less weight. Although majority committee voting with no 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 start converging faster and just barely overtake majority committee vote at the end. In other words, QBC with majority voting starts off better but improves more slowly while QBC with random foresst starts off worse but improves more rapidly. Each method has its own trade-offs, and 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.

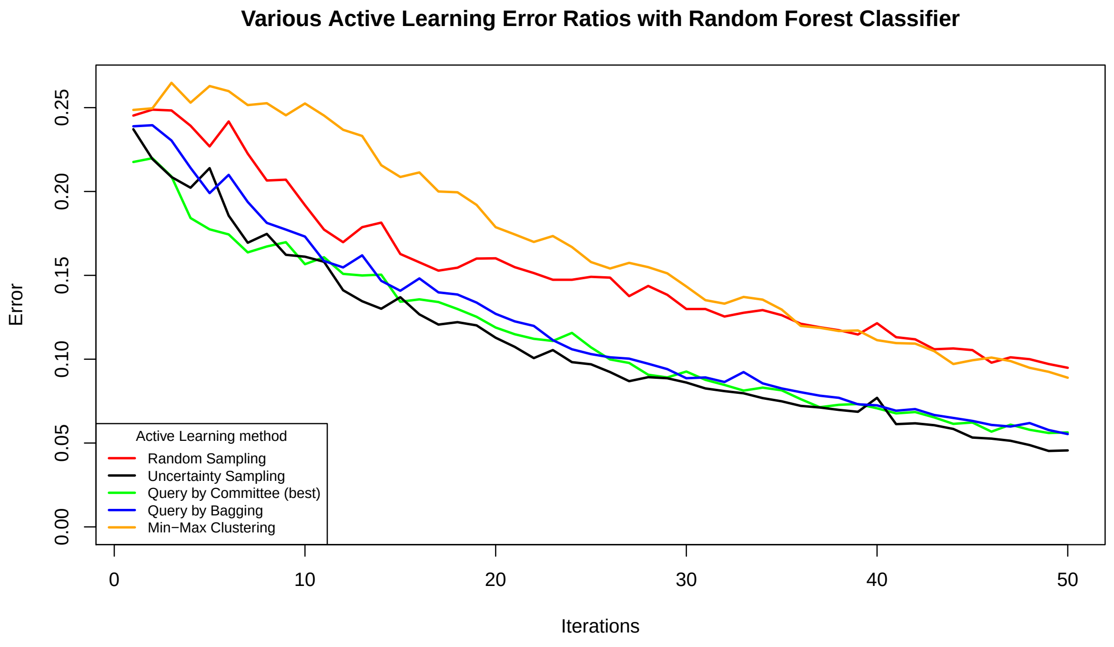

Figure 3.3: Aggregated active learning simulation results.

Figure 3.3 summarizes the performance of the "best" QBC method against the other active learning methods along with random sampling as a control (a baseline for comparisons). 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 each 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 with points within clusters, the min-max methodology allows the learner to further query within clusters (to sample more labels) without having to start by searching through the entire space for the 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.* [20], the min-max clustering algorithm would have begun with better performance.

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 five random forest classifiers (these parameters are summarized in Table 3.1) and also uses a random forest for the main classification model.

There appears to be another "tie" for query by committee and uncertainty sampling. QBC starts off better, but the faster convergence in uncertainty sampling eventually outperforms QBC. Unlike the case of QBC with a random forest versus majority committee vote classification model (Figure 3.2), uncertaintly sampling overtakes QBC much earlier. To be more specific, QBC is overtaken near the tenth iteration. Thus, out of all methods described in Table 3.1, **uncertainty sampling** most closely embodies the qualities that characterize a strong active learner: intelligent selection of queries that allow the main classifier to learn interests more quickly.

Figure 3.4, which shows performance plots with confidence intervals for each active learrning method, may also be of interest to the reader.

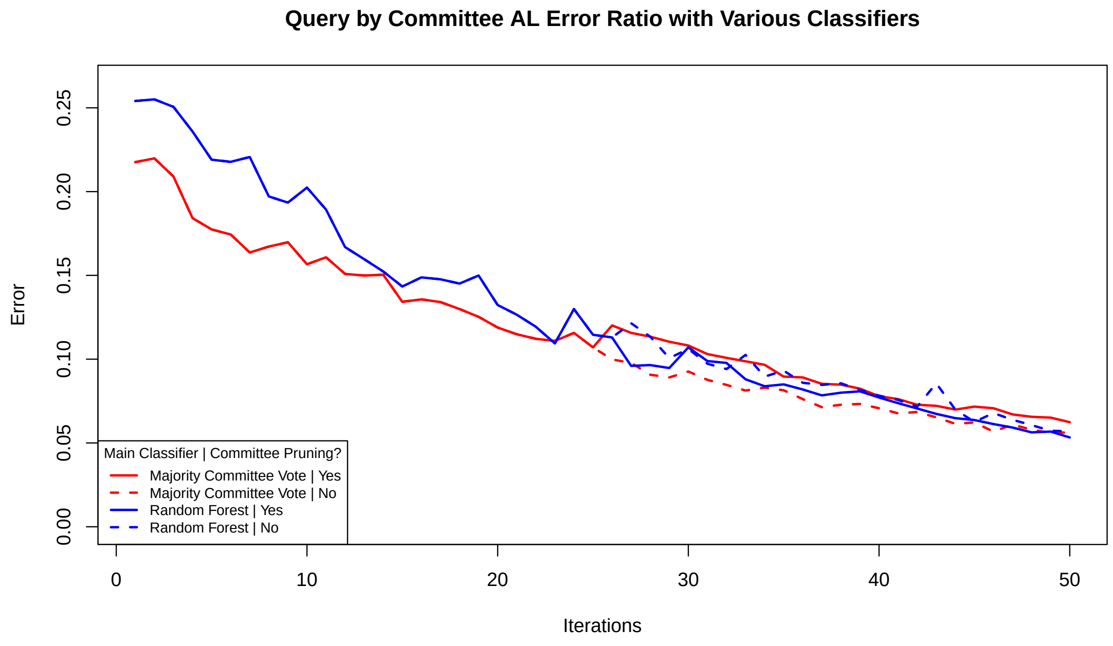

Figure 3.4: Active learning simulation results with confidence intervals.

Extensions

Naturally, there are many other parameters that could have been tested, which were not tested in this simulator in the interest of time. For example, the classification model in the active learning method (not the overall method) to aid in query selection does not have to be a random forest (This applies to US and QBB); perhaps another would have worked better despite the intuition that, if the active learning methods optimizes over the same classification model that the main program will use, the learner will perform better. As noted earlier, initialization is important for min-max clustering, and the performance of the learner may 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 QBC and QBB). The tuning parameters of ϵ and r in the query by committee/bagging methods could also be perturbed to see the effect on error ratio. Finally, different committees could be fed into the QBC method, and different starting points for pruning (i.e. $iter/3$ or $iter/4$ instead of $iter/2$) could also be tested. It may also be of interest to dig into why the active learner suddenly does worse the moment a committee is pruned. The reader is invited to pursue these as further extensions of this work should they wish to.

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 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 reevaluate their numerical methods by selecting 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. various stocks). 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.

4.1 (not actually a) Literature review

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 $\overrightarrow{v^1}$ on $\hat{G}^1$ 1 that "summarizes" some property it has. Similarly, compute $\overrightarrow{v^2}$ on $\hat{G}^2$ to capture the same property that $\overrightarrow{v^1}$ does. Finally, compute $f(\overrightarrow{v^1}, \overrightarrow{v^2})$ 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 $(\overrightarrow{v^1}, \overrightarrow{v^2})$. Compute $f(\hat{G}^1, \hat{G}^2)$. Again, a high value indicates strong dissimilarity while a low value indicates similarity.

We focus on the graph summarization comparison method because it fulfills both desired criteria for the visualization system. To be more specific, we seek to better understand existing graph summarization methods: the qualities that each metric reveals about a graph and associated advantages and drawbacks. Understanding these metrics is important due to the complex nature of graphs. It is natural to ask why one cannot plot the graphs themselves (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; it is unclear what the best arrangement might be at the start, and it is unfeasible for an analyst to simply 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 as each indicates a relationship which isn't quite like all the others.

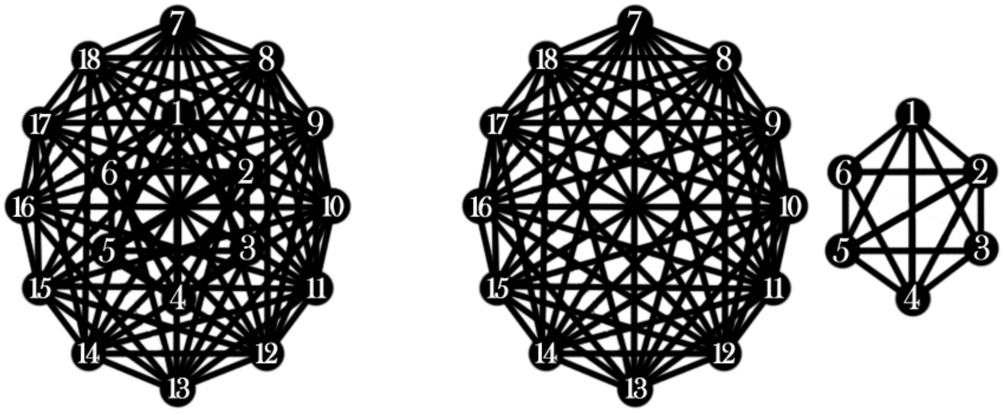

Figure 4.1: Suppose we have a cluster graph $G = (V, E)$ with clusters C^1, C^2 where $|V| = 18$. Let C^1 be composed of nodes $(V_1, ..., V_6)$, and C^2 be composed of nodes $(V_7, ..., V_{18})$. *Left:* C^1, the smaller cluster, has been placed inside C^2, the larger cluster. It is difficult to discern 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 in the first place! It just so happens that $E_{13,17}$ is the missing edge. C^1 is more sparse, which makes it easier to visually perceive anomalies in the data.

Graph summarization acts as a "proxy" for visually exploring the graph itself, and each metric can be "parsed" and combined afterwards 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 which enable this level of analysis 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 VS allows the user to use visual qualities of a plot (many plots) to better understand numerical methods while graph summarization allows the user to numerical methods to better understand the visual qualities of a graph.

With an understanding of the properties of various graph summarization methods, the graph summarization method results may be better interpreted. The method computes a list of metrics (l^1, l^2) for each graph in order to provide a single vector $\overrightarrow{d^{1,2}}$ that quantifies the difference between the two graphs with some function (e.g. Euclidean distance, L2, etc.). As such, a natural byproduct of the computation is (l^1, l^2), which (armed with an understanding of the values) provide insights on the qualities of the visual graph and numeric 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 "best" 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 properly, so a normalization method is proposed for each metric difference. In section 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 in Chapter 3. The code for all methods may be found in Appendix A.9.5. It should be noted that there are various distance functions that could be used. Euclidean distance, L1, and L2 are currently implemented (The code for the distance functions may also found in Appendix A.9.5).

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

4.2.1 Centrality

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

$$c = \sum_{i=1}^{len(V)} \max_{\forall j \in \{1,\ldots,len(V)\}} \overrightarrow{c_j} - \overrightarrow{c_i}$$

The importance of the 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 $\mathcal{C}$.

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 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 the level of a node and not a graph, this issue would be accounted for (though normalization is not as trivial). It is more difficult to parse a node-level metric especially given many variables, so this paper does not delve into 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 an edge $E_{i,j}$, the degree of nodes V_i and V_j are collected in two distinct vectors $\overrightarrow{v^1}, \overrightarrow{v^2}$. Then the assortativity metric of a graph $\hat{G}$ is $\mathcal{A} = \rho(\overrightarrow{v^1}, \overrightarrow{v^2})$ where ρ 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 standardized. 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.

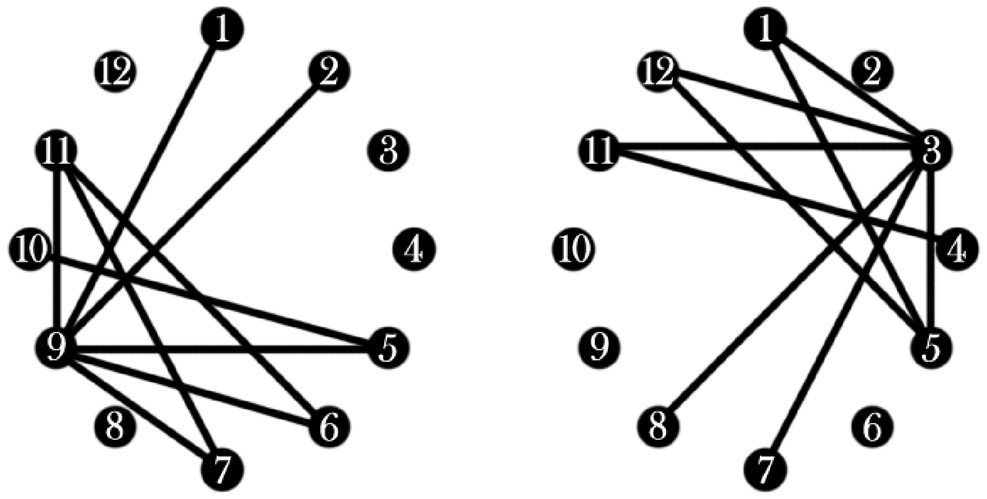

Figure 4.2: A graph pair with rotated edges. Graph summarization differences (L2 difference function):

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

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), the random walk procedure takes advantage of 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 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.
- **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 between them. As such, edges which connect separate subgraphs should 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 real way or need to normalize $\mathcal{S}$ as 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 $\hat{G}^2 = (V^2, E^2)$ via the community graph summarization metric (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 set labellings, but they are numbered differently; it is clear, then, that a traditional distance metric such as Euclidean distance will not capture the relationship properly. The Jaccard distance compares the actual sets rather than their literal values. The metric is already normalized between 0 and 1. The Jaccard distance is obtained by subtracting the Jaccard similarity, which is computed by the `cluster_similarity` function in the `clusteval` package, from 1.

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. This increase is extremely significant when considering that the "best case" scenario where the community (6,7,8,9,10,11,12) from Figure 4.3 (left) is completely split into (6,7,8,9) and (10,11,12) Figure 4.3 (right) yields a Jaccard difference of 0.3870968. 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.

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 pairs of variables. Each element $M_{i,j}$ contains the shortest path distance between node i to node j. If no path exists, it is conventional to set the value to infinity. However, since the main 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 earlier to be *the path which traverses the smallest number of edges*, so a *direct path* between nodes has path length 1. Thus, 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.

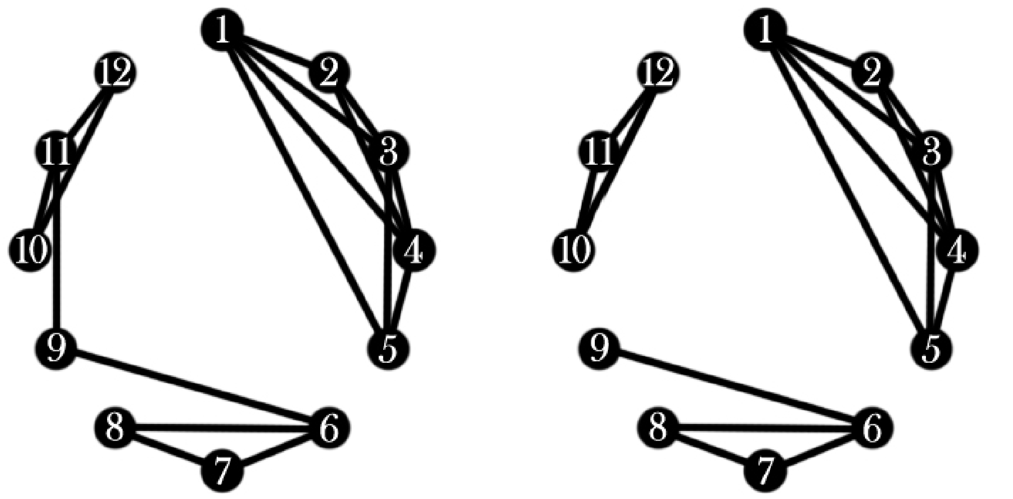

Figure 4.3: A graph pair with two and three clusters, respectively. Only one edge has been removed, so the graphs do not appear that extremely different. Graph summarization differences (L2 difference function):

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

Furthermore, the entries across the diagonal must be 0. Thus, the matrix can be unfurled into a vector $\overrightarrow{m}$ with the values of all $\binom{|V|}{2}$ unique pairs. It is less trivial to normalize the distance matrix difference metric. It would be extremely incorrect, for instance, to normalize by dividing the raw distance by $z = dist\big((|V|-1), 0\big) * len(\overrightarrow{m})$ 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 bound 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 clearly node-specific as the metric relies on the shortest paths between all unique pairs of nodes. As such, edge rotation is not a problem, but clustering becomes an issue when there are few clusters. Consider Figure 4.4; one graph has two physical clusters and the other doesn't (but appears to be clustered visually). The removal of edge (4,5) forces the shortest path between node pairs involving one from each cluster to drop to 0, having a strong effect on the distance matrix distance since everything is now interconnected. The actual distance matrix difference for the graph pair pictured in Figure 4.4 is 0.7708, which suggests that the graphs in the pair are very different when they are actually quite similar visually as 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 values will change to 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.

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. Then, the difference normalization will naturally follow. The edge connectivity of a graph is computed with the `edge_connectivity` function from the `igraph` package.

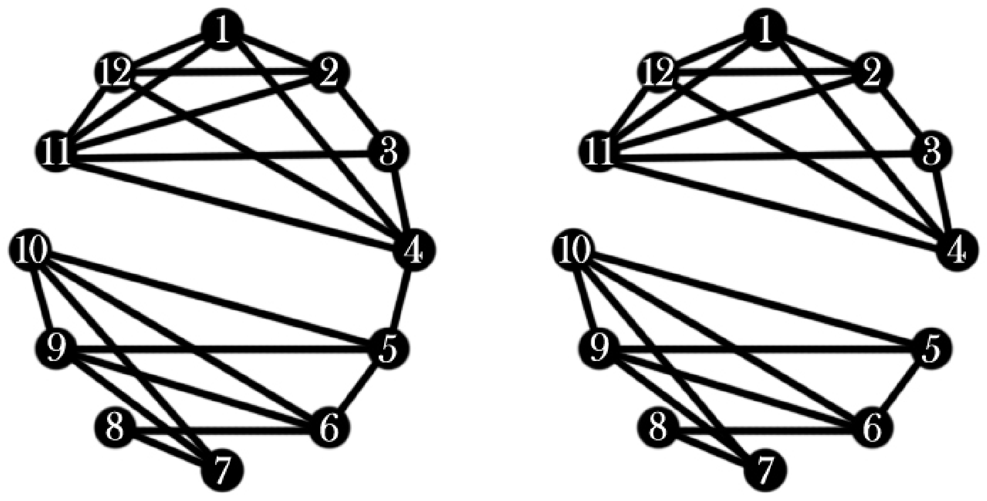

Figure 4.4: A graph pair with one (no) and two clusters, respectively. Only one edge has been removed, so the graphs do not appear that extremely different. Graph summarization differences (L2 difference function):

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

It is natural to see where this metric could fall through the cracks as it also is unaffected by 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 exactly same!

4.2.6 Edge density histogram

Edge density 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 is a signal for density while a larger number of low-degree nodes is a signal for sparsity. For $\hat{G}$, the edge density is summarized in a vector $\overrightarrow{v}$ 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 $\overrightarrow{v_i}$ is the number of nodes which correspond to bin i. This metric may be normalized by computing $\frac{\overrightarrow{v_k}}{V}$ for all $k \in \{1, ..., b\}$ since there are a total of V nodes which also 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 various metrics in bins, which discards more detailed information about each individual node; the number of high 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 the graphs are the exact same!

4.3 Similarity selection

Given a set $\mathcal{S}$ of various graph pairs, we would like to select $(G^i, G^j) \in \mathcal{S}$ where (G^i, G^j) is the most similar graph pair. To be more specific, we wish to select graph $\hat{G}^i$ where $\hat{G}^i, i \in \{1, ..., k\}$ is most similar to the "base" graph $\hat{G}$. $\hat{G}^i$ is selected out of k given graphs each associated with a vector $\overrightarrow{d^i}$ of their differences (Again, $i \in \{1, ..., k\}$). Each element $\overrightarrow{d^i_m}$ for $m \in \{1, ..., 6\}$ corresponds to one of the six summarization methods listed earlier in Section 4.2. We propose the following methodology to select the most similar graph pair:

1. Compare each difference metric $\overrightarrow{d^i_m}$ against $\overrightarrow{d^{i'}_m}$ 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
$$\max_{i \in \{1,...,k\}} \sum_{m=1}^{6} \mathbf{1}_{\overrightarrow{d^{i}_{m}} > \overrightarrow{d^{i'}_{m}}} \text{ where } i \neq i' \in \{1, ..., k\}$$

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 $\overrightarrow{d_{i,j}}$, as it compares each metric to its own metric rather than computing an aggregate "difference" between graphs $\hat{G}^i$ and $\hat{G}^j$. In the next Section 4.3.1, we demonstrate the viability of this strategy with a simulation study. This method is later applied in Chapter 5 to compare the visual graph (output from the VS) and the various correlation graphs (with correlation coefficients explained in Section 1.2) The similarity selection method is coded in Appendix A.9.6.

4.3.1 Simulation study

We now perform a simulation study to demonstrate the viability of this 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

```
1: 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 $V_k$ with the given PDF. )
2:     $G^i \leftarrow G$
3:     loop from $f = 1$ to $e$:
4:         Select $E_i$ uniformly at random and delete from $G^i$
5:         Select $V_i$ uniformly at random
6:         Select $V_j$ at random where $p_k = \frac{deg(V_k)}{\sum_{x=1}^{|V|} deg(V_x)}$ $\forall\ k \in \{1, ..., |V|\}$
7:         while ($V_i = V_j$ or $(V_i, V_j)$ exists):
8:             Reselect $V_j$ at random with the same PDF $p$*
9:         Add edge $(V_i, V_j)$ to $G^i$
10:    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 should be 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 given the summarization methodologies discussed in Section 4.2, and the most similar graph pair is selected with the similarity selection method described earlier. The results are then recorded for 1000 trials, and an average is taken. The code for the simulation study may be found in Appendix A.9.7.

4.3.2 Results

The results are as follows:

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

(Continued on next page)

Table 4.1: (continued)

Distance function	**Number of swaps**	**Prob**(G^1)	**Prob**(G^2)
	$G^1 : 10, G^2 : 50$	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

It is clear, then, that the proposed similarity selection method is likely to properly select the most similar graph pairs on average. To visualize the similarity between 10 vs 10 and 10 vs 50 swaps, please review Figures 4.5 and 4.6, which plot the graphs resulting from the swaps in the final simulation trial. Since the control and the true simulations were run separately and each trial seeded with its trial number, the base graph and G^1 (which has 10 swaps in each simulation) are, expectedly, the same. It is G^2 that differs.

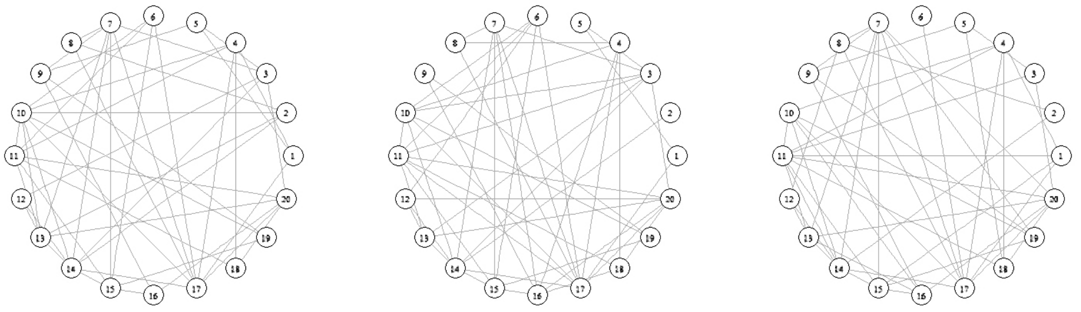

Figure 4.5: *Left:* Base graph $G(V, E)$ with $V = 20$ nodes and $E = 50$ edges. *Middle and Right:* Two random graphs built from G by randomly swapping 10 edges. Distance: L2. Seed: 1000

Extensions

Although the normalization methods proposed in Section 4.2 have bounded the graph summarization values between 0 and 1, it does not necessitate that the values are

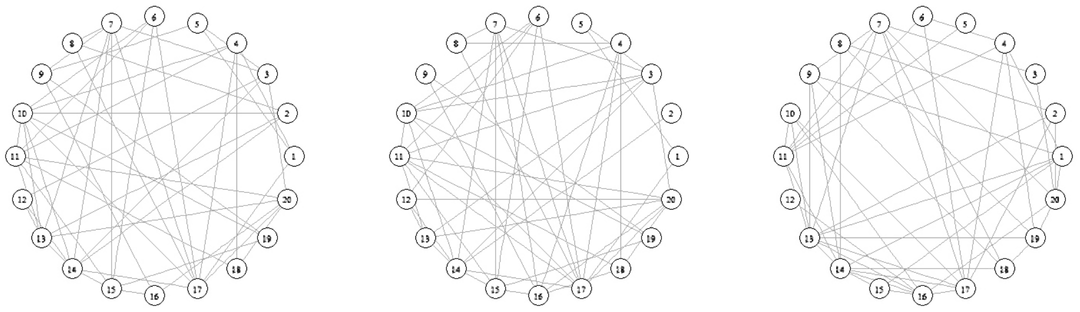

Figure 4.6: *Left:* Base graph $G(V, E)$ with $V = 20$ nodes and $E = 50$ edges.*Middle:* Random graph built from G by randomly swapping 10 edges. *Right:* Random graph built from G by randomly swapping 50 edges. Distance: L2. Seed: 1000

uniformly distributed between 0 and 1. An alternative method of normalization leverages the XXXXXXXXXX to simulate a uniform distribution. A further extension would be to implement such a procedure, which is as follows:

- Simulate n random graphs $G = (V, E)$. If implementing for the simulation above, all simulated graphs would have $|V| = 20, |E| = 50$.
- Compute the difference between each pair of graph (or the base graph???????)
- Find the p-value?????

Chapter 5

Application and results

5.1 Application of the visualization system

Let X be a $n \times d$ data matrix where there are n observations of d variables. The application of the visualization system is as follows:

1. Create four different numerical correlation graphs $G_i^{\text{num}} = (V, E)$ for all $i \in \{1, ..., 4\}$, one for each correlation coefficient described in Section 1.2.

2. Run the VS on the same dataset. Stage 1 of the system utilizes the active learning algorithm selected in Chapter 3 to learn what is "correlated" and "not correlated." Stage 2 of the system then iterates through all possible unlabeled plot pairs and returns a graph $G = (V, E)$ where

   For $i, j \in \{1, ..., d\}$, $E_{i,j}$ exists if the plot of j against i is "correlated"

   As a reminder, correlation is used loosely to refer to a visual interpretation of the mathematical term.

3. Utilize the graph comparison algorithm selected in Chapter 4 for each pair (G, G_i^{num}) for all $i \in \{1, ..., 4\}$. Select $G_i^{\text{num}} = (V, E)$ such that $GC(G, G_i^{\text{num}}) \leq GC(G, G_j^{\text{num}})$ for all $j \neq i$. The numerical correlation graph G_i^{num} is the graph which best matches the visual interpretation of the relationships among the dataset's variables.

5.2 Stock selection methodology

Given a correlation graph $G = (V, E)$, we wish to select k stocks (represented as vertices) such that each stock is as independent as possible of the other stocks. 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 then becomes one of “how to measure and detect dependence from the observation of the two random variables”[16]. With the correlation graph G, we now have a measure of dependence. However, it was made clear that, in general, it is incorrect to say that “uncorrelated” equates “independence”. Regardless, non-correlation is still a useful tool to inform stock selection for portfolio management (Section 1.3). So, we select k stocks such that each stock is as *uncorrelated* as possible with the other stocks.

As V_i and V_j have no edge when $\text{Corr}(i, j)$ is below some threshold p, we might consider the following stock selection strategy:

Algorithm 8 Naive stock selection strategy

```
1: procedure ($k$ is the number of stocks to select and $A$ is the adjacency matrix of $G = (V, E)$)
2:     $d \leftarrow \text{len}(A)$
3:     $z \leftarrow \text{Perm}(d, k)$*
4:     $min = \infty$
5:     $index = 0$
6:     loop from $i = 1$ to $\binom{d}{k}$:
7:         $c = 0$
8:         loop from $j = 1$ to $k$:
9:             loop from $l = 1$ to $k$:
10:                $c \leftarrow c + A_{z_{i,j}, z_{i,l}}$
11:        If $c < min$ then
12:            $min = c$
13:            $index = i$
14:    return $z_{index,}$
```

*Perm(d, k) is a function that returns all possible permutations of k values selected from d i.e. $z_{1,} =< 1, 5, 7 >$ is a permutation for $d = 10, k = 3$.

This becomes computationally difficult as d increases and does not account for average stock returns; 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 positive return averaged over time. We let p_d and p_v denote the threshold values for drift and volatility respectively. These values will be dependent on the size of the dataset as we wish to preselect p_d and p_v such that $k < d < 2k$ to reduce the computational burden. The selection strategy then becomes

Algorithm 9 Adjusted stock selection strategy

```
1: procedure ($k$ is the number of stocks to select and $A$ is the adjacency matrix of
   $G = (V, E)$. Let $D$ be the vector of sample average returns and $V$ be the vector
   of sample standard deviation.)
2:     function Drift
3:         $d \leftarrow \text{len}(A)$
4:         $rmv = [\,]$
5:         loop from $i = 1$ to $d$:
6:             If $D_i \leq p_d$ then $rmv$.append($i$)
7:         return rmv
8:     loop from $i = 1$ to len($rmv$):
9:          Delete row $rmv_i$ and column $rmv_i$ from $A$
10:    function Volatility
11:        $d \leftarrow \text{len}(A)$
12:        $rmv = [\,]$
13:        loop from $i = 1$ to $d$:
14:            If $V_i \leq p_v$ then $rmv$.append($i$)
15:        return rmv
16:    loop from $i = 1$ to len($rmv$):
17:         Delete row $rmv_i$ and column $rmv_i$ from $A$
18:    function Selection
19:        $d \leftarrow \text{len}(A)$
20:        $z \leftarrow \text{Perm}(d, k)$*
21:        $min = \infty$
22:        $index = 0$
23:        loop from $i = 1$ to $\binom{d}{k}$:
24:            $c = 0$
25:            loop from $j = 1$ to $k$:
26:                loop from $l = 1$ to $k$:
27:                    $c \leftarrow c + A_{z_{i,j}, z_{i,l}}$
28:            If $c < min$ then
29:                $min = c$
30:                $index = i$
31:        return $z_{index,}$
```

*Perm(d, k) is a function that returns all possible permutations of k values selected from d i.e. $z_{1,} =< 1, 5, 7 >$ is a permutation for $d = 10, k = 3$.

5.3 Healthcare stock data

We select 43 stocks from the healthcare industry to apply the system to. There are 4113 observations of the stocks from 1998 to 2014. We have cleaned the data to get rid of the dependency on time so that each observation of a stock may better simulate an independent draw from some distribution. This is done by fitting a regression to the values and using the residuals as the new dataset.

For $i \in \{1, ..., 4\}$, we employ a "buy and hold" tactic to compare the performance of each portfolio P_i selected utilizing the methodology described in Section 5.2), which correspond to correlation graphs $G_i^{\text{num}} = (V, E)$. While a rebalancing method performs better in practice [11], the "buy and hold" is better-suited for observing the portfolio performances over time, providing a more concrete basis of comparison for the selected correlation coefficient.

5.4 Results

Here, we have a comparative time plots of the "buy and hold" performance (yearly returns) of the S&P 500, stocks selected from the final numerical correlation graph $G_i^{\text{num}} = (V, E)$, and other correlation graphs $G_j^{\text{num}} = (V, E)$ where $j \neq i$. We hope to find that correlation graph i, which was selected from the usage of the VS, is a top performer.

We may also analyze the VS graph output $G = (V, E)$ against the final selection G_i^{num} by asking the following questions: What new links were formed? Which links were deleted? Did we actually find the plots of those links interesting/not interesting? If it is possible (if the graphs are not too dense), then I would also include a visualization of G and G_i^{num}.

Chapter 6

Conclusion

The proliferation of high-dimensional datasets has led analysts to rely on unverifiable numerical estimators. We are interested in visualizing high-dimensional correlation graphs as a way to verify numerical tests of dependence. This is important in asset management as it is desirable to select a portfolio of stocks that are as independent as possible. High-dimensional visualization is problematic because (1) there are too many potential plots to sort through manually (To be specific, there are $\binom{d}{2}$ plots where d is the number of variables e.g. stocks that we are interested in), which also means that (2) it is tedious to verify numerical results with visual results and vice versa (Section 1.1). In this work, we presented a visualization tool that actively learns user preferences, applies the fitted classifier to unlabeled data, provides visualization tools for the active learning output (a visual graph $G = (V, E)$), and outputs the difference among the numerical graph $G^{\text{num}} = (V, E)$ and G (Chapter 2).

As a specific response to the two aforementioned problems, we focused on the active learning and graph comparison components of the visualizer systems. We discussed two main approaches to active learning and provided algorithms for different active learning methods (Chapter 3). A simulation study indicated that $- - --$ would be the best learning algorithm for the financial application of the VS system.

Similarly, we discussed various measures of graph distance and performed a simulation study to determine which methodology to implement in the VS system for the financial application (Chapter 4).

We then ran the VS system with $----$ in stage 1 and used $----$ to compute the graph comparison of the output G with G_i^{num} for all $i \in \{1, ..., 4\}$, the correlation graphs for Pearson, Spearman's, Kendall's, and distance correlation coefficient respectively. $----$ most closely matched the visual graph. We then utilized the stock selection methodology described in Section 5.2 to select a portfolio of $k = 10$ stocks P_i for all $i \in \{1, ..., 4\}$ based on G_i^{num} based on data from year 1. Yearly returns were computed with the remainder of the observations in a simulation of the "buy and hold" strategy. As expected, the portfolio corresponding the correlation graph selected with the aid of the VS was one of the top performers (??).

The visualization tool presented in this work is an important step in streamlining the future of clean analysis. It provides a systematic way for confirming and/or suggesting dependencies among variables that match our visual concept of a dependence and produces an explicit decision tree that allow others to understand and replicate the data analysis process. This alleviates the problems associated with high-dimensional datasets and allows the user to quickly see ways in which 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."

6.1 Further extensions

This section details further extensions on improving other aspects of the visualization system. To improve the methods discussed in this paper, see the final "extensions" section in the Chapter 3 for active learning and Chapter 4 for graph comparison.

6.1.1 Estimator selection

Estimator selection involves actively fitting the best model as opposed to "checking" a numerical model thats been given. For instance, rather than requiring a numerical model in order to check for graph distance (Chapter 4), the VS would select a numerical model without requiring the analyst to fit one beforehand. This problem is more difficult to define, and the value that the visualization system adds is not as concrete.

6.1.2 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.

((When can the system remove outliers? What criteria should it use?))

6.1.3 Graphical models and improved stock selection

A single correlation (discussed in Section 1.2) 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. 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. Although it may seem simple to numerically quantify the dependencies with correlation as conditional dependencies are less intuitive to compute, 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, situations where the correlation is close to 1 or 0, then the transitivity of correlation can be recovered and observed in the relevant data [19]. 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. The correlation between observed prices of Google and Apple stock will clearly and erroneously be positive without considering the way the stocks are connected to the other observed variable (Silicone stock). Given that correlation tends to be transitive, a correlation graph can have too many edges. This goes against the concept of "sparsity" (Section 1.3) by cluttering the resulting space of observation variables that explain the response variable for the user, leaving the analyst with an uninformative and unaccountable numerical solution.

Graphical models alleviate some of the problems associated with correlation graphs, but they have their own set of problems, as well. Let $G = (V, E)$ be an undirected graph with vertices $V_1, ..., V_d$ (a d-dimensional distribution) and edges $E_{i,j} \in \{0, 1\}$. We set $E_{i,j} = 1$ when there is an edge between V_i and V_j, and 0 otherwise. Do not draw an edge between V_i and V_j iff $V_i \perp V_j$ given V_k where $k \in \{1, ..., d\} \setminus \{i, j\}$. In other words, do not draw an edge if

$$\mathrm{P}(V_i, V_j | V_k) = \mathrm{P}(V_i | V_k)\mathrm{P}(V_j | V_k)$$

This result 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. While there are simplifications that can be made for plotting, the solution is not always so clear. However, the conditional independence of graphical models is more of a global property than correlation is because, for every pair of variables, it conditions on all the remaining variables. Returning to the universe of Google, Apple, and Silicone stocks, conditioning Google on Silicone and Apple on Silicone makes the relationship between the two clearly uncorrelated. Thus, although conditional independence tends to be more difficult to determine, it will tend to give a sparser network that is more interpretable for the analyst.

Correlation graphs and graphical models each have their own niche to fill, though a graphical model rebalancing has been shown to strongly outperform traditional portfolio management methods [11]. It is important, therefore, for the VS to support both models. The system would need to be modified to allow for plotting V_i against V_j conditional on V_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 this scatter plot, similar to regression.

6.1.4 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 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 (showing graphs that are most alike one after the other), users are less able to distinguish between differences than if they are shown graphs from different ends of the spectrum at different times [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 display matters, as well, 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 their understanding of the plots, careful display ordering advances the accuracy of user responses. User responses can be thought of as observations of the users true preferences, and the ordering of plots as a way to fine-tune the precision of the classifier.

6.1.5 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 for this is due to the vagueness of defining "independence" on a non-uniform domain and range (Section 2.1.1). Wickham *et al.* propose a line-up protocol that is similar to the Rorscach test where subjects are asked to interpret abstract blots of ink [21]. In the line-up test, users are asked to identify the real data from a set of n plots where $n1$ plots are synthetically generated (Figure6.1). Identification of the raw data against the synthetic data is then an indicator that the current fitted model is a poor fit of the user's preferences. In the context of the classification problem, the learner may generatae four "uninteresting" plots (following the proposed decision tree) with one "interesting" plot and asks the user if he/she can identify the interesting plot. If the user is able to consistently identify the interesting plot, it is an indication that the current decision tree is a close fit of the users preferences.

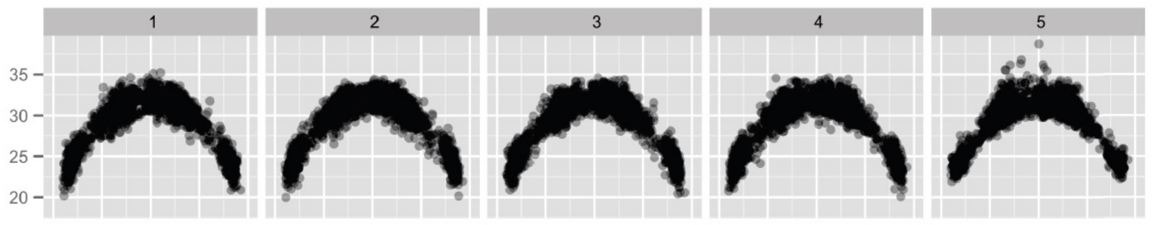

Figure 6.1: A line-up test for $n = 5$. Consistently identifying the raw data against the synthetic data indicates that the fitted model may not be good enough. Figure from Wickham *et al.* [21]

6.1.6 Edge-weighted graphs

Edge-weighted graphs also stray from the black and white classification of correlation graphs (Section 1.2) and graphical models (Section 6.1.3) which have been discussed earlier. In an edge-weighted graph, edges can be weighted depending on the type of conditional dependence or correlation. Negative or positive values are assigned 1 or -1, respectively. A value of 0 (no edge) still implies conditional independence or uncorrelated variables. This problem is more difficult as the number of classification labels changes from two to three, and the graph has become more information but complex. Active learning methods may need to be more generalized as in the case of uncertainty sampling (Section 3.2.1).

6.1.7 Rejection classification

So far, classification has been discussed in terms of black and white, interesting versus non-interesting. While interacting with the data visually, the user's concept of what's interesting in the specific dataset may evolve over time; at the beginning, they have 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 dont 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 oracle provide labels in order to better understand user preferences. By allowing for this option, the active learner may run for too long and/or return a poorly-defined tree.
- The system can contain a third option that permits the user to "recycle" the plot. This allows to user to return to the plot later after learning more about what the data looks like and understanding his/her own preferences better, and it ensures that the active learner will eventually receive data on the ambiguous plots that it has queried. The main concern is that this could potentially debalance an ordering procedure (Section 6.1.4), but the system can strategically insert the recycled plot between two plots it differs 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 implementation

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 implementation

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 implementation

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 implementation

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 implementation

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 Simulation implementation

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="Various␣Active␣Learning␣Error␣Ratios␣with␣Random␣
           Forest␣Classifier",
        xlab="Iterations", ylab="Error", col = "green")
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␣
           Classifiers",
        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))

### Plot 95% confidence intervals

par(mfrow=c(3,2))

# Random Sampling
graphics::plot()
graphics::lines()

# Uncertainty Sampling
graphics::plot()
graphics::lines()

# Min-Max Clustering
graphics::plot()
graphics::lines()

# Query by Bagging
graphics::plot()
graphics::lines()

# Query by Committee (best)
graphics::plot()
graphics::lines()
```

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

A.9.5 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 $\overrightarrow{d^{1,2}}$ 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) foro 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(
                    0,igraph::gorder(g1)-1,distf) * length(a))
            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.9.6 Similarity selection

Given a set $\mathcal{S}$ of various graph pairs, the similarity selection method selects the most similar pair $(G^i, G^j) \in \mathcal{S}$. 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.9.7 Simulation study

The 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)
```

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] 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.

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

[12] 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.

[13] 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.

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

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

[16] 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.

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

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

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

[20] 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.

[21] 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.

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