APPLICATIONS OF MACHINE LEARNING FOR FINANCIAL ACCOUNTING

BY

KELECHI M. IKEGWU

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Informatics
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2021

Urbana, Illinois

Doctoral Committee:

Professor Robert J. Brunner, Chair, Director of Research
Professor Theodore Sougiannis
Assistant Professor Jeff McMullin
Assistant Professor Joseph Yun

Abstract

foo.

Acknowledgments

First and foremost, I would like to thank my advisor, Robert Brunner, for his guidance, patience, and support during my graduate studies at University of Illinois at Urbana-Champaign. I also wish to thank my committee members, Theodore Sougiannis, Jeff McMullin, and Joseph Yun, for their help and support. I am also grateful to Matias Carrasco Kind, William Biscarri, Samantha Thrush,and ....

— Kelechi Moise Ikegwu

This work was supported in part by foo

This work was partially funded by the Graduate College Fellowship program at the University of Illinois. This work utilizes resources provided by the Innovative Systems Laboratory at the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign.

Table of Contents

Chapter 1

Introduction

The economic goal of a society is to maximize wealth by allocating resources efficiently. Capital markets (such as the stock market) provide a method to help allocate resources efficiently. The prediction and inference of future financial events are often critical to achieve the economic goal. Firm specific information contribute to how resources (i.e labor, capital, and natural resources) are allocated in capital markets. The prediction of a future state for a firm (such as the firm's profitability, financial condition, or revenue growth) is an important factor to contribute to how resources should be allocated. There are many methods used to predict or inform about the future state of a firm.

1.1 Machine Learning

Machine Learning encompasses a vast set of statistical tools for understanding data and making predictions and inference. These tools fall into supervised or unsupervised categories. For data-driven problems typically we have a response variable (or labels) referred to as Y and p independent variables (or features) refereed to as X. If an assumption is made that there is some relationship between Y and X the relationship can be expressed as:

$$Y = f(X) + \epsilon \tag{1.1}$$

In eq. 1.1 f is an unknown function that models the relationship between X and Y with ϵ noise. In most cases it is not possible to perfectly model f with real-world data and f is estimated ($\hat{f}$). In eq.1.1 if Y is a quantitative variable then regression techniques are used

estimate f, however if Y is a qualitative variable then classification techniques are used. $\hat{f}$ is sought for inference where one wants to estimate f to understand how $\hat{Y}$ (predicted labels) changes as one adjusts features. $\hat{f}$ is also sought for prediction where the only concern is the accuracy of predictions for Y. Machine learning provides a variety of approaches to estimate f for prediction and inference (see **?**).

Machine learning problems typically fall into either supervised learning, semi-supervised learning, or unsupervised learning. With supervised learning one wants to fit a model on labels using a set of features. Semi-supervised learning consists of an environment with features and some corresponding labels. Typically in this setting one wishes to use the given features and partial labels to determine what the remaining missing labels are. Lastly with unsupervised learning one has features and no associated labels and seeks to find structure between the features. Currently, the research in this book will only utilize supervised learning.

1.1.1 Supervised Learning

In a supervised learning paradigm it is not enough to only have $\hat{f}$ estimate f well on a dataset. $\hat{f}$ is often sought to learn a dataset well enough to generalize well to new data that $\hat{f}$ has never seen before. To achieve this goal, a common methodology is to partition the data into a training set and testing set. The training set is used to train the model. The testing data is only used to assess the $\hat{f}$'s ability to generalize well to new data. The ratio of data used to train and assess the model is typically determined by the practitioner. A specific [1] way to estimate $\hat{f}$'s performance is use an error metric such as mean squared error (MSE) given by

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2 \tag{1.2}$$

[1] There are many error metrics that can be used however, for the sake of brevity MSE is the only error metric that will be discussed.

In eq. 1.2 y_i and x_i are the i^{th} observations of the labels and features respectively. $\hat{f}(x_i)$ is the prediction of $\hat{f}$ for the i^{th} observation and y_i is the label for the i^{th} observations. There will be a training MSE indicating the mean error of $\hat{f}$ learning the training data. Subsequently there is a testing MSE associated with mean error of $\hat{f}$ on the withheld testing data. A model with a very small testing MSE generalizes well to the withheld data. Often, the training MSE and testing MSE differ where the training MSE underestimates the MSE for the test MSE **?**. To combat this issue there are many approaches that can produce a more accurate MSE of how $\hat{f}$ will generalize to new data.

1.1.2 Validation Approaches

An alternative approach to training and assessing $\hat{f}$ on training and testing sets respectively is to create a validation dataset by randomly sampling the training dataset. The sampling method is determined by the researcher/practitioner. $\hat{f}$ is then trained on the training data and the trained model is assessed with validation data. The validation MSE is more indicative of the test MSE. However there are issues with this approach. For example the validation data-set is comprised of randomly sampled data. Given the random data the validation MSE highly variable. To combat this issue cross-validation can be used to decrease the variability of the validation MSE.

A common cross-validation technique is k-fold cross-validation where the training data is split into k separate folds (see Figure 1.6). The first fold is withheld and used as the validation data-set to assess the model's performance. The remaining $k-1$ folds are used to train the model. For the next iteration the same procedure is followed where the 2nd fold is withheld to serve as the validation data-set to assess the model (see Figure 1.7) and the other $k-1$ folds are used. This process repeats k times and produces k MSE scores $(MSE_1, MSE_2, ..., MSE_k)$. The k-fold CV estimate is computed by averaging the MSE fold values

$$CV = \frac{1}{k}\sum_{i=1}^{k} MSE_i \tag{1.3}$$

where k is the number of folds.

Selecting a large value k can be computationally expensive as the model must be retrained on k separate folds. In addition to this, the selection of k can effect the model's ability to generalize to new data well. Typically with a smaller value of k you have more bias and a larger value of k has more variance. For example consider when $k = 3$ and $k = 10$, the training folds will have n_{train} observations where $n_{train} = \frac{(k-1)n}{k}$ and the validation fold will have $n_{validation}$ observations where $n_{validation} = \frac{k-(k-1)n}{k}$. When $k = 10$, 90% of the data is used to train the model and the remaining 10% is used to assess the model. The model is trained on very similar datasets and the predictions from the model on the cross-validation datasets are highly correlated. This results in a model with high variance. If $k = 3$ 66.7% of the data is used to train the model and the remaining 33.3% of the data is used to assess the model. CV averages fewer predictions that are less correlated with each other, since the overlap between the training sets are smaller. This results in a CV score with lower variance and higher bias.

1.2 Network Science

A network (or graph) consists of a collection of nodes joined by connections between the nodes (commonly referred to as edges). Networks can be formed from data to offer a representation of the structure of data which subsequently allows many analysis techniques to be performed. Network Theory involves the study of networks as a representation of relationships between nodes. Throughout this section a network will be denoted by G, the number of edges in a network will be denoted by m, and the number of nodes will be denoted by n. A graph with n nodes and m edges is denoted as $G(n, m)$.

Graphs do not have to specify the direction of an edge. Graphs with this property are

referred to as a undirected graph. For example if Node A and Node B are connected a undirected graph will not distinguish whether Node “A" is connected to Node “B" or vice-versa; an undirected graph will only retain information about nodes A and B having a connection. Figure 1.1 shows an example of a undirected graph $G(7, 16)$. The amount of edges a particular node has is referred to as the degree of the node. In Figure 1.1, the node “FB" has a degree of 6 because it is connected to 6 other nodes. Degree is a common metric used in network theory.

It is also possible for a graph to retain additional information about edges. Another source of information are the direction of the edges; graphs that store information about the directions of edges are called directed graphs. For example if Node “A" is connected to Node “B" a directed graph would contain information that Node “A" has an edge to Node “B" but Node “B" does not have an edge to Node “A". Figure 1.2 shows an example of a directed graph; in Figure 1.2 there is an edge from “‘JNJ" to “FB" however there is no edge from “FB" to “JNJ".

Another source of information that graphs can retain are edge weights and are called weighted networks (or weighted graphs). Instead of having a binary connection to determine whether an edge is present or not it can be useful to measure the strength between connections of nodes.For example if you can have an attribute to determine the strength between nodes such as TE estimates, TE estimates can be used as the weight value of an edge to determine the strength of information flow between nodes. Edge weights are applicable to both directed and undirected graphs ergo it is possible to have a undirected weighted graph or directed weighted graph. Figure 1.3 contains an example of a directed weighted network.

Under certain circumstances it is more convenient to express a graph as an adjacency matrix. For example consider the graph in Figure 1.1, if this were expressed as a table with column and row headers Table 1.1 would be produced.

However it is mathematically more convenient to express Table 1.1 as an adjacency matrix (see Eq 1.4).

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{1.4}$$

Similar to Table 1.1 each matrix element (A_{ij}) in A is assigned a 1 or 0 based on if there is a connection between a row index (i) and column index j. You'll notice in Eq 1.4 that the matrix is symmetric because the direction of edges are not taken into account; ergo if node i and node j are connected A_{ij} and A_{ji} have a value of 1. It is also possible to represent directed graphs by assigning a 1 to a matrix element A_{ij} where node i is connected to node j. Lastly you can express the weight of edges in adjacency matrices by assigning non-binary values to matrix elements.

1.2.1 Analysis of Networks

After forming a network from data, common metrics are often sought to help describe the structure of the network. The first metric are the degree of nodes where the degree can be defined in Eq 1.5 as the sum of edges for a particular node. In Figure 1.4 the node sizes are scaled based on the degree .

$$k_i = \sum_{j=1}^{n} A_{ij} \tag{1.5}$$

where A is an adjacency matrix. Subsequently you can have degrees for directed networks such as in-degree which counts the amount of edges that are connected to node i (see Eq

1.6). Whereas out-degree counts the amount of edges that node i is connected to (see Eq 1.7).

$$k_i^{in} = \sum_{j=1}^{n} A_{ij} \tag{1.6}$$

$$k_i^{out} = \sum_{i=1}^{n} A_{ij} \tag{1.7}$$

The density of the network provides a numerical value for the portion of potential connections in a network that are actually connected. If all potential connections in a network are present then the density will have a numerical value of 1 and if there are no connections in an undirected network the density will be 0. The maximum number of potential connections in a network is $n(n-1)/2$. The degree summed for all nodes is m (for directed networks the summed in-degree and out-degree for all nodes is m). The density for a graph can be defined as:

$$D_u = \frac{2m}{n(n-1)} \tag{1.8}$$

Centrality is another metric often sought to measure the importance of nodes in networks. Measuring the degree of nodes in relation to other nodes is called degree centrality (see Eq 1.9).

$$C_i^d = k_i = \sum_{i=1}^{n} A_{ij} \tag{1.9}$$

1.3 Information Transfer in Financial Markets

The price discovery process is important to achieve the economic goal of a society. Information is embedded in price and in an efficient market changes rapidly (if not instantaneously) when new information arrives. In particular how information transfers from one price to

another is useful in the price discovery process. For example Apple Inc. announced that their new line of computers (Macs) would use central processing units (CPUs) developed in house and transition away from Intel) CPUs. In ideal settings this information will be reflected in Apple's stock price and Intel's stock price.

In the example above it is probable that additional information can be reflected in Intel's price change at a given moment. However, it is difficult to determine price changes for specific events. The entire set of information that is reflected in the price change at a given time period is unknown. Prior studies have examined the price changes over long windows such as days or weeks (see **?** and **?**) or shorter windows such as days. These methodologies may have missed much of the price change process given the assumption that information effects the price within seconds. Given the theory that markets are efficient and reflect information into price within seconds after the information is known, we seek suitable methods to measure information flow in financial markets.

1.3.1 Information Theory

Information theory provides a framework for studying the quantification, storage, and communication of information **?**. Information theory has the potential to be useful in the price discovery process given that it can offer alternative approach for understanding information in price. In particular it offers alternative methods to understand information transfer between dynamic processes (such as stock prices). In this dissertation, I measure information flow in financial markets with the relatively new measure transfer entropy (see **??** and **??**). Transfer Entropy (TE) quantifies the reduction in uncertainty in one random process from knowing past realizations of another random process (see **??**). For example it is unlikely to predict Intel's closing price tomorrow. However knowing Intel's price today reduces the amount of uncertainty in the prediction for it's price tomorrow. In addition to this, knowing Apple's price today may help further reduce the uncertainty in the future prediction. If Apple's price today helps to further reduce the uncertainty then TE will be positive indicating

an information transfer from Apple to Intel.

In 2000, Schreiber **?** discovered TE and coined the name "transfer entropy," although Milian Palus **?** also independently discovered the concept as well. To define TE, one must begin with the building blocks of information theory [2]. Shannon Information is defined as:

$$h(x) = \log_2 \frac{1}{p(x)} \tag{1.10}$$

where x is an event and $p(x)$ is probability of x occurring. This definition implies that values of x with small probabilities contain more information. Consequently values of x that are more probable (or common) contain less information. Entropy is the amount of uncertainty or disorder in a random process.

$$H(X) = \frac{1}{n} \sum_{i=1}^{n} \log_2 p(x) \tag{1.11}$$

Entropy is defined in eq. 1.11 is the weighted average of shannon information. Conditional entropy is the entropy after conditioning on another process (Y):

$$H(X|Y) = \sum_{y \in \Omega_y} p(y) H(X|y) \tag{1.12}$$

In eq. 1.12 $y \in \Omega_y$ represents the occurrence of y in a set of possible events for y and $H(X|y)$ represents conditional entropy on a single event y (or $H(X|y) = \sum_{x \in \Omega_x} \frac{p(x|y)}{\log_2 p(x|y)}$). Mutual Information quantifies the amount of information shared across random variables:

$$I(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) \tag{1.13}$$

In eq. 1.13 consider $H(X) - H(X|Y)$, $H(X)$ computes the entropy of X and $H(X|Y)$ computes the entropy of X conditioned on Y. The difference between $H(X)$ and $H(X|Y)$ captures the shared information between X and Y. Mutual Information in a directed measure ergo

[2] For simplicity discrete events are used to introduce these concepts.

$I(X, Y) = I(Y, X)$.

Lagged mutual information $I(X_t : Y_{t-k})$ can be used as a time-asymmetric measure of information transfer from Y to X where X and Y are both random processes, k is a lag period, and t is the current time period. However, lagged mutual information is unsatisfactory as it does not account for a shared history between the processes X and Y **?**. While similar to lagged mutual information, transfer entropy (TE) also considers the dynamics of information and how these dynamics evolves in time **?**. TE is also an asymmetric measure of information transfer. Ergo, TE computed from process A to process B may yield a different result than TE computed from B to A. The information theoretic framework and these measures have led to a variety of applications in different research areas **?**, **?**.

TE considers the shared history between two processes via conditional mutual information. Specifically, TE conditions on the past of X_t to remove any redundant or shared information between X_t and its past. This also removes any information in the process Y about X at time t that is in the past of X **?**. Transfer entropy T (where the transfer of information occurs from Y to X) can be defined as:

$$T_{Y \to X}(t) \equiv I(X_t : Y_{t-k} | X_{t-k}) \tag{1.14}$$

Kraskov Kraskov et al. (2004) shows that transfer entropy can be expressed as the difference between two conditional mutual information computations:

$$T_{Y \to X}(t) = I(X_t | X_{t-k}, Y_{t-k}) - I(X_t | X_{t-k}) \tag{1.15}$$

The intuition of this definition is that TE measures the amount of information in Y_{t-k} about X_t after considering the information in X_{t-k} about X_t. Put differently, TE quantifies the reduction in uncertainty about X_t from knowing Y_{t-k} after considering the reduction in uncertainty about X_t from knowing X_{t-k}.

1.4 Figures and Tables

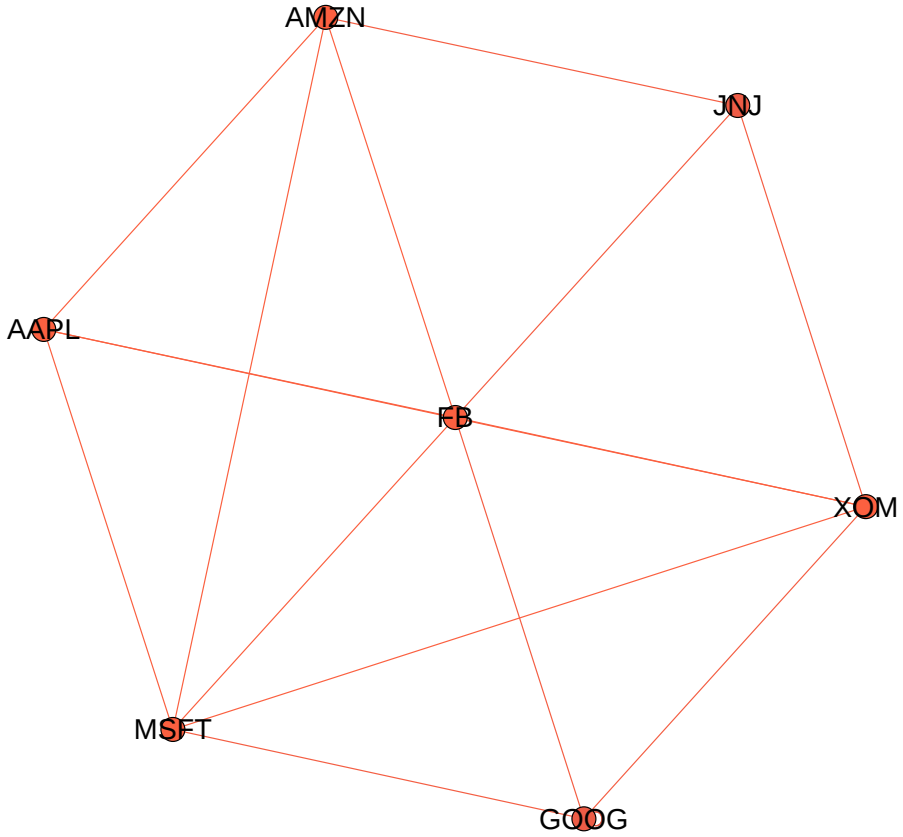

Figure 1.1: This figure contains an example of an undirected graph with 7 nodes and 16 edges. This graph is formed from randomly selecting stock symbol tickers from a set of tickers and forming connections randomly. The purpose of this data is solely to demonstrate the basics of network theory. The nodes are defined as circles and have ticker symbols overlaid on them. Edges are formed between nodes with lines between them. Since this is an undirected graph there is no directional information ergo no distinction as to whether one node is connected to another or vice-versa. The only information represented from an undirected graph is that they are connected. Lastly the amount of connections a node has is referred to as degree. So the node FB has 6 edges or a degree of 6.

	MSFT	AAPL	AMZN	FB	JNJ	GOOG	XOM
MSFT	0	1	1	1	0	1	1
AAPL	1	0	1	1	0	1	1
AMZN	1	1	0	1	1	0	0
FB	1	1	1	0	1	1	1
JNJ	0	0	1	1	0	0	0
GOOG	1	1	0	1	0	0	1
XOM	1	1	0	1	0	1	0

Table 1.1: This table contains an example of simulated data. This table was formed from randomly selecting stock symbol tickers from a set of tickers. The purpose of this data is solely to demonstrate the basics of network theory. Here the columns and row indicies belong to the selected tickers. The values contain either a 1 to represent if there is a connection between the ticker at a particular row index i and the ticker of a particular column j or a 0 if there is no connection.

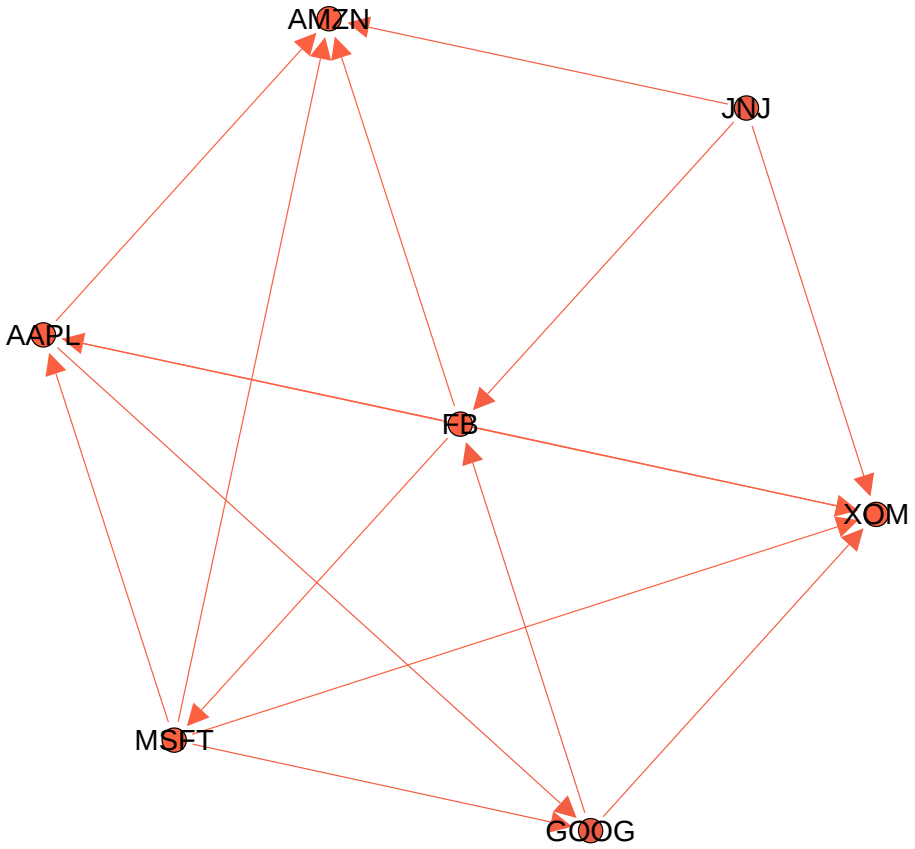

Figure 1.2: This graph is nearly identical to the graph in Figure 1.1 except this graph is a directed network. This means that it contains directional information between the nodes. This directional information is communicated based on the position of the arrow in a line. For example, the node FB has an edge to $MSFT$ however $MSFT$ does not have an edge to FB. The directional information allows other metrics to be used such as in-degree (which represents the amount of edges directed toward a node) and out-degree (which represents the amount of edges directed away from a node. In this example FB has an in-degree of 2, an out-degree of 4 and a degree of 6.

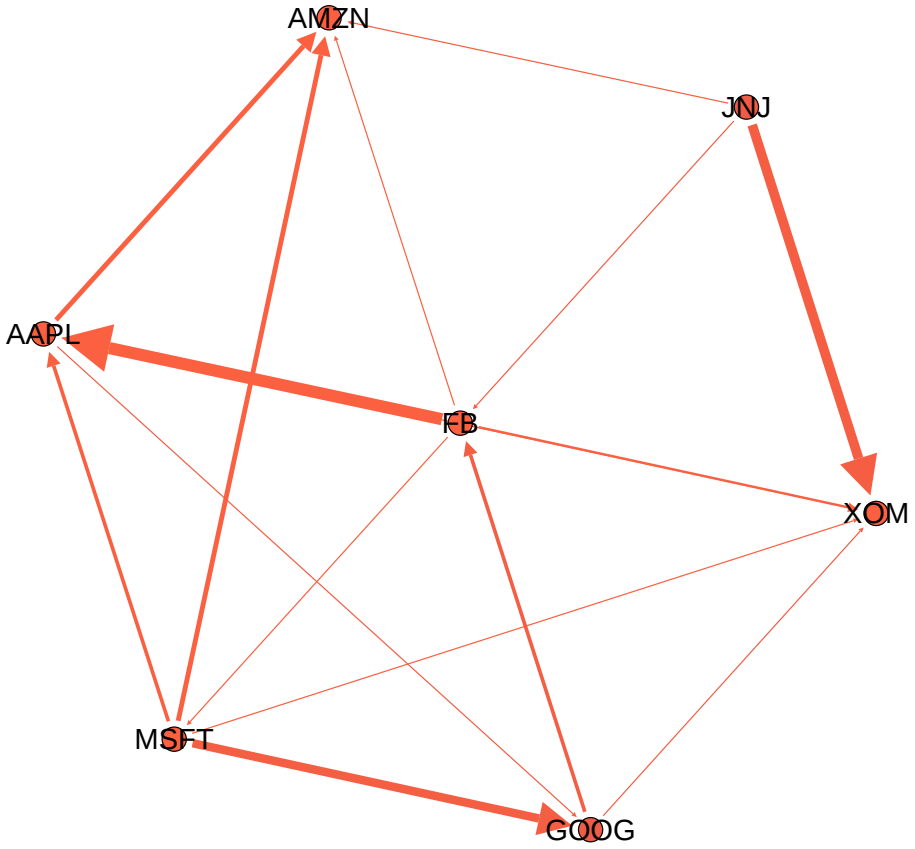

Figure 1.3: A nearly identical graph to the graph in Figure 1.2. This directed graph contains weighted edges where a thick edge represents a high weight value (or strong connection). The smaller the edge weight value the thinner the edge will be. Here FB has a strong connection to $AAPL$ whereas FB has a weak connection to $AMZN$.

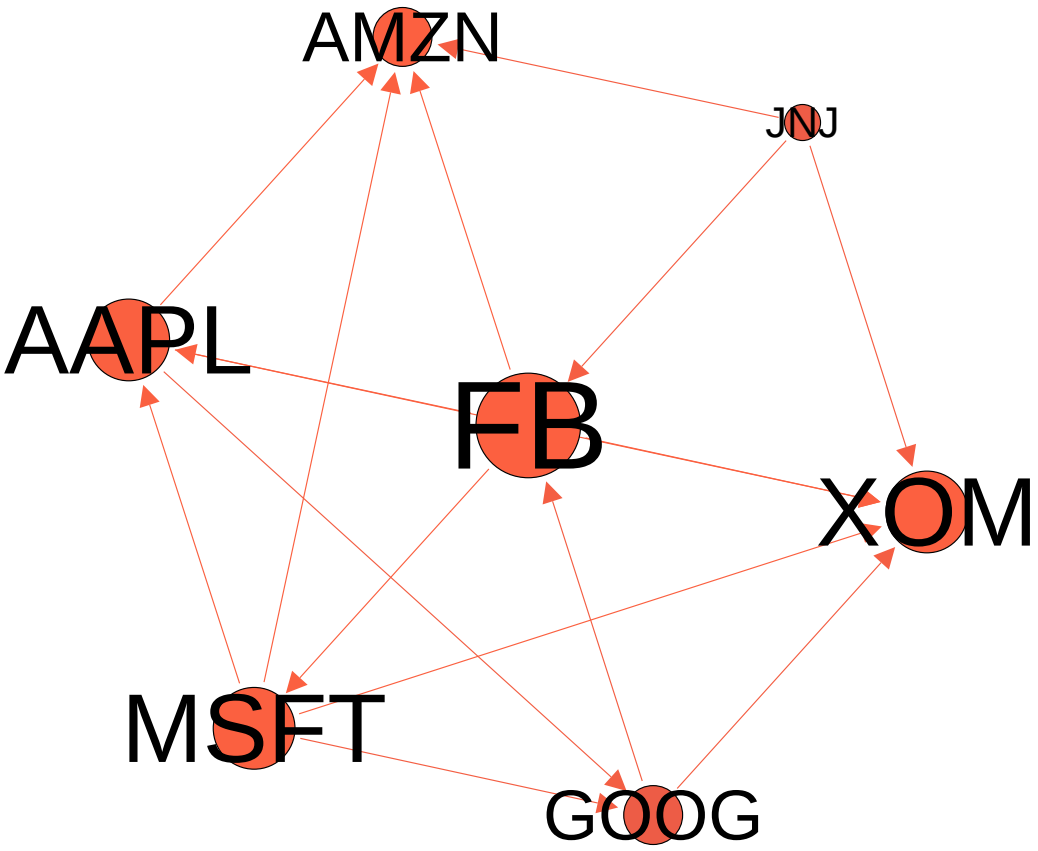

Figure 1.4: A nearly identical graph to the graph in Figure 1.2. Here the node sizes are scaled based on the degree. The higher the degree the larger the node size and consequently the smaller the degree the smaller the node size.

Figure 1.5: This figure shows how a data set can be partitioned for a supervised machine learning paradigm. The “Data Set" is partitioned into 2 unequal sets with the“Train Set" having more data assigned to it than the “Test Set". The “Train Set" and “Test Set" ratios can vary and is typically determined by the practitioner. There are many ways to partition the data for the train and test sets. For example the dataset can be partitioned sequentially from the data, or form partitions based on the data being randomly sampled.

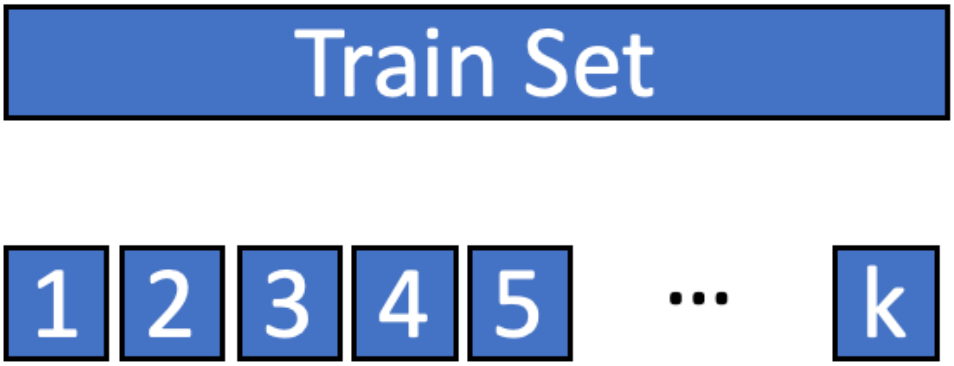

Figure 1.6: “Train Set" here represents “Train Set" in Figure 1.5. In this figure the data is split into k folds. How the k folds are created can vary. A common technique is to randomly divide the “Train Set" observations into k equal folds.

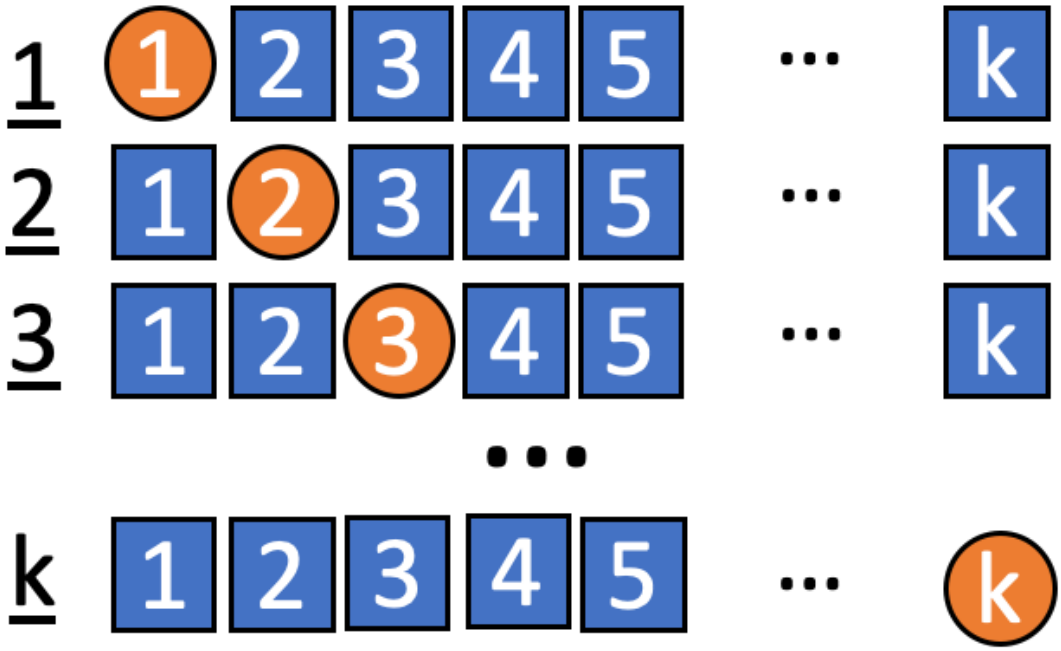

Figure 1.7: This figure breaks down how k-Fold Cross Validation works. These folds are created from "Train Set" in Figure 1.6. For the first iteration (where you see 1 underlined), the 1st fold (circle labeled 1) is used to assess the models ability and the remaining folds (squares) are used to train the model. The 2nd iteration follows the same procedure as the 1st except only the 2nd fold is withheld and the other folds are used. This repeats k times.

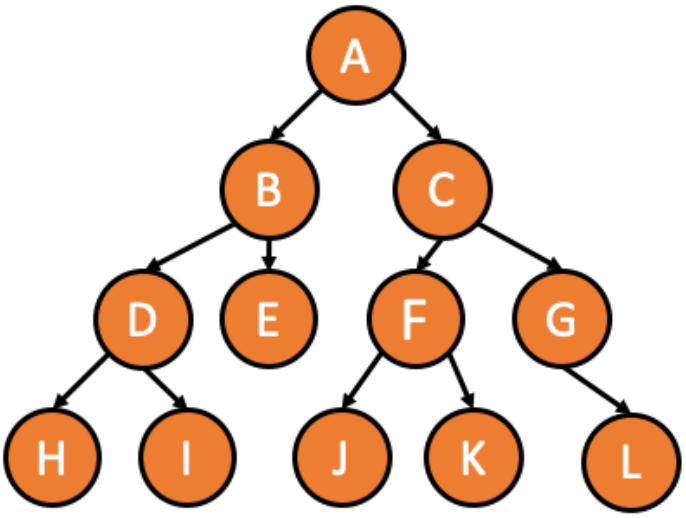

Figure 1.8: This figure shows an example of a Decision Tree. This is essentially a graph with no cycles. Each circle represents a node (the label of the node is a letter inside of the node) and the arrows represent directional edges (branches) to another node. Node A is the root node of this tree and is where the decision begins. Potential outcomes from the root node lead to different states that are represented in nodes B and C. B and C are the children of A. Nodes B and C have children and their children has children. Nodes H, I, E, J, K, and L are the leaf nodes of this tree because they have no children.

1.5 References

Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Phys. Rev. E*, 69:066138, Jun 2004. doi: 10.1103/PhysRevE.69.066138.

Chapter 2

Standard Machine Learning Language

2.1 Introduction

Machine Learning has simplified the process of solving problems in a variety of fields (see Bakshi and Bakshi (2018) or Monahan (2018) for examples). However Domingos (2012) noted that there are a number of nuisances that should be taken into consideration when developing machine learning pipelines. If these nuisances are not taken into consideration one may not receive satisfactory results. To combat these issues we introduce Standard Machine Learning Language (SML) targeted at domain experts that want to utilize machine learning to solve research questions.

The overall objective of SML is to provide a level of abstraction which simplifies the development process of machine learning pipelines. Consequently this enables students, researchers, and industry professionals without a background in developing machine learning pipelines to solve problems in different domains with machine learning (see Listing 2.1 for an example). In the subsequent sections related works are discussed, followed by defining the grammar used to create queries for SML. The architecture of SML is described, lastly SML is applied to use-cases to demonstrate how it reduces the complexity of solving problems that utilize machine learning.

This chapter contains material from the following working paper:

- Kelechi Ikegwu, Micheal Hao, Nerraj Asthana, and Robert Brunner. Standard machine learning language: A language agnostic framework for streamlining the development of machine learning pipelines. 2017. URL `https://github.com/lcdm-uiuc/Publications/blob/master/2017_Kelechi_Mike_Brunner/main.pdf`

2.2 Prior Works

There are prior related works that attempt to provide a level of abstraction for developing machine learning models. Rizzolo and Roth (2010) created a tool called LBJava based on a programming paradigm called Learning Based Programming (see Roth (2005)). Learning Based Programming is an extension of conventional programming that creates functions using data driven approaches. LBJava utilizes machine learning to create these functions and abstract the details from the user. What differentiates SML from LBJava is that it offers a higher level of abstraction by providing a query like language which allows people who aren't experienced programmers to use SML.

TPOT (see Olson et al. (2016)) is a tool implemented in Python that creates and optimizes machine learning pipelines using genetic programming. Given cleaned data, TPOT preprocessing the data, performs feature selection, and the construction of machine learning models. Given the task (classification, regression, or clustering) TPOT uses genetic programming to tune model parameters and select features to determine the most optimal model to use. Similar to TPOT Kotthoff et al. developed Auto-WEKA which automates the selection of learning algorithms and tuning hyper-parameters for implemented models in WEKA see (Frank et al. (2005)).

Subsequently Komer et al. created Hyperopt-Sklearn that provides automated algorithm selection from models in the Scikit-learn machine learning library [1] similar to Auto-Weka. Feurer et al. introduced improvements upon Hyperopt-Sklearn by taking into account past performance on similar datasets and constructing ensembles from optimized models. What differentiates SML from these prior works is that it provides an agnostic language to reduce the amount programming that one has to write and offers a visualization framework to assess the models performance visually.

[1] See Pedregosa et al. (2011) for an introduction to Scikit-learn.

2.3 Grammar

The SML language is a domain specific language with grammar implemented in Bakus-Naur form (BNF) (see Backus (1959)). Each expression has a rule and can be expanded into additional terms. Listing 2.1 is an example of how one would perform classification on a dataset using SML. The query in listing 2.1 reads from a dataset, performs a 80/20 split of training and testing data respectively, and performs classification on the 5th column of the hypothetical dataset using columns 1,2,3, and 4 as predictors. In the subsequent subsections SML's grammar in BNF form is defined in addition to the keywords.

2.3.1 Grammar Structure

This subsection is dedicated to defining the grammar of SML in terms of BNF. A *Query* can be defined by a delimited list of actions where the delimiter is an AND statement; with BNF syntax this is defined as:

$$< Query >::=< Action > | < Action > AND < Query > \tag{2.1}$$

An *Action* in (2.1) follows one of the following structures defined in (2.2) where a $Keyword$ is required followed by an $Argument$ and/or $OptionList$.

$$\begin{aligned} &< Action >::=< Keyword >< Argument > \\ &| < Keyword >< Argument > (< OptionList >) \\ &| < Keyword > (< OptionList >) \end{aligned} \tag{2.2}$$

A $Keyword$ is a predefined term associating an *Action* with a particular string. An $Argument$ generally is a single string surrounded by quotes that specifies a path to a file. Lastly, an $Argument$ can have a multitude of options (2.3) where an *Option* consist of

an $OptionName$ with either an $OptionValue$ or $OptionValueList$. An $OptionName$ and $OptionValue$ consist of a single string. An $OptionList$ (2.4) consist of a comma delimited list of options and an $OptionValueList$ (2.5) consist of a comma delimited list of $OptionValues$.

$$\begin{aligned} &< Option >::=< OptionName >=< OptionValue > \\ &\quad | < OptionName >= [< OptionValueList >] \end{aligned} \tag{2.3}$$

$$< OptionList >::=< Option > | < Option >, < OptionList > \tag{2.4}$$

$$\begin{aligned} &< OptionValueList >::=< OptionValue > \\ &\quad | < OptionValue >, < OptionValueList > \end{aligned} \tag{2.5}$$

To put the grammar into perspective the example *Query* in Listing 2.1 has been transcribed into BNF format and can be found in Listing 2.2. In Listing 2.2 the first $Keyword$ is $READ$ followed by an $Arugment$ that specifies the path to the dataset. Next an $OptionValueList$ contains information about the delimiter of the dataset and the header. We then include the AND delimiter to specify an additional $Keyword\ SPLIT$ with an $OptionValueList$ that tells us the size of the training and testing partitions for the dataset specified with the $READ\ Keyword$. Lastly, the AND delimiter is used to specify another $Keyword$ $CLASSIFY$ which performs classification using the training and testing data from the result of the $SPLIT\ Keyword$ followed by an $OptionValueList$ which provides information to SML about the features to use (columns 1-4), the label we want to predict (column 5), and the algorithm to use for classification. The next subsection describes the functionality for all $Keyword$s of SML.

2.3.2 Keywords

Currently there are 8 *Keyword*s in SML [2]. These *Keyword*s can be chained together to perform a variety of actions. In the subsequent subsections we describe the functionality of each *Keyword*.

Reading Datasets

When reading data from SML one must use the *READ Keyword* followed by an *Argument* containing a path to the dataset. *READ* also accepts a variety of *Option*s. The first *Query* in listing 2.3 consist of only a *Keyword* and *Argument*. This *Query* reads in data from "/path/to/dataset". The second *Query* includes an *OptionValueList* in addition to reading data from the specified path; the *OptionValueList* specifies that the dataset is delimited with semicolons and does not include a header row.

Cleaning Data

When NaNs, NAs and/or other missing values are present in dataset we clean these values in SML by using the *REPLACE Keyword*. Listing 2.4 shows an example of the *REPLACE Keyword* being used. In this *Query* we use the *REPLACE Keyword* in conjunction with the *READ Keyword*. SML reads from a comma delimited dataset with no header from the path "/path/to/dataset". Then we replace any instance of "NaN" with the mode of that column in the dataset.

Partitioning Datasets

It's often useful to split a dataset into training and testing datasets for most tasks involving machine learning. This can be achieved in SML by using the *SPLIT Keyword*. Listing 2.5

[2]Detailed documentation providing examples and describing all of the keywords of SML are publicly available on github: https://github.com/lcdm-uiuc/sml/tree/master/dataflows

shows an example of a SML *Query* performing a 80/20 split for training and testing data respectively by utilizing the *SPLIT Keyword* after reading in data.

Creating Models

In SML, one can create create a model to either perform classification, regression, or clustering. To use a classification model in SML one would use the *CLASSIFY Keyword*. SML has the following classification models implemented: Support Vector Machines, Naive Bayes, Random Forest, Logistic Regression, and K-Nearest Neighbors. Listing 2.6 demonstrates how to use the *CLASSIFY Keyword* in a *Query*. Clustering models can be utilized by using the *CLUSTER Keyword*. SML currently has K-Means clustering implemented. Listing 2.7 demonstrates how to use the *CLUSTER Keyword* in a *Query*. Regression models use the *REGRESS Keyword*. SML currently has the following regression algorithms implemented: Simple Linear Regression, Ridge Regression, Lasso Regression, and Elastic Net Regression. Listing 2.8 demonstrates how to use the *REGRESS Keyword* in a *Query*.

Saving/Loading Models

It's possible to save models and reuse them later. To save a model in SML one would use the *SAVE Keyword* in a *Query*. To load an existing model from SML one would use the *LOAD Keyword* in a *Query*. Listing 2.9 shows the syntax required to save and load a model using SML. With any of the existing queries using *REGRESS*, *CLUSTER*, or *CLASSIFY Keyword*s attaching *SAVE* to the *Query* will save the model.

Visualizing Datasets and Metrics of Algorithms

When using SML it's possible to visualize datasets or performance of models (such as learning curves or ROC curves). To do this the *PLOT Keyword* must be specified in a *Query*. Listing 2.10 shows an example of how to use the *PLOT Keyword* in a *Query*. We apply the same operations to perform clustering in Listing 2.7 however we utilize the *PLOT Keyword*.

2.4 SML's Architecture

With SML's grammar defined, enough information has been presented to explain SML's architecture. When SML receives a *Query* in the form of a string, it is passed to the parser. The grammar is then used to parse through the string to determine the actions to perform. The actions are stored in a dictionary and given to one of the following phases of SML: Model Phase, Apply Phase, or Metrics Phase. Figure 2.1 shows a block diagram of this process.

The model phase is for constructing a model. The *Keyword*s that generally invoke the model phase are: $READ$, $REPLACE$, $CLASSIFY$, $REGRESS$, $CLUSTER$, and $SAVE$. The apply phase is for applying a preexisting model to new data. The *Keyword* that invokes the apply phase is $LOAD$. It's often useful to visualize the data that one works with and beneficial to see performance metrics of a machine learning model. By default if you specify the $PLOT$ *Keyword* in a *Query*, SML will execute the metrics phase.

The last significant component of SML's architecture is the connector. The connector connects drivers from different libraries and languages to achieve an action a user wants during a particular phase (see Figure 2.2). If one considers applying linear regression on a dataset, during the model phase SML calls the connector to retrieve the linear regression library in this case SML uses sci-kit learn's implementation however, if we wanted to use an algorithm not available in sci-kit learn such as a Hidden Markov Model (HMM) SML will use the connector to call another library that supports HMM.

2.5 Interface

They're multiple interfaces available for working with SML. We have developed an alpha version of a web tool which allows users to write queries and get results back from SML through a web interface (see Figure 2.3). There's also a REPL environment available that allows the user to interactively write queries and displays results from the appropriate phases

of SML. Lastly, users have the option to import SML into an existing pipeline to simplify the development process of apply machine learning to problems.

2.6 Use Cases

We tested SML's framework against ten popular machine learning problems with publicly available data sets. We applied SML to the following datasets: Iris Dataset [3], Auto-MPG Dataset [4], Seeds Dataset [5], Computer Hardware Dataset [6], Boston Housing Dataset [7], Wine Recognition Dataset [8], US Census Dataset [9], Chronic Kidney Disease [10], Spam Detection [11] which were taken from UCI's Machine Learning Repository (see Lichman (2013)). We also applied SML to the Titanic Dataset [12].

As mentioned in footnote 2 there are detailed examples and explanations for all 10 data sets. In this paper we discuss in detail the process of applying SML to the Iris Dataset and the Auto-MPG dataset. In particular we compare the process for using machine learning to solve the problems presented by the datasets with SML against traditionally writing code. We forgo comparing SML to prior works in section 2.2. For both of these data sets we used the same libraries and programming language in SML for writing code to solve these use cases.

Iris Dataset

Listing 2.11 shows all of the code required to perform classification on the Iris dataset using SML in Python. In listing 2.11 data is read in from a specified path named "iris.csv" of

[3]https://archive.ics.uci.edu/ml/datasets/Iris
[4]https://archive.ics.uci.edu/ml/datasets/Auto+MPG
[5]https://archive.ics.uci.edu/ml/datasets/seeds
[6]https://archive.ics.uci.edu/ml/datasets/Computer+Hardware
[7]https://archive.ics.uci.edu/ml/datasets/Housing
[8]https://archive.ics.uci.edu/ml/datasets/Wine
[9]https://archive.ics.uci.edu/ml/datasets/US+Census+Data+(1990)
[10]https://archive.ics.uci.edu/ml/datasets/Chronic_Kidney_Disease
[11]https://archive.ics.uci.edu/ml/datasets/Spambase
[12]https://www.kaggle.com/c/titanic

a subdirectory called "data". performs a 80/20 split, uses the first 4 columns to predict the 5th column, uses support vector machines as the algorithm to perform classification and finally plot distributions of features from the dataset and metrics of the classification model's performance. Appendix A illustrates what is required to perform the same operations using Python and sci-kit learn. The *Query* in listing 2.11 and the code in Appendix A use the same 3rd party libraries implicitly or explicitly. It's also worth noting that the code in Appendix A is publicly available and well documented [13] and it is out of the scope of this paper. Instead the complexities required to produce such results with and without SML are outlined. The result for both snippets of code are the same and can be seen in Figure 2.4.

Auto-Mpg Dataset

Listing 2.12 shows the SML *Query* required to perform regression on the Auto-MPG dataset in Python. In listing 2.12 we read data from a specified path, the dataset is separated by fixed width spaces and we choose not to provide a header for the dataset. Next we perform a 80/20 split, replace all occurrences of "?" with the mode of the column. We then perform linear regression using columns 2-8 to predict the 1st label. Lastly, we visualize distributions of features from the dataset and metrics of our algorithm. Appendix B. demonstrates what's required to perform the same operations using scikit learn [14]. The outcome of both processes are the same and can be seen in Figure 2.5.

2.6.1 Discussion

For the Iris and Auto-MPG use cases the same libraries and programming language were used to perform regression and classification. The amount of work required to perform a task and produce the following results in Figure 2.5 and Figure 2.4 significantly decreases when

[13]For a detailed documentation describing this code visit: https://github.com/lcdm-uiuc/sml/blob/master/dataflows/plot/iris_svm-READ-SPLIT-CLASSIFY-PLOT.ipynb

[14]For a detailed documentation describing this code visit: https://github.com/lcdm-uiuc/sml/blob/master/dataflows/plot/autompg_linear_regression-READ-SPLIT-REGRESS-PLOT.ipynb

SML is utilized. Constructing each SML query used less than 10 lines of code however, implementing the same procedures without SML using the same programming language and libraries needed 70+ lines of code. This provides evidence that SML simplifies the development process of solving problems with machine learning.

2.7 Future Work

While we have formally introduced an agnostic framework a lot of work remains to be done to improve SML. In the future we plan to extend the connector to support more machine learning libraries and additional languages. SML's web application can be extended out to include additional functionality to improve the ease of use. Feature selection, model selection, and parameter optimization are additional areas to add to SML. In addition to improving SML it can also be tested by researchers in a comparative analysis against the works outlined in section 2.2 to determine how beneficial SML is against alternative frameworks.

2.8 Conclusion

To summarize we introduced an agnostic framework that integrates a query-like language to simplify the development of machine learning pipelines. We provided a high level overview of it's architecture and grammar. We then applied SML to machine learning problems and demonstrated how the amount of the code one has to write significantly decreases when SML is used. The source code and detailed documentation for SML is open-sourced and publicly available on github [15].

SML provides a realm of possibility to rapidly develop machine learning pipelines with an agnostic language to solve problems. This attractive aspect can boost the productivity of researchers which utilize machine learning. Abstracting the complexities of machine learning

[15] https://github.com/lcdm-uiuc/sml

with a tool like SML can foster new research and solve problems in different disciplines at a faster rate.

2.9 Figures and Listings

2.9.1 Figures

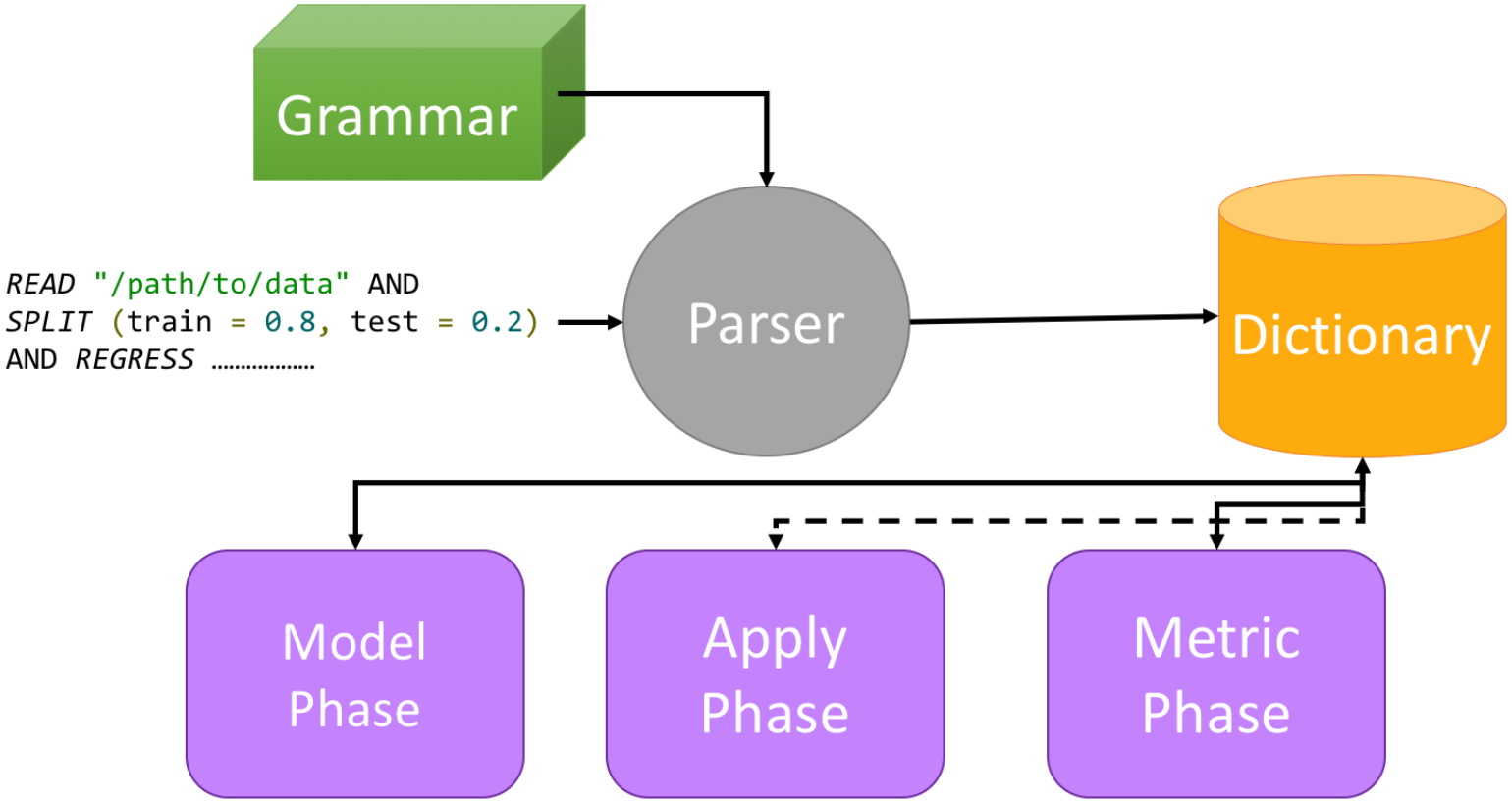

Figure 2.1: Block Diagram of SML's Architecture

![h]

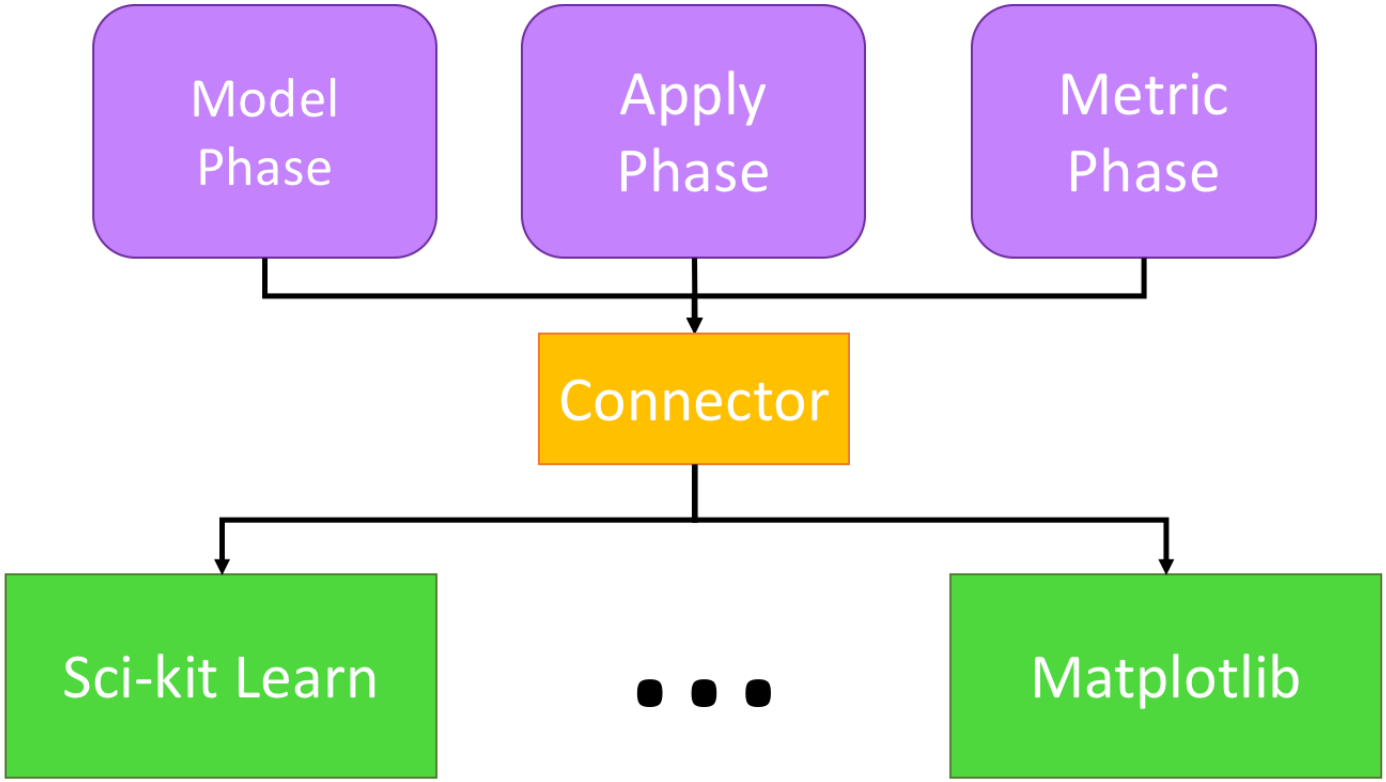

Figure 2.2: Block Diagram of SML's Connector

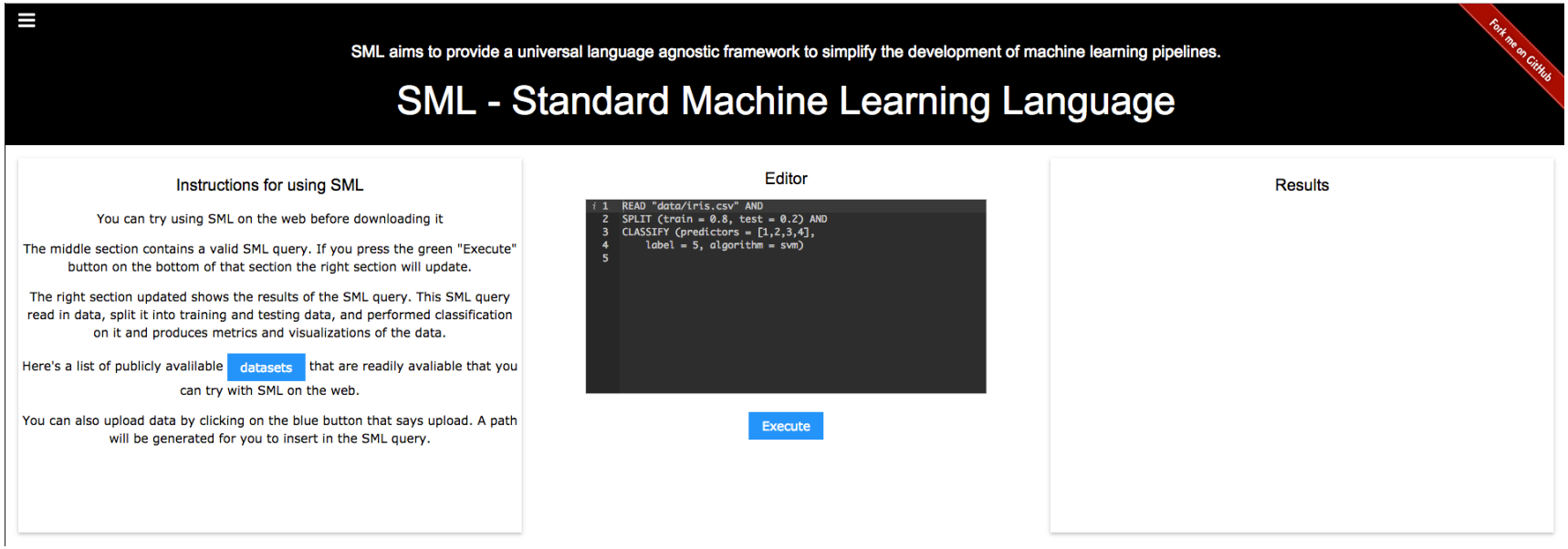

Figure 2.3: Interface of SML's website. Currently users can read instructions and examples of how to use SML are on the left pane. In the middle pane users can type an SML *Query* and then hit the execute button. The results of running the *Query* through SML are then displayed on the right pane.

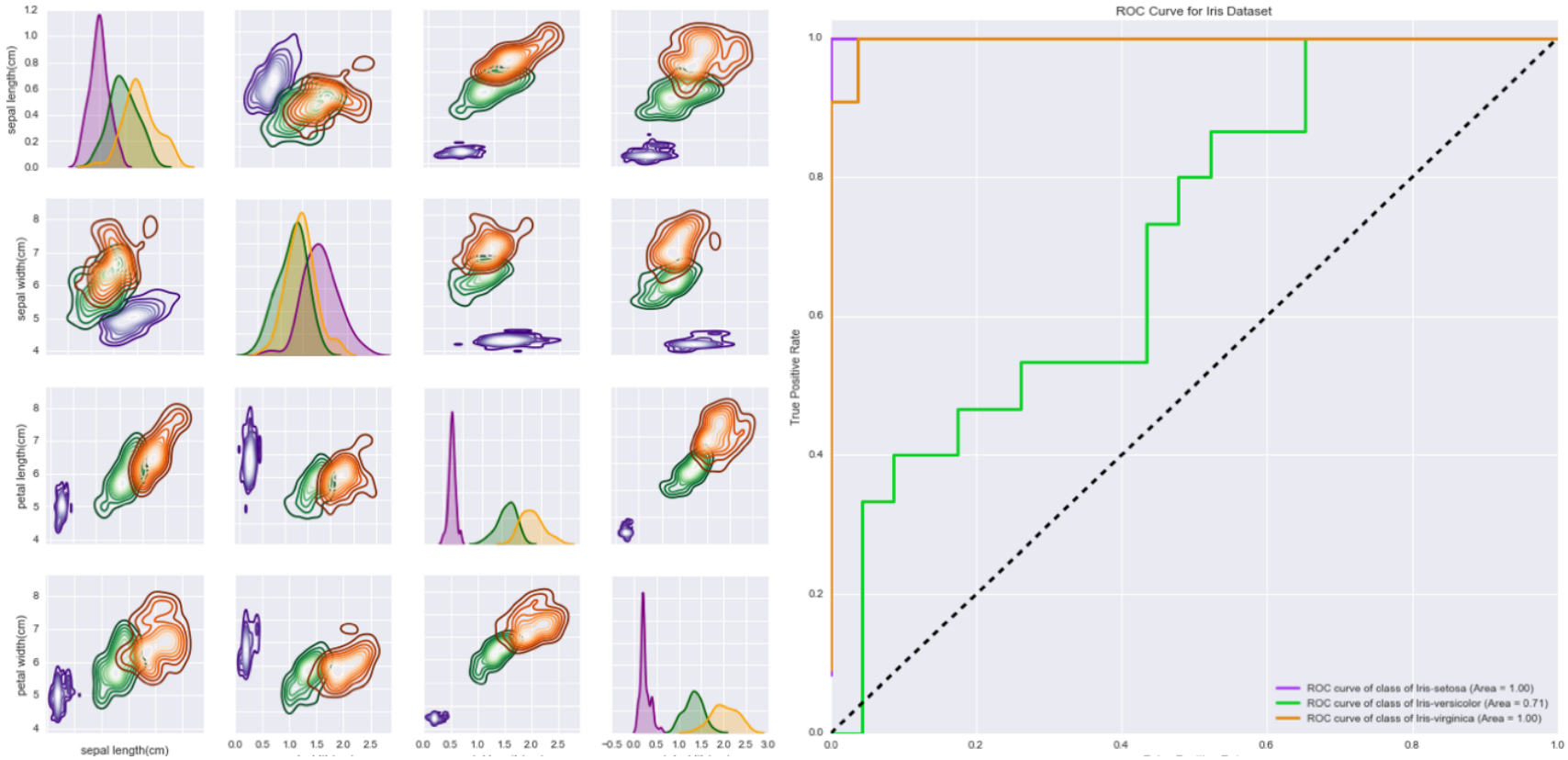

Figure 2.4: The SML *Query* in Figure ?? and the code in Figure ?? produce these results. The subgraph on the left is a lattice plot showing the density estimates of each feature used. The graph on the right shows the ROC curves for each class of the iris dataset.

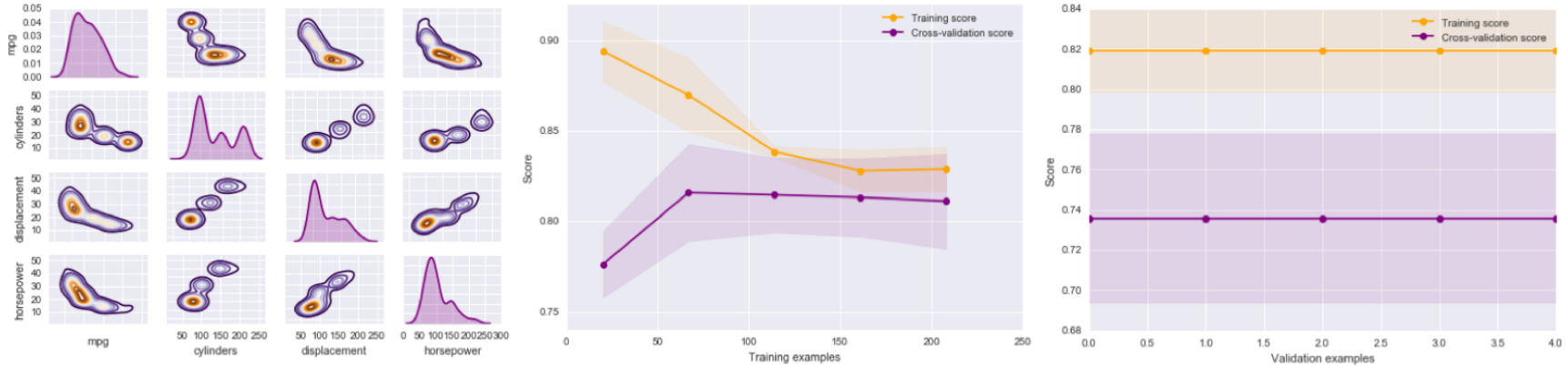

Figure 2.5: The SML *Query* in Figure **??** and the code in Appendix B. produce these results. The subgraph on the left is a lattice plot showing the density estimates of each feature used. The top right graph shows the learning curve of the model and the graph on lower right shows the validation curve.

2.9.2 Listings

```
READ "/path/to/data" (separator = ";", header = None)
AND SPLIT (train = 0.8, test=0.2) AND CLASSIFY
(predictors =[1,2,3,4], label = 5, algorithm = svm)
```

Listing 2.1: Example of a SML Query Performing Classification.

```
<Keyword> <Argument> (<OptionList>)
AND <Keyword> (<OptionList>) AND <Keyword>
(<OptionList>)
```

Listing 2.2: Here the example *Query* in listing 2.1 is defined in BNF format.

```
READ "/path/to/data"
READ "/path/to/data" (separator = ";", header = None)
```

Listing 2.3: Examples using the *READ Keyword* in SML.

```
READ "/path/to/data" (separator = ";", header = None)
AND REPLACE (missing = "NaN", strategy = "mode")
```

Listing 2.4: An example utilizing the *REPLACE Keyword* in SML.

```
READ "/path/to/data" (separator = ";", header = None)
AND SPLIT (train = 0.8, test = 0.2)
```

Listing 2.5: Example using the *SPLIT Keyword* in SML.

```
READ "/path/to/data" (separator = ";", header = None)
AND SPLIT (train = 0.8, test = 0.2) AND CLASSIFY
(predictors = [1,2,3,4], label=5, algorithm=svm)
```

Listing 2.6: Example using the *CLASSIFY Keyword* in SML. Here we read in data and create training and testing datasets using the *READ* and *SPLIT Keywords* respectively. We then use *CLASSIFY Keyword* with the first 4 columns as features and the 5th column to perform classification using a support vector machine.

```
READ "/path/to/data" (separator = ";", header = None)
AND SPLIT (train = 0.8, test = 0.2) AND  CLUSTER
(predictors = [1,2,3,4],  algorithm=kmeans)
```

Listing 2.7: Example using the CLUSTER Keyword in SML. Here we read in data and create training and testing datasets using the READ and SPLIT Keywords respectively. We then use CLUSTER Keyword with the first 4 columns as features and perform unsupervised clustering with the K-Means algorithm.

```
READ "/path/to/data" (separator = ";", header = None)
AND SPLIT (train = 0.8, test = 0.2) AND  CLUSTER
(predictors = [1,2,3,4],  label=5, algorithm=ridge)
```

Listing 2.8: Example using the *REGRESS Keyword* in SML. Here we read in data and create training and testing datasets using the *READ* and *SPLIT Keywords* respectively. We then use *REGRESS Keyword* with the first 4 columns as features and the 5th column to perform regression on using ridge regression.

```
SAVE "/path/to/save/model"
LOAD "/path/to/save/model"
```

Listing 2.9: Example using the *LOAD* and *SAVE Keywords* in SML.

```
READ "/path/to/data" (separator = ";", header = None)
AND SPLIT (train = 0.8, test = 0.2) AND  CLUSTER
(predictors = [1,2,3,4],  algorithm=kmeans)
AND PLOT
```

Listing 2.10: Example using the *PLOT Keyword* in SML.

```
from sml import execute
query = 'READ "../data/iris.csv" AND \
SPLIT (train = 0.8, test = 0.2) AND \
CLASSIFY (predictors = [1,2,3,4],  label = 5, algorithm=svm) AND \
```

```
5 PLOT'
6
7 execute(query, verbose=True)
```

Listing 2.11: SML *Query* that performs classification on the iris dataset using support vector machines. It's important to note that detailed documentation is publicly available in [13] and the purpose of this figure is to highlight the level of the level of complexity relative to an SML query.

```
1 from sml import execute
2 query = 'READ "../data/auto-mpg.csv" AND \
3 REPLACE (missing = "?", strategy = "mode") AND \
4 SPLIT (train = 0.8, test = 0.2) AND \
5 REGRESS (predictors = [2,3,4,5,6,7,8], label = 1, algorithm=simple) AND \
6 PLOT'
7
8 execute(query, verbose=True)
```

Listing 2.12: SML *Query* that performs regression on the Auto-MPG dataset using Linear Regression.

2.10 References

John W. Backus. The syntax and semantics of the proposed international algebraic language of the zurich acm-gamm conference. In *IFIP Congress*, pages 125–131. Butterworths, London, 1959.

K. Bakshi and K. Bakshi. Considerations for artificial intelligence and machine learning: Approaches and use cases. In *2018 IEEE Aerospace Conference*, pages 1–9, 2018. doi: 10.1109/AERO.2018.8396488.

Pedro Domingos. A few useful things to know about machine learning. volume 55, pages 78–87, New York, NY, USA, October 2012. ACM. doi: 10.1145/2347736.2347755. URL `http://doi.acm.org/10.1145/2347736.2347755`.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Auto-sklearn: Efficient and robust automated machine learning. pages 123–143.

E. Frank, M. A. Hall, G. Holmes, R. Kirkby, B. Pfahringer, and I. H. Witten. *Weka: A machine learning workbench for data mining.*, pages 1305–1314. Springer, Berlin, 2005. URL `http://researchcommons.waikato.ac.nz/handle/10289/1497`.

Kelechi Ikegwu, Micheal Hao, Nerraj Asthana, and Robert Brunner. Standard machine learning language: A language agnostic framework for streamlining the development of machine learning pipelines. 2017. URL `https://github.com/lcdm-uiuc/Publications/blob/master/2017_Kelechi_Mike_Brunner/main.pdf`.

Brent Komer, James Bergstra, and Chris Eliasmith. Hyperopt-sklearn. pages 105–121.

Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka: Automatic model selection and hyperparameter optimization in weka. pages 89–103.

M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

S. Monahan. Financial statement analysis and earnings forecasting. 2018.

Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. *CoRR*, abs/1603.06212, 2016. URL `http://arxiv.org/abs/1603.06212`.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

N. Rizzolo and D. Roth. Learning based java for rapid development of nlp systems. In *LREC*, Valletta, Malta, 5 2010. URL `http://cogcomp.cs.illinois.edu/papers/RizzoloRo10.pdf`.

D. Roth. Learning based programming. *Innovations in Machine Learning: Theory and Applications*, 2005. URL `http://cogcomp.cs.illinois.edu/papers/Roth05.pdf`.

Chapter 3

Directional Profitability Predictions with Random Forests

3.1 Introduction

An economic goal of a society is to maximize wealth. In order to maximize wealth, resources such as labor, capital, and natural resources must be allocated efficiently to firms. Capital markets (such as the stock market) allow for capital to be allocated to said firms efficiently. The profitability of firms can effect a society, whether through job growth/loss, services offered, provision of goods, or personal gain/loss (see Spiceland et al. (2018)). Predicting the future profitability of a firm is an important endeavor as it is central to the valuation of the firm and the securities they issue (see Monahan (2018)).

A study by Bradshaw et al. (2012) provides evidence that analysts predictions are not superior to random walk predictions. Monahan (2018) conducts a review and discussion on accounting research related to out of sample profitability predictions. He provides evidence that models in prior works have lower out-of-sample accuracy than random walk models. This provides motivation to seek an alternative method given the short-comings of traditional methods. Sarsam et al. (2020), Hang and Banks (2019), Choudhury et al. (2021), and Yang (2020) are relatively recent examples which exhibit the effectiveness of machine learning for prediction and inference in a variety of applications. Machine learning methods will be

This chapter contains material from the following working paper:

- Vic Anand, Robert Brunner, Kelechi Ikegwu, and Theodore Sougiannis. Predicting profitability using machine learning. 2019. URL `http://dx.doi.org/10.2139/ssrn.3466478`

explored to predict the future profitability of firms.

An approach outlined in Monahan (2018) is to select: what to predict (labels), the predictors (or features), a regression model where the functional form is determined by the researcher, and the training and testing datasets. It is easier with a parametric approach to make an assumption about the functional form of f (see 1.1) and estimate a set parameters however, it is likely that the functional form will not match the true unknown form of f which would provide poor results. Given the objective to find the best f in eq. 1.1 that predicts Y reasonably well, non-parametric methods will be used. In addition to this cross-validation as described in section 1.1.2 will be used in order to avoid overfitting the model. Lastly, machine learning will be used to predict directional changes rather than the magnitude of changes. This choice is made to account for working with a smaller dataset. Ball and Brown (1968) and Ou and Penman (1989) have shown that the future directional change of profitability is a useful indicator.

The research objective is to determine if a machine learning model can outperform a random walk model. This chapter will describe the datasets used in this study and the methods used to predict the directional change of profitability. Then a comparative analyses will be conducted with the proposed model and random walk model; followed by a discussion of results and potential future directions of this research.

3.2 Data

The data selection process described in Hou et al. (2012) is followed in this research. All observations from the fiscal years 1963 - 2016 are used from Compustat fundamentals annual file. This data set is merged with the CRSP monthly returns file which includes all securities listed on the NYSE, Amex, and Nasdaq. The securities selected have share codes 10 or 11 which indicate that they are ordinary common shares of US companies, and exclude ADR's and REITs. Observations are removed if total assets, common equity, dividends, income

before extraordinary items, or accurals are missing which results in 194,172 observations. Given the objective to predict the future direction of profitability for firms each firm must have at least 3 observations; the rationale behind this choice is to be able to compute 2 price changes for a firm. Excluding firms with fewer than 4 years of observations the final dataset consist of 166,925 observations.

The following variables were retrieved and computed for this research: Return on common equity (ROE), Return on assets (ROA), return on net operating assets (RNOA), Cash flow from operations (CFO), and free cash flow (FCF). Table 3.1 contains more details about the variables used from compustat and table 3.2 shows how the profitability measures are computed.

Table 3.3 provides descriptive statistics on variables retrieved from Compustat. Outliers are present in most variables and all of them have positively skewed distributions given that the mean is greater than the median. Table 3.4 provides the descriptive statistics on profitability measures. Since the raw variables are used to construct the profitability measures, their values are also extreme. However, unlike the raw variables, the distributions of the targets are negatively skewed (the mean is smaller than the median). Table 3.5 shows the percentage of times the target variables increase in the holdout period (2012-2015). With the exception of CFO and FCF during 2013, there are no relatively significant class imbalances for directional changes in profitability measures.

Tables 3.6, 3.7, 3.8, 3.9, and 3.10 show the pearson correlations between variables that will serve as features in the random forests. Since correlations of raw variables are extreme due to large outliers, we truncate each variable at 2.5% on each side of its distribution. Many of the features in these tables are correlated nevertheless the proposed machine learning method is insensitive to outliers. Table 3.11 shows the autocorrelations of truncated profitability features. With the exception of CFO, the second-order autocorrelations are marginally smaller than the first order auto correlations for all profitability measures.

3.3 Methods

3.3.1 Cross Validation for Time Series

The supervised learning paradigm described in section 1.1.1 will be employed. 90% of the data is used for training and the remaining 10% is used for testing. The first 90% of data contains data from the fiscal years 1963-2011 and the last 10% of data is from 2011-2016. Given the objective to predict the year to year change we are unable to predict using features for firms in the fiscal year 2016 because the fiscal year 2017 is not in the sample. Given the time-series nature of the data, blocked K-fold cross-validation as described in Cerqueira et al. (2019) is used. The differences between blocked k-fold cross-validation and k-fold cross validation is that the observations are not randomly shuffled and then divided into k folds. Instead the observations are split into K blocks of contiguous observations.

3.3.2 Tree based Machine Learning Methods

Many machine learning models have the potential to provide high accuracy scores however, there is a trade-off between interoperability and performance (see James et al. (2014), Chapter 2). Given that the research objective is more aligned with prediction than inference we place more emphasis on model performance. Tree based models are flexible, simple to implement, and have high performance (see James et al. (2014), Chapter 8). In addition to this tree based methods are able to handle multicollinearity and non linearity better than linear regression methods which are traditionally used in the financial accounting setting (see Monahan (2018)).

3.3.3 Decision Trees

Decision Trees are the building blocks of tree-based machine learning methods. Decision Trees are directed graphs with no cycles meaning there are no nodes with edges to itself and

there is no more than 1 edge connecting two nodes together. Edges in trees are referred to as branches. Consider three nodes node A, B, and C if node A has a branch to nodes B and C then, B and C are the children of node A; node A in this case is the parent of nodes B and C. The node that starts the decision tree does not have a parent and is referred to as the root node. Lastly, a node with no children signify a stopping point in the decision tree and are referred to as a leaf node (see Figure 1.8 for an example).

In a supervised machine learning paradigm, decision trees are given data to train on. Each node in the decision tree will represent the training features (or feature space, i.e $X_1, ..., X_p$) partitioned into particular regions. Using the CART algorithm (see Breiman et al. (1984)) a decision tree will create m distinct non-overlapping regions R. The root node will contain a rule to partition the feature space into 2 non-overlapping regions (R_1 and R_2). The process will repeat recursively i.e. R_1 will be split by a splitting criteria to partition the data into 2 additional regions (R_3 and R_4) and sub-sequentially create 2 additional regions from R_2 (R_5 and R_6).

In a classification setting a decision tree at a given region will determine the most optimal feature to select from the set of features to partition the data. The selection is typically based on the feature's ability to minimize error. A common metric is gini-impurity where, gini-impurity measures the variance across C classes. Gini-impurity can be defined as:

$$G_{index} = \sum_{i=1}^{C} p(i) \cdot (i - p(i)) \tag{3.1}$$

where C are the number of classes in a label and $p(i)$ is the probability of of an observation being labeled class i. A small value of gini-impurity indicates that a particular region has many observations from a single class.

Decision trees have stopping criteria. For example, we stop creating additional regions when a child has less than a defined number of observations at a particular region or when the difference of gini-impurity between the parent and it's children is less than a specified

amount. Lastly, to make a prediction about values of Y from another set of data (containing the same features), new observations can be passed into the trained tree. The trained tree will select the appropriate region for an observation based on the splitting criteria rules.

Decision Trees offer an intuitive explanation of the results it produces. However there are some caveats to this method. A few additional observations or any modification to observations in a feature space can drastically change a decision tree because it is uses a greedy search to build a tree (see Koning and Smith (2017)). Meaning that at each state of the tree building process a node is split using a local optimal choice instead of a choice that will find the optimal choice on a global level.

3.3.4 Random Forests

Random forest have the ability to overcome some of the shortcomings that decision trees face. Random forest are an ensemble of decision trees used to make predictions. However a random forest is less interpretable than a single decision tree. A random forest will create B decision trees. For each decision tree, the random forest will assign a random dataset to each decision tree in the forest by sampling the data with replacement (or bootstrapping). In addition to this the random forest will assign a subset of random features (with replacement) for each decision tree to use.

The rationale behind selecting only a subset of features is to decorrelate trees meaning that if a feature is important it will not always be used as a top level split producing unique decision trees. Each tree can then be built and make predictions using the random data and features in the method outline in the Decision Trees subsection at 3.3.3. To make predictions in a classification setting B decision trees will make a prediction using the features from the bootstrapped data. The most common prediction class is used as the prediction for the forest of trees (which is referred to as majority vote).

3.4 Random Forest Implementation to Predict Directional Profitability

For this research, changes in annual profitability (ROE, ROA, RNOA, CFO, and FCF) are coded as +1 for an increase and -1 for a decrease for individual firms in the data-set described in section 3.2. The features used are described in section 3.6. With data sorted by fiscal year, for a change in a specific profitability measure that feature set (see tables 3.6, 3.7, 3.8, 3.9, and 3.10) the data will be partitioned into training and testing sets. The first 90% of observations (fisical years between 1963-2011) are used to train and build B trees. The remaining 10% of observations (2012-2015) are used to make predictions out of sample.

While building B decision trees, cross-validation as described in 3.3.1 will be employed. The training data will have k folds (where $k = 5$) grouped by fiscal year. This means that each particular fold will consist of observations for particular firm-years. Each decision tree in the forest will bootstrap from each fold and select random features. The decision tree will then train and build a tree based on the methodology outlined in 3.3.3. Then the random forest of trees will make predictions on the validation sets with the process outlined in section 3.3.4. Finally the trained random forest will use make predictions out-of-sample on observations from the test set.

3.5 Random Walk Benchmark Models

The random walk is a model of a stochastic process. Campbell et al. (1997) (hereafter CLM) described it by the following equation:

$$P_t = \mu + P_{t-1} + \epsilon_t \tag{3.2}$$

In eq. 3.2, P_t is the value of some state variable at time t, μ is a drift term that we will assume is 0, and ϵ_t is a disturbance, or increment. CLM note that a common assumption

is that the disturbance term is independent and identically distributed, with mean zero and variance σ^2, i.e. $\epsilon_t \sim IID(o, \sigma^2)$.

The random walk is a mathematical description of a process, not a forecasting tool. However, some papers (i.e Bradshaw et al. (2012)) that wish to use the random walk as a benchmark for their results treat it as a forecasting tool by taking the expected value of both sides of eq. 3.2. That yields the following:

$$E[P_{t+1}] = P_t \Leftrightarrow E[P_{t+1} - P_t] = 0 \tag{3.3}$$

These papers then reason that, since the expected value of P_{t+1} is P_t, the best forecast of P_{t+1} is also P_t. However, the random walk does not actually predict P_{t+1}. It simply says that the expected magnitude of change in P at any time is zero. In other words, if you observe changes in P over a long time series, the average change will be zero.

To be consistent with prior literature, we treat the random walk model as a forecasting tool. However, our work introduces a complication. Prior literature forecasts the magnitude of some variable. By contrast, we forecast the direction of change in our variables. Applying the random walk to our setting therefore requires some additional work. CLM notes that the expected value of ϵ_t is zero, but without additional assumptions, we cannot know the proportion of increases and decreases in ϵ_t.

3.5.1 Random Walk: Benchmark 1

Rearranging equation 3.2 and setting the drift term to zero yields $P_t - P_{t-1} = \epsilon_t$. Thus, the change in profitability is exactly equal to the disturbance term. That implies that the sign of the change in profitability is equal to the sign of the disturbance term. By assumption, $\epsilon_t \sim IID(0, \sigma^2)$, and CLM (p.32) state that "perhaps the most common distributional assumption for the innovations or increments ϵ_t is normality." Normality implies a 50-50 chance of an up or down movement, since the normal distribution is symmetric. Therefore, our first random

walk prediction should simply be that profitability is equally likely to increase or decrease.

Under the assumption that the error term is equally likely to increase or decrease, we find that the random walk model has an expected accuracy of 50%. To see this, assume the testing subsample contains N observations and $0 \leq p \leq 1$ is the fraction of increases. Then p of those observations are increases, and $(1-p)N$ are decreases. For any observation, the random walk will predict increase 50% of the time and decrease 50% of the time. Thus, of the p increases, on average $0.5pN$ will be classified accurately. Similarly, of the $(1-p)N$ decreases, $0.5(1-p)N$ will be classified accurately. For the overall sample, the fraction that will be accurately classified is:

$$Accuracy = \frac{0.5pN + 0.5(1-p)N}{pN + (1-p)N} = \frac{1}{2}$$

Note that the classification accuracy of this interpretation of the random walk is invariant to the fraction of increases in the data. For any value of p, if we assume increases and decreases are equally likely, this random walk model will accurately predict on average the outcome 50% of the time.

3.5.2 Random Walk: Non-normal Error Term

The random walk model requires that the error term has zero mean and finite variance, $\epsilon_t \sim I(0, \sigma^2)$. This restriction does not imply normality of the error term: ϵ_t could have any arbitrary probability of increase as long as its expected value is zero. For example, if $\frac{2}{3}$ of the time ϵ_t increases by 1, and $\frac{1}{3}$ of the time it decreases by 2, it has mean zero. A distributional assumption other than normality may be appropriate if the actual fraction of increases in the data is not 50%.

Anand et al. (2019) showed in Appendix B that even if we assume ϵ_t will increase 40%-60% of the time then the classification accuracy ranges from 48%-52% which is very close to the 50% by assuming a normally distributed error term. Table 3.5 shows that, for most

measures in most years, the actual fraction of increases in our sample are very close to 50%. Even in the most extreme case (in 2013, CFO and FCF increases 43.3% and 42.5% of the time).

3.5.3 Random Walk: Benchmark 2

As an alternative interpretation of applying the random walk model to forecasting, consider the following equations:

$$P_t = P_{t-1} + \epsilon \tag{3.4}$$

$$P_{t-1} = P_{t-2} + \epsilon_{t-1} \tag{3.5}$$

Subtracting eq. 3.5 from eq. 3.4 yields:

$$(P_t - P_{t-1}) = (P_{t-1} - P_{t-2}) + (\epsilon_t - \epsilon_{t-1})$$

$$\Delta P_t = \Delta P_{t-1} + (\epsilon_t - \epsilon_{t-1}) \tag{3.6}$$

If we take the expected value of both sides, equation 3.6 reduces to $\Delta P_t = \Delta P_{t-1}$ since, by assumption, $E[\epsilon_t] = 0, \forall t$. Under this interpretation, in expectation an increase in a profitability measure in period $t-1$ will persist into period t. Thus, our second random walk prediction is that the sign of the change in a measure in one period will persist into the subsequent period.

3.6 Comparative Analyses

To answer the research objective, comparative analyses are conducted between the benchmark random walk models (described in 3.5) and the random forest models (described in 3.3.4). We employ multiple random forest to predict specific profitability measures and add additional features to each random forest for each successive analysis. This provides insight as to whether specific features are informative and yield more accurate estimates.

We perform the analyses in two phases. In the first phase, we test whether Random Forests performs better with winsorized or un-winsorized data. In the second phase we employ multiple random forests to learn the incremental value of additional variables used in fundamental analysis.

3.6.1 Phase 1: Effect of Winsorization

The purpose of phase 1 is to test the effect of winsorization on predictive accuracy. Our prior is that winsorization will have no effect on predictive accuracy of random forests. In phase 1, we use a minimal information set in which each firm-year observation consists of the firm identifier (PERMNO), fiscal year, the contemporaneous value of a measure (e.g. ROE), and the label or change that occurs in the subsequent year (i.e. increase or decrease). We create three random forests for this phase. We train the first random forest on unwinsorized data and refer to this as analysis 1-1. We train the second random forest on data in which the measure has been winsorized at 1% on each side and refer to this as analysis 1-2. We repeat the same methodology as analysis 1-2 where the measure has been winsorized at 2.5% on each side and refer to this analysis as 1-3.

3.6.2 Phase 2: Effect of Adding Additional Fundamental Information

In phase 2, we progressively add information about each firm's industry. We begin with analysis 2-1 where SIC codes are added as a feature. Analysis 2-2 adds industry information. For each firm-year observation, the industry mean, median, and standard deviation for the measure are added. For example consider predicting the change in annual RNOA for IBM during the fisical year 2000. The industry mean, median, and standard deviation of RNOA for IBM's Industry is used as features during the year 2000. Industry is determined by the firm's 4-digit SIC code.

Analysis 2-3 repeats analysis 2-2 while including the lag of the profitability measure. For example if the measure is ROE, then for IBM in the year 2000 it's ROE from 1999 is included as a feature in addition to other features from analysis 2-2. For the final analysis 2-4, analysis 2-3 is repeated while including the 2nd lag of the profitability measure. Following the IBM example for analysis 2-3 it's ROE from 1998 is included as a feature.

3.7 Results

3.7.1 Phase 1 Results and Discussion

Phase 1 tests the predictive power of random forests with minimal information. It also tests the effect of winsorizing the input data, a common practice in accounting research. In phase 1, we create three random forests per profitability measure and use a minimal set of features. For each random forest, the only predictors (features) in each observation are a firm identifier (PERMNO), fiscal year, and the contemporaneous value of the profitability measure. The label is an categorical variable that indicates whether profitability increased or decreased in the subsequent year. The non-categorical measures in the input data are winsorized at three levels, 0%, 1%, and 2.5% on each side. Analysis 1-1 uses a random forest

to make predictions on the non-winsorized data, 1-2 uses the 1% winsorized data, and 1-3 uses the 2.5% winsorized data.

We begin with the default values for model parameters in sci-kit learn version 0.22 (see Pedregosa et al. (2011)). We varied the amount of trees used in a forest for each measure where we try to predict the change in profitability. We found for all measures that there was no significant improvement in performance after using 30 decision trees in a random forest, see Figure 3.1. In Figure 3.1 all subplots share the same x-axis where the x-axis represents the number of trees used in a forest. The y-axis for each subplot represents the accuracy score for a particular measure.

In each of the subplots the dashed line with the circle markers represents the mean training scores during cross-validation. The shaded region surrounding the dashed line with circle markers represent the variance of the training accuracy scores during cross validation. This shaded region is not visible because the variance of the $k-1$ training scores are very small. The solid line with star markers represent the mean cross-validation accuracy score and the shaded in region represents the variance of the cross-validation scores for each fold.

Given the large gap between the mean training and cross-validation accuracy scores it is probable that the random forest are over-fitting the data. This means with the current model parameters (while varying the amount of trees used the random forest) the model is learning information from the training data that is noise. This impedes the model's ability to generalize to the cross-validation data well. To decrease the variance (or flexibility) of the model we vary the depth of each decision tree in the forest.

Figure 3.2 shows the mean training and cross validation scores for a random forest with 30 trees as the maximum depth is varied. As the depth increases the model becomes more flexible. Across all profitability measures one can see that the model is able to learn from the training data fairly well with high training accuracy scores. However the highly flexible models are not able to generalize well to the cross-validation data and perform worst than other models with a smaller maximum depth. Ergo, we select the maximum depth which

yields a model with relatively low variance and bias for each profitability measure. We forgo displaying similar plots for analysis 1-2 and 1-3 because the effect and accuracy scores are similar.

Table 3.12 shows the model maximum depth selected for Analyses 1-1, 1-2, and 1-3. Tables 3.13, 3.14, 3.15, 3.16, and 3.17 show the mean training, mean cross-validation, and testing scores for the ROE, ROA, RNOA, CFO, and FCF measures respectively. In addition to showing the cross-validation scores we show the random walk models performance on the data, and the feature importance of each variable.

The results show that for each measure the random forest is able to outperform the random walk models in predicting the change of profitability with a minimum set of features. In addition to this, excluding the ROE measure we find no significant difference in mean testing accuracies between winsorized and non-winsorized data. Analysis 1-1 (no winsorization) and 1-3 (2.5% per side winsorization) had a mean test accuracy of 55.2% however, analysis 1-2 (1% per side winsorization) had a mean test accuracy of 56.8%. The remaining analyses for the other profitability measures have a mean testing accuracies within 1% of each other. This provides evidence that winsorization has a negligible effect on performance out of sample.

To summarize, traditional approaches have struggled in the past to out-perform random walks. With a minimum amount of features (company identifier, profitability measure, and fiscal year), we find that random forest are able to out-perform random walks in predicting annual profitability changes. In addition to this we find that winsorizing the data (which is a traditional method used to handle outliers in financial accounting) has a negligible effect on the performance of a random forest model.

3.7.2 Phase 2 Results and Discussion

Given that there is no significant effect of winsorization with respect to the random forest models performance we forgo using the winsorized datasets. Additional random forests are

created where each successive forest incrementally adds features. In analysis 2-1 the firm's industry is included as a feature. Analysis 2-2 includes descriptive statistics about the firm's industry at a given year. The specific statistics used are mean, median, and standard deviation. Analysis 2-3 adds the previous lag value of the profitability measure (i.e the profitability measure at $t-1$) and analysis 2-4 includes the two previous lag values of the profitability measure (i.e $t-1, t-2$). Tables 3.18, 3.19, 3.20, 3.21, and 3.22 report the results from the 2nd set of analyses.

Similar to the results of the first set analyses all random forest models out perform the random walk models. Comparing the second set of analyses to analysis 1-1 (non winsorizied data with a minimum set of features), we find mixed results for each profitability measure. For CFO and FCF we find improvements across all set of analyses. However for ROE, ROA, and RNOA there is little to no improvement from analysis 1-1. We find that CFO and FCF random forest models utilize the feature CFO_t and FCF_t the most throughout all analyses. This provides evidence that the best feature to predict the future directional change of a measure is the current value of the measure for CFO and FCF.

RNOA, ROA, and ROE set of analyses still utilize $RNOA_t$, ROA_t, and ROE_t respectively more than any other feature however, they utilize it significantly less than the random forest models that predict CFO and FCF. For example in analysis 2-3 for RNOA, ROA, ROE, CFO, and FCF the previous lagged measures are the 2^{nd} most used by the model. For CFO and FCF the feature importance of CFO and FCF are 69.2% and 78.5% more than the lagged value. However the ROE, ROA, and RNOA measures use their current respective values at t 45.7%, 50.3%, and 45.8% more than the lagged values. In addition to this we find that ROE's performance improves more than ROA and RNOA. This further provides evidence that the most information about future profitability changes are in the current value of the profitability measure.

3.8 Conclusion and Future Work

We explored the utility of random forest trees in a supervised learning paradigm to predict the annual direction of profitability for firms with minimal information. The motivation of this study stems from the fact that accounting literature shows that traditional regression methods cannot produce models that are superior than random walk models at predicting out of sample. In addition to this we explore the advantages of following a supervised learning paradigm to predict directional change. For example random forests are insensitive to multicollinearity and properly utilizing the supervised paradigm discovers a functional form which generalizes well to new data.

We implement and train random forest trees in a classification setting using a large sample of US firms over the period 1963-2011. We generate out-of-sample predictions of directional changes (increases or decreases) in five profitability measures: return on equity (ROE), return on assets (ROA), return on net operating assets (RNOA), cash flow from operations (CFO), and free cash flow (FCF) from 2011-2016. We found that the classification accuracy for each measure outperformed the random walk models. For the out of sample predictions the accuracy does not tend to significantly decline with a four year forecast horizion. We found that models with a strong reliance on the current value of a profitability measure provided more accurate estimates of the future directional change in profitability.

These results can be further improved by selecting a model that yields better performance but offers less interpretability. Given that we focus on a minimal set of features in this research we can incorporate additional features for additional performance gains. Utilizing more data we can extend the methodology outlined in this paper to a regression setting and make predictions about the magnitudes of profitability.

The results shown in the paper are promising since that we can outperform a random walk model for all of the profitability measures used in this study. The methods used in this chapter are robust to econometric issues such as multicollinearity and outliers. When used

correctly the methodology outlined in this paper can produce models that generalize well to out of sample data. Machine learning has simplified and solved problems in many aspects of our lives. The results show the benefit that it can have on predicting the profitability of firms in an financial accounting research setting.

3.9 Figures and Tables

3.9.1 Figures

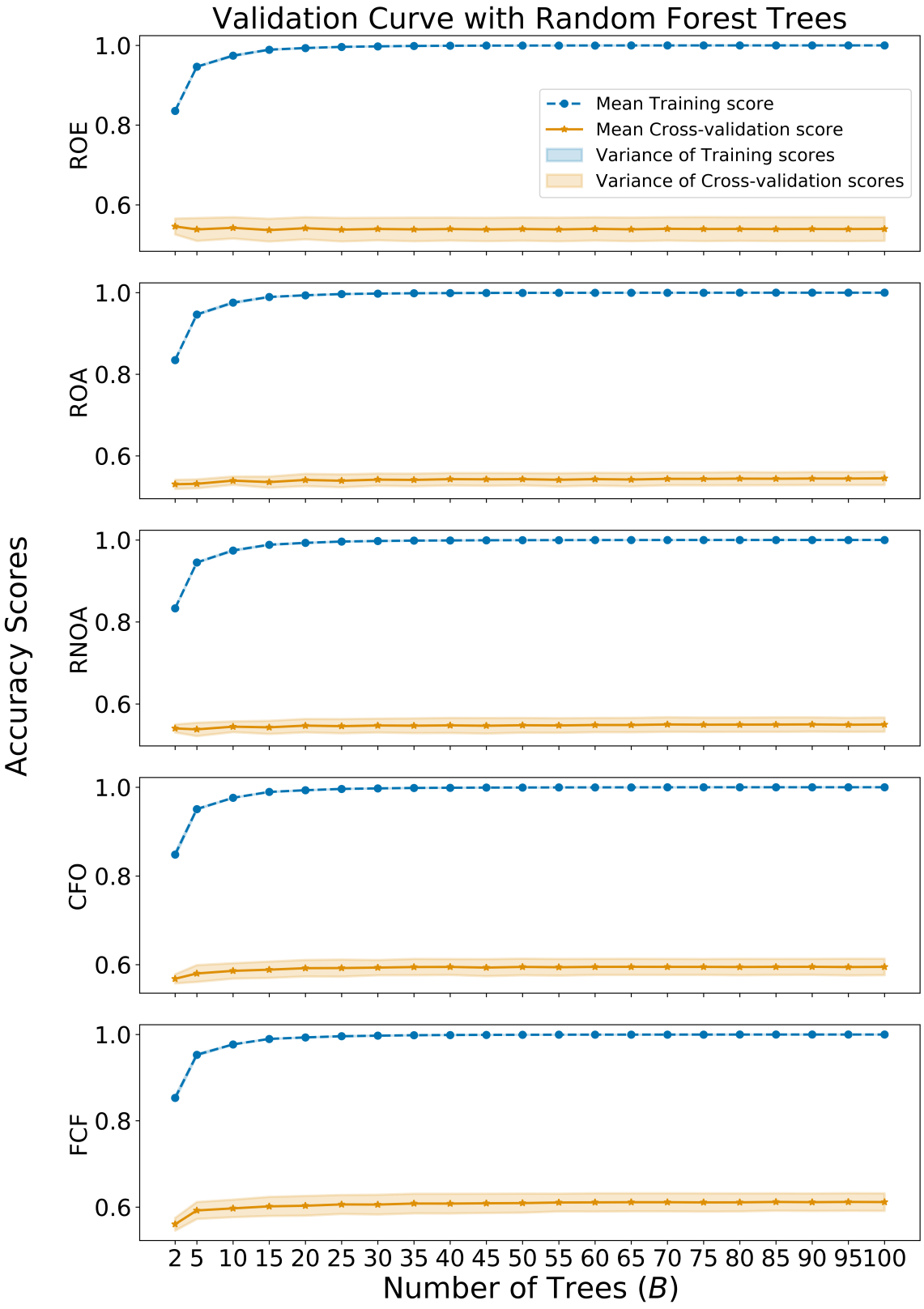

Figure 3.1: This figure shows validation curves for each measure in Analysis 1-1 varying the amount of trees.

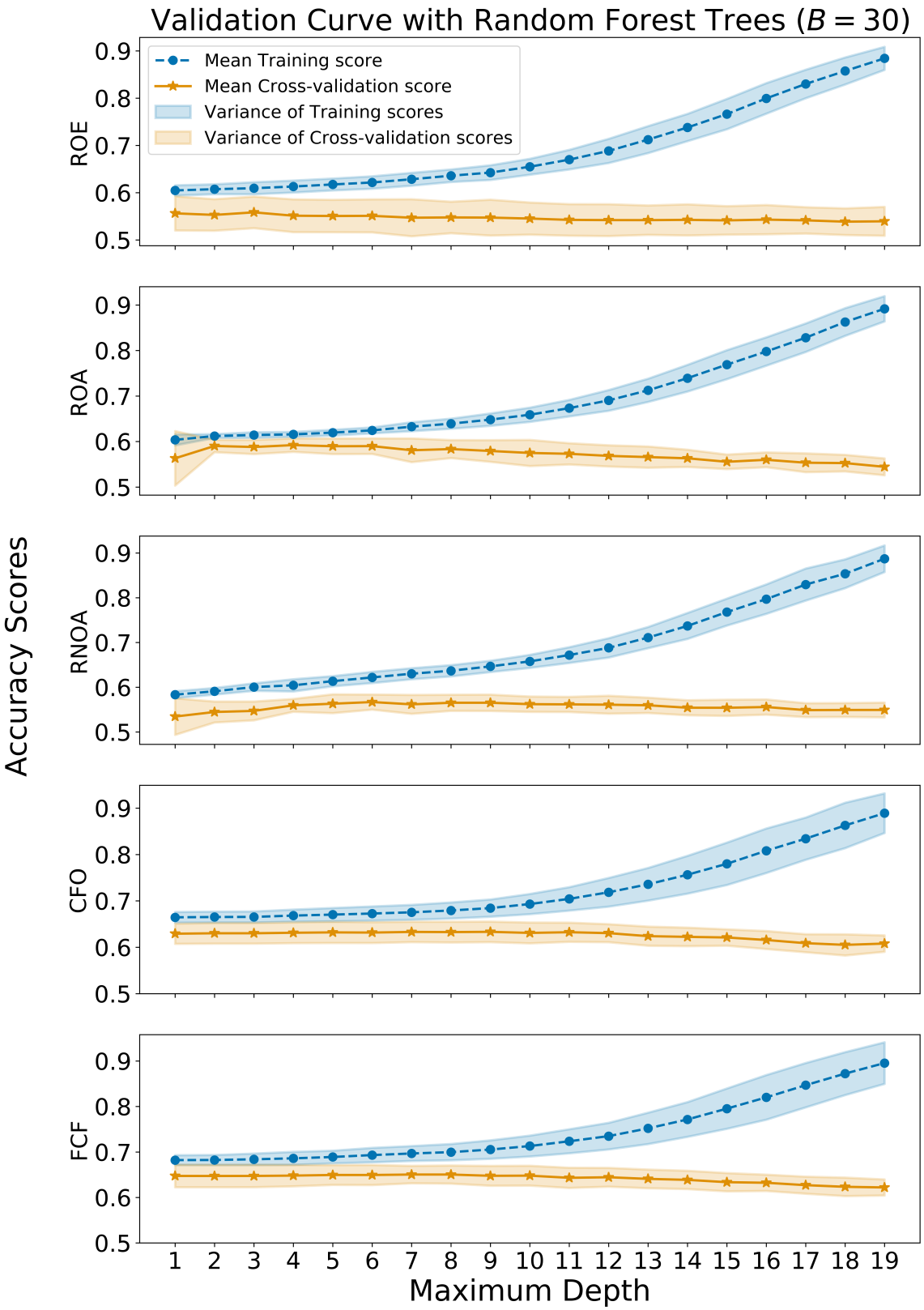

Figure 3.2: This figure shows the validation curves for each measure in Analysis 1-1 varying the max depth using 30 trees in a forest.

3.9.2 Tables

Variable	**Code**
Current Assets – Total	ACT
Assets – Total	AT
Capital Expenditures	CAPX
Common Equity – Total	CEQ
Cash and Short-Term Investments	CHE
Debt in Current Liabilities (short-term debt)	DLC
Long-Term Debt – Total	DLTT
Depreciation and Amortization	DP
Data year – fiscal	FYEAR
Earnings (income before extraordinary items)	IB
Earnings (income before extraordinary items, from statement of cash flows)	IBC
Interest and Related Income – Total	IDIT
Short-term investments – Total	IVST
Current Liabilities – Total	LCT
Liabilities – Total	LT
Operating Activities – Net Cash Flow	OANCF
Operating Income before Depreciation	OIBDP
CRSP permanent number	PERMNO
Property, Plant, and Equipment – Total (Net)	PPENT
Standard Industry Classification	SIC
Income Taxes Payable	TXP
Extraordinary Items and Discontinued Operations (Statement of Cash Flows)	XIDOC
Interest Expense	XINT

Table 3.1: Computstat Variables

Variable	Definition
ROE_t	$\frac{IB_t}{0.5(CEQ_t+CEQ_{t-1})}$
ROA_t	$\frac{IB_t+XINT_t}{0.5(A_t+A_{t-1})}$
$RNOA_t$	$\frac{OI_t}{0.5\times(NOA_t+NOA_{t-1})}$
CFO_t	$\frac{OANCF_t}{0.5(A_t+A_{t-1})}$
FCF_t	$\frac{OANCF_t-CAPX_t}{0.5(A_t+A_{t-1})}$

Table 3.2: Profitability Measures

Variable	Count	Mean	Std	Min	1%	25%	50%	75%	99%	Max
AT	167,718.00	3,344.65	37,775.67	0	2.08	32.21	139.24	754.55	45,009.92	2,573,126.00
CAPX	167,213.00	103.64	665.28	-401.61	0	0.99	5.29	30.46	1,833.88	37,985.00
CEQ	167,718.00	733.70	4,746.54	-59,640.00	-46.55	13.94	59.19	272.44	11,881.49	255,550.00
DP	166,456.00	73.70	465.94	-7.91	0.02	0.95	4.20	23.01	1,250.90	22,016.00
IB	167,718.00	85.78	859.57	-99,289.00	-227.97	-0.32	3.38	25.86	1,844.83	53,394.00
OANCF	167,718.00	181.91	1,590.52	-110,560.00	-81.17	0.10	6.70	51.69	3,353.03	129,731.00
OIBDP	167,026.00	264.67	1,742.21	-76,735.00	-53.35	1.57	13.19	81.83	4,516.75	81,730.00
PPENT	166,826.00	620.94	3,822.74	0	0.05	4.86	24.81	154.47	12,305.75	252,668.00
XINT	167,718.00	49.38	639.00	-0.88	0	0.12	1.28	10.41	648.83	57,302.00

Table 3.3: Descriptive statistics for Compustat variables.

Variable	Count	Mean	Std	Min	1%	25%	50%	75%	99%	max
ROE	160,653.00	0.00	0.50	-4.47	-2.52	-0.01	0.10	0.17	1.10	2.97
ROA	160,653.00	0.02	0.18	-1.33	-0.85	0.01	0.06	0.10	0.26	0.35
RNOA	160,653.00	0.07	1.07	-12.03	-4.00	0.02	0.10	0.17	3.64	10.66
CFO	160,653.00	0.04	0.16	-0.91	-0.68	0.01	0.07	0.12	0.34	0.41
FCF	160,653.00	-0.03	0.17	-1.07	-0.76	-0.07	0.01	0.06	0.27	0.34

Table 3.4: Descriptive statistics for profitability measures.

Fisical Year	ROE	ROA	RNOA	CFO	FCF
2012	47.6%	48.4%	47.9%	49.4%	51.9%
2013	48.9%	49.7%	47.6%	43.3%	42.5%
2014	46.6%	45.5%	44.7%	49.1%	51.5%
2015	48.4%	51.0%	50.2%	50.8%	53.6%

Table 3.5: Percent increases of annual profitability.

	ROE	Ind ROE Mean	Ind ROE Median	Ind ROE Std	ROE_L1	ROE_L2
ROE	1.00	0.20	0.39	-0.11	0.53	0.39
Ind ROE Mean		1.00	0.35	-0.63	0.14	0.12
Ind ROE Median			1.00	-0.19	0.33	0.29
Ind ROE Std				1.00	-0.12	-0.10
ROE_L1						0.54
ROE_L2						1.00

Table 3.6: Pearson Correlations for ROE features.

	ROA	Ind ROA Mean	Ind ROA Median	Ind ROA Std	ROA_L1	ROA_L2
ROA	1.00	0.53	0.49	-0.42	0.74	0.63
Ind ROA Mean		1.00	0.89	-0.84	0.48	0.43
Ind ROA Median			1.00	-0.60	0.45	0.42
Ind ROA Std				1.00	-0.38	-0.36
ROA_L1					1.00	0.74
ROA_L2						1.00

Table 3.7: Pearson Correlations for ROA features.

	RNOA	Ind RNOA Mean	Ind RNOA Median	Ind RNOA Std	RNOA_L1	RNOA_L2
RNOA	1.00	0.10	0.24	0.01	0.36	0.22
Ind RNOA Mean		1.00	0.25	0.26	0.06	0.03
Ind RNOA Median			1.00	0.09	0.18	0.14
Ind RNOA Std				1.00	0.01	0.01
RNOA_L1					1.00	0.36
RNOA_L2						1.00

Table 3.8: Pearson Correlations for RNOA features.

	FCF	**Ind FCF Mean**	**Ind FCF Median**	**Ind FCF Std**	**FCF_L1**	**FCF_L2**
FCF	1.00	0.49	0.45	-0.31	0.62	0.53
Ind FCF Mean		1.00	0.89	-0.71	0.39	0.36
Ind FCF Median			1.00	-0.46	0.37	0.33
Ind FCF Std				1.00	-0.28	-0.26
FCF_L1					1.00	0.61
FCF_L2						1.00

Table 3.9: Pearson Correlations for FCF features.

	CFO	**Ind CFO Mean**	**Ind CFO Median**	**Ind CFO Std**	**CFO_L1**	**CFO_L2**
CFO	1.00	0.50	0.47	-0.32	0.67	0.60
Ind CFO Mean		1.00	0.90	-0.70	0.42	0.40
Ind CFO Median			1.00	-0.47	0.40	0.38
Ind CFO Std				1.00	-0.29	-0.28
CFO_L1					1.00	0.66
CFO_L2						1.00

Table 3.10: Pearson Correlations for CFO features.

	1 Lag	2 Lags
ROE	0.16	0.15
ROA	0.28	0.27
RNOA	0.06	0.04
FCF	0.22	0.21
CFO	0.23	0.23

Table 3.11: Autocorrelations in Profitability Measures.

Analysis (**Degree of Winsorization**)	**1-1** (**None**)	**1-2** (**1% per side**)	**1-3** (**2.5% per side**)
ROE	3	2	3
ROA	2	2	2
RNOA	4	2	4
CFO	2	2	2
FCF	2	2	2

Table 3.12: Shows the maximum depth selected for each analysis and measure combination with $B = 30$.

	Return on Equity (ROE)		
	Analysis (Degree of Winsorization)		
	1-1 (None)	**1-2 (1% per side)**	**1-3 (2.5% per side)**
Mean Training Accuracy	61.1%	61.4%	61.1%
Mean Cross Validation Accuracy	55.5%	58.5%	55.5%
Mean Testing Accuracy	55.2%	56.8%	55.2%
Random Walk 1	50%		
Random Walk 2	48%		
Test Scores by Year			
2012	57.5%	60.4%	57.5%
2013	53.8%	57.1%	53.8%
2014	55.2%	54.2%	55.2%
2015	54.1%	55.4%	54.1%
Feature Importance			
PERMNO	2.7%	2.7%	2.7%
Fisical Year	19%	30%	19%
ROE	78.2%	67.2%	78.2%

Table 3.13: Analyses 1-1, 1-2, and 1-3 results for the profitability measure ROE.

	Return on Equity (ROA)		
	Analysis (Degree of Winsorization)		
	1-1 (None)	**1-2 (1% per side)**	**1-3 (2.5% per side)**
Mean Training Accuracy	61.1%	61.2%	61.1%
Mean Cross Validation Accuracy	57.9%	58.1%	57.9%
Mean Testing Accuracy	58.5%	58.5%	58.5%
Random Walk 1	50%		
Random Walk 2	48.1%		
Test Scores by Year			
2012	60.6%	60.4%	60.6%
2013	57.6%	58.2%	57.6%
2014	58.9%	58%	58.9%
2015	56.8%	57.2%	56.8%
Feature Importance			
PERMNO	2.6%	2.5%	2.6%
Fisical Year	32.2%	32.2%	32.2%
ROA	65.2%	65.2%	65.2%

Table 3.14: Analyses 1-1, 1-2, and 1-3 results for the profitability measure ROA.

	Return on Net Operating Assets (RNOA)		
	Analysis (Degree of Winsorization)		
	1-1 (None)	**1-2 (1% per side)**	**1-3 (2.5% per side)**
Mean Training Accuracy	61.2%	62.2%	61.2%
Mean Cross Validation Accuracy	56.5%	59.8%	56.5%
Mean Testing Accuracy	54.1%	53.3%	54.1%
Random Walk 1	50%		
Random Walk 2	47.6%		
Test Scores by Year			
2012	59%	55.6%	59%
2013	50%	53.3%	50%
2014	52.7%	50.1%	53%
2015	54.1%	54.3%	54.2%
Feature Importance			
PERMNO	3.2%	1.8%	3.3%
Fisical Year	16.2%	28.4%	16.2%
RNOA	80.5%	69.8%	80.5%

Table 3.15: Analyses 1-1, 1-2, and 1-3 results for the profitability measure RNOA.

	Cash Flow from Operations (CFO)		
	Analysis (Degree of Winsorization)		
	1-1 (None)	**1-2 (1% per side)**	**1-3 (2.5% per side)**
Mean Training Accuracy	66.5%	66.5%	66.5%
Mean Cross Validation Accuracy	63%	63.1%	0.63%
Mean Testing Accuracy	58.7%	58.5%	58.6%
Random Walk 1	50%		
Random Walk 2	38.2%		
Test Scores by Year			
2012	62.5%	62.2%	62.5%
2013	55.2%	54.9%	55.2%
2014	59%	58.9%	59%
2015	57.9%	0.58%	57.8%
Feature Importance			
PERMNO	7.6%	7.6%	7.6%
Fisical Year	19.6%	19.6%	19.5%
CFO	72.8%	72.8%	72.8%

Table 3.16: Analyses 1-1, 1-2, and 1-3 results for the profitability measure CFO.

	Free Cash Flow (FCF)		
	Analysis (Degree of Winsorization)		
	1-1 (None)	**1-2 (1% per side)**	**1-3 (2.5% per side)**
Mean Training Accuracy	68.3%	68.4%	68.3%
Mean Cross Validation Accuracy	64.7%	64.8%	64.7%
Mean Testing Accuracy	59.1%	59.1%	59.1%
Random Walk 1	50%		
Random Walk 2	39%		
Test Scores by Year			
2012	63.2%	63.2%	63%
2013	55.6%	55.7%	55.6%
2014	60%	60%	59.8%
2015	57.9%	57.6%	57.9%
Feature Importance			
PERMNO	3.7%	3.7%	3.7%
Fisical Year	20.4%	20.4%	20.5%
FCF	75.8%	75.8%	75.8%

Table 3.17: Analyses 1-1, 1-2, and 1-3 results for the profitability measure FCF.

	Return on Equity (ROE)			
	Analysis			
	2-1	**2-2**	**2-3**	**2-4**
Mean Train Accuracy	60.5%	62.4%	62.4%	62.5%
Mean Cross Validation Accuracy	55.4%	56.0%	57.2%	56.4%
Mean Test Accuracy	55.0%	56.6%	56.8%	56.5%
Random Walk 1	50%			
Random Walk 2	48%			
Test Scores by Year				
2012	57.2%	58.2%	60.0%	60.0%
2013	53.2%	55.4%	54.8%	55.3%
2014	55.4%	57.4%	56.5%	54.2%
2015	53.9%	55.5%	55.9%	56.4%
Feature Importance				
PERMNO	10.8%	2.8%	2.8%	2.1%
FYEAR	32.8%	13.5%	8.8%	10.8%
ROE	50.1%	68.7%	61.1%	52.5%
SIC	6.3%	1.2%	1.0%	1.0%
Industry ROE Mean		5.2%	5.0%	4.3%
Industry ROE Median		5.3%	3.3%	3.0%
Industry ROE std.		3.3%	2.6%	2.5%
ROE_{t-1}			15.4%	18.7%
ROE_{t-2}				5.4%

Table 3.18: Analyses 2-1, 2-2, 2-3, and 2-4 results for the profitability measure ROE.

	Return on Assets (ROA)			
	Analysis			
	2-1	**2-2**	**2-3**	**2-4**
Mean Training Accuracy	62.2%	61.9%	61.4%	61.4%
Mean Cross Validation Accuracy	59.0%	58.3%	58.7%	57.3%
Mean Testing Accuracy	58.6%	58.1%	58.1%	58.3%
Random Walk 1	50%			
Random Walk 2	48.1%			
Test Scores by Year				
2012	60.8%	60.7%	60.4%	60.5%
2013	58.2%	58.5%	57.8%	57.7%
2014	58.5%	57.6%	58.3%	58.4%
2015	56.8%	55.7%	55.9%	56.8%
Feature Importance				
PERMNO	4.0%	1.1%	0.8%	0.7%
FYEAR	15.9%	13.3%	5.0%	7.4%
ROA	78.3%	70.1%	66.7%	56.0%
SIC	1.8%	1.1%	0.8%	0.7%
Industry ROA Mean		5.0%	3.6%	3.5%
Industry ROA Median		7.4%	5.6%	4.0%
Industry ROA std.		2.1%	1.0%	0.8%
ROA_{t-1}			16.5%	21.3%
ROA_{t-2}				5.7%

Table 3.19: Analyses 2-1, 2-2, 2-3, and 2-4 results for the profitability measure ROA.

Return on Net Operating Assets (RNOA)

	Analysis			
	2-1	**2-2**	**2-3**	**2-4**
Mean Train Accuracy	60.8%	60.8%	61.9%	61.9%
Mean Cross Validation Accuracy	55.9%	56.3%	57.1%	56.6%
Mean Testing Accuracy	52.4%	52.7%	52.5%	52.8%
Random Walk 1	50%			
Random Walk 2	47.6%			
Test Scores by Year				
2012	59.0%	57.0%	57.4%	57.8%
2013	49.2%	49.8%	52.2%	52.1%
2014	47.5%	49.6%	46.9%	47.8%
2015	53.8%	54.4%	53.5%	53.4%
Feature Importance				
PERMNO	3.6%	1.0%	1.4%	0.9%
FYEAR	15.3%	12.2%	6.7%	7.8%
$RNOA$	79.6%	63.0%	61.7%	56.6%
SIC	1.5%	0.6%	0.9%	0.8%
Industry RNOA Mean		6.7%	4.5%	4.1%
Industry RNOA Median		14.3%	6.2%	6.8%
Industry RNOA std.		2.1%	2.7%	2.2%
$RNOA_{t-1}$			15.9%	16.2%
$RNOA_{t-2}$				4.5%

Table 3.20: Analyses 2-1, 2-2, 2-3, and 2-4 results for the profitability measure RNOA.

Cash Flow from Operations (CFO)

	Analysis			
	2-1	**2-2**	**2-3**	**2-4**
Mean Training Accuracy	67.5%	67.8%	68.8%	69.2%
Mean Cross Validation Accuracy	63.9%	64.0%	64.5%	64.6%
Mean Testing Accuracy	60.2%	59.8%	60.2%	60.1%
Random Walk 1	50%			
Random Walk 2	38.2%			
Test Scores by Year				
2012	62.6%	63.2%	64.1%	63.2%
2013	62.6%	58.5%	57.5%	58.1%
2014	59.0%	60.1%	60.4%	60.3%
2015	56.5%	57.3%	58.7%	58.7%
Feature Importance				
PERMNO	2.4%	1.6%	1.8%	1.4%
FYEAR	4.9%	3.5%	3.2%	2.8%
CFO	91.1%	82.1%	75.8%	75.4%
SIC	1.6%	1.7%	1.6%	1.4%
Industry CFO Mean		3.8%	4.5%	3.7%
Industry CFO Median		5.4%	4.4%	4.5%
Industry CFO std.		1.9%	2.1%	1.9%
CFO_{t-1}			6.6%	5.1%
CFO_{t-2}				3.8%

Table 3.21: Analyses 2-1, 2-2, 2-3, and 2-4 results for the profitability measure CFO.

	Free Cash Flow (FCF)			
	Analysis			
	2-1	**2-2**	**2-3**	**2-4**
Mean Training Accuracy	69.2%	69.5%	69.5%	69.6%
Mean Cross Validation Accuracy	65.2%	65.2%	65.5%	65.4%
Mean Testing Accuracy	60.8%	60.7%	61.3%	61.7%
Random Walk 1	50%			
Random Walk 2	39%			
Test Scores by Year				
2012	62.3%	64.1%	62.2%	63.9%
2013	63.4%	60.4%	63.9%	63.3%
2014	60.6%	60.6%	61.1%	61.5%
2015	56.9%	57.8%	57.9%	58.2%
Feature Importance				
PERMNO	2.1%	1.0%	1.0%	0.9%
FYEAR	5.0%	4.1%	2.5%	2.4%
FCF	91.5%	82.1%	82.2%	81.1%
SIC	1.3%	1.1%	1.0%	0.8%
Industry FCF Mean		3.9%	4.4%	2.8%
Industry FCF Median		6.0%	3.8%	5.0%
Industry FCF std.		1.9%	1.6%	1.6%
FCF_{t-1}			3.6%	3.2%
FCF_{t-2}				2.1%

Table 3.22: Analyses 2-1, 2-2, 2-3, and 2-4 results for the profitability measure FCF.

3.10 References

Vic Anand, Robert Brunner, Kelechi Ikegwu, and Theodore Sougiannis. Predicting profitability using machine learning. 2019. URL http://dx.doi.org/10.2139/ssrn.3466478.

Ray Ball and P Brown. Empirical evaluation of accounting income numbers. *Journal of Accounting Research*, 6(2):159–178, 1968.

Mark T. Bradshaw, Michael S. Drake, James N. Myers, and Linda A. Myers. A re-examination of analysts' superiority over time-series forecasts of annual earnings. *Review of Accounting Studies*, 2012. URL https://doi.org/10.1007/s11142-012-9185-8.

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

John Y. Campbell, Andrew W. Lo, and A.Craig MacKinlay. *The Econometrics of Financial Markets*. Princeton University Press, 1997. ISBN 9780691043012. URL http://www.jstor.org/stable/j.ctt7skm5.

Vitor Cerqueira, Luis Torgo, and Igor Mozetic. Evaluating time series forecasting models: An empirical study on performance estimation methods, 2019.

Prithwiraj Choudhury, Ryan T. Allen, and Michael G. Endres. Machine learning for pattern discovery in management research. *Strategic Management Journal*, 42(1):30–57, 2021. doi: https://doi.org/10.1002/smj.3215. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/smj.3215.

Weiqiang Hang and Timothy Banks. Machine learning applied to pack classification. *International Journal of Market Research*, 61(6):601–620, 2019. doi: 10.1177/1470785319841217.

Kewei Hou, Mathijs A. van Dijk, and Yinglei Zhang. The implied cost of capital: A new approach. *Journal of Accounting and Economics*, 53(3):504 – 526, 2012. ISSN 0165-4101. doi: https://doi.org/10.1016/j.jacceco.2011.12.001. URL http://www.sciencedirect.com/science/article/pii/S0165410111000966.

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014. ISBN 1461471370.

M. Koning and C. Smith. *Decision Trees and Random Forests: A Visual Introduction for Beginners*. Amazon Digital Services LLC - Kdp Print Us, 2017. ISBN 9781549893759. URL https://books.google.com/books?id=Hi_CtAEACAAJ.

S. Monahan. Financial statement analysis and earnings forecasting. 2018.

Jane A. Ou and Stephen H. Penman. Financial statement analysis and the prediction of stock returns. *Journal of Accounting and Economics*, 11(4):295 – 329, 1989. ISSN 0165-4101. doi: https://doi.org/10.1016/0165-4101(89)90017-7. URL `http://www.sciencedirect.com/science/article/pii/0165410189900177`.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Samer Muthana Sarsam, Hosam Al-Samarraie, Ahmed Ibrahim Alzahrani, and Bianca Wright. Sarcasm detection using machine learning algorithms in twitter: A systematic review. *International Journal of Market Research*, 62(5):578–598, 2020. doi: 10.1177/1470785320921779.

J. David Spiceland, James F. Sepe, and Mark Nelson. ntermediate accounting. chapter 1. McGraw-Hill Education, 2018.

Ying Yang. Research on the optimization of the supplier intelligent management system for cross-border e-commerce platforms based on machine learning. *Information Systems & e-Business Management*, 18(4):851 – 870, 2020. ISSN 16179846.

Chapter 4

Information Transfer Estimation on Large Data

4.1 Introduction

Transfer entropy (TE) is an information measure that quantifies information transfer between processes evolving in time (see Chapter 1.3.1). Transfer entropy has a plethora of potential applications in canonical systems, neuroscience, social media, and financial markets. Prior work with respect to financial applications have typically used long windows in their research. For example Marschinski and Kantz (2002) measured information transfer between two financial time series (the DAX stock index and the Dow Jones) to determine to what extent one index determined the behavior of another. The data used by Marschinski and Kantz (2002) sampled data every minute between May 2000 and June 2001; after cleaning the data 63,867 observations were used in Marschinski and Kantz (2002)'s study. More recently Sandoval (2014) used daily price data from 2003 to 2012 to detect causal relationships between 197 largest firms (globally) with Transfer Entropy.

Given the assumption in Chapter 1.3 that information is reflected in price and in an efficient market changes rapidly (if not instantaneously) then smaller frequencies of observations (or long time windows) such as monthly, daily, hourly, or by the minute may be insufficient

This chapter contains material from the following publication:

- K. M. Ikegwu, J. Trauger, J. McMullin, and R. J. Brunner. Pyif: A fast and light weight implementation to estimate bivariate transfer entropy for big data. In *2020 SoutheastCon*, pages 1–6, 2020. doi: 10.1109/SoutheastCon44009.2020.9249650

in capturing the price change process. A reason for this is not being able to capture the entire price change process between long windows, ergo shorter windows are needed. However shorter time windows will have finer resolution of data which requires more observations in a dataset.

In figure 4.1 consider that we are looking at a univariate dataset for 30 days. If the frequency of observations (time windows) in a 30 day period are daily then there's 30 observations in the dataset. If the time windows are hourly we can expect 720 observations in a 30 day period and if the time window between observations occurs on a minute level then we have 43,200 observations. If we continue these calculations down to the 1 second level then there are 2,592,000 observations in a 30 day window. We find that existing open source implementations are not suited to estimate information flow for datasets with large observations. Given the need to examine information transfers on finer resolutions of data we propose an open source software implementation to estimate TE on large datasets.

4.2 Estimating Transfer Entropy

Prior research has developed a few approaches for estimating TE. There's a requirement to specify some or all of the following parameter choices: the time between periods, length of the time series, number of past observations that inform the future observations and the direction of information transfer in the approaches. The correct approach depends on the underlying data. There are many techniques for estimating mutual information. Khan et al. (2007) explored the utility of different methods for mutual information estimation and many of the methods are applicable to estimate TE.

4.2.1 Kraskov Estimator

Kraskov et al. (2004) outlined a widely used method to estimate Transfer Entropy with k-nearest neighbors. Note that entropy can be estimated with:

$$\hat{H}(X) = -\frac{1}{n}\sum_{i=1}^{n} ln(\hat{p}(x_i)) \tag{4.1}$$

Kraskov et al. expanded this definition to estimate entropy to:

$$\hat{H}(X) = -\frac{1}{n}\sum_{i=1}^{n} \psi(n_x(i)) - \frac{1}{k} + \psi(n) + ln(c_{d_x}) + \frac{d_x}{n}\sum_{i=1}^{n} ln(\epsilon(i)) \tag{4.2}$$

where n are the number of data points, k is a parameter to select the amount of the nearest neighbors, d_x is the dimension of x, and c_{d_x} is the volume of the d_x-dimensional unit ball. For two random variables X and Y, let $\frac{\epsilon(i)}{2}$ be the distance between (x_i, y_i) and it's k^{th} neighbor be denoted by (kx_i, ky_i). Let $\frac{\epsilon_x(i)}{2}$ and $\frac{\epsilon_y(i)}{2}$ be defined as $||x_i - kx_i||$ and $||y_i - ky_i||$ respectively. $n_x(i)$ is the number of points x_j such that $||x_i - x_j|| \leq \epsilon_x(i)/2$, $\psi(x)$ is the digamma function where

$$\psi(x) = \Gamma(x)^{-1}\frac{d\Gamma(x)}{dx} \tag{4.3}$$

and $\Gamma(x)$ is the ordinary gamma function. Lastly $\psi(1) = -C$ where $C = 0.5772156649$ and is the Euler-Mascheroni constant.

Joint entropy between X and Y can then be estimated as:

$$\hat{H}(X,Y) = -\psi(k) - \frac{1}{k} + \psi(n) + ln(c_{d_x}c_{d_y}) + \frac{d_x + d_y}{n}\sum_{i}^{n} ln(\epsilon(i)) \tag{4.4}$$

where d_y is the dimension of y, and c_{d_y} is the column of the d_y-dimensional unit ball. Using

$\hat{H}(X), \hat{H}(Y)$, and $H(\hat{X}, Y)$ mutual information can be estimated as:

$$\hat{I}(X,Y) = \psi(k) - \frac{1}{k} - \frac{1}{n}\sum_{i=1}^{n}[\psi(n_x(i)) + \psi(n_y(i))] + \psi(n) \tag{4.5}$$

where $n_y(i)$ is the number of points y_j such that $||y_i - y_j|| \leq \frac{\epsilon_y(i)}{2}$.

The basic idea of mutual information estimation with the Kraskov's estimator is for each observation i to count the number $n_x(i)$ (or $n_y(i)$) of points less than the distance between x_i (or y_i) and the k^{th} neighbor. The distance or $\epsilon(i)$ fluctuates for each observation consequently $n_x(i)$ and $n_y(i)$ fluctuate. The counts are averaged over all observations which results in mutual information as defined in eq. 4.5.

To estimate TE with Kraskov's estimator recall from eq. 1.15 that TE can be expressed as the difference between two mutual information computations. To estimate TE with this method the first mutual information is between Y's future value, Y's lagged values, and the X's lagged values. The second mutual information is the mutual information between X and it's lagged values . The difference between the first and second mutual informations will compute TE. With an embedding of 1 TE can be estimated with:

$$\begin{aligned} TE_{Y\to X} = &\left[\psi(k) - \frac{1}{k} - \frac{1}{n}\sum_{i=1}^{n}[\psi(n_{xy}^{(t-1)}(i)) + \psi(n_{xy}^{(t)}(i))] + \psi(n)\right] - \\ &\left[\psi(k) - \frac{1}{k} - \frac{1}{n}\sum_{i=1}^{n}[\psi(n_{x}^{(t)}(i)) + \psi(n_{x}^{(t+1)}(i))] + \psi(n)\right] \end{aligned} \tag{4.6}$$

Here n_{xy} is the number of points y_j such that $||y_i - y_j|| \leq \epsilon_y(i)/2$ and $||x_i - y_j|| \leq \epsilon_y(i)/2$.

4.2.2 Additional Estimators

Khan et al. (2007) also explored the utility of Kerenal Density Estimation, Edgeworth approximation of differential entropy to calculate Mutual Information and adaptive partitioning

of the XY plane to estimate the joint probability density, which can be used to estimate mutual information. Ultimately Khan et al. (2007) found that a KDE estimator and Kraskov estimator outperform other methods with respect to their ability to capture the dependence structure of random processes. Anderson and McMullin (2018) examined the properties of the Kraskov estimator between equities and their underlying options. They ultimately provided evidence that incorrect parameter choices lead to smaller TE estimates and that Kraskov's method has a downward bias indicating that it underestimates information transfer. Nevertheless, it is relatively insensitive to the parameter selection.

4.3 PyIF

PyIF [1] is our proposed software implementation to estimate bivariate TE on large data and is open sourced. PyIF currently only supports using the Kraskov estimator to estimate TE. PyIF utilizes recent advancements in hardware to parallelize & optimize operations across CPUs and Cuda compatible GPUs (see NVIDIA et al. (2020)). Given the iterative nature of eq. 4.6 it is possible to speed up the estimation by performing computations in parallel. In particular we focus our efforts on the parallelization & optimization across operations to obtain n_x & n_{xy} in eq. 4.6 faster.

PyIF is a python implementation which utilizes 5 well-known and actively supported python libraries: SciPy (see Virtanen et al. (2020)), NumPy (see Harris et al. (2020)), scikit-learn (see Pedregosa et al. (2011)), nose (see Pellerin (2021)), and numba (see Lam et al. (2015)). SciPy is an open source Python library used for a variety of STEM applications. NumPy is a part of SciPy's ecosystem and is an open source package that provides convenient ways to perform matrix manipulations and useful linear algebra capabilities. Scikit-learn is a popular open source library for machine learning and nose is another open source library that is useful for testing code to ensure that it will produce the correct outcome. Lastly,

[1] PyIF can freely be downloaded from: `https://github.com/lcdm-uiuc/PyIF`

numba is a python compiler that can compile Python code for execution on multicore CPUs and CUDA-capable GPUs.

PyIF's interface requires you to supply X and Y, two numpy arrays with Nx1 dimensions. Optional arguments can be passed in such as k which controls the number of neighbors used in KD-tree nearest neighbor searches, *embedding* which controls how many lagged periods are used to estimate transfer entropy and a boolean argument GPU can be used to specify if you want to use a CUDA compatible GPU for transfer entropy estimation. Lastly another boolean argument $safetyCheck$ can be used to check for duplicates rows in your dataset. This boolean argument is there to help prevent a more subtle error that can occur when multiple data points in a bivariate dataset have identical coordinates. For duplicated observations this can lead to several points that have an identical distance to a query point during k nearest neighbors search which violates assumptions of the Kraskov estimator. A solution that is used in practice and that we recommend is to add a small amount of noise to your dataset to avoid this error.

4.4 Comparative Analysis

We compare PyIF's ability to estimate Transfer Entropy against existing implementations with respect to computational performance. We present all of the data and code used to estimate TE for all implementations [2]. Each implementation in this comparative analysis estimates TE on four simulated bivariate datasets of different sizes. The estimated TE values are roughly the same for each implementation and we forgo comparing the actual values since this is random simulated data. We make the assumption that there is relatively little to no information transfer between the random processes. We run each of the implementations (excluding Transfer Entropy Toolbox) on nano, a cluster of eight SuperMicro servers with

[2]The data and code can freely be downloaded from:
`https://github.com/lcdm-uiuc/Publications/tree/master/2020_Ikegwu_Traguer_McMullin_Brunner`

Intel Haswell/Broadwell CPUs and NVIDIA Tesla P100/V100 GPUs hosted by the National Center of Super Computing Applications at the University of Illinois at Urbana-Champaign. We used one node which contains two E5-2620 v3 Intel Xeon CPU's and 2 NVIDIA P100 GPUs with 3584 cores. We refer to this as the first analysis.

We conduct the same analysis on different hardware to compare PyIF to Transfer Entropy toolbox because of MATLAB licensing issues with the National Center of Super Computing Applications. We use an Engineering Workstation with an Intel Xeon Processor E5-2680 v4 hosted by Engineering IT shared services at the University of Illinois at Urbana-Champaign. We use a single CPU core and up to 16GB of RAM to estimate TE with Transfer Entropy toolbox and PyIF. This workstation does not offer CUDA compatiable GPUs to use for either PyIF or Transfer Entropy Toolbox so we forgo comparing the GPU implementations. This workstation has a CPU time limit of 60 minutes meaning that if any process uses 100% of a CPU core for more than 60 minutes the process is terminated. We refer to this as the second analysis.

4.4.1 IDTxl

The first implementation that we compare PyIF to is the Information Dynamics Toolkit xl (IDTxl). IDTxl is an open source Python toolbox for network inference (see Wollstadt et al. (2019)). Currently IDTxl relies on NumPy, SciPy, CFFI (which is another open source library that provides a C interface for Python code) (see Rigo and Fijalkowski (2018)), H5py which is a Python package that is used to interface with HDF5 binary data format (see Collette (2013)), JPype (see Menard and Nell (2018)) which is a Python module that provides a Java interface for Python code, and Java jdk which is a developer kit to develop Java applications and applets. IDTxl has additional functionality besides estimating TE however we only use IDTxl's capability to estimate TE on a bi-variate dataset.

4.4.2 TransEnt

TransEnt is a R package that estimates transfer entropy (see Mount et al. (2015)). Currently TransEnt relies on Rcpp which acts as a interface to C++ from R. TransEnt also relies on a C++ library called Appromixate Nearest Neigbors (ANN) (see Arya and Mount (1998)) which performs exact and approximate nearest neighbor searches. Currently the package has been removed from CRAN, however this software can be used and installed from Mount et al. (2015)'s github repo [3].

4.4.3 RTransferEntropy

RTransferEntropy is a R package that estimates transfer entropy between two time series Simon et al. (2019). Currently the RTransferEntropy package relies on Rcpp, and the future package which supports performing computations in parallel to decrease the wall time. We include both the parallel implementation of RTransferEntropy and the default implementation for completeness in the results.

4.4.4 Transfer Entropy Toolbox

Transfer Entropy Toolbox is an open source MATLAB toolbox for transfer entropy estimation (see Lindner et al. (2011)). This code's dependencies include: the Statistics & Machine Learning toolbox which provides functions to analyze and model data; the FieldTrip toolbox which is used for EEG, iEEG, MEG, and NIRS analysis; the parallel computing toolbox that performs parallel computations of multicore CPUs and GPUs; the signal processing toolbox that provides functions to analyze, preprocess, and extract features from sampled signals; the TSTOOL toolbox which is a toolbox for nonlinear time series analysis. TSTOOL no longer exists and cannot be download from it's official homepage [4]. Nevertheless, the developers of Transfer Entropy toolbox include pre-compiled mex files of TSTOOL that will

[3] Mount et al. (2015)'s Github Repo: `https://github.com/Healthcast/TransEnt`

[4] `http://www.dpi.physik.uni-goettingen.de/tstool/`

work with this implementation. At the time of writing, Transfer Entropy toolbox has not been updated since the year 2017.

4.4.5 Data

We create four bivariate datasets for this comparative analysis. Each dataset contains two time series with randomly generated values between 0 and 1. The first dataset contains 1000 observations, the second dataset contains 10,000 observations, the third dataset contains 100,000 observations, and the fourth dataset contains 1,000,000 observations. We used the seed number 23 for the pseudo-random number generator for reproducibility. We will refer to the first dataset, second dataset, third dataset, and fourth dataset as the micro dataset, small dataset, medium dataset, and the large dataset respectively.

4.5 Results

We report the results for Analysis 1 in Tables 4.1, 4.2, 4.3, and 4.4. Each table contains the wall time to estimate TE using the variety of implementations on the different data sets described in the section 4.4. The higher the wall time the longer it took for the specific implementation to estimate TE. The number in the relative performance column indicates how many times slower (or faster) PyIF (CPU) is to a particular implementation. After estimating TE using all of the implementations outlined in the comparative analysis section we found that PyIF scales better on larger data.

Excluding the TransEnt implementation, the CPU implementation of PyIF (or PyIF (CPU)) takes less time to estimate TransferEntropy than all other implementations. The R package TransEnt has a better performance in terms of speed than PyIF (CPU) for the micro dataset and the small dataset. However PyIF (CPU) is able to estimate transfer entropy in less time than all other implementations for the medium dataset and large dataset. PyIF (GPU) outperforms PyIF (CPU) for the small, medium and large datasets. Figure

4.2 visualizes this explanation. We suspect that the optimizations performed by Numba contribute to PyIF having a larger wall time than TransEnt on the micro and small datasets.

The results for Analysis 2 are in Table 4.5. Although the Transfer Entropy Toolbox exceeds the CPU time limit for the large dataset, the results show that PyIF is able to scale better than Transfer Entropy Toolbox for the other three datasets. PyIF's wall times are less than Transfer Entropy toolbox's wall times excluding the Micro Dataset. Figure 4.3 visualizes this explanation.

4.6 Conclusion

An important issue is addressed regarding large datasets with respect to estimating bi-variate TE We introduce a fast solution to estimate TE with a small amount of dependencies. On large data our implementation PyIF is up to 1072 times faster utilizing GPUs and up to 181 times faster utilizing CPUs than existing implementations that estimate bi-variate TE. PyIF is also open sourced and publicly available on github for anyone to use. For future work we plan to improve the existing code base to increase the computational performance of PyIF even further. In addition to this we plan to implement additional estimators outlined in section 4.2 to estimate bi-variate TE. This boost in computational performance will enable researchers to estimate bi-variate TE much faster for a variety of research applications.

4.7 Figures and Tables

4.8 Figures

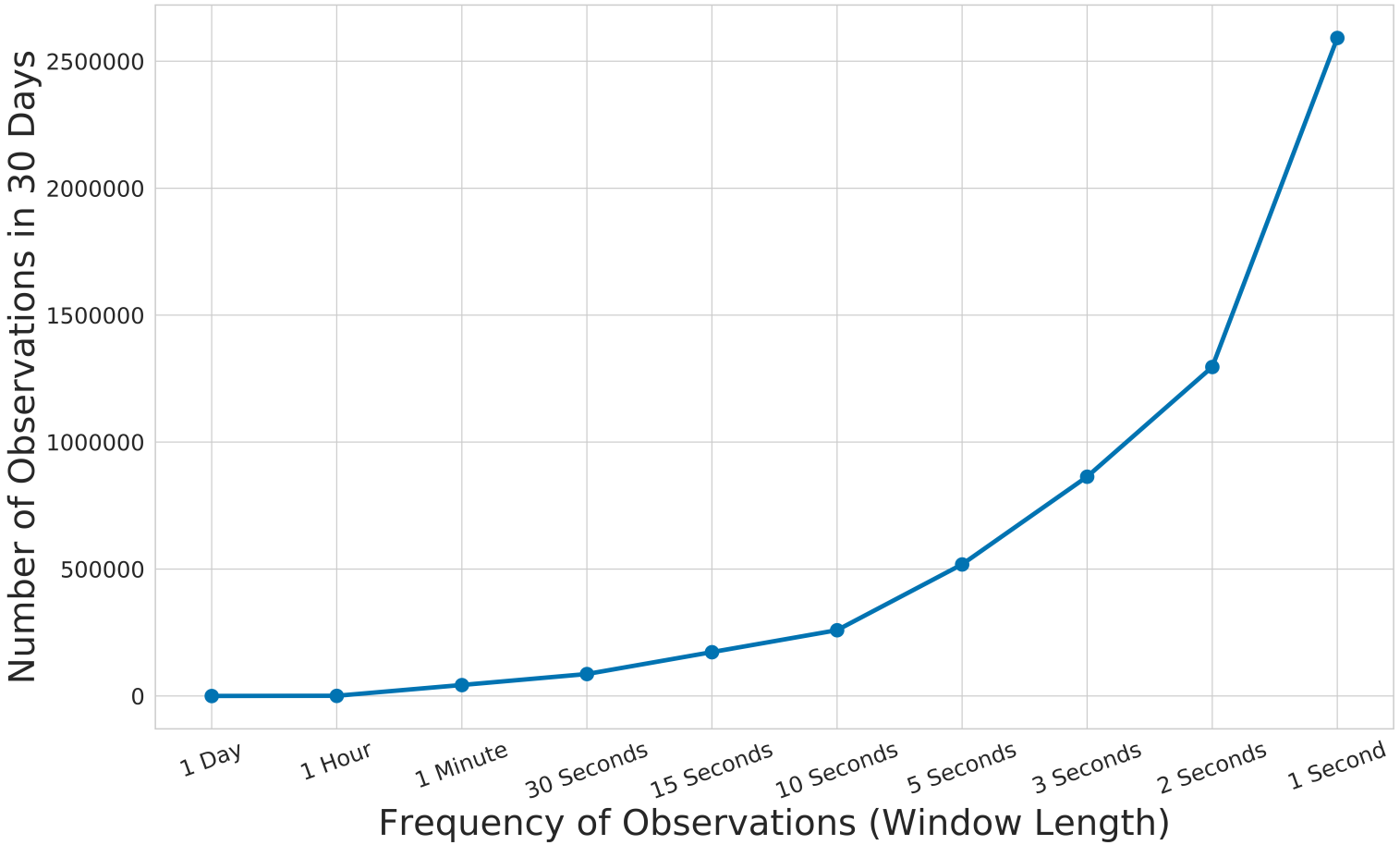

Figure 4.1: This figure shows the amount of observations for 30 days based on the frequency of observations.

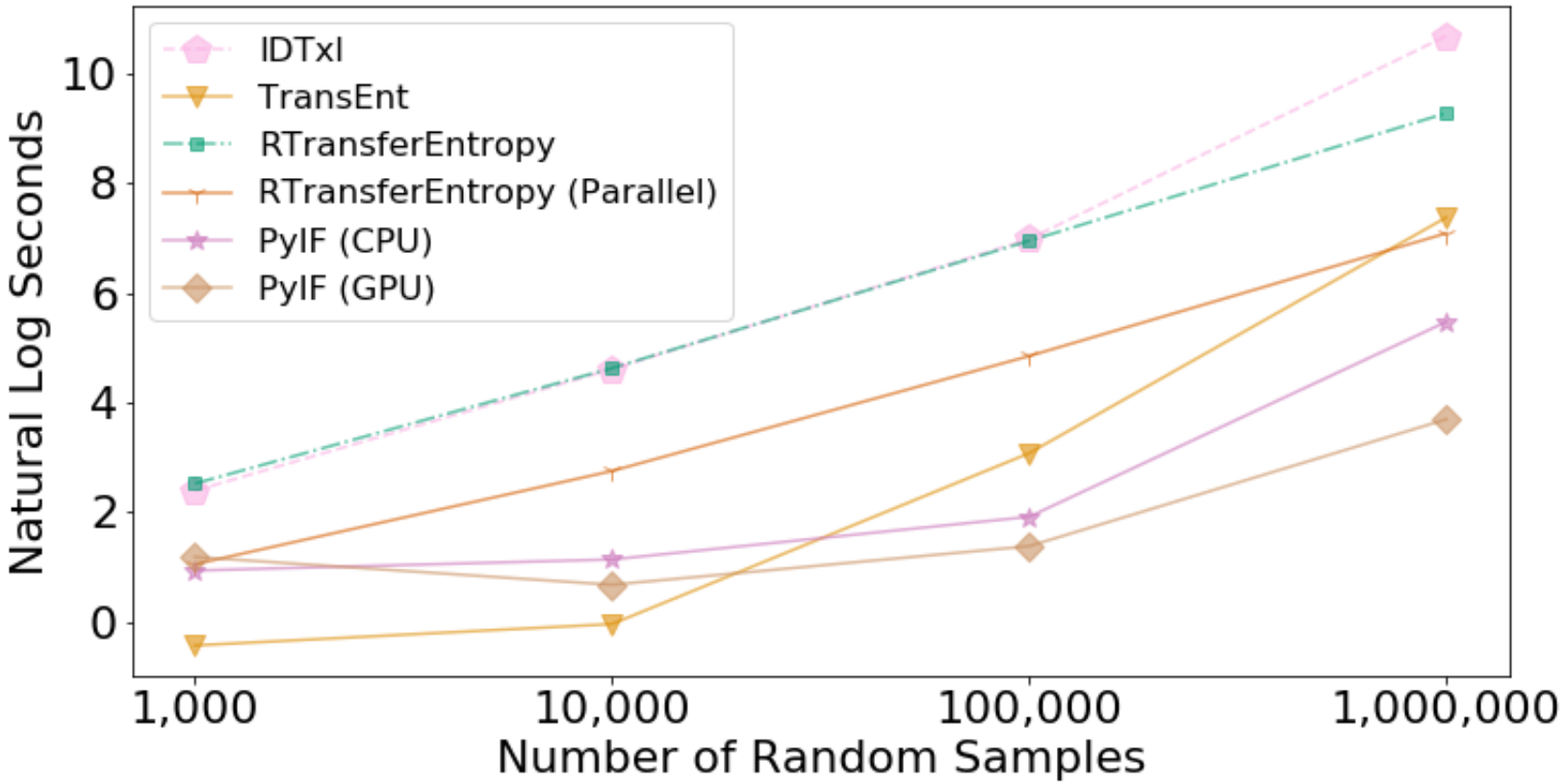

Figure 4.2: This figure shows the natural log time(in seconds) to estimate Transfer Entropy for each implementation (excluding Transfer Entropy Toolbox) for each dataset used in this study.

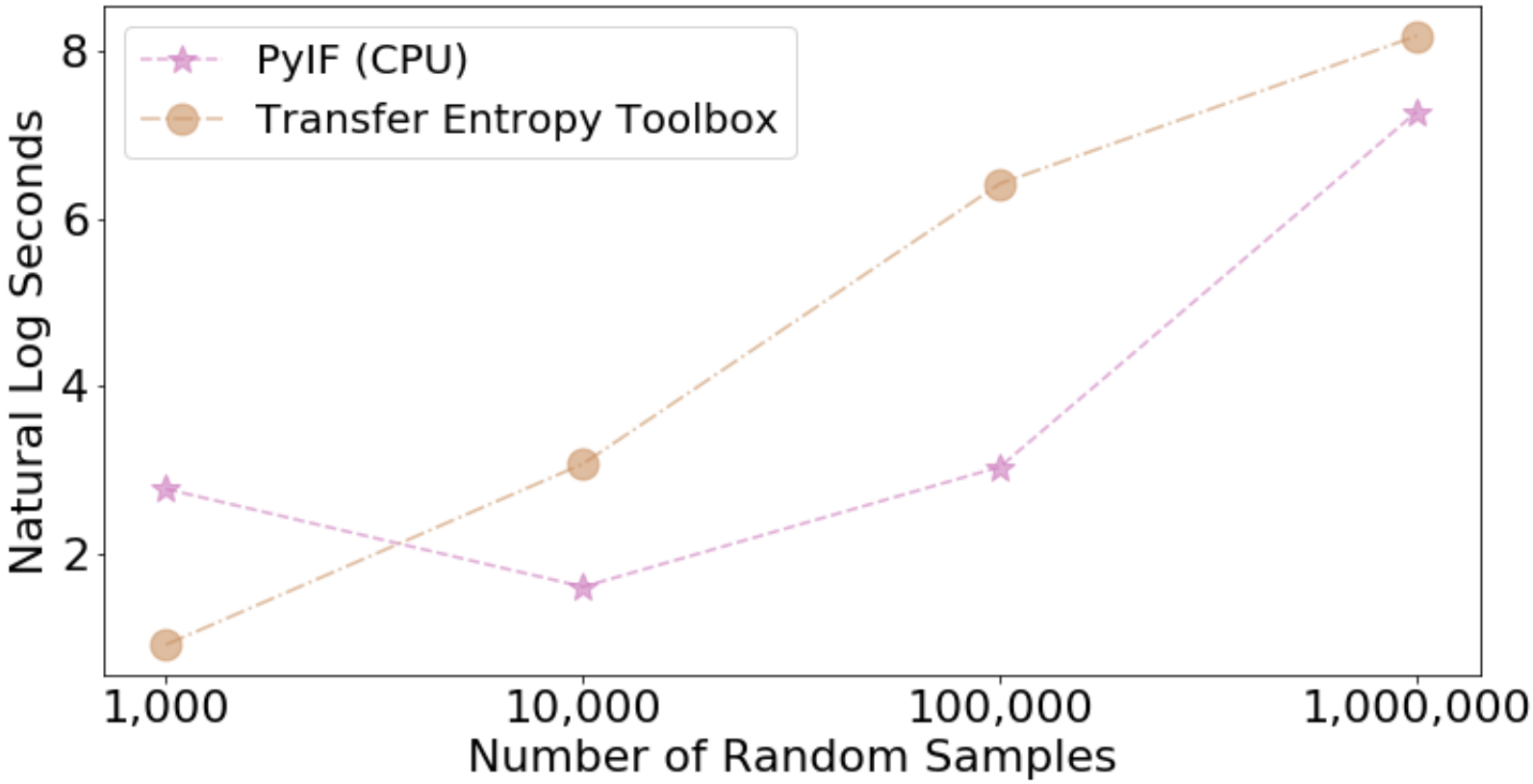

Figure 4.3: This figure shows the natural log time(in seconds) to estimate Transfer Entropy between PyIF and Transfer Entropy Toolbox on an Engineering Workstation as described in the section Comparative Analysis. Transfer Entropy Toolbox exceeded the maximum allowable CPU runtime for the Large Dataset.

4.9 Tables

Implementation	Wall Time (in seconds)	Relative Performance to PyIF (CPU)
IDTxl	10.98	4.28
TransEnt	0.656	0.25
RTransferEntropy	12.492	4.87
RTransferEntropy (Parallel)	2.876	1.12
PyIF (CPU)	2.564	1
PyIF (GPU)	3.282	1.28

Table 4.1: Micro Data results for the first analysis.

Implementation	Wall Time (in seconds)	Relative Performance to PyIF (CPU)
IDTxl	100.23	31.94
TransEnt	0.968	0.308
RTransferEntropy	102.228	32.57
RTransferEntropy (Parallel)	15.703	5
PyIF (CPU)	3.138	1
PyIF (GPU)	1.98	0.63

Table 4.2: Small Data results for the first analysis.

Implementation	**Wall Time (in seconds)**	**Relative Performance to PyIF (CPU)**
IDTxl	1070.749	152.89
TransEnt	21.708	3.03
RTransferEntropy	1036.661	152
RTransferEntropy (Parallel)	127.281	18.66
PyIF (CPU)	6.82	1
PyIF (GPU)	3.996	0.58

Table 4.3: Medium Data results for the first analysis.

Implementation	**Wall Time (in seconds)**	**Relative Performance to PyIF (CPU)**
IDTxl	43150.129	181.97
TransEnt	1585.942	6.68
RTransferEntropy	10592.77	44.67
RTransferEntropy (Parallel)	1188.636	5.01
PyIF (CPU)	237.122	1
PyIF (GPU)	40.231	0.16

Table 4.4: Large Data results for the first analysis.

Implementation	Wall Time (in seconds)	Relative Performance to PyIF (CPU)
Micro Dataset Results (1000 Obs.)		
PyIF (CPU)	16.049	1.00
Transfer Entropy Toolbox	2.5012	0.15
Small Dataset Results (10,000 Obs.)		
PyIF (CPU)	4.989	1.00
Transfer Entropy Toolbox	21.6880	4.347
Medium Dataset Results (100,000 Obs.)		
PyIF (CPU)	20.915	1.00
Transfer Entropy Toolbox	616.8712	29.49
Large Dataset Results (1,00,000 Obs.)		
PyIF (CPU)	1455.725	1.00
Transfer Entropy Toolbox	> 3600	> 2.47

Table 4.5: Results for the second analysis.

4.10 References

Matthew Anderson and Jeff McMullin. Detecting information flows in markets, Nov 2018.

S. Arya and D. Mount. Ann: library for approximate nearest neighbor searching. 1998.

Andrew Collette. *Python and HDF5*. O'Reilly, 2013.

Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020. doi: 10.1038/s41586-020-2649-2.

K. M. Ikegwu, J. Trauger, J. McMullin, and R. J. Brunner. Pyif: A fast and light weight implementation to estimate bivariate transfer entropy for big data. In *2020 SoutheastCon*, pages 1–6, 2020. doi: 10.1109/SoutheastCon44009.2020.9249650.

Shiraj Khan, Sharba Bandyopadhyay, Auroop Ganguly, Sunil Saigal, David Erickson, Vladimir Protopopescu, and George Ostrouchov. Relative performance of mutual information estimation methods for quantifying the dependence among short and noisy data. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 76:026209, 09 2007. doi: 10.1103/PhysRevE.76.026209.

Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Phys. Rev. E*, 69:066138, Jun 2004. doi: 10.1103/PhysRevE.69.066138.

Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6, 2015.

Michael Lindner, Raul Vicente, Viola Priesemann, and Michael Wibral. Trentool: A matlab open source toolbox to analyse information flow in time series data with transfer entropy. *BMC Neuroscience*, 12(1), 01 2011. doi: 10.1186/1471-2202-12-119. URL `http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3287134&tool=pmcentrez&rendertype=abstract`. Citations 56.

R. Marschinski and H. Kantz. Analysing the information flow between financial time series. *The European Physical Journal B - Condensed Matter and Complex Systems*, 30:275–281, 2002.

Steve Menard and Luis Nell. "jpype documentation". 2018. URL https://jpype.readthedocs.io/en/latest/.

ANN Library: David Mount, Sunil Arya. Transfer Entropy Packge: Ghazaleh Haratinezhad Torbati, and Glenn Lawyer. *TransferEntropy: The Transfer Entropy Package*, 2015. URL https://CRAN.R-project.org/package=TransferEntropy. R package version 1.5.

NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. Cuda, release: 10.2.89, 2020. URL https://developer.nvidia.com/cuda-toolkit.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Jason Pellerin. "nose. 2021. URL https://nose.readthedocs.io/en/latest/.

Armin Rigo and Maciej Fijalkowski. "cffi documentation". 2018. URL https://cffi.readthedocs.io/en/latest/.

L. Sandoval. Structure of a global network of financial companies based on transfer entropy. *Entropy*, 16:4443–4482, 2014.

Behrendt Simon, Dimpfl Thomas, Peter Franziska J., and Zimmermann David J. Rtransferentropy — quantifying information flow between different time series using effective transfer entropy. *SoftwareX*, 10(100265):1–9, 2019. URL https://doi.org/10.1016/j.softx.2019.100265.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Patricia Wollstadt, Joseph T. Lizier, Raul Vicente, Conor Finn, Mario Martinez-Zarzuela, Pedro Mediano, Leonardo Novelli, and Michael Wibral. Idtxl: The information dynamics toolkit xl: a python package for the efficient analysis of multivariate information dynamics in networks. *Journal of Open Source Software*, 4(34):1081, 2019. doi: 10.21105/joss.01081. URL https://doi.org/10.21105/joss.01081.

Chapter 5

Introduction

5.1 foo

foo

5.2 References

Appendix A

Standard Machine Learning Language Supplemental Code

A.1 Iris Python Code

This shows the code required to replicate the same actions of the SML *Query* in Figure **??**. It's important to note that detailed documentation is publicly available in [13], the purpose of this figure is to highlight the level of complexity relative to a SML query.

```
1  import pandas as pd
2  import numpy as np
3
4  from sklearn.preprocessing import label_binarize
5  import sklearn.cross_validation as cv
6  from sklearn.multiclass import OneVsRestClassifier
7  from sklearn.svm import SVC
8  from sklearn.metrics import roc_curve, auc
9
10 import matplotlib.pyplot as plt
11 import seaborn as sns
12 names = ['sepal length(cm)', 'sepal width(cm)', 'petal length(cm)', 'petal
       width(cm)', 'species']
13 data = pd.read_csv('../data/iris.csv', names=names)
14
15 iris_classes = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
16 features = np.c_[data.drop('species',1).values]
17 labels = label_binarize(data['species'], classes=iris_classes)
18 n_classes = labels.shape[1]
```

```
19
20 x_train, x_test, y_train, y_test = cv.train_test_split(features, labels,
       test_size=0.25)
21 svm = OneVsRestClassifier(SVC(kernel='linear', probability=True))
22 l = svm.fit(x_train, y_train)
23 predict_score = model.decision_function(x_test)
24 test_set_results = model.score(x_test, y_test) * 100
25 print ('SVM Prediction Accuracy = {0:6.2f}%'.format(test_set_results) )
26  fpr = dict()
27 tpr = dict()
28 roc_auc = dict()
29
30 for i in range(n_classes):
31 fpr[i], tpr[i], _ = roc_curve(y_test[:, i], predict_score[:, i])
32 roc_auc[i] = auc(fpr[i], tpr[i])
33 plt.rcParams['figure.figsize']=(12,12)
34 # Class Info
35 columns = [0,1,2,3]
36 cmap_class = ['Purples_r', 'Greens_r', 'Oranges_r', 'Greys_r' ]
37 color_class1D = ['purple', 'darkgreen', 'orange', 'grey']
38 column_headers =  data.columns.values.tolist() # Grab headers from df
39 column_headers = [column_headers[x] for x in columns] # Map headers to indices
       selected
40
41 label = 'species'
42 fig, ax = plt.subplots(len(columns), len(columns))
43 for ic, cc, cc1D in zip(iris_classes, cmap_class, color_class1D):
44   iris_class_data = data.loc[data.species == ic] # sep class
45
46   #Generate kde plot matrix for class
47   for col1, i in enumerate(columns):
48       for col2, j in enumerate(columns):
```

```
49              if i == j:
50                  sns.kdeplot(iris_class_data[iris_class_data.columns[col1]], ax=
                        ax[col1][col2], color=cc1D, shade=True, legend=False)
51              else:
52                  sns.kdeplot( iris_class_data[iris_class_data.columns[col1]],
                        iris_class_data[iris_class_data.columns[col2]], ax=ax[col1][
                        col2], cmap=cc)
53              # Formatting
54              if j == 0:
55                  ax[i,j].set_xticklabels([])
56                  ax[i,j].set_ylabel(column_headers[i])
57                  ax[i,j].set_xlabel('')
58                  if i == len(columns)-1:
59                      ax[i,j].set_xlabel(column_headers[j])
60              elif i == len(columns)-1:
61                  ax[i,j].tick_params(axis='y', which='major', bottom='off')
62                  ax[i,j].set_yticklabels([])
63                  ax[i,j].set_xlabel(column_headers[j])
64                  ax[i,j].set_ylabel('')
65              else:
66                  ax[i,j].set_xticklabels([])
67                  ax[i,j].set_xlabel('')
68
69                  ax[i,j].set_yticklabels([])
70                  ax[i,j].set_ylabel('')
71
72  plt.show()
73  plt.close()
```

A.2 Auto-MPG Python Code

This shows the code required to replicate the same actions of the SML *Query* in Figure **??**. It's important to note that detailed documentation is publicly available in [14], the purpose of this figure is to highlight the level of complexity relative to a SML query.

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.cross_validation import train_test_split
from sklearn.learning_curve import learning_curve, validation_curve

import matplotlib.pyplot as plt
import seaborn as sns

plt.rcParams['figure.figsize']=(12,12)
sns.set()

names = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', '
    acceleration', 'model_year', 'origin', 'car_name']

#load dataset
data = pd.read_csv('../data/auto-mpg.csv', sep = '\s+', header = None, names =
     names)
data_clean=data.applymap(lambda x: np.nan if x == '?' else x).dropna()
X = data_clean[['cylinders', 'displacement', 'horsepower', 'weight', '
    acceleration', 'model_year', "origin"]]
#Select target column
y = data_clean['mpg']
#Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,
```

```
    test_size=0.2)
25
26 # Define and train  linear regression model
27 estimator = linear_model.LinearRegression()# Generate Learning Cures
28 train_sizes, train_scores, test_scores = learning_curve(estimator, X_train,
    y_train)
29 # Train Linear Regression Model
30 estimator.fit(X_train, y_train)# Generate Validation Curves
31 param_range = np.arange(0, 5)
32
33 v_train_scores, v_test_scores = validation_curve(estimator, X_test, y_test,
    param_name='normalize', param_range=param_range)
34
35 score = estimator.score(X_test, y_test)
36 print('Accuracy :', score)
37 g = sns.PairGrid(data_clean, palette='PuOr_r')
38 g = g.map_diag(sns.kdeplot, shade=True) # can't add color arg...
39
40 g = g.map_upper(sns.kdeplot, cmap='PuOr_r')
41 g = g.map_lower(sns.kdeplot, cmap='PuOr_r')
42
43 plt.show()
44 plt.close()
45
46 color_pal = ['purple', 'dark green', 'orange', 'grey'] # For 1-D KDE
47 cmap_pal = ['PuOr_r'] # For 2-D KDE
48 classes = [] # May not have a class for categories
49
50 column_headers =  data_clean.columns.values.tolist() # Grab headers from df
51 column_headers = [column_headers[x] for x in columns] # Map headers to indices
    selected
52
```

```
fig, ax = plt.subplots(len(columns), len(columns))
if not classes:
  for col1, i in enumerate(columns):
      for col2, j in enumerate(columns):
          if i == j:
              sns.kdeplot(data_clean[data_clean.columns[col1]], ax=ax[col1][
                  col2], color=color_pal[0], shade=True, legend=False)
          else:
              sns.kdeplot( data_clean[data_clean.columns[col1]], data_clean[
                  data_clean.columns[col2]], ax=ax[col1][col2], cmap=cmap_pal
                  [0])

           # Formatting
           if j == 0:
               ax[i,j].set_xticklabels([])
               ax[i,j].set_ylabel(column_headers[i])
               ax[i,j].set_xlabel('')
                if i == len(columns)-1:
                   ax[i,j].set_xlabel(column_headers[j])
            elif i == len(columns)-1:
               ax[i,j].tick_params(axis='y', which='major', bottom='off')
               ax[i,j].set_yticklabels([])
               ax[i,j].set_xlabel(column_headers[j])
               ax[i,j].set_ylabel('')
            else:
               ax[i,j].set_xticklabels([])
               ax[i,j].set_xlabel('')
               ax[i,j].set_yticklabels([])
               ax[i,j].set_ylabel('')
plt.show()
plt.close()

```

```
82 plt.figure()
83 plt.xlabel("Validation examples")
84 plt.ylabel("Score")
85
86 v_train_scores_mean = np.mean(v_train_scores, axis=1)
87 v_train_scores_std = np.std(v_train_scores, axis=1)
88 v_test_scores_mean = np.mean(v_test_scores, axis=1)
89 v_test_scores_std = np.std(v_test_scores, axis=1)
90
91 plt.fill_between(param_range, v_train_scores_mean - v_train_scores_std,
       v_train_scores_mean + v_train_scores_std, alpha=0.1, color="orange")
92 plt.fill_between(param_range, v_test_scores_mean - v_test_scores_std,
       v_test_scores_mean + v_test_scores_std, alpha=0.1, color="purple")plt.plot
       (param_range, v_train_scores_mean, 'o-', color="orange", label="Training
       score")
93
94 plt.plot(param_range, v_test_scores_mean, 'o-', color="purple", label="Cross-
       validation score")
95
96 plt.legend(loc="best")
97 plt.show()
98 plt.close()
```