SEMANTIC NEWS ANALYSIS & PREDICTION

THESIS

Presented to the Graduate Council of
Texas State University-San Marcos
in Partial Fulfillment
of the Requirements

for the Degree

Master of SCIENCE

by

Seth R. Orell, Jr., B.S.

San Marcos, Texas
August 2011

SEMANTIC NEWS ANALYSIS & PREDICTION

Committee Members Approved:

Anne Hee Hiong Ngu, Chair

Byron Gao

Rodion Podorozhny

Approved:

J. Michael Willoughby
Dean of the Graduate College

ACKNOWLEDGEMENTS

I would like to thank my wife, my parents, and my friends for supporting me throughout my research. A special thank-you goes to my thesis chair, Dr. Anne H. Ngu for her patience and guidance.

This manuscript was submitted on June 9th, 2011.

TABLE OF CONTENTS

LIST OF TABLES

ABSTRACT

SEMANTIC NEWS ANALYSIS & PREDICTION

by

SETH R. ORELL JR.

Texas State University–San Marcos

August 2011

SUPERVISOR PROFESSOR: ANNE HEE HIONG NGU

Active stock trading firms have a need for quick analysis of financial news items. News affects markets. Predicting how a news article may move a stock's price can give a trader an edge over competitors and this involves the automatic understanding of a news item's semantics. Years of research on semantic Web Services has yielded a variety of techniques to discern or provide meaning beyond the basic WSDL syntax. I believe that this research into Web Service semantics has relevance in other fields, specifically the content analysis of news as it applies to markets.

The purpose of the present study is to determine if specific academic models of Web-based semantic analysis can be utilized to provide market price predictions. The study's design allows for an objective measure of accuracy by comparing predictions against actual market changes. In the study, I explore the application of current "Top-Down"

Web service semantic analyzers to distill the various approaches into abstract concepts. I take a common approach of textual content matching and apply it with and without synonym-analysis (a form of spread activation) with promising results.

Using the securities in the Russell 1000 Index (chosen for market liquidity and activity), I collected corresponding news articles from Reuters for 8 months. For each article, I pulled one-minute snapshots of market data for the article's publishing date and corresponding security. I then divided the news items into two groups: an in-sample learning set and an out-of-sample input set. The in-sample set of news provided "predictions" for price movement and I could contrast this against what the input item actually did in the market.

Simple semantic analysis produced encouraging results with a rate of return (profit) better than random for shorter hold durations (one to five minutes). A synonym-based strategy showed a stronger return for longer hold periods (thirty to forty-five minutes). Both strategies performed better than a random matching approach, which lost money for every hold duration. These results show potential for similar and broader market analysis using established academic models of semantic Web analysis.

"I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers. A 'Semantic Web', which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The 'intelligent agents' people have touted for ages will finally materialize."

Sir Timothy Berners-Lee
W3C Director

"What's worth doing is worth doing for money."

Gordon Gekko
Wall Street, 20th Century Fox, 1987

CHAPTER 1 : A GENTLE INTRODUCTION

THE SEMANTIC WEB

Semantics imply meaning. The “Semantic Web” (as proposed (1)) provides meaning to the World Wide Web, but not for us. We can readily understand what we see on the Web; computers cannot. Web services, while incorporating a widely used standard with precise syntax, lack the ability to express the meaning of their content. At least, a clear standard for describing the meaning of Web services has not emerged. For the remainder of this paper, we will focus on Web service semantics as opposed to other Web content.

Web services provide a loosely-coupled interface to Service Oriented Architecture over current transport standards (e.g., HTTP). Businesses view Web services as a complementary field to their own core business model (2). The re-use of existing Web services appeals to businesses because they can take advantage of these existing services (their own and others) and compose them into new business services. While positively promoting code reuse, these compositions become difficult to construct by the Spartan Web APIs alone. Web developers need context and semantics to effectively provide service composition.

Automatic Web service composition solves a major problem in the proposed “Semantic Web” - computers understanding and communicating with each other without human intervention. Some say the ultimate goal (3) of Web services is the automation of this composition. For Web services to properly interact with each other as pieces of composite applications, the entities involved must agree upon the semantics that will govern their interaction. “This is part of a broader problem known as the semantic interoperability problem. Semantic interoperability enables Web services to interact with each other despite their semantic nuances.” (Papazoglou, p. 548)

Researchers have long studied various methods of solving this semantic interoperability problem and their approaches vary. However, these approaches typically fall within two general classes that I will call bottom-up and top-down[1]. Bottom-up semantic analysis begins with the Web service author providing design-time semantic content for specially-purposed services (4) to consume. So, as the Web service advertises its operations, inputs, and outputs, it simultaneously provides context (metadata) to express the meaning of each.

Let us suppose a Web service operation returned a make/model of a car. The SOAP (Simple Object Access Protocol – although it evolved beyond this acronym with W3C standard 1.2) version of this might look as follows (Figure 1.1):

```
<make>Chevrolet</make>
```

Figure 1.1. Example SOAP Markup.

A bottom-up approach would annotate this line with a description that describes the content. A rudimentary version of this markup might look something like Figure 1.2.

[1] These terms are well-used and part of the public domain when used to describe Web services – I do not claim authorship of either term.

```
<make rdf:about=http://dbpedia.org/resource/automobile/manufacturer>

      Chevrolet

</make>
```

Figure 1.2. Bottom-Up, RDF Addendum to SOAP Markup.

This relatively simple bit of markup increased our character count by 62 (over 200%) and this type of semantic description language can get quite verbose in a more detailed application. Clearly, bottom-up semantics come at a cost much greater than that to provide basic content, but other methods exist that require no extra markup to extrapolate meaning: top-down analysis.

With a little bit of domain knowledge about the Web service (e.g., a car dealership locater, social networking, gene sequencing) computers can successfully discover meaning in a manner similar to the way you or I do – by reading and associating. Top-down Web service composition takes the available Web service descriptors and extracts meaning from what they expose already to map related content.

For example, a computer can easily recognize the following method signature (Figure 1.3) and determine it receives a string and returns a decimal.

```
public decimal GetStockQuote( string symbol )
```

Figure 1.3. Sample Method Signature.

However, you or I can immediately discern that the method is providing a stock price for a given symbol. We associate the words to our understanding of the English language and arrive at a meaning without too much thought. We took the existing information in the method signature and, noticing that "stock," "quote," and "symbol" all refer to financial market terms, correctly determined that the return result (of type decimal) will be a "price." We picked up on the information provided by the original programmer and his/her intent on a future consumer; i.e., we utilized the author's "self-documenting" code (5). Computers, while not quite so intuitive, can achieve remarkable accuracy (6) given the right domain context.

Researchers have worked on this problem even before Berners-Lee's proclamation of "Web 3.0" (1) and numerous papers (6; 7; 8; 9; 10) exist on the top-down style of garnering semantics and composing services. My research aims to build on this published knowledge and apply the same analysis methods to other sources besides Web services, specifically RSS-supplied financial news.

During my studies in Web services engineering, I also worked as a software engineer in a proprietary trading firm. There, I encountered a need for interpreting financial news in a manner that would indicate whether it would affect a stock price. The company received news but could not automatically associate how it may affect a stock's market price. I noticed how this fit the pattern of the Web service "semantic interoperability problem" and began to jot down ideas that became the seeds for my subsequent research and this thesis.

STUDY QUESTION

CAN I APPLY ACADEMIC SEMANTIC WEB RESEARCH TOWARD BASIC STOCK MARKET PREDICTIONS?

While top-down semantic analysis can, in some cases (6), validate the meanings they detect, most rely on some sort of external (human) authentication. In this research, I explored how market movement can provide an objective[2] way to verify the semantic analysis. In effect, as a price changes, it "grades" the predictions given by the analyzer.

At a high level, my approach involves two types of input: financial/business news articles, and market data. News items came from RSS feeds in near real-time. I fetched historical market data after enough time had passed for it to be available[3]. I stored these data for later evaluation with different types of semantic analyzers. For the analysis, I took top-down approaches used by Web service researchers attempting to solve the semantic interoperability problem and applied them to the news articles. Ultimately, the short-term movement of the stock price of the company in the news article provided feedback to the semantics learned.

To keep the research focused, I limited the depth of analysis to "good" or "bad." That is, the result set for any semantic approach falls into either a good news (price will go up) category or a bad news (price will go down) category. This step closely resembles the established data mining task of "classification" (11) with only two types of grouping: good and bad.

By connecting two or more news articles, I attempt to form a meaningful link between them so that I can predict the behavior of the market. The ability to predict links is central to many data mining tasks (12) and extensive literature exists on the topic. The problem is still relevant, however, and can be summarized simply as "given two entities in a

[2] Market prices are only partially objective, as they are ultimately driven by the decisions of humans – even in the cases of humans programming the algorithms that trade in the markets.
[3] Real-time market data is expensive and not needed for my analysis. Once the data is just 15 minutes old, the cost drops considerably

network, should there be a link between them?" (13) The matching engine I use is built on top of this research and owes much of its efficiency to this related literature. (14; 15)

To simplify analysis, I limited the scope of market movement to a short time period after the release of the news article. Based on these simple categorizations, I "tested" my results by either buying on good news or short-selling on bad news (with imaginary money in an imaginary market). After the time period ended, I realized any profits or losses and recorded these as output.

As an example, a particular approach (2) involves simple parsing of the news content - stripping out proper nouns, removal of stop-words, etc. – and using the remaining words to rank the article. We know the time of the news story and can observe what happened to the stock price in the minutes/hours after the newsroom released the article. Based upon how much the price moved, we then apply weights – positive or negative - to these words. Over time, and over many news stories, the common words lose significance. This leaves me with a collection of essential words that, upon receipt of a current news article, can immediately provide an analysis of what the stock price is likely to do. The preceding example could be repeated with two or three word phrases as well.

Another approach, as researched by Syeda-Mahood et al. (7), takes advantage of the common language used (English) coupled with a simple thesaurus-like mapping to provide a degree of synonymy. For example, if one news story mentions "fraud" and another mentions "cheat," we may associate these words as similar meaning – "steal" and increase the weights of the original words as if they appeared more than once. I borrow this technique from bottom-up, ontological semantic approaches and record how it may add value to the overall correctness.

THESIS CONTRIBUTIONS

The main contributions of this thesis are as follows:

- I show that existing "Top-Down" semantic Web analysis techniques applied to financial news enables me to build a low-cost system for automatic prediction of stock market movement.
- I developed a workflow process (SNAP) that can be used to streamline the scoring of an unseen financial news item using the past movement of stock prices.
- I show that my matching/prediction engine holds slight promise for simple keyword matching but demonstrates great potential for synonym matching. I also report that randomly matching news items likely results in no potential profit.

ASSUMPTIONS

I approached this new research with a few assumptions. First, news articles can affect market prices. I will not assume that every article moves the market, or even that news has a long-term effect on a stock price (16), but we can readily observe news affecting short-term market pricing on any financial cable network broadcast. Secondly, the authors of the news articles write their stories without knowledge of any specific downstream parsing/analysis. Harder to verify, we must rely on statistics and sheer numbers to provide some insulation between the article writers and this, admittedly small, demographic who wishes to automatically extract meaning from their words. Finally, I assume that news articles use similar words to express similar meaning. The target audience of breaking financial news expects a compact and quickly-digestible article. The journalist writing the article has little allowance for word-play or sesquipedalianism[4] .

RELATED WORK

AUTOMATIC GENERATION OF BLAST SERVICE TYPES

Top-down semantic analyzers can thrive with deep domain-specific knowledge. A 2005 paper from Ngu, Rocco, Critchlow, & Butler titled “Automatic Discovery and Interaction with Bioinformatics Web” looks into how a computer, given a special domain, could continually and automatically classify “BLAST Web services by pattern identification.” (6)

BLAST stands for Basic Local Alignment Search Tool and contains an algorithm for comparing primary biological sequence information. It allows a researcher to compare a query sequence with a library and identify sequences that resemble the original beyond a certain threshold. This research involved crawling the Web looking for BLAST services. Once discovered, they utilized pattern matching (via regular expressions) to identify services of a particular type and then unify (compose) these sources behind a single interface that updates and maintains its sources without manual intervention.

Within the scope of BLAST biological sequencing, their “data-type learner” engine showed 100% precision in recognizing alignment sequence in an unseen document. This type of accuracy becomes possible because of the deep knowledge of a focused domain.

[4] Sesquipedalianism (adj) – given to using long words; (of a word) containing many syllables

SEMANTIC MATCHING IN WEB SERVICE COMPOSITION

Semantic analysis can also provide the context necessary for further domain-specific analysis. In 2008, Aviv Segev published research (2) that used a Web service's WSDL syntax to generate context that could be used to match similar services. This approach involved both text extraction and tokenization. The process then fed these tokenized words into a Web-based search engine and then extracted context from the results. While admittedly slow, this process shows promise for composing Web services without previous domain knowledge.

Another, similar 2005 study (7) from Syeda-Mahmood, Shah, Akkirunu, et al. follows a similar model. They sought to match Web services by taking WSDL syntax and extracting the names of operations, inputs, and outputs. They then tokenized/parsed these names and used a simple English thesaurus to group like-minded operations and services without knowing what it was the services did. For example, during their processing, words like "ID" and "Id" both became "Identifier" via a dictionary. Words like "stock" and "inventory" intersect once pumped into a thesaurus. All of this is done without regard to the domain of the service.

Woogle, an "intelligent [W]eb service search engine," (10) offers another approach to garnering semantics from Web services. It also focuses on WSDL syntax, but applies a concept its researchers call "clustering" to find meaningful concepts from parsed words. The research around Woogle suggests that "parameters tend to express the same concept if they occur together often." Their approach involves each individual term starting as its own cluster. The algorithm then proceeds in a greedy fashion, creating associations that exceed certain thresholds. While this procedure can be quite complex, Web service descriptions are typically sparse which keeps the input size relatively small.

BLOGSCOPE'S ONLINE BLOGOSPHERE ANALYSIS

Professionally published news articles aren't the only source of new content injected into the Web. The ability to publish commentary as a blog has allowed anyone with a computer to report on current events. The term "blogosphere" describes the universe of blogs as a connected community.

In 2006, researchers at the University of Toronto's computer science department began systematically collecting and analyzing these blogs to find patterns and trends (17). Similar to my approach, BlogScope analyzes the actual textual content of the blog posts and not the tags. In this manner, they try to form their own semantic associations instead of relying on the bottom-up analysis of blog tags.

PROPRIETARY NEWS-TO-PRICE RESEARCH

Active trading companies have a long-standing need to make quick decisions – they seek to capitalize in short-term (hours, minutes, or even seconds) movements of stock prices. Many 3rd-party solutions exist to provide predictions of market prices, and many more companies actively develop their own solutions. However, the strategies employed by these companies are tightly guarded. Even some of the current published research on this topic (e.g. AZFinText (18; 19) and NewsCATS (20)) only reveals high-level workflows. The devil remains in the details.

However, some academic researchers have published their algorithms and strategies for solving this problem. Recent work from S. Wang et al. attempts to find relationships between news events and stock market price changes using an ontology-based data mining approach (21). By classifying news into categories and then applying expert-rules reasoning, these researchers offer another method of predicting stock market impact.

My approach seeks not to compete with the prevailing commercial products available to trading firms, but to explore whether promising approaches in Web service semantics have value to the trading industry. Further – and perhaps more interesting – by showing a forward link from Web service semantic analyzers to the financial industry, I hope to demonstrate that we are trying to solve a common problem and that a common solution may exist.

CHAPTER 2 METHODOLOGY

REQUIREMENTS AND SPECIFICATIONS

MODELING AN APPROACH

Over time, enterprising people have tried numerous techniques for predicting the behavior of the stock market. From neural networks to astrology to throwing darts at the financial pages, all manner of approaches have yielded all manner of results. Regardless of one's favorite prediction method, the business of active trading (or "day trading") remains a popular business both in the U.S.A. and around the world.

The ultimate requirement for this work is to predict the market. With that lofty goal out in the open, let's refine our objective a bit to see if we can show any potential for predicting the market. Further, we wish to accomplish this using techniques of published research into Semantic Web Composition. As stated earlier, our top-down approach will utilize various string-matching tools and algorithms used with success in other studies.

To test my study question, I need two basic sets of things: data, and tools to analyze the data. The next sections in this chapter describe both the kinds of data required and the specific technical tools I used to perform my evaluation.

DATA TYPES

Data drive this study. While avoiding a deep explanation of market terms, I can categorize my data as follows:

- Exchanges - An exchange is an organized market where tradable financial instruments are bought and sold. Narrowing this list to standard, electronic-friendly, stock markets keeps the study focused on securities that are of interest to day traders.
- Companies - On a particular exchange, shares of ownership in corporations (often called “stocks”) are bought and sold at whatever prices the market will bear. The company’s name is often abbreviated into its stock “symbol.” For example, Microsoft Corporation uses the abbreviation “MSFT”.
- Tickers - When a stock changes hands in an exchange, the exchange transaction both records and publishes the details of the sale including the volume (quantity), price, and date/time. These transactions make up the “tickers” often seen scrolling across a screen on a news broadcast or trading floor. These records give us a view of the stock’s price over time.
- News Items - These data are similar to a news article you may read in a local paper. Except, these items come from feeds that capture additional information like publication date/time, and the company affected.

TOOLS USED

To accumulate, navigate, and aggregate the data this study needs, I had to construct tools to fetch, organize, and analyze these data. With many years of professional development using Microsoft’s .NET framework, I chose it as the primary architecture for the project.

The start of my research coincided with the release of .NET 4.0 and the new ADO.NET Entity Framework (EF). I took this opportunity to learn EF while it modeled my data in code. EF sits on top of a MS-SQL database and abstracts the relational (logical) schema of the data as it resides in the database into a conceptual schema that I can access through code.

I chose to implement all coding in C# (v4.0), which has enough flexibility and power to efficiently handle all the designs I constructed. The C# language should look familiar to Java or C++ programmers and should allow anyone to follow the workflow and logic.

All development occurred with the Visual Studio 2010 (VS) IDE which provided an ideal environment to keep files organized. Its built-in compiler and debugger gave me visibility into how the code functioned and allowed me to focus on creating tools instead of managing builds.

EXTERNAL RESOURCES

Whenever possible, I tried to use existing tools and resources. Further, I looked for free and/or open-source versions if any were available. For instance, Google Finance provides a free Web service endpoint to get opening prices for given securities. I used this service daily to fill in gaps in my traditionally acquire[5]d market data. The remainder of the market data came from a professional Web site that specialized in historical (older than fifteen minutes) stock market indicators.

Reuters also provides feeds in a standard format called “Really Simple Syndication”[6] (RSS) which is specified in XML. This allowed me to programmatically extract the RSS feeds into News Items which I could store in my database. These free feeds began the chain of collecting data and gave direction to what market data we needed.

[5] I paid for it.

[6] Also less-commonly called “Rich Site Summary” and “RDF Site Summary”

Open-source projects, such as the HTML Agility Pack[7]and Apache Lucene[8] provided algorithms to handle complicated text parsing and powerful text association. Both of these libraries play integral parts inside the data acquisition and data analysis, respectively. Without tools like these, this project would have taken considerably more time and effort.

ALGORITHMS AND DATA STRUCTURES

SLEW OF DATA

As mentioned earlier, data drives this project; so much so that I ended up with volumes of it. Early on in my study, it became apparent that the free version of Microsoft's SQL Server would not suffice (data size exceeded limitations). Soon it became clear that my greatest research challenge would be how to handle, arrange, and aggregate data in a fashion that would complete in a reasonable amount of time[9].

This led me to develop the program in phases with my top priority to gather data. I ran the news and market data "fetcher" nightly for months to gather the many gigabytes of data I required for my upcoming analysis. All the while, I refined the code, learned more about how to best use Entity Framework, and simply get out of my own way to let the data write.

With additional resources, I could easily parallelize this process. But my equipment was limited to a single (albeit relatively beefy) computer. For future studies, I could readily move to a server farm or cloud setup to dramatically speed up throughput.

[7] http://htmlagilitypack.codeplex.com/
[8] http://incubator.apache.org/projects/lucene.net.html
[9] Early versions of the data fetching program took 8-12 hours to complete

SPEED BEATS SPACE

Using the proper data structures made some of my problems easier. With a 64-bit operating system and 8GB of RAM[10] I could afford to build in-memory structures that offered faster look-ups than a similar SQL query to disk. By building a Dictionary - a generic HashTable - I could spend O(n) time performing slow SQL reads, but only once. Then, I could use the Dictionary over and over with O(1) look-ups. These could be memory intensive, but selective building and then releasing of these data structures greatly sped up my data access and allowed complicated analysis to occur. I only had to be aware of my total system memory usage so I didn't exceed the physical capacity of the machine and start paging/swapping to disk as this greatly degraded performance.

Another key technology feature that allowed me to map data into memory is a relatively new feature[11] to .NET called Language Integrated Query, or LINQ. This component allows programmers to query any data structure similar to how one would query a database. This allowed me to continue to accurately query data in memory as succinctly as I would writing SQL set queries.

Once I could determine what set of data I needed to select, LINQ let me treat my database as a long-term storage solution and not as a query engine. My queries executed one layer upward, in the code. This gave me a tremendous performance boost to allow complex analysis to happen.

[10] The maximum amount my research laptop could handle
[11] Since .NET 3.5 (11-09-2007)

ENTER LUCENE

During my research, I began to encounter papers dealing with semantic matching using an open-source search engine library called “Lucene.” (22; 23) Having worked with the Apache Lucene project in the recent past, I knew it to be a powerful and fast indexing/search tool. What I discovered was that researchers were using Lucene to map semantic relationships, manage large ontology data repositories, and to quantitatively measure the success of concept mappings. All of these approaches differ somewhat from Lucene’s advertised functionality[12] and they gave me the idea that perhaps I could bend the library to meet my needs as well.

The Lucene indexer takes input text, parses and then indexes it. Its search library then uses a combination of algorithms to perform fast, accurate searches. Under the hood, Lucene uses a customizable mixture of a Boolean Model and a Vector Space Model to match and retrieve documents (24). Boolean and Vector Space are two common theoretical models of search.

In a pure Boolean model, we either find a match or we do not. With this search archetype, the engine does not score nor associate any relevance with matching documents. This model merely “identifies a subset of the overall corpus as matching the query.” (24)

Vector space modeling takes both our search queries and our indexed input and models them as vectors in a high dimensional space. Each specific indexed term becomes a dimension in this search pattern. The vector distance between a search query and nearby terms computes into a score to rank the likelihood of a match.

[12] Lucene is advertised as a “free, open-source information retrieval software library”

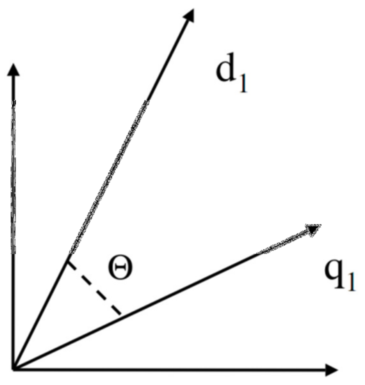


Figure 2.1. Vector Space Model.

Figure 2.1 (above) illustrates a typical vector space model where documents (d) and queries (q) are represented as vectors such that $d_j = (w_{1,j}, w_{2,j}, \dots, w_{t,j})$ and $q = (w_{1,q}, w_{2,q}, \dots, w_{t,q})$. The deviation of the angle θ between the vectors measures how close the query matches the document.

Lucene also considers the rarity of the term matched when calculating scores. It determines the frequency of the indexed term within the global space of all terms. For example, if we matched a term that is not very common in the indexed set of data, this match would produce a higher weighted score than a more frequently encountered term. This algorithm is similar to the standard Term Frequency - Inverse Document Frequency (TF-IDF) used in many information retrieval systems. (25)

PUTTING IT ALL TOGETHER

The whole system can be abstracted into the architectural diagram below (Figure 2.2). Each section of work is driven by the data entering its boundary. In the following chapters, I cover each of the three sections in detail and explain how they support one another.

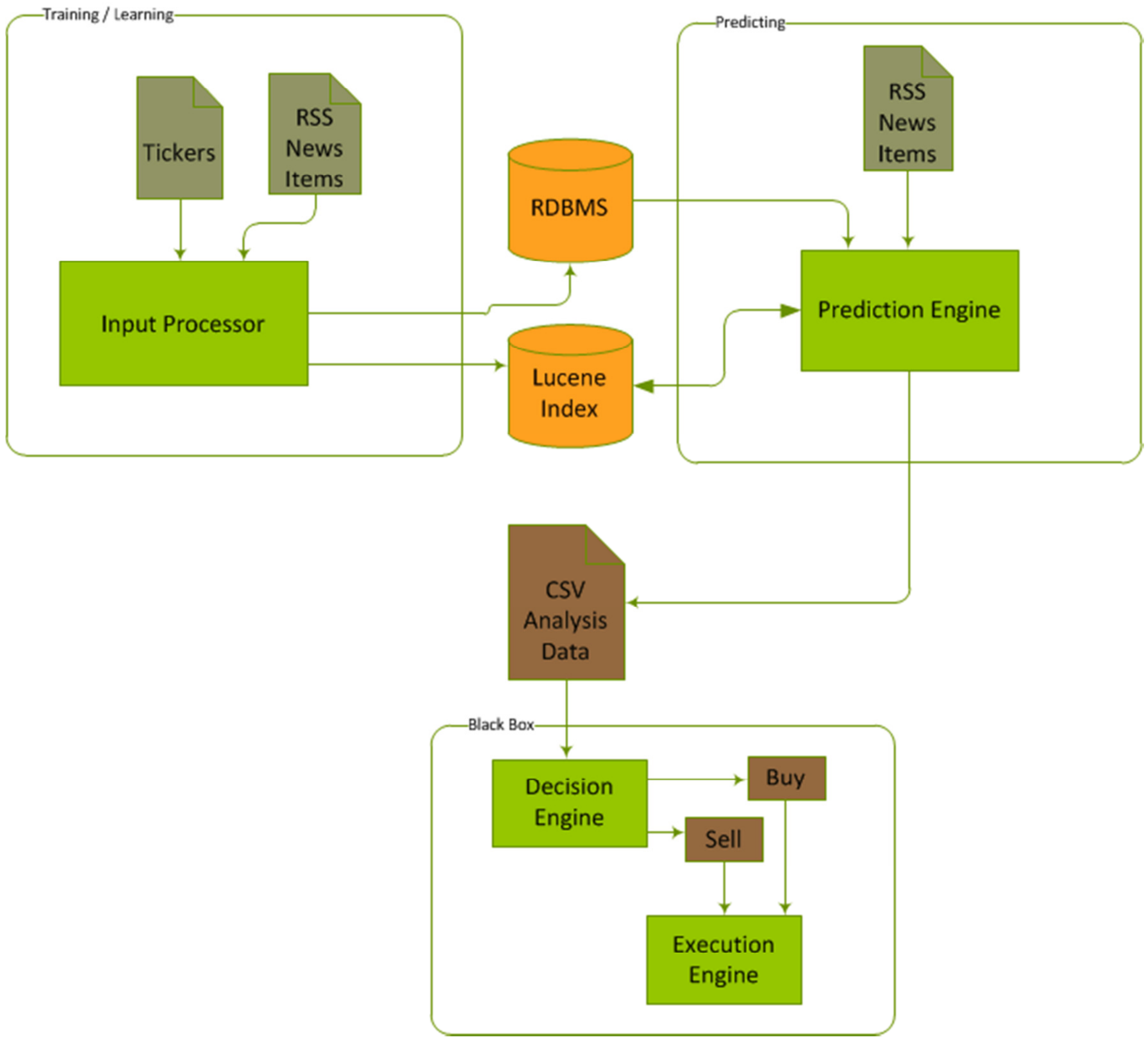


Figure 2.2. Architectural Diagram.

CHAPTER 3 CONCEPTUAL DEFINITIONS

A PROBLEM, REVISITED

The traders know that news affects stock prices. Dozens of flat-panel televisions tuned to a popular Financial News Network illuminate the otherwise dimly lit trading floor. When news hits, they react quickly to stay ahead of the curve.

I helped design and develop a software product that matched incoming news items to traders that held a position (either owning or owing shares of a company) by the particular company. I routed this news right to their desktop trading station so they didn't have to even look up to receive timely and relevant news.

However, the one thing I could not provide was the meaning of the news item: its semantic content. The trader had to scan the news to determine if it meant the position was worth holding (the price would go up) or they needed to get out of the position (the price would fall). What I wanted was a way to give the trader advice as to which direction the stock's price would likely go. Even something as simple as a red light (sell!) or green light (buy or hold) to accompany the article could provide an edge to the trader and profits for the company.

A natural extension of this workflow is to route these trading decisions (buy/sell/hold) to a "black box" that could execute the trades automatically. Of course, both the traders and the business would need evidence that such a process would work effectively before risking real money.

ON-THE-FLY MAPPING

Clearly, I have no way to pre-populate the semantic bits of the news item. If I wanted a “bottom-up” semantic approach, I would expect the news item to have markup like Figure 3.1:

```
<NewsItem rdf:about=thisIsGoodNews>

    Here is the body of a news item that must be good, according to our RDF

markup!

</NewsItem>
```

Figure 3.1. If Only We Had Markup This Good!

But, I do not receive this. My real news item has no RDF tags to tell me semantic meaning. I must derive the intent in real time. Hence, I now have a similar problem to the automatic Web service composition I mentioned earlier. I have two sets of services (news providers and traders), a clearly-defined domain, and test data to score results.

I want to use the first service (news) to influence the second (trading), and have to find common ground between the two. If I can boil down the data to its core pieces, I can form ontologies that will let me map meaning.

Fortunately, both the news and the trader relate to a company (or two, or three). As stated above, this lets me route news items to traders who have an interest in the company mentioned. However, a second relationship exists if we divide our news items in a crucial way - past and present.

Past news items come with a benefit - I readily know what the stock price did after the news arrived. I can quickly query an old news item to find its impact on the company's stock price. My ontology mapping now involves the "new" news item relating to the "old" one. Between these two relationships, I show how certain semantic Web mapping techniques hold promise for securities trading. In particular, I show how search libraries can facilitate this with speed and flexibility.

For example, before I start predicting, I "train" my engine by parsing and indexing months of previously captured news articles. Then, when I receive a new article ("Widgets-R-Us (WRU) proclaims lower than expected earnings this quarter..."), I similarly parse it and have the matching engine find me the best match from my pool of training articles ("Fizzbang's (FZB) low quarterly earnings disappoint investors...").

Once I have my match, I get the stock symbol for the matched article (FZB) and look up its historical price at the exact time of the FZB news event ($14.52) and at specified periods afterward ($14.27 after 5m, $14.21 after 10m, $14.34 after 20m, etc.) This suggests to me that the stock described in the new news article (WRU) will lose value over the next five to fifteen minutes. I give an instruction to short-sell[13] and then exit the position in ten minutes.

SEARCH SOLUTIONS

For semantic Web services to automatically interact and communicate meaningfully, they must know that the service to whom they connect contains meaningful content. Regard-

[13] A market transaction in which an investor sells borrowed securities in anticipation of a price decline and is required to return an equal number of shares at some point in the future.

less of the approach of the service's semantic discovery (bottom-up/top-down), as it seeks out appropriate endpoints from other services, it *searches*.

Search, with respect to Web service composition, can involve utilizing common, powerful programming tools like Regular Expressions. This kind of search involves a specialized formal language that allows for a flexible means to match strings or patterns of strings (26).

Other formal languages are commonly used in search, such as Structured Query Language (SQL) for relational database searches, XQuery for XML queries, and LINQ for .NET containers. As such, "search" pervades much of modern programming in one form or fashion.

This project uses keyword, synset, and spread-activation search strategies to associate news articles. These searches aim to match semantics and concepts between articles as a means to predict price.

SNAP WORKFLOW

I capture this entire process of news analysis, semantic mapping, and price prediction in a workflow I call SNAP, which stands for Semantic News Analysis & Prediction.

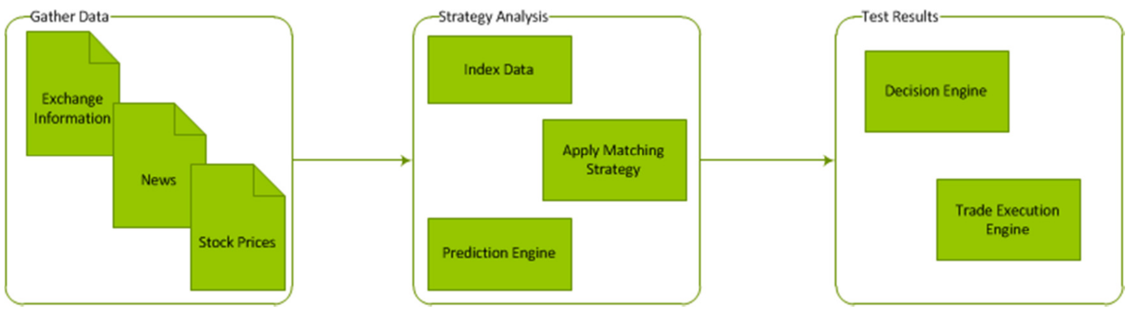


Figure 3.2. SNAP Workflow.

This workflow, as illustrated in Figure 3.2 (above), consists of three core areas: data gathering, strategy analysis, and testing results. Each section draws from the lessons learned from other semantic Web research. Over the next few chapters, I explain in detail the specifics of each phase of the workflow and how prior investigation guided my hand.

CHAPTER 4 GATHERING OF DATA

To fully vet out some of the basic tenets of this study, I required a large amount of data. The data would serve as both a learning set and a test set for all applied strategies. Therefore, I needed to define exactly what kinds of data I required and how I aimed to get them. Once identified, I needed to collect and store these data to allow for analysis and testing. This chapter covers how I determined which data to pull, how I physically acquired it, and how I stored and organized it to facilitate analysis.

IDENTIFYING DATA NEEDS

My data needed to represent what an active trader finds useful during short-term trading. This meant I wanted to follow companies whose stock traded frequently and consistently without the trade causing significant movement in price (i.e., they had market liquidity). With advice from experienced market technologists, I narrowed the pool of available companies down to the one thousand stocks in the Russell 1000 Index. These stocks represent the largest (by total market capitalization) one thousand companies in the U.S. equity market and account for approximately 90% of this market (http://www.russell.com/Indexes/data/fact_sheets/us/Russell_1000_Index.asp).

News articles provide the content that I want to semantically correlate with each other. They feed the analysis engine and drive each semantic strategy. To that end, once I refined the scope of companies I began polling for news data. For eight months, I nightly iterated over each of the one thousand securities and requested news related to that company from available Web services (www.google.com/finance/company_news). Each news item contained a link to the body which I subsequently downloaded, integrated into my news object, and stored to disk.

Even though I asked for news articles for a particular company, I discovered that the provider could associate the article with many companies. For example, I could receive the same news item for stock MSFT (Microsoft) as I did for AAPL (Apple) if the news item spoke of the entire technology sector. To keep from storing duplicates, I keyed the articles by a combination of source, title, and publication date.

When analyzing, I further restricted this pool to a single, global news agency: Reuters. Their strict policy toward advocating journalistic objectivity (http://handbook.reuters.com/index.php/Main_Page) helps negate any potential down-stream bias toward semantic analysis. In addition, by choosing one provider, and thus one formatting style, this allowed me to minimize parsing oddities and best extract the news from the extensive page markup.

Unlike all other data collected during the study, news items were time-sensitive. In other words, I could not request news for AAPL for a particular date. So, if I missed a day's news, I could not go back and retrieve it specifically. This required a nightly task to run the data acquisition program fired by the built-in Windows scheduler.

THE BIG PULL

My project includes the objective "scoring" of test results by comparing how a predicted effect would change a stock's price. I researched many services that provide delayed[14]

[14] at least a 15 minute delay

stock price data and found many (google.com/finance, yahoo.com/finance, etc.) that provided free quotes, but only per day/week/month. I needed the ability to get prices throughout a trading day. After a few consultations with some experienced trading technology specialists, I decided on an affordable[15] premium service that provided all the price information I would need.

This service (www.tradingphysics.com), allowed prices as granular as per second in the form of a “ticker” object. This ticker contained the average buy and sell prices for the time duration specified. In my study, I chose a five-second ticker interval. I knew I would initially measure results in minutes but may wish to zoom in somewhat, so five-second slices split the difference between flexibility and space/speed.

Once I finished the nightly pull of news items, I would examine the data to find each item’s date of publication and company affected. I then contacted my market data provid-er and pulled tickers for the given date for the given company. I then filtered down the results to include only those that occurred during New York Stock Exchange (NYSE) trading hours (0930 to 1600 Eastern). Even so, these tickers took up the vast majority of space relative to any other data type in the project (over 47 million rows).

THE DATA MODEL

Storing all the data necessary for the project required a data model to define relationships and the shape of data. As stated previously, this project used Microsoft’s ADO.NET Entity Framework (EF). This allowed me to abstract the relational schema of the data on the database and present a conceptual schema to my program.

[15] $20 per month

For example, the Exchange schema has a one-to-many relationship with the Company schema (one Exchange can have many Companies). Entity Framework presents this relationship to the programmer in its fully hydrated form as an aggregate root. That is, my Exchange object contains an enumerable property called “Companies” that represent this one-to-many relationship.

```
var myExchange = databaseContext.Exchange.Where( x => x.Name == “NYSE”);

var nyseCompanies = myExchange.Companies;
```

Figure 4.1. Entity Framework Syntax.

RELATIONS (ENTITIES)

The three main actors, or entities, are Companies, News Items, and Tickers. A secondary entity, NewsToCompanies, manages the many-to-many relationship between a News Item and a Company. Another entity for Exchange helped with some grouping, but was largely unused.

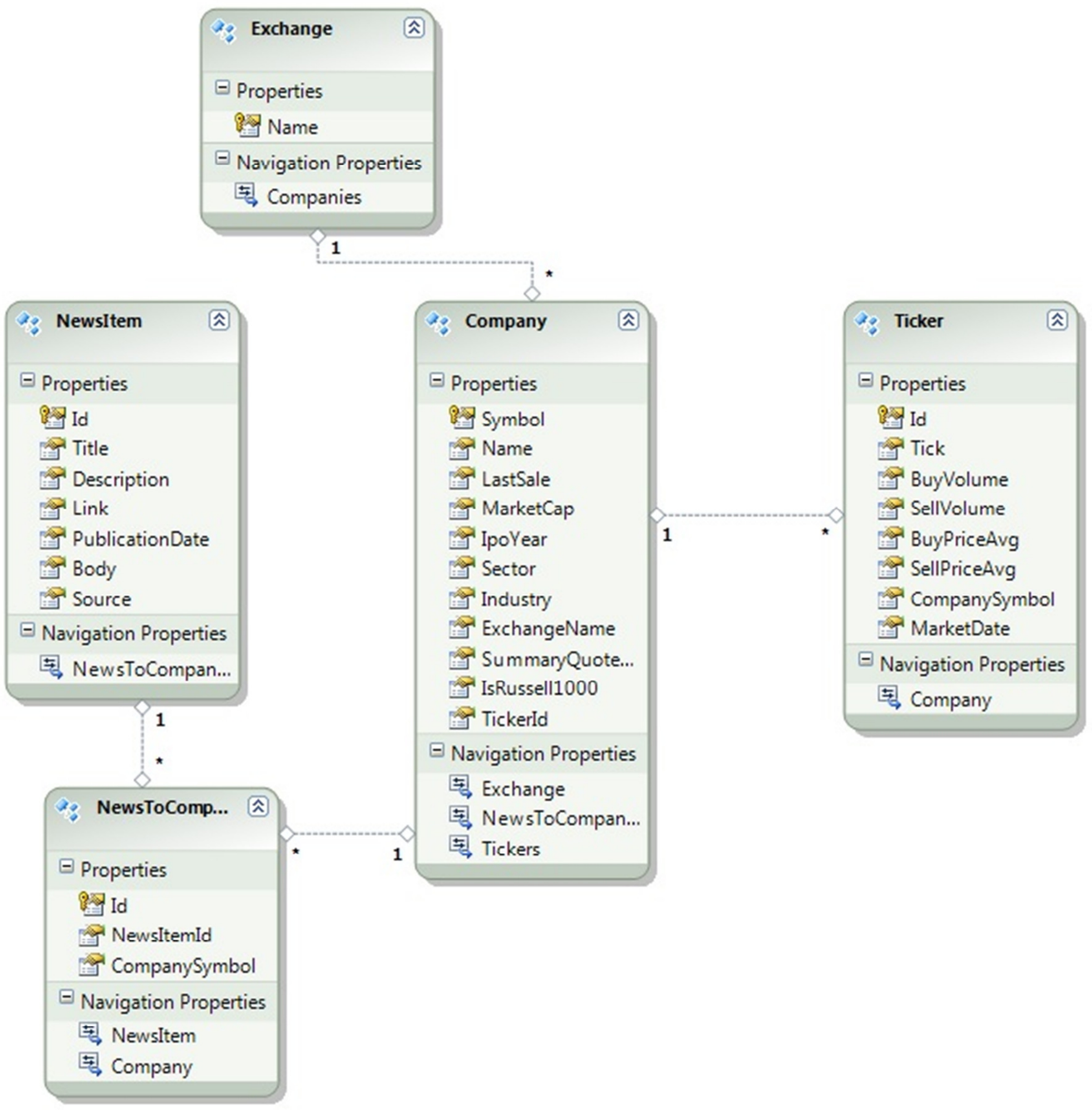


Figure 4.2. Entity Model View.

The diagram above (figure 4.2) closely resembles an Entity-Relationship (ER) model but it has some special distinctions. At the bottom of each entity is a list of “Navigation Properties.” These allow the program to traverse forward and backward through relation-ships without having to do any JOIN operations. For example, if I had a reference to a particular company, I could find all the news items associated with that company by the following code:

```
List<NewsItem> newsItems = company.NewsToCompanies.Select(x => x.NewsItem);
```

Figure 4.3. Select All News Items for A Company.

Under the covers, Entity Framework creates SQL that includes the proper JOIN statements to get the data I'm after.

CHAPTER 5 DATA ANALYSIS

THE QUANTITATIVE TRADING MODEL

My data collection process (SNAP) now allows me to put together and test the strategies gleaned from various top-down semantic Web analyses. Similar to other many algorithmic (or quantitative) traders, this study utilizes three logical components:

1. A data stream component
2. A strategy component
3. An execution component

These three components correspond to SNAP workflow sections Data Gathering, Matching/Prediction Engine, and Trade Execution Simulator, respectively. Together, these components provide a "proving ground" for any strategies I may wish to test. Strategies that fare well could be applied to a real-time matching engine. Those that fare poorly could be adjusted further or discarded. For now, I explore how the components work together to test our strategies.

My data become "streamed" as I consume them, sequentially, over time. Thus, my collection of news items becomes a feed in the order it originally arrived. The strategy component takes my semantic analysis and matches streamed news with historic news to predict likely outcomes. Finally, an execution component "realizes" my prediction against real market prices. My testing model achieves this by splitting my input into two sections - a training set and a test set.

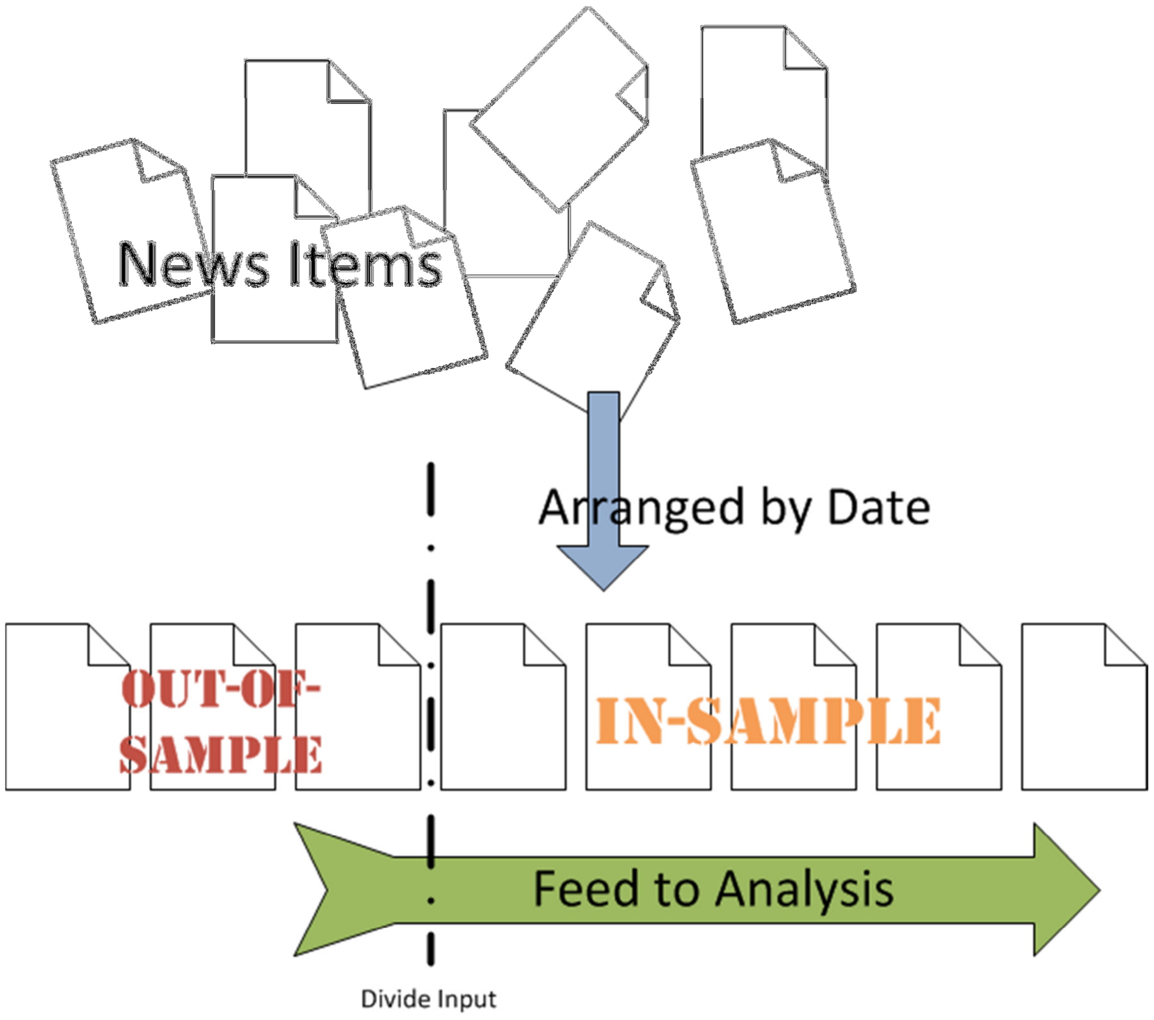


Figure 5.1. Data Flow.

The diagram above (Figure 5.1) illustrates how I take my news data and divide them into two distinct groups. Common to many trading models, I am setting up an "in-sample" testing environment where a standard body of data can train strategic models (Narang, R. K.). I can then test these predictions against another set of data, either live or recorded, to see if it yields any profits. This portion of the testing phase is called “out-of-sample” testing.

Unlike many quantitative models, this study differs from most black-box trading research in that my prediction strategies come not from analyzing the learning set of data, but from preparing it for matching. This preparation involves special parsing and indexing to allow for near real-time lookup. The in-sample data set becomes both a target for matching out-of-sample data and a means of outcome prediction. That is, based on the news match, I can forecast how the price should react.

This study applies three semantic “strategies” garnered from published academic research on Web service composition:

1. A simple, textual matching that uses vector-space modeling
2. A more complex version of the first that uses a basic “spread activation” on top of vector-space modeling
3. A completely random association of data to function as our test set

STRATEGY ONE: SIMPLE TEXT MATCHING

The root of all the my semantic strategies can be stated as follows: given a news item from “out-of-sample” stream, match it to the single, most semantically similar news story from an “in-sample” set. Then, look up how the in-sample match affected stock prices when it was originally published and use the price change as a predictor for the new news item. The only difference between my strategies will be how I match the news articles.

Simple, textual analysis can take the form of a Regular Expression search, such as the pattern matching of Ngu, Rocco, Critchlow, & Butler’s work in Bioinformatics (26).

Similarly, the keyword matching of Schumaker & Chen (18) and Rocha, Schwabe, & Poggi de Aragao (22) demonstrate that an effective semantic matching strategy does not require extensive and complicated algorithms as long as the user can work within a known domain. Since I know the confines of my domain (financial news), I can directly apply their techniques.

My simple matching process takes in a news article and runs the text through a preparatory analyzer to normalize and remove stop words such as “a”, “for”, “of”, and a couple dozen more (24). The analyzer then feeds the remaining stream of words into a Lucene-powered matching engine that utilizes a combination of vector-space modeling and straight Boolean modeling to both find and rank (score) any matches.

For example, an out-of-sample (new) news article contains “Mega Corp. announced that the board of directors of the company increased the company’s quarterly dividend rate 25% to $0.10 per common share from $0.08 per common share.” This text, after removing stop words and normalizing, would look something like “Mega Corp. announce board director company increase company quarterly dividend rate 25% $0.10 common share $0.08 common share”.

This would likely trigger a match against an in-sample (previously seen) news article that started out with “Zappum Electric announced that it has declared a regular quarterly cash dividend of $0.46 a share on the company's common stock, increasing the dividend 9.5% from the previous $0.42 a share.” and parsed down into “Zappum Electric announce declare regular quarterly cash dividend $0.46 share company common stock increase dividend 9.5% previous $0.42 share”.

This leads us to my first controllable parameter to fine-tune my strategy: match-score. By using a minimum score filter, I can eliminate noise and dial in better matches. This, in combination with the other adjustable parameters, allows me to balance my results to maximize potential profit. I will re-visit this concept later in the study.

Once I determine our matched set of news items, the analyzer queries market data for the date of publication and symbol affected[16]. I look at what the stock's price was at the moment of publication and capture this. Next, I capture the price at various time-spans after publication (1m, 3m, 5m, 10m, 20m, & 45m). I captured and stored all this information for the particular strategy for later testing.

STRATEGY TWO: SYNONYM SETS AND SPREAD ACTIVATION

The first strategy keys on words and their frequency and location. What if I could apply the same strategy to the meaning behind the word? I can approximate this by transforming keywords into synonyms, that is, words that are interchangeable with the keyword, both lexically and semantically. I'll not replace the original keyword, instead we will augment it.

For example, the statement "quick brown fox" (I have already lost the word "The" to the stop-word analyzer) feeds into a synonym engine that transforms the phrase. The engine then iterates over the words and creates cognitive synonyms (synsets) and injects them into the stream. After processing the first word in our phrase, my output may look like "quick|fast|rapid brown fox."

[16] To minimize complexity, I added a filter to process news items that only affected one company. Removing this filter is worthy of exploration in a future study.

Thus, if I had a news article enter the system that spoke of “farming,” I would have a high likelihood of matching another article about “agriculture” due to one including the other in its synset. In the simple keyword version, these two articles would have to only rely on enough common words to trigger an association. This hybrid spread-activation technique (22) should more closely model how humans connect concepts between similar text.

Implementing an engine to spread out semantics from an input stream involves an intermediate searching step. This interposed search utilizes an extracted version of WordNet - a system developed at Princeton University’s Cognitive Science Laboratory, driven by psychology professor George Miller. WordNet creates a network of “meaningfully related words and concepts” (27) and advertises itself as “a useful tool for computational linguistics and natural language processing.”

Extracted as a lexical database, the WordNet component allows me to create a second-stage input that contains not only the original keywords, but all their semantic cousins. This, admittedly large, news item could now be fed into the same Lucene matching engine as in our simple case. I collected matches and their related scores and followed the same path to generate outputs in the same form as our first strategy.

STRATEGY THREE: RANDOM MATCHING

The last strategy in the study most closely resembles the earlier-mentioned dart throwing. This matcher takes an input news item and pairs it with a random news item from the in-sample set. It makes predictions based upon how the price of the in-sample item moved and relates these to the input item. I repeat this matching numerous times to smooth the distribution and composed outputs that matched the other strategy results.

A random set gives the study a baseline to gauge performance. In his best-selling personal finance book, economist Burton Malkiel quite cleverly wrote that “a blindfolded monkey throwing darts at a newspaper’s financial pages could select a portfolio that would do just as well as one carefully selected by experts.” (28) The random strategy is our monkey, and he has made his picks; let’s see how they fared against the others.

CHAPTER 6 EXECUTION ENGINE

Once I produce the outputs, I can begin to test the predictions. My out-of-sample data set now becomes important as I "score" the matches with potential profits or losses. And, whether the strategy makes or loses money determines how effective it is.

```
5587,2356,33.11000,-0.00031908104658583280153158 9,-
0.00031908104658583280153158 9,0.00095724313975749840459476 71,0.001595405232929
1640076579451,0.0041480536056158264199106573,0.0063816209317166560306317805,0.
00030202355783751132588341 89,0.00030202355783751132588341 89,0.0009060706735125
339776502567,0,0.0009060706735125339776502567,0.0018121413470250679553005134

5596,3371,37.88250,0,0,0,0,-0.010302999543216769019946201 1,-
0.010302999543216769019946201 1,-0.021447898105985613409885831 2,-
0.0074572691876196132778987659,0.0065333597307463868540882993,0.02157988517125
32171847158978,-0.0029697089685210849336764997,-0.0048175278822675377812974329

5659,3495,58.00000,0.0016621054552089034847655537,0.002189381499726327312534 20
91,-0.0004561211457763181901112936,-
0.0017040686006203247582557927,0.0005473453749315818281335523,-
0.0005473453749315818281335523,0.00068965517241379310344827 59,0,0.000344827586
2068965517241379,0.0012068965517241379310344828,-
0.0013793103448275862068965517,0.0005172413793103448275862069
```

Figure 6.1. Sample Outputs.

My outputs come from files with rows of data similar to figure 6.1, above. The columns represent a serialized form of an AnalysisData object. This class represents:

- the Id of the “new” news item
- the Id of the matching news item
- the price of the stock at the time of the event
- the projected price changes over time intervals (1, 3, 5, 10, 20, & 45 minute) in percent
- the actual price changes over time intervals (1, 3, 5, 10, 20, & 45 minute) in percent.

Based on the AnalysisData input, the Execution Engine makes a simple decision: buy, sell, or do nothing. It determines this choice based upon the second of our controllable parameters: percentage threshold. This allows the engine to ignore changes that are sufficiently close to zero and choose the “do nothing” option. In my model, this has little bearing except to dampen both profit and loss somewhat. In the real world, where trading costs become a “friction” for every trade, a user may want to prohibit trading that produces less profit than the cost of the trade itself.

RISK MANAGEMENT

Common to most quantitative trading models (29), the execution engine uses size limiting to mitigate market risk. Often called Equal Position Weighting, this rule says that “if a position is good enough to own, no other information is needed.” This greatly simplifies the decision making tree that the engine needs to traverse.

In this engine, I set the size of the order to one hundred shares, regardless of price. A "common," or "round" lot[17] in securities trading, my one-hundred-share-size is either purchased or shorted (selling assets borrowed from a third party) at the price at the time of the news event. This gives the user a "position" in the stock. The only task remaining is to unload the position after the price moves.

REALIZING THE PREDICTION

My AnalysisData provides predicted price movements at various time intervals after entering into the position. In a real-world model, I would now feed my instructions (buy/sell/no-op) to an execution engine connected to an electronic market. In my test environment, I will use a simple execution emulator that instantly actualizes both buys and sells at exactly our price.

Once the engine executes our execution instructions, I decide how long to hold on to our new positions before realizing any profit/loss. This is the third controllable parameter in the study: Time-to-Realize.

My predictions provide intervals at which I can realize our profits/losses: 1m, 3m, 5m, 10m, 20m, and 45m. As this calculation is straightforward and fast, the execution engine provides results for all six intervals. Likewise, I can quickly calculate profit and loss for each percentage threshold level and produce outputs to demonstrate where real-world profitability "sweet spots" may exist.

[17] purchase amounts below 100 shares are called an "odd lot"

```
** Threshold Percentage: 0.08500%

Action at One minutes yields $0.0000 profit.

Action at Three minutes yields $148.1056 profit.

Action at Five minutes yields ($49.3685) profit.

Action at Ten minutes yields ($21.9416) profit.

Action at Twenty minutes yields ($82.2809) profit.

Action at Forty-Five minutes yields ($416.8899) profit.
```

Figure 6.2. Prediction Engine Sample Result.

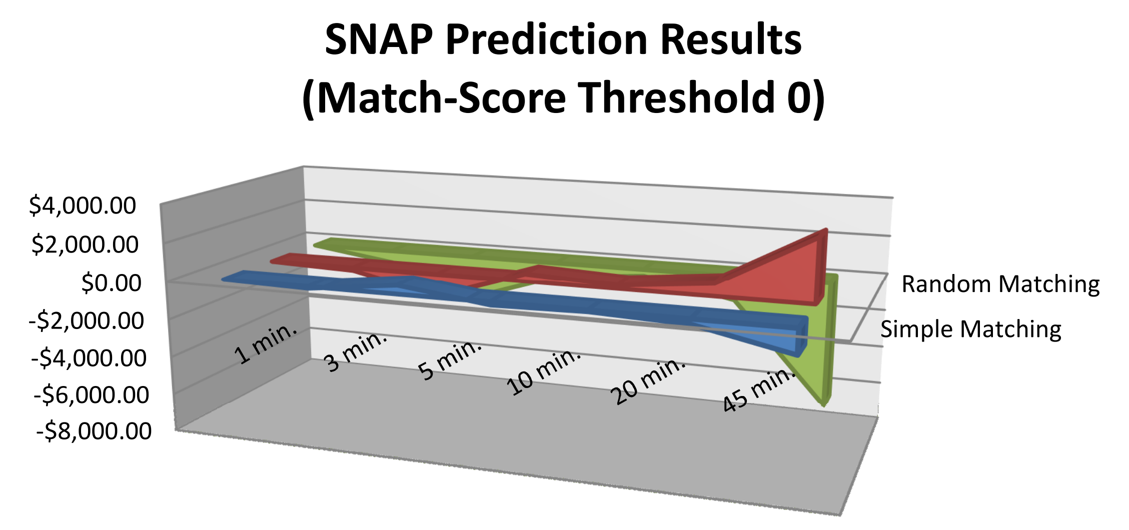
The engine produces output similar to the above Figure 6.2, where we can see that, for a particular threshold percentage of 0.085%, this strategy would lose money if the black box holds the position longer than three minutes. I took many output snapshots at a range of values for our controllable parameters. The next chapter discusses and analyzes these results.

CHAPTER 7 CONCLUSIONS

STATEMENT OF RESULTS

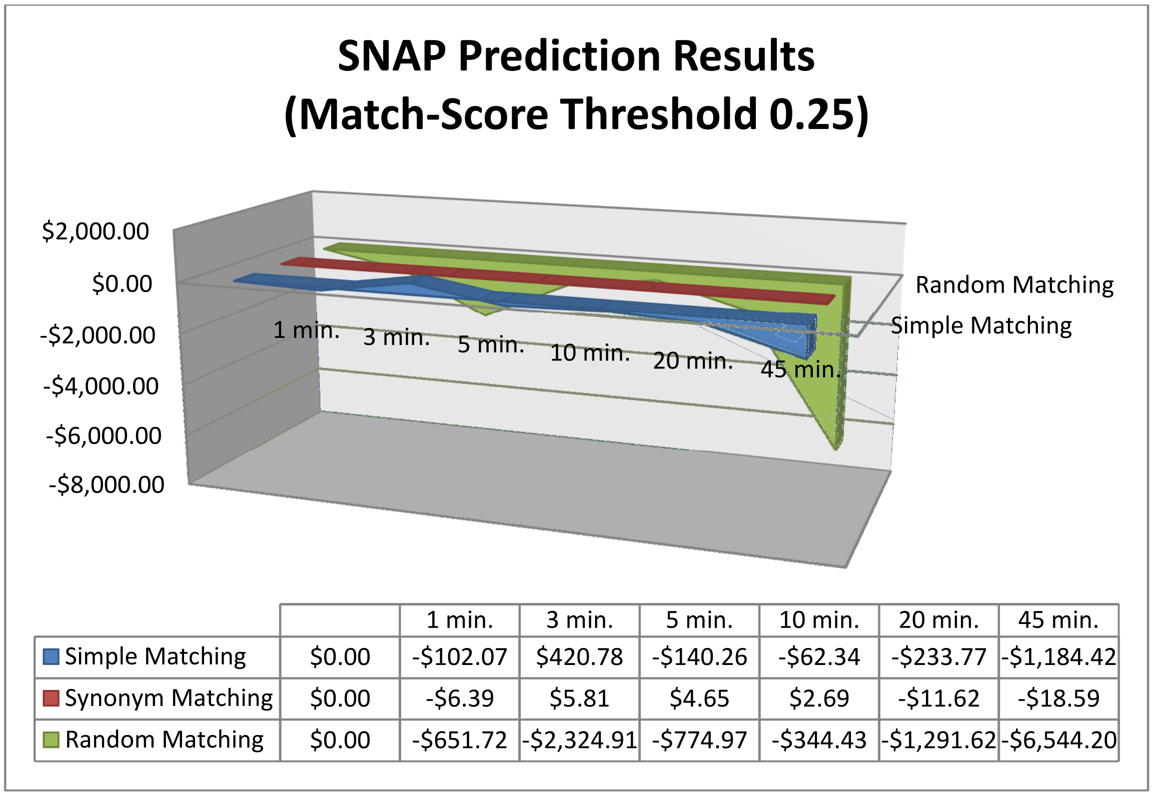
I conducted four sample runs while adjusting the minimum match-score with values 0.0, 0.25, 0.5, and 1.0. I experienced a sharp drop-off of all match-scores just beyond the 1.0 mark and those runs did not yield significant data. Other strategies began to yield little or no data as the match-score increased. I will discuss this below.

Each run used the Simple Analyzer and then the Synonym Analyzer. A run with random news matches provides a baseline contrast. With each match-score change, the results shifted somewhat. Figures 7.1 through 7.4 (below) show potential profit/loss at each position hold-time.



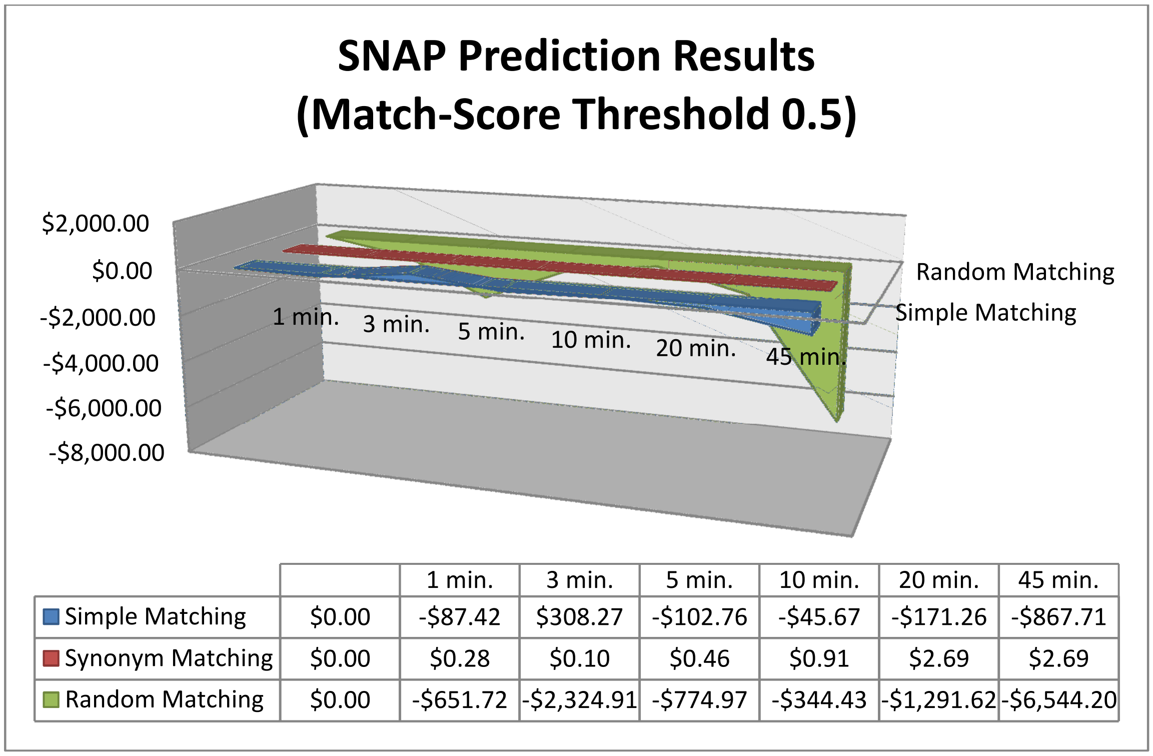
		1 min.	3 min.	5 min.	10 min.	20 min.	45 min.
🟦 Simple Matching	$0.00	-$93.76	$440.87	-$146.96	-$65.31	-$244.93	-$1,240.97
🟥 Synonym Matching	$0.00	-$136.73	-$1,157.86	$385.95	$171.54	$643.26	$3,259.17
🟩 Random Matching	$0.00	-$651.72	-$2,324.91	-$774.97	-$344.43	-$1,291.62	-$6,544.20

Figure 7.1. Results Summary at Threshold 0.



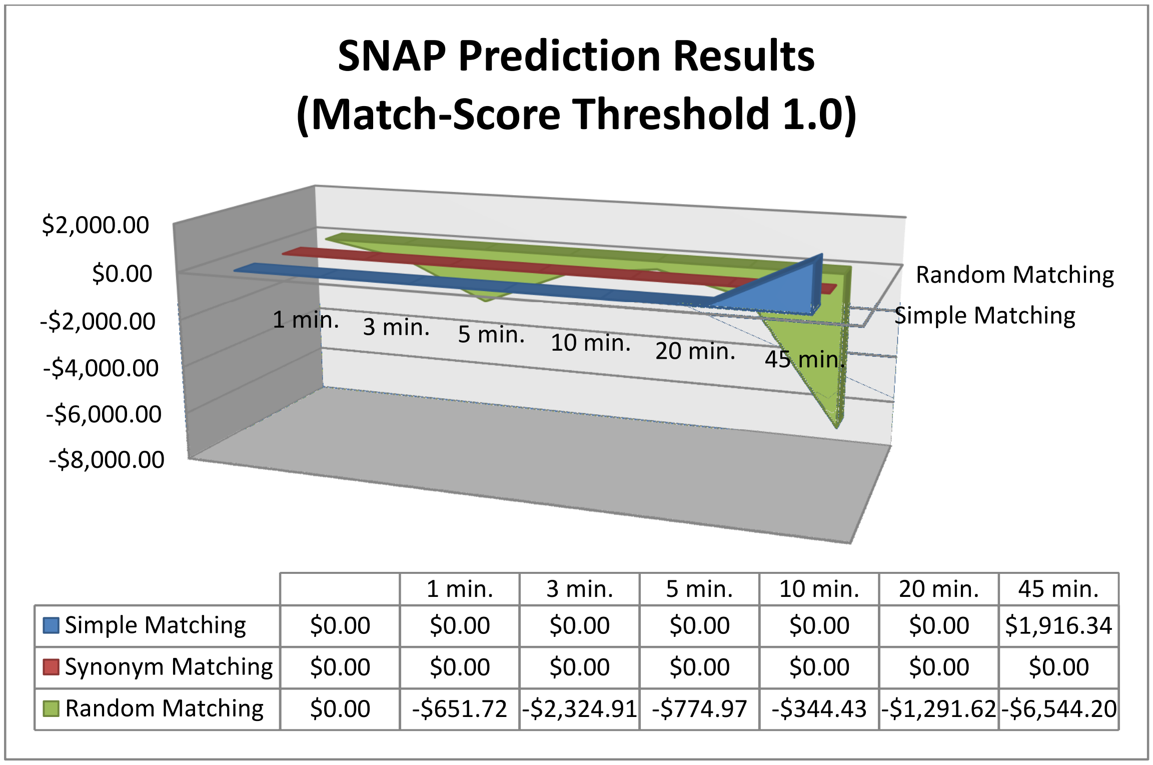
		1 min.	3 min.	5 min.	10 min.	20 min.	45 min.
🟦 Simple Matching	$0.00	-$102.07	$420.78	-$140.26	-$62.34	-$233.77	-$1,184.42
🟥 Synonym Matching	$0.00	-$6.39	$5.81	$4.65	$2.69	-$11.62	-$18.59
🟩 Random Matching	$0.00	-$651.72	-$2,324.91	-$774.97	-$344.43	-$1,291.62	-$6,544.20

Figure 7.2. Results Summary at Threshold 0.25.



		1 min.	3 min.	5 min.	10 min.	20 min.	45 min.
🟦 Simple Matching	$0.00	-$87.42	$308.27	-$102.76	-$45.67	-$171.26	-$867.71
🟥 Synonym Matching	$0.00	$0.28	$0.10	$0.46	$0.91	$2.69	$2.69
🟩 Random Matching	$0.00	-$651.72	-$2,324.91	-$774.97	-$344.43	-$1,291.62	-$6,544.20

Figure 7.3. Results Summary at Threshold 0.5.



		1 min.	3 min.	5 min.	10 min.	20 min.	45 min.
🟦 Simple Matching	$0.00	$0.00	$0.00	$0.00	$0.00	$0.00	$1,916.34
🟥 Synonym Matching	$0.00	$0.00	$0.00	$0.00	$0.00	$0.00	$0.00
🟩 Random Matching	$0.00	-$651.72	-$2,324.91	-$774.97	-$344.43	-$1,291.62	-$6,544.20

Figure 7.4. Results Summary at Threshold 1.0.

A couple of patterns warrant attention. First, the Simple strategy appears to have a consistent bump at the three minute mark. I expect the market to move in the direction of the news, and then recover. These results gently support that theory.

Secondly, the Synonym strategy gave promising early results with its best showing occurring in the lowest match-score set. However, its performance quickly degraded as the match-score threshold increased. A quick look into the raw (pre-execution) Analysis Data helps explain the outcome.

Each data run produced Analysis Data files (Figure X in chapter six). Lower matching score thresholds mean more results. Toward the high end of the match-scores (at 1.0), the Synonym Matcher found no matching news items and produced empty output. This explains the zeros in the Profit/Loss column at the 1.0 mark.

As the threshold increases from 0.0 to 1.0, the analyzer recorded fewer and fewer matches. Figure 7.5 (below) illustrates the data drop-off

Match-Score Threshold	Simple output size (KB)	Synonym output size (KB)
0.00	115	120
0.25	108	2
0.50	73	1
1.00	37	0

Figure 7.5. Data Size over Threshold Increase.

PROBLEMS LEFT UNRESOLVED

Clearly, the poor output of the Synonym strategy with higher matching score thresholds demands some further attention in the future. This strategy demonstrated promising results when left to run unrestricted. The match-score filter may not be the best way to refine these strategies.

For the Synonym strategy, if I had the resources, I would fine tune the matching of the expanded synset with the original article. I would like to explore expanding the in-sample articles as well, and then applying a simple matching strategy over both modified sets.

The positive showing of the Simple matching strategy at short intervals shows promise that an uncomplicated approach may be advantageous over a complex solution. What separates this study from a more definitive one is the amount of sample data.

As mentioned previously, the large volume of data became cumbersome as the study progressed. However, I need more data to further vet all approaches (Simple, Synonym, and beyond) and I've reached the capacity of most personal workstations.

For future research, I recommend a distributed database over a network of (relatively) small servers. If I were to mold the study in this direction, I would explore fast, distributed, in-memory databases like MongoDB or CouchDB. Faster data access would allow tighter cycles of experimentation. Current test runs border on the tedious with an over-the-counter computer system.

BIBLIOGRAPHY

1. **Berners-Lee, Tim.** *Weaving the Web.* San Francisco : Harper, 1999.

2. *Circular context-based semantic matching to identify web service composition.* **Segev, Aviv.** New York : ACM, 2008. ISBN: 978-1-60558-107-1.

3. **Papazoglou, Michael.** *Web Services: Principles and Technology.* s.l. : Prentice Hall, 2007. ISBN-13: 978-0321155559.

4. *Swoogle: A search and metadata engine for the semantic web.* **Finin, Tim, et al., et al.** s.l. : ACM Press, 2004. ISBN:1-58113-874-1.

5. **McConnell, Steve.** *Code Complete: A Practical Handbook of Software Construction.* s.l. : Microsoft Press, 2004. ISBN-13: 978-0735619678.

6. *Automatic Generation of Data Types for Classification of Deep Web Sources.* **Ngu, Anne H. H., Buttler, David and Critchlow, Terence.** s.l. : DILS, 2005.

7. *Searching Service Repositories by Combining Semantic and Ontological Matching.* **Syeda-Mahmood, Tanveer, et al., et al.** s.l. : Proceedings of the IEEE International Conference on Web Services, 2005. ISBN:0-7695-2409-5.

8. *Semantic methods for service categorization - an empirical study.* **Gal, Avigdor, Segev, Aviv and Toch, Eran.** s.l. : Proceedings International Workshop on Semantic Data and Service Integration, 2007.

9. *Generating Semantic Descriptions of Web And Grid Services.* **Babik, Marian, et al., et al.** s.l. : Proceedings of the Cracow Grid Workshop, 2005. ISBN 83-915141-5-3.

10. **Dong, Xin, Madhavan, Jayant and Halevy, Alon.** Mining Structures for Semantics. *ACM SIGKDD Explorations.* 2004, Vol. 6, 2.

11. **Han, Jiawei.** *Data Mining: Concepts and Techniques.* San Francisco : Morgan Kaufmann Publishers Inc., 2000. ISBN:1558609016.

12. *The Time-Series Link Prediction Problem with Applications in Communication Surveillance.* **Huang, Zan and Lin, Dennis K. J.** 2, Linthicum : INFORMS Journal on Computing, 2009, Vol. 21. ISSN: 1526-5528 doi>10.1287/ijoc.1080.0292.

13. **O'Madadhain, Joshua, Hutchins, Jon and Smyth, Padhraic.** Prediction and ranking algorithms for event-based network data. *ACM SIGKDD Explorations.* 2005, Vol. 7, 2.

14. *The Link Prediction Problem for Social Networks.* **Liben-Nowell, David and Kleinberg, Jon.** s.l. : ACM New York, 2003. ISBN: 1-58113-723-0.

15. *Models and Methods for Prediction Problem of Evolving Graphs.* **Chapanond, Anurat and Krishnamoorthy, Mukkai S.** Taipei : ISI 2008. IEEE International Conference, 2008. Print ISBN: 978-1-4244-2414-6.

16. *What Moves Stock Prices?* **Cutler, David M., Poterba, James M. and Summers, Lawrence H.** s.l. : NBER Working Paper Series, 1989, Vol. w2538.

17. *BlogScope: A System for Online Analysis of High Volume Text Streams.* **Bansal, Nilesh and Koudas, Nick.** s.l. : In VLDB, 2007.

18. *A Discrete Stock Price Prediction Engine Based on Financial News.* **Schumaker, Robert P. and Chen, Hsinchun.** 1, s.l. : IEEE Computer - COMPUTER, 2010, Vol. 43.

19. *Textual analysis of stock market prediction using breaking financial news: The AZFin text system.* **Schumaker, Robert P. and Chen, Hsinchun.** 2, s.l. : ACM Transactions on Information Systems (TOIS), 2009, Vol. 27.

20. *Forecasting Intraday Stock Price Trends with Text Mining Techniques.* **Mittermayer, Marc-Andre.** s.l. : Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04), 2004. ISBN:0-7695-2056-1.

21. *An ontology based framework for mining dependence relationships between news and financial instruments.* **Want, Shanshan, et al., et al.** 10, Tarrytown, NY : Pergamon Press, Inc., 2011, Vol. 38.

22. *A Hybrid Approach for Searching in The Semantic Web.* **Rocha, Cristiano, Schwabe, Daniel and Aragao, Marcus Poggi.** s.l. : Proceedings of the 13th International Conference on World Wide Web, 2004. ISBN:1-58113-844-X.

23. *PowerMap: Mapping the Real Semantic Web on the Fly.* **Lopez, Vanessa, Sabou, Marta and Motta, Enrico.** s.l. : ISWC-06. Volume 4273 of LNCS, 2006.

24. **McCandless, Michael, Hatcher, Erik and Gospodnetic, Otis.** *Lucene In Action, 2nd Ed.* s.l. : Manning, 2010. ISBN: 1933988177.

25. *Interpreting TF-IDF term weights as making relevance decisions.* **Wu, Ho Chung, et al., et al.** 3, s.l. : ACM Transactions on Information Systems (TOIS), 2008, Vol. 26.

26. *Automatic Discovery and Inferencing of Complex Bioinformatics Web Interfaces.* **Ngu, Anne H., Rocco, Daniel and Critchlow, Terence.** 4, s.l. : World Wide Web, 2005, Vol. 8.

27. **Fellbaum, Christiane.** *WordNet: An Electronic Lexical Database.* s.l. : Bradford Books, 1998.

28. **Malkiel, Burton G.** *A Random Walk Down Wall Street.* s.l. : W. W. Norton & Company, 2007. ISBN-13: 978-0393330335.

29. **Narang, Rishi K.** *Inside the Black Box: The Simple Truth About Quantitative Trading.* s.l. : Wiley, 2009. ISBN-13: 978-0470432068.

VITA

Seth Rowell Orell, Jr., the son of LTC Seth R. Orell and Kathleen O. Orell, completed his high school work at Westlake High School, Austin, Texas, in 1988. After many years as a professional drummer, karate instructor, and nightclub D.J., he entered the University of Texas at Austin where he received the degree of Bachelor of Science in Computer Science. During the following years he worked as a software engineer for a stock trading company where the idea for this thesis was born. In September 2008, he entered Texas State University-San Marcos to pursue the degree of Master of Science in Computer Science.

Permanent Address: seth.orell@gmail.com

This thesis was typed by Seth R. Orell, Jr.