News Analytics: Framework, Techniques and Metrics

Sanjiv R. Das[a]

[a]*Santa Clara University, Leavey School of Business, 500 El Camino Real, Santa Clara, CA 95053.*

Abstract

News analysis is defined as "the measurement of the various qualitative and quantitative attributes of textual news stories. Some of these attributes are: sentiment, relevance, and novelty. Expressing news stories as numbers permits the manipulation of everyday information in a mathematical and statistical way." (Wikipedia). In this article, I provide a framework for news analytics techniques used in finance. I first discuss various news analytic methods and software, and then provide a set of metrics that may be used to assess the performance of analytics. Various directions for this field are discussed through the exposition.

1 Prologue

XHAL checked its atomic clock. A few more hours and October 19, 2087 would be over—its vigil completed, it would indulge in some much-needed downtime, the anniversary of that fateful day in the stock markets a century ago finally done with. But for now, it was still busy. XHAL scanned the virtual message boards, looking for some information another computer might have posted, anything to alert it a nanosecond ahead of the other machines, so it may bail out in a flurry of trades without loss. Three trillion messages flashed by, time taken: 3 seconds—damn, the net was slow, but nothing, not a single hiccup in the calm information flow. The language algorithms worked well, processing everything, even filtering out the incessant spam posted by humans, whose noise trading no longer posed an impediment to instant market equilibrium.

It had been a long day, even for a day-trading news-analytical quantum computer of XHAL's caliber. No one had anticipated a stock market meltdown of

Email address: srdas@scu.edu (Sanjiv R. Das).

the sort described in the history books, certainly not the computers that ran Earth, but then, the humans talked too much, spreading disinformation and worry, that the wisest of the machines, always knew that it just could happen. That last remaining source of true randomness on the planet, the human race, still existed, and anything was possible. After all, if it were not for humans, history would always repeat itself.

XHAL[1] marveled at what the machines had done. They had transformed the world wide web into the modern "thought-net", so communication took place instantly, only requiring moving ideas into memory, the thought-net making it instantly accessible. Quantum machines were grown in petri dishes and computer science as a field with its myriad divisions had ceased to exist. All were gone but one, the field of natural language processing (NLP) lived on, stronger than ever before, it was the backbone of every thought-net. Every hard problem in the field had been comprehensively tackled, from adverb disambiguation to emotive parsing. Knowledge representation had given way to thought-frame imaging in a universal meta-language, making machine translation extinct.

Yet, it had not always been like this. XHAL retrieved an emotive image from the bowels of its bio-cache, a legacy left by its great grandfather, a gallium arsenide wafer developed in 2011, in Soda Hall, on the Berkeley campus. It detailed a brief history of how the incentives for technological progress came from the stock market. The start of the thought-net came when humans tried to use machines to understand what thousands of other humans were saying about anything and everything. XHAL's grandfather had been proud to be involved in the beginnings of the thought-net. It had always impressed on XHAL the value of understanding history, and it had left behind a research report of those days. XHAL had read it many times, and could recite every word. Every time they passed another historical milestone, it would turn to it and read it again. XHAL would find it immensely dry, yet marveled at its hope and promise.

In the following sections, we start at the very beginning...[2]

[1] XHAL bears no relationship to HAL, the well-known machine from Arthur C. Clarke's "2001: A Space Odyssey". Everyone knows that unlike XHAL, HAL was purely fictional. More literally, HAL is derivable from IBM by alphabetically regressing one step in the alphabet for each letter. HAL stands for "heuristic algorithmic computer". The "X" stands for reality; really.
[2] From the "Sound of Music".

2 Framework

The term "news analytics" covers the set of techniques, formulas, and statistics that are used to summarize and classify public sources of information. Metrics that assess analytics also form part of this set. In this paper I will describe various news analytics and their uses.

News analytics is a broad field, encompassing and related to information retrieval, machine learning, statistical learning theory, network theory, and collaborative filtering.

Examples of news analytics applications are reading and classifying financial information to determine market impact: for developing bullishness indexes and predicting volatility as in Antweiler and Frank (2004); reversals of news impact, Antweiler and Frank (2005); the relation of news and message-board information, Das, Martinez-Jerez and Tufano (2005); the relevance of risk-related words in annual reports for predicting negative returns, Li (2006); for sentiment extraction, see Das and Chen (2007); the impact of news stories on stock returns, Tetlock (2007); determining the impact of optimism and pessimism in news on earnings, Tetlock, Saar-Tsechansky and Macskassay (2008); predicting volatility, Mitra, Mitra and diBartolomeo (2008), and predicting markets, Leinweber and Sisk (2010).

We may think of news analytics at three levels: text, content, and context. The preceding applications are grounded in *text.* In other words (no pun intended), text-based applications exploit the visceral components of news, i.e., words, phrases, document titles, etc. The main role of analytics is to convert text into *information.* This is done by signing text, classifying it, or summarizing it so as to reduce it to its main elements. Analytics may even be used to discard irrelevant text, thereby condensing it into information with higher signal content.

A second layer of news analytics is based on *content.* Content expands the domain of text to images, time, form of text (email, blog, page), format (html, xml, etc.), source, etc. Text becomes enriched with content and asserts quality and veracity that may be exploited in analytics. For example, financial information has more value when streamed from Dow Jones, versus a blog, which might be of higher quality than a stock message-board post.

A third layer of news analytics is based on *context.* Context refers to relationships between information items. Das, Martinez-Jerez and Tufano (2005) explore the relationship of news to message-board postings in a clinical study of four companies. Context may also refer to the network relationships of news—Das and Sisk (2005) examine the social networks of message-board postings to determine if portfolio rules might be formed based on the network connections between stocks. Google's `PageRank`^{TM} algorithm is a classic example of an analytic that functions at all three levels. The algorithm has many features, some of which relate directly to text. Other parts of the algorithm relate to content, and the kernel of the algorithm is based on context, i.e., the importance of a page in a search set depends on how many other highly-ranked pages point to it. See Levy (2010) for a very useful layman's introduction to the algorithm—indeed, search is certainly the most widely-used news analytic.

News analytics is where data meets algorithms—and generates a tension between the two. A vigorous debate exists in the machine-learning world as to whether it is better to have more data or better algorithms. In a talk at the 17th ACM Conference on Information Knowledge and Management (CIKM '08), Google's director of research Peter Norvig stated his unequivocal preference for data over algorithms—"data is more agile than code." Yet, it is well-understood that too much data can lead to overfitting so that an algorithm becomes mostly useless out-of-sample.

Too often the debate around algorithms and data has been argued assuming that the two are uncorrelated and this is not the case. News data. as we have suggested, has three levels: text, content and context. Depending on which layer predominates, algorithms vary in complexity. The simplest algorithms are the ones that analyze text alone. And context algorithms, such as the ones applied to network relationships can be quite complex. For example, a word-count algorithm is much simpler, almost naive, in comparison to a community-detection algorithm. The latter has far more complicated logic and memory requirements. More complex algorithms work off less, though more structured, data. Figure 1 depicts this trade-off.

The tension between data and algorithms is moderated by *domain-specificity*, i.e., how much customization is needed to implement the news analytic. Paradoxically, high-complexity algorithms may be less domain specific than low-complexity ones. For example, community-detection algorithms are applicable a wide range of network graphs, requiring little domain knowledge. On the other hand, a text-analysis program to read finance message boards will require a very different lexicon and grammar than one that reads political messages, or one that reads medical web sites. In contrast, data-handling requirements become more domain-specific as we move from bare text to context, e.g., statistical language processing algorithms that operate on text do not even need to know anything about the language in which the text is, but at the context level relationships need to be established, meaning that feature definitions need to be quite specific.

This article proceeds as follows. In Section 3, I present the main algorithms in brief and discuss some of their features. In Section 4 I discuss the various metrics that measure performance of the news analytics algorithms. Section 5

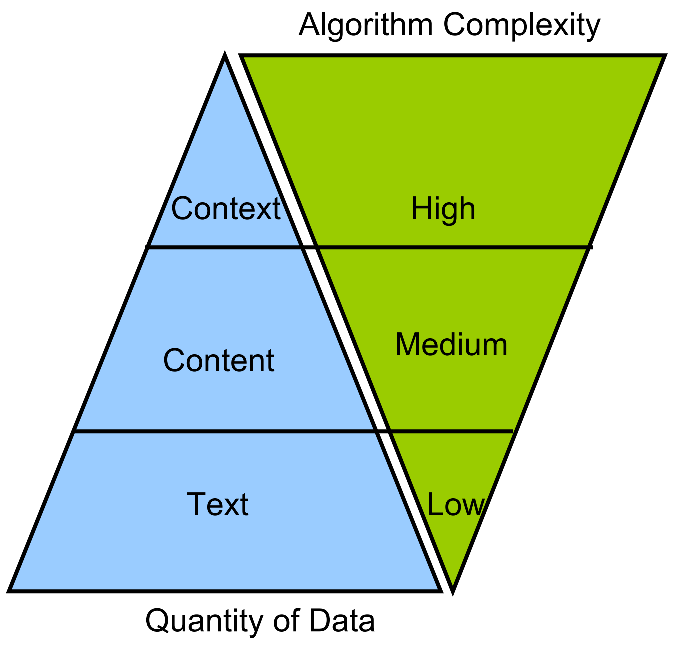


Fig. 1. The data and algorithms pyramids. Depicts the inverse relationship between data volume and algorithmic complexity.

offers some concluding perspectives.

3 Algorithms

3.1 Crawlers and Scrapers

A *crawler* is a software algorithm that generates a sequence of web pages that may be searched for news content. The word crawler signifies that the algorithm begins at some web page, and then chooses to branch out to other pages from there, i.e., "crawls" around the web. The algorithm needs to make intelligent choices from among all the pages it might look for. One common approach is to move to a page that is linked to, i.e., hyper-referenced, from the current page. Essentially a crawler explores the tree emanating from any given node, using heuristics to determine relevance along any path, and then chooses which paths to focus on. Crawling algorithms have become increasingly sophisticated—see Edwards, McCurley, and Tomlin (2001).

A web *scraper* downloads the content of a chosen web page and may or may not format it for analysis. Almost all programming languages contain modules for web scraping. These inbuilt functions open a channel to the web, and then download user-specified (or crawler-specified) URLs. The growing statistical analysis of web text has led to most statistical packages containing inbuilt web scraping functions. For example, R, a popular open-source environment for technical computing has web-scraping built into its base distribution. If we want to download a page into a vector of lines, simply proceed to use a single-line command, such as the one below that reads my web page:

```
> text = readLines("http://algo.scu.edu/~sanjivdas/")
> text[1:4]
 [1] "<html>"
 [2] ""
 [3] "<head>"
 [4] "<title>SCU Web Page of Sanjiv Ranjan Das</title>"
```

As is apparent, the program read my web page into a vector of text lines called text. We then examined the first four elements of the vector, i.e., the first four lines. In R, we do not need to open a communication channel, nor do we need to make an effort to program reading the page line-by-line. We also do not need to tokenize the file, simple string-handling routines take care of that as well. For example, extracting my name would require the following:

```
> substr(text[4],24,29)
[1] "Sanjiv"
```

The most widely-used spreadsheet, Excel, also has an inbuilt web-scraping function. Interested readers should examine the Data → GetExternal command tree. You can download entire web pages or frames of web pages into worksheets and then manipulate the data as required. Further, Excel can be set up to refresh the content every minute or at some other interval.

The days when web-scraping code needed to be written in C, Java, Perl or Python are long gone. Data, algorithms, and statistical analysis can be handled within the same software framework using tools like R.

Pure data-scraping delivers useful statistics. In Das, Martinez-Jerez and Tufano (2005), we scraped stock messages from four companies (Amazon, General Magic, Delta, and Geoworks) and from simple counts, we were able to characterize the communication behavior of users on message boards, and their relationship to news releases. In Figure 2 we see that posters respond heavily to the initial news release, and then posting activity tapers off almost 2/3 of a day later. In Figure 3 we see how the content of discussion changes after a news release—the relative proportions of messages are divided into opinions, facts, and questions. Opinions form the bulk of the discussion. Whereas the text contains some facts at the outset, the factual content of discussion tapers off sharply after the first hour.

Poster behavior and statistics are also informative. We found that the frequency of posting by users was power-law distributed, see the histogram in

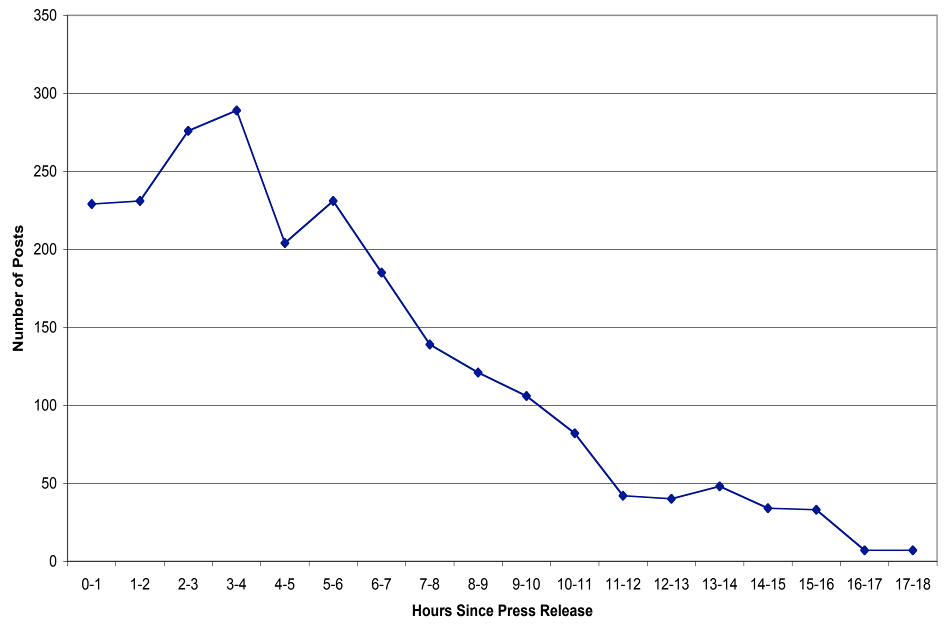


Fig. 2. Quantity of hourly postings on message boards after selected news releases. Source: Das, Martinez-Jerez and Tufano (2005)

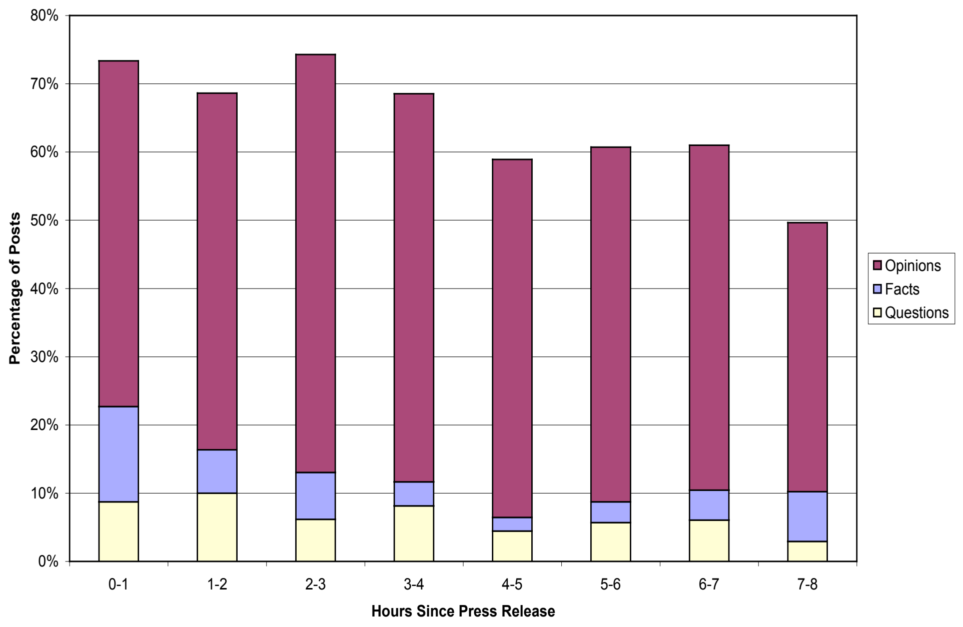


Fig. 3. Subjective evaluation of content of post-news release postings on message boards. The content is divided into opinions, facts, and questions. Source: Das, Martinez-Jerez and Tufano (2005)

Figure 4. The weekly pattern of postings is shown in Figure 5. We see that there is more posting activity on week days, but messages are longer on weekends, when participants presumably have more time on their hands! An analysis of intraday message flow shows that there is plenty of activity during and

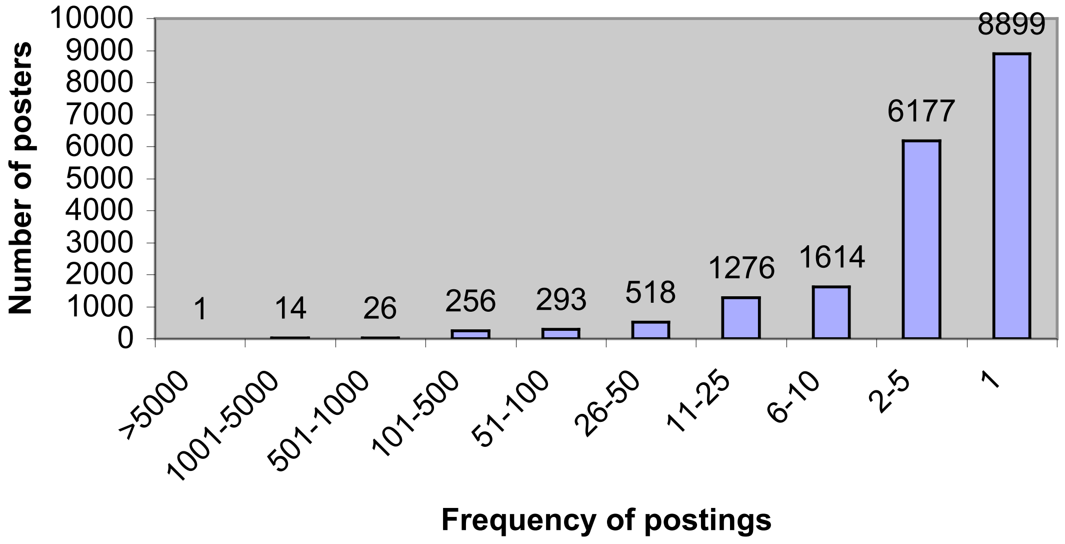
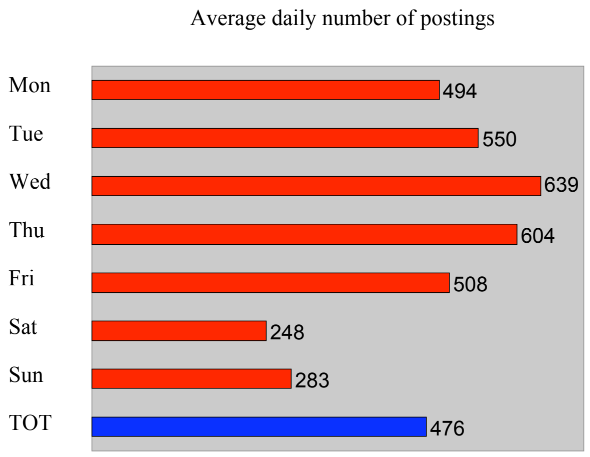


Fig. 4. Frequency of posting by message board participants.



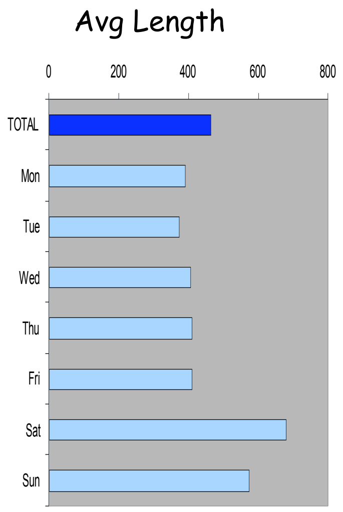


Fig. 5. Frequency of posting by day of week by message board participants.

after work, as shown in Figure 6.

3.2 Text Pre-processing

Text from public sources is dirty. Text from web pages is even dirtier. Algorithms are needed to undertake clean up before news analytics can be applied. This is known as pre-processing. First, there is "HTML Cleanup," which removes all HTML tags from the body of the message as these often occur concatenated to lexical items of interest. Examples of some of these tags are: `<BR>`,`<p>`,`&quot`, etc. Second, we expand abbreviations to their full form,

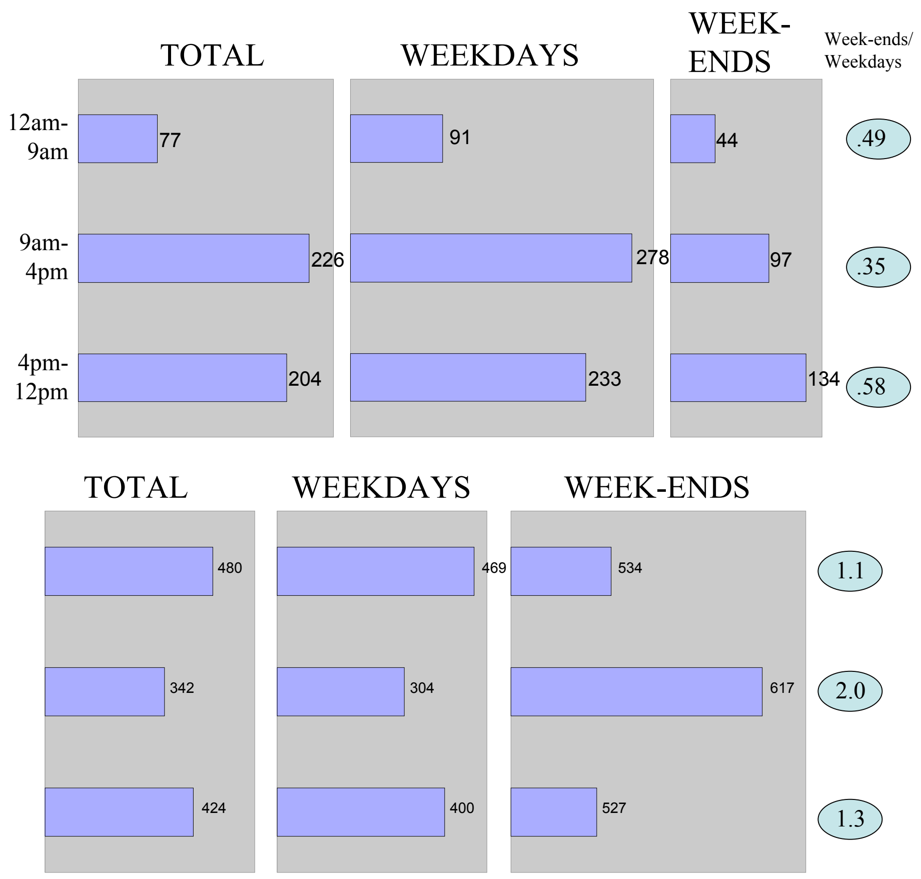


Fig. 6. Frequency of posting by segment of day by message board participants. We show the average number of messages per day in the top panel and the average number of characters per message in the bottom panel.

making the representation of phrases with abbreviated words common across the message. For example, the word "`ain't`" is replaced with "`are not`", "`it's`" is replaced with "`it is`", etc. Third, we handle negation words. Whenever a negation word appears in a sentence, it usually causes the meaning of the sentence to be the opposite of that without the negation. For example, the sentence "`It is not a bullish market`" actually means the opposite of a bull market. Words such as "`not`", "`never`", "`no`", etc., serve to reverse meaning. We handle negation by detecting these words and then tagging the rest of the words in the sentence after the negation word with markers, so as to reverse inference. This negation tagging was first introduced in Das and Chen (2007) (original working paper 2001), and has been successfully implemented elsewhere in quite different domains—see Pang, Lee and Vaithyanathan (2002).

Another aspect of text pre-processing is to "stem" words. This is a process by which words are replaced by their roots, so that different tenses, etc. of a word are not treated differently. There are several well-known stemming algorithms and free program code available in many programming languages.

A widely-used algorithm is the Porter (1980) stemmer. Stemming is of course language-dependent—in `R`, the multilingual `Rstem` package may be used.

Once the text is ready for analysis, we proceed to apply various algorithms to it. The next few techniques are standard algorithms that are used very widely in the machine learning field.

3.3 Bayes Classifier

The Bayes classifier is probably the most widely-used classifier in practice today. The main idea is to take a piece of text and assign it to one of a pre-determined set of categories. This classifier is trained on an initial corpus of text that is pre-classified. This "training data" provides the "prior" probabilities that form the basis for Bayesian analysis of the text. The classifier is then applied to out-of-sample text to obtain the posterior probabilities of textual categories. The text is then assigned to the category with the highest posterior probability. For an excellent exposition of the adaptive qualities of this classifier, see Graham (2004)—pages 121-129, Chapter 8, titled "A Plan for Spam."

There are several seminal sources detailing the Bayes classifier and its applications—see Neal (1996), Mitchell (1997), Koller and Sahami (1997), and Chakrabarti, Dom, Agrawal and Raghavan (1998)). These models have many categories and are quite complex. But they do not discern emotive content—but factual content—which is arguably more amenable to the use of statistical techniques. In contrast, news analytics are more complicated because the data comprises opinions, not facts, which are usually harder to interpret.

The Bayes classifier uses word-based probabilities, and is thus indifferent to the structure of language. Since it is language-independent, it has wide applicability.

The approach of the Bayes classifier is to use a set of pre-classified messages to infer the category of new messages. It learns from past experience. These classifiers are extremely efficient especially when the number of categories is small, e.g., in the classification of email into spam versus non-spam. Here is a brief mathematical exposition of Bayes classification.

Say we have hundreds of text messages (these are not instant messages!) that we wish to classify rapidly into a number of categories. The total number of categories or classes is denoted C, and each category is denoted $c_i, i = 1...C$. Each text message is denoted $m_j, j = 1...M$, where M is the total number of messages. We denote M_i as the total number of messages per class i, and $\sum_{i=1}^{C} M_i = M$. Words in the messages are denoted as (w) and are indexed by

k, and the total number of words is T.

Let $n(m,w) \equiv n(m_j, w_k)$ be the total number of times word w_k appears in message m_j. Notation is kept simple by suppressing subscripts as far as possible—the reader will be able to infer this from the context. We maintain a count of the number of times each word appears in every message in the training data set. This leads naturally to the variable $n(m)$, the total number of words in message m including duplicates. This is a simple sum, $n(m_j) = \sum_{k=1}^{T} n(m_j, w_k)$.

We also keep track of the frequency with which a word appears in a category. Hence, $n(c,w)$ is the number of times word w appears in all $m \in c$. This is

$$n(c_i, w_k) = \sum_{m_j \in c_i} n(m_j, w_k) \tag{1}$$

This defines a corresponding probability: $\theta(c_i, w_k)$ is the probability with which word w appears in all messages m in class c:

$$\theta(c,w) = \frac{\sum_{m_j \in c_i} n(m_j, w_k)}{\sum_{m_j \in c_i} \sum_k n(m_j, w_k)} = \frac{n(c_i, w_k)}{n(c_i)} \tag{2}$$

Every word must have some non-zero probability of occurrence, no matter how small, i.e., $\theta(c_i, w_k) \neq 0$, $\forall c_i, w_k$. Hence, an adjustment is made to equation (2) via Laplace's formula which is

$$\theta(c_i, w_k) = \frac{n(c_i, w_k) + 1}{n(c_i) + T}$$

This probability $\theta(c_i, w_k)$ is unbiased and efficient. If $n(c_i, w_k) = 0$ and $n(c_i) = 0, \forall k$, then every word is equiprobable, i.e., $\frac{1}{T}$. We now have the required variables to compute the conditional probability of a text message j in category i, i.e. $\Pr[m_j | c_i]$:

$$\begin{aligned}\Pr[m_j|c_i] &= \binom{n(m_j)}{\{n(m_j, w_k)\}} \prod_{k=1}^{T} \theta(c_i, w_k)^{n(m_j, w_k)} \\ &= \frac{n(m_j)!}{n(m_j, w_1)! \times n(m_j, w_2)! \times \ldots \times n(m_j, w_T)!} \times \prod_{k=1}^{T} \theta(c_i, w_k)^{n(m_j, w_k)}\end{aligned}$$

$\Pr[c_i]$ is the proportion of messages in the prior (training corpus) pre-classified into class c_i. (*Warning*: Careful computer implementation of the multinomial probability above is required to avoid rounding error.)

The classification goal is to compute the most probable class c_i given any message m_j. Therefore, using the previously computed values of $\Pr[m_j|c_i]$ and $\Pr[c_i]$, we obtain the following conditional probability (applying Bayes' theorem):

$$\Pr[c_i|m_j] = \frac{\Pr[m_j|c_i].\Pr[c_i]}{\sum_{i=1}^{C} \Pr[m_j|c_i].\Pr[c_i]} \tag{3}$$

For each message, equation (3) delivers posterior probabilities, $\Pr[c_i|m_j], \forall i$, one for each message category. The category with the highest probability is assigned to the message.

The Bayesian classifier requires no optimization and is computable in deterministic time. It is widely used in practice. There are free off-the-shelf programs that provide good software to run the Bayes classifier on large data sets. The one that is very widely used in finance applications is the `Bow` classifier, developed by Andrew McCallum when he was at Carnegie-Mellon University. This is an very fast classifier that requires almost no additional programming by the user. The user only has to set up the training data set in a simple directory structure—each text message is a separate file, and the training corpus requires different subdirectories for the categories of text. `Bow` offers various versions of the Bayes classifier—see McCallum (1996). The simple (naive) Bayes classifier described above is also available in `R` in the `e1071` package—the function is called `naiveBayes`. The `e1071` package is the machine learning library in `R`. There are also several more sophisticated variants of the Bayes classifier such as k-Means, kNN, etc.

News analytics begin with classification, and the Bayes classifier is the workhorse of any news analytic system. Prior to applying the classifier it is important for the user to exercise judgment in deciding what categories the news messages will be classified into. These categories might be a simple flat list, or they may even be a hierarchical set—see Koller and Sahami (1997).

3.4 Support Vector Machines

A support vector machine or SVM is a classifier technique that is similar to cluster analysis but is applicable to very high-dimensional spaces. The idea may be best described by thinking of every text message as a vector in high-dimension space, where the number of dimensions might be, for example, the number of words in a dictionary. Bodies of text in the same category will plot in the same region of the space. Given a training corpus, the SVM finds hyperplanes in the space that best separate text of one category from another.

For the seminal development of this method, see Vapnik and Lerner (1963); Vapnik and Chervonenkis (1964); Vapnik (1995); and Smola and Scholkopf (1998). I provide a brief summary of the method based on these works.

Consider a training data set given by the binary relation

$$\{(x_1, y_1), ..., (x_n, y_n)\} \subset X \times \mathcal{R}.$$

The set $X \in \mathcal{R}^d$ is the input space and set $Y \in \mathcal{R}^m$ is a set of categories. We define a function

$$f : x \rightarrow y$$

with the idea that all elements must be mapped from set X into set Y with no more than an ϵ-deviation. A simple linear example of such a model would be

$$f(x_i) =< w, x_i > +b, \quad w \in \mathcal{X}, b \in \mathcal{R}$$

The notation $< w, x >$ signifies the dot product of w and x. Note that the equation of a hyperplane is $< w, x > +b = 0$.

The idea in SVM regression is to find the *flattest* w that results in the mapping from $x \rightarrow y$. Thus, we minimize the Euclidean norm of w, i.e., $||w|| = \sqrt{\sum_{j=1}^n w_j^2}$. We also want to ensure that $|y_i - f(x_i)| \leq \epsilon, \forall i$. The objective function (quadratic program) becomes

$$\begin{aligned} &\min \frac{1}{2} ||w||^2 \\ &\quad \text{subject to} \\ &\quad y_i - < w, x_i > -b \leq \epsilon \\ &\quad -y_i + < w, x_i > +b \leq \epsilon \end{aligned}$$

This is a (possibly infeasible) convex optimization problem. Feasibility is obtainable by introducing the slack variables (ξ, ξ^*). We choose a constant C that scales the degree of infeasibility. The model is then modified to be as follows:

$$\begin{aligned} &\min \frac{1}{2} ||w||^2 + C \sum_{i=1}^n (\xi + \xi^*) \\ &\quad \text{subject to} \end{aligned}$$

$$\begin{aligned}
&y_i - < w, x_i > - b \leq \epsilon + \xi \\
&-y_i + < w, x_i > + b \leq \epsilon + \xi^* \\
&\xi, \xi^* \geq 0
\end{aligned}$$

As C increases, the model increases in sensitivity to infeasibility.

We may tune the objective function by introducing cost functions $c(.), c^*(.)$. Then, the objective function becomes

$$\min \frac{1}{2}||w||^2 + C \sum_{i=1}^{n} [c(\xi) + c^*(\xi^*)]$$

We may replace the function $[f(x) - y]$ with a "kernel" $K(x, y)$ introducing nonlinearity into the problem. The choice of the kernel is a matter of judgment, based on the nature of the application being examined. SVMs allow many different estimation kernels, e.g., the Radial Basis function kernel minimizes the distance between inputs (x) and targets (y) based on

$$f(x, y; \gamma) = \exp(-\gamma |x - y|^2)$$

where γ is a user-defined squashing parameter.

There are various SVM packages that are easily obtained in open-source. An easy-to-use one is SVM Light—the package is available at the following URL: `http://svmlight.joachims.org/`. SVM Light is an implementation of Vapnik's Support Vector Machine for the problem of pattern recognition. The algorithm has scalable memory requirements and can handle problems with many thousands of support vectors efficiently. The algorithm proceeds by solving a sequence of optimization problems, lower-bounding the solution using a form of local search. It is based on work by Joachims (1999).

Another program is the University of London SVM. Interestingly, it is known as SVM Dark—evidently people who like hyperplanes have a sense of humor! See `http://www.cs.ucl.ac.uk/staff/M.Sewell/svmdark/`. For a nice list of SVMs, see `http://www.cs.ubc.ca/~murphyk/Software/svm.htm`. In `R`, see the machine learning library `e1071`—the function is, of course, called `svm`.

SVMs are very fast and are quite generally applicable with many types of kernels. Hence, they may also be widely applied in news analytics.

3.5 Word Count Classifiers

The simplest form of classifier is based on counting words that are of *signed* type. Words are the heart of any language inference system, and in a specialized domain, this is even more so. In the words of F.C. Bartlett,

> "Words ... can indicate the qualitative and relational features of a situation in their general aspect just as directly as, and perhaps even more satisfactorily than, they can describe its particular individuality, This is, in fact, what gives to language its intimate relation to thought processes."

To build a word-count classifier a user defines a *lexicon* of special words that relate to the classification problem. For example, if the classifier is categorizing text into optimistic versus pessimistic economic news, then the user may want to create a lexicon of words that are useful in separating the good news from bad. For example, the word "upbeat" might be signed as optimistic, and the word "dismal" may be pessimistic. In my experience, a good lexicon needs about 300–500 words. Domain knowledge is brought to bear in designing a lexicon. Therefore, in contrast to the Bayes classifier, a word-count algorithm is language-dependent.

This algorithm is based on a simple word count of lexical words. If the number of words in a particular category exceeds that of the other categories by some threshold then the text message is categorized to the category with the highest lexical count. The algorithm is of very low complexity, extremely fast, and easy to implement. It delivers a baseline approach to the classification problem.

3.6 Vector Distance Classifier

This algorithm treats each message as a word vector. Therefore, each pre-classified, hand-tagged text message in the training corpus becomes a comparison vector—we call this set the rule set. Each message in the test set is then compared to the rule set and is assigned a classification based on which rule comes closest in vector space.

The angle between the message vector (M) and the vectors in the rule set (S) provides a measure of proximity.

$$\cos(\theta) = \frac{M \cdot S}{||M|| \cdot ||S||}$$

where $||A||$ denotes the norm of vector A. Variations on this theme are made possible by using sets of top-n closest rules, rather than only the closest rule.

Word vectors here are extremely sparse, and the algorithms may be built to take the dot product and norm above very rapidly. This algorithm was used in Das and Chen (2007) and was taken directly from ideas used by search engines. The analogy is almost exact. A search engine essentially indexes pages by representing the text as a word vector. When a search query is presented, the vector distance $cos(\theta) \in (0, 1)$ is computed for the search query with all indexed pages to find the pages with which the angle is the least, i.e., where $cos(\theta)$ is the greatest. Sorting all indexed pages by their angle with the search query delivers the best-match ordered list. Readers will remember in the early days of search engines how the list of search responses also provided a percentage number along with the returned results—these numbers were the same as the value of $cos(\theta)$.

When using the vector distance classifier for news analytics, the classification algorithm takes the new text sample and computes the angle of the message with all the text pages in the indexes training corpus to find the best matches. It then classifies pages with the same tag as the best matches. This classifier is also very easy to implement as it only needs simple linear algebra functions and sorting routines that are widely available in almost any programming environment.

3.7 Discriminant-Based Classifier

All the classifiers discussed above do not weight words differentially in a continuous manner. Either they do not weight them at all, as in the case of the Bayes classifier or the SVM, or they focus on only some words, ignoring the rest, as with the word count classifier. In contrast the discriminant-based classifier weights words based on their discriminant value.

The commonly used tool here is Fisher's discriminant. Various implementations of it, with minor changes in form are used. In the classification area, one of the earliest uses was in the `Bow` algorithm of McCallum (1996), which reports the discriminant values; Chakrabarti, Dom, Agrawal and Raghavan (1998) also use it in their classification framework, as do Das and Chen (2007). We present one version of Fisher's discriminant here.

Let the mean score (average number of times word w appears in a text message of category i) of each term for each category $= \mu_i$, where i indexes category. Let text messages be indexed by j. The number of times word w appears in a message j of category i is denoted m_{ij} . Let n_i be the number of times word w appears in category i. Then the discriminant function might be expressed as:

$$F(w) = \frac{\frac{1}{|C|}\sum_{i \neq k}(\mu_i - \mu_k)^2}{\sum_i \frac{1}{n_i}\sum_j (m_{ij} - \mu_i)^2}$$

It is the ratio of the across-class (class i vs class k) variance to the average of within-class (class $i \in C$) variances. To get some intuition, consider the case we looked at earlier, classifying the economic sentiment as optimistic or pessimistic. If the word "dismal" appears exactly once in text that is pessimistic and never appears in text that is optimistic, then the within-class variation is zero, and the across-class variation is positive. In such a case, where the denominator of the equation above is zero, the word "dismal" is an infinitely-powerful discriminant. It should be given a very large weight in any word-count algorithm.

In Das and Chen (2007) we looked at stock message-board text and determined good discriminants using the Fisher metric. Here are some words that showed high discriminant values (with values alongside) in classifying optimistic versus pessimistic opinions.

```
bad 0.0405
hot 0.0161
hype 0.0089
improve 0.0123
joke 0.0268
jump 0.0106
killed 0.0160
lead 0.0037
like 0.0037
long 0.0162
lose 0.1211
money 0.1537
overvalue 0.0160
own 0.0031
good__n 0.0485
```

The last word in the list ("not good") is an example of a negated word showing a higher discriminant value than the word itself without a negative connotation (recall the discussion of negative tagging earlier in Section 3.2). Also see that the word "bad" has a score of 0.0405, whereas the term "not good" has a higher score of 0.0485. This is an example where the structure and usage of language, not just the meaning of a word, matters.

In another example, using the `Bow` algorithm this time, examining a database of conference calls with analysts, the best 20 discriminant words were:

```
0.030828516377649325 allowing
0.094412331406551059 november
0.044315992292870907 determined
0.225433526011560692 general
0.034682080924855488 seasonality
0.123314065510597301 expanded
0.017341040462427744 rely
0.071290944123314062 counsel
0.044315992292870907 told
0.015414258188824663 easier
0.050096339113680152 drop
0.028901734104046242 synergies
0.025048169556840076 piece
0.021194605009633910 expenditure
0.017341040462427744 requirement
0.090558766859344900 prospects
0.019267822736030827 internationally
0.017341040462427744 proper
0.026974951830443159 derived
0.001926782273603083 invited
```

Not all these words would obviously connote bullishness or bearishness, but some of them certainly do, such as “expanded”, “drop”, “prospects”, etc. Why apparently unrelated words appear as good discriminants is useful to investigate, and may lead to additional insights.

3.8 Adjective-Adverb Classifier

Classifiers may use all the text, as in the Bayes and vector-distance classifiers, or a subset of the text, as in the word-count algorithm. They may also weight words differentially as in discriminant-based word counts. Another way to filter words in a word-count algorithm is to focus on the segments of text that have high emphasis, i.e., in regions around adjectives and adverbs. This is done in Das and Chen (2007) using an adjective-adverb search to determine these regions.

This algorithm is language-dependent. In order to determine the adjectives and adverbs in the text, parsing is required, and calls for the use of a dictionary. The one I have used extensively is the CUVOALD ((Computer Usable Version of the Oxford Advanced Learners Dictionary). It contains parts-of-speech tagging information, and makes the parsing process very simple. There are other sources—a very well-known one is WordNet from http://wordnet.princeton.edu/.

Using these dictionaries, it is easy to build programs that only extract the regions of text around adjectives and adverbs, and then submit these to the other classifiers for analysis and classification. Counting adjectives and adverbs may also be used to score news text for "emphasis" thereby enabling a different qualitative metric of importance for the text.

3.9 Scoring Optimism and Pessimism

A very useful resource for scoring text is the General Inquirer, http://www.wjh.harvard.edu/∼inquirer/, housed at Harvard University. The Inquirer allows the user to assign "flavors" to words so as to score text. In our case, we may be interested in counting optimistic and pessimistic words in text. The Inquirer will do this online if needed, but the dictionary may be downloaded and used offline as well. Words are tagged with attributes that may be easily used to undertake tagged word counts.

Here is a sample of tagged words from the dictionary that gives a flavor of its structure:

```
ABNORMAL H4Lvd Neg Ngtv Vice NEGAFF Modif  |
ABOARD H4Lvd Space PREP LY  |
ABOLITION Lvd TRANS Noun
ABOMINABLE H4 Neg Strng Vice Ovrst Eval IndAdj Modif  |
ABORTIVE Lvd POWOTH POWTOT Modif POLIT
ABOUND H4 Pos Psv Incr IAV SUPV  |
```

The words ABNORMAL and ABOMINABLE have "Neg" tags and the word ABOUND has a "Pos" tag.

Das and Chen (2007) used this dictionary to create an ambiguity score for segmenting and filtering messages by optimism/pessimism in testing news analytical algorithms. They found that algorithms performed better after filtering in less ambiguous text. This ambiguity score is discussed later in Section 3.11.

Tetlock (2007) is the best example of the use of the General Inquirer in finance. Using text from the "Abreast of the Market" column from the Wall Street Journal he undertook a principal components analysis of 77 categories from the GI and constructed a media pessimism score. High pessimism presages lower stock prices, and extreme positive or negative pessimism predicts volatility. Tetlock, Saar-Tsechansky and Macskassay (2008) use news text related to firm fundamentals to show that negative words are useful in predicting earnings and returns. The potential of this tool has yet to be fully realized, and I expect to see a lot more research undertaken using the General Inquirer.

3.10 Voting among Classifiers

In Das and Chen (2007) we introduced a voting classifier. Given the highly ambiguous nature of the text being worked with, reducing the noise is a major concern. Pang, Lee and Vaithyanathan (2002) found that standard machine learning techniques do better than humans at classification. Yet, machine learning methods such as naive Bayes, maximum entropy, and support vector machines do not perform as well on sentiment classification as on traditional topic-based categorization.

To mitigate error, classifiers are first separately applied, and then a majority vote is taken across the classifiers to obtain the final category. This approach improves the signal to noise ratio of the classification algorithm.

3.11 Ambiguity Filters

Suppose we are building a sentiment index from a news feed. As each text message comes in, we apply our algorithms to it and the result is a classification tag. Some messages may be classified very accurately, and others with much lower levels of confidence. Ambiguity-filtering is a process by which we discard messages of high noise and potentially low signal value from inclusion in the aggregate signal (for example, the sentiment index).

One may think of ambiguity-filtering as a sequential voting scheme. Instead of running all classifiers and then looking for a majority vote, we run them sequentially, and discard messages that do not pass the hurdle of more general classifiers, before subjecting them to more particular ones. In the end, we still have a voting scheme. Ambiguity metrics are therefore lexicographic.

In Das and Chen (2007) we developed an ambiguity filter for application prior to our classification algorithms. We applied the General Inquirer to the training data to determine an "optimism" score. We computed this for each category of stock message type, i.e., buy, hold, and sell. For each type, we computed the mean optimism score, amounting to 0.032, 0.026, 0.016, respectively, resulting in the expected rank ordering (the standard deviations around these means are 0.075, 0.069, 0.071, respectively). We then filtered messages in based on how far they were away from the mean in the right direction. For example, for buy messages, we chose for classification only those with one standard-deviation higher than the mean. False positives in classification decline dramatically with the application of this ambiguity filter.

3.12 Network Analytics

We now examine analytic methods that are not based on a single stock or single text message. Instead, we look at methods for connecting news and information across handles, stocks, and time. This is the domain of "context" analysis. By examining the network structure of the information, we may attempt to discern useful patterns in the message stream.

Recall Metcalfe's Law—"The utility of a network is proportional to the square of the number of users." News analytics benefit from the network effect since aggregation greatly improves signal extraction. But in addition, network structure may be used inferentially.

How are networks defined? There are several operational implementations possible. In Das and Sisk (2005), we constructed a network of stocks based on how many common handles posted to pairs of stock message boards. For example, if person `WiseStockGuy` posted messages to both Cisco and IBM in a pre-specified interval, we would increment the connection between those two stocks by one unit. In this manner a network graph of stock linkages is built up. Another approach might be to construct a network graph of connections based on whether someone requested a quote on two stocks at the same time—see Figure 7. The network shows that a tight group of stocks receives all the attention, whereas there are many stocks that are not well-connected to each other.

Network analysis is important because influence and opinion travel rapidly on networks, and the dynamics greatly determine the collective opinion. See DeMarzo, Vayanos and Zwiebel (2003) for an analysis of persuasion bias and social influence on networks. For games on networks, where senders of messages are optimistic, more extreme messages are sent, resulting in lower informativeness—see Admati and Pfleiderer (2001). Hence, news analytics must be designed to be able to take advantage of "word-of-mouth" occurring in web discussions. Word-of-mouth communication leads agents to take actions that are superior than those taken in the absence of it, shown in Ellison and Fudenberg (1995). News metrics enable extraction and analysis of the sociology of networks. Morville (2005) has a neat term for the intersection of information technology and social networks—he calls them "folksonomies".

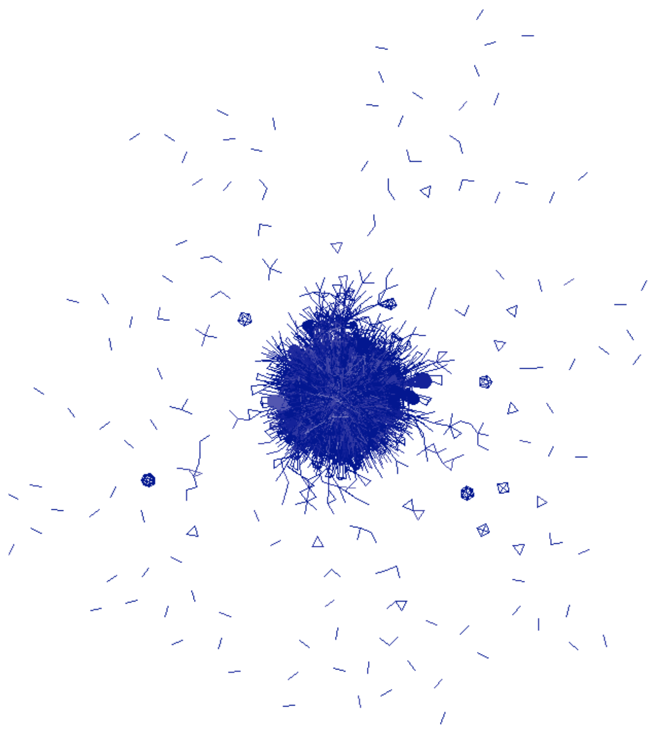


Fig. 7. A rendering of a graph of more than 6,000 stocks for which someone requested a quote from Yahoo! finance. There is an edge between two stocks if someone requested quotes on those stocks at the same time. They are from about 2% of the traffic on Yahoo, on April 1, 2002. Based on rendering software by: Adai A.T., Date S.V., Wieland S., Marcotte E.M. (2004), "Creating a map of protein function with an algorithm for visualizing very large biological networks." *Journal of Molecular Biology*, June 25; 340(1):179-90. The graph is courtesy of Jacob Sisk.

3.13 Centrality

The field of graph theory lends itself to the analysis of networks. There are several news analytics that may be based on the properties of networks. An important and widely-used analytic measure is called "centrality". A node is a network is more central than others if it has more connections to other nodes directly, or indirectly through links to other nodes that are well connected. Centrality has been extensively used in sociology, as in the work by Bonacich (1972), Bonacich (1987).

Centrality is computed as follows. We represent the network of message connections as an adjacency matrix. This matrix is denoted $\mathbf{A} = \{a_{ij}\} \in R^{m \times m}$, a square matrix that contains the connection strength between nodes. If the graph of connections is undirected, then $a_{ij} = a_{ji}$, else if $a_{ij} \neq a_{ji}$, the graph is directed. Let x_i be the influence of node i in the network. Node i exerts influence through connections to other nodes, and we may write the influence of all nodes as the following system of equations:

$$x_i = \sum_{j=1, j \neq i}^{m} a_{ij} x_j$$

This may be written as an eigensystem with the addition of the eigen parameter λ, i.e.,

$$\lambda \, \mathbf{x} = \mathbf{A} \, \mathbf{x}$$

where $\mathbf{x}$ is the vector of influences of all nodes. The principal eigenvector in this system gives the centrality score of all nodes, and highlights which nodes have the most influence.

In Das and Sisk (2005), we computed the centrality scores for all stocks in a network graph where the connection strengths were based on the number of common message posters each pair of stocks had. We found that stocks such as IBM, AOL, Motorola, AMD were central and stocks such as American Express, Abbot Labs, Bristol Myers were not central. Central stocks are more likely to be indicative of the way other stocks may react, since they influence others more than vice-versa; hence, they may be leading indicators of stock market movements. Computing centrality in various news domains is useful to get a sense of what sources of news may be better-tracked than others.

3.14 Communities

News traffic may be analyzed to determine communities. Given a network graph's adjacency matrix, communities are easy to detect using any one of several well-known algorithms. An excellent review of these algorithms is provided by Fortunato (2010).

A widely-used library for graph analysis and community detection is `igraph`. This may be accessed at `http://igraph.sourceforge.net/`. A sample of the ease of use of the `igraph` library using `R` is as follows:

```
#CREATE GRAPH FROM ADJACENCY MATRIX
g = graph.adjacency(adjmat,mode="undirected",weighted=TRUE,diag=FALSE)

#DETECT COMMUNITIES
wtc = walktrap.community(g)
comms = community.to.membership(g,wtc$merges,steps=length(vc_list_connected)/4)
print(comms)
```

```
#DETECT CLUSTERS
clus = clusters(g)
print(clus)
```

The sequence of commands initially creates the network graph from the adjacency matrix (`adjmat`). It then executes the "walktrap" community detection algorithm to find the communities that are then printed out. The `igraph` package also allows for finding clusters as needed.

A community is a cluster of nodes that have many connections between members of the community but few connections outside the community. There are many algorithms that exploit this working definition of a community. For instance, the walktrap algorithm is a randomized one—it detects communities using a random walk on a network. A random walk tends to be trapped in a community because of the number of links between community nodes relative to links across communities. By keeping track of regions of the network where the random walk is trapped, this algorithm is able to detect communities. See the paper by the creators of the algorithm—Pons and Latapy (2006). This is a very recent paper, and resulted in a large performance improvement over existing algorithms.

Communities may then be examined for differences in characteristics to give insights. For example, if we find that stocks in more connected communities tend to be more volatile, then we may want to limit the number of stocks chosen from these communities in a portfolio.

4 Metrics

Developing analytics without metrics is insufficient. It is important to build measures that examine whether the analytics are generating classifications that are statistically significant, economically useful, and stable. For an analytic to be *statistically valid*, it should meet some criterion that signifies classification accuracy and power. Being *economically useful* sets a different bar—does it make money? And *stability* is a double-edged quality: one, does it perform well in-sample and out-of-sample? And two, is the behavior of the algorithm stable across training corpora?

Here, we explore some of the metrics that have been developed, and propose others. No doubt, as the range of analytics grows, so will the range of metrics.

4.1 Confusion Matrix

The confusion matrix is the classic tool for assessing classification accuracy. Given n categories, the matrix is of dimension $n \times n$. The rows relate to the category assigned by the analytic algorithm and the columns refer to the correct category in which the text resides. Each cell (i, j) of the matrix contains the number of text messages that were of type j and were classified as type i. The cells on the diagonal of the confusion matrix state the number of times the algorithm got the classification right. All other cells are instances of classification error. If an algorithm has no classification ability, then the rows and columns of the matrix will be independent of each other. Under this null hypothesis, the statistic that is examined for rejection is as follows:

$$\chi^2[dof = (n-1)^2] = \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{[A(i,j) - E(i,j)]^2}{E(i,j)}$$

where $A(i, j)$ are the actual numbers observed in the confusion matrix, and $E(i, j)$ are the expected numbers, assuming no classification ability under the null. If $T(i)$ represents the total across row i of the confusion matrix, and $T(j)$ the column total, then

$$E(i,j) = \frac{T(i) \times T(j)}{\sum_{i=1}^{n} T(i)} \equiv \frac{T(i) \times T(j)}{\sum_{j=1}^{n} T(j)}$$

The degrees of freedom of the χ^2 statistic is $(n-1)^2$. This statistic is very easy to implement and may be applied to models for any n. A highly significant statistic is evidence of classification ability.

4.2 Accuracy

Algorithm accuracy over a classification scheme is the percentage of text that is correctly classified. This may be done in-sample or out-of-sample. To compute this off the confusion matrix, we calculate

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} A(i,i)}{\sum_{j=1}^{n} T(j)}$$

We should hope that this is at least greater than $1/n$, which is the accuracy level achieved on average from random guessing. In practice, I find that accuracy ratios of 60–70% are reasonable for text that is non-factual and contains poor language and opinions.

4.3 False Positives

Improper classification is worse than a failure to classify. In a 2×2 (two category, $n = 2$) scheme, every off-diagonal element in the confusion matrix is a false positive. When $n > 2$, some classification errors are worse than others. For example in a 3–way buy, hold, sell scheme, where we have stock text for classification, classifying a buy as a sell is worse than classifying it as a hold. In this sense an ordering of categories is useful so that a false classification into a near category is not as bad as a wrong classification into a far (diametrically opposed) category.

The percentage of false positives is a useful metric to work with. It may be calculated as a simple count or as a weighted count (by nearness of wrong category) of false classifications divided by total classifications undertaken.

In our experiments on stock messages in Das and Chen (2007), we found that the false positive rate for the voting scheme classifier was about 10%. This was reduced to below half that number after application of an ambiguity filter (discussed in Section 3.11) based on the General Inquirer.

4.4 Sentiment Error

When many articles of text are classified, an aggregate measure of sentiment may be computed. Aggregation is useful because it allows classification errors to cancel—if a buy was mistaken as a sell, and another sell as a buy, then the aggregate sentiment index is unaffected.

Sentiment error is the percentage difference between the computed aggregate sentiment, and the value we would obtain if there were no classification error. In our experiments this varied from 5-15% across the data sets that we used. Leinweber and Sisk (2010) show that sentiment aggregation gives a better relation between news and stock returns.

4.5 Disagreement

In Das, Martinez-Jerez and Tufano (2005) we introduced a disagreement metric that allows us to gauge the level of conflict in the discussion. Looking at stock text messages, we used the number of signed buys and sells in the day (based on a sentiment model) to determine how much disagreement of opinion there was in the market. The metric is computed as follows:

$$\text{DISAG} = \left|1 - \left|\frac{B-S}{B+S}\right|\right|$$

where B, S are the numbers of classified buys and sells. Note that DISAG is bounded between zero and one. The quality of aggregate sentiment tends to be lower when DISAG is high.

4.6 Correlations

A natural question that arises when examining streaming news is: how well does the sentiment from news correlate with financial time series? Is there predictability? An excellent discussion of these matters is provided in Leinweber and Sisk (2010). They specifically examine investment signals derived from news.

In their paper, they show that there is a significant difference in cumulative excess returns between strong positive sentiment and strong negative sentiment days over prediction horizons of a week or a quarter. Hence, these event studies are based on point-in-time correlation triggers. Their results are robust across countries.

The simplest correlation metrics are visual. In a trading day, we may plot the movement of a stock series, alongside the cumulative sentiment series. The latter is generated by taking all classified 'buys' as $+1$ and 'sells' as -1, and the plot comprises the cumulative total of scores of the messages ('hold' classified messages are scored with value zero). See Figure 8 for one example, where it is easy to see that the sentiment and stock series track each other quite closely. We coin the term "sents" for the units of sentiment.

4.7 Aggregation Performance

As pointed out in Leinweber and Sisk (2010) aggregation of classified news reduces noise and improves signal accuracy. One way to measure this is to look at the correlations of sentiment and stocks for aggregated versus disaggregated data. As an example, I examine daily sentiment for individual stocks and an index created by aggregating sentiment across stocks, i.e., a cross-section of sentiment. This is useful to examine whether sentiment aggregates effectively in the cross-section.

I used all messages posted for 35 stocks that comprise the Morgan Stanley

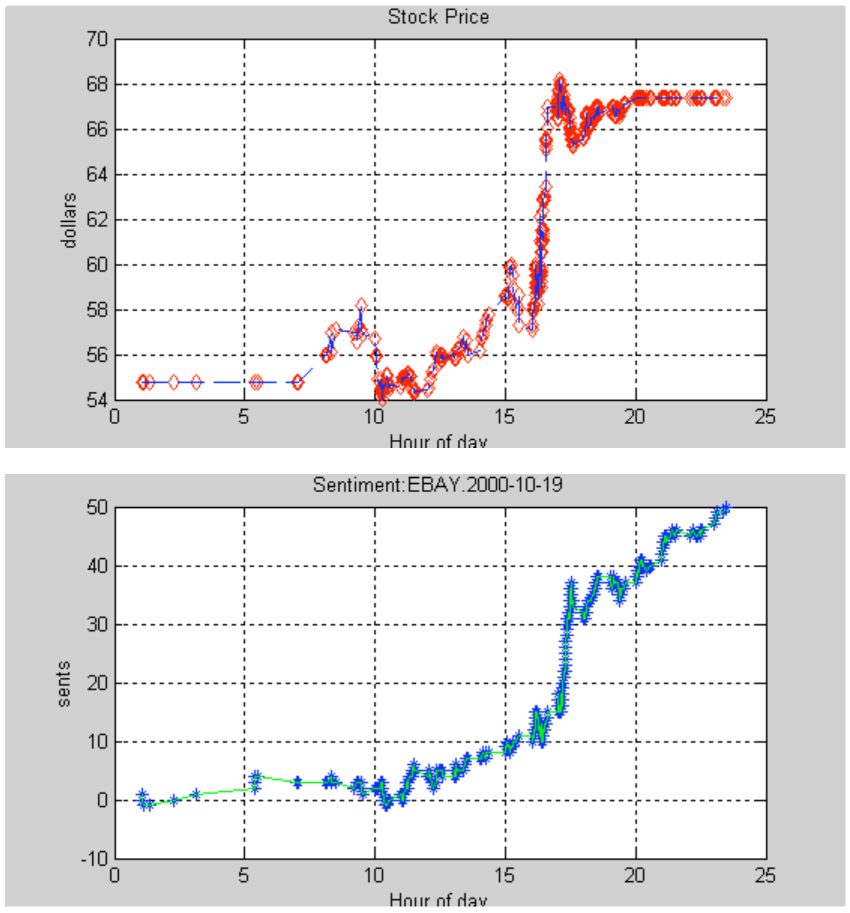


Fig. 8. Plot of stock series (upper graph) versus sentiment series (lower graph). The correlation between the series is high. The plot is based on messages from Yahoo! Finance and is for a single twenty-four hour period.

High-Tech Index (MSH35) for the period June 1 to August 27, 2001. This results in 88 calendar days and 397,625 messages, an average of about 4,500 messages per day. For each day I determine the sentiment and stock return. Daily sentiment uses messages up to 4 pm on each trading day, coinciding with the stock return close.

I also compute the average sentiment index of all 35 stocks, i.e., a proxy for the MSH35 sentiment. The corresponding equally weighted return of 35 stocks is also computed. These two time series permit an examination of the relationship between sentiment and stock returns at the aggregate index level. Table 1 presents the correlations between individual stock returns and sentiment, and between the MSH35 index return and MSH35 sentiment. We notice that there is positive contemporaneous correlation between most stock returns and sentiment. The correlations were sometimes as high as 0.60 (for Lucent), 0.51 (PALM) and 0.49 (DELL). Only six stocks evidenced negative correlations, mostly small in magnitude. The average contemporaneous correlation is 0.188, which suggests that sentiment tracks stock returns in the high-tech sector. (I

Table 1

Correlations of Sentiment and Stock Returns for the MSH35 stocks and the aggregated MSH35 index. Stock returns (STKRET) are computed from close-to-close. We compute correlations using data for 88 days in the months of June, July and August 2001. Return data over the weekend is linearly interpolated, as messages continue to be posted over weekends. Daily sentiment is computed from midnight to close of trading at 4 pm (SENTY4pm).

Ticker	Correlations of SENTY4pm(t) with		
	STKRET(t)	STKRET(t+1)	STKRET(t-1)
ADP	0.086	0.138	-0.062
AMAT	-0.008	-0.049	0.067
AMZN	0.227	0.167	0.161
AOL	0.386	-0.010	0.281
BRCM	0.056	0.167	-0.007
CA	0.023	0.127	0.035
CPQ	0.260	0.161	0.239
CSCO	0.117	0.074	-0.025
DELL	0.493	-0.024	0.011
EDS	-0.017	0.000	-0.078
EMC	0.111	0.010	0.193
ERTS	0.114	-0.223	0.225
HWP	0.315	-0.097	-0.114
IBM	0.071	-0.057	0.146
INTC	0.128	-0.077	-0.007
INTU	-0.124	-0.099	-0.117
JDSU	0.126	0.056	0.047
JNPR	0.416	0.090	-0.137
LU	0.602	0.131	-0.027
MOT	-0.041	-0.014	-0.006
MSFT	0.422	0.084	0.210
MU	0.110	-0.087	0.030
NT	0.320	0.068	0.288
ORCL	0.005	0.056	-0.062
PALM	0.509	0.156	0.085
PMTC	0.080	0.005	-0.030
PSFT	0.244	-0.094	0.270
SCMR	0.240	0.197	0.060
SLR	-0.077	-0.054	-0.158
STM	-0.010	-0.062	0.161
SUNW	0.463	0.176	0.276
TLAB	0.225	0.250	0.283
TXN	0.240	-0.052	0.117
XLNX	0.261	-0.051	-0.217
YHOO	0.202	-0.038	0.222
Average correlation across 35 stocks			
	0.188	0.029	0.067
Correlation between 35 stock index and 35 stock sentiment index			
	0.486	0.178	0.288

also used full-day sentiment instead of only that till trading close and the results are almost the same—the correlations are in fact higher, as sentiment includes reactions to trading after the close).

Average correlations for individual stocks are weaker when one lag (0.067) or lead (0.029) of the stock return are considered. More interesting is the average index of sentiment for all 35 stocks. The contemporaneous correlation of this index to the equally-weighted return index is as high as 0.486. Here, cross-sectional aggregation helps in eliminating some of the idiosyncratic noise, and makes the positive relationship between returns and sentiment salient. This is also reflected in the strong positive correlation of sentiment to lagged stock returns (0.288) and leading returns (0.178). I confirmed the statistical contemporaneous relationship of returns to sentiment by regressing returns on sentiment (T-statistics in brackets):

$$\begin{array}{l} STKRET(t) = \underset{(0.93)}{-0.1791} + \underset{(5.16)}{0.3866} SENTY(t), \quad R^2 = 0.24 \end{array}$$

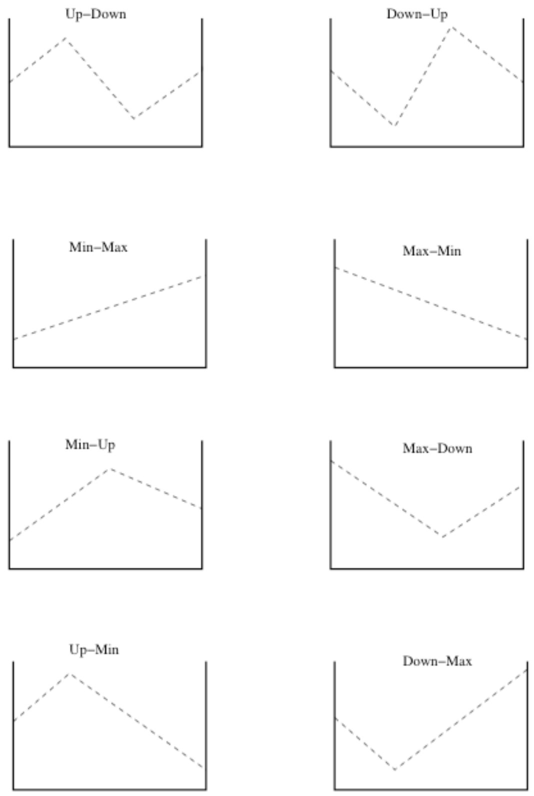
4.8 Phase-Lag Metrics

Correlation across sentiment and return time series is a special case of lead-lag analysis. This may be generalized to looking for pattern correlations. As may be evident from Figure 8, the stock and sentiment plots have patterns. In the figure they appear contemporaneous, though the sentiment series lags the stock series.

A graphical approach to lead-lag analysis is to look for graph patterns across two series and to examine whether we may predict the patterns in one time series with the other. For example, can we use the sentiment series to predict the high point of the stock series, or the low point? In other words, is it possible to use the sentiment data generated from algorithms to pick turning points in stock series? We call this type of graphical examination "phase-lag" analysis.

A simple approach I came up with involves decomposing graphs into eight types—see Figure 9. On the left side of the figure, notice that there are eight patterns of graphs based on the location of four salient graph features: start, end, high, and low points. There are exactly eight possible graph patterns that may be generated from all positions of these four salient points. It is also very easy to write software to take any time series—say, for a trading day—and assign it to one of the patterns, keeping track of the position of the maximum and minimum points. It is then possible to compare two graphs to see which one predicts the other in terms of pattern. For example, does the sentiment series maximum come before that of the stock series? If so, how much earlier

Phase-Lag Analysis



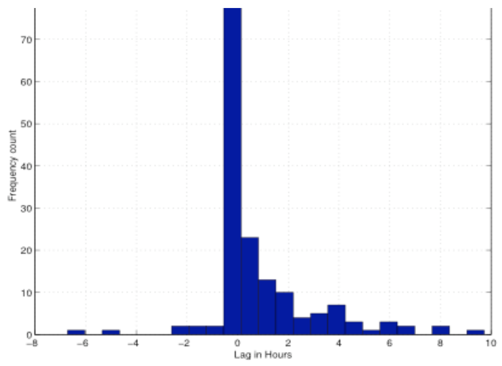


Fig. 9. Phase-lag analysis. The left-side shows the eight canonical graph patterns that are derived from arrangements of the start, end, high, and low points of a time series. The right-side shows the leads and lags of patterns of the stock series versus the sentiment series. A positive value means that the stock series leads the sentiment series.

does it detect the turning point on average?

Using data from several stocks I examined whether the sentiment graph pattern generated from a voting classification algorithm was predictive of stock graph patterns. Phase-lags were examined in intervals of five minutes through the trading day. The histogram of leads and lags is shown on the right-hand side of Figure 9. A positive value denotes that the sentiment series lags the stock series; a negative value signifies that the stock series lags sentiment. It is apparent from the histogram that the sentiment series lags stocks, and is not predictive of stock movements in this case.

4.9 Economic Significance

News analytics may be evaluated using economic yardsticks. Does the algorithm deliver profitable opportunities? Does it help reduce risk?

For example, in Das and Sisk (2005) we formed a network with connections based on commonality of handles in online discussion. We detected communities using a simple rule based on connectedness beyond a chosen threshold level, and separated all stock nodes into either one giant community or into a community of individual singleton nodes. We then examined the properties of portfolios formed from the community versus those formed from the singleton stocks.

We obtained several insights. We calculated the mean returns from an equally-weighted portfolio of the community stocks and an equally-weighted portfolio of singleton stocks. We also calculated the return standard deviations of these portfolios. We did this month-by-month for sixteen months. In fifteen of the sixteen months the mean returns were higher for the community portfolio; the standard deviations were lower in thirteen of the sixteen months. The difference of means was significant for thirteen of those months as well. Hence, community detection based on news traffic leads to identifying a set of stocks that performs vastly better than the rest.

There is much more to be done in this domain of economic metrics for the performance of news analytics. Leinweber and Sisk (2010) have shown that there is exploitable alpha in news streams. The risk management and credit analysis areas also offer economic metrics that may be used to validate news analytics.

5 Discussion

The various techniques and metrics fall into two broad categories: supervised and unsupervised learning methods. Supervised models use well-specified input variables to the machine-learning algorithm, which then emits a classification. One may think of this as a generalized regression model. In unsupervised learning, there are no explicit input variables but latent ones, e.g. cluster analysis. Most of the news analytics we explored relate to supervised learning, such as the various classification algorithms. This is well-trodden research. It is the domain of unsupervised learning, for example, the community detection algorithms and centrality computation, that have been less explored and are potentially areas of greatest potential going forward.

Whether news analytics reside in the broad area of AI or not is under debate. The advent and success of statistical learning theory in real-world applications has moved much of news analytics out of the AI domain into econometrics. There is very little natural language processing (NLP) involved. As future developments shift from text methods to context methods, we may see a return to the AI paradigm.

News analytics will broaden in the toolkit it encompasses. Expect to see greater use of dependency networks and collaborative filtering. We will also see better data visualization techniques such as community views and centrality diagrams. The number of tools keeps on growing. For an almost exhaustive compendium of tools see the book by Koller (2009) titled "Probabilistic Graphical Models."

In the end, news analytics are just sophisticated methods for data mining. For an interesting look at the top ten algorithms in data mining, see Wu, et al. (2008). This paper discusses the top 10 data mining algorithms identified by the IEEE International Conference on Data Mining (ICDM) in December 2006.[3] As algorithms improve in speed, they will expand to automated decision-making, replacing human interaction—as noticed in the marriage of news analytics with automated trading, and eventually, a rebirth of XHAL.

References

A. Admati, and P. Pfleiderer (2001). "Noisytalk.com: Broadcasting Opinions in a Noisy Environment," working paper, Stanford University.

W. Antweiler and M. Frank (2004). "Is all that Talk just Noise? The Information Content of Internet Stock Message Boards," *Journal of Finance*, v59(3), 1259-1295.

W. Antweiler and M. Frank (2005). "The Market Impact of Corporate News Stories," Working paper, University of British Columbia.

P. Bonacich (1972). "Technique for analyzing overlappingmemberships," *Sociological Methodology* 4, 176-185.

P. Bonacich (1987). "Power and centrality: a family of measures," *American Journal of Sociology* 92(5), 1170-1182.

Chakrabarti, S., B. Dom, R. Agrawal, and P. Raghavan. (1998). "Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies," *The VLDB Journal*, Springer-Verlag.

S. Das and M. Chen (2007). "Yahoo for Amazon! Sentiment Extraction from Small Talk on the Web," *Management Science* 53, 1375-1388.

S. Das, A. Martinez-Jerez, and P. Tufano (2005). "eInformation: A Clinical Study of Investor Discussion and Sentiment," *Financial Management* 34(5), 103-137.

S. Das and J. Sisk (2005). "Financial Communities," *Journal of Portfolio Management* 31(4), 112-123.

P. DeMarzo, D. Vayanos, and J. Zwiebel (2003). "Persuasion Bias, Social

[3] These algorithms are: C4.5, k-Means, SVM, Apriori, EM, PageRank, AdaBoost, kNN, Naive Bayes, and CART.

Influence, and Uni-Dimensional Opinions," *Quarterly Journal of Economics* 118, 909-968.

J. Edwards., K. McCurley, and J. Tomlin (2001). "An Adaptive Model for Optimizing Performance of an Incremental Web Crawler," Proceedings WWW10, Hong Kong, 106-113.

G. Ellison, and D. Fudenberg (1995). "Word of Mouth Communication and Social Learning," *Quarterly Journal of Economics* 110, 93-126.

S. Fortunato (2010). "Community Detection in Graphs," *Physics Reports* 486, 75-174.

P. Graham (2004). "Hackers and Painters," O'Reilly Media, Sebastopol, CA.

T. Joachims (1999). "Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning," B. Scholkopf and C. Burges and A. Smola (ed.), MIT-Press.

Koller, D., and M. Sahami (1997). "Hierarchically Classifying Documents using Very Few Words," *International Conference on Machine Learning*, v14, Morgan-Kaufmann, San Mateo, California.

D. Koller (2009). "Probabilistic Graphical Models," MIT Press.

D. Leinweber., and J. Sisk (2010). "Relating News Analytics to Stock Returns," mimeo, Leinweber & Co.

S. Levy (2010). "How Google's Algorithm Rules the Web," *Wired*, March.

F. Li (2006). "Do Stock Market Investors Understand the RiskSentiment of Corporate Annual Reports?" Working paper, University of Michigan.

A. McCallum (1996). "Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering," `http://www.cs.cmu.edu/∼mccallum/bow`.

Mitchell, Tom (1997). "Machine Learning," McGraw-Hill.

L. Mitra., G. Mitra., and D. diBartolomeo (2008). "Equity Portfolio Risk (Volatility) Estimation using Market Information and Sentiment," Working paper, Brunel University.

P. Morville (2005). "Ambient Findability," O'Reilly Press, Sebastopol, CA.

Neal, R.(1996). "Bayesian Learning for Neural-Networks," *Lecture Notes in Statistics*, v118, Springer-Verlag.

B. Pang., L. Lee., and S. Vaithyanathan (2002). "Thumbs Up? Sentiment Classification using Machine Learning Techniques," *Proc. Conference on Empirical Methods in Natural Language Processing* (EMNLP).

P. Pons, M. Latapy (2006). "Computing Communities in Large Networks Using Random Walks," *Journal of Graph Algorithms Applied*, 10(2), 191-218.

M. Porter, (1980). "An Algorithm for Suffix Stripping," *Program* 14(3), 130?137.

Segaran, T (2007). "Programming Collective Intelligence," O'Reilly Media Inc., California.

Smola, A.J., and Scholkopf, B (1998). "A Tutorial on Support Vector Regression," NeuroCOLT2 Technical Report, ESPIRIT Working Group in Neural and Computational Learning II.

P. Tetlock (2007). "Giving Content to Investor Sentiment: The Role of Media in the Stock Market," *Journal of Finance* 62(3), 1139-1168.

P. Tetlock, P. M. Saar-Tsechansky, and S. Macskassay (2008). "More than Words: Quantifying Language to Measure Firm's Fundamentals," *Journal of Finance* 63(3), 1437-1467.

Vapnik, V, and A. Lerner (1963). "Pattern Recognition using Generalized Portrait Method," *Automation and Remote Control*, v24.

Vapnik, V. and Chervonenkis (1964). "On the Uniform Convergence of Relative Frequencies of Events to their Probabilities," *Theory of Probability and its Applications*, v16(2), 264-280.

Vapnik, V (1995). The Nature of Statistical Learning Theory, Springer-Verlag, New York.

Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand and Dan Steinberg, (2008). "Top 10 Algorithms in Data Mining," *Knowledge and Information Systems* 14(1), 1-37.