Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

# Use News to Make Us Rich!

Semester Thesis

Fabian Brun

brunf@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

**Supervisors:**
Philipp Brandes
Prof. Dr. Roger Wattenhofer

April 7, 2013

# Acknowledgements

Many thanks to Trendiction for providing us with the data to do the analysis.

# Abstract

This thesis adds news articles as a data source to an already existing framework for algorithmic trading. It implements a simple machine learning algorithm to perform sentiment analysis on these news articles. This approach of using the news sentiment to predict the market is then evaluated with two different strategies. The results show that the chosen algorithm only reaches parity with the real market performance due to too few learning data. They also indicate that the basic idea of using the news sentiment could lead to profits when used in algorithmic trading.

# Contents

Chapter 1

# Introduction

Algorithmic trading has become an important aspect of the financial market nowadays. It is widely used by main market actors like investment banks, pension funds or hedge funds. According to a recent NY Times article [1], profits from high-speed trading peaked at \$4.9 billion in 2009 (accounting for 73% of the U.S. equity trading volume [2]), which dropped to an estimated \$1.25 billion (and approximately 50% of the U.S. equity trading volume) in 2012.

High-Frequency Trading (HFT) algorithms especially gained media attention after the *May 6, 2010 Flash Crash*, when the Dow Jones dropped by over 600 points in 5 minutes (already being down more than 300 points, totaling in almost 1000 points lost). The following investigations showed that due to an attempt of automatically selling a lot of contracts in a very short period of time, other HFT algorithms kicked in and accelerated the drop by "[beginning to] quickly buy and then resell contracts to each other – generating a 'hot-potato' volume effect as the same positions were passed rapidly back and forth" [3]. Another incident happened on August 1, 2012: Knight Capital lost around \$440 million in about 45 minutes when one of their HFT algorithms went crazy.

## 1.1 Motivation

As shown by the mentioned incidents, the complexity of HFT algorithms is barely within our grasp, which is also why there is a tendency to more regulation in that field. However, algorithmic trading can also be carried out with low-frequency algorithms, e.g., where trading happens only once a day, thus minimizing the risk as well as the negative impact in case errors occur. Such algorithms usually work on mathematical models of the market and use historical stock data to predict the future. This thesis seeks to use news articles as the source for predicting the development of the stock market, similar to what human traders already do, but with an automated and unsupervised algorithm.

## 1.2 Related Work

Sentiment analysis in general is a rather new field in research. Early work applied machine learning techniques at the document level to detect the sentiment in product reviews [4] and movie reviews [5], and was quite successful in doing so. And while both of them only classified with one binary label (either "positive" or "negative"), later approaches tried to extend this. One method was to add more levels to a single feature [6]: Rather than just labelling reviews as "positive"/"negative", give them more quantity ("3 out of 5 stars"). Or, add more features and predict sentiments for different aspects of an entity [7]. For instance, a restaurant could be rated in terms of food, ambience, and service.

Applying sentiment analysis to algorithmic trading is not a new idea. Most of the work has been done by using Twitter as news source. An analysis of almost 10 million tweets from 2008, using neural networks, showed that these tweets can be used to predict the stock market by a few days [8]. A follow-up paper in 2011 suggested that even using a much simpler sentiment analysis method on tweets, reasonable profits can be achieved [9].

While the former research was about the content of individual tweets, another analysis conducted in 2012 focussed on the volume and interaction patterns of tweets. Their findings suggest that the number of tweets is correlated to the trading volume, and to a lesser extent, to the stock price. Additionally, they propose to not only look at the content, but also the relationship between tweets, e.g., distinct topics [10].

Chapter 2

# Background

This chapter describes the basic technologies used in this thesis.

## 2.1 Trading Framework

The basis for this thesis is the trading framework created by Thomas Bürli in 2012 [11]. It uses different layers of abstraction, the relevant ones for implementing a trading strategy being:

- A **Strategy**, the high-level container which signals to buy or sell a certain amount of stocks, built upon
- the **Council**, which gathers information and creates a descision based on the information provided by
- different **Agents**, which provide advice based on some calculations with the underlying data, using
- an **Indicator**, which does the heavy computations on the data.

Figure 2.1 shows how the strategy uses these layers. The indicator is the most important part of any strategy: It takes the stock data and calculates specific measures with this data. An agent then uses these measures to return the advice to either buy, sell or do nothing (based on its calculations) and the confidence it has about that advice. In general, both values are derived from the return value of the indicator directly.

The framework also features a simulator, which uses said strategy to simulate its outcome (based on a starting balance). Furthermore, this outcome can be compared to the actual performance of the overall market (e.g., S&P 500 index).

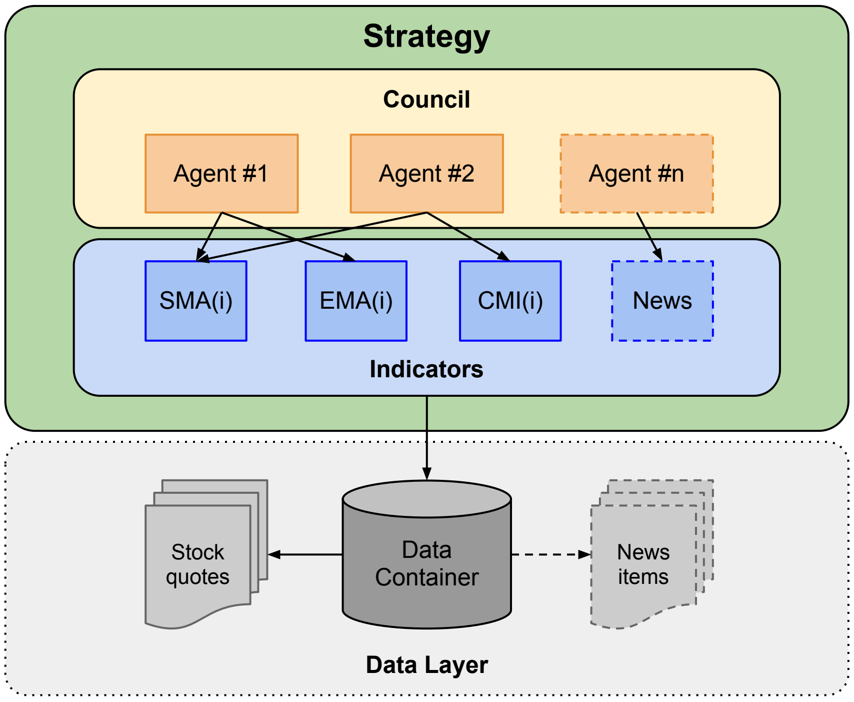

Figure 2.1: Overview about how a Strategy uses the framework layers.

## 2.2 Natural Language Processing

One of the biggest challenges for computers today still is the understanding and interpretation of natural language. A main focus in research nowadays is on machine learning techniques, preferrably unsupervised. Whereas earlier the algorithms were based on large sets of hand crafted rules (extracted from known content), the machine learning approach is capable of adapting to previously unseen patterns. Also with machine learning, the more input data is used to train the algorithm, the more accurate it usually gets – something a hand-written rule set cannot deliver. The exception to this last rule is called *overfitting*, which happens if the statistical model does not describe the underlying relationship but random noise: It memorizes training data instead of learning to generalize from it. Overfitting normally occurs when the model is rather complex.

There are different fields in natural language processing (NLP) like spell checking, translating text from one language into another, composing a summary of given content, or even speech recognition. They often use more basic NLP techniques like part-of-speech tagging (determing the type of a word in the given context, e.g., if it is a verb or an adjective) or word stemming.

### 2.2.1 Machine Learning

Basically, machine learning tries to extract a statistical model out of the initial training data, also called *generalization*. With a good model, such an algorithm can then classify new data based on the trained experience.

For the model to be good, it is necessary that the training data is categorized somehow, e.g., having a "positive" or "negative" meaning. If the categorization can be performed from the testing data itself, the algorithm is called *unsupervised*. An example of this are movie reviews, which have already been categorized by the reviewer (e.g., "3 of 5 stars"). If the categorization is done by hand (e.g., labelling names as "male" or "female"), it is a *supervised* algorithm. The former may seem to be more desirable because of the lack for manual categorization and the easy scalability to big data sets. However, unsupervised categorization is not always possible, and supervised algorithms may result in more accurate categories (and therefore better classification afterwards).

### 2.2.2 Sentiment Analysis

The NLP technique used in this thesis is the sentiment analysis. Its main goal is to understand the subjective meaning of a document, in the simplest form as either *positive* or *negative* (often, there is also the *neutral* label).

A way to do such an analysis is to classify a document based on the words which occur in it. Using a dictionary of "positive" and "negative" words, a simple strategy could be to assign a label using the amount of positive and negative words used in the document. However, this has limited significance, since it does not include the context of these words. In other words, this approach assumes that the words in an article are statistically independent of each other: The mere presence (or absence) of one word is unrelated to the presence (or absence) of other words. The classifier can then simply check for specific features in a new article.

As an enhancment to the above strategy, Turney (2002) [4] restricts the input data set to combinations of adjectives and adverbs (using a part-of-speech tagger) and uses the statistical independance of these words to some reference words (the "semantic association", Church and Hanks 1989 [12]) for the labelling mechanism.

## 2.3 News Aggregator

Aggregating news can be done in different ways, for instance crawling specific news sites. In this thesis a commercial news aggregator service, Trendiction[1],

[1] www.trendiction.com

will be used to get content from all around the internet. They not only crawl classic news sites, but also other sources like blogs and social media sites. There is an application programming interface (API), where previously defined feeds can be queried for the aggregated content. Defining a new feed starts the content aggregation and is as simple as specifying keywords to search for. Trendiction assigns a unique identifier to each feed, later referred to as *feed id*.

### 2.3.1 API Description

Except for the item's *title* and *content*, the Trendiction Feed API delivers some more metadata which are of interest. For example, there are three timestamps associated with every item: the time it was published, the time it got indexed by Trendiction, and the time it was updated (if there was an update). The publishing time is the most important one of them.

Since Trendiction does its own processing of the articles, additional attributes are exposed, already including a sentiment level (negative, neutral, or positive) which will be used to compare with our own approach. Other attributes are:

- the *language* of the content,
- the item's *type* and *subtype* (e.g., *ARTICLE/NEWSSITE*),
- the *fluencylevel* (continuous text with real sentences gets a higher value than simple lists of keywords without punctuation),
- the *spam* and *porn* level (the latter to signal explicit content).

Trendiction also analyzes the sources of each item in terms of *influence* and *visibility* to other web sites as well as the *engagement* of users with that source, and exposes them both as raw and normed values (based on the Gaussian distribution among all URLs).

Chapter 3

# Implementation

To implement a news-based strategy, the framework described in Section 2.1 needs to be enhanced in the following ways:

1. Introduction of a news aggregator script as well as a new data store for this news data.
2. Implementation of some basic machine learning algorithms to perform a simple sentiment analysis ourselves.
3. A new type of indicator which can handle news data (the current ones only support stock quotes).
4. Implementation of a custom agent (using the new indicator) and a custom strategy (using the custom agent).

This chapter gives insight into the implementation of the above items. The programming language used for that is Python[1] (version 2.7).

## 3.1 News Aggregation

A script queries the data source every hour for new content and stores the retrieved data locally.

### 3.1.1 API Client

Trendiction has a Java based API client with graphical user interface (GUI). Since the trading framework is written in Python, and this thesis only needs access to two specific API endpoints, an own lightweight API wrapper has been implemented in the `Trendiction.py` file.

[1] www.python.org

The first endpoint of interest is getting a list of all defined search feeds. This is used to automatically retrieve newly defined search feeds without altering the script's source code.

The second endpoint is specific to one such feed and retrieves up to 500 content items from the API per request. The starting point can be chosen arbitrarily with a timestamp (0 to get the very first items). Every response contains a `next_request` field, which contains a full URL to retrieve more data. If no more data is available, this URL is cached to continue fetching data seamlessly the next time the fetching script is run.

### 3.1.2 Filtering

The search feeds are defined as loosely as possible, normally using only the company name (and possibly its variations) as keyword. The idea is to gather as much data as possible in the search feed, and then filter afterwards on the local client. The filtering is mostly based on the attributes delivered by Trendiction (outlined in Section 2.3.1). The following 5 filters have been implemented:

- a *spam* filter;
- a *language* filter, removing non-English items;[2]
- a *fluency* filter, excluding items which are not considered continous text by Trendiction;
- a *media* filter, excluding multimedia items (e.g., YouTube videos);
- a *porn* filter, removing explicit content.

Applying this filtering also leads to huge savings in terms of space requirements for data storage: About 60% of the aggregated content does not pass the filtering step.

### 3.1.3 Data Caching

The data source used only delivers data for the past seven days. An additional data caching layer has therefore been introduced. This also allows for all other scripts (and especially the framework) to work offline, in particular without the network latency costs when querying for data. The relevant code can be found in the `TrendictionCache.py` file.

To prevent data from being cached multiple times, a hash is calculated on every cache item's content text and the feed id it belongs to. If the hash already

[2]This can also be configured in the search feed definition, and actually has been done after a few days of aggregating; nevertheless, the filter stays in place for older entries.

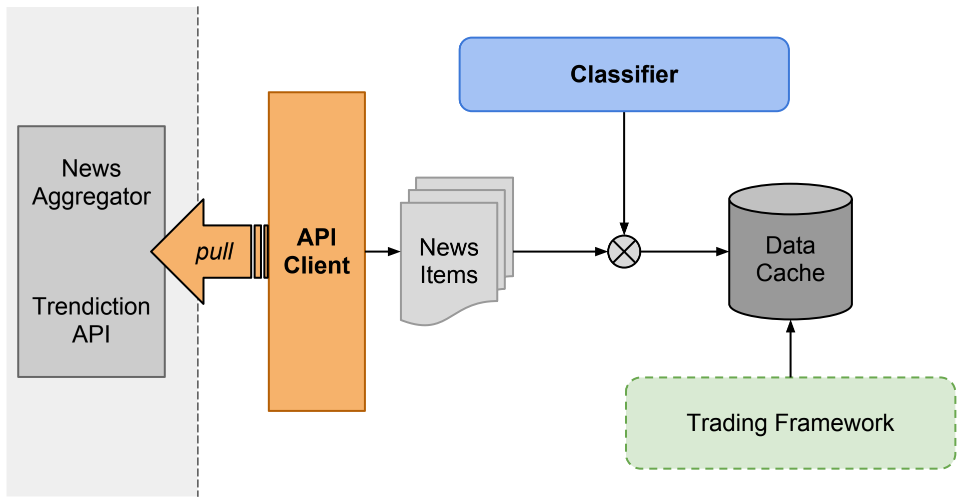

Figure 3.1: Overview about the lower level components for fetching news items.

exists in the cache, the item is skipped. This means that an item can only show up multiple times in the cache if it has been associated to multiple search feeds.

The cache is then split into two overlapping components:

1. Storing the **raw data** as returned by the API. No indexing is performed on that data. This component is not yet used by the framework and therefore completely optional. However, it is useful to examine the filtering as described in Section 3.1.2.

2. Storing the **filtered data**, with the most important fields (like the publishing time) extracted for easier indexing of all the data.

For the cache backend, an SQL database is used, supporting various kinds of dialects (e.g., SQLite[3] or MySQL[4]). Figure 3.1 shows the whole process: First, the data is pulled from the Trendiction servers by the API client. The API client then filters the data, and the remaining news items are classified with the custom sentiment classifier described in Section 3.2.3. Finally, the news items are stored in the cache.

[3]www.sqlite.org
[4]www.mysql.com

## 3.2 Sentiment Analysis

This thesis uses a unsupervised machine learning algorithm. Before this algorithm can classify new articles, it has to be trained on past data, which in turn has to be categorized first. This is primarily done by separate scripts, most of them which do not interact with the rest of the trading framework in any way. The code is based on the Natural Language Toolkit (NLTK) for Python.[5]

### 3.2.1 Categorization

To avoid manual categorization, this thesis assumes that the better a stock performed on a given day, the better the news must have been. And since news have an influence not only on the date they are published, but also a little bit earlier (e.g., “insider” information) and even later, the following unsupervised algorithm is used:

1. For a given feed $F$, load the stock prices for the training period (see Section 4.1 for details).
2. For every day $D$ in that training period:
   (a) Calculate the stock performance in the interval $TF = timeframe(D) = [D-1, D, D+1]$ using the relative change between the close price on day $X \in TF$ and day $X-1$, and the maximum as well as the minimum performance in the training period.
   (b) Normalize the performance, mapping negative values to the interval $[-1, 0)$ and positive values to the interval $(0, 1]$ using:
   $$normalize(day) = \begin{cases} perf(day)/Perf_{max} & \text{if } perf(day) > 0 \\ perf(day)/Perf_{min} & \text{if } perf(day) < 0 \end{cases}$$
   (c) Map the performance to the discrete values $\{-1, 0, 1\}$ with the following mapper function:
   $$label(day) = \begin{cases} +1 & \text{if } normalize(day) >= 0.25 \\ -1 & \text{if } normalize(day) <= -0.25 \\ 0 & \text{else} \end{cases}$$
   The threshold values $\{-0.25, 0.25\}$ are chosen such that the “decisive” labels (-1 and +1) have a slightly higher probability than the “indecisive” label 0.

[5]www.nltk.org

(d) Sum up the mapped values for the days in $TF$; this is the category $C$ for the day $D$:

$$C = category(D) = \sum_{day \in TF} label(day)$$

3. Store the tuple $(F, D, C)$.

This algorithm produces the categories $\{-3, -2, -1, 0, 1, 2, 3\}$. It does not categorize single articles, but days: Every article of feed $F$, published on the same day $D$, gets the same category $C$.

This categorization step can easily be extended to use a more complex algorithm. It is decoupled from the training step, so it can be run independently.

### 3.2.2 Training

The classifier used in this thesis is based on the *NaivesBayesClassifier* class from the NLTK library.[6]

As a first step before training, the 2000 most common words are extracted from all the articles in the training set. To collect them only once, the resulting list of words is cached on disk. Every article, split up into single words, is then fed to the training algorithm:

1. A *feature extractor* looks up which of the 2000 most common words are contained in the article, using a binary approach (`True` if the article contains word $W$, `False` otherwise).
2. The *category* for the (feed, day) combination is retrieved from the database.
3. Both information is then fed to the NLTK classifier to update its internal state.

Since the amount of articles to train the classifier can be quite large, the main training is done feed by feed and day by day. Interrupting the training in a safe way is possible after processing a feed. The internal state of the classifier up to that moment is serialized and persisted to disk, so that the training can be continued afterwards.

[6] http://nltk.googlecode.com/svn/trunk/doc/api/nltk.classify.naivebayes.NaiveBayesClassifier-class.html

### 3.2.3 Classification

The trained classifier can now be loaded from disk anytime it is needed. An article to be classified gets split into words and passed to the feature extractor (the same as used for training). This feature set is consumed by the classifier's `classify` method, which returns the category it deems most appropriate based on the training data.

Since the classification does not change unless the classifier changes, the classification is added to the news item in the data cache. This is done at the time of fetching news items from the news aggregator (as seen in Figure 3.1), and there is an update script for items where the classifications have changed.

As a result, the strategies for the framework only need to query the news data cache, and don't have to classify by themselves, saving time on every simulation run.

## 3.3 Strategies

This section covers the last two items of the list in the beginning of this chapter. Two new strategies are added to the framework:

- The `TrendictionNews` strategy, based on the sentiment value which is calculated by Trendiction, the news aggregator.
- The `SimpleNews` strategy, using the sentiment value from the self-trained classifier.

Both share the same agent implementation and basically the same indicator implementation, since both strategies work in a very similar way on the cached news data. They only differ in the data attributes about the news items they use.

### 3.3.1 News Indicator

The way the indicator layer works is by the singleton pattern.[7] It makes sure that only one instance of a class ever gets created; subsequent instantiations all return the same instance. With the new indicator type, a strategy now has to declare in the beginning which one to use.

Like the original `StockIndicator`, the `NewsIndicator` recieves the stock quotes from the strategy. In addition, it is capable of loading articles from the news data store. Currently two news indicators have been implemented.

[7]http://en.wikipedia.org/wiki/Singleton_pattern

One is available as the `trendiction` method, gathering the sentiment values as delivered by the news aggregator. It combines them per day and returns a table with two columns: The total sentiment and the amount of articles per day.

The second indicator is available as the `sentiment` method. It has the same return value as the `trendiction` indicator, but uses the previously trained classifier to calculate the sentiment value.

### 3.3.2 News Agent

To calculate the advice for a day $D$, the agent combines the sum of sentiments over the two days $\{D-1, D\}$. It then normalizes that value with the maximum aggregated sentiment. The confidence value for every vote is simply the aggregated sentiment normalized with the amount of articles on that day.

$$\begin{aligned} aggregated\_sentiment(day) &= \sum_{article \in \{day-1, day\}} sentiment(article) \\ vote(day) &= \frac{aggregated\_sentiment(day)}{\max(aggregated\_sentiment)} \\ confidence(day) &= \frac{|aggregated\_sentiment(day)|}{articles(day)} \end{aligned}$$

In this formula, $sentiment(article)$ is the getter method to retrieve the sentiment from an article's metadata, and $articles(day)$ the getter method to retrieve the amount of articles on the specified day.

Chapter 4

# Evaluation

For the evaluation of the two sentiment-based strategies discussed in Section 3.3, a total of 12 companies were randomly chosen from the Standard & Poor's 500 (S&P 500) index:[1]

| **Company name** | **Symbol** | **Sector** |
|---|---|---|
| Apple Inc. | AAPL | Information Technolgy |
| C. R. Bard, Inc. | BCR | Health Care Equipment |
| BMC Software, Inc. | BMC | IT Consulting |
| Bemis Company, Inc | BMS | Materials (Paper Packaging) |
| CIGNA, Inc. | CI | Health Care |
| CSX Transportation | CSX | Industrials (Railroads) |
| Duke Energy Corp. | DUK | Energy |
| Invesco Ltd. | IVZ | Financial Services |
| Lockheed Martin | LMT | Aerospace & Defense |
| NetApp, Inc. | NTAP | Information Technolgy |
| Paccar Inc. | PCAR | Industrials (Heavy Equipment) |
| Vornado Realty Trust | VNO | Real Estate Investment |

Table 4.1: The companies chosen to test the algorithms on.

## 4.1 Data Set

The news aggregation period spans 68 days, starting on January 14th 2013 and ending on March 22nd 2013. The Trendiction-based sentiment strategy is evaluated over the whole period, because we only "consume" the sentiment. On the other hand, with the self-trained classifier and applying a 60/40 split, the *training data* covers 43 days (until February 25th) while the *testing data* covers the remaining 25 days (from February 26th). The filtered training data set contains

[1]http://www.standardandpoors.com/indices/sp-500/en/eu/?indexId=spusa-500-usduf–p-us-l–

216’212 items in total. Table A.1 lists the daily amount of articles for every watched stock.

All in all, a training data period of only 68 days is really short. Machine learning algorithms usually work best when trained on data from several months (and not weeks, as in this thesis). This is a major shortcoming for the evaluation of the implemented strategies.

## 4.2 Strategies

Figure 4.3 shows two typical plots of the relative stock price change against the (summed up and normalized) sentiment value as reported by Trendiction. The main difference between both plots is that the upper one (showing NTAP) has higher jumps in the stock price from day to day than the lower one (showing VNO), which makes the sentiment change look smaller (because these values are only in the range $[-1, 1]$).

### 4.2.1 TrendictionNews Strategy

The analysis of this strategy spanned the whole period of the data set, since the sentiment is already calculated when we pull the data from the news aggregator service. Table 4.2 shows the total returns achieved for all stocks under test compared to the stock performance for the same period.

With the Trendiction sentiment, the strategy performed worse on most stocks than the real market did. There are two exceptions: With AAPL and BCR, the

| **Stock** | **Total Return (%)** | **Real Performance (%)** |
|---|---:|---:|
| AAPL | -3.5219 | -8.1105 |
| BCR | -1.3629 | -2.6792 |
| BMC | 7.1505 | 7.6489 |
| BMS | 0.0063 | 13.4476 |
| CI | 0.6215 | 10.9813 |
| CSX | 0.1382 | 17.1053 |
| DUK | 0.0660 | 7.4734 |
| IVZ | 0.0000 | 3.2364 |
| LMT | -1.1470 | -0.8916 |
| NTAP | -4.8449 | 2.3185 |
| PCAR | 0.6356 | 8.7117 |
| VNO | -0.3345 | -0.8949 |
| S&P 500 | | 5.8211 |

Table 4.2: Total return for TrendictionNews strategy over complete data period.

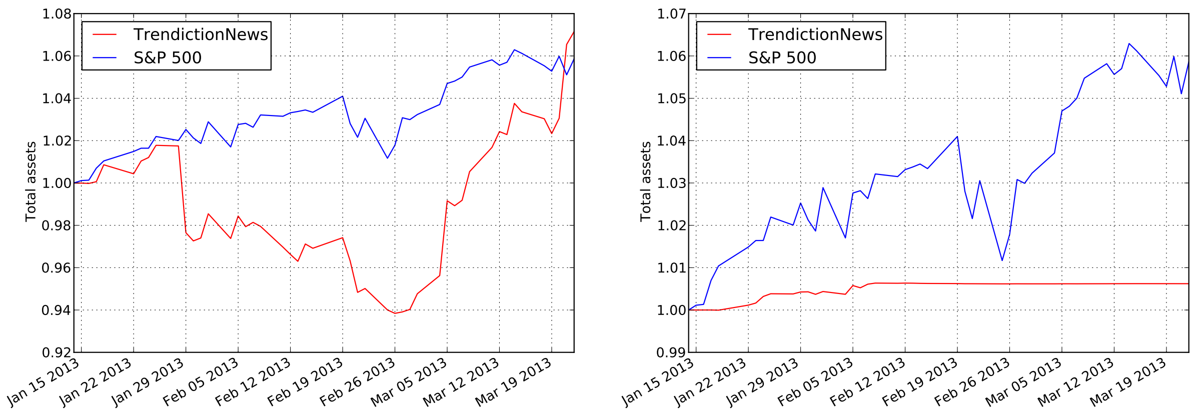

Figure 4.1: Weekly asset return for BMC (left) and CI (right).

strategy lost less than the real market. On the other hand, with either BMS, CI, CSX, NTAP or PCAR the strategy performed far worse. A special case is IVZ, where the news data indicated a non-positive sentiment on all days and therefore the strategy never bought any shares.

Figure 4.1 shows two example asset return graphs (against the S&P 500 as reference). As one can see from the graph for CI, the strategy achieved all of the slightly positive return in the first three weeks. Afterwards, there were no more shares left in the portfolio, and there were no more positive sentiment aggregates, so the strategy did not buy new ones. As a result, the asset return stayed constant.

### 4.2.2 SimpleNews Strategy

Based on the custom sentiment from the self-trained classifier, this strategy performed better overall, as seen in Table 4.3. It could not outperform the real stocks in all cases except one, but was quite close to the real performance most of the time. The notable exception here is AAPL, where the strategy achieved a bigger return than the market. Figure 4.2 shows the weekly asset return for AAPL.

| **Stock** | **Total Return (%)** | **Real Performance (%)** |
|---|---|---|
| AAPL | 5.0339 | 4.0760 |
| BCR | 2.2256 | 2.2561 |
| BMC | 13.8131 | 13.7811 |
| BMS | 4.1186 | 5.9398 |
| CI | 5.0706 | 6.8154 |
| CSX | 4.8342 | 8.0000 |
| DUK | 1.1662 | 1.7728 |
| IVZ | 8.4329 | 8.8156 |
| LMT | 4.5997 | 4.7457 |
| NTAP | -1.3226 | -0.6222 |
| PCAR | 5.4843 | 7.8681 |
| VNO | -0.7708 | -0.5855 |
| S&P 500 | | 4.6403 |

Table 4.3: Total return for SimpleNews strategy over testing data period.

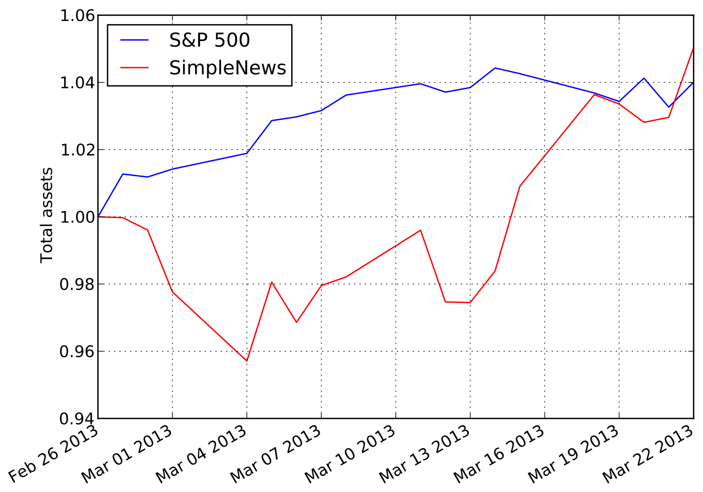

Figure 4.2: Weekly asset return for AAPL.

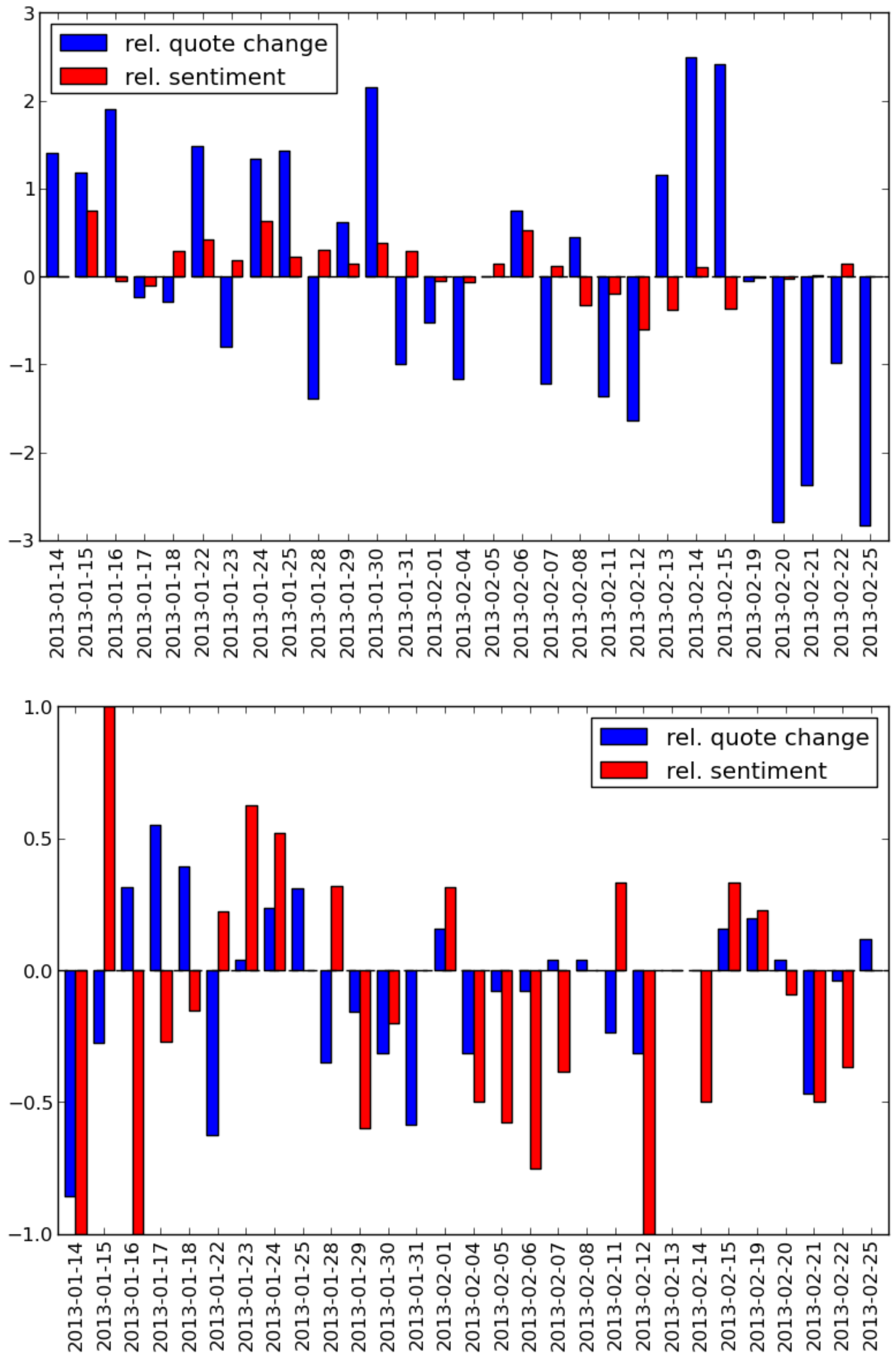

Figure 4.3: Stock price performance and reported sentiment values for NTAP (top) and VNO (bottom).

# Chapter 5
# Conclusion

The goal of this thesis was to introduce news articles as a new data source to the framework, which has been accomplished in a way which allows for further extensions. The example strategies built upon this new data source performed not as expected, especially the one based on the pre-calculated sentiment.

The main problem here seems to be the short period of time in which data has been gathered (only ten weeks). More data would have allowed for a more thorough model of the relationship between news sentiments and the stock market. The simple assumption of “good news equals increasing stock prices” is not true when looking at general news from all over the internet (not focussing on specialized data sources).

## 5.1 Outlook

There is now a foundation for using news to predict the stock market with the framework, but there is room for more.

All in all it would be interesting to look at the performance of both strategies with more data at hand (e.g., at least six month of training data). Also, the implemented strategies are both very simple, so one addition could be to implement a more complex strategy. For example, the custom sentiment analysis could be enhanced either with other machine learning approaches or by using different news aggregators (e.g., focus on more relevant articles). One could also combine different sources to power the strategy.

# Bibliography

[1] Popper, N.: High-speed trading no longer hurtling forward. New York Times (October 2012)

[2] Iati, R.: The real story of trading software espionage. Advanced Trading (June 2009)

[3] SEC, CFTC: Findings regarding the market events of may 6, 2010 (September 2010)

[4] Turney, P.D.: Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews. In: Proceedings of the Association for Computational Linguistics. (2002)

[5] Pang, B., Lee, L., Vaithyanathan, S.: Thumbs up? sentiment classification using machine learning techniques. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing. (2005)

[6] Pang, B., Lee, L.: Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In: Proceedings of the Association for Computational Linguistics. (2005)

[7] Snyder, B., Barzilay, R.: Multiple aspect ranking using the good grief algorithm. In: Proceedings of the Joint Human Language Technology/North American Chapter of the ACL Conference. (2007)

[8] Bollen, J., Mao, H., Zeng, X.J.: Twitter mood predicts the stock market. CoRR **abs/1010.3003** (2010)

[9] Lazer, R.C..M.: Sentiment analysis of twitter feeds for the prediction of stock market movement (2011)

[10] Ruiz, E.J., Hristidis, V., Castillo, C., Gionis, A., Jaimes, A.: Correlating financial time series with micro-blogging activity (2012)

[11] Bürli, T.: Make us rich! (August 2012)

[12] Church, K.W., Hanks, P.: Word association norms, mutual information, and lexicography. In: Proceedings of the 27th Annual Conference of the ACL. (1989)

[13] Wischmann, S.: Make us richer! (March 2013)

Chapter 6

# Figures and Tables

## List of Figures

## List of Tables

#  Appendix A
# Appendix

## A.1 Framework Installation

This section explains how to install the framework in a Ubuntu 12.10 system. Prior to setting up the framework, some system packages have to be installed first. The following command makes sure all prerequisites are fulfilled:

```
sudo apt-get install python2.7 python-virtualenv python-pip \
libfreetype6-dev libpng12-dev
```

Python can be used in a so called "virtual environment".[1] This has the advantage that you do not clutter up the system's Python package index as well as no specific rights are needed to install: Everything is installed locally into a directory the user can write to. Use the following commands to install the framework into such a virtualenv, sitting in the `moneymaker` subdirectory in the user's home directory (replace `REPOSITORY` with the URL to the source code repository):

```
mkdir ~/moneymaker && cd ~/moneymaker
virtualenv --no-site-packages env
source env/bin/activate
git clone REPOSITORY src
cd src
pip install distribute -U
pip install numpy
python setup.py develop
```

[1] www.virtualenv.org

## A.2 How to Use

The general framework usage is the same as described in [13] (a follow-up thesis to the original one by Thomas Bürli [11]). However, there are now some other scripts needed to perform a sentiment analysis. The preferred way to use these scripts is by the *IPython* shell. The commands below assume you have set up the code as described in Section A.1.

An important file is `Sentiment.py`, especially the variables `START_DATE` and `STOP_DATE` defined at the top of the file. They influence a lot of other places in the code, e.g. the daterange from which the indicator fetches news data from the cache, or the training data period for the classifier.

### A.2.1 News Fetching

The `Trendiction.py` file, when run as a python script, pulls all new data from the Trendiction web servers:

```
~/env/bin/python ~/src/moneymaker/Trendiction.py
```

You may want to run that command as a cronjob (e.g., every hour) to continously pull news data.

When you define new feeds at the Trendiction admin page, they will automatically be picked up by the pull script. If such a new feed should not be pulled, add the Trendiction feed id to the `BLACKLIST` list in the `TrendictionAPI` class. There is also a list `TEST_SET`, which contains the feed ids to test the algorithms on. This list is not automatically updated when a new feed has been defined.

### A.2.2 Sentiment Analysis

For the classifier to be trained, the training set must be categorized. This can be done in an IPython shell like this:

```
import datetime as dt
from moneymaker import Sentiment
start = dt.date(2013, 04, 01)  # chose whatever you like
Sentiment.categorize(start)
```

These commands runs the categorizer loop over all data between the defined `start` date and the `STOP_DATE` (this can take a while).

To train the classifier, you must first calculate the word frequency distribution, then you can train it iteratively. In the IPython shell:

```
from moneymaker import Trendiction, Sentiment
feeds = Trendiction.TrendictionAPI().get_feeds(test_set=True)
Sentiment.freq_dist(feeds)  # stores result on disk
Sentiment.train_continously(feeds)  # stores result on disk
classifier = Sentiment.get_classifier()
```

The `train_continously` step may take a really long time. It goes through all the news data from the feeds listed in the `TEST_SET` variable. Interrupting this step (e.g. through `CTRL+C`) will result in the current feed to be processed to the end, and then stopping the script. When restarting it, make sure to only pass in the feeds which have not yet been processed.

When the classifier is ready, the news fetching script will automatically classify new content (see Figure 3.1).

If the classifier gets retrained later, the classifications have to be updated. There is also a script which does the heavy lifting. Using the IPython shell again:

```
from moneymaker import scripts, Trendiction
feeds = Trendiction.TrendictionAPI().get_feeds(test_set=True)
scripts.update_classifications(feeds)
```

## A.3 Article Count

Table A.1 lists the amount of news items (per day and feed, after filtering) contained in the training set.

| | AAPL | BCR | BMC | BMS | CI | CSX | DUK | IVZ | LMT | NTAP | PCAR | VNO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2013-01-14 | 513 | 20 | 20 | 7 | 5 | 4 | 3 | 2 | 10 | 2 | 2 | 1 |
| 2013-01-15 | 528 | 29 | 26 | 7 | 4 | 6 | 8 | 4 | 14 | 8 | 1 | 1 |
| 2013-01-16 | 524 | 37 | 55 | 24 | 5 | 8 | 12 | 6 | 26 | 18 | 2 | 1 |
| 2013-01-17 | 467 | 212 | 317 | 167 | 45 | 41 | 82 | 74 | 216 | 58 | 13 | 11 |
| 2013-01-18 | 508 | 326 | 225 | 228 | 50 | 80 | 82 | 80 | 283 | 78 | 38 | 13 |
| 2013-01-19 | 382 | 245 | 150 | 84 | 60 | 39 | 33 | 28 | 150 | 28 | 15 | 4 |
| 2013-01-20 | 363 | 148 | 113 | 112 | 26 | 68 | 44 | 48 | 137 | 16 | 11 | 1 |
| 2013-01-21 | 617 | 249 | 226 | 163 | 46 | 71 | 74 | 88 | 234 | 61 | 14 | 5 |
| 2013-01-22 | 591 | 253 | 280 | 205 | 51 | 328 | 81 | 78 | 250 | 124 | 15 | 8 |
| 2013-01-23 | 638 | 279 | 313 | 154 | 65 | 311 | 84 | 91 | 301 | 126 | 20 | 8 |
| 2013-01-24 | 621 | 251 | 218 | 173 | 44 | 110 | 68 | 78 | 301 | 90 | 16 | 25 |
| 2013-01-25 | 640 | 301 | 142 | 136 | 50 | 65 | 78 | 89 | 192 | 41 | 16 | 10 |
| 2013-01-26 | 405 | 197 | 132 | 135 | 28 | 58 | 54 | 36 | 199 | 25 | 11 | 5 |
| 2013-01-27 | 441 | 233 | 122 | 74 | 32 | 43 | 69 | 56 | 152 | 21 | 5 | 2 |
| 2013-01-28 | 778 | 237 | 281 | 178 | 65 | 110 | 101 | 88 | 304 | 100 | 9 | 21 |
| 2013-01-29 | 737 | 233 | 400 | 158 | 65 | 94 | 109 | 90 | 277 | 112 | 12 | 5 |
| 2013-01-30 | 767 | 216 | 262 | 157 | 54 | 68 | 164 | 53 | 248 | 73 | 11 | 5 |
| 2013-01-31 | 794 | 291 | 244 | 147 | 38 | 62 | 196 | 85 | 269 | 75 | 37 | 6 |
| 2013-02-01 | 1869 | 302 | 242 | 133 | 40 | 68 | 158 | 86 | 259 | 79 | 34 | 19 |
| 2013-02-02 | 609 | 176 | 158 | 94 | 34 | 50 | 57 | 18 | 172 | 20 | 13 | 9 |
| 2013-02-03 | 884 | 206 | 136 | 69 | 16 | 71 | 48 | 29 | 161 | 12 | 21 | 3 |
| 2013-02-04 | 1017 | 312 | 276 | 193 | 121 | 76 | 115 | 64 | 218 | 77 | 23 | 34 |
| 2013-02-05 | 1207 | 320 | 308 | 165 | 91 | 55 | 271 | 101 | 252 | 98 | 22 | 26 |
| 2013-02-06 | 1581 | 274 | 322 | 156 | 34 | 103 | 168 | 70 | 322 | 118 | 20 | 16 |
| 2013-02-07 | 5595 | 190 | 273 | 132 | 159 | 72 | 116 | 105 | 293 | 104 | 15 | 13 |
| 2013-02-08 | 3783 | 61 | 59 | 50 | 17 | 18 | 21 | 17 | 43 | 12 | 4 | 2 |

Table A.1: The article count for each company and day.

| | AAPL | BCR | BMC | BMS | CI | CSX | DUK | IVZ | LMT | NTAP | PCAR | VNO |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2013-02-09 | 686 | 12 | 5 | 20 | 2 | 3 | 4 | 3 | 15 | 0 | 0 | 0 |
| 2013-02-10 | 678 | 16 | 8 | 13 | 0 | 5 | 6 | 2 | 13 | 2 | 5 | 1 |
| 2013-02-11 | 1210 | 44 | 23 | 15 | 2 | 3 | 8 | 11 | 20 | 3 | 1 | 2 |
| 2013-02-12 | 1417 | 70 | 29 | 19 | 6 | 5 | 9 | 7 | 21 | 5 | 1 | 1 |
| 2013-02-13 | 2092 | 66 | 81 | 25 | 5 | 10 | 18 | 41 | 26 | 16 | 5 | 4 |
| 2013-02-14 | 1679 | 273 | 99 | 106 | 64 | 45 | 17 | 12 | 22 | 9 | 15 | 12 |
| 2013-02-15 | 3343 | 160 | 147 | 105 | 13 | 44 | 19 | 43 | 43 | 11 | 10 | 3 |
| 2013-02-16 | 6209 | 188 | 148 | 119 | 28 | 33 | 63 | 26 | 108 | 13 | 9 | 2 |
| 2013-02-17 | 8560 | 180 | 130 | 123 | 14 | 39 | 48 | 44 | 150 | 19 | 8 | 1 |
| 2013-02-18 | 13474 | 261 | 195 | 198 | 37 | 45 | 49 | 66 | 272 | 42 | 7 | 6 |
| 2013-02-19 | 16873 | 313 | 315 | 273 | 127 | 54 | 101 | 77 | 316 | 147 | 14 | 13 |
| 2013-02-20 | 17990 | 261 | 261 | 438 | 80 | 67 | 90 | 111 | 314 | 144 | 12 | 11 |
| 2013-02-21 | 16220 | 319 | 316 | 293 | 44 | 75 | 102 | 125 | 330 | 126 | 15 | 12 |
| 2013-02-22 | 16650 | 329 | 222 | 176 | 27 | 86 | 84 | 82 | 365 | 115 | 13 | 19 |
| 2013-02-23 | 12219 | 221 | 148 | 96 | 23 | 42 | 41 | 39 | 250 | 21 | 13 | 11 |
| 2013-02-24 | 10015 | 232 | 79 | 98 | 27 | 43 | 43 | 31 | 271 | 27 | 15 | 5 |
| 2013-02-25 | 16322 | 307 | 237 | 166 | 25 | 81 | 115 | 141 | 379 | 69 | 19 | 11 |

Table A.1: The article count for each company and day.

## A.4 Original Problem

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Prof. R. Wattenhofer

### Lab/BA/SA/Group: Use News to Make Us Rich!

#### Motivation and Informal Description

In the recent past the algorithmic trading has seen enormous growth and is a good place to make lots of money. It is now responsible for more than 70% of the trades in the US. A very important subclass are the high frequency trading (HFT) algorithms. These algorithms usually hold stocks or certificates only for a brief time, sometimes only for a few seconds or even milliseconds and earn money by making thousands of trades a day. But since these algorithms increase the volatility of the market, they are becoming the target of a financial regulations which would destroy that business model.

Therefore, we want to return to systematic algorithmic trading to get rich. We already developed a simple framework and implemented some basic strategies with it. We want you to extend this framework. But we want more than simple algorithms which just use the past stock data, we want to be smarter. Your job will be to use the information given by a news crawling service to react more accurately than other algorithmic traders and thus, to make us rich.

#### Requirements

Good programming skills (preferably in Python) and a genuine interest in the financial markets. The student(s) should be able to work independently on this topic!

**Interested? Please contact us for more details!**

#### Contact

- Philipp Brandes: philipp.brandes@tik.ee.ethz.ch, ETZ G64.2