4\_Twitter\_Mining

Nicholas Mitchell

March 13, 2016

Contents

1 Social media data 1

1.1 Google Trends . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

1.2 Facebook . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

1.3 Twitter . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

2 Twitter Mining 3

2.1 Why use Twitter? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2.2 Requirements on data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2.3 Sources of Twitter data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2.3.1 Twitter API . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

2.3.2 Third Party Providers . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2.3.3 Twitter Advanced Search . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

3 Constructing a web-scraper 4

3.1 What is a web-scraper? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

3.1.1 Types of web-scraping . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.2 How does our web-scraper work? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

3.3 Stability considerations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

3.4 Parsing the HTML code . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

3.5 How is the HTML code parsed? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

3.6 Pre-processing tweet text . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

3.7 Final output for sentiment analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

4 Time-series analysis - N versus P 9

NOTE:

Any time you see a † symbol, it means there is a link attached that you can click on to go to a website. This

is to help people reading a printed version later on, as the links will all be in an appendix.

1 Social media data

There are many possible sources of social media data that could be incorporated into a statistical model,

and naturally it is the Big Three: Google, Facebook and Twitter, who spring to mind. While the means

exist to obtain data from all three, there are also limitations that apply to each.

1

1.1 Google Trends

Google makes a lot of data freely available, for example the number of times a given word or phrase was

’Googled’. The search engine does, however, apply certain filter and pre-processing steps to the data before

making it available. What remains is a great tool for making comparisons between terms, plotting their

relative popularity against one another over a long time period, an example of which is shown in FigX.

Figure 1: A placeholder for the real image

There are two issues that make Google Trends data difficult (but not impossible) to use in the context

of this study. The first issue is that the frequency of the data is (at the time of writing) limited to weekly

aggregates for timelines longer than three months. This means a method of interpolation would have to be

implemented before the data could be combined with daily financial market data over this study’s desired

time-line of two or more years. Daily data is available for time-lines shorter that three months, which leads

nicely on to the second issue. The pre-processing of the data is not transparent; the exact methods used

are not published and so any interpretation could perhaps be fallacious. The data is clearly normalised, the

maximum ’popularity’ in each extracted data set being 100 and so naïve attempts to stitch many threemonth

data sets together - such as linear combinations - would be in vain, as the final time-line could not be

considered homogeneous in scale. With a different objective in mind, the Google search data does present

an interesting case. Hamid and Heiden (2014) were able to show how Google search volumes could be used

to increase forecasting accuracy for market phases of relatively high volatility1. In the context of this study,

however, the task must be deferred as discussed in Future extensions.

1.2 Facebook

To use Facebook as a source of data, it is necessary to create a special account for software developers (which

is free). The downside, however, is that only the publicly available information of your own friends who also

have a developer account may freely be obtained. This is a large limitation, as it would significantly reduce

the amount of data available and narrows down the pool of social media data specifically to a biased subset of

users, i.e. data for people who are involved in software development and data mining. This is unfortunately

not the target group of this study and so rules out the use of data Facebook.

1.3 Twitter

The third option is Twitter, which has been extensively mined for its large flow of information [References

to several examples]. The following section explains why Twitter is such a popular choice as a source of

social media data, justifying its selection for this study. The current best practices of extracting data are

then summarised, along with a brief explanation of the procedure defined by this study.

1This is an interesting direction that could potentially be built upon with the Twitter data accumulated for this study

2

2 Twitter Mining

2.1 Why use Twitter?

The social media data used for sentiment analysis (see ChapterX) was sourced exclusively from the online

social media platform Twittery. The first post (in Twitter terminology, a tweet) was made in March 2006

via short message service (SMS), the entire service running off of a single laptop. In the ensuing months the

platform began its ascent to popularity, steadily expanding its user-base after its official launch in summer

2006. Not only individuals, but everyone from news companies and sports teams to artists and presidents

use Twitter to update their follows, with the potential to reach anybody with an internet connection. A

study into the monthly count of unique visitors that top digital media properties receive showed that Twitter

was ranked 14th with 118,000 unique visitors 2 in January 2016 alone. Perhaps somewhat unsurprisingly,

the top two spots for the same time period were occupied by Google and Facebook, each with ca. 245,000

and ca. 207,000 unique visitors, respectively.

There are several reasons why Twitter data is an attractive candidate as an explanatory variable in a

study such as this one. First and foremost, it is content that is created on a continuous basis with almost

no filter 3. In short, users may post their thoughts regarding any topic, at any time, for anyone to read.

This makes the data an excellent tool for capturing the sentiments and emotions across extremely large

demographics of users, in real time.

2.2 Requirements on data

At the time of writing there are several clear ways in which it is possible to obtain Twitter data, each outlined

in the section Sources of Twitter datay along with their corresponding benefits and drawbacks. When

considering which route to take, it should be kept in mind exactly what kind of analysis will ultimately be

performed on the data. The context of this study necessitates that the information obtained fulfils certain

criteria. Regarding the data of each individual tweet, there are two criteria:

Criterion 1: each tweet must contain the tweet text

Criterion 2: each tweet must contain a timestamp

The last requirements concerns the population of tweets obtained:

Criterion 3: the collective corpus of tweets must span a timeline of at least two years

In order to perform sentiment analysis on the Twitter data, it is imperative that the text string is obtained,

fulfilling Criterion 1. If only meta-data were to be received, e.g. the creation time and point of origin of a

tweet, sentiment analysis would be impossible. Criterion 2 ensures that the Twitter data (and therefore the

results of the sentiment analysis) can be reliably aggregated into daily data. This allows for coherent usage

with daily financial market data. Although the timeline specified by Criterion 3 may appear somewhat

arbitrary (and it is!), a minimum timeline of several years is commonly desired for time-series analysis of

financial markets. For discussion as to why this is the case, please refer to sectionX (’Time series analysis -

N versus P’).

2.3 Sources of Twitter data

2.3.1 Twitter API

Twitter offers an application programme interface (API) to allow programmatic connections to its databases.

This is commonly achieved using languages such as Python, JavaScript and R, but can be implemented using

any language capable of establishing an API connection. The service is free, requiring only that users create

2Statistics compiled by comScorey. Unique signifies a user with identifiable IP address. Total numbers of hits, i.e. simple page

visits, including those from machines and referals over other sites reach around 3 billion per month according to SimilarWeby.

3The only limit imposed on users is the 140-character limit placed on each tweet. Twitter’s actual definition is slightly more

detailedy.

3

a developer account, obtaining secure identification methods using a token system. Furthermore, the tweet

data is very clean and there are many tools4 already available that parse and display that data.

There are two restrictions placed on the API. The first is to safeguard the Twitter servers from being

overrun, namely that each user may make only a certain number of requestsy in a given time-frame, which

translates into a limit of approximately 10,000 tweets in a fifteen-minute timeframe. The second restriction

limits the API’s reach into the past to approximately seven days. This means that it is impossible to collect

and create a time-series of the required length for this study. While it is possible to implement and automate

a script to collect tweets at a given frequency5, one would have to still wait e.g. two years minus seven days

to obtain a time-series that is two years in length. For this reason, the Twitter API methodology was not a

feasible option for this study.

2.3.2 Third Party Providers

It is possible to gain access to the complete Twitter archives, spanning back to Twitter’s inception. This is

facilitated by a third party company called Union Metrics via their Echo product liney. There are interactive

analytics tools built in to the console, which allow the slicing and drilling of the entire database with visual

representations. This is aimed at commercial users needing to make strategic marketing decisions, rather

than perform statistical analysis or make quantitative forecasts.

Although the product is extensive and offers many features, it has two drawbacks. Firstly it is not a

free service; requiring a corporate level monthly subscription. Secondly, the offering is not optimised for

independent data analysis, as restrictions on exporting the data would impede full usage of the data within

alternative software packages. Both constraints rule out this is a valid otion for this study, with the second

constraint being particularly large for any parties interested in quantitative forecasting.

2.3.3 Twitter Advanced Search

The Twitter Advanced Searchy (TAS) web interface allows any user to search for tweets in any time period,

displaying tweets that match a given search term. The tweets are displayed in reverse chronological order

(the most recent tweet is at the top of the webpage) and each tweet is displayed with its key information.

The HTML code being rendered, however, holds additional information, matching all that is available via

the API and third party options. There isn’t only the tweet text, username and timestamp, but rather a

whole host of meta data including e.g. the number of times the tweet has since been retweeted (re-posted or

shared by another user) or favourited (marked as a favourite by another user) and even the longitude-latitude

coordinates of the user at the time of posting6. SectionX goes into more detail about how this data may be

located and extracted from the HTML code.

The web interface is free to use, contains the entire Twitter archive and also, being Twitter’s advanced

search, allows for filtering of tweets beyond a date range. For example, the natural language of the tweets

(English, Portuguese, etc.) can be used as a filter as well as a longitude-latitude coordinates from which a

tweet was posted. Tweets for individual users or containing specific hashtags7 can also be selected. This

study uses solely the common search function, returning all tweets that contain the user-specified word(s).

The single disadvantage of this approach is that it involves using an interactive interface, i.e. it is not

designed to be utilised programmatically. This created significant challenges within the scope of this study,

including the development of a customised web-scraper, as shall be explained in the following section.

3 Constructing a web-scraper

3.1 What is a web-scraper?

To explain this, a good analogy between the internet and an encyclopaedia can be used. Imagine we would

like to find all the pages in the encyclopaedia that contain information regarding a topic of interested, for

4The most useful implementation in R is currently the twitteR package, which is a one-stop-shop for cleanly extracting

tweets, ready for analysis with common R functions.

5The author has already implemented such as system, available on request.

6The coordinates accuracy is approximately a 1.5 km radius, which should guarantee some level of privacy.

7Hashtags provide and unmoderated way to help to link tweets from different users and locales by theme.

4

example "chocolate". We would look in the index for our search term and find all topics involving chocolate

to be listed with their page and section numbers. The term given to such a mechanism is web-crawlery

and is (simplistically speaking) approximately what search engines such as Google, Yahoo and Bing carry

out each time somebody uses their search functions. They look all the pages in their encyclopaedia and the

returned search results are those (web-)pages containing the word "chocolate"8. The data that this study is

interested in, however, is not the page number i.e. the internet address of certain information, but rather

the contents held at those addresses.

Assuming the information provided by a web-crawler is already known (in our case the internet

address of TAS, using our analogy, we visit the specified page and make a copy of all the information that

is stored there. Just as one could write out a copy of any information visible in an encyclopaedia, it is

possible to make a copy of all visible information (plus additional background meta data) presented on a

website. This is because, in order for the website to be displayed in a browser, all the required information

must first be transferred (downloaded) to the local device and stored in the form of HTML code, which the

browser then interprets and renders. It is then this HTML code that is copied, or scraped, leading to the

term web-scraper 9.

In order to obtain all required information from TAS, the first major objective of this study was to create

a web-scraper that was able to visit the TAS interface, manipulate the webpage and make a copy of the

underlying information i.e. the HTML code.

3.1.1 Types of web-scraping

Web-scraping can be performed in two ways: with a visible browser interface (e.g. what a user sees when

using Microsoft Internet Explorer or Google Chrome), or via a headless browser. The latter refers to a

method whereby a computer connects to a web-address and collects the information held there (the HTML

code), but does not render that code in a browser, meaning the user does not see any actual webpages10.

This method is preferable over the former as it does not require as much computational power and does

not consume much working memory on the local device, meaning it can be executed relatively quickly and

for a large number of websites. In such a framework it is the connection speed between local device and

target that is the limitation. Headless browsers are however (at the time of writing11) limited to static

web-addresses, meaning that the information is held at an address and does not change. However as was

previously mentioned, TAS has a dynamically loading interface and so requires the former approach, which

is described in the following section.

3.2 How does our web-scraper work?

To provide the functionality required to manipulate a browser via its graphical user interface (GUI) - as the

case is using TAS - a software development tool called Selenium WebDrivery was used12. This facilitated the

automation of web-page manipulation. To name several examples, Selenium WebDriver is able to perform

actions such as clicking, scrolling and entering text into text-fields - all specified programmatically.

As inputs, TAS takes a search term (plus any filters that a user adds) along with a date range. As output,

the youngest 20 tweets in the date range are return, all of which contain that search term. Once the user

has scrolled to the bottom of the page, the next 10 tweets are loaded. This process continues until the end

of the date range is reached, i.e. once the oldest tweet within the date range has been loaded and displayed.

At this point any attempt by the user to keep scrolling will have no effect - no more tweets will be loaded.

A date range and the search term, as well as a filter to receive only tweets written in English13 are all

able to be specified simply through their inclusion within the target URL14. Selenium then enters this URL

8Web-crawling also includes how the search engines obtain their information (i.e. the encyclopaedia) to begin with. An

explanation of this does not lie within the scope of this study. Heydon and Najork (1999)y provide a good starting point.

9Also referred to as web harvesting and web data extraction.

10Headless browsing is a technique often used for debugging purposes, as errors can be detected without visualisation i.e.

without rendering the underlying information. This accelerates the process of web-development.

11Progress is being madey in the development of headless browsers for tasks such as scrolling dynamically loaded webpages

12A detailed technical explanation of this step shall not be provided here.

13Although Twitter includes this as an option within TAS, it is not guaranteed to classify the language with 100% accuracy.

14A Uniform Resource Locator (URL) can contain several elements, but usually essential are a protocol (http:) and a hostname

(www.twitter.com). More specific locations are then appended as necessary, commonly separated by a forward slash.

5

into the browser’s address field and visits the page.

Once the browser has reached the URL and the first 20 tweets have been loaded, a basic process is

followed and can be summarised by the following steps:

1. scroll to the bottom of the page

2. wait long enough for the next 10 tweets to be loaded

3. scroll to the bottom of the page again

4. Repeat steps 2. and 3. until no more tweets load

A programme to automate this process was written in Python, importing the Selenium WebDriver package.

A full description of the automation process is described by Algorithms 1 and 2 .

3.3 Stability considerations

As previously mentioned there are computational constraints to consider when working with a browser. In

the case of this task it was the working memory15 that posed this problem. Because the web browser receives,

stores and renders the information for all tweets, the amount of memory required climbs very quickly. Certain

steps can however be taken to reduce this burden, and can be divided into two branches: programmatic and

organisational.

In terms of organisation, it was necessary to create batch-processes to perform scrolling sessions, which

provided control and stability when scrolling downwards over the extremely long dynamically loaded webpage,

TAS. Due to the fact that the number of tweets posted that contain a given search term - over any given

time-span - cannot be known in advance, the size of the batches had to be determined heuristically. The size

determined by the user as a variable defined as the scroll.limit, which tells tells Selenium how many times

to scroll down - pausing for a given time between each scroll to allow TAS to respond and load the next 10

tweets.

The greatest gain in performance made through programmatic technique was gained by creating a custom

broswer profile that the Selenium package then called upon when opening the browser. Within such a profile

(depending on the choice of browser used16, it is possible to make tweaks such as to prevent images from

being downloaded and rendered, which is of course the main cause of memory allocation. Furthermore, one

can provide a chosen identity to present a target address with, which can determine the form of the data

a target supplies to a visitor. Presenting oneself, for example, as a 2008 version of a browser could limit

the quality of certain meta data that a target sends, with lower quality meaning less information, leading to

lower memory requirements. These ’tricks’ were necessary to allow each scrolling session to run as long as

possible before significantly eroding performance or possibly crashing, losing all progress.

In order to design an algorithm capable of recursing over the desired time-span of over two years on the

interface, it was necessary to first carry out many tests to obtain some heuristic parameters for a batchprocess.

When the Twitter timeline is being scrolled, it is thought of as reaching ever further back into the

past. Each time the user scrolls downards and loads more tweets, the newly loaded tweets are older than

the previous tweets.

The target URL is composed manually for each batch, including start and end dates for one of the two

week time blocks and one of the thirteen seach terms. This URL is then fed into

The scroll.count variable is initialised to zero. The scroll.limit must be sufficient for periods with many

tweets to be fully scrolled, not to lose any tweets. This limit of course differs with each search term as some

return many more tweets than others17. It was heuristically determined that a value of approximately 500

Algorithm 1 describes the iterative process performed by the web scraper for each scroll session defined.

15Random Access Memory (RAM).

16Drivers for Mozilla Firefox, Google Chrome and others exist, however ChromeDrivery proved itself to be the most reliable

when highly customised.

17Compare the number of tweets obtained for search terms ’interest rates’ and ’bull market’.

6

Input: target URL, scroll.count, scroll.limit

Output: HTML code

while scroll.count less than scroll.limit do

scroll to bottom of page;

if at end of page then

wait 3 seconds for next tweets to load;

current position becomes this one;

else

scroll to bottom of page;

end

end

begin

copy HTML source code

end

Algorithm 1: Iterative web-scraping algorithm for a dynamically loading website

[Algorithm 2] shows how [Algorithm 1] is then extrapolated into a batch process that will scrape tweets

over the desired time-span. Each scroll session covers a data range of two weeks.

Input: time-span start and end dates

Output: HTML code of TAS results

while scroll.count less than scroll.limit do

scroll to bottom of page;

if at end of page then

wait 3 seconds for next tweets to load;

current position becomes this one;

else

scroll to bottom of page;

end

end

begin

copy HTML source code

end

Algorithm 2: Batch-process algorithm to recursively scrape over desired time-span

7

3.4 Parsing the HTML code

The parts of the data that are useful for this study are summarised in Table 1 below. The Description

column outlines the data available for each tweet, whereas the Usage column describes what was ultimately

extracted for use in the modelling.

Data Description Usage

timestamp A millisecond accurate timestamp The calendar day

tweet-ID A unique identifier Remove any duplicates

tweet text The text string (max.140 characters) Sentiment analysis

times retweeted Number of times a tweet was retweeted As variable and weighting factor\*

times favourites Number of times a tweet was favourited As variable and weighting factor\*

Table 1: Summary of Twitter data usage

\* This is explained in detail in SectionXX

3.5 How is the HTML code parsed?

As was alluded to in Section, TAS represents the most accessible means of obtaining the desired Twitter

data. Before a suitable web-scraper can be outlined, a description of the interface offered by TAS must be

given. TAS is a dynamically loading webpage interface to a database. This means, that it holds a great deal

of information, but when it called upon it displays only a small portion of the results to begin with; the

next portion of results being loaded as soon as the user has scrolled to the bottom of the webpage. This

is common feature implemented by many websites that host data-heavy content, as it enhances the user

experience by delivering a lazy evaluation or just in time approach - data is loaded only at the moment it is

required. Other examples are the Google image search results page and a Facebook user’s main news feed.

In the case of TAS, roughly 20 results (i.e. 20 tweets) for a given search are first returned, with the next

10 being loaded dynamically once the user has scrolled to bottom of the page. Given this, it immediately

becomes clear, which difficulties are introduced by using TAS when trying to obtain data over a time-span

of several years: it is necessary to scroll to the bottom of a webpage for every 10 tweets.

To put this into perspective, the Twitter data that was used in the final analysis of this study contained

2,350,217 tweets.

Search term Total tweets Time-span coverage

(days) (% of 982 days)

bear market 47,924 963 98.1

bull market 74,937 965 98.3

dow jones 250,112 982 100.0

dow SPDR 1,628 700 71.3

dow wallstreet 26,395 921 93.8

federal reserve 378,970 904 92.1

financial crisis 261,500 922 93.9

goldman sachs 289,485 909 92.6

interest rates 396,765 857 87.3

market volatility 60,858 970 98.8

obama economy 202,654 908 92.5

oil prices 219,766 785 79.9

stock prices 139,223 982 100.0

Table 2: Breakdown of the total number of tweets extracted by search term

8

3.6 Pre-processing tweet text

1. Here we show what kind of things had to be removed (example of the hex code for smileys etc.)

2. Some things left in because certain sentiment models actually use them, e.g. :-)

3. Explanation of how regex code using Perl engine and hex-codes was used with a code snippet linked

in the appendix.

4. Before and after of one tweet.

3.7 Final output for sentiment analysis

Everything can be interpreted by the SA models to produce reliable results. Here we summarise the final

product of the work detailed in the chapter and summarise how it is stored. Description of overall process.

"Gone from HTML code to raw text, cleaned text, ready to process. Next chapter explains how SA was

performed." Link here to the flow-charts in appendix?

4 Time-series analysis - N versus P

When conducting time-series analysis, there are no hard-and-fast rules governing how many time-periods

must be included to guarantee model robustness. It is a question whose answer changes depending on the

data being used. There is a trade-off to be found between three main components: the number of periods

available, the number of covariates used (i.e. the number of model parameters to be estimated) and lastly

the level of noise within the data. [Reference - Book by Hyndman?].

There are other factors that should be taken into consideration in the context of financial markets,

and that is of trends and cycles. There are times in which an asset (e.g. a single company stock, oil prices

or an entire index) tends to move in one directions. It shows some level of momentum. The event of such a

cycle changing may be labelled a fraction or break in the assets price-path. The approach taken here to deal

with this facet of financial time-series is to make extensive use of a model parameters named frame-size,

which describes how many time-periods are used for each model that is fitted. Its usage is explained in more

details in [section: modelling, parameters, shifting the time-frame]

9

4\_Gradient\_Boosting

Nicholas Mitchell

March 23, 2016

Contents

1 What is boosting? 2

2 Building on the basics 2

3 Gradient boosting 4

3.1 The objective . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

3.2 General properties . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

3.3 Naive functional gradient descent . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

4 Component-wise functional gradient descent 8

4.1 Definition and properties . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

4.2 Parameter selection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

4.2.1 Learning rate: \_ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

4.2.2 Stopping iteration: mstop . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

5 Stochastic gradient boosting 12

6 Families of distributions 12

6.1 Gaussian . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

6.2 Binomial . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

6.3 Gamma . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

6.4 Inspection within R . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

7 Further work 13

List of Algorithms

1 Component-wise functional gradient boosting . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

1

1 What is boosting?

There were several important developments that were necessary to arrive at the model that this study

ultimately uses. They are each addressed in the following sections, and may be summarised as follows:

1. The first boosting algorithm was defined: AdaBoost BibTex Freund et al., 1996, Freund and

Schapire, 1997

2. AdaBoost was re-formulated as gradient descent, taking a special loss function BibTex Breiman et

al. 1998, Breiman 1999

3. AdaBoost was generalised to allow a multiplicity of loss functions: gradient boosting BibTex

Friedman et al. 2000, Friedman 2001

The generalised final product, gradient boosting, can handle regression, classification and ranking problems,

making it a tool with applications to many problems, in many different contexts.

As listed above, the first notions of boosting stem from work by Freund and Schapire insert Bibtex

reference, in which they developed an algorithm named AdaBoost (full form: adaptive boosting1) that

gradually improves model performance on a given data set over a number of iterations. Instead of defining

one complex function, AdaBoost iteratively fits a simple model to the data set. Before the first iteration, all

data points are considered of equal importance, and so uniformly weighted. In the case of regression with

square loss, the residual errors produced from each iteration (labelled the shortcomings) are subsequently

used to re-weight the data points. Each falsely classified point receives more weight (proportional to the

log-odds of the previous iteration’s weighted error) and, in a similar fashion, the weights of the correctly

classified data points are decreased. Finally, all weights are normalised to sum to unity before the next

iteration2. This weight adjustment steps instructs the naive model how to prioritise its fit or classification in

the following iteration - targeting areas where the model performs weakly. It is possible that many mistakes

are made by the naive base-learner throughout the process at every stage; however, the weights reflect how

many times each point was classified falsely, and it is the sum of these weights that defines the final model.

The shortcomings are the residuals from each fit, which are used to iteratively re-fit a function. The

model is updated after each iteration of re-modelling on the shortcomings, thus incrementally improving the

model and decreasing the observed error on the data set. If this overall error is thought of as a function of

the model to be minimised, then the residuals correspond to the negative gradient of that function. In order

to generalise the AdaBoost methodology for other loss functions, (as is described in the following sections),

it is more useful to retain this concept of gradients when referring to the shortcomings of a model. Reexpressing

the boosting methodology of AdaBoost in this generalised way: we are making approximations

to the gradient of a loss function in each step - this leads to the descriptive name functional gradient

descent (FGD).

2 Building on the basics

The simple idea of linear regression may be thought of as a fundamental idea behind much of the boosting

methodology used in this study. It is what is often used as a starting point3 when building a boosting-based

model and so may be considered an elementary building block e.g. when defining in a generalised linear

model (GLM).

We consider a function, f : X ! Y, that describes a data set by mapping an input, X =

fX1;X2; : : : ;Xng, to a response variable Y. Given the input and output variables, it is this function,

which is to be approximated. In the classic example of linear regression, this means placing a straight line

over or through the data set, which is as close as possible to as many of the points within that data set as

possible. The goodness of fit of that line can be quantified and so assessed by some metric on the distance

1A qualitative description only is given here. For a more detailed illustration, please refer to the referenced literature.

2As each step is reliant on the previous, the models that are fitted are not independent from one another. Within the

context of machine learning, this is often termed an ensemble method.

3Here we mean in terms of a base-learner and initial go-to method before more complicated models are considered. See

Section base-learners for more detail.

2

found between the line and each of the individual points. In the case of the ordinary least squares (OLS)

methodology4, this is achieved by summing the squares of the distances between each point and the line,

and then adjusting the line so that defined sum becomes as small as possible; thus the problem is one of

minimisation5. To help visualise this, the distances are highlighted by red arrows in Figure 1, which is a

short excerpt of some sentiment analysis data (see Section XX Data prep-exploration for more information

regarding the data used in this study). It is the sum of these lengths squared that is to be minimised by

moving the line. We notice that the lines are all vertical, which is because we are interested only in the

distance (also called the error or the residual) between the outcome variable that our line would predict for

a given input value on the x-axis, and the true outcome, i.e. the y-coordinate of the corresponding point

from the data set.

Figure 1: Sentiment scores of term "dow jones" versus Dow Jones returns of following day. Ten consecutive

days are randomly selected and displayed with their residual errors as red arrows to the regression line,

which is produced via a linear fit over all 695 data points. Inset: the same plot without error highlighting,

including all data points.

The line represents our function, an approximation to the data set, and describes a one-dimensional

response variable Y with one explanatory variable X. As will be shown in the following sections, this is

often the case in component-wise gradient boosting, where a single variable vector is regressed on the single

response variable (or iteratively on the residuals of consecutive fits).

4If errors are assumed to be normally distributed, we could here, instead, refer to the maximum likelihood estimation

(MLE)y.

5More generally speaking the problem is one of optimisation; minimisation when optimising over a loss function, however

maximisation when optimising over a likelihood function

3

3 Gradient boosting

3.1 The objective

The goal of gradient boosting is the same as any other (in this case, forecasting) model: to fit a function to a

data set that accurately describes the data and allows for new input data points to make predictions on the

outcome variable, all while minimising error. Consider a sample containing n values for each of p predictor

variables X = (X1;X2; : : : ;Xp)T, along with a corresponding one-dimensional outcome variable: Y 6. The

goal is then to find the optimal function f\_(X) that describes the given data and allows predictions to be

made for Y. The method should not be a black-box 7, i.e. it must be transparent at all stages, and the final

model f\_ should permit the interpretation of results and interactions between features.

A classic (simple) approach might fit additive regression models using MLE, such as the the linear model

example described in Section linear-model, applied to each predictor individually. The output of such a

model would then take the form shown in Equation (1), which has the structure of an additive predictor

provided by a general additive model (GAM) Bibtex Hastie EOSL and the functions ^ f1 + : : : + ^ fp each

correspond to the linear functions (later defined as the base-learners). These functions depend only on the

predictor variables that were used as inputs. We shall see that single variables are generally use as inputs

for each base-learner; however, subsets of variables may be allocated to each base-learner.

f\_(X) = \_0 + f1(X1) + f2(X2) + : : : + fp(Xp) (1)

There are several points to consider when using such a model, i.e. when selecting base-learners and

the subsets of variables to be used as input. Two examples, both of which indeed present themselves in

this study’s data set, are (1) high levels of pairwise-correlation between predictors, and (2) cases in which

we have wide data sets (p > n). The questions that arise regard matters such as the explanatory power

attainable in the face of m highly correlated variables being included in one model, and then which of those

variables provides the most information. A model that can cope with such input data should ideally include

an in-built method of variable selection that can deal with multi-collinearity, and also may return a sparse

model, i.e. not all predictors must be included in the final model. Gradient boosting provides a framework in

which those concerns are addressed satisfactorily, particularly because each step of the modelling procedure

is transparent, allowing errors in modelling assumptions to be identified.

The naive approach to variable selection by means of exhaustively fitting models for all possible subsets

of predictors is not an option when the data sets are wide, i.e. with large p. More systematic methods of

variable selection, especially the case of wide data sets, can be difficult to perform and do not guarantee

the optimal solution. Examples of step-wise search techniques include backward and forward selection,

which avoid exhaustive model fitting. While reducing the number of models fitted, these methods cannot

guarantee optimality as the explanatory power of feature interactions is not necessarily considered (most

easily demonstrated with forward selection) insert bibtex to Fukunaga Chapter10.

3.2 General properties

Gradient boosting provides a fitting method that minimises an empirical risk (loss) function8 w.r.t. a given

prediction function f. The risk function component is modular, meaning it may take many forms, in each

case describing a particular loss function, which is to be minimised9. Examples are the L1 (Laplace), L2

(Gaussian), Huber, exponential and negative log-likelihood loss functions10. In order to minimise such

functions, they must be convex and continuously differentiable so that the first derivative may be solved over

6Vector notation is used here, where X1 represents the vector (x1; x2; : : : ; xn) for p = 1.

7Neural networks are the standard example of such a model. High predictive accuracy may be obtained, but the modelling

process does not allow for much interpretability. The characteristic of the final model being non-identifiable increases difficulty

of further analysis. Recent advancements with visualisationsy into the understanding of the hidden layers have, however, been

made.

8Empirical risk and loss function are used synonymously. Depending on the context, other names are also given to the same

principle, for example: a cost function (neural networks) and the hinge loss (support vector machines), to name just a few.

9This terminology is commonly used in data miningy communities; however, the distinction between minimising loss functions

in data mining and maximising likelihood functions in classical statistics is rather hazy.

10For more detail on unbiased estimators that minimise a risk, refer to Ch. 3 of BibTex Pfanzagl

4

any interval. There are particular applications of loss function minimisation that make the even stronger

assumption that the loss-function be Lipschitz continuous insert Bibtex Differential Privacy, which

places an upper bound on the magnitude of the gradient of the function over an interval. Two exceptions to

continuous differentiability are the L1 and hinge loss functions, which are non-differentiable at their inflection

points. In software implementations this discontinuity is often set to zero as to prevent a computational

error being thrown (the gradient being undefined at the inflection points), thus allowing stable usage11. The

results of boosting are generally in the form of an additive predictive function, as depicted by Equation

(1), and so are readily interpreted. The further benefits that are reaped by using component-wise gradient

boosting12 will become apparent as the methodology is explored in the following sections, but may first be

summarised here by stating that boosting:

1. is applicable for use with a wide range of loss functions (as described above),

2. is able to perform variable selection during the fitting process,

3. can perform well in situations where p >> n,

4. inherently addresses multicollinearity between predictors, shrinking effect estimates towards zero,

5. optimises prediction accuracy with respect to a given risk function, and

6. offers high transparency throughout the modelling iterations.

3.3 Naive functional gradient descent

We first define the basic approach to gradient boosting, describing how gradient descent is performed. At

the same time, specific applications are introduced within the context of this study. The component-wise

version that was ultimately used is presented in the sections that follow. We start with the initial model,

slightly extending that given in Section basic-problem, by again considering a one-dimensional response

variable Y, but with a p-dimensional set of predictors X = (X1;X2; : : : ;Xp)T. A model that aims to fit a

model that explains this relationship may be presented thusly:

f\_ := argmin EY;X

f(\_)

[\_(Y; f(X))] (2)

where \_ is a loss function with the properties described in Section 3.2. The optimal model ^ f(\_) = f\_

therefore minimises the expected loss, i.e. the error. This study looks at several loss functions13 that are

able to be inserted into (2); two examples of which are the absolute loss when the Binomial (or Bernoulli)

method is used - illustrated in Equation (3) - and the squared loss in Gaussian regression - illustrated in

Equation (4).

\_

`1

= Y 􀀀 f(X) (3)

\_

`2

= (Y 􀀀 f(X))2 (4)

Modelling is performed on a sample set with n realisations X = (x1; x2; : : : ; xn) and Y = (y1; y2; : : : ; yn)

of X and Y, respectively, and so an exact expectation for Equation (2) is unknown. Instead, boosting

algorithms minimise an empirical risk function (the observed mean) w.r.t. some approximation function,

f(\_), given as:

11The L1 loss function has been shown, theoretically as well as experimentally, to achieve superior performance to the

alternatives in certain cases - especially in sparse models, where few coefficients are found to be non-zero(Insert XX Reference

to EOSL p.611-613).

12Component-wise gradient boosting is specifically addressed in Section 4.

13The Gaussian L2 loss function is used in Section XX; the binomial L1 loss function is used in Section XX.

5

R =

1

n

Xn

i=1

\_(Yi; f(Xi)) (5)

Referring once again to the principle of modularity, either of the two loss functions given in Equations

(3) and (4) (plus any other defined loss functions that fulfils the requirement of convexity and continuous

differentiability) may be inserted into the empirical risk function given by Equation (5). Equation (6) shows

the case for the L2 loss function.

R =

1

n

Xn

i=1

(Yi 􀀀 f(Xi))2 (6)

The next step is to establish the iterative gradient descent method, which minimises R = R(f(1); f(2); : : : ; f(n))

w.r.t. the approximation function, f(\_); where f(1) = f(X1); f(2) = f(X2); : : : ; f(n) = f(Xn). We notice that

these each are merely output numbers, meaning we can simply treat f(Xi) as parameters and take derivatives

with respect to Equation (5), which for the case of an L2 loss function, as in Equation (6), yields:

@R

@f(Xi)

=

@

@f(Xi)

Xn

i=1

\_(Yi; f(Xi))

!

=

@

@f(Xi)

(\_(Yi; f(Xi))) = f(Xi) 􀀀 Yi (7)

which illustrates the relationship between the residuals and the negative gradient of the cost function,

thereby demonstrating the reason why the former may be interpreted, in a more general manner, via the

latter. By simple re-arrangement:

Yi 􀀀 f(Xi) = 􀀀

@R

@f(Xi)

(8)

We define initial conditions by first setting the approximation functions f(\_) to some offset values

^ f[0]

(1); : : : ; ^ f[0]

(n), where an iteration counter m is set to zero (shown in the superscript of f(\_)). In each step, an

approximate to the gradient of the loss function is computed and used to update the approximation functions

as follows:

0

BB@

^ f[m]

(1)

...

^ f[m]

(n)

1

CCA

=

0

BB@

^ f[m􀀀1]

(1)

... ^ f

[m􀀀1]

(n)

1

CCA

+ \_ \_

0

BB@

􀀀 @

@f(1)

( ^ f[m􀀀1]

(1) )

...

􀀀 @

@f(n)

( ^ f[m􀀀1]

(n) )

1

CCA

(9)

where \_ is the learning rate, or step length factor - see Section 4.2.1 for more discussion on this model

parameter. With each iteration of the model, we notice the approximation functions (and so the coefficients),

as shown in Equation (9), are all simultaneously incremented14 in each iteration, by an amount proportional

to the gradient of the loss function. This is explicitly analogous to the shortcomings of AdaBoost (the

residuals themselves) being used to improve the model (Section 1) and embodies the principle of steepest

descent: the coefficients take the shortest path to their final estimates that minimise the loss function.

Because of this feature, the algorithm is also know as greedy Bibtex - Friedmann and EOSL. The process is

illustrated in Figure 2, which represents a contour plot of a fictitious cost function. The process is repeated

until the algorithm converges to the values ^ f[mstop]

(1) ; : : : ; ^ f[mstop]

(n) , which correspond the best approximation to

the loss function’s minimum. Here, mstop represents the number of iterations required to reach this minimum.

There are several weaknesses to this naive version of FGD, one of which is that any structural relationship

between the approximation functions ^ f[mstop]

(1) ; : : : ; ^ f[mstop]

(n) that act upon the data set are ignored. Simple

14Because all values f(Xi) are updated in each iteration, this procedure may be referred to as batch gradient descent. This

highlights the difference to component-wise gradient descent, described in Section 4.

6

Figure 2: A contour plot representing the surface of a simple loss function. The approximation function follows

the principle of steepest descent from its (blue) initiation point - crossing each contour line orthogonally

- reaching the (red) global minimum at iteration number mstop.

relationships are assumed: ^ f[m]

(1) ! Yi; : : : ; ^ f[m]

(n) ! Yn, which may fail to capture all the information held in

the model variables. The algorithm defined in the following section addresses this weakness and improves

further upon the progress this naive FGD method has made - describing the primary modelling tool used

for this study.

7

4 Component-wise functional gradient descent

4.1 Definition and properties

With the two concepts of boosting and gradient descent having been defined, as well as the method of

(batch) gradient boosting Friedman 1991, the final enhancement is to be introduced; defining componentwise

gradient boosting, which adds the last features that were outlined in Section 3.2. Namely, that a form

of variable selection be implemented15 within the boosting process. In addition to variable selection, as

will be shown, this also inherently provides a certain amount of assurance of performance in the face of

multicollinearity. Algorithm (1) defines the iterative procedure in which component-wise boosting minimises

the empirical risk R (given in Equation (5)) over the approximation function, f.

Algorithm 1: Component-wise functional gradient boosting

Input: loss function, \_; base-learners; counter, m; learning rate, \_

Output: optimal prediction function: f\_

Step 1. Initialise the n-dimensional function estimate ^ f[0] with offset values /\* e.g. := 0 \*/

Step 2. Specify a set of P base-learners; initialise the counter, m := 0

Step 3. Increase m by one

Step 4. a. Compute the negative gradient of the loss function: u[m] = 􀀀@\_

@f

b. Evaluate the negative gradient at the previous iteration’s model estimate

c. Fit each of the P base-learners to the resulting negative gradient

d. Select the base-learner that best fits u[m] by some criterion /\* e.g. SSE \*/

e. Set ^u[m] equal to the best fitting base-learner

f. Update current estimate ^ f[m] := ^ f[m􀀀1] + \_ \_ ^u[m]

repeat

Steps 3 and 4

until m = mstop;

return the prediction function that minimises \_: f\_ = ^ f[mstop]

Step 1 sets the initial function estimate set to a zero-vector. The P base-learners that are specified in

Step 2 are generally simple estimators that each take a pre-defined set of input variables and provide a

univariate response. Each of them may take different subsets of the (entire) model’s input variables; the

subsets are usually relatively small, in order to make use of the model’s features. The base-learners that

are provided to a model for implementation within Algorithm (1) provide the tool that allows the modeller

to stipulate structural assumptions regarding the model. Beyond simple the grouping of variable subsets,

several further options are available, such as methods to introduce categorical and ridge-penalised effects -

refer to mboost\_tutorial for more information on these options.

In this study, the base-learners are least-squares estimators, with input of single predictor variables.

Therefore, each base-learner fits a simple linear model against the negative gradient for each of the individual

predictor variables. Coupled with selecting only the best base-learner (in Step 4.c), this individual predictor

modelling highlights how component-wise gradient boosting is still able to perform, should there be relatively

high levels of multicollinearity among the predictors. If two variables are highly correlated and subsequently

both perform strongly, only the best is selected, meaning that both predictors are not necessarily included

in the final model - thus reducing multicollinearity effects16. There is, however, an element of uncertainty

15This is of great importance for the concerned data sets, as the inclusion of lagged variables produces data sets where p > n.

16This says that the two predictors are almost parallel in the sample space and so would be expected to explain the same

dimensions of variance in the response variable.

8

in which predictor will be selected over the many iterations. If the correlation between two predictors is

extremely high (e.g. > 0:7), which predictor is selected at each step may not be consistent - therefore it must

be mentioned that there are limits to this facet of the variable selection feature of component-wise gradient

boosting. This is taken into consideration within the empirical segment of this study, as discussed in Section

correlation\_cutoff.

In Step 4, the computed negative gradient estimate is evaluated at the vector estimate of the previous

iteration’s approximation function, ^ f[m􀀀1]

􀀀

X>i

\_

, yielding Equation (10):

u[m] =

\_

u[m]

i

\_

i=1;: : : ;n

:=

\_

􀀀

@

@f

\_

\_

Yi; ^ f[m􀀀1] 􀀀

X>i

\_\_\_

(10)

The criterion that is used (in Step 4.d.) to select the best performing base-learner is the sum of squared

errors (SSE), however this may be adapted to the model, for example in the case the base-learners should

become more complicated and take forms other than linear models. L1 absolute error might be a good

alternative if the model should be more robust to outliers. In addition to the variable selection carried out

in Step 4.d., there is an inherent model selection process also taking place. As previously mentioned, the

choice of base-leaner provides a means to specifying structural assumptions, and the efficacy of those choices

can be seen in this step. The learning rate, \_, for use in Step 4.f. should be a real number lying on the

interval [0; 1). More discussion on this parameter can be found in Section 4.2.1. The last major point of

interest within Algorithm (1) is the parameter mstop. A method to approximate an optimal value for this

important parameters is described in more detail in Section 4.2.2.

The model description given in Section 4 initialises the weights to zero. There are several reasons why

this is a reasonable choice. Firstly, initialising the values to zero means that in the case of a particular

variable never representing the closest approximation to the negative gradient of the loss function, u - by

not once producing the base-learner fit with the lowest error over all variables - this variable is never selected

and so its weight never incremented. Keeping in mind that these weights correspond to the coefficients of

the variables in the final model, this then equates to the final value of this coefficient at completion of the

gradient descent remaining untouched, equal to zero, ergo the variable is not selected for the final model.

This is part of the intuition behind the inherent feature of variable selection presented by component-wise

gradient boosting17. The second useful property of using zero as the initial weights is that, regardless of

whether a coefficient evolves to be positive or negative at completion, the starting point was the same. To

a moderate extent, this symmetry additionally facilitates the direct comparison of variable importance via

their coefficients magnitudes, which tell us how far each base-learner progressed the model along the surface

of the loss function as is approached the minimum. This is of course a function of the negative gradient and

the learning rate.

This section has summarised the main methodology used within this study; however some preliminary

testing was also completed using several variants of this model. Some extra information explaining their

usage is explained in the following sections.

4.2 Parameter selection

4.2.1 Learning rate: \_

The learning rate, \_, is commonly held constant throughout the boosting process, which has proven to be a simple but effective method. To see why this approach is effective, we must inspect the

magnitude of the increments to our approximation function during the gradient descent, not only the scalar

learning rate. One might consider different learning rates and their effect on the speed of approach to the

loss-function’s minimum (given there being only one global minimum). Given a tiny learning rate, the speed

of approach would be extremely slow; however, offering a very close approximation to the minimum as a

by-product. Selecting a large learning rate would conversely allow for a rapid descent towards the minimum;

however, offering relatively little precision. The truth, however, is that the increments that are added to

17Any possibility of base-learner being incremented more than one time and providing a final value f

[mstop]

i equal to zero

is completely ruled out by the assumption of the loss-function being convex. The increments of for one particular base-learner

(and so the variables contained) can only be of one sign, positive or negative, meaning the summation may never converge to

zero.

9

our approximation function at each iteration are indeed adaptive - in terms of the negative gradient, which

must decrease as the gradient descent approaches the minimum, by the definition of the loss function being

convex. This can be seen in Figure 3, where a simple one-dimensional case is demonstrated.

Figure 3: A loss function for the one-dimensional case; the size of the red points along the curve representing

the incremental addition to approximation (prediction) function during gradient descent. Holding \_ constant,

it is clear that the nominal step-size, \_ \_

\_

􀀀 @

@f \_(Y; f)

\_

, does indeed adapt in size at each iteration.

A fictitious loss function is plotted for the one-dimensional case (the parabola: y = x2 + 1

2 ), where the

colour gradient of the curve reflects the magnitude of the negative gradient, u =

\_

􀀀 @

@f \_(Y; f)

\_

; dark blue

indicates a steep gradient, which slowly lightens as the function levels out to its minimum. A value for \_

of 0:1 is defined, giving the size of the red points reflects the magnitude, \_ \_ u, by which the approximation

function is incremented during gradient descent. It is clear that gradient descent will steps of ever decreasing

length as the minimum is approached, the decrease in step-site proportional to the reduction in the gradient.

Therefore there is no obvious benefit gained by e.g. adaptively decreasing \_ over the iteration process. The

accuracy this method offers, via the self-adapting step-length, is assumed to provide sufficient precision when

approximating the loss function’s minimum, even though \_ is held constant.

Of course the constant value of \_ is still a model parameter to be optimised, if not for precise approximations,

for the the gradient descent to be performed as efficiently as possible in terms of computational cost.

For performance it is known to suffice to use a small value, e.g. 0:1 BibTex to (Schmid Hothorn 2008b),

as in Figure fig:grad-descent. Furthermore, to dynamically adapt the step-size factor to the iteration

count of the negative gradient does not improve the estimates of f\_, and will only increase the computational

cost of running a descent to convergence. It is worth noting, additionally, that large values may prevent

convergence to the minimum, and could even lead to cases of divergence18.

4.2.2 Stopping iteration: mstop

As discussed in the previous sections, there are two main input parameters that have an effect on the overall

performance of the model: \_ and mstop. The learning rate, \_, must be assigned a sensible value (which

depends upon the input data); however, has the least overall effect of the two parameters. It must lie on the

interval [0; 1), and the authors of the mboost package found a small value, e.g. 0:1 to consistently produce

reasonable results link to Schmid Hothorn 2008b. The usage of this variable is discussed more within

Section empirical work. The critical model parameter is the number of boosting iterations, i.e. when

to end the gradient descent algorithm, which requires optimisation with regards to the data set at hand.

Up until this point, this was merely labelled as the point when Algorithm (1) converges, mstop; in practice

18To show this, one must simply use the argument presented in Figure 3, instead using a large value of \_ to see that the

minimum may easily be overshot.

10

however, the exact value is less well defined and must be optimised empirically. One must consider the reality

of overfitting the model to the in-sample (training) data set. If the model trains too closely to the data, the

resulting prediction function will likely perform badly in out-of-sample testing. It is therefore necessary to

perform some manner of cross validation on the results obtained from the gradient descent procedure.

One could perform the cross-validation in a number of ways, for example, using (1) k-fold cross validation,

(2) sub-sampling and (3) bootstrapping methods - all of which are implemented within the mboost package,

via the function: cvrisk(). This study exclusively made use of bootstrapping methods, whereby the number

of cross-validation replications used was 25. The output model objects (created by the mboost\_fit function)

do not simply contain the final approximation function with the coefficients of the selected variables, but

rather the information from every iteration from the gradient descent. The cross-validation can then bootstrap

the results at each boosting iteration and record the error. Executing 25 bootstrapped replications

then allows the average (squared) error to be computed - the iteration that holds the minimum value from

this set of results indicates the optimal value of iterations, mstop.

Consider a model that was produced from component-wise boosting, running in total for 100 iterations.

Figure 4 19 illustrates the cross-validation methodology on the outcome, illustrating how the optimal number

of iterations, mstop, is identified. The iteration number is selected, where the minimum error over the 25-

bootstrapped samples is found - the process is labelled an early stopping strategy, which aims to optimise

the final models prediction accuracy.

Figure 4: Example of 25-fold bootstrap crossvalidation

for a model with 100 iterations. Twenty-five

light grey lines shows the error at each iteration for

the twenty five bootstrapped samples. The black line

displays the average over all bootstrap results. The

minimum of the averaged error is highlighted with a

dashed vertical line mstop = 33.

The mstop parameter affects the level of relative

complexity20 that the chosen model exhibits. If the

final approximation function is selected, which was

produced at the hundredth iteration (in the example

given), the model is likely to include many parameters,

which were selected in an exhaustive search for

the minimum of the loss function. Parameters that

perhaps have little effect on the outcome variable

may have been included, leading to overfitting. In

comparison, the value of mstop obtained from (bootstrap)

cross-validation not only has a lower average

squared error (by definition), but is also likely to

return much simpler model. Less iterations would

have been performed, meaning that less variables in

a wide data set can have been selected, and those

that were selected are the influential variables. As

can be seen from Figure 4, the error reaches a minimum

quit early on, and plateaus out. This means

the approximation function at iteration mstop = 33

explains just as much variance in the data set as the

model at iteration m = 100, so selecting the simpler

model is good practice (by arguments of model

parsimony, i.e. ’Ockham’s Razor’).

The mboost literature mboost\_tutorial also

discusses usage of alternative criteria in order to locate

the mstop value. The example given is that of

the Akaike Information Criterion (AIC). It is suggested

that this method, however, tends to produce

larger values of mstop, which overshoot the minimised squared error shown through cross-validation.

19Source: mboost\_tutorial.

20The term ’complexity’ is naturally somewhat subjective, being model dependent.

11

5 Stochastic gradient boosting

The introduction of a stochastic component to gradient boosting has proved to be a great tool in reducing

the standard errors in a final prediction model by variance reduction, especially when the predictors show

signs of correlation. The idea of variance reduction is already seen in older machine learning algorithms

such as bagging (bootstrap aggregation) and random forests insert reference to Breiman. The former

creates many subsets of the training data to fit many models, and taking the average produces a results with

smaller errors than would’ve been found by using the whole training data. The bootstrapping procedure is

out-of-bag (OOB) with replacement, meaning some variables will be selected many times and other perhaps

not at all, depending on the setup. Random forests enhances bagging further still by saying (in terms of

classification trees): at each stage when a tree has been fitted to one of the bootstrapped samples, select a

subsets m of the p variables from the terminal nodes at random, then select only the best variable among

those m variables for the next split point.

This insertion of a stochastic procedure - randomly selecting from the variables after fitting the tree -

squeezes as much variance reduction as possible into the modelling and therefore reduces the final error

as much as possible when averaging. It is this notion that is applied to gradient boosting, thus making it

stochastic. The random selection can occur at one of two places, creating either a random subsets of the

training data (à la bagging), or a random subset of the features found at each step (à la random forests).

The randomness in the model fit and reduction in final estimate errors, coupled with a slower loss-function

descent, may also hinder over-fitting and so improve the models ability to generalise to out-of-sample data.

Other than the inclusion of this simple step into the method that was set out in Section 3, no other

changes are made to the iterative procedure. As one may expect, removing information at each iteration

can mean that some of the other model hyper-parameters must be adjusted, i.e. differing optimal values are

likely to be found. The tendency is for the model to require a larger number of iterations to descend along

the surface of loss function to the minimum.

Additionally, of course, there is the introduction of a new hyper-parameter, namely the proportion of

data or features that are to be selected at random. This parameters can also be optimised for; however,

Friedman link to 1999 paperFig1-Page5 found that values between 50 % and 80 % provided the lowest

errors.

A small practical note: the R package mboost has the functionality of the bagging approach, sub-sampling

the input data, but does not support the random forest enhancement. The functionality is, however, provided

by the gbm package.

6 Families of distributions

Depending on the requirements of the model, a specific family must be selected. The mboost package in R

supplies many families. They are listed, along with their properties in "Model-based Boosting in R" in Table

4 BibTex reference to mboost\_tutorial. This section briefly outlines the practical aspects of several

further families of regressors that are used within this study. An simple outline is provided to explain in

which situation each family may be used, indicating why the methods were required in several aspects of the

empirical work.

6.1 Gaussian

This family was used extensively for the general linear models performed on all data sets in order to predict

stock market movements, discussed further in Section XX. The Gaussian family is used in order to provide

the conditional mean of a continuous response. In this case, the assumption is that the conditional outcome

distribution, YjX, is normally distributed and that the loss function is the negative Gaussian log-likelihood,

which is equivalent to the L2 loss - given in Equation (11):

\_(Y; f(X)) =

1

2

\_ (Y 􀀀 f(X))2 (11)

12

6.2 Binomial

This family is used in order to model a binomial class response21: {0, 1}. Just as the Gaussian family,

the binomial family was used on all data sets in this study to predict the direction of the market, but

without regard for the magnitude. Analogously to the Gaussian family, the probability parameters may be

approximated through the minimisation of the negative binomial log-likelihood - given in Equation (12):

\_(Y; f(X)) = 􀀀[Y \_ log(P(Y = 1 j X)) + (1 􀀀 Y ) \_ log(1 􀀀 P(Y = 1 j X))] (12)

6.3 Gamma

This family allows predictions to be made purely of the magnitude of stock market movements, with no

regard for the direction. The gamma distribution, implemented as the GammaReg family within the mboost

package, and provides a continuous non-negative response, required for such a model. This function uses

the negative gamma-likelihood coupled with the logarithmic link function. For more information on this

distribution and estimates of its parameters, refer to mboost\_tutorial and choi wette.

6.4 Inspection within R

Listing 1 illustrates how one may inspect a family contained within the mboost package, directly within the

R console. On line 1 the mboost package is first loaded into the session. The details of the Gaussian family

are called up on line 3 and the information about the negative gradient on line 10. Similar operations may

be performed for many of the families that the mboost package contains.

1 R > library(mboost)

2

3 R > Gaussian()

4

5 Squared Error (Regression)

6

7 Loss function: (y - f)^2

8

9

10 R > slot(Gaussian(), "ngradient")

11

12 function (y, f, w = 1)

13

14 y - f

Listing 1: An example of how to investigate the properties of an implemented family within the mboost

package - here the example of the Gaussian family.

7 Further work

Add this to the further work section at the end

A convenient side-effect of this methodology’s variable selection ability is that it is no longer necessary

to separate variable selection and model fitting, such as is often the case with wide data sets. An interesting

topic might be the effectiveness of feature selection, in comparison to techniques of feature reduction such as

principal component analysis (PCA), where condensing many predictors into a handful that are frequently

able to explain large amounts of the variance in a data set. The results from such models, however, are not

21As a practical note, the Binomial() family within the mboost package returns values {-1, 1} in its binary response, for

reasons of computational efficiency.

13

easily interpreted directly, and are not necessarily able to be linked retrospectively to the input. This means

it can be impossible to say how what impact each individual predictor had on the final model22.

22An R package exists, named FactoMinoR [insert Bibtext Ref XX], which allows some level of further analysis and interpretation

of PCA results w.r.t. the input variables.

14