D3.1.1 Regression models of trends in streaming data

Dr. Sina Samangooei, University of Southampton
Daniel Preotiuc, University of Sheffield
Dr. Jing Li, University of Sheffield
Prof. Mahesan Niranjan, University of Southampton
Dr. Nicholas Gibbins, University of Southampton
Dr. Trevor Cohn, University of Sheffield

Abstract.
FP7-ICT Strategic Targeted Research Project (STREP) ICT-2011-287863 TrendMiner
Deliverable D3.1.1 (WP3.1)

We present initial work on building predictive financial and political opinion models. We address various concerns including: a large scale text data preprocessing tool; a scalable mapreduce based test setup for algorithm development; time series data models and processing; financial and political time series data via web APIs; a regression model for financial data and correlation analysis for political data.

Keyword list: text processing, tokenisation, language detection, stemming, twitter, regression, correlation, financial, political

Project	TrendMiner No. 287863
Delivery Date	May 1, 2012
Contractual Date	May 1, 2012
Nature	Deliverable
Reviewed By	Thierry Declerck & Mauro Navarra
Web links	http://github.com/sinjax/trendminer
Dissemination	PU

TrendMiner Consortium

This document is part of the TrendMiner research project (No. 287863), partially funded by the FP7-ICT Programme.

DFKI GmbH
Language Technology Lab
Stuhlsatzenhausweg 3
D-66123 Saarbrcken
Germany
Contact person: Thierry Declerck
E-mail: declerck@dfki.de

University of Southampton
Southampton SO17 1BJ
UK
Contact person: Mahensan Niranjan
E-mail: mn@ecs.soton.ac.uk

Internet Memory Research
45 ter rue de la Rvolution
F-93100 Montreuil
France
Contact person: France Lafarges
E-mail: contact@internetmemory.org

Eurokleis S.R.L.
Via Giorgio Baglivi, 3
Roma RM
00161 Italia
Contact person: Francesco Belini
E-mail: info@eurokleis.com

University of Sheffield
Department of Computer Science
Regent Court, 211 Portobello St.
Sheffield S1 4DP
UK
Contact person: Kalina Bontcheva
E-mail: K.Bontcheva@dcs.shef.ac.uk

Ontotext AD
Polygraphia Office Center fl.4,
47A Tsarigradsko Shosse,
Sofia 1504, Bulgaria
Contact person: Atanas Kiryakov
E-mail: naso@sirma.bg

Sora Ogris and Hofinger GmbH
Linke Wienzeile 246
A-1150 Wien
Austria
Contact person: Christoph Hofinger
E-mail: ch@sora.at

Hardik Fintrade Pvt Ltd.
227, Shree Ram Cloth Market,
Opposite Manilal Mansion,
Revdi Bazar, Ahmedabad 380002
India
Contact person: Suresh Aswani
E-mail: m.aswani@hardikgroup.com

Changes

Version	Date	Author	Changes
1.0	22.03.2012	Kalina Bontcheva	creation
1.1	03.04.2012	Sina Saman-gooei	fleshing out the structure
1.2	17.04.2012	Sina Saman-gooei	Work description of the preprocessing tool
1.3	17.04.2012	Sina Saman-gooei	Background reading in section 3 financial stuff, integration of the paper in section 2
1.4	20.04.2012	Daniel Preotiuc	Integration of the paper in section 2, final version of section 2
1.5	21.04.2012	Sina Saman-gooei	Tidied up some grammatical horrors in section 2 and 3. Section 3.2 is now in a finalish state
1.6	23.04.2012	Sina Saman-gooei	Added Sheffield's contribution to chapter 3. Finished section 1. Added executive summary and preamble.
2.0	27.04.2012	Sina Saman-gooei	Minor corrections suggested by Mauro made by Sina
2.1	30.04.2012	Trevor Cohn	Minor corrections to section 3.2.

Executive Summary

The aim of work package 3 is to develop machine learning methods capable of modelling real-valued time series based on contemporaneous textual input. The text is derived from social media, initially using the streaming feed of public Twitter status messages, and the time series are derived from the two use cases of the project: political polling and financial markets. This first deliverable includes our initial work in constructing tools and basic machine learning components for developing such models.

The central component of the deliverable is the processing pipeline which takes a stream of Twitter status messages ('tweets'), to which it applies a series of language processing algorithms – namely tokenization, language detection, word stemming and sentiment analysis. The processed data is then aggregated over a time-period to produce features suitable for describing movements in a time-series, e.g., word frequencies or aggregate sentiment relating to a given person, party or company. The sheer scale of Twitter data complicates even this simple level of processing, necessitating a parallel architecture in order to perform these steps in a real-time setting. For this reason we have developed our pipeline for feature processing and machine learning in the Apache Hadoop framework, an open source implementation of Google's MapReduce parallel architecture.

In designing the pipeline, our intention was to reuse a number of pre-existing NLP tools. However we discovered that although a few freely available tools did exist, they either could not be directly applied to our data, or else were implemented very inefficiently and consequently were not able to meet our high performance runtime requirements. For these reasons we developed a number of processing components ourselves, which we have built into the pipeline. Our implementation has been made open source, and we anticipate considerable interest from the community as it is the first such tool to our knowledge for processing social media text.

The second part of this deliverable provides initial analyses of the two project use cases using a year of processed Twitter data.[1] Previous research into predicting political opinion and financial market value time-series has made extensive use of careful data filtering combined with sentiment analysis in order to measure the aggregate sentiment of the users of social media with respect to certain events, people or issues. For example, this has taken the form of searching for the sentiment attached to tweets including 'Obama' for predicting the winner of the last US presidential election, or the number of tweets expressing a 'calm' sentiment for tracking the Dow Jones industrial index. Our analysis

[1]This data was downloaded from the gardenhose streamining API.

has aimed at replicating these approaches using our own Twitter data, processing pipeline and time-series data (UK polls before the 2010 election, and the Apple share price).

The deliverable includes tools for measuring correlations between a feature and our time series, and linear regression for predicting subsequent values for the time series. Our findings are that the previous techniques cannot be directly ported to our data, but require careful tuning, particularly in filtering for relevant messages and making good use of sparse data.

Contents

Chapter 1

Introduction

1.1 Relevance to TrendMiner

Deliverable D 3.1.1, scheduled to be completed at a very early stage of the project (M6) is intended to establish a working dataset and initial algorithms for future use in the project. The envisaged difficulties of having to deal with very large volumes of data by setting up of the computing infrastructure and the software pipeline for preprocessing and feature extraction. Preliminary regression analysis is included to confirm that the whole pipeline works satisfactorily in extracting features from tweet data. The focus of this report is in describing the implementations of each of the modules in the pipeline. For MicroBlog data, these differ from conventional text analysis approaches in a number of ways, and these challenges have been addressed and a system working to satisfactory performance has been achieved.

1.1.1 WP 3 Description

WP3 of TrendMiner is aimed at the development of machine learning methods for mining streaming media. Regression analysis to extract correlations between response variables of relevance to the two application domains and input features extracted from text is the starting point of the study. More difficult problems of learning in a non-stationary environment using more advanced models, modelling spatial and demographic variations in the information extracted and evaluation of both the above in the application areas are aspects of this WorkPackage.

1.1.2 Relevance to project objectives

The work reported in this deliverable provides the computational framework and software pipeline for pre-processing large volumes of tweet data. The outputs will be used in all analyses to be carried out in WP3.

1.1.3 Relation to other workpackages

Similar to 1.1.2, the preprocessing infrastructure is essental for the provision of basic data and features in all parts of the project whereever data is required.

Chapter 2

The Tools

In this section we describe our new open-source framework for efficient text processing of streaming OSN (Online Social Network) data. Our system is focused on a real world scenario where fast processing and accuracy is paramount. For this purpose, we make use the MapReduce framework for distributed computing.

OSNs have seen a rapid rise in number of users and activity in the past years. The accompanying availability of large amounts of data pose a number of new natural language processing (NLP) and computational challenges.

There are several challenges for text processing tools when faced with OSN data. These include the short length of messages, inconsistent capitalization patterns, ad-hoc abbreviations, uncommon grammar constructions and threaded discussions of friends in a network structure. When standard text processing tools, like part-of-speech taggers or named entity recognition systems, have been applied to OSN data their results have shown a significant drop in accuracy [Gimpel et al., 2011, Ritter et al., 2011], making their use in a pipeline untenable. Several researchers have addressed some of these problems in the past years, creating specific text processing tools for OSN data. Our system is the first to unify these tools with an emphasis on the adaptation to real world scenarios that include processing batches of millions of items or online data processing.

We propose a framework that can combine existing tools whilst being extensible to allow the addition of future components. Moreover, the system is built in as pipeline with interchangeable modules which gives the end user control over what processing steps are required for their given application. In order to achieve this, we keep the format of the original data and at each step of the pipeline we augment the output of the previous steps with extra fields corresponding to the results of preceding steps. The system is built using Twitter data[1] and the JSON data format[2], but we can easily adapt it to data from other OSNs (e.g. Facebook/Google+ status updates) that share similar features.

[1]`https://dev.twitter.com/docs/streaming-api/methods`

[2]`http://www.json.org/`

Another important challenge posed by OSN data is the sheer size of the datasets that we need to process. Our setting that includes massive datasets and the I/O bound nature of the analysis lends itself perfectly to using the MapReduce distributed framework. Text analysis tasks can mostly be done in parallel in the *map* part while the aggregation into feature vectors is achieved in the *reduce* part of the framework.

In section 2.1 we present the architecture of our system and the modules we have already implemented. A detailed description for developers of the tools and their usage is presented in section 2.2.

2.1 Preprocessing Tool

In this section we describe the architecture of our system and the modules we have implemented to date. For these modules we also provide quantitative and qualitative evaluation. We present how our modules currently integrate in the system, with the planned modules and with the analysis described in the next chapter.

2.1.1 Modules

We have implemented and tested to date 3 components of our preprocessing pipeline in our analysis tools. All three stages are implemented in pure Java. These are: Twitter specific **tokenization**, short text **language detection** and **stemming**. We present a brief description of all these modules in the next subsections.

Tokenization

Tokenization is an essential part in any text analysis system and is normally amongst the first steps to be performed in a preprocessing pipeline. Its goal is to divide the input text into units called tokens, with each of these being a word, a punctuation mark or some other sequence of characters that holds a meaning of its own.

Tokenization of OSN data, and in particular of Twitter data, poses challenges beyond those present in processing traditional sources (e.g. newswire). In particular, we must handle URLs, sequences of punctuation marks, emoticons, Twitter conventions (e.g. hashtags, @-replies, retweets), abbreviations and dates. Our system handles all of these challenges as we show in the qualitative evaluation. The implementation works through a chain-able set regexes which define patterns of token to “protect” in the first phase, followed by blank space delimitation of the remaining tokens for the second phase. This modular design means that this tokeniser can be easily extended to protect different tokens in the future. This is useful for general analysis as protected terms include urls, twitter hashtags and emoticons.

Note that the current version only works with latin scripted languages and their conventions (e.g. delimitation between words by punctuation or whitespace) and an extension to other languages (e.g. Asian languages) is planned for future versions.

Language Detection

There are many language detection systems readily available to be used for this task. The main challenge for these when faced with OSN data is the short number of tokens (10 tokens/tweet on average) and the noisy nature of the words (abbreviations, misspellings). Due to the length of the text, we can make the assumption that one tweet is written in only one language. Most language detection tools work by building n-gram language models for each language and then assigning the text to the most probable language from the trained model.

We choose to use the language detection method presented in Lui and Baldwin [2011] which we have reimplemented in Java. We choose this method over others for the following reasons: it is reported as being the fastest, it is standalone and comes pre-trained on 97 languages, it works at a character level without using the script information (this way we need to feed only the text field) and it was used by other researchers with good results [Han and Baldwin, 2011].

Stemming

Stemming is the linguistic process that reduces inflected and derived words to their root form. We use a pure Java implementation of the traditional Porter Stemmer, which is the standard stemmer used in NLP and Information Retrieval tasks. We chose the Snowball stemmer [Porter, 2001] backed by the Terrier Snowball stemmer implementation [Ounis et al., 2006].

2.1.2 Architecture

In this section we present the architecture of our pipeline of preprocessing tools. We identified two main use cases. Firstly, the batch analysis of several terabytes of tweets and applying filters for keywords, language, etc in order to compute aggregate counts of features, sentiment, etc over them when dealing with archival scenarios. Secondly, the analysis of millions of tweets per hour when dealing with real time analysis scenarios. To address these concerns, we propose a set of command line tools. The tools implement the stages of our preprocessing pipeline, the stages of which were presented in section 2.1.1.

We expect that any particular task in the pipeline applied to an individual tweet will have little processing requirements as compared to the I/O requirements of reading, preprocessing and outputting in a useable format several terabytes of compressed tweets.

This *I/O bound* nature of twitter analysis has been addressed in the past (both by various authors as well as Twitter's own in house development team[3]) with the use of clusters of machines with shared access to distributed tweets using the MapReduce framework and distributed filesystem. MapReduce is a software framework for distributed computation. MapReduce was introduced by Google in 2004 to support distributed processing of massive datasets on commodity server clusters. Logically, the MapReduce computational model consists of two steps, Map and Reduce, which are both defined in terms of inputs and outputs of $< key, value >$ pairs. For example, the keys could be filenames, and the values could be the actual contents of the files, or the keys could be the line number of a text file. In the case of processing twitter data, our keys are null and our values are individual twitter statuses held in the standard Twitter JSON format. Any analysis is augmented to this JSON map. We discuss the data format consistency considerations in greater detail below.

We take advantage of the relatively mature Apache Hadoop[4] MapReduce framework to apply distributed processing to tweets. When interacting with Hadoop it is possible to dictate the map and reduce functions using either *Hadoop Streaming*[5] or writing custom Java tools interacting with the underlying Hadoop Java libraries. Hadoop streaming allows the specification of the mappers and reducers through POSIX like standard in and standard out enabled command line utilities. This allows for quick prototyping using any language the user wishes, but doesn't provide the flexibility exposed when using Hadoop as a library in Java. Instead we choose to implement a Hadoop enabled preprocessing tool written in Java. This tool exposes the various stages of our preprocessing pipeline as modes. The inner components of the tool are shared between two separate tools: a local command line utility (primarily for local testing purposes) and a Hadoop processing utility. The individual stages of the preprocessing pipeline are implemented in pure Java and exposed as modes in the tools. In the local utility, individual tweets are loaded one at a time and each selected pipeline stage is applied to each tweet as it is loaded. In the Hadoop implementation the map stage is used to load each tweet wherein each preprocessing step is applied to an individual tweet and emitted by the Mapper and the Null reducer is used as no further processing needs to occur after map. The key consideration in the design of these tools are:

Modularity Our tools are engineered for extensibility. Firstly, the Hadoop and Standard tools are both driven through the same "mode" specifications and implementations. To implement a new mode which works in both tools, a simple Java interface is implemented which specifies a single function which accepts a twitter status and adds analysis to the status's analysis field. Furthermore, multiple implemented modes can be executed in a single invocation of the tool. Concretely, this results in multiple analysis being performed on a single tweet while it is in memory (in the Standard

[3]`http://engineering.twitter.com/2010/04/hadoop-at-twitter.html`
[4]`http://hadoop.apache.org/`
[5]`http://hadoop.apache.org/common/docs/r0.15.2/streaming.html`

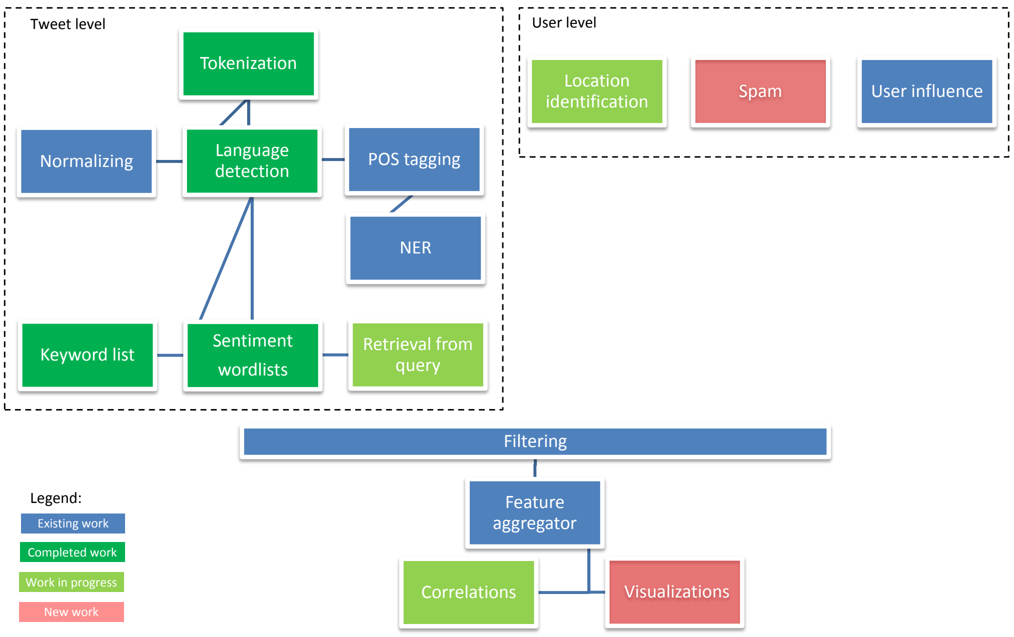

Figure 2.1: Module diagram

tool) or multiple analysis being performed in the single map task (in the Hadoop tool). A diagram of the current stage of the modules, how they relate to each other and with the analysis performed in the next chapter is presented in figure 2.1.2.

Data Consistency Related to modularity is the notion of data consistency. Components of the pipeline may run in isolation, or as a chain of preprocessing tasks. To this end each component must be able to predict what data is available and be able to reuse or reproduce the output of preceding stages it relies upon. Concretely this means that the original twitter status data must remain unchanged and instead the Twitter status JSON map is augmented with an "analysis" entry which is itself a map that holds all the output of the pipeline's stages. Implemented components of the pipeline use this analysis map to retrieve the output of previous stages[6] and they also add their own analysis to this map.

Reusability To guarantee the repeatability of all our experiments, we will make each stage of our preprocessing pipeline available to the wider community in a form which can be easily used to reproduce our results or achieve novel results in different experiments. To these ends the current versions of the both tools are made available in the OpenIMAJ multimedia library[7]. In doing so we allow third parties

[6]A stage has a unique name which it uses to store data.
[7]`http://openimaj.org`

Table 2.1: Example tweet tokenisations

Tweet A	"@janecds RT _badbristal np VYBZ KARTEL - TURN & WINE¡ WE DANCEN TO THIS LOL? http://blity.ax.lt/63HPL"
Tokens A	[@janecds, RT, _badbristal, np, VYBZ, KARTEL, -, TURN, &, WINE, <, WE, DANCEN, TO, THIS, LOL, ?, http://blity.ax.lt/63HPL]
Tweet B	"RT @BThompsonWRITEZ: @libbyabrego honored?! Everybody knows the libster is nice with it...lol...(thankkkks a bunch;))"
Tokens B	[RT, @BThompsonWRITEZ, :, @libbyabrego, honored, ?!, Everybody, knows, the, libster, is, nice, with, it, ..., lol, ..., (, thankkkks, a, bunch, ;))]

to access future releases and module extensions, as well as complete source code access and the ability to preprocess tweets following our methodologies.

2.1.3 Evaluation

In this section we focus on testing the accuracy of the implemented components in the system. In order to produce useful results, our system needs to perform its tasks both quickly and with high accuracy. An evaluation of the running times for our tools is presented in section 2.2.3.

Tokenization

For the tokenization module we will evaluate the performance qualitatively by presenting some tweets that pose tokenization problems specific to microblogging text. Some representative examples are presented in Table 2.1 and we can conclude that our tokenizer handles OSN text very well.

Language Detection

Language detection of short and noisy text has been shown to be a challenging problem. Baldwin and Lui [2010] reports a decrease in performance from around 90-95% down to around 70% with state-of-the-art language detection algorithms when restricting the input text's length.

We test the method that we integrated to our pipeline on the same microblog dataset used by Carter et al. [2012]. They report an accuracy of 89.5% when classifying into 5 different languages. Our accuracy is 89.3% on 2000 tweets using a 97-way classifier. For our setting, in which we want to assign texts to many languages, we conclude that our language identification system performs well, but with room for future improvement. Improvements are suggested in [Carter et al., 2012] where they use microblog specific information to improve accuracy to up to 97-98% when discriminating between 5 languages.

2.2 Standard and Hadoop TwitterPreprocessingTool

In this section we describe, with examples, exactly how the implementations of tools described in the previous sections can be used. This includes an in depth description of the various options of the tools, how various stages of the pipeline can be invoked and how different output modes and filtering operations can be used. In this section we also briefly describe the usage of the hadoop implementation of the tool.

The TwitterPreprocessingTool is an open source command line tool written in java. It allows command line POSIX like access to java implementations of the text preprocessing pipeline discussed in Section 2.1.1. The tool is written with modularity and usability in mind. Behind the scenes, the tool uses modular java implementations of the various pipeline stages which can be used as library functions in java programs independently of the tool. Furthermore, components written for the single machine command line tool are automatically available in the HadoopTwitterPreprocessingTool.

2.2.1 Basic Usage

Listing 2.1: TwitterPreprocessingTool help

```
Usage: java -jar JClusterQuantiser.jar [options...] [files...]
 --encoding (-e) STRING                 : The outputstreamwriter's text encoding
 --input (-i) STRING                    : Input tweets
 --input-file (-if) VAL                 : Get a set of inputs as listed in a
                                          file
 --mode (-m) [TOKENISE | LANG_ID |      : How should the tweets be processed.
 PORTER_STEM]                           :
 --n-tweets (-n) N                      : How many tweets from the input should
                                          this be applied to.
 --output (-o) STRING                   : Tweet output location
 --output-mode (-om) [APPEND |          : How should the analysis be outputed.
 CONDENSED | ANALYSIS]                  :
 --quiet (-q)                           : Control the progress messages.
 --remove-existing-output (-rm)         : If existing output exists, remove it
 --time-before-skip (-t) N              : Time to wait before skipping an entry
 --verbose (-v)                         : Be very loud (overrides queit)
Preprocess tweets for bag of words analysis
```

When ran with no options, help information is displayed as shown in Listing 2.1. In the default mode, input is taken from the standard input and output to the standard output. Therefore, the most basic mode of operation of the tool is that a single pipeline process is selected and ran directly on input from the *stdio*. In Listing 2.2 we demonstrate a simple invocation of the tool using curl and the Twitter API[8].

In this example we make a request to the Twitter API for a stream of tweets. This

[8]https://dev.twitter.com/docs/streaming-api

Listing 2.2: cURL example

```
curl -u $USER:$PASSWORD https://stream.twitter.com/1/statuses/sample.json | \
java -jar TwitterPreprocessingTool.jar \
  -m TOKENISE -m PORTER_STEM -m LANG_ID -q
```

Listing 2.3: cURL example

```
{
  "text": "as␣my␣TL␣keeps␣refreshing␣;;␣taking␣notes",
  "id": 192274076037947394,
  "created_at": "Tue␣Apr␣17␣15:31:26␣+0000␣2012",
  #... OTHER TWITTER DATA ...,
  "analysis": {
    "stemmed": ["as", "my", "TL", "keep", "refresh", ";;", "take", "note"],
    "langid": {
      "confidence": 0.0038440583169374962,
      "language": "en"
    },
    "tokens": {
      "unprotected": ["as", "my", "TL", "keeps", "refreshing", "taking", "notes"],
      "protected": [";;"],
      "all": ["as", "my", "TL", "keeps", "refreshing", ";;", "taking", "notes"]
    }
  }
}
```

stream is fed directly into the TwitterPreprocessingTool which in turn is told to tokenize, stem and language detect each tweet. The output of this example is shown in Listing 2.3. Beyond this basic, though powerful, use case this tool provides a set of input, analysis mode and output options to help better obtain desired results. These are described in further detail below.

Input

The TwitterPreprocessingTool takes input in 3 major ways:

- **stdio:** Input from the standard input. This is achieved with either no options or `-i -`
- **single file:** Input is taken from a single file, this can be achieved with `-i filename`
- **multiple files:** A single file is given with filenames, one per line. Each file is treated as a source of tweets. `-if filecontainingfilenames`

Regardless of input method, each line in the input is treated as a single tweet. By default, every line in all inputs is analysed unless the `--n-tweets N` option is specified in which case N tweets are analysed (across multiple files if specified). The encoding of the data is assumed to be UTF-8 unless specified otherwise using the `--encoding` option. Either raw text can be inputed per line, or json from the twitter streaming API. At present if json is provided, an attempt is made to read the JSON as a TwitterStatus message. In future versions the tool will be extended to support arbitrary data input formats, allowing the specification of the location of the text field in the particular format.

Analysis Mode Selection

At present, 3 key components of the pipeline have been implemented in the TwitterPreprocessingTool:

- **Tokenisation** A java implementation of the tokenizer as mentioned in 2.1.1, accessible through the option `-m TOKENISE`. As output, this step makes a distinction between tokens specifically protected by patterns, and those that were not. This separation is helpful for the stemming mode as described below.
- **Language Detection** A pure java implementation of language detector mentioned in Section 2.1.1 specified using `-m LANG_ID`. Currently the language detector implementation loads a single language model as trained by the algorithm's original authors. For future implementations the construction and specification of different language models will be implemented.
- **Stemming** This mode can be activated using the `-m PORTER_STEM` option. Stemming uses both the language detection and tokenisation steps as part of its operation. The language detector is used to guarantee that the correct stemmer is applied to tokens and the tokenisation step is used to apply stemming only to non-protected tokens. This guarantees that stemming is not incorrectly applied to tokens for which it makes no sense and is in fact damaging to do so. Such tokens include hashtags and urls whose whole initial form is important and must be maintained.

When launched, the tool can be configured to apply multiple preprocessing steps through the specification of multiple `-m` options. For each input, each requested mode is applied in series. When a mode is ran over the text of a given social media item (e.g. a tweet), the output of the analysis is held in a separate "analysis" data field. For each mode selected, specific keys are added to this analysis construct (see Listing 2.3). A result of this approach is that no given mode is ever applied multiple times. For example, if both language detection and stemming are specified using `-m LANG_ID -m PORTER_STEM`, the language detection is performed only once and the output used by the stemming step.

Output Mode

Various output mode options are available, namely:

- **Full mode** in this mode the entire json of the input is maintaned and analysis requested is simply added as another field. This mode can be launched using `-om APPEND`.

Listing 2.4: HadoopTwitterPreprocessingTool help

```
hadoop jar target/HadoopTwitterPreprocessingTool.jar
Option "--mode (-m)" is required
Usage: java -jar JClusterQuantiser.jar [options...] [files...]
 --encoding (-e) STRING                  : The outputstreamwriter's text encoding
 --input (-i) STRING                     : Input tweets
 --input-file (-if) VAL                  : Get a set of inputs as listed in a
                                           file
 --mapper-mode (-mm) [STANDARD |         : Choose a mapper mode.
 MULTITHREAD]                            :
 --mode (-m) [TOKENISE | LANG_ID |       : How should the tweets be processed.
 PORTER_STEM]                            :
 --n-tweets (-n) N                       : How many tweets from the input should
                                           this be applied to.
 --output (-o) STRING                    : Tweet output location
 --output-mode (-om) [APPEND |           : How should the analysis be outputed.
 CONDENSED | ANALYSIS]                   :
 --quiet (-q)                            : Control the progress messages.
 --remove-existing-output (-rm)          : If existing output exists, remove it
 --return-immediately (-ri)              : If set, the job is submitted to the
                                           cluster and this returns immediately
 --reudcer-mode (-redm) [NULL |          : Choose a reducer mode mode.
 DAY_SPLIT]                              :
 --time-before-skip (-t) N               : Time to wait before skipping an entry
 --verbose (-v)                          : Be very loud (overrides queit)
Preprocess tweets for bag of words analysis
```

- **Partial mode** in this mode, selective components of the original are maintained plus the analysis. By default these selective components are the "id" and "created_at" date. This mode can be selected using `-m CONDENSED` and components to maintain can be selected using `-te component1 -te component2` etc.
- **Analysis mode** in this mode only analysis data is outputted, all other context information is lost. This can be selected using `-m ANALYSIS`

By default, all output is piped to the standard output, output files can be set using the `-o location` option. Existing output in the specified location will not be overwritten unless specified using `-rm`.

2.2.2 Hadoop implementation

The tool described in this section has been written in a modular way, detaching input, processing and output of the data from each other. This allowed the relatively simple step of the development of a version of the tool that works against the Hadoop Map/Reduce framework. Through Hadoop's implementation of map-reduce and the distributed file system HDFS, users can take advantage of multiple machines to apply a given set of preprocessing stages to large datasets of social media items in a scalable way.

This is actualized in the HadoopTwitterPreprocessingTool. As shown in Listing 1, the options are mostly identical to the TwitterPreprocessingTool. In fact both tools are backed by identical code, the important difference being only in how the user runs the tools. By

Table 2.2: Number of tweets (in millions) analyzed and created in an hour. Analysis performed: tokenization and language detection

Time	Standard	Hadoop	10% Twitter	Total Twitter
1 hour	0.51	7.6	1	10

using `hadoop` instead of `java` and specifying data input and output as locations which exist on the HDFS in the `-i` and `-o` options, the tool runs across multiple machines and takes advantage of machine parallelism to handle larger datasets. Timing improvements which we have registered using this tool on our relatively small cluster are discussed in further detail in the next subsection.

2.2.3 Running times

The results in Table 2.2 show timings of both our Standard and Hadoop tweet preprocessing tools. Both experiments were run on tweets generated in one day on October 10th, 2010. The local tool was run on a single core whilst the Hadoop tool was run on a Hadoop cluster of 6 machines, totaling 84 virtual cores across 42 physical cores. Our timings show that on our relatively small Hadoop cluster our tools can preprocess tweets in the order of those created in a single day on Twitter in total [9], making our tools appropriate for analyzing the tweets we have access to in real time. More importantly for our purposes, we can easily analyze 10% of the tweets generated per hour in under 10 minutes. Furthermore, due to Hadoop's ability to scale with the addition of new machines, we believe that the addition of a few machines will allow our tools to scale easily as Twitter grows in popularity.

Using the Hadoop tool we have extracted and preprocessed all tokens from 2010 and used these in our analysis in Chapter 3.

[9] `http://techcrunch.com/2011/10/17/twitter-is-at-250-million-tweets-per-day/`

Chapter 3

Initial Analysis

The modern world is filled with various multimedia sources of information, regularly updated with increasingly staggering amounts of rich data which, to a large extent, have not been exploited. Amongst these sources, social media communities (Blogs, Twitter, Facebook, etc.), have seen increasing user uptake and have therefore attracted a great deal of interest from the research community during the last decade. Amongst the many use cases of social media [Marwick, 2011], users take time to express opinions towards celebrities, topics of interest as well as current events. This kind of contemporaneous creation activity results in a potentially powerful resource for mining information and opinion about real world events. In this chapter we present preliminary work conducted towards the integration of textual data sources with financial and political predictive models. The rest of this chapter is organised as follows. In Section 3.1 we outline our approach to using textual information sources to improve predictive financial data models. This includes an exploration into the existing approaches, our strategy for dealing with large scale textual data sources, an exploration of implemented tools and frameworks and initial results from a simple linear regression model. In Section 3.2, our approach to mining political opinion is described, including problem statement, algorithms, and experimental results.

3.1 Financial Markets

In this section we outline Southampton's initial work towards the development of regression models of trends in streaming data. Towards this, we explored methods to analyse data from sources and in formats which we expect to address in Trendminer. This includes work towards: storing and analysing archival textual media streams in a scalable way as well as tools for acquiring and representing text and financial data streams. To these ends, an in house hadoop map-reduce environment was expanded with more machines and more storage capability. The cluster was also modified to allow the analysis of split-able compressed text data. On this environment's distributed file system, 3 years of tweets[1] are stored in a compressed, split-able format, providing the test data for our initial experiments and explorations.

Beyond these preparation activities, we have implemented some basic, yet extensible, frameworks for the representation, preprocessing and analysis of time series data. This includes efficient map-reduce text token preprocessing algorithms; data structures for time series representation; synchronised collections of time series; smoothing and interpolation of time series as well as some initial work on linear regression models of synchronised time series.

The structure of the rest of this section is as follows. In Section 3.1.1 we explore existing work. The field of econometrics is clearly vast, so we mainly concentrate on approaches which attempt to integrate textual time series data with financial data models to improve predictions in some way. In Section 3.1.2 we explore the data we hope to process including discussion on: sources, representations and processing techniques for both textual and financial data streams. In Section 3.1.3 we describe and evaluate some initial regression techniques, namely the construction of a time lagged linear regression model using Ordinary Linear Regression to learn model parameters from data. We show that by involving contextual text as an independent model variable we estimate model parameters which produce less error than using historical financial data alone.

3.1.1 Background

Under the assumption of a semi-strong Efficient-Market hypothesis (EMH) [Fagan, 2009, chapter 11], the current market price is a direct reflection of all current events, historical events and past and present price information. Behavioural finance challenges this notion by emphasising the importance of behavioural and social factors, including overall social mood with regards to given companies or the economy as a whole as the main variable which effects market price. Though it has been argued that these perspectives are contradictory, many researchers have successfully incorporated corpuses of secondary information into economic models with both ideologies as starting points. The overarching message is that incorporation of secondary sources of information[2] can improve the predictive ability of economic models, whether the intuition is that: “markets are in fact a direct reflection of this information[3]” or “social sentiment is the main market effector”[4].

The importance of such contextual secondary information has not escaped past econometric research. Traditional sources of such secondary information include the Gallup Well-being index[5] or the University of Michigan Consumer Sentiment Index as a measure of social sentiment; or investment newsletters of curated financial information [Da

[1] 10% twitter data from the Garden Hose stream care of Sheffield

[2] those beyond historical prices and indexes

[3] assuming EMH

[4] assuming Behavioural economic theory

[5] `http://www.well-beingindex.com/`

and Engelberg, 2010]. Though useful, such data is expensive to curate and as an alternative secondary information can also be found from both financial and non-financial text sources which discuss current events, opinions etc. To our knowledge there exist 2 main sources from which authors have incorporated information aggregated from textual content into financial models. In the past main source was the news[6] and in recent years social media sources. In their recent work [Mao, 2011] compare these sources, as well as search engine data and traditional surveys, in terms of their financial data correlation and predictive ability. In the rest of this section we highlight various interesting works in the area of textual information integration with with financial models.

Several authors have attempted to integrate traditional news sources into financial models. A popular approach is to derive some notion of sentiment of the economy as a whole or sentiment towards specific companies and combine this sentiment metric with historic market price values in a linear regression model. Tetlock [2007] use the General Inquirer (GI) [7] system to encode news reports from the Wall Street Journal. They perform a PCA against the 77 categories to which words can belong in the GI model and choose the first principal component of this analysis as their metric. They define this derived metric as being some notion of "pessimism" and show that it improves model predictive ability over pure linear regression without text, as well as the simple inclusion of GI categories such as Negative or Weak. In a similar vein, Tetlock and Tsechansky [2008] uses GI's notion of Negative and Positive words to incorporate information from textual sources. Choosing news articles related to companies in the S&P index from a mixture of the Dow Jones News Service and the WSJ, a ratio of the negative words to the positive words is used as part of a linear regression with historic time data. They show negative sentiment derived from news sources at points in the past can predict low earnings for individual companies. Going beyond these simple linear models Schumaker [2009] experimented with several regression models using SVMs. They show that regression using past price is improved upon through the involvement of textual data. Their study is interesting because they don't attempt to extract sentiment, and instead learn directly from bags of words, identified noun phrases and identified named entities which they extract from news articles using the Arizona Text Extractor (AzTeK).

Comments and articles on various kinds of online social media provide an easily collectable, extremely large, regularly updated corpus of textual information which has the potential to correlate with financial time series. It is therefore of no surprise that authors have attempted to integrate these sources of information, opinion and sentiment into their financial models. An early example by [Antweiler and Frank, 2004] attempted to explore the financial significance of text from the Yahoo and Raging Bull financial message boards. Using Naive Bayes and SVM based classifiers they coded forum messages as bullish, bearish or neutral. They showed correlation and predictive ability between these variables, trading volume and volatility. Blogs have also served as a rich sources of information, authors have successfully extracted sentiment towards particular companies from

[6]either traditional or web

[7]`http://www.wjh.harvard.edu/~inquirer/`

financial blogs [O'Hare et al., 2009]. Zhang [2010] show that blog, twitter and news polarities extracted from documents referring to particular companies correlate well with stock return for those companies. This correlation varies in potency between industries and reduces in potency as polarity is measured further in the past with respect to return. Similar studies and results have also been shown for other social media sources including constructing financial models by: analysing anxiety, worry and fear on LiveJournal [Gilbert, 2010]; calculating Gross National Happiness using Facebook [Karabulut, 2011]; and measuring mood using Twitter [Bollen and Mao, 2011].

3.1.2 Data

The primary effort by Southampton to date has been the preparation infrastructure to handle textual and financial test data. These preparation efforts were split between frameworks to analyse and process time series data and techniques for dealing with the extremely large scales of textual data proposed to be integrated with financial models.

A set of java tools and classes have been written for the modelling and processing of time series. We have developed efficient data representations of arbitrary events over time, as well as techniques for smoothing, averaging and clustering time series. Beyond this we have also developed data models for extensibly processing synchronised time series. This is the notion that two time series (e.g. words and prices) can be measured over the same time period and can therefore be meaningfully processed together in some way. Using this notion, regression and statistical processors have been implemented. Example usage of these data models and processes are shown below in the sections where example text and financial time series data is presented.

The issue of data scale is inherent with all proposed data sources of Trendminer, regardless of whether the data is in the form of: realtime streams or bulk archival data; news sources or blogs; or from social media networks such as Facebook, twitter or google plus. Due to the nature of this data, the primary concern of any algorithms or modelling techniques developed during Trendminer must be the ability to deal with data on extremely large scales in a timely manner. To effectively build such scalable algorithms, it was required that we build a large scale testing infrastructure. It was important that this infrastructure not only be able to deal with large scales "theoretically", but to also permit practical tests within reasonable timescales against datasets of realistic sizes. Only with such a test systems can an honest effort be made towards generation of truly scalable algorithms. On this note, in the tests conducted below, we use Twitter as our primary textual data source. However, we are confident that our approaches will extend well to other data sources (news articles, Wikipedia, Youtube comments etc.) as we believe the challenges they pose (scale, fuzzy and incomplete data etc.) are mirrored well by twitter data.

In the rest of this section we describe our data testing infrastructure, the source, format and preprocessing of our textual data and also the source of our financial data.

Infrastructure

Our initial experiments have attempted to address problem of data scale by using a hybrid MapReduce framework based on the Apache Hadoop software framework. The MapReduce framework defines a model for distributed computation over large volumes of data. One of the key ideas of the framework is that data is stored on a distributed filesystem across all the nodes. This allows data transfer to be minimised by pushing the computation to where the data is stored, rather than pulling data across the network from the storage to the compute node. It is assumed that the data is in the form of records which are stored as key-value pairs. The keys and values could be anything from chunks of text to binary data such as images.

We have extended our in house hadoop cluster to consist of 6 nodes, 3 with 16 virtual cores and 11Gb of RAM, 2 with 8 virtual cores and 32Gb or RAM and 1 machine with 24 virtual cores (4 of which are not used by mapreduce) and 16Gb or RAM. Spread across the machines are 52Tb of storage space mounted on a distributed file system (HDFS). We have installed Cloudera's latest stable version of hadoop[8] as well as some modifications such as Twitter's hadoop LZO module[9] which allows for compressed, yet split-able files. This setup allows us to run 84 map tasks in parallel, though fewer can be run depending on task memory requirements.

Text Data - Twitter 2010

For our tests, we have decided to model financial activity in 2010. To this end we have loaded all garden hose tweets from 2010 onto our cluster. The year is held in the form of 1 file per day which in their lzo compressed format, total to 782Gb for 2010[10]. Each file contains the tweets for a single day held as twitter stream statuses [11], the format our tools described in Chapter 2 were designed to deal with.

As the first stage in preparing this data for analysis, tweets were language detected and tokenised using the HadoopTwitterPreprocessingTool. Only tweets detected as english were considered in our initial linear model. Furthermore we chose not to stem the tokens in our initial analysis to maintain process simplicity. That is to say, though we expect better results if words deemed to *mean* the same thing were clustered in terms of count in a particular time period, we believe the most useful baseline results to explore initially are those which take the least amount of effort to reproduce and yet have the potential to produce reasonable results. Furthermore, to save disk space of the processed output, we specifically dropped certain components of the original twitter status, holding only our generated analysis, the tweet id, the original text and the date of creation. For reference, the command using to launch the HadoopTwitterPreprocessingTool can be seen

[8] `http://www.cloudera.com/`

[9] `http://bit.ly/JfrJHm`

[10] Each day comes to between 1Gb and 4Gb, some days data missing due to various technical issues

[11] http://bit.ly/TWTRSTAT

Listing 3.1: Command used to preprocess tweets from 2010.

```
hadoop tool.jar -m TOKENISE -m LANG_ID \
        -om CONDENSED -te id -te created_at -te text \
        -ri -rm -i <tweets location on HDFS>
```

in Listing 3.1

This command took between 1 to 3 hours per day of tweets (depending on the volume of tweets in that day), the year was analysed within the space 6 days over which roughly 2.3 billion tweets were processed and emitted by the map reduce tasks. This process represents the most expensive portion (so far investigated) of the analysis algorithm. Note that these timings complement those in Chapter 2 and show the tweets generated in a single day can be tokenised and language detected extracted in well under a single day, meaning these algorithms are potentially executable on chunked streams of data in real time. These analysed tokens are the source of data for the exploration of political data in Section 3.2.

Tokens, though interesting, cannot be used alone for regression analysis. A numerical representation of each textual token is required which gives some notion of its relative importance at a given point in time. By representing the importance of a token through some metric it is a relatively simple step to building selective tokens into a linear regression model. One simple technique would be the count of a given token's occurrence within some time period. We have developed a map reduce implantation of this simple occurrence counter.

Though simple, this metric has a major issue with regards to the relative importance of words. Common words (stop words etc.) may appear at a high rate through all time periods. Also, no special distinction is made for values given to a token w_i that appears x times throughout all time periods $t \in [1, 2, ..., T]$, as compared to a token w_j which also appears x times but only in a single time period $t = \tau$. Arguably at τ the token w_j should be rewarded higher significance than w_i even though technically they appeared the same number of times. A metric which encapsulates this notion well in document analysis is the TF-IDF metric which states that the importance of a given term w_i in a particular document d is a function of its count in a given document $tf(w_i, d) = |\{t \in d\}|$ multiplied by a weighting factor of its relevance across ALL documents which can be measured as the inverse of the fraction of documents which contain the term w_i i.e. $idf(w, D) = log(|D|/d \in D : t \in d)$.

In their paper on event detection in tweets Weng et al. [2011] note that in the context of twitter, the exact count of a term in a given tweet is not of consequence given the relative short nature of a tweet. Instead what matters is the number of tweets containing a particular term in a particular time period, in this formulation a time period can be thought of as a document in the definition of TF-IDF. To this end, they recommend a metric called DF-IDF which is described in Equation 3.1.

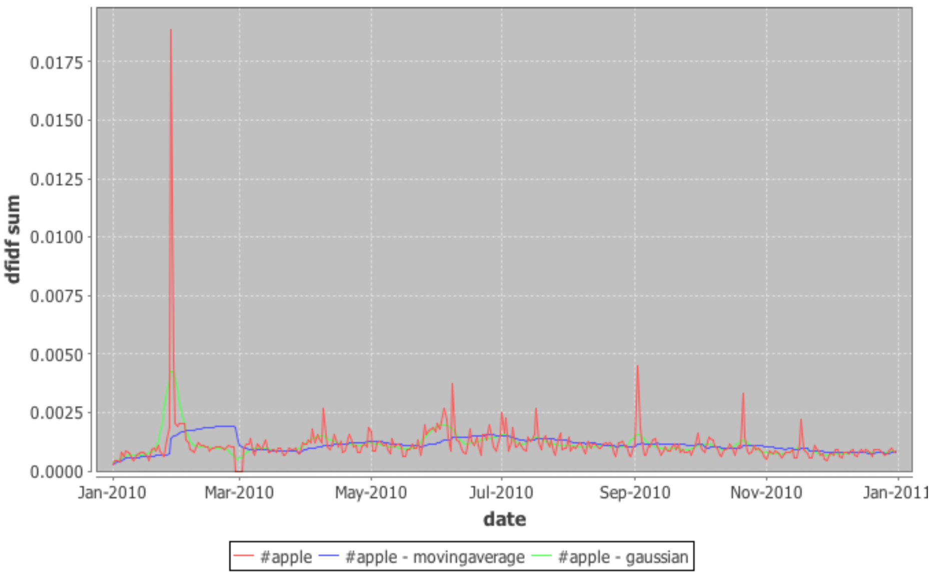

Figure 3.1: Example time series generated form the DF-IDF tool. The peak coincides with apple's main news event in 2010 announcing the iPad

$$DFIDF(w_i, T_c) = \frac{N(w_i, T_c)}{N(T_c)} \times log\left(\frac{\sum_{j=1}^{T_c} N(j)}{\sum_{j=1}^{T_c} N(w_i, j)}\right) \tag{3.1}$$

This formulation states that the DF-IDF metric of a given term w_i at a time T_c is a function of the ratio of the number of tweets which contain that term $N(w_i, T_c)$ and the total number of tweets at that time $N(T_c)$ weighted by the inverse of the ratio of the number of times the word has ever been seen up to that point in time $\sum_{j=1}^{T_c} N(j)$ and the total number of tweets seen up to that point in time $\sum_{j=1}^{T_c} N(w_i, j)$. The rational here is that if a term hasn't been seen often before T_c and is suddenly observed many times, it will get a higher score than a term that has been seen many times in the past. Also scores are measured as a ratio to the number of words seen, which normalises for varying amounts of tweets across time (heteroscedasticity).

We have implemented a map-reduce tool which given a set of tokenised tweets from our previous tool can generate a DF-IDF time series for all distinct tokens, or a predefined set of tokens [12]. By selecting only the english language tweets of the previous tool, this DF-IDF token analysis tool can create this DF-IDF metric for all 963 million english tweets of 2010 in 3 hours. From this process we can generate time series as in Figure 3.1. This diagram also shows some of the time series processing we have implemented including a gaussian processor and a moving average of 30 days.

[12] in our tests we use words likely to be related to the AAPL stock

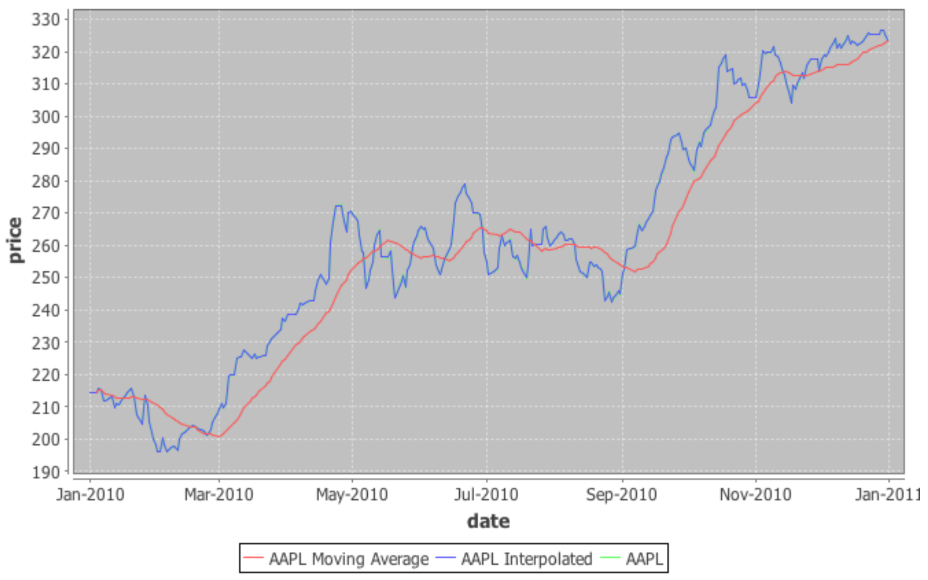

Figure 3.2: Example time series generated from the Yahoo Finance API for the AAPL stock daily High value in 2010

Financial Data

As compared to the extraction of political data as time series, the acquisition of financial data is relatively trivial, easily achieved thanks to the various web APIs[13] available which allow access to historical financial data and stock tickers. We have implemented a module which encapsulates some of the functionality of the Yahoo! Finance Stock API. We've constructed a class which given stock name, a beginning and end date we can get the opening value, high value, low value, closing value, trading volume, and adjusted close value of the stock. This data is provided for every day of trading[14] between the two dates, sources of lower higher granularity (i.e. by hour, minute or second) are available but are not required for this initial analysis. As with the DF-IDF scores we can represent these financial scores in our timeseries framework, a rendering of which we show in Figure 3.2. Note that here we show the moving average as well as using timeseries interpolation to estimate values for weekends and holidays for the year 2010, which at this resolution match those of actual values very closely.

3.1.3 Linear Regression for AAPL

Here we describe Ordinary Least Squares, the model we have selected to model and predict financial time series. A classical model for linear regression is to represent some

[13]Yahoo - `http://finance.yahoo.com/`, Google - `http://www.google.co.uk/finance`

[14]The markets close on weekends and public holidays

dependant variable as a linear weighted sum of a set of independent variables, plus some constant.

$$y = \mathbf{x}\mathbf{b} + b_0$$

The constant b_0 can be blended in with the rest of $\mathbf{b}$ assuming the independent variables are augmented with an extra 1 entry. To construct a predictive model from data, the task becomes learning the values for the weightings $\mathbf{b}$; as given $\mathbf{b}$, we can estimate values of y for a novel observations $\mathbf{x}$ (i.e. estimate stock price given term scores and historic prices for example). This can be achieved by formulating a set of observations of the dependant variable $\mathbf{y}$ and an associated set of observations $\mathbf{X}$. In Ordinary Least Squares (OLS) we define the Sum of Squared Error (SSE) for a given $\mathbf{b}$ as:

$$\begin{aligned} SSE(\mathbf{b}) &= (\mathbf{X}\mathbf{b} - \mathbf{y})^2 \\ SSE(\mathbf{b}) &= (\mathbf{X}\mathbf{b} - \mathbf{y})(\mathbf{X}\mathbf{b} - \mathbf{y})' \end{aligned}$$

This is a quadratic expression and since the value for $SSE(\mathbf{b}) > 0$ by differentiating this function, setting the differential to 0 and rearranging with respect to $\mathbf{b}$ we can create an estimate for the $\mathbf{b}$ which minimises the sum of squared error from between true values $\mathbf{y}$ given some observations $\mathbf{X}$.

$$\mathbf{b} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$$

Because the inversion in the first half of this equation can become numerically unstable, a common approach is to apply the *pseudo-inverse* using the singular value decomposition.

$$\begin{aligned} \mathbf{X} &= \mathbf{U}\mathbf{D}\mathbf{V}^T \\ \mathbf{b} &= \mathbf{V}\mathbf{D}^{-1}\mathbf{U}^T\mathbf{y} \end{aligned}$$

This fairly simple linear model serves as the basis for our financial time series model. Given the above, y becomes the price we hope to predict, but $\mathbf{x}$ can take various values. For example, a simple model may state that $\mathbf{x}$ is simply time (i.e. $\mathbf{x} = \{t_y\}$), and that the price of a stock y is dependant only on time. We choose another approach which says the independent variables $\mathbf{x}$ are historic values for a given price. That is to say:

$$\begin{aligned} \mathbf{x} &= [1, price(t-m-n), ..., price(t-m))] \\ price(t) &= [1, price(t-m-n), ..., price(t-m))] \cdot \mathbf{b} \\ &= b_0 + b_1 * price(t-m-n) + ... + b_n * price(t-m) \end{aligned}$$

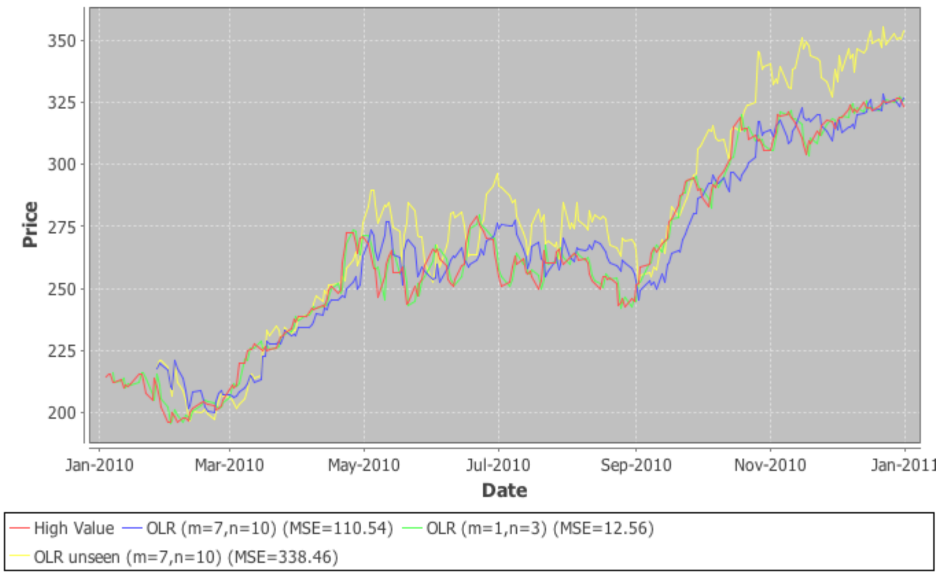

Figure 3.3: End of Day high value for the AAPL stock regressed using various Ordinary Linear Regression (OLR) models of historic time.

Where t is a time that must be estimated, m is some offset into the past and n is a window size of prices in the past[15]. This is equivalent to predicting a price m days into the future using the prices from n days in the past up until now.

In Figure 3.3 we show the results of learning such a model from only historical financial data. In this figure we show both varying window sizes and offsets and also the effect of learning the model on all the data vs a subset. In these experiments, in the unseen case, the subset selected for training are prices from the first 4 months of the year (i.e. all prices from january 1st until May 1st). Note that the further the window selected from the past, the less accurate the regression. Also, the model learnt on data from only the first half of the year is worse at predicting the latter half than the model learnt on the whole year. Though visually obvious, we enumerate this difference using the Mean Squared Error (MSE) between the predicted price and the actual price.

Extending the ordinary least square model to include more information is a simple matter of including more variables into $\mathbf{x}$ and providing appropriate training data. With this in mind it is trivial for us to construct an OLR model for the observation of any number of words and historic prices. In Figure 3.4 we show the results of learning such models from data. Specifically we use the DF-IDF techniques described in the previous section to generate time series for various words which are related to the AAPL stock, specifically: “apple”,“#apple”, “iphone” and “ipad”. In our regression results we show varying window sizes, offsets and training data, using the same window configurations for extracting data from the DF-IDF time series as the historic high value time series. Note

[15]e.g. if we set $m = 2$, $n = 3$ and t =July 30th, we mean that the price on July 30th is a function of the prices from July 25th to 27th inclusive.

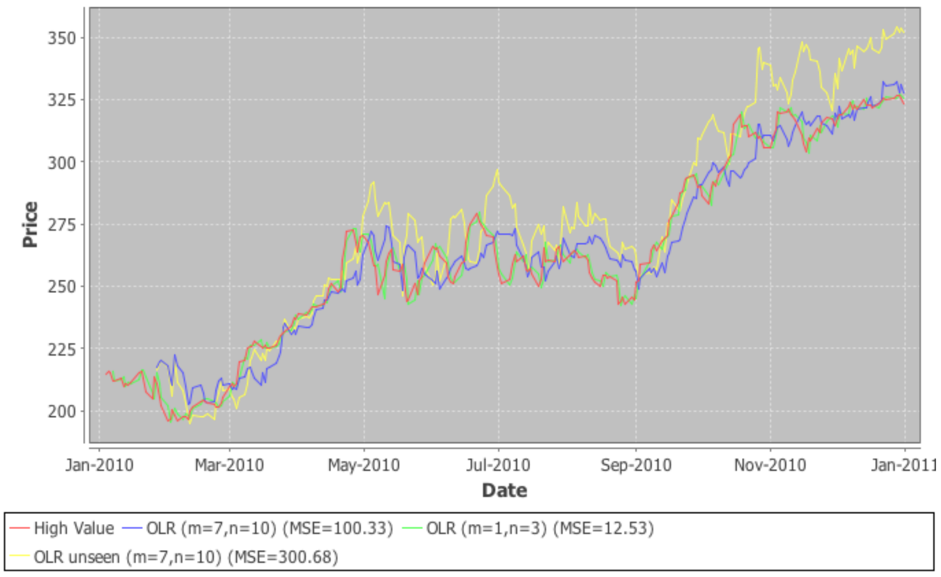

Figure 3.4: End of Day high value for the AAPL stock regressed using various Ordinary Linear Regression (OLR) models of historic time and DF-IDF scores.

that across all configurations, the involvement of the occurrences of the selected words do indeed improve price predictions. However, the amount improved varies greatly with window configuration.

3.1.4 Conclusion

In this section we have presented Southampton's initial work on integrating large scale textual information sources into financial models. We presented an investigations into current approaches to this problem and attempted to implement an end to end textual information integration procedure which simulates some existing approaches. Firstly, we have developed a set of scalable tools which, given the tokens produced by the tools discussed in Chapter 2, allow simple counting and more sophisticated DF-IDF encoding a token's importance on an individual day. We have also produced tools which allow easy access to historical financial information; data models for time series as well as a framework which allows various kinds of fundamental time series processing. Finally, we deliver a linear regression model which uses the time series representations of stock activity and token activity to improve the prediction of stock prices through the integration of textual information on top of historical financial data. All software described in this section is made available on the project git repository[16] mentioned in the introduction section.

[16]`http://github.com/sinjax/trendminer`

3.2 Political Opinion

In this section we highlight Sheffield's initial exploration of political opinion analysis using textual information sources. Public opinion plays a significant role in politics and the media, especially in the run up to an election. A traditional approach to measuring public opinion is a poll. A polling company contacts a group of randomly selected citizens and asks them a series of standard questions that are related to various candidates and issues. The collected public opinion data can then be used to estimate the views of the entire population.

Though useful, such sources of public opinion are expensive to curate. With the increasing popularity of social networks, on which general opinion is expressed by various members of the public, we ask: "is it possible to predict public opinion using textual information aggregated from social networks?" In this section we first review related work that seeks to address this question, and then conduct our own experiments on US and UK political opinion data, using as input the stream of Twitter status messages.

3.2.1 Problem Statement

Our main task is the prediction of polling trends based on sentiment analysis a computational study on public political opinions. This task entails many challenges, particularly the engineering challenge of processing the vast amounts of textual data available efficiently, and in a real-time setting. The amount of streaming status messages from Twitter is immense, with at least 340 million tweets published daily. Even given our limited access to the garden hose stream (10% of all messages), we are currently pulling down 14 gigabytes every day (compressed). As described in Chapter 2, we use the Hadoop map-reduce framework in order to process this data in parallel and in a timely fashion.

Although a huge amount of tweets are generated every day, the tweets related to political topics are limited. Further, the usage of Twitter by UK residents accounts for only a small fraction of the total number of tweets, and other countries in Europe have even smaller footprints.[17] This raises the problem of data sparsity: given that only a few tweets on a given day will relate to domestic politics, how can we make the best user of these? While course filtering approaches may work when data is plentiful, this will not be the case for many of our applications. Related, we are likely to see rapid changes in text frequencies over the course of time, as a simple consequence of our sparse corpora leading to significant effects of noise. Finally, it will be critical in developing our models to select discriminative features, bringing in both domain knowledge and robust computational analysis.

A number of approaches have been proposed for prediction of real valued time-series,

[17] `http://www.techinasia.com/twitter-world/` reports the number of Twitter users segregated by country. UK is ranked 4th with about a quarter as many users as the top ranked US. Spain, France and Germany are much lower with ranks 9, 16 and 18 and less than a tenth as many users as the US.

largely in the natural language processing literature. These have taken the forms of either a carefully engineered heuristic, which is shown to correlate well with the time-series, or some form of regression based model. Based on textual contents, [Yogatama et al., 2011] applied generalised linear models to predict a scientific communitys response to an article, e.g., the number of its downloads after publication in the first year, whether it will be cited or not in the future [Yogatama et al., 2011]. In [Joshi et al., 2010], linear regression was used to predict a movies opening weekend revenue by review texts. Surprisingly, regression models can also be adopted to forecast disease and financial markets. [Culotta, 2010] combined filtering with regression to detect future influenza outbreaks through analysing 500 millions of tweets. Similarly, in [Lampos and Cristianini, 2011], actual rainfall in a given location and time, as well as the regional influenza-like illness rates can be inferred from the contents of tweets by applying a constrained version of ordinary least squares regression. Kogan et al. constructed a regression model of to predict market risk from 10,000 financial reports [Kogan et al., 2009]. [O'Connor et al., 2010] linked text sentiment to public opinion time series using Twitter status messages. Given the strong similarity of their setting to ours, and the simplicity of their approach, we have chosen to replicate their work. We now describe their method in detail.

[O'Connor et al., 2010]'s study modelled the 2010 US presidential race and US consumer confidence. They defined topic keywords by hand to suit their applications: *economy*, *job*, and *jobs* for consumer confidence; and *obama* and *mccain* for the election and presidential approval. Only the tweets containing a topic keyword were used for analysis. Next, the following steps are executed:

1. count the number of positive and negative terms for the tweets on a given day;[18]
2. calculate the ratio of the positive counts to negative counts for each day;
3. smooth the ratio by a moving average $A_t = \frac{1}{k}(x_{t-k+1} + x_{t-k+2} + \cdots + x_t)$ over a window of past k days;
4. finally the smoothed ratios are compared to the time-series data using correlation analysis, then used as input to a regression model.

The last step analyzed the correlation between the daily sentiment ratios x_j and poll outcomes y_t. This was done using a linear least-squares regression model $y_{t+L} = b + a\sum_{j=0}^{k-1} x_{t-j} + \varepsilon_t$, where a is the slope, b is a bias, ε_t is the Gaussian noise, and the lag parameter indicates L days before the poll outcome. The lag allowed the model to be tuned to predict the time-series values in the past, current or future; their work showed that performance peaked when predicting the values in the current week.

We chose to replicate this algorithm, first on their US data and subsequently for the UK. Unfortunately we did not have access to historical Twitter data for the the full period

[18]This used a hand-generated list of words deemed to be positive or negative, as defined in the subjectivity lexicon in OpinionFinder `http://www.cs.pitt.edu/mpqa/`.

of their study, which covered 2 years. Instead we only have a little over two months of data, from 24/07/2009 to 11/10/2009. We applied the Twitter processing pipeline from chapter 2 to this data to perform tokenisation. Next we filtered the tweets using the keyword *jobs*. Figure 3.5 shows the sentiment ratios (step 3, above) with smoothing constant of $k = 1, 3, 5, 7$. Figure 3.6 reports the correlation between the sentiment ratio and Gallup Economic Confidence Index during this period. The correlation reported in [O'Connor et al., 2010] was 73.1% when using a 15-day smoothing window for the same experiment. However, as shown in Figure 3.6, the text-poll correlation we observe is considerably weaker – the correlation coefficient is only 32.2%. We suggest that this is a consequence of the short period of time-series data we used in our experiment, and were we to use the full 2 years of data we would see a different result.[19] In the next sub-section, we apply this method to UK political data to test whether the UK poll trend also can be predicted from text input.

3.2.2 UK Data

Compared with the various US public opinion polls (e.g., index of Consumer Sentiment from the Reuters and Gallup Organizations Economy Confidence index), there are fewer UK political polls available. We chose to use YouGov[20], a polling company that runs frequent polls. Most importantly, they make their full poll data available free of charge and have an extensive archive.

We developed a semi-automated system for crawling the YouGoc archive and pulling down all the poll results, which are in PDF format. We then run a conversion tool[21] to convert the PDF into an Excel spreadsheet. Finally, we automatically extract the poll outcomes from these spreadsheets.

YouGov has a poll frequency of roughly 5 polls per week, although the frequency changes over time, particularly in the run-up to an election. Each poll was conducted on about 30,000 people over the course of two days. Our experiment pays special attention to the pre-election period in 2010, a 5-month period between 01/01/2010 and 06/05/2010. We explore the correlation between the text and the poll outcome using twitter data from this 5 month period analysed using the process described in Sections 3.1.2 and 3.1.2. Figure 3.7 shows the pre-election poll trend of YouGov for three representative parties (Conservative, Labour, Liberal Democrat) in UK, where Con-Lead shows the leading percentage of Conservative. For this figure and in our experiments we used simple window smoothing over the poll outcomes to reduce the noise from spurious results where there are closely spaces polls and to provide a value for days in which there were no polls.

We process the tweets using the standard pipeline for tokenisation and language de-

[19]Note that the index exhibited significant fluctuations during the 2 years, however the index was fairly stable in our sub period.

[20]`http://labs.yougov.co.uk/`

[21]http://www.wondershare.net/pdf-converter/

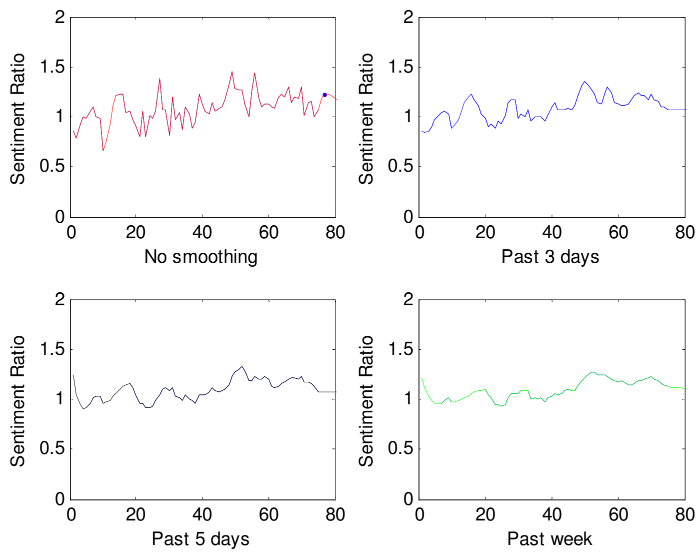

Figure 3.5: Daily sentiment ratio values for tweets containing the term *jobs*. These are smoothed with different sized windows for each of the plots.

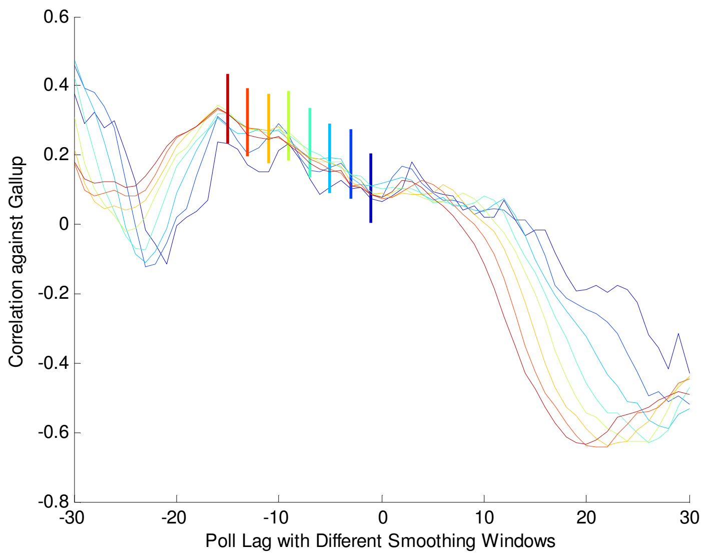

Figure 3.6: Correlation between the sentiment ratio and Gallup Economic Confidence Index. The x-axis shows the lag coefficient, L, which is the day offset between the text appearing and the date when the index value was reported. The different coloured curves are obtained using different smoothing windows, $k = 1, 3, 5, 7, 9, 11, 13, 15$, which can be identified by the vertical bars which are positioned such that $L = -k$.

tection as described in Section 2.2, taking only English language tweets. An example of extracted tweet is shown in Listing 2.3. Note that this filtering meant that our data included tweets not just from the UK, but also from other English speaking countries, in particular the USA. This will have adversely affected the precision of our subsequent filtering and the sentiment statistics derived from the data. We are seeking to address this by developing a method for automatic location detection to provide a more accurate filtering of the data.

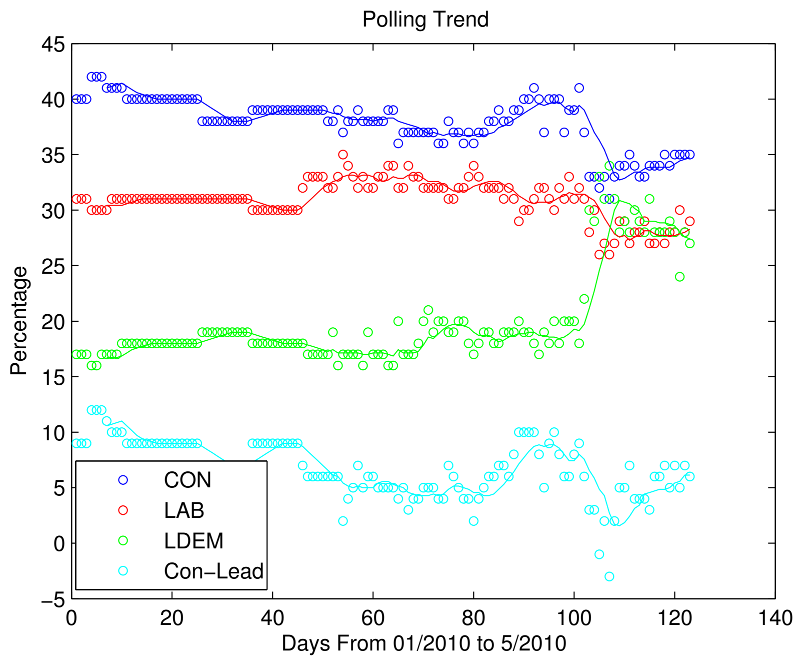

Figure 3.7: The Polling trend of YouGov

In order to adapt [O'Connor et al., 2010]'s method to our data, we first define a list of keywords for each of the three prominant political parties in the UK, namely the Conversatives, Labour and the Liberal Democrats. These terms will be used to filter the data, and the resulting sentiment ratio compared with the poll outcomes for the given party. For the Conservative party (con) we used the terms *david*, *cameron*, *george*, *osborne*, *conservative*; for the Labour party (lab), *gordon*, *brown*, *milliband*, *labour*; and for Liberal Democrats (lib), *clegg*, *liberal*, *democrat*. Also, we experimented with adding some common keywords to the lists to capture general economic sentiment: *jobs*, *job*, *economy*. We should stress that these keyword lists are very important, and that further tuning of these by hand has the potential to greatly improve our results. We then run the O'Connor algorithm on this filtered data, and relate the smoothed sentiment ratios to the YouGov poll outcomes.

In order to efficiently process the large amounts of data present in this corpus, we developed a Map-Reduce algorithm for solving Step 1. This consists of two components:

Table 3.1: Map-Reduce algorithm for counting sentiment words. Here we show the input and output formats for processing batches of full-days worth of tweets.

Mapper	Input: 1. lists of tweet files, each contains tweets for a number of days:tweets_pre1.list, tweets_pre2.list, tweets_pre3.list 2. three lists of keywords for con, lab, and lib, respectively: keywords_con, keywords_lab, keywords_lib

	Output:
	1. a single file is generated for each day, containing tweet file name, topic, positive counts, negative counts and sentiment ratio.
	For example:
	`tweets.2010-02-14.filtered_en  con  19687  9716   2.03`
	`tweets.2010-02-14.filtered_en  lab  21209  10930  1.94`
	`tweets.2010-02-14.filtered_en  lib  19375  9482   2.04`

Reducer	Input: 1. the output from Mapper.

	Output:
	1. three separate files for con, lab, and lib, respectively: results_con, results_lab, results_lib.
	For Example:
	`tweets.2010-02-13.filtered_en  con  22965  11348  2.02`
	`tweets.2010-02-14.filtered_en  con  19687  9716   2.03`
	`tweets.2010-02-15.filtered_en  con  21977  11118  1.98`

the Mapper and the Reducer (see Chapter 2 for details of the architecture). For our task of counting sentiment words for each topic, the task and data format used in the Mapper and the Reducer algorithms are illustrated in Table 3.2.2. The source code was written in Python, and subsequent analysis and plotting of figures was done using Matlab.

After counting the sentiment bearing words, the subsequent steps of the algorithm are conducted and the correlation of tweets and polling results for the Conservatives are given in Figure 3.8. Note that all but the most heavily smoothed curves peak at a lag of about zero (e.g., the green curve). This corresponds to predicting the poll outcome on the same day of the tweets (recall the polling period is only two days). Note also that the correlation between tweets and UK polling data is much better than for the US consumer confidence data, as shown in Figure 3.6. We think this is a consequence of two factors: firsly, we have used more polling data, and therefore have a more reliable estimate of the correlation, and secondly our keyword lists allow for high-precision filtering of the tweets to those relevant to UK politics. We anticipate that with further tuning of these lists – initially by hand, and then automatically – we can further improve performance.

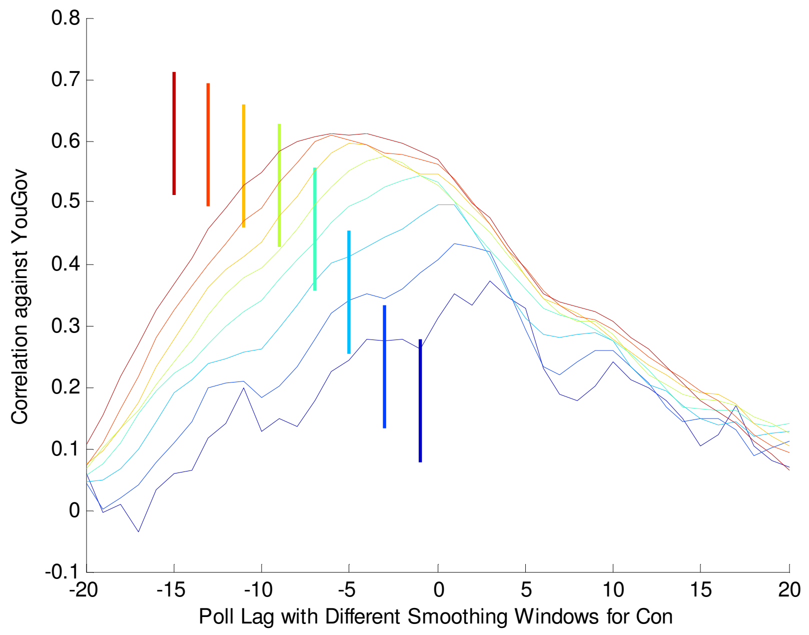

Figure 3.8: Correlation against YouGov

3.2.3 Discussion and Conclusions

Based on the experimental results on both US data and UK data, we found smoothing is a critical step to deal with the high variability problem, where a large k may ignore fine-grained changes of sentiment ratio. Therefore, choosing a suitable k is important for specific tasks using different data. We have shown that the lag parameter in the regression model, is also very helpful for correlating tweets with polls. This allows for tuning the outputs against historical versus current or future data. Although satisfactory results have been obtained on our UK data, there are still some issues that need to be concerned. Location is an important factor in analysing political opinions. According to the geographic boundaries, UK can be divided into 5 regions, namely London, Rest of South, Midlands/Wales, North, and Scotland. As seen in Figure 3.9, the voting intention in different regions is highly divergent, and although they tend to follow similar patterns this is not uniformly true (see the sharp decline in London in early 2012). Separately modelling the voting intensions in each region may help to enhance our overall predictions for each party. Sparse data is also a key problem. Although the quantities of text are staggeringly large, after filtering for UK political terms we often end up with very little data. This is illustrated in Figure 3.10 which shows the frequency of the terms *cameron*, *conservative* and *economy* (some of the keywords we used for the Conservative party). The frequencies vary dramatically between days, which is sometimes due to a speech or news article, or simply a viral tweet being passed around the Twitter network. This second problem is more insidious, as the text frequencies easily become non-representative of the overall opinion of the Twitter users, leading to invalid results. Careful filtering is required to mitigate this effect.

We need to pay attention to the keyword-selection in order to get large numbers of tweets that are relevant to our topics of study, such that statistics derived from this data are sound and have low bias. We plan to develop more robust automatic techniques for disambiguating entity mentions in order to obtain higher recall at finding relevant data. Additionally, we plan to extract other types of complementary features besides the sentiment ratios, and for each learn an independent weight in a regression model.

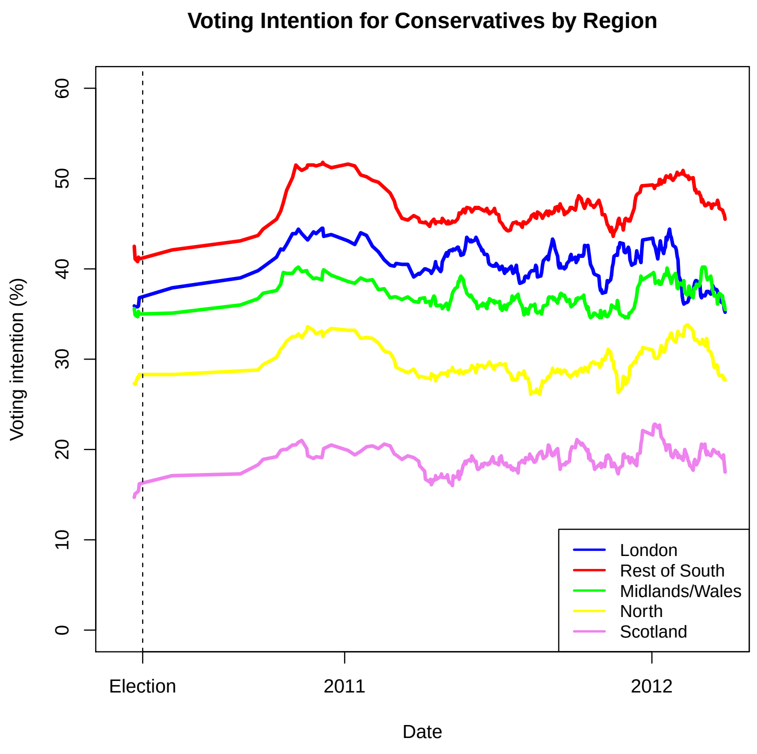

Figure 3.9: Voting intension of different regions for the Conservative party.

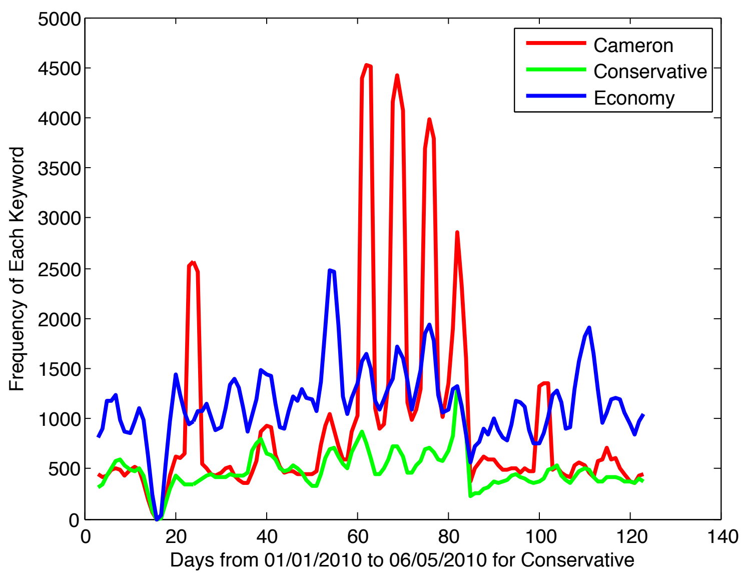

Figure 3.10: Frequencies of the various terms in the run-up to the 2010 UK election. The zero on day 17 is due to an issue with the crawler crashing, so no data was available.

Bibliography

Predicting Financial Markets: Comparing Survey,News, Twitter and Search Engine Data. Dec. 2011.

W. Antweiler and M. Z. Frank. Is All That Talk Just Noise? The Information Content of Internet Stock Message Boards. *The Journal of Finance*, 59(3):1259–1294, June 2004.

T. Baldwin and M. Lui. Language Identification: The Long and the Short of the Matter. In *Proc. NAACL HLT '10*, pages 229–237, Stroudsburg, PA, USA, 2010. URL `http://www.aclweb.org/anthology/N10-1027`.

J. Bollen and H. Mao. Twitter mood predicts the stock market. *Journal of Computational Science*, 2011.

S. Carter, W. Weerkamp, and E. Tsagkias. Microblog Language Identification: Overcoming the Limitations of Short, Unedited and Idiomatic Text. *J. LRE*, 2012.

A. Culotta. Detecting influenza outbreaks by analyzing Twitter messages. In *Science*, pages 594–604, 2010.

Z. Da and J. Engelberg. The Sum of All FEARS: Investor Sentiment and Asset Prices. *SSRN eLibrary*, 2010.

S. Fagan. Handbook of Empirical Economics and Finance - Aman Ullah, David E. A. Giles - Google Books. *Handbook of Empirical Economics and . . .* , 2009.

E. Gilbert. Widespread worry and the stock market. In *Proceedings of the International Conference on . . .* , 2010.

K. Gimpel, N. Schneider, B. O'Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yogatama, J. Flanigan, and N. A. Smith. Part-of-Speech Tagging for Twitter: Annotation, Features, and Experiments. In *Proc. ACL '11*, pages 42–47, 2011. ISBN 978-1-932432-88-6. URL `http://dl.acm.org/citation.cfm?id=2002736.2002747`.

B. Han and T. Baldwin. Lexical Normalisation of Short Text Messages: makn sens a #twitter. In *Proc. NAACL/HLT '11*, pages 368–378, 2011. ISBN 978-1-932432-87-9. URL `http://dl.acm.org/citation.cfm?id=2002472.2002520`.

M. Joshi, D. Das, K. Gimpel, and N. A. Smith. Movie Reviews and Revenues: An Experiment in Text Regression. In *The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2010.

Y. Karabulut. Can Facebook Predict Stock Market Activity? Sept. 2011.

S. Kogan, D. Levin, B. R. Routledge, and J. S. Sagi. Predicting Risk from Financial Reports with Regression. In *In Proc. NAACL Human Language Technologies Conf*, pages 594–604, 2009.

V. Lampos and N. Cristianini. Nowcasting Events from the Social Web with Statistical Learning. *ACM Transactions on Intelligent Systems and Technology*, 3(4):51–56, 2011.

M. Lui and T. Baldwin. Cross-domain Feature Selection for Language Identification. In *Proc. IJCNLP '11*, pages 553–561, November 2011. URL `http://www.aclweb.org/anthology/I11-1062`.

A. Marwick. I tweet honestly, I tweet passionately: Twitter users, context collapse, and the imagined audience. *New Media & Society*, 2011.

B. O'Connor, R. Balasubramanyan, B. R. Routledge, and N. A. Smith. From Tweets to Polls : Linking Text Sentiment to Public Opinion Time Series. In *Proceedings of the International AAAI Conference on Weblogs and Social Media*, 2010.

N. O'Hare, M. Davy, A. Bermingham, P. Ferguson, P. Sheridan, C. Gurrin, and A. F. Smeaton. Topic dependent sentiment analysis of financial blogs. In *Proceeding of the 1st international CIKM workshop*, pages 9–16, New York, New York, USA, 2009. ACM Press.

I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and C. Lioma. Terrier: A High Performance and Scalable Information Retrieval Platform. In *Proceedings of ACM SIGIR'06 Workshop on Open Source Information Retrieval (OSIR 2006)*, 2006.

M. Porter. Snowball: A language for stemming algorithms. 2001.

A. Ritter, S. Clark, Mausam, and O. Etzioni. Named Entity Recognition in Tweets: An Experimental Study. In *EMNLP '11*, 2011.

R. Schumaker. Textual analysis of stock market prediction using breaking financial news: The AZFin text system. *ACM Transactions on Information Systems (. . .* , 2009.

P. Tetlock. Giving content to investor sentiment: The role of media in the stock market. *The Journal of Finance*, 2007.

P. Tetlock and M. S. Tsechansky. More than words: Quantifying language to measure firms' fundamentals. *The Journal of . . .* , 2008.

J. Weng, Y. Yao, E. Leonardi, and e. al. Event Detection in Twitter. *Fifth International AAAI Conference on ...*, 2011.

D. Yogatama, M. Heilman, B. OConnor, and C. Dyer. Predicting a Scientific Communitys Response to an Article. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 594–604, 2011.

W. Zhang. Trading strategies to exploit blog and news sentiment. *ICWSM'10*, 2010.