Norwegian University of Science and Technology

TDT4501 Computer Science, Specialization Project

Better Movie Recommendations with Twitter Data

Author:
Jonas Myrlund

Supervisors:
Prof. Heri Ramampiaro
Prof. Helge Langseth

December 2013

"just setting up my twttr"

The first Tweet ever, by Jack Dorsey on March 21, 2006

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Abstract

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

Master of Science in Computer Science

Better Movie Recommendations with Twitter Data

by Jonas Myrlund

Many movie recommendation systems today use collaborative filtering techniques at their core. An important challenge for collaborative filtering techniques left today is their ability to handle data sparsity. This work attempts to solve the *new item problem* by predicting movie ratings based on the perceived sentiment in Twitter search results.

The system is able to achieve a low MAE for top movies because its predictions happen to average around the average benchmark rating. It is completely unable to consistently distinguish the quality of famous top-rated movies from unknown B-films. Furthermore, it is unable to process around half of the movie titles due to a lack of Twitter search results.

Acknowledgements

I would like to thank my supervisors, Heri Ramampiaro and Helge Langseth, for lots of good advice, while at the same time allowing me a great extent of freedom in planning and executing this project.

I would also like to thank the people behind the DatumBox service, who made this project a joy to implement.

Contents

List of Figures

List of Tables

Abbreviations

MAE	**M**ean **A**verage **E**rror
RMSE	**R**oot **M**ean **S**quare **E**rror
SaaS	**S**oftware **as** **a** **S**ervice
SVM	**S**upport **V**ector **M**achine

Chapter 1

Introduction

1.1 Motivation

Many movie recommendation systems today use collaborative filtering techniques at their core. Although collaborative filtering has many advantages that have let it attain its position as one of the dominant algorithms in the field, there are still quite a few weaknesses left to remedy. Among others, an important challenge for collaborative filtering techniques left today is their ability to handle data sparsity [1].

I will take a somewhat untraditional approach in an attempt to mitigate this issue: can data mined from one of today's largest sources of user-generated content, Twitter, contribute enough relevant information to help in solving the problem of data sparsity?

In this project I will attempt to predict movie ratings based on the perceived sentiment in Twitter search results.

1.1.1 Problems in collaborative filtering

The data sparsity problem has several sides to it [1].

Here, we will take a closer look at the *cold start* problem – or more specifically: the *new item problem*. It occurs when a new item enters the system and there is no rating history to base similarity measures on, leaving a barebones collaborative filtering algorithm without anything to base predictions on – until, of course, some users rate it.

Our evaluation data set does not have movies released more recently than 2005, but as the new item problem applies to the addition of new items into a system, not the novelty of an item itself, this shouldn't constitute a barrier.

Furthermore, many collaborative filtering algorithms have a problem explaining why they come up with their predictions. If we're able to reliably map Twitter content back to our collaborative filtering predictions, we could look into using it as a way of providing context to our recommendations.

1.1.2 Twitter

Twitter is one of the largest sources of user-generated content available today. It launched in 2006, was incorporated in 2007, and has seen active user growth ever since. At the time of writing, Twitter has more than 230 million registered users sending around 500 million Tweets per day.[1]

One of the most interesting things about Twitter is its simplicity. Each message is limited to 140 characters in length, for no other apparent reason than to force its author to formulate messages very concisely, as well as significantly lower the threshold for publishing content compared to traditional blogging services [2].

Furthermore, 76% of Twitter's active users are on mobile – enabling use of the service from anywhere. Combined with Twitter's well-established REST API, this provides us with a robust source of real-time data on almost any subject.

I'll delve further into aspects of using Twitter as a data source in section 2.2.

1.2 Research Questions

In this thesis, I will look for a fit between collaborative filtering's weaknesses and Twitter's strengths. Specifically, I will examine if it is possible to use sentiment extracted from Twitter messages to predict movie ratings.

The main hypothesis is that for each movie, there is a correlation between its user ratings and the sentiment of Twitter messages about it. It is based on a few assumptions:

[1] Numbers from `https://about.twitter.com/company`.

1. Twitter users write positive things about movies they like, and negative things about movies they don't like.
2. Twitter users express this sentiment within the Twitter messages themselves, not just in linked content.
3. When a movie is referenced in a Twitter message the title will most likely be mentioned, and spelled correctly.
4. It is viable to sentimentally classify texts shorter than 140 character, often written in informal language.
5. It is viable to separate between referenced movies, and other entities or phrases with the same title.

As we shall see, many of these assumptions hold up poorly – or hardly at all – rendering the main hypothesis fallacious.

1.3 Overview

This paper is organized as follows.

Chapter 2 surveys relevant literature, similar applications, provides an in-depth analysis of Twitter data and the Twitter API, and reviews the sentiment classifier chosen for the implementation. Chapter 3 describes the system design, and reasoning behind central design choices. Chapter 4 describes the implementation, specifically the way the APIs are called, as well as central algorithms and how they are employed. In chapter 5 we'll look at the execution results, and see how they evaluate. Chapter 6 summarizes the most important takeaways, and suggests some further work.

Chapter 2

Survey

2.1 Similar applications

Some people have attempted the same task as in this thesis, though with another type of social data. Singh et al. [3] investigated a "formulation, where [they] combined the content-based approach with a sentiment analysis task to improve the recommendation results." Their approach is very similar to our approach, but differs in two important ways:

1. It uses user reviews from IMDB as content source[1], and not a more general source of sentiment-carrying content – as in our case, with Twitter.

2. It is generates recommendations based on genre input.

2.2 Twitter data

Micro-blogging services such as Twitter have enormous amounts of data on almost every topic imaginable. Content is limited in length, and users react to each others' content by "re-tweeting", "favoriting" or "replying to" it. This leaves us with a source of textual data that is:

Instant Users express reactions to events as they experience them.

[1]IMDB does not provide open API access at the time of writing.

Weighted Users weigh each others' content by interacting with it.

Concise Due to limitations on content length, users must express themselves concisely.

Now, for a look at the ways in which we are able to access this data.

2.2.1 Relevant parts of the Twitter API

The various endpoints in the Twitter REST API is divided into 16 categories, spanning everything from search and user timelines to suggested users to follow and spam reporting. A full overview of the available API endpoints is available on Twitter's REST API pages[2].

Of these, only two categories are of relevance, namely *Search* and *OAuth*.

OAuth is only relevant because it enables us to search, so we won't elaborate further on how we use it.

2.2.1.1 The Twitter search API

The Twitter search API[3] is a JSON-based REST API, and it takes the following notable parameters:

`q`
: A UTF-8, URL-encoded search query of 1,000 characters maximum, including operators.

`result_type`
: Specifies what type of search results you would prefer to receive.

In addition, there are parameters to control the number of messages returned, language, date ranges etc.

The `q` parameter, the search query, is obviously of great importance, as it is the one we primarily will be using to target our search.

[2] https://dev.twitter.com/docs/api/1.1

[3] https://dev.twitter.com/docs/api/1.1/get/search/tweets

The `q` parameter supports a wide range of operators.[4] It supports the typical boolean operators (AND, OR, and NOT), as well as specifying exact search phrases, but it also supports more domain-specific operators:

From/to/referencing user

Only return messages from or to a specific user, or simply mentioning a specific user somewhere in its message.

Date ranges

Support for "since" and "until" constraints.

Other

Support for things like only returning messages with links, messages submitted via a specific client type, and messages asking a question.

All operators can be combined, and the regular space serves as an AND clause in this regard.

In addition to the aforementioned operators, the API supports returning tweets matching either a positive or negative sentiment – a feature which should seem very relevant to the task at hand. However, no information was seemingly available on the mechanics behind it, and after some manual testing it was concluded that it also lacked quite severely in its precision.

The other parameter central to our application is the `result_type`. It allows us to switch between three possible heuristics for which results that are returned from the search. We have the choice between three valid values: `recent`, `popular`, or `mixed` tweets – the latter being a combination of the first two, and also the default.

To know which one to choose, we need to know what constitutes a popular tweet.

2.2.1.2 What constitutes a popular Tweet?

On Twitter, there are several ways of interacting with the system. Terms like "follow", "mention", "favorite", and "retweet" are all more or less domain-specific to Twitter, so before moving on – let's break them down.

[4]The full list is available here: https://dev.twitter.com/docs/using-search

Follow

Users consume each others' content by following each other. The number of followers users have range from 0 to more than 40 million. Following is a one-way relationship, and there is often a big difference in the number of users *following* and *being followed by* a user.

Reply

Users can mention each other in tweets by prepending a username with "@". This same mechanism is used to reply to others' content. When replying, the content the Tweet was replying to is stored along with the reply, forming a conversation tree.

Favorite

Users can favorite content, which notifies the content owner and boosts the content in search results etc. It is also trivial to extract all content a particular user has favorited.

Retweet

When a user chooses to retweet another user's content, that content is "forwarded" to the user's followers.

One of the greatest benefits of Twitter as a data source is that all these interactions are completely transparent[5], and can thus be used freely when querying for popular content.

2.2.2 Novelty of available data

As briefly mentioned, part of the hypothesis is that Twitter data is usable for predicting ratings for new content, where collaborative filtering systems perform worse than otherwise.

Novelty, however, is an inherently fundamental quality of Twitter data, even as a data source. The Twitter search API simply does not expose data more than about a week old. As it is put in their search API guidelines as of December 2013 [4], "The Search API is not complete index of all Tweets, but instead an index of recent Tweets. At the moment that index includes between 6-9 days of Tweets."

[5]Users can make their own content "private", but their interactions with other users' non-private content are nonetheless public.

This quality makes Twitter data less than useful when it comes to predicting ratings for movies old enough to have calmed in public debate – a category which, not surprisingly, includes the vast majority of available movies.

2.2.3 Twitter as a Data Source

There are vast opportunities in managing to understand a data source with the qualities discussed above, but alas – the diversity and free nature of Twitter as a publishing platform comes at a price: *there is a lot of noise*. That is not to say that there is little relevant information, but when the service is designed to have such a low threshold for contributing content, a low signal-to-noise ratio seems inevitable. We will return to the issue of noise in chapter 5.1, where we try to evaluate correlations between movie titles in our test data and Tweets about the same titles.

This problem of finding content carrying relevant information turns out to perhaps be the biggest challenge of the entire study.

2.3 The sentiment classifier

Much has already been written on the topic of sentiment analysis of Twitter data [5–9]. This paper, however, is not about improving or analyzing these methods and techniques. What is required is a technique that is good enough to *indicate* the sentiment of *a set of messages*.

For sentiment analysis, I have therefore selected the Twitter sentiment classifier provided by DatumBox, a machine learning SaaS. They provide a broad suite of services within many fields of machine learning. See figure 2.1 for an overview of their current functionality within the document classification category[6].

Naive Bayes classifiers in general, as well as details regarding the DatumBox implementation, are described in section 4.3.

[6]The complete list is currently available at `http://www.datumbox.com/api-sandbox/`.

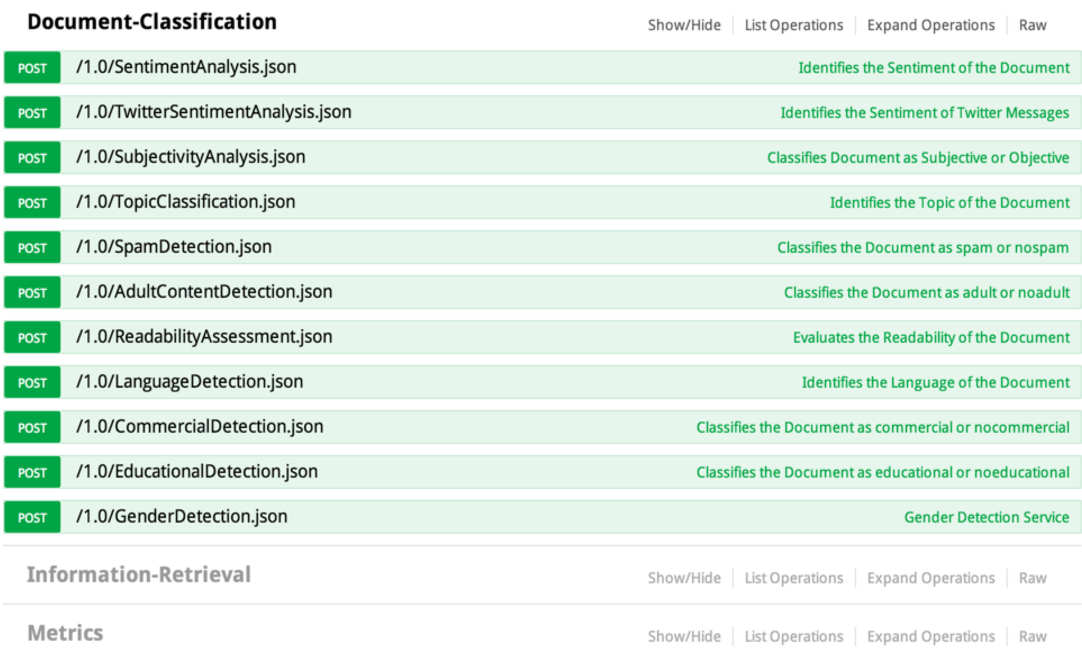

FIGURE 2.1: The current endpoints in the DatumBox API in the category "Document Classification".

2.3.1 Alternative approaches

Two of the most relevant alternatives to the Naive Bayes, SVM and Max Entropy, are also perfectly capable of performing sentiment classification. This section will describe them briefly, with emphasis on how they perform the sentiment classification task. This is well described in the literature [10], so this survey will not go into great detail.

Max Entropy has been shown to perform several text classification tasks often better than the Naive Bayes, but it also sometimes performs worse [11]. The underlying principle of maximum entropy is that without further knowledge, uniform distributions should always be preferred. Training data lay constraints on the distribution, and let us know where the most uniform solutions are.

A Support Vector Machine (SVM) is a large-margin classifier, as opposed to both Naive Bayes and Max Entropy's probabilistic approaches. For a typical two-category case, classifying documents as either positive or negative, SVM attempts to find a hyperplane represented by a vector $\vec{\sigma}$ which not only separates the document vectors $\vec{d}$ of one class from the other, but which also maximizes this margin.

Since none of these approaches are particularly dominant in the literature, the simplest solution – the Naive Bayes classifier provided to us through the DatumBox API – was chosen.

2.4 The Netflix rating dataset

For evaluating the results, the Netflix rating dataset was used as a benchmark.

The accompanying Readme file descibes the dataset in the following way:

> The movie rating files contain over 100 million ratings from 480 thousand randomly-chosen, anonymous Netflix customers over 17 thousand movie titles. The data were collected between October, 1998 and December, 2005 and reflect the distribution of all ratings received during this period. The ratings are on a scale from 1 to 5 (integral) stars.

The data used in this work consists of two different types of files.

First, an overview of available movies is available in CSV format, with lines containing the following fields:

1. Movie ID
2. Year of release
3. Movie title

In a separate directory, 17770 files named by their associated movie ID contain lines of individual ratings, with the following attributes:

1. Customer ID
2. Rating
3. Date

Unfortunately, the Netflix rating dataset is no longer publicly available, allegedly due to a lawsuit regarding privacy concerns[7].

For more details on how the data was used to evaluate results, please see chapter 5.

[7] http://www.wired.com/threatlevel/2009/12/netflix-privacy-lawsuit/

Chapter 3

Design

This chapter will describe the system in a top-down manner. After explaining the initial design requirements, the system viewed as a whole will be described, then the large logical modules and their relationships, then their individual qualities.

3.1 Design choices

Seen as a black box, the system consumes metadata and emits predictions.

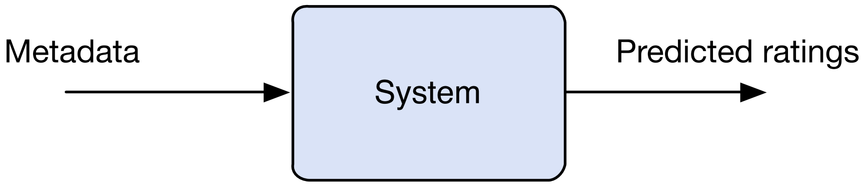

FIGURE 3.1: The system consumes metadata, and emits predicted ratings.

The system is inherently data-driven. Every step consumes data, processes it, and emits data – each conforming to simple interfaces in both ends.

More specifically, the system consists of three logical steps:

1. Data gathering
2. Sentiment analysis
3. Rating prediction

The steps sequentially process data for the next one to use, as illustrated in figure 3.2, and invoke each other through simple interfaces. In the next sections, each step will be briefly described. Implementation details are deferred to chapter 4.

FIGURE 3.2: The high-level system design, depicted in the manner in which data flows through it.

Although the steps below are designed to handle *sets of movies* at a time, to simplify, let us examine how they work for each single movie.

3.2 Data gathering

The data gathering step consumes movie metadata, and emits a list of Twitter messages.

It performs the following two rough steps:

1. Query Twitter for relevant content based on given metadata.
2. Extract textual messages from search results.

As we shall see in later chapters, actually retrieving relevant content from Twitter is quite a bit harder than it should initially seem.

3.3 Sentiment analysis

The sentiment analysis step assigns a sentiment class to each associated Twitter message from the previous step. The messages are labeled as either “positive”, “neutral”, or “negative”.

Other work [8] has included a separate label for messages that were unreadable by the human annotator, but seeing as we’re using a third-party sentiment classifier we are unable to experiment with this distinction.

3.4 Rating prediction

The rating prediction step takes a list of sentiments, and emits a predicted rating value.

To turn the sentiment classes into a single rational number, two steps are taken:

1. Each sentiment class is mapped to a real number value.
2. The real values are reduced into a single prediction by averaging them.

Chapter 4

Implementation

This chapter will go through the algorithms and API calls being used by the application.

In addition to the three steps in chapter 3 – "data gathering", "sentiment analysis", and "rating prediction" – this chapter will also touch upon how the evaluation mechanisms were implemented. However, evaluation results and related discussions are deferred to chapter 5.

4.1 Languages and tools

The system was implemented in the Python programming language, and has no further technical requirements. However, it supports using SQLite[1] to store and query various movie metadata, and to cache content classifications to avoid hitting the remote APIs too much during testing.

The final application is a highly configurable command-line tool. Its interface co-evolved with its requirements, and lets the user control most of its parameters. Figure 4.1 shows the result of running the final program with the `--help` argument.

4.1.1 Important data structures

The only somewhat complex entity being processed by the system is the Twitter messages. These are represented in the system as instances a simple `Tweet` class. It has a

[1] http://www.sqlite.org/

```
usage: core.py [-h] [-t {recent,popular,mixed}] [-n N]
               [--max-tweets MAX_TWEETS] [--threshold THRESHOLD] [--normalize]
               [--top-movies-only] [--plot] [--show-errors] [-d]
               [{mse,compare}]

Parses sentences.

positional arguments:
  {mse,compare}

optional arguments:
  -h, --help            show this help message and exit
  -t {recent,popular,mixed}
                        type of search
  -n N                  number of movies to process
  --max-tweets MAX_TWEETS
                        max number of tweets to process per title
  --threshold THRESHOLD
                        number of tweets required to start sentiment analysis
  --normalize           normalize twitter ratings to match average mean of
                        netflix data
  --top-movies-only     only load top movies
  --plot                plot the results and show them
  --show-errors         plot the standard deviations and other errors
  -d, --debug           print traces and parse trees
```

FIGURE 4.1: Screenshot of the application's help menu.

small set of default values and utility methods, but in practice, the objects behave as dictionaries.

Each `Tweet` object is augmented with additional metadata on its way through the system, until the "rating predicion" step, where they are reduced to real-numbered predictions and discarded.

4.2 Data gathering

As was soon found out after the initial tinkerings with the Twitter API began, the service has a high noise ratio. An example of this is the search results depicted in figure 4.2. A simple search for the movie title "The Usual Suspects" returns only noisy results – none of them express any sort of sentiment about the actual movie.

Due to this noisiness, every opportunity was taken to refine and tweak the search queries used to gather content. Luckily, the Twitter API has a quite extensive search interface, with many ways of tweaking the results (details in section 2.2.1.1).

After much trial and failure, the following settings seem to yield good results for typical well-known movies:

FIGURE 4.2: A simple search for “The Usual Suspects” gives only noisy results.

Title as exact phrase

Ensure that the title is searched for in its entirety, as a singular phrase and not just the individual words. In the Twitter API this is done by enclosing the title in quotation marks, as such: `"The Usual Suspects"`, instead of `The Usual Suspects`.

Exclude noisy terms

Exclude tweets containing the following terms[2]: `download`, `stream`, `#nw`, `#nowwatching`, and `RT`.

Add domain keyword as term

Some movies with a degree of cult status, such as “The Usual Suspects”[3], can inspire usage of their titles as expressions in other contexts. This can lead to noisy results when they are used without referring to the movie. Some examples of this can be seen in figure 4.3. Adding the domain term `movie` to the query rids us of many of these, as shown in figure 4.4. As we will see later, this still does not suffice for a lot of less-known movies.

[2]Any time a set of irrelevant results shared a common term, it would be added to the list. There are probably many ways of fine-tuning this further.

[3]http://www.imdb.com/title/tt0114814/

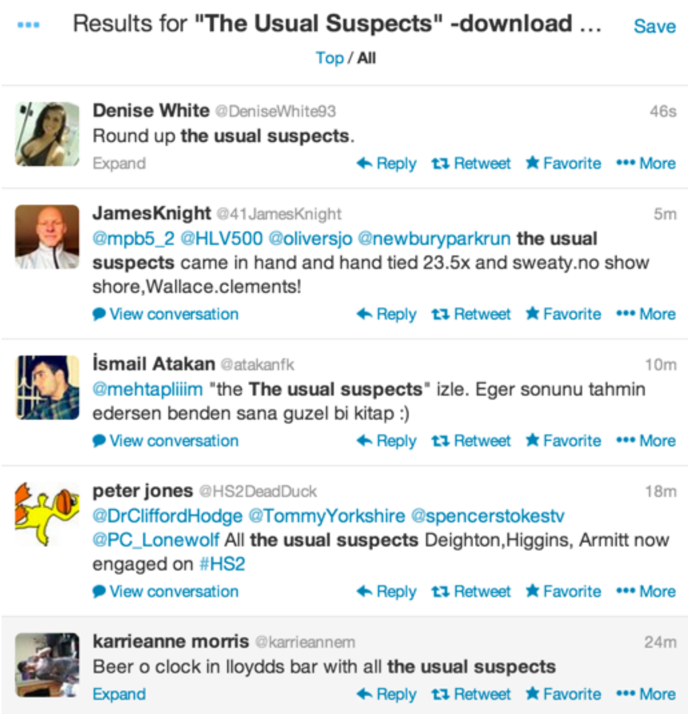

FIGURE 4.3: Without a domain term, many queries still return a lot of noise. This is the result of searching for `"The Usual Suspects" movie -download -stream -#nw -#nowwatching -RT`.

For the choice among the three modes of search – "popular", "recent", or "mixed" – as much diversity as possible was desired. Although popular Tweets were initially preferred, there were simply not enough of them to warrant this choice (more often than not, zero or one). Therefore, to enable a fair comparison between popular and not-so-popular movie titles, the result type of choice became "mixed".

4.3 Sentiment analysis

A central part of the system is the sentiment classification of the Twitter results. This sentiment analysis could well have been implemented locally, but for simplicity's sake it has been offloaded to an external service called DatumBox[4], as it seems to employ a reasonable choice of algorithm, and performs well enough for our needs.

This section will take an in-depth look at the techniques the DatumBox service utilizes.

To get us started, this is how DatumBox themselves describe their classifier [12]:

[4] datumbox.com

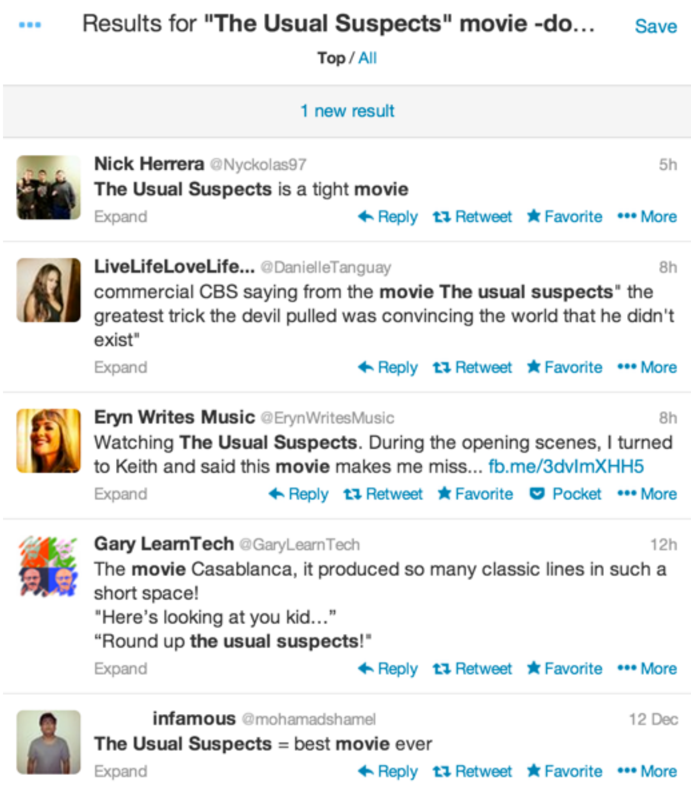

FIGURE 4.4: The results of searching for `"The Usual Suspects" movie -download -stream -#nw -#nowwatching -RT`. A lot of noise has been eliminated, and most results are about the movie in questions.

In order to detect the Sentiment of the tweets we used our Machine Learning framework to build a classifier capable of detecting Positive, Negative and Neutral tweets. Our training set consisted of 1.2 million tweets evenly distributed across the 3 categories. We tokenized the tweets by extracting their bigrams and by taking into account the URLs, the hash tags, the usernames and the emoticons.

In order to select the best features we used several different algorithms and at the end we chose the Mutual Information. Finally after performing several tests with various models and configurations we selected the Binarized Naïve Bayes as the best performing classifier for the particular problem (strangely enough Naïve Bayes beat SVM, Max Entropy and other classifiers which are known to perform usually better than NB). To evaluate the results we used the 10-fold cross-validation method and our best performing classifier achieves an accuracy of 83.26%.

There exists work which supports this results, where Naive Bayes outperforms SVM and Max Entropy in classifying Twitter messages [5].

The next sections will in turn describe feature selection by Mutual Information, and the Naive Bayes classifier.

4.3.1 Feature selection by Mutual Information

When selecting features one needs need a selection criteria. Typical approaches include Max-Dependency, Max-Relevance, and Min-Redundancy [13]. As an example, we'll have a quick look at the Max-Relevance scheme.

Max-Relevance involves choosing the features with the highest relevance to the target class c, and we typically characterize the concept of relevance in terms of mutual information. The mutual information of two random, discrete variables X and Y is given by (4.1).

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)} \tag{4.1}$$

Given a feature vector f of length n, we want its m most relevant features with respect to class c. The relevance of each feature f_i is measured in order of its mutual information with the class c, ie. $I(f_i;c)$.

Thus, to extract the m most relevant features with regard to class c, we simply order them in descending order by their $I(f_i;c)$, and keep the first m elements.

4.3.2 The Naive Bayes classifier

To describe the Naive Bayes classifier we will use the bag-of-features framework [10].

Let $f_1, f_2, \ldots, f_m$ be a set of m features, and let $n_i(d)$ be the number of times f_i occurs in document d. We can then represent each document d as a vector $\vec{d} = (n_1(d), n_2(d), \ldots, n_m(d))$. The training data being "evenly distributed across the 3 categories" [12], the probability distribution $P(C)$ over the possible categories, C, is constant.

We can therefore use the Maximum Likelihood approach to build our hypothesis, and simplify our calculations.

First, Bayes' rule tells us that

$$P(c|d) = \frac{P(c)P(d|c)}{P(d)} \tag{4.2}$$

where neither $P(d)$ nor $P(c)$ play any role in predicting c.

By assuming that the features f_i are conditionally independent given their classes, we can formulate a maximum likelihood hypothesis which can be used to classify documents:

$$h_{\mathrm{ML}}(d) = \operatorname*{argmax}_{c \in C} \prod_{i=1}^{m} P(f_i|c)^{n_i(d)} \tag{4.3}$$

4.3.3 Other issues

When calling the DatumBox API, the best results were achieved when removing the movie title itself from the query. Quite a lot of titles have sentiment-carrying words in their titles[5], and this obviously confused the classifier quite a bit.

4.4 Rating prediction

For rating prediction, the goal is to convert a set of sentiments into a single real number. The test set from Netflix uses integers from 1 to 5 inclusive, so the number should be within that range.

A simple linear solution was implemented, where a single sentiment s is mapped into a real rating r in the following way:

[5] "Breaking Bad" consequently scoring way below "Cheers" was a rather clear cut case.

$$r(\text{Positive}) \to 5$$
$$r(\text{Neutral}) \to 3$$
$$r(\text{Negative}) \to 1$$

The predicted rating $\hat{r}$ of a set of sentiments S of size n is defined as the mean of its individual sentiment values:

$$\hat{r}(S) = \frac{1}{n}\sum_{s \in S} r(s) \tag{4.4}$$

4.5 Implementation of evaluation system

To be able to consistently map the predictions back to the test set, the sample movies are drawn from the test set's own database. This set is comprised of a little over 17 thousand movies, and the only metadata available – apart from the title – is the year of release.

However, it soon became interesting to examine the difference in performance for popular and less-popular movies, and the system thus needed the ability to make this distinction. Therefore, IMDB's top 250 movies[6] were dumped to a CSV file, and the 174 movies that matched ones in the evaluation set were marked as popular. This enabled the system to optionally restrict the drawn test sample to only those present in this list.

Although the movies' metadata is preloaded into a SQLite database, their individual benchmark ratings are stored in CSV files on disk and are loaded on demand for each prediction to be evaluated. There is relatively little to gain from speeding up this process, as the two remote APIs – Twitter and DatumBox – are consulted many times for every prediction made, and are a much bigger bottleneck in this implementation.

[6] http://www.imdb.com/chart/top

Chapter 5

Evaluation

- When rating popular and well-known movies[1] the predicted ratings achieve a correlational coefficient of 0.75 with regard to average Netflix ratings for the same movie.

- B-movies or older less-known movies rarely collect enough Twitter search results to warrant any further analysis.

- Movies with titles that are fairly common words or expressions in their own right achieve very low precision, and often return only noise. This is hard to detect without manual interference. Need to perform some sort of Named Entity Disambiguation, maybe something like the techniques outlined in Cucerzan [14] or Sarmento [15] (is elaboration needed?).

5.1 Evaluation metrics

To find out how the Twitter-based predictions fare, we will simply use the average of the available Netflix ratings of the same titles as benchmark ratings.

To compute error, the MAE (Mean Average Error) metric will be used. With predictions p and benchmark ratings r, we will compute the MAE of N sample movies in the following way:

[1]The sample in question consisted of "Pulp Fiction", "The Shining", "Mission: Impossible", "The Matrix", "The Godfather", "Forrest Gump", and "A Clockwork Orange".

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}|p_i - r_i| \tag{5.1}$$

As a baseline metric, the MAE of the average of all the benchmark ratings is used:

$$\mathrm{MAE}_{\mathrm{baseline}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\bar{r} - r_i)^2} \tag{5.2}$$

To measure the correlation between predictions and ratings, Pearson's r is used. It is calculated in the following way:

$$\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y} \tag{5.3}$$

Pearson's r describes the linear correlation between two variables. It takes on values between -1 and 1, where a value of 1 is total positive correlation, 0 is no linear correlation, and -1 is a total negative correlation.

5.1.1 Why MAE?

Two error metrics were considered for evaluating the system: MAE and RMSE. We specifically consider MAE because of its simplicity, and RMSE because of its widespread use in the domain of evaluating movie recommendation algorithms[2].

First, let's have a look at RMSE. It is calculated in the following manner for N predictions p and benchmark ratings r:

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(p_i - r_i)^2} \tag{5.4}$$

The main downside to RMSE is that it is complex. It varies with three different qualities of the errors, namely

[2] It was the only error metric targeted by the Netflix Prize, whence the Netflix training data originate.

1. the variability within the distribution of error magnitudes
2. the square root of the number of errors
3. the average error magnitude (MAE)

The MAE, on the other hand – as defined in (5.1) – is a lot simpler.

The goal for the evaluation of this application is to enable us to reason over the applicability of the general approach taken, not to objectively compare this specific application to other competing systems. For this, the simplicity and clarity of MAE is prioritized over RMSE's expressiveness, and will serve as the main error metric.

5.2 Evaluations of samples

Predictions were evaluated over two runs, run A and run B: the first with movie titles sampled from the entire Netflix catalog, and the second with movie titles sampled from IMDB's top 250 list. The samples were evaluated in 4 batches of 10 movie titles each.

Single batches of 10 are detailed for illustrative purposes. Their results are consistent with the magnitude of those gathered from evaluating other corresponding samples. These runs, however, we only characterize by their MAE and correlational coefficients here.

Both runs were performed with the same system parameters, apart from the top movie constraint in the second run, the most important ones listed in table 5.1.

Max. document count	At most 25 Twitter documents were analyzed per title.
Threshold	5 Twitter messages required to start sentiment analysis.
Result type	"Mixed"

Table 5.1: Parameters for evaluated test runs.

5.2.1 Run A: Sample drawn from entire test set

This run gives a MAE of 1.08, and a correlational coefficient of 0.39. Details of individual item ratings and predictions are available in table A.1.

These numbers, however, vary greatly between each run. Subsequent runs with other randomly sampled movies yield the following:

1. MAE: 1.16, correlational coefficient: 0.16
2. MAE: 0.62, correlational coefficient: -0.56
3. MAE: 0.84, correlational coefficient: -0.11

Specifically, it is worth noting that the predictions in general grossly overshoot the benchmarks, which becomes especially clear in the comparative line plot in figure 5.1.

For run A, it is worth pointing out that 7 movie titles were discarded for not meeting the required threshold of 5 Twitter search results. On average, approximately 50% of the titles are discarded on this criteria.

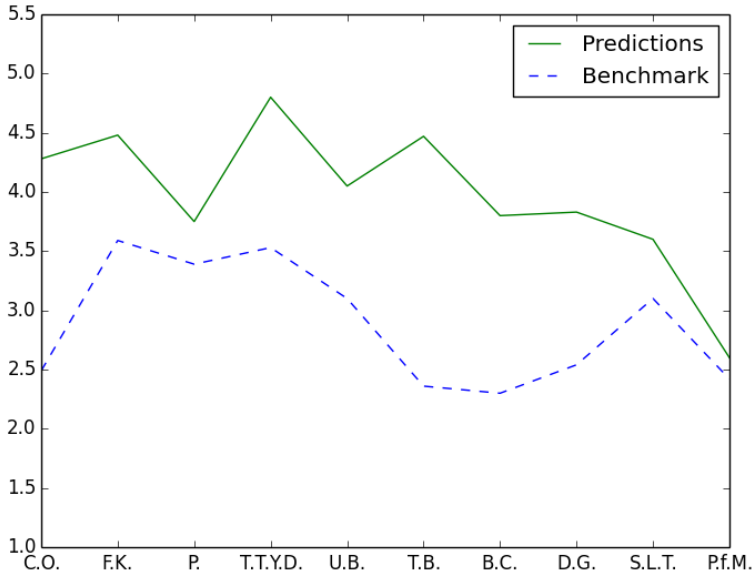

Figure 5.1: Run A: Line plot of predictions vs. benchmark value.

5.2.1.1 Noisy Twitter search results

Most of the Twitter results listed in table A.1 were in fact not about the movie in question. In fact, *all* the search results that were evaluated on behalf of the first movie, "Chill Out", were about watching other movies, only mentioning "chill out" in the same message more or less at random.

The same is the case for several other movies in the listed sample run.

5.2.2 Run B: Sample drawn from IMDB's top 250 list

This run gives a MAE of 0.38, and a correlational coefficient of 0.40. Again, details are available, in table A.2.

Subsequent runs with the same parameters yield the following:

1. MAE: 0.78, correlational coefficient: 0.16
2. MAE: 0.58, correlational coefficient: -0.27
3. MAE: 0.52, correlational coefficient: 0.23

The comparative plot in figure 5.2 is better adjusted with regard to overshooting the benchmark values. However, the plot fails to emphasize is that the degree of correlation is approximately the same in both runs.

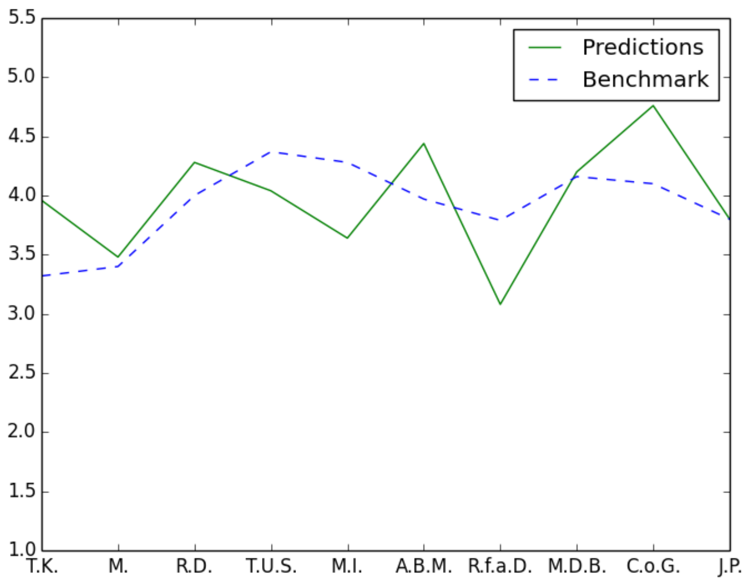

FIGURE 5.2: Run B: Line plot of predictions vs. benchmark value.

5.2.3 Evaluation result comparison

Up to this point, the predictions have only been compared with the benchmarks. Here, they will be compared to each other.

First, the benchmarks. Let the set of benchmark ratings for run 1 be denoted R_1, and similarly the benchmark ratings from run 2 R_2. As expected, the average benchmark ratings from run 2 – the top movies – are clearly above the ones from run 1.

$$\bar{R}_1 = 2.882 \tag{5.5}$$

$$\bar{R}_2 = 3.919 \tag{5.6}$$

A difference of more than 1.0. A number along these lines is what we want to see as the difference between the average predictions from the two runs.

However, when we perform the same mean value comparison for the *predictions* from the two runs, denoted P_1 and P_2, we see a different pattern:

$$\bar{P}_1 = 3.966 \tag{5.7}$$

$$\bar{P}_2 = 3.968 \tag{5.8}$$

A difference of only 0.002 – a mere nothing.

The difference becomes especially revealing when compared visually, as in figure 5.3.

This result repeats in approximately the same way for every run: the average rating predictions seem to be completely disassociated with the benchmark ratings of the test set.

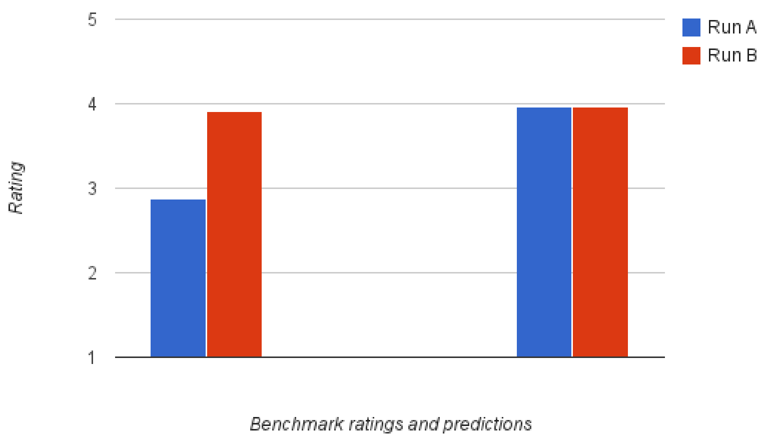

Figure 5.3: Comparison of ratings and predictions, per run. Left: ratings, right: predictions.

Chapter 6

Conclusion

As was shown from the two test runs in chapter 5, prediction results are very unreliable. The numbers describe a recommendation system with little or no overall connection to real-world ratings.

The system is able to achieve a low MAE for top movies because its predictions happen to average around the average benchmark rating. It is completely unable to consistently distinguish the quality of famous top-rated movies from unknown B-films. Furthermore, it is unable to process around half of the movie titles due to a lack of Twitter search results.

6.1 Suggestions for further work

Although the sentiment analysis approach taken in this project fails miserably, my impression of Twitter being an extremely interesting data source has not been weakened. Its qualities, as described in section 2.2, are shared with very few other data sources (if any), and its API is both efficient, well-designed and powerful.

There are several improvements to the techniques employed in this project that are possible to explore. Noisy data having been identified as the main weakness of the processed data, it would be interesting to see an approach where the Twitter messages are filtered before sentiment analysis is initiated. A conditional random fields (CRF) approach would be especially interesting to explore, being an active area of research in the field of opinion mining [16–18]. Other improvements may be applied to the sentiment classification step itself, where several other approaches report good results [5–9].

One could also look into using the data in other ways than predicting real-numbered ratings. Extracting more descriptive characteristics about a movie could serve as an interesting approach, for instance by extracting sentiment-carrying adjectives like "hilarious", "gritty", "exciting" etc. from the search results.

Also worth noting are the ways in which the specific techniques used in this project can be applied to other data sources. Building on the original motivation in section 1.1 and the current fact that the Twitter search API only returns data from the last 6-9 days (see section 2.2.2), the techniques might work better predicting ratings for new movies.

Appendix A

Evaluation Results

A.0.1 Run A: Sample drawn from entire test set

Title	Prediction	Variance	Benchmark rating
Chill Out	4.28	1.37	2.49
First Kid	4.48	1.21	3.59
Paparazzi	3.75	1.39	3.39
That Thing You Do!	4.80	0.60	3.53
Undercover Brother	4.05	1.59	3.10
The Backyard	4.47	1.36	2.36
Black Cat	3.80	1.70	2.30
Dangerous Game	3.83	1.52	2.54
So Little Time	3.60	1.80	3.10
Play for Me	2.60	1.96	2.42

TABLE A.1: Test run for a sample of movies randomly selected from the entire test set.

A.0.2 Run B: Sample drawn from IMDB's top 250 list

Title	**Prediction**	**Variance**	**Benchmark rating**
The Kid	3.96	1.57	3.32
Metropolis	3.48	1.17	3.40
Reservoir Dogs	4.28	1.11	4.00
The Usual Suspects	4.04	1.51	4.37
Monsters, Inc.	3.64	1.76	4.28
A Beautiful Mind	4.44	1.33	3.97
Requiem for a Dream	3.08	2.00	3.79
Million Dollar Baby	4.20	1.60	4.16
City of God	4.76	0.11	4.10
Jurassic Park	3.80	1.60	3.80

Table A.2: Test run for a sample of movies randomly selected from IMDB's top 250 list.

Bibliography

[1] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, January 2009. ISSN 1687-7470. doi: 10.1155/2009/421425. URL http://dx.doi.org/10.1155/2009/421425.

[2] Akshay Java, Xiaodan Song, Tim Finin, and Belle Tseng. Why we twitter: Understanding microblogging usage and communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis*, WebKDD/SNA-KDD '07, pages 56–65, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-848-0. doi: 10.1145/1348549.1348556. URL http://doi.acm.org/10.1145/1348549.1348556.

[3] Vivek Kumar Singh, Mousumi Mukherjee, and Ghanshyam Kumar Mehta. Combining a content filtering heuristic and sentiment analysis for movie recommendations. In K.R. Venugopal and L.M. Patnaik, editors, *Computer Networks and Intelligent Computing*, volume 157 of *Communications in Computer and Information Science*, pages 659–664. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-22785-1. doi: 10.1007/978-3-642-22786-8_83. URL http://dx.doi.org/10.1007/978-3-642-22786-8_83.

[4] Twitter Inc. Using the twitter search api, October 2013. URL https://dev.twitter.com/docs/using-search.

[5] Alec Go, Lei Huang, and Richa Bhayani. Twitter sentiment analysis. *Entropy*, 17, 2009.

[6] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, pages 1–12, 2009.

[7] Alexander Pak and Patrick Paroubek. Twitter as a corpus for sentiment analysis and opinion mining. In *LREC*, 2010.

[8] Apoorv Agarwal, Boyi Xie, Ilia Vovsha, Owen Rambow, and Rebecca Passonneau. Sentiment analysis of twitter data. In *Proceedings of the Workshop on Languages in Social Media*, pages 30–38. Association for Computational Linguistics, 2011.

[9] Efthymios Kouloumpis, Theresa Wilson, and Johanna Moore. Twitter sentiment analysis: The good the bad and the omg! In *ICWSM*, 2011.

[10] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics, 2002.

[11] Kamal Nigam. Using maximum entropy for text classification. In *In IJCAI-99 Workshop on Machine Learning for Information Filtering*, pages 61–67, 1999.

[12] Vasilis Vryniotis. How to build your own twitter sentiment analysis tool, September 2013. URL http://blog.datumbox.com/how-to-build-your-own-twitter-sentiment-analysis-tool/.

[13] H. Peng, Fulmi Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(8):1226–1238, 2005. ISSN 0162-8828. doi: 10.1109/TPAMI.2005.159.

[14] Silviu Cucerzan. Large-scale named entity disambiguation based on Wikipedia data. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 708–716, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D/D07/D07-1074.

[15] Luís Sarmento, Alexander Kehlenbeck, Eugénio Oliveira, and Lyle Ungar. An approach to web-scale named-entity disambiguation. In *Proceedings of the 6th International Conference on Machine Learning and Data Mining in Pattern Recognition*,

MLDM '09, pages 689–703, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-03069-7. doi: 10.1007/978-3-642-03070-3_52. URL http://dx.doi.org/10.1007/978-3-642-03070-3_52.

[16] Yejin Choi, Claire Cardie, Ellen Riloff, and Siddharth Patwardhan. Identifying sources of opinions with conditional random fields and extraction patterns. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 355–362, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics. doi: 10.3115/1220575.1220620. URL http://dx.doi.org/10.3115/1220575.1220620.

[17] Niklas Jakob and Iryna Gurevych. Extracting opinion targets in a single- and cross-domain setting with conditional random fields. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 1035–1045, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1870658.1870759.

[18] Bishan Yang and Claire Cardie. Extracting opinion expressions with semi-markov conditional random fields. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 1335–1345, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=2390948.2391100.