# Chapter 29

# Sentiment Analysis and Social Media

Luigi Curini and Robert A. Fahey

## ‘How Does It Make You Feel?’

Sentiment analysis (sometimes also called tone analysis (Grimmer and Stewart, 2013) or opinion mining (Dave et al., 2003)) is a set of methods designed to answer the same deceptively simple question – how do people feel about a given subject? More precisely, given a specific topic (which might be a product in the case of a marketing analysis, or a politician, party or policy in the case of a political analysis), do the authors of a certain set of texts express positive or negative feelings with regard to that topic?

This is the kind of task which humans generally find intuitive and simple, but which transpires to be exceptionally complex and difficult for computers to perform. A class instructor reading over their end of term feedback forms will quickly form a sense of whether students enjoyed the course or not. A computer performing text analysis on the same data, however, faces a number of daunting problems. A text analysis algorithm will lack context, so students’ references to specific topics or events during the semester will be rendered meaningless. It may find some of the casual or idiomatic language students use difficult to parse or recognise (in this, at least, many instructors may sympathise). Moreover, given the inherently stochastic nature of

text and working from such a limited set of data – just a few dozen pieces of text – it may be unable to establish any consistent framework for what a 'positive' or 'negative' piece of feedback actually looks like.

In spite of this difficulty, sentiment analysis has become an essential tool in many fields across the social sciences. Like many other text analysis techniques, its great strength lies in its ability to handle large volumes of data. While a human being may make a more reliable and nuanced judgement of sentiment on a single piece of text, or a small number of texts, it is impossible for a human to read and analyse the volume of texts generated by, for example, product reviews on a major website, or the customer feedback form for a popular product. The problem of volume is compounded further when social media is the focus of study; a major event can generate millions of posts per minute on a site like Twitter and even a relatively obscure topic may leave researchers with hundreds of thousands, if not millions, of posts to analyse. Effective application of sentiment analysis to social media has created remarkable new possibilities for political and social researchers – for example, real-time sentiment analysis of Twitter data has allowed researchers to see how audiences are reacting moment-to-moment to election broadcasts and candidate debates (Wang et al., 2012; Smailović et al., 2015).

This chapter will introduce a range of different approaches which are used for sentiment analysis, broadly categorising them according to the form of methodology employed and discussing the benefits and limitations of each approach. There is no single 'best in breed' approach to sentiment analysis and a researcher's choice of technique will depend heavily on the kind of texts they wish to analyse and the resources – in terms of both time and money and technical skill – at their disposal.

## Approaches to Sentiment Analysis

Sentiment analysis approaches can broadly be divided into two major categories – dictionary techniques and supervised learning techniques. Dictionary techniques, as the name suggests, involve constructing a dictionary of sentiment-scored terms and applying it to a text using an algorithm which uses the dictionary scores to calculate the sentiment of the overall text. These approaches range from relatively naïve sentiment scoring of individual words through to more complex techniques, for example taking into account the surrounding context of a word in determining its scoring and weight. Supervised learning techniques, on the other hand, rely on the creation of a labelled set of data – usually by getting human coders to classify the sentiments of a sub-sample of the data to be analysed. This human-coded data is then used to train an algorithm which will classify the remainder of the (unlabelled) data. While many supervised learning approaches to sentiment analysis, described below as 'classification methods', are effectively special cases of the broader text classification methodologies outlined in Chapter 28 of this Handbook, another group of approaches – which we refer to herein as 'aggregate methods' – do not try to assign a specific classification to every piece of text in a corpus, instead aiming to probabilistically estimate the distribution of sentiment across the corpus as a whole. These aggregate methods are especially well suited to corpuses with very large numbers of very short texts – for example, Twitter posts – where individual messages might not contain enough information for standard classification approaches to make a reliable determination about sentiment.

## Dictionary approaches

Dictionary, or lexicon-based, approaches to sentiment analysis rely, at the most basic level, on identifying specific words as being positive or negative and calculating the sentiment of a text based on those scores.[1] The most challenging aspect of this approach is compiling the sentiment dictionary itself; many different ways of tackling this problem have been developed, such as crowdsourcing lexicon entries through services such as Mechanical Turk (Haselmayer and Jenny, 2017); using a thesaurus to iteratively 'snowball' out from an initial set of human-chosen positive and negative seed words; estimating positive and negative weights for new words based on the frequency of their co-occurrence with known words in a large text corpus; or using pointwise information to calculate the probability of given words appearing in positively or negatively coded documents. This process is generally complex and resource-intensive, and generally beyond the scope of a political science research project; while there may be circumstances in which a political science researcher decides to compile their own sentiment dictionary (either selecting terms by hand, an example of which can be seen in Rooduijn and Pauwels' (2011) work on identifying populism in text, or by refining existing lists of words to make the sentiment scoring more relevant to a specific field, such as in Loughran and Mcdonald (2011)), it is much more common to use an existing dictionary that has been compiled and tested by researchers in the fields of natural language processing or computational linguistics.

The availability of pre-compiled sentiment dictionaries – such as the MPQA Subjectivity Lexicon (Wilson et al., 2005), the Hu & Liu Opinion Lexicon (Hu and Liu, 2004), SentiWordNet (Baccianella et al., 2010), AFINN (Nielsen, 2011) or EmoLex (Mohammad and Turney, 2013) – makes dictionary-based sentiment analysis an attractive option for researchers simply due to its

simplicity. In fact, popular text analysis software packages such as R's tidytext (Silge and Robinson, 2017) or Python's NLTK (Bird et al., 2009) come with a variety of sentiment dictionaries pre-installed, making it very easy for researchers to try out dictionary approaches on their text data. The major advantages of dictionary-based sentiment analysis approaches lie in their speed and simplicity, allowing researchers to rapidly assess the sentiment expressed in very large datasets with minimal use of resources. While this makes these approaches attractive for certain applications, especially where real-time analysis is required, significant concerns over the accuracy of these methods limit their potential in political science research.

The major weakness of these pre-compiled dictionaries is that they are by nature very generalised and lack the domain-specific features which might be required for sentiment analysis of a topic that has specific associated vocabulary and language features. On a broad level, this means that some dictionaries are not appropriate for classifying certain types of text – a dictionary trained using a corpus of formal language (such as the text content of Wikipedia or a large corpus of newspaper articles) would perform poorly in classifying the sentiment in a casual, slang-heavy corpus of social media posts, and vice versa. To address this problem, domain-specific pre-compiled sentiment dictionaries have been created: AFINN and the VADER Sentiment Lexicon (Gilbert, 2014) are designed for the kind of text commonly found in social media posts, while other dictionaries encompass vocabulary related to a specific topic (for example, the Loughran & McDonald sentiment lexicon for financial texts: Loughran and McDonald, 2011). These dictionaries go some way towards making lexicon-based approaches viable for a wider range of applications, but fundamental problems remain. Dictionaries are static, while language is ever-evolving – especially in the political domain, where words and

phrases can take on new meanings or implications very rapidly during the course of an electoral campaign or political news cycle.

A further challenge for dictionary approaches relates to how the scoring of the lexical dictionary is applied to the text. In the simplest possible model, the positive and negative sentiment scores for each word in the text are added together to yield a final score. While this approach is intuitive, its drawbacks are easy to see. Consider the following sentence from a review of a movie:

> *The movie has a fantastic cast, an interesting concept and amazing special effects – but it is utterly boring*.

A naïve calculation based on a sentiment dictionary would almost certainly find that this sentence was positive due to the presence of a large number of positive words (‘fantastic’, ‘interesting’, ‘amazing’, ‘special’) and only one negative word (‘boring’) – yet to a human reader it’s very obvious that the overall tone of the sentence is in fact negative. In order to improve on the accuracy of this kind of approach, algorithms have been developed which take into account additional features of the text. Where the naïve dictionary approach is a pure ‘Bag of Words’ approach, which is common to many text analysis methodologies and treats a document purely as a collection of words, ignoring the order of those words or the relationships between them, more advanced algorithms draw instead on the field of Natural Language Processing (NLP). A simple but effective example is the VADER algorithm, which incorporates a number of features that improve the accuracy of dictionary-based classification – for example, it understands various forms of negation (so ‘not good’ and ‘this wasn’t good’ would both be treated as negative despite ‘good’ being a positive word), contrastive conjunctions such as ‘but’ or ‘however’, words which alter the intensity of a sentiment (like ‘utterly’ in the example sentence above) and the presence

of emoji in social media posts. These improvements make VADER and similar algorithms more effective than naïve dictionary-based sentiment scoring without increasing the complexity for researchers using the tool. Even more advanced ways of applying sentiment dictionaries continue to be developed by NLP researchers, but often introduce a degree of technical complexity or resource-intensive computation which puts them beyond the reach of most researchers seeking to implement a practical sentiment analysis: the Stanford Sentiment Treebank (Socher et al., 2013), for example, achieved an improvement in classification accuracy over other approaches by using an artificial neural network to process sentence structure, but this introduces a computational and technical overhead which is likely to be impractical for most researchers.

The domain-specificity of sentiment dictionaries means that a given dictionary can only be used effectively to classify sentiments in text that follows broadly the same linguistic styles and standards as the corpus used to create that dictionary. As a consequence, while a large number of sentiment dictionaries exist for English, and dictionaries have also been created for other major languages, there are a significant number of languages for which few, if any, high quality sentiment dictionaries exist (Mohammad, 2016). This significantly restricts the potential for using sentiment dictionary approaches in analysis of text in minority languages, or in cross-linguistic analyses; while attempts to apply English-language sentiment dictionaries to other languages using machine translation tools have been made, there is little evidence to support the effectiveness of such an approach. Nonetheless, sentiment dictionaries remain a useful set of techniques for use in exploratory analysis of data, or for providing supporting evidence in research where sentiment analysis is not the primary analytic tool being employed.

## Supervised learning approaches

Unlike dictionary-based approaches, which make *a priori* assumptions about the positive or negative meanings of specific words in a lexicon, supervised learning approaches begin with a blank slate each time, making no assumptions about the sentiment or meaning of any word in the corpus. Instead, a sub-set of the corpus is classified by human coders and this labelled set is used to train a machine learning algorithm, which will then estimate the sentiment of the rest of the corpus based on the patterns and correspondences found in the labelled set. One set of algorithms, the classification algorithms, attempt to accurately classify the sentiment of each individual unit of text in the corpus (in this regard resembling the dictionary approach described above); another, the aggregate algorithms (sometimes called proportional classification algorithms (Wiedemann, 2018)), focus instead on accurately estimating the distribution of sentiment across the entire corpus.

The drawback of such approaches is immediately obvious – you cannot proceed without first creating labelled data. While a pre-compiled sentiment dictionary allows you to instantly start exploring the sentiments in a corpus of text data, supervised learning approaches require that you first classify a sample of the data by hand. This brings with it the usual problems associated with human content coding – the need to train coders, to verify the consistency of their work using inter-coder reliability measures and to decide on an appropriate sampling approach and sample size to classify.

Of these, the question of how much data is needed to train an algorithm is a common sticking point for researchers. The most common response is ‘as much as possible’; Hopkins and King (2010) offer 500 as a rule of thumb. While algorithms have been shown to have a learning

curve which eventually flattens out, leading to diminishing returns from the creation of additional labelled data (Figueroa et al., 2012), the complexity of text classification and the resource constraints faced by most researchers make it unlikely that they will reach this point, so as a general rule more labelled data is better. In more practical terms, we can think of the volume of labelled data required as a function of the number of inputs (i.e. the size of the vocabulary being used in the texts) and the number of outputs (the number of categories to classify). A text corpus which addresses a wide range of different issues will require more labelled data than a more narrowly focused corpus; for example, a set of social media posts addressing a specific policy initiative will have a smaller lexicon of relevant vocabulary than a set of posts about a candidate in a major election, as the latter will likely include posts discussing a range of different issues and policies. As such, more labelled data would be required for the latter data set, even if the actual sizes of the two corpuses were the same. The choice of algorithm is also relevant to this question; as a general rule, aggregate algorithms require less labelled data than classification algorithms.

The other main challenge to keep in mind when dealing with building an appropriate and reliable set of labelled data relates to the sampling of documents. Classification methods implicitly assume that the labelled set is a random sample from the population of documents to be coded, in terms of the opinions expressed (but see below the discussion about aggregate algorithms) as well as, crucially, the language employed. This is because supervised learning methods use the relationships between categories and features in the labelled set to classify the remaining documents. Such algorithms are flexible enough to classify any kind of data that can be broken down into ‘tokens’ (features), but simply cannot classify the sentiment of any feature which was not encountered in the labelled data. As such, if an important word or other such

feature (like an emoji or a hashtag) was not included in the texts classified by human coders, the trained algorithm will miss its relevance to the sentiment of the texts it classifies. This presents particular difficulty when all the data are not available at the time of coding of the labelled set, for example in situations where all the data have not yet been digitised or, as is often the case with sentiment analysis on social media, where data will continue to be produced in the future. This causes situations where the difference in the language between the labelled set and the unlabelled set grows over time, such as when new words and phrases, or new meanings of existing words and phrases, appear in the latter set but not in the former set. For example, a politician may introduce a new slogan during a speech which was not part of the common lexicon at the outset of a campaign period, or a common word may take on a new meaning through association with a scandal or a major policy. Alternatively, the inverse may occur, and words, phrases and their meanings may exist in the labelled set but not in the unlabelled data. This is most common in long-term analyses; for example, in the years after the fall of the Berlin Wall, terminology related to the Cold War largely disappeared from political rhetoric. Some of these challenges may be partially addressed with the use of distributed word embeddings, a technique which has become popular in natural language processing and computational linguistics (Goldberg, 2016; Pennington et al., 2014; see also Chapters 26 and 55 of this Handbook). Word embeddings represent words in a continuous vector space in which words with similar meanings are mapped closer to each other. Using a set of word embeddings appropriate to the texts being analysed can allow new words encountered in the unlabelled texts to be classified according to their similarity to words that were present in the labelled texts (Rudkowsky et al., 2018). This does not, however, account for the arising of new meanings for words, such as the naming of new scandals – word embeddings produced from early 1970s texts

would correctly identify ‘Watergate’ as being close in meaning to the word ‘hotel’, but would need to be regenerated from new texts to identify its new meaning closer to ‘scandal’ and its consequent importance to political sentiment.

Once the sample has been classified, the subsequent process is broadly the same regardless of which kind of algorithm is used. As a result, researchers often train multiple algorithms and test to see which approach, and which algorithm, is more effective for their specific case. This testing is performed by repeatedly splitting the labelled sample data into ‘training’ and ‘test’ sets, training the algorithm on the former and then testing it on the held-out data in the latter – a process known as cross-validation. The algorithm which performs most effectively and consistently in these tests is then used to classify the entirely unseen texts in unlabelled data set.

For the purposes of explaining classification and aggregate algorithms, let us assume that we have a labelled corpus (classified by human coders) of $N$ distinct texts, sampled from the overall corpus (which we label $C$) we wish to classify. Let us denote by $D = \{D0, D1, D2, \ldots, DM\}$ the set of $M + 1$ possible *categories* (that is, sentiments) expressed in the texts. $D0$ denotes the likely most prevalent category in the data – generally, ‘Off-Topic’ texts or those which express opinions/sentiments not relevant with respect to the analysis, that is, the *noise* in this framework. Noise is commonly present in any corpus of texts crawled from social networks and the internet in general and should be taken into account when we evaluate a classification method (a point to which we will return below).

Suppose that after pre-processing the text (removing irrelevant words and features, stemming or lemmatising as appropriate, and so on) we are left with $L$ features (word stems, or tokens). The document-term matrix $S$ representing our corpus then has $N$ rows and $L$ columns, with each document $j$ being represented by a vector $S_j$ of length L. The value of each element of

the vector $S_j$ may either be the frequency with which that feature appeared in document *j*, or a binary value – 1 if the feature appeared at all, 0 if it did not. This latter representation is commonly used for short documents such as social media posts, since the frequency of a given word is most likely to be 0 or 1 anyway; for longer documents the range of word frequencies is much greater and of more importance. If the lengths of the documents being studied differ significantly (which is usually not the case for social media posts) it is also necessary to account for this in their vector space representation, for example by weighting the vocabulary frequency inversely to the length of the document. Note that the overall document-term matrix (*S*) is often quite a large data set but very sparse, in that it contains a large number of zero values since each document only contains a small sub-set of the overall lexicon of the corpus *L*.

Regardless of the subsequent approach taken to sentiment analysis, this document-term matrix *S* will be the basic data set used. To ensure the robustness of the entire estimation strategy, cross-validation is performed. This process entails dividing the labelled data into a number of equal-length, randomised segments (usually between 5 and 10) and using each segment in turn to train the algorithm being tested. Each of these trained algorithms is then tested against the remainder of the labelled data, which is called the "held-out" data – so for example, if the labelled data is divided into five segments, the algorithm being evaluated would be trained five times, each time on 20% of the data, and tested each time against the remaining, or held-out, 80% of the data. . If an algorithm performs well on each step of cross-validation (for example in terms of its accuracy, that is, the proportion of correctly classified documents)[2] it can be expected to perform reliably on entirely unseen data as well (that is, *C-N*); conversely, an algorithm which fails on a given cross-validation step may be poorly suited to a specific permutation of features that arises in the data.

## Classification algorithms

There is a very wide variety of different classification algorithms, ranging from those which will be familiar to most researchers with a background in statistics, such as Naïve Bayes, through to complex and computationally intensive algorithms such as Neural Networks and Random Forests (see Dreiseitl and Ohno-Machado, 2002 for details of many of these models). The differences between classification algorithms are the subject of very extensive literature in statistics and computer science which is beyond the scope of this chapter – an in-depth overview of the field as a whole is provided in Aggarwal (2014). For the purposes of the political science researcher attempting to perform sentiment analysis, the key thing to note is that the training process for any one of these models takes the same input (a training subset of the term-document matrix *S* described above) and gives the same output, a model (or function) which predicts the category $D_j$ to which a given document *j* belongs given that its features are represented by the vector $S_j$. This model can be represented as $P(D|S)$ – for a given document *j* and set of categories *M*, it will find the value of *m*, that is, the classification, which maximises the model $P(D_{m=0,1,\dots,M}|S_j)$. Expressing this as a matrix model, the classification algorithm is $P(D) = P(D|S)P(S)$ – where *P(D)* is a vector of length M+1, *P(D|S)* is a matrix of conditional probabilities and *P(S)* is a vector representing the distribution of text vectors across the corpus of texts. This means that regardless of the broad differences in how the algorithms function internally, it is possible to train them using the same data, directly compare their results, and select the one which is most effective for your specific data and usage case. In fact, popular machine learning packages such as Python's *scikit-learn* (Pedregosa et al., 2011) provide pipeline structures which are designed to automate this entire process of training, testing and comparing algorithms.

Once trained, classification algorithms can be used to predict the sentiment classification of an entirely unseen text. This approach has some significant advantages over the dictionary-based approach. It is by nature domain-specific; the algorithms base their predictions on words or patterns which were relevant to the classification categories in the training set and can often identify the sentiment of a phrase which would be entirely missed by a dictionary approach. A good example is political catchphrases – in the 2016 US presidential election, the phrase 'lock her up' was strongly associated with negative sentiment towards Hillary Clinton, while 'build the wall' was associated with positive sentiment towards Donald Trump. An effective classification algorithm trained on tweets related to the election would correctly identify the relevance of those terms, while a sentiment dictionary approach would treat both phrases plausibly as being neutral. Classification algorithms are also language-independent; while some languages require more complex pre-processing steps before the algorithms can be applied, the algorithms themselves do not know or care which language the words (or 'tokens') are in, making this approach perfectly effective even in 'resource-scarce' languages which lack high-quality sentiment dictionaries.[3]

Nevertheless, classification algorithms are by no means perfect. This is in part due to the properties of the sentiment analysis task itself: not only is language intrinsically stochastic, but there is also a subjective element to the sentiment classification of many edge cases and disagreement even between human coders is normal. Moreover, classification algorithms generally rely on the Bag of Words approach and are unaware of sentence structure; this provides for fast and effective classification but fails on certain edge cases (for example, the Bag of Words representation of the sentence 'I'm voting for Clinton because Trump is the most corrupt candidate ever' is exactly the same as the representation for 'I'm voting for Trump because Clinton is the most corrupt candidate ever', so a classifier would be unable to distinguish

between these polar opposite sentiments). As such, no classifier should be expected to achieve or even approach 100% accuracy in testing against held-out data – in fact, approaching 100% accuracy is likely a sign that the algorithm has been overfitted to the training data and will not be effective at classifying unseen data.

A further problem with classification algorithms can arise when a single category is predominant in the data – as is commonly the case with social media data, where the 'Off-Topic' category (a catch-all featuring all documents which did not include a topic or sentiment of interest to the study) is often vastly over-represented compared to other classifications. This means that during the training process the algorithm 'learns' that the vast majority of feature vector configurations are associated with this dominant category *D0*; only a small subset of specific vectors express any other category $D_j$. As a result, for a large proportion of input vectors $S_j$ the output of $P(D_i|S_j)$ will be zero, or extremely low, for all categories other than the dominant one ($i=0$). As a result, the trained model will tend to overestimate the dominant category when classifying unseen texts. This effect is especially strong when the dominant category is a catch-all, 'off-topic' category. This category will contain all of the 'noise' that is present in data collected from social media; only a specific sub-set of input vectors will correspond to the other categories being trained, with every other possible vector (a much larger domain) being implicitly 'noise'. Depending on the objectives of the researcher, this over-estimation of a dominant category can sometimes be unproblematic – but where the objective is to find the overall distribution of sentiment within a data set, this systematic overestimation can result in significant bias. In the next section we will discuss a new class of algorithms which have been developed also to deal with this bias.

## Aggregate algorithms

While classification algorithms aim to predict the sentiment of every individual piece of text in a data set, it is often the case that a researcher is looking for the distribution of sentiment within a corpus of texts and does not actually require an item-level classification. As a result, researchers end up aggregating the item-level classifications, which means that some of the trade-offs that are made by classifiers in the name of providing a single classification for every item – such as the issue of over-estimation of a dominant category, as discussed above – are actually unnecessary and may be needlessly reducing the accuracy of the prediction on the overall distribution. For example, a method that classifies 60% of documents correctly into one of eight categories might be judged successful and useful for classification. However, because the individual category percentages still might be off by as much as 40 percentage points, the same classifier may be useless for some social science purposes (if individual-level errors do not cancel each other out). Recognising this problem has led to the development of a new set of algorithms which treat the text corpus in an aggregate manner from the outset – they do not provide classifications for individual texts at any point in the process, instead producing results showing the distribution of sentiments within the overall corpus. The first of these algorithms to be released was ReadMe (Hopkins and King, 2010), which has since been joined by iSA (Ceron et al., 2016); while these algorithms have different strengths, they use similar inputs and produce similar outputs, making it possible to train and test them against one another in a similar manner to the algorithm selection process used for classification algorithms.

These aggregate approaches are less flexible than classification approaches, since they cannot be applied to any research that actually requires the classification of individual texts at

any point – but in return they offer two major advantages. First, because they are not constrained by the need to produce a classification for every text, they can treat classifications as probabilities rather than binary predictions – so instead of decisively coming down on the side of a text being either 'positive' or 'negative' (or whatever categories are being used) when it actually shows some features of multiple categories, the algorithm can in essence treat it as being partially positive and partially negative when calculating its impact on the sentiment distribution of the entire corpus. This kind of approach, moreover, can deal well with situations in which the relative frequency of categories in the training set is very unevenly distributed – such as in the above mentioned case of a category containing off-topic 'noise' being statistically dominant within the data set – or where the distribution in the training set is quite different from the frequency in the unseen data set.

Expressed in statistical terms, aggregate algorithms follow from the insight of Hopkins and King (2010) that the estimation process used by classification algorithms and shown in the prior section – $P(D) = P(D|S)P(S)$ – can be reversed; they propose a solution as follows:

$$P(\mathrm{D}) = [\mathrm{P}(\mathrm{S}|\mathrm{D})^{\mathrm{T}}\mathrm{P}(\mathrm{S}|\mathrm{D})]^{-1}\mathrm{P}(\mathrm{S}|\mathrm{D})^{\mathrm{T}}\mathrm{P}(\mathrm{S})$$

In essence, this formula uses an inverse matrix to allow the estimation not of $P(D|S)$ (the probability of category *D* given vector *S*), but of $P(S|D)$ – the probability of vector *S* given category *D*. Given sufficient coded texts for each category *D*, it is actually possible to estimate this significantly more accurately than $P(D|S)$ – the trade-off being that we must estimate it across the entire text corpus, not on a per-document basis.

The secondary benefit of these algorithms is related to the required properties of the labelled set of documents. First, it is often possible to achieve good results with a smaller set of labelled data than is required to train an accurate classification algorithm. Moreover, aggregate

algorithms do not require that the labelled set should be a representative sample of the full population of texts; the only demand for selection of labelled data is that the language used in the labelled documents to express some given concept must be the same as in the whole population of texts, and that the labelled data must include sufficient examples of each individual category to be classified as to permit generalisation.

While aggregate algorithms are not perfectly suited to all kinds of analysis, their specific strengths make them particularly attractive not only for analyses that rely entirely on aggregate data, but also for real-time applications (where it is likely not possible to generate large amounts of training data on the fly) and for projects that lack the resources to undertake human coding of a large sample of the data.

## Sentiments on Social Media

One of the most common applications of sentiment analysis in recent years has been in uncovering the sentiments being expressed about topics on social media. While the field of sentiment analysis itself far predates social networks (or indeed widespread internet usage), the volumes of text data produced on social media are often orders of magnitude greater than any prior data source in the political or social science fields. Combined with the unprecedented level of access to the preferences and views of a large section of the public represented by social media posts, this has made the ability to programmatically calculate sentiment across large datasets extremely important.[4]

Using social media data for sentiment analysis introduces a number of opportunities and challenges for researchers (some of them already discussed in the previous pages). In essence, social media data is generally made up of a large number of short texts, which makes it ideal for

this kind of analysis. When applying sentiment analysis to longer texts such as newspaper articles or political speeches, it is necessary to decide what the actual unit of analysis will be – an entire document, a paragraph or an individual sentence. While some social media platforms do permit the posting of long texts, posts are generally short enough to settle on the individual post as a unit of analysis – especially on Twitter, a platform whose open nature makes it especially useful to researchers and whose strict character limit makes it easy to treat every post as a distinct unit for analysis.

## The pre-processing stage

Social media posts are often written and presented in a very different style to formal texts such as newspaper articles or speeches, however. Slang, abbreviations and shorthand are common (especially on Twitter, whose tight character limits encourage brevity). Along with platform-specific text features like @usernames and #hashtags, posts may also include images, web links and emoji (pictographs). In some regions, users commonly construct faces from punctuation marks and other characters to express emotions using ‘smileys’ or ‘kaomoji’. These text features create special challenges for researchers, not least because many kinds of text analysis software can’t handle them – it’s not uncommon for punctuation marks to be removed in pre-processing (which therefore removes smileys and kaomoji) and for emoji either to be ignored or to result in software errors. Handling of web links can also create problems; software often splits links up into multiple tokens (e.g. changing ‘http://www.google.com’ into ‘http’, ‘www’, ‘google’ and ‘com’) and the high incidence of tokens like ‘http’ and ‘www’ (which appear in every URL) can result in feeding bad data to the sentiment analysis process. It is therefore important for researchers to pre-process social media data (texts) in a way that is sensitive to these specific text features; in particular, approaches which dispose of emoji or smileys should be avoided, since

these features often have a very direct impact on the sentimental content of a text. A single emoji at the end of a sentence can entirely change the sentimental tone of that sentence, so any sentiment analysis approach which ignores emoji is excluding important data from the outset.[5]

## The data access riddle

A further problem with social media data relates to the relevance and reliability of the data itself. Gathering representative data from social media given the limitations placed by factors such as API limits (programmatic limitations on the amount of data that can be downloaded in a given time period) and account privacy settings is a major challenge that is beyond the scope of this chapter, but researchers wishing to perform sentiment analysis on social media data need to make themselves aware of these issues and develop a strategy to tackle them. More specifically related to sentiment analysis is the problem of keyword specificity – the keywords used to gather and filter social media data must be chosen very carefully in order to ensure that the data is actually relevant to the topic being studied. There are several problems which can emerge in this regard, the simplest of which is the use of a keyword which actually has different meanings in different cultural or geographic contexts. For example, in a project on the Kenyan elections which one of the authors participated in, the name of a presidential candidate, ‘Uhuru’, was used as a search keyword on Twitter; it only emerged in later analysis that this word is also used in a completely different way by some right-wing groups in the United States, resulting in a large amount of irrelevant data being downloaded. Without removing these irrelevant results from the data by some means, sentiment analysis would have given skewed results due to calculating sentiment for posts that actually had nothing to do with the election we were studying. This problem can be tackled in several different ways: limiting the posts gathered to a specific country

or language can often be effective, but if this is impossible, adding the already discussed 'off-topic' classification to a supervised learning approach can allow human coders to effectively train the algorithm to recognise these irrelevant posts. A more complex problem arises when posts include multiple keywords which are being studied; for example, a post might mention the names of both candidates in an election, making it difficult to judge which candidate the sentiment classification actually relates to. One approach which can be effective in solving this problem is training a supervised learning algorithm to identify not just overall sentiment, but sentiment related to specific topics – so for example, in analysing a corpus of tweets related to the 2016 election, an algorithm might be trained to identify 'Trump-Positive', 'Trump-Negative', 'Clinton-Positive', 'Clinton-Negative' and 'Off-Topic' classifications ('Neutral' might also be an option) instead of a simple polar 'Positive' or 'Negative' classification.

## Non-European languages

Related to the challenges and opportunities with respect to social media data is a further set of challenges which arise from working with data in non-European languages. In particular, different languages often require different pre-processing before being converted into a Bag of Words or a document–term matrix, a tabular view with the corpus' vocabulary on one axis and the documents themselves on another, allowing each document to be represented as a sparse vector of vocabulary frequencies. East Asian languages such as Chinese and Japanese, for example, do not use spaces to separate their words and therefore require a more complex 'tokenisation' step for dividing up sentences into their component vocabulary. There is also a variety of different approaches to simplifying the vocabulary whose usage differs depending on the language being analysed: 'stemming' (which converts all conjugated words into their

dictionary form) and ‘lemmatisation’ (which reduces words back to a more basic semantic meaning, so for example the words ‘preside’, ‘president’ and ‘presidential’ might all be converted into ‘presid’) require different approaches in different languages, and may not be applicable to some languages at all.

A further complication arises from the lack of high-quality resources for sentiment analysis in some languages. As already noted, this is especially problematic for dictionary- or lexicon-based approaches; while there are a wealth of different sentiment dictionaries and algorithms available for English, some languages may have none at all, or may only have poor-quality or incomplete dictionaries. In these cases, supervised learning approaches are essential; the great advantage of supervised learning (both classifier and aggregate approaches) is that the algorithms are entirely language-agnostic. In fact, many of these algorithms are also used to classify data that isn’t even text, such as sounds or images – as long as the features encountered in the unlabelled data to be classified also appeared in the labelled documents, it doesn’t matter at all what those features actually are. Supervised learning algorithms will classify a document–term matrix of Japanese words just as easily as one made up of English words and will not care if some of those ‘words’ are actually other text features like emoji, smileys or hashtags.

## A Worked Example

We will conclude with a practical example of the different techniques we have outlined above, showing the differences between the methods and their outcomes and pointing out potential pitfalls researchers may encounter with different sets of data. For the purposes of this example, we will use the Stanford Large Movie Review Dataset (Maas et al., 2011) – a data set which includes 50,000 movie reviews drawn from the IMDB website and classified as ‘positive’ and

‘negative’ according to the score the user gave the film, with reviews scored 1 to 4 considered ‘negative’ and those scored 7 to 10 considered ‘positive’. Since these are user-submitted reviews, they include many of the same features we would expect from social media texts – misspellings, slang and grammatical mistakes are commonplace, which can pose problems for some sentiment analysis approaches. For the purposes of this example we randomly sampled 5,000 reviews to treat as labelled data, with the remaining 45,000 being treated fictionally as unlabelled data and used to test the accuracy of the various approaches.

For the purposes of this worked example, we will use Python with a number of popular and readily available scientific packages. A full version of the script, with additional information and details of the packages and techniques used, is available in the online Appendix.[6]

## Loading and pre-processing

We first load the data into the script – in this case, the data set is distributed as a large folder of text files, one review per file, but it is also common to need to import data from CSV files, Excel files or databases. Regardless of the source, it is often easiest to convert the data into a list of text strings. For training purposes, we also need the classifications of the test data, which should also be stored in a list (in the same order as the documents to which the classifications belong).

The pre-processing which needs to be carried out on the text differs according to the text source and the language being analysed, but this case is typical – we remove HTML tags from the text, convert it all to lowercase and strip out all punctuation. For longer texts, it may be preferable to divide the text into sentences so that a sentence-by-sentence sentiment analysis can be carried out; for these short texts, we ignore sentence boundaries and other punctuation.

## Dictionary/lexicon sentiment analysis

To demonstrate the performance of dictionary analysis approaches on these texts, we can use two of the built-in dictionaries in the NLTK (Natural Language ToolKit) package for Python – the Liu & Hu Sentiment Lexicon and the VADER Sentiment Scoring tool. The Liu & Hu lexicon is the simplest approach – for this we simply divide up each review into its component words (a process known as 'tokenisation') and check each word to see if it is listed as having a positive or negative polarity in the lexicon. Documents with more positive than negative words are treated as 'positive' and vice versa; documents whose negative and positive words cancel one another out (or which have no such words) are 'neutral'. Since this approach requires no 'training', we could employ the dictionary to directly label all the 50,000 reviews in our corpus. However, to allow a direct comparison of dictionary sentiment analysis with the supervised approaches employed below, we have instead estimated the scores for the 45,000 'unlabelled' reviews. Despite the simplicity of the approach, it managed to correctly score 69.6% of the reviews in this corpus – detailed results can be seen in Table 29.1.

| | *Actual-Negative* | *Actual-Positive* |
|---|---|---|
| *Predicted-Negative* | 14597 | 4475 |
| *Predicted-Neutral* | 1662 | 1300 |
| *Predicted-Positive* | 6234 | 16732 |

**Table 29.1: Results for Liu & Hu Sentiment Lexicon**

The second approach we use is the VADER Sentiment Analysis algorithm. This is a more context-sensitive approach to sentiment scoring and gives a 'polarity score' for each document – a score scaled between -1 (completely negative) and +1 (completely positive). This can be useful

in some situations and VADER can often correctly identify the sentiment of edge cases which more simple lexicon approaches miss. However, on this set of test documents VADER yields largely similar results to the simpler Liu & Hu lexicon, with 69.6% of reviews correctly classified – largely due to a very high rate of negative reviews being incorrectly classified as positive. The algorithm was much more successful at classifying positive reviews, achieving 85.2% accuracy on these.

| | *Actual-Negative* | *Actual-Positive* |
|---|---|---|
| *Predicted-Negative* | 12142 | 3329 |
| *Predicted-Neutral* | 20 | 9 |
| *Predicted-Positive* | 10331 | 19169 |

**Table 19.2: Results for VADER Sentiment Analysis**

Note that both of these approaches are only applicable to the English language – for any other language, a different lexicon would need to be used (or created).

## Preparing a document–term matrix

Most approaches to sentiment analysis rely on the previously mentioned Bag of Words approach, which represents each document as a collection of words – ignoring their position and relationship to other words. While this loses significant amounts of information, it has proved very effective for a range of text mining applications. For a simple approach like the Liu & Hu lexicon analysis shown above, it's sufficient simply to divide the sentence up into its component words, but more complex approaches such as classification algorithms require a development of

a *vector space model* – which converts each document into a vector recording the presence or frequency of each vocabulary word. This representation of a corpus is known as a *document–term matrix*, with each row being a document, and each column a vocabulary word. This matrix can grow extremely large since the range of vocabulary in a corpus is often in the order of thousands of words. Two key approaches are used to manage this; first, the vocabulary list is trimmed by excluding very common and uncommon words (which are of little use for distinguishing documents from one another) and by stemming or lemmatisation – processes which reduce words down to shorter 'stems' representing their core meaning. It is often possible to reduce a corpus' vocabulary list to a few hundred words in this way. Second, the document–term matrix is stored as a 'sparse matrix' – an approach allowing computers to store large matrices where most values are zero in a fraction of the space normally required.

To construct a document–term matrix, we need to specify a tokeniser (a software function which splits sentences into component words, then applies stemming, lemmatisation and other such processes). In English and other European languages this step is relatively simple due to the spaces between words; in languages such as Japanese or Chinese, an additional piece of software is required to detect the word boundaries and carry out tokenisation. For Chinese, the Stanford Segmenter is a popular tool, while analysing Japanese is usually carried out using MeCab or Janome. Other languages, such as Korean, also require this step. A mistake in this tokenisation process can result in 'junk data' in the document–term matrix, so it's important to pick appropriate tools for the language you are working with.

Next, the tokeniser is used to train a 'vectoriser' which will analyse the corpus to find which vocabulary terms are relevant; after this 'fitting' process, it can be used to generate a document–term matrix for the entire corpus, or a vector for a single unseen piece of text which

will be compatible with the vectors in the document–term matrix (that is, each position on the vector will represent the same vocabulary term as it would in a row of the matrix).

## Classification algorithms

The process of training and testing a classification model involves ‘fitting’ the model to a set of training data and then scoring its performance at predicting the classifications of a set of test data. This process is carried out using multiple different models in order to select the most effective one for a specific set of data. Using the movie review data set, we tested a selection of classification models using a ‘labelled’ set of 5,000 randomly selected reviews. To ensure the robustness of this process, *cross validation* is used, wherein the data is divided into multiple ‘folds’ and the model is repeatedly trained and tested on different folds of data. In this example, the cross-validation process uses three folds (five to ten folds are also commonly used; the more labelled data you have, the more folds you can effectively use in this process), so the 5,000 labelled set is randomly divided into three subsets. We reserve one subset (the test set) and train the model on all other subsets (the training set). We then test the model on the reserved subset and record the prediction error, before repeating this process until each of the three subsets has served as the test set. We finally compute the average of the three recorded errors. This is called the cross-validation error and serves as a performance metric for the model. The results for the models tested can be seen in Table 29.3.

| Model Name | Average Accuracy | Cross-Val Error |
|---|---|---|
| Multinomial Naïve Bayes | 0.822 | +/- 0.030 |
| Bernoulli Naïve Bayes | 0.808 | +/- 0.015 |
| Support Vector Classifier | 0.813 | +/- 0.014 |
| Linear Support Vector Classifier | 0.501 | +/- 0.000 |
| Stochastic Gradient Descent | 0.800 | +/- 0.028 |
| Random Forest (10 trees) | 0.730 | +/- 0.019 |
| Neural Network (Multi-Layer Perceptron) | 0.811 | +/- 0.021 |

**Table 29.2: Initial Results for Classification Algorithms**

There is no one-size-fits-all classification algorithm; in this specific instance, the linear support vector classifier performed worst, with the random forest also achieving only marginally better scores than the dictionary-based approaches outlined above. For other data, however, these algorithms may outperform the others; it is important to test a variety of algorithms to see which is best suited to the specific combination of data and classifications being used. In this instance, the Multinomial Naïve Bayes classifier proved the most effective, with 82.2% accuracy on the held-out data set.

This is not the end of the process of training an effective classification model, however. Most algorithms also have a range of 'hyper-parameters' – assumptions and modifiers which can be set to different values prior to training – that can significantly impact performance. Finding

the right set of hyper-parameters for a certain task is also largely a case of trial and error. However, several different packages, both in Python and R, provide ways to automate this task; this is known as a 'grid search', allowing researchers to exhaustively search through every combination of a set of hyper-parameters to find the best performing model. This process can take a lot of time – often in the order of several hours for algorithms with complex sets of parameters – but often yields better performance than the default parameter set. For example, in our case we were able to locate a set of parameters for the support vector classifier model which boosted its performance to 82.3% – a full percentage point higher than the algorithm's performance in our initial test. This model would likely be considered sufficiently accurate and reliable for use in a research project. In the present example, it is possible to test that accuracy by scoring the trained algorithm's performance on the 'unlabelled' set of 45,000 reviews; for the final trained algorithm (the support vector classifier with the best hyper-parameters chosen through the grid search procedure), those results are shown in Table 29.4 below. The algorithm achieved 82.9% accuracy overall, 84.6% on the positive reviews and 81.2% on the negative reviews (more detailed scoring, including the precision, recall and f1-score metrics mentioned above, can be found in the online Appendix). Note that in a real research situation you would have to trust the accuracy of your trained algorithm at this point, since the rest of your data really would be unlabelled.

| | *Actual-Negative* | *Actual-Positive* |
|---|---|---|
| *Predicted-Negative* | 18261 | 3464 |
| *Predicted-Positive* | 4232 | 19043 |

**Table 29.3: Results for Final Classification Algorithm**

One additional possibility which we have not explored in this worked example is combining different algorithms to achieve better accuracy than a single algorithm would manage on its own. This is called an ‘ensemble’ approach and comes in two basic forms. The first, ‘model averaging’, involves training multiple different algorithms and then combining the results of those which showed acceptable performance on the test set, either by blending their predictions or by running them independently and allowing them to ‘vote’ on the classification of the text. The second, ‘model stacking’, was introduced by Wolpert (1992) and involves feeding the predictions from an ensemble of algorithms into another algorithm as parameters for a final prediction. Further details of these two approaches can be found in Sardinha (2017).

## Aggregate algorithms

Although aggregate algorithms function very differently from classification algorithms in many regards, they take exactly the same input – the document–term matrix – so we can re-use the matrix from the previous step. For this example we are using the Python version of iSA, PyiSA – at the time of writing there is no Python implementation of the other major aggregate algorithm, ReadMe, but this example could be recreated in R to test ReadMe’s performance.

The procedure followed by iSA is very similar to the one used by the classification models shown above. First, we use the labelled documents to train the algorithm. Then the algorithm will predict the classification of the unlabelled documents. Unlike the classification models, however, iSA returns an estimate of the distribution of each classification within the overall corpus (labelled plus unlabelled set of documents), and also estimates its own standard errors. If you wished, it would also be possible to run a cross-validation of the iSA algorithm by checking the

consistency of results from different sub-folds of the labelled data – we have not done so in this instance, but the procedure would be functionally the same as for a classification algorithm.

In our test on the movie reviews, iSA estimated that 51.4% of the reviews were negative (std. error 0.01) and 48.6% were positive (std. error 0.01) – very close to the actual 50:50 distribution within the overall corpus. While the classification algorithms also came close to estimating the correct distribution, this was largely due to the balance of mis-classifications (positives labelled as negative, and vice versa) being evenly distributed. If the sentiments were not evenly divided in the corpus, or if there were additional categories (e.g. a 'neutral' or 'off-topic' category), as already noted above, this could have produced significant skew in the results of the classification algorithms, which the aggregate algorithms would have handled more robustly. Furthermore, the aggregate algorithms are better able to cope with smaller amounts of labelled training data, while the performance of classification algorithms can drop off quickly when the amount of training data is reduced. For example, if we reduce the amount of labelled data in the above example to 1,000 reviews (a reasonably realistic number for many research projects), the accuracy of the best classification algorithm we could train fell to 77.5% – a drop of more than 6% – while iSA's estimation remains extremely accurate; in fact, its accuracy improved despite the smaller amount of training data, in this case estimating the split in the corpus at 50.1% to 49.9%.

# References


Aggarwal, Charu C., ed. 2014. *Data Classification: Algorithms and Applications*. CRC Press.

Baccianella, Stefano, Andrea Esuli and Fabrizio Sebastiani. 2010. Sentiwordnet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining. *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC '10),* Pp. 2200–2204.

Bird, Steven, Ewan Klein and Edward Loper. 2009. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly Media.

Ceron, Andrea, Luigi Curini and Stefano Maria Iacus. 2016. iSA: A Fast, Scalable and Accurate Algorithm for Sentiment Analysis of Social Media Content. *Information Sciences* 367–368: 105–124.

Ceron, Andrea, Luigi Curini and Stefano Maria Iacus. 2017. *Politics and Big Data: Nowcasting and Forecasting Elections with Social Media*. Routledge.

Ceron, Andrea, Luigi Curini and Stefano Maria Iacus. 2019. ISIS at Its Apogee: The Arabic Discourse on Twitter and What We Can Learn from That about ISIS Support and Foreign Fighters. *Sage Open*. https://doi.org/10.1177/2158244018789229

Cunliffe, Emma, and Luigi Curini. 2018. ISIS and Heritage Destruction: A Sentiment Analysis. *Antiquity* 92(364): 1094–1111.

Dave, Kushal, Steve Lawrence and David M. Pennock. 2003. Mining the Peanut Gallery: Opinion Extraction and Semantic Classification of Product Reviews. In *Proceedings of the 12th International Conference on World Wide Web*, pp. 519–528. ACM.

Dreiseitl, Stephan and Lucila Ohno-Machado. 2002. Logistic Regression and Artificial Neural Network Classification Models: A Methodology Review. *Journal of Biomedical Informatics* 35(5): 352–359.

Fahey, Robert A., Tetsuya Matsubayashi and Michiko Ueda. 2018. Tracking the Werther Effect on Social Media: Emotional Responses to Prominent Suicide Deaths on Twitter and Subsequent Increases in Suicide. *Social Science & Medicine* 219: 19–29.

Figueroa, Rosa L., Qing Zeng-Treitler, Sasikiran Kandula and Long H. Ngo. 2012. Predicting Sample Size Required for Classification Performance. *BMC Medical Informatics and Decision Making* 12(1): 8.

Gilbert, C. J. and Eric Hutto. 2014. Vader: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text. In *Eighth International Conference on Weblogs and Social Media (ICWSM-14).* http://comp.social.gatech.edu/papers/icwsm14.vader.hutto.pdf, accessed 5 November 2018.

Goldberg, Yoav. 2016. A Primer on Neural Network Models for Natural Language Processing. *Journal of Artificial Intelligence Research* 57: 345–420.

Grimmer, Justin and Brandon M. Stewart. 2013. Text as Data: The Promise and Pitfalls of Automatic Content Analysis Methods for Political Texts. *Political Analysis* 21(3): 267–297.

Haselmayer, Martin and Marcelo Jenny. 2017. Sentiment Analysis of Political Communication: Combining a Dictionary Approach with Crowdcoding. *Quality & Quantity* 51(6): 2623.

Hopkins, Daniel J. and Gary King. 2010. A Method of Automated Nonparametric Content Analysis for Social Science. *American Journal of Political Science* 54(1): 229–247.

Hu, Minqing and Bing Liu. 2004. Mining and Summarizing Customer Reviews. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 168–177. ACM.

Laver, Michael and John Garry. 2000. Estimating Policy Positions from Political Texts. *American Journal of Political Science* 44(3): 619–634.

Loughran, Tim and Bill McDonald. 2011. When Is a Liability Not a Liability? Textual Analysis, Dictionaries, and 10-Ks. *The Journal of Finance* 66(1): 35–65.

Maas, Andrew L., Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, Christopher Potts. 2011 Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 142–150. Portland, Oregon, USA: Association for Computational Linguistics. http://www.aclweb.org/anthology/P11-1015.

Mohammad, Saif M. 2016. Sentiment Analysis: Detecting Valence, Emotions, and Other Affectual States from Text. In Meiselman, Herbert (ed.), *Emotion Measurement*, pp. 201–237. Elsevier.

Mohammad, Saif M. and Peter D. Turney. 2013. Crowdsourcing a Word-Emotion Association Lexicon. *Computational Intelligence* 29(3): 436–465.

Nielsen, Finn Årup. 2011. A New ANEW: Evaluation of a Word List for Sentiment Analysis in Microblogs. ArXiv Preprint ArXiv:1103.2903.

Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Edouard Duchesnay. 2011. Scikit-Learn: Machine Learning in Python. *Journal of Machine Learning Research* 12: 2825−2830.

Pennington, Jeffrey, Richard Socher and Christopher Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543.

Rooduijn, Matthijs and Teun Pauwels. 2011. Measuring Populism: Comparing Two Methods of Content Analysis. *West European Politics* 34(6): 1272–1283.

Rudkowsky, Elena, Martin Haselmayer, Matthias Wastian, Jenny Marcelo, Stefan Emrich, Michael Sedlmair. 2018. More than Bags of Words: Sentiment Analysis with Word Embeddings. *Communication Methods and Measures* 12(2–3): 140–157.

Sardinha, Jovan. 2017. An Introduction to Model Ensembling – Weights and Biases. *Medium*. https://medium.com/weightsandbiases/an-introduction-to-model-ensembling-63effc2ca4b3, accessed 24 June 2019.

Silge, Julia and David Robinson. 2017. *Text Mining with R: A Tidy Approach*. O'Reilly Media.

Smailović, Jasmina, Janez Kranjc, Miha Grčar, Martin Žnidaršič and Igor Mozetič. 2015. Monitoring the Twitter Sentiment during the Bulgarian Elections. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015*. IEEE International Conference, pp. 1–10. IEEE.

Socher, Richard, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, Christopher Potts. 2013 Recursive Deep Models for Semantic Compositionality over a Sentiment Treebank. In *Proceedings of EMNLP*, pp. 1631–1642.

Wang, Hao, Dogan Can, Abe Kazemzadeh, François Bar and Shrikanth Narayanan. 2012. A System for Real-Time Twitter Sentiment Analysis of 2012 US Presidential Election Cycle. In *Proceedings of the ACL 2012 System Demonstrations*, pp. 115–120. Association for Computational Linguistics.

Wiedemann, Gregor. 2018. Proportional Classification Revisited: Automatic Content Analysis of Political Manifestos Using Active Learning. Social Science Computer Review 37(2): pp. 135-159.

Wilson, Theresa, Janyce Wiebe and Paul Hoffmann. 2005. Recognizing Contextual Polarity in Phrase-Level Sentiment Analysis. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pp. 347–354. Association for Computational Linguistics.

Wolpert, David H. 1992. Stacked Generalization. *Neural Networks* 5(2): 241–259.

[1] Sentiment analysis is just one type of analysis a dictionary method can perform. The general concept of dictionaries makes them relatively easy and cheap to apply across a variety of problems, including the identification of words that separate categories (for example policy categories) and measuring the frequency of those words in different texts. See Laver and Garry (2000) for an approach in this regard.

[2] To assess model performance, beyond accuracy, other commonly used statistics are recall (a measure of what proportion of actual instances of a given category the algorithm correctly identified) and precision (a measure of how many of the times the algorithm identified a category were actually correct, as against how many times were false positives). The f1 score is another commonly used measurement that combines both recall and precision on a per-category basis.

[3] For a supervised learning approach applied to Arabic language, see for example Ceron et al. (2019) or Cunliffe and Curini (2018); an example in the Japanese language can be found in Fahey et al. (2018).

[4] For a recent review of works within the social science field that apply such approach, see Ceron et al. 2017.

[5] The pre-processing stage of the analysis (that is, the series of decisions a researcher makes on which types of word or features of a text to include or exclude from analysis and how to treat those text features, for example whether to apply stemming and lemmatisation) can have large consequences for the quality of the results of any automated text analysis. See Haselmayer and Jenny (2017) for a study that shows how pre-processing decisions impact on sentiment analysis.

[6] See: http://robfahey.co.uk/sentiment_example.zip or http://www.luigicurini.com/uploads/6/7/9/8/67985527/appendix_handbook.rar