Rheinische Friedrich-Wilhelms-Universität Bonn
Institute of Computer Science III

SentiMenthal
Text Mining on Smartphones

Master Thesis

Supervisor: Jun.-Prof. Dr. Alexander Markowetz

Second Supervisor: Prof. Armin B. Cremers

Andrii Karmeliuk

Bonn, January 23, 2014

Declaration of Authorship

I hereby certify that this thesis has been composed by me and is based on my own work, unless stated otherwise. No other person's work has been used without due acknowledgement in this thesis. All references and verbatim extracts have been quoted. Every source of information, including graphs and data sets, has been specifically acknowledged.

Bonn, January 23, 2014 Andrii Karmeliuk

Abstract

With stress and mental diseases spreading rapidly in developed countries, it is becoming important to increase individual awareness about own state of mind. To help people track their emotional condition, we developed a semi-automated approach utilizing the achievements in computer science, psychology and linguistics. The result is presented in a form of an application that targets users of smartphones. We concentrate our analysis on short textual messages, which generate large amounts of informative and easily accessible data. With apparition of large social communication platforms such as Facebook, WhatsApp and Skype, textual data gathering has become broadly available. We build a smartphone application (SentiMenthal) for performing an in-place sentiment analysis of messages sent by users throguh the aforementioned platforms. The application uses proven techniques from machine learning and opinion mining fields. Additionally, multiple charting options for in-depth analysis are provided.

Contents

1 Introduction

With stress and mental diseases spreading rapidly in developed countries, it is becoming important to increase individual awareness about own state of mind. Traditional psychological and psychiatrical approaches do not seem to solve the issue effectively since many people are reluctant to resort to counseling. Additionally, standard psychological treatment methodologies involve a requirement for patient's self-analysis. The results of such assessment are often biased towards one's perception of the situation, and they become distorted since memories fade over time and are susceptible to influence of current mental state. Therefore, a more practical, scalable, precise and effort-preserving solution needs to be found.

Tremendous amount of information that people transfer with the help of mobile devices introduces novel ways to track human behavior. Users produce numerous opinions regarding different topics. In this work, we created an application for smartphones to extract and analyze personal messages of users. The main contribution of this thesis is allowing regular users of smartphones to track their mood fluctuations based on private messages they are sending. Such application aims at improving self-awareness of users and potentially helping psychiatrists diagnose their patients. The application delivers seamless process of gathering, analyzing and presentation of data.

Our work would not have been possible without tremendous technical progress in the area of mobile devices during the last five years. Technological advances have enabled regular pocket devices to process large amounts of data. Particularly, smartphones have become ubiquitous in everyday life, and this market continues to grow. These portable devices are becoming hubs for electronic communications, shopping and web-browsing. Apparition of an open-source Android operating system has open ways for developers to explore and harness computational power of smartphones to its full extent.

Substantial amount of work has been done in the area of sentiment analysis of freely available text messages. Biggest attention was devoted to the Twitter microblogging service, a social communication hub for virtually instant exchange of short public messages - “tweets”. The service is interesting for researches particularly due to its vast database of user-generated messages. The length limit of 140 symbols for tweets creates challenges for processing such data since many noise tokens are introduced, such as new abbreviations, slang, and misspellings. To our best knowledge, none of the published works address sentiment analysis problem on mobile devices.

To help people track their emotional condition, we develop a semi-automated approach to sentiment analysis of text messages sent from mobile devices. The result is presented in a form of an application that targets users of smartphones. We concentrate our analysis on short textual messages, which generate large amounts of informative and easily accessible data. With apparition of large social communication platforms such as Facebook, WhatsApp and Skype, textual data gathering has become broadly available. We build a smartphone application called SentiMenthal for performing an in-place sentiment analysis of messages sent by users through the aforementioned platforms. The application uses proven techniques from machine learning and opinion mining fields. Additionally, multiple charting options for in-depth analysis are provided.

Our solution is innovative since it prefers seamless work and high performance over accuracy and variety of features. The application is developed with mobile devices limitations in mind. A proper attention is given to functional and intuitive user-interface. Security of sensitive data is also taken into account: no user data is transferred elsewhere from the owner’s mobile device.

The rest of the thesis is organized as following. First, we discuss the foundations of psychology of emotions and moods. In Chapter 3 we give a brief overview of general notions within the Natural Language Processing field. Chapter 4 provides technical details and realizations of popular approaches in the field of Sentiment Analysis. In Chapter 5 we give a brief overview of the structure of Android operating system platform. Chapter 6 introduces the SentiMenthal application developed alond with details of core architectural decisions. In Chapter 7 we continue with implementation details of the text preprocessing module. Next, we describe implementation techniques and decisions for text analysis module. Chapter 9 outlines the results gained from synthetic evaluation of the application. Finally, we conclude our work with the discussion of the results and present our vision on potential future developments on the topic.

2 Emotion and Mood

Emotions as a unique and complex artifact of reactive human behaviour have been studied for decades. However, there is still no absolute agreement on the precise definition of the term "emotion". The terms *emotion*, *mood* and *affect* are often used interchangeably. Thus, to avoid confusion, it is important to clarify major differences.

To start with, "affect" is a generic term used to describe both mood and emotion classes. It can be seen as a superclass for the two. Moods and emotions are similar in the sense that they reflect human feelings, but contradicting in other aspects [Fernández-Tobías et al., 2013]:

- Emotions are provoked by specific events, but source of moods apparition stays mainly unidentified.
- Emotions are fast-paced and intense, whereas moods are long-lasting and of low intensity.
- Emotions are classifiable into numerous distinct entities, while moods are generally split into two categories: ones with positive and negative effect.
- Emotions are often revealed by explicit facial expressions, in contrast, moods rarely show obvious impact.
- Emotions are believed to be reactive and not controllable, self-initiating in response to incoming events. Moods have a cognitive nature and reveal themselves after deep thought processes.

Thus, mood can be seen as a compound of emotions, differing from them in intensity, influence and duration.

Emotions could be ignited by various factors of human interaction with the surrounding world as well as by one's internal factors. External stimuli are physical (cognitition through 5 human senses). Internal stimuli are physiological, induced by one's hormonal state or memory and thought process state. All mentioned factors also overlap with the person's initial mental state.

When we talk about emotion or mood detection, we implicitly assume that they are extracted them from a single source, e.g. text or speech. This task turns out to be highly sophisticated since even a very slight change in wording (text) or tone (speech) may be a sign of the completely different meaning for the whole processed unit of communication. Negation, sarcasm and irony detection provide a challenge for text and speech analysis. Currently the best natural language processing toolkits are barely capable of detecting these aspects of communication. We will discuss the technical challenges in the chapter 3. In order to successfully identify aforementioned objectives and work with them, precise terminology and background facts should be stated clear.

The chapter proceeds as follows: first, we describe emotions and mood in detail; then we give an overwiev of existing models in emotion's assessment along with their advantages and disadvantages; then we list generic means of emotion and mood detection; finally, we conclude with statistics regarding world mental disease spread and transit to a discussion of particular problems solved by natural language processing in the next chapter.

2.1. Emotion and Mood

Emotions and moods are an essential part of any human being. Emotions can be considered as a certain response of a self to incoming information. Moods, as mentioned before, can be seen as composed out of emotions and represents a state originated from a combination of emotions.

A reasonable and systematic attempt to improve the understanding and classiffying phenomena of emotions has been made by defining and discussing of core properties of emotion. They are the following:

To better understand the nature of emotions, we briefly review their core properties. For example, [Scherer, 2005] proposed the following characteristics:

- *Rapidity of Change*: Emotional state is seen as non-discreet and modifying swiftly in time. Rapid changes in the environment, with which an individual interacts, as well as changes in his internal conditions account for this property. In turn, this implies the need for constant reevaluation and adjustment to the environment. Different emotions are believed to change with varying rapidities.
- *Intensity*: Relatively high intensity of response activity is a distinguishing feature of emotions (e.g. compared to moods). Different emotions, in turn, also vary in intensity: for instance, annoyance, anger and rage have similar response patterns, but differ with respect to magnitude.

- *Duration*: The duration of an emotion must be relatively short in order not to tax the resources of the organism and to allow behavioral flexibility.
- *Behavioral Impact*: Following from the items described above, this global property that reflects how certain emotion can influence individual choices, decision making and also mood as a consequence.

Knowing the main distinguishing characteristics of an emotion, we can proceed to considering possible mechanisms that cause emotions to emerge. It is still an open question how human beings produce emotions in response to events that intiate them. Five competing theories regarding this matter can be highlighted: Cannon-Bard Theory, Facial Feedback Theory, James-Lange Theory, Lazarus Theory and Schachter-Singer Theory [allpsych.com, 2013].

Cannon-Bard Theory (Figure 2.1) emphasizes the role of the brain in producing physiological responses. "Theory argues that we experience physiological arousal and emotional at the same time, but gives no attention to the role of thoughts or outward behavior."

James-Lange Theory (Figure 2.3) "argues that an event causes physiological arousal first and then we interpret this arousal. Only after our interpretation of the arousal can we experience emotion. If the arousal is not noticed or is not given any thought, then we will not experience any emotion based on this event."

Lazarus Theory (Figure 2.4) "states that a thought must come before any emotion or physiological arousal. In other words, you must first think about your situation before you can experience an emotion."

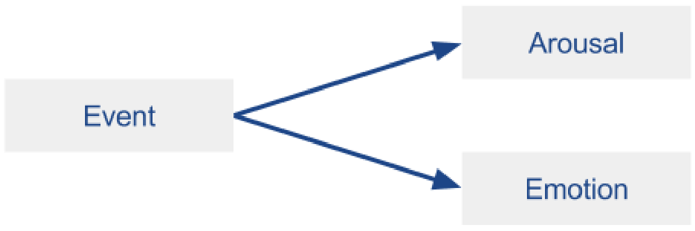


Figure 2.1.: Cannon-Bard Theory Schema.

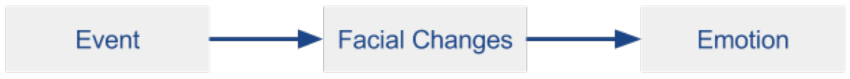


Figure 2.2.: Facial Feedback Theory Schema.

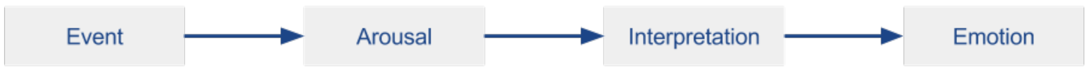


Figure 2.3.: James-Lange Theory Schema.

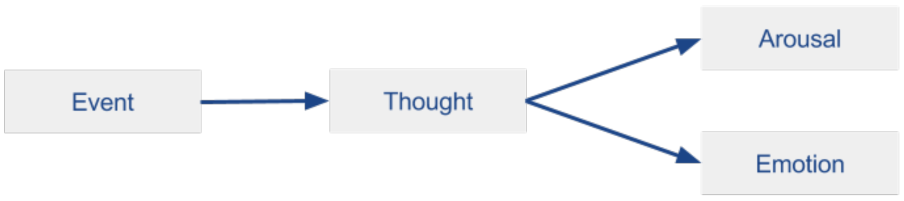


Figure 2.4.: Lazarus Theory Schema.

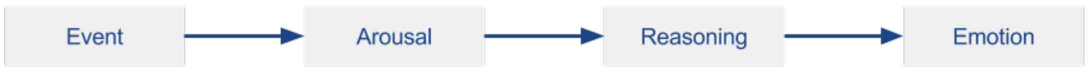


Figure 2.5.: Schachter-Singer Theory Schema.

Schachter-Singer Theory (Figure 2.5) "an event causes physiological arousal first. You must then identify a reason for this arousal and then you are able to experience and label the emotion."

Facial Feedback Theory (Figure 2.2) "According to the facial feedback theory, emotion is the experience of changes in our facial muscles. In other words, when we smile, we then experience pleasure, or happiness. When we frown, we then experience sadness. it is the changes in our facial muscles that cue our brains and provide the basis of our emotions. Just as there are an unlimited number of muscle configurations in our face, so to are there a seemingly unlimited number of emotions." This theory is partially verified by experiments described in [Kahneman, 2011].

The first four of the described above theories range in proximity to two competing approaches to identification of the driving force of emotions: physiological (physical) and cognitive (mental). For instance, the Cannon-Bard theory emphasizes the importance of the central nervous system in triggering emotions [Pelachaud, 2013], taking the side of the cognitive approach. In contrast, the James-Lange theory picks in favor of the physical approach, as it puts physiological arousal in front of any emotion. Modern theories such as Schachter-Singer Theory rely on the combination of the two approaches.

Finally, we want to provide a short overview of the classifications of emotions. This issue has been subject to hot debate in the literature, with lots of different taxonomies proposed. Here we mention the most prominent approaches, which categorize emotions with respect to the way people interpret them: either as discrete entities or as a continuous state flowing from one emotion to another. Discrete models attempt to structurize all discrete emotions into some categorization scheme. Such approach is not always viable and easy to follow since researchers tend to use inconsistent sets of classes for categorization and some classes could not be fully understood in foreign languages [Zimmermann et al., 2006]. In contrast, continuous models (also referred to as dimensional models) allow for fuzzy boundaries between different emotional states. For example, the Circumplex model uses the notion of *core affect* to describe shared concept of emotion and mood. It is involved in apparition of both mood and emotion phenomena, and differs in values of its two constituents: *arousal* and *valence* [Russell, 2003][Scherer, 2005]. Figure 2.6 illustrates schematic representation of the core affect concept.

Another example of dimensional model is a variation of Circumplex – Plutchik's Wheel of Emotions. It comprises eight basic emotions: anger, anticipation, disgust, fear, joy, sadness, surprise, and trust. These emotions enter this classification as essential since humans developed them through evolutionary processes (behavioral adaptation). The model allows several emotions to be experienced simultaneously, giving rise to a broad spectrum of possible complex emotions such as affection, confidence, euphoria, loneliness, curiosity (see Figure 2.7). To put it briefly, Plutchik's theory is built around six basic postulates [Plutchik, 1991]:

1. There is a small number of pure or primary emotions.

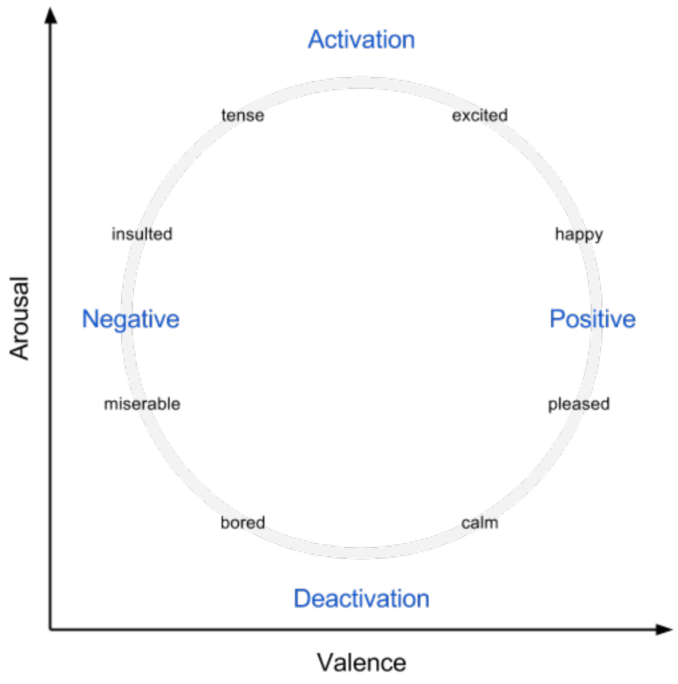


Figure 2.6.: Core Affect - Arousal on the vertical axis, valence on the horizontal axis.

2. All other emotions are mixed; that is, they can be synthesized by various combinations of the primary emotions.
3. Primary emotions differ from each other with regard to both physiology and behavior.
4. Primary emotions in their pure form are hyphothetical constructs or idealized states whose properties can only be inferred from various kinds of evidence.
5. Primary emotions may be conceptualized in terms of pairs of polar opposites.
6. Each emotion can exist in varying degrees of intensity or levels of arousal.

Note that Plutchik's approach can separate basic emotions from derived and compound ones. If both basic and derived types are present in a classification scheme, then it is called *multilevel*. If all emotions are classified as equal and independent of each other, then we would have a *horizontal* (*single-level*) scheme [Ekman and Davidson, 1994]. Even though such attempts to categorize have been made (as in [Ekman and Davidson, 1994]), the taxonomy is still not completely established [Russell and Barrett, 1999].

As we can see, there is no unique approach to classification of emotions. However, the vast majority of theories distinguish between positive and negative emotions

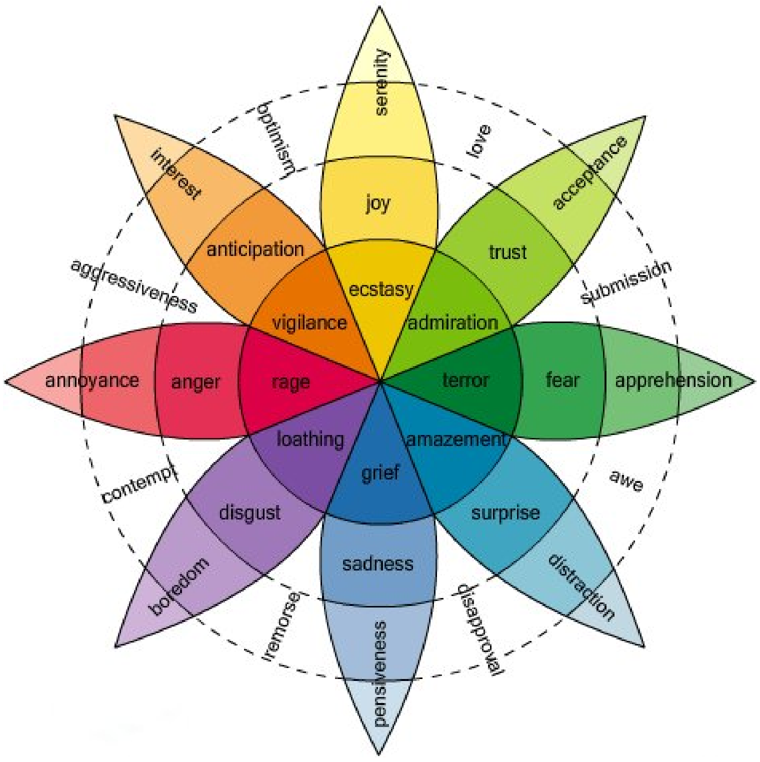
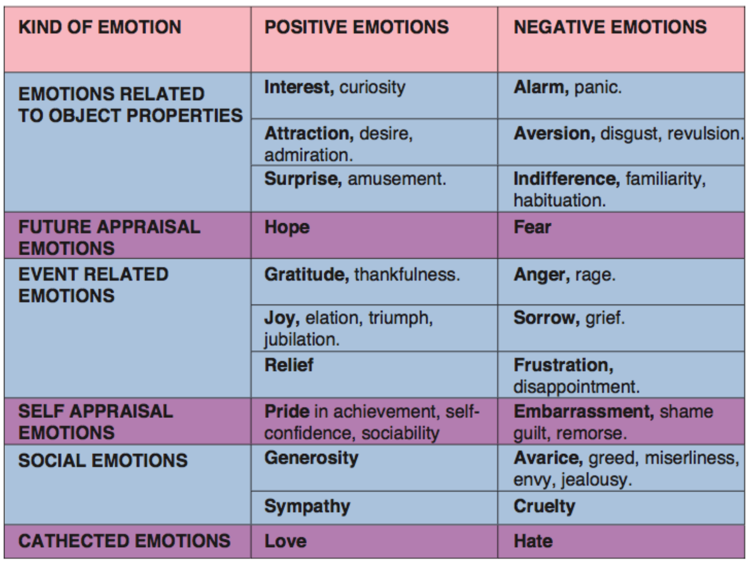


Figure 2.7.: Robert Plutchik's Wheel of Emotions.

(see [Solomon and Stone, 2002] for the discussion). For instance, [Robinson 2009] proposed one of the categorization of emotions into positive and negative classes, which are presented in 2.8. This distinction also finds support in neuroscience: the valence hypothesis states that the right hemisphere of the human brain is specialized for negative emotions, while the left hemisphere is dominant for positive emotions. Multiple neuroimaging studies provide evidence in favor of valence hypothesis (see [Demaree et al., 2005] and [Wager et al., 2003] for an overview), as well as earlier eye-tracking and facial electromyography studies ([Ahern and Schwartz, 1979], [Schwartz et al., 1979]). Taking into account the absence of agreement on the taxonomy of emotions and potential difficulty of detecting slight differences between closely related feelings, we focused on the detection of positive and negative emotions when developing our application.

2.2. Emotion and Mood Detection

Emotion and mood detection is a task of classifying units of human communications according to a predefined set of labels. Difficulties with precise classification arise from the fact that natural language constructs can have different combinations of sentiment and subjectivity. Moreover, emotions are communicated through a large set of tools, such as facial expressions, voice, body language and skin color. Figure 2.9 illustrates all possible relations for Emotion, Opinion, Objectivity, Subjectivity



KIND OF EMOTION	POSITIVE EMOTIONS	NEGATIVE EMOTIONS
EMOTIONS RELATED TO OBJECT PROPERTIES	**Interest,** curiosity	**Alarm,** panic.
	Attraction, desire, admiration.	**Aversion,** disgust, revulsion.
	Surprise, amusement.	**Indifference,** familiarity, habituation.
FUTURE APPRAISAL EMOTIONS	**Hope**	**Fear**
EVENT RELATED EMOTIONS	**Gratitude,** thankfulness.	**Anger,** rage.
	Joy, elation, triumph, jubilation.	**Sorrow,** grief.
	Relief	**Frustration,** disappointment.
SELF APPRAISAL EMOTIONS	**Pride** in achievement, self-confidence, sociability	**Embarrassment,** shame guilt, remorse.
SOCIAL EMOTIONS	**Generosity**	**Avarice,** greed, miserliness, envy, jealousy.
	Sympathy	**Cruelty**
CATHECTED EMOTIONS	**Love**	**Hate**

Figure 2.8.: Eleven pairs of positive and negative emotions [Robinson, 2009].

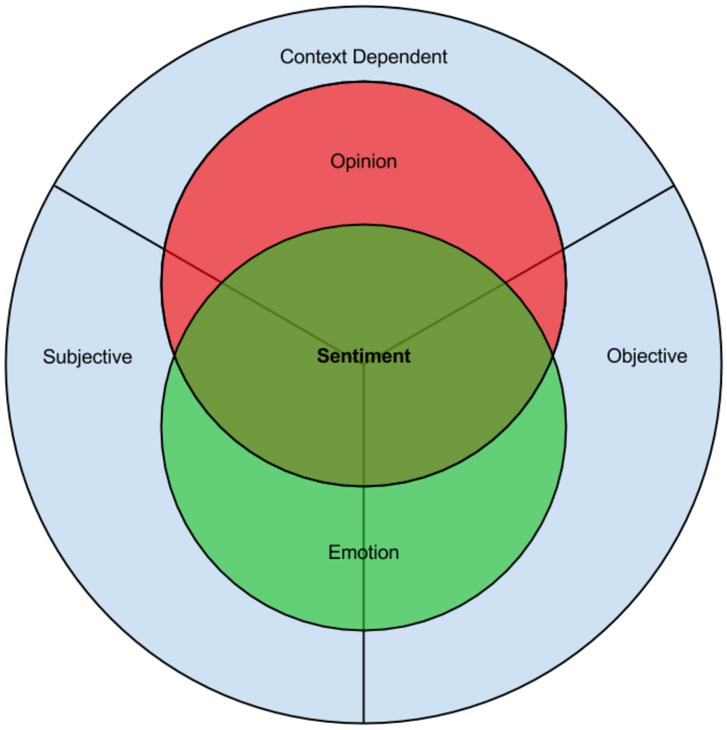


Figure 2.9.: Emotion, Opinion, Objectivity, Subjectivity and Context Dependency relations.

and Context Dependency. As we can see, the presence of one of any aspect does not imply the presence of any other. As well as the presence of an aspect may "result" in any of the outcomes - subjective, objective or it can be context dependent.

2.2.1. Recognition in Text

Written text carries large but limited amount of information regarding emotional state of the creator. When we have text as an input data for emotion recognition, this entails the use of predefined vocabularies of sentiment-rich words. Eventual processing is done with help of tools available from Natural Language Processing (NLP) (so called lexical-based approach) and machine learning branches of research. These two approaches are discussed in detail in Chapter 4. From a structural point of view, emotion detection in written text relies on investigating the following concepts: emoticons, n-grams, bag-of-words, part of speech tagging etc.

Sometimes it is truly hard for people to express their feelings properly by merely writing a text. *Emoticons* were invented as means of quick and unambiguous communication of feelings. They usually consist of 2-3 characters and reflect some certain emotion, mood or condition. Although, known to be used since 1857, a true rise of popularity of emoticons took place with computer-based communications becoming common. A first man to propose a standardized use of these special expressions on the Internet was Scott Fahlman. In 1982 he posted his suggestion on the computer science general board of the Carnegie-Mellon University [CMU.edu, 2013]. These days, emoticons or "*smileys*" are used ubiquitously, hundreds of symbol combinations exist with corresponding graphical representations. Sufficient amounts of data can be gathered to make statistics show interesting patterns in usage - according to gender, geographical location, rural vs. urban place of living. Potentially, emoticons can be used for personality traits detection. With respect to symbol representation, are two classes of emoticons: western-type and eastern-type. The latter are known to be harder to detect and interpret due to higher complexity and more variations[Ptaszynski et al., 2011].

3 Natural Language Processing

Over 7 billion of people living on Earth do communicate with help of 6500 living languages. Among most widespread are: Standard Chinese (1365 million), English (1000 million), Spanish (528 million), Hindi (490 million) [ling.gu.se, 2013, Ethnologue.com, 2013a, Ethnologue.com, 2013b]. In science these languages are referred to as "Natural Languages", therefore the respective field of study is known as "Natural Language Processing" (NLP). Of biggest interest in the field are the following tasks: machine translation, sentiment analysis (opinion mining), question answering, speech recognition, information extraction. Each of these tasks can be split into generic subtasks, such as part-of-speech tagging (POS-tagging), negation handling, stemming, lemmatization to name a few.

3.1. Text Preprocessing

Textual data written by humans needs preprocessing cleansing. Natural languages are hard to work with due their high complexity and linguistic variability. In simple words, there are many ways of saying the same, but with different words. Negation, sarcasm and irony currently are the most difficult problems in natural language processing. Additionally, even for humans it is a hard task to detect metaphorical language. These aspects of natural language make context and sentiment analysis an extremely complicated tasks. Moreover, noise in forms of misspellings, abbreviations, slang words is frequent in texts sent over social networks.

To overcome noise and put textual data to an organized form, a text-preprocessing phase is required. And to make available machine learning enabled classificators for mobile devices applications - a noticable reduction of veature space sizes is needed. Further we provide a list of most popular and proven text-preprocessing techniques:

- Stemming and lemmatization. These two text-preprocessing techniques are similar and often mixed up. Stemming is the process of extraction of a stems (syntactical roots) from words. Similar words that have overcome stemming are mapped to a shared stem. Words that share the same stem do not necessarily have equal lexical bases, but are rather similar in sets and combination of symbols they consist from. For instance, words *equals*, *equalizing* and *equalled* are stemmed by Porter Stemmer algorithm to a stem *equal*. Stemming can lead to loss of contextual information since it only takes into account a structural similarities of words. On the other hand, lemmatization technique takes into account the context of the word by performing a dictionary lookup. A difference between the two methods becomes clear, if we compare how they handle verb forms, comparative and superlative adjective forms and other morphology variating language constructions. A stemmer does not associate words such as *old-elder-eldest,* to be of a same kind, but a lemmatizer is able to reduce all three to *old*. In turn, lemmatization is computationally more intensive and may be infeasible for mobile devices.
- Stopwords removal. A remarkably simple, yet powerful method. Stopwords are words that do not carry any meaning or sentiment influence when used. In general case, these words can be removed without loss of valuable information. However, an attention should be given not to break named entities, as they may contain stopwords. These words constitute a substantial part in texts (30-30% of corpus size), therefore, their removal may be beneficial for applications with limited resources.
- Spelling correction. Required approach for fixing of misspelled words (intentionally or not). Informar language contains large amounts of spelling mistakes and invented abbreviations. Another type of errors fixed by this method is intentional misspelling by addition of extra (repeated) letters, as in *helllooooo*.
- Replacement of named entities. Removal of personal names of entities - locations, people's names, brands, company names.

3.2. Estimating Vocabulary

Natural language processing is about working with large amounts of textual data. This implies working with large term dictionaries for separate languages. An efficient data-structure for this application type is needed. In 1970 Burton Bloom ([Bloom, 1970]) introduced a technique to work with large sets of data. Bloom filter matches well for this purpose. It is a probabilistic data-structure, meaning it allows to save space and lower the look-up times on large datasets.

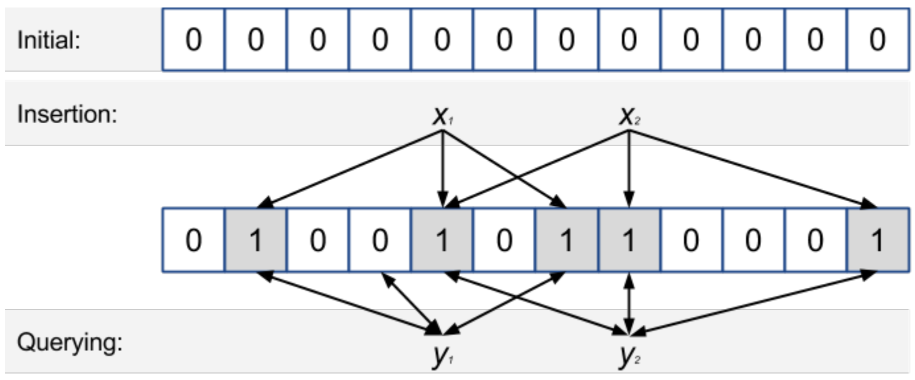


Figure 3.1.: Bloom Filter.

[W3.org, 2013]

Bloom filter is essentially a bit-string of length m, representing a set $S = \{x_1, x_2 ... x_n\}$ of n elements. An empty filter has all bits set to 0 (Figure 3.1). Elements are added with help of k different hash-finctions $h_1 ... h_k$. Each hash-function takes a new element on input and outputs a postition in bit-string. A corresponding bit is then set to 1. Chosen hash-functions have to have the output given with a uniform random distribution in the range $\{1, ..., m\}$. An element addition is made by feeding it to the k hash-functions and setting the corresponding bits to 1.

(the number k and type of hash-functions can differ depending on application)

Element can be tested for inclusion in a similar manner. But instead of altering the bit-string, we check the bits according to given by hash-functions array of indices. Element is considered to be present in the set only if all the bits checked are 1. The chosen hash-functions are designed in the way, that their output values may coincide while given different input values. This means that some string $s1$ may trigger the Bloom filter to show for $s2$ a true inclusion. More precisely, Bloom filter may give false positives, false negatives are impossible. In general, it is impossible to remove an element after it has been added to the filter.

False positive rate. To show applicability of a chosen filter setup, we can estimate a probability of having a false positive result. Given all n elements of S are inserted to the filter, the probability that a specific bit is 0 is:

$$p' = (1 - \tfrac{1}{m})^{kn} \approx e^{-kn/m}.$$

Also, let $\rho = \frac{\text{\# of 0-bits}}{m}$, then $\mathbb{E}(\rho) = p'$, therefore the probability of a false positive:

$$(1 - \rho)^k \approx (1 - p')^k \approx (1 - p)^k.$$

Bloom filter size. To pick the right bit-string length for our Bloom filter, we depart from a false positive rate ϵ we are willing to tolerate. We derive a lower bound on m, number of bits. If u is the size of the universe U, such that $U \supset S$, then our m-bit string has to be able to map all $\binom{u}{n}$ sets. For each set $X \in S$, let $F(X)$ be a string of length m (mapping). Since we are looking for a lower bound, consider all sets S', such that $S' \cap S = \emptyset$ and $S' \subset U$. All sets S' are *rejected,* i.e., $F(X')$, where $X' \in S'$, does not produce a valid (unique) bit-string. All sets $X \in S$ are in turn *accepted*: m-bit string $s = F(X)$. For any particular X accepted by s, it may also accept $\epsilon(u-n)$ other elements of the universe. Therefore, string s accepts at most $n + \epsilon(u-n)$ elements. It can also represent any of the $\binom{n+\epsilon(u-n)}{n}$ subsets of $n + \epsilon(u-n)$ of size n, but no more. When we use a bit-string of length m, we can have 2^m distinct strings. These strings must represent all of the $\binom{u}{n}$ sets. We get the following:

$$2^m \binom{n+\epsilon(u-n)}{n} \geq \binom{u}{n} \Leftrightarrow m \geq \log_2 \frac{\binom{u}{n}}{\binom{n+\epsilon(u-n)}{n}} \approx \log_2 \frac{\binom{u}{n}}{\binom{\epsilon u}{n}} \geq log_2 \epsilon^{-n} = n \log_2(1/\epsilon).$$

Bloom filter vs. Hashing. During hashing, each element of a set is hashed into $\Theta(\log n)$ bits, where n is the number of elements in a set. Given we opt to use $2\log_2 n$ bits per element, the resulting hash table size will be $2n \log_2 n/8$ bytes with a collision probability $1/n^2$. As we can see the size of such table can become large with big number of elements we are willing to hash. Additionally, the probability of a false positive will be $n/n^2 = 1/n$. The matter of choice between the two approaches boils down to a decision whether we want to accept a relatively higher error rate in exchange for space savings.

3.3. Evaluation Measures

We evaluate classification performance using four well-known metrics: accuracy, precision, recall and F-measure. Accuracy is the simplest, yet indicative performance measure. The formula is the following:

$Acc = \frac{TP+TN}{TP+TN+FP+FN}$

The main reason that makes classifier performance evaluation task difficult is the highly unbalanced data. It often appears that the data given is skewed, i.e., there are significantly more elements of one class than of the others. We use the folloing generic concepts(?) for evaluation: *true positive (TP)* - a sample is classified as positive and has a positive label; *true negative (TN)* - a sample is classified as negative and has a negative label; *false positive (FP)* - a sample is classified as positive, but has a negative label; *false negative (FN)* - a sample is classified as negative, but has a positive label.

For some applications we would like to keep track of certain types of errors. Whether we want to minimize occurrences false positives or false negatives - we use *precision* or *recall* measures respectively.

Precision: $\pi = \frac{TP}{TP+FP}$

Recall: $\rho = \frac{TP}{TP+FN}$

Depending on application, one measure is more important than the other. We can easily get recall of 1 by simply classifying all samples as positive. But, at the same time, precision will be very low. A standard measure to balance out the two is the *F-measure*. The F-measure is a harmonic mean of precision and recall:

$F = \frac{1}{\alpha/\pi+(1-\alpha)/\rho} = \frac{(\beta^2+1)\pi\rho}{\beta^2\pi+\rho}$ where $\beta^2 = \frac{1-\alpha}{\alpha}$

The default setup used is the balanced F-measure, where precision and recall get equal weights. For this we set $\alpha = 1/2$ or $\beta = 1$ [Manning et al., 2008]. The resulting measure is known as F_1-measure:

$$F_1 = \frac{2\pi\rho}{\pi+\rho} = \frac{2TP}{2TP+FN+FN}$$

3.3.1. Resubstitution and x-Fold Cross Validation Testing

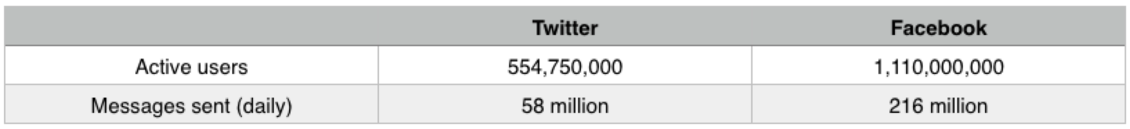
Are two variants for efficient and representative evaluation of classification performance. These two methods account for scarce dataset and, therefore, perform reasonably well with such cases. The first one makes an accuracy assessment by running classification testing with the exact same dataset, that was used for training. Resubstitution testing outputs a value that shows how well a classifier is trained with the given dataset. Put in other words, this can be a measure of how flexible, how fast can the classification model adapt to small datasets. On the other hand, the resulting value may be indicative of data consistency. If the dataset cantains many similar samples that contradict each other by labels - this may rseult in a low resubstitution value for the classifier. The x-fold cross validation testing gives output on how a classifier configuration is able to handle previously unseen data. Available dataset is split into two parts (as a rule of thumb 9/10 for training, 1/10 for testing) for every epoch of training. Commonly 5 to 10 training-testing rounds are sufficient to grasp the searched property value of a classifier.

4 Sentiment Analysis

Human emomtions, moods, opinions and decisions are studied by the sentinment analysis field, also known as opinion mining. A direct objective and an outcome of such analysis is a classification of selected subject. Classification can be binary, with arbitrary number of classes or even outputting continuous values. Subject selection for analysis varies widely by granularity (e.g., word - phrase - sentence - paragraph - document) and source (text, speech, facial expressions). As we mentioned in Chapter 3, this thesis focuses on sentiment analysis of text.

The area of research has gotten much attention with exponential growth of media communications popularity. Among the most influential are social networking (including blogging and micro-blogging platforms), online marketplaces, online gaming and other online entertainment services (e.g. Netflix on-demand video service or IMDB.com - internet movie database). Tremendous amounts of opinionated data are produced daily. And, more importantly, this data is stored and available to be processed. To illustrate the scale of data available for analysis, see Figure 4.1 [Statisticbrain.com, 2013a][Statisticbrain.com, 2013b].

Who could be interested in doing this kind of analysis of data? To start, the users (generators of the data) in pursuit of tracking own mood fluctuations - as a usecase for progressing quantified-self trend. On the other hand, there are vast



	Twitter	Facebook
Active users	554,750,000	1,110,000,000
Messages sent (daily)	58 million	216 million

Figure 4.1.: Twitter and Facebook Statistics.

possible applications for enterprises willing to get implicit feedback either from their current or potential customers. The feedback, naturally, can be obtained regarding the whole company, one product or particular service provided. Important to note, that obtaining this information is cost-effective and at the same time customer-friendly (i.e., no need for explicit customer satisfaction surveys).

Sentiment Analysis is a field of study within NLP with main objective in human emotions and opinions detection from written text. It has drawn subst with substantial proliferation of online communication (opinion sharing) platforms. This fact has made enormous amounts of user-generated data available for processing. As well, worth to mention, there are other important contributions to the field: technology advancements and new kinds of applications outbreak. Machine Learning algorithms have made an outstanding leap in performance and variety of available tools. High-quality and more important open-source machine learning tools became available to the general public, e.g., Apache Mahout [Apache.org, 2013], Weka [cs.waikato.ac.nz, 2013], MLOSS Platform [MLOSS.org, 2013], NLTK [NLTK.org, 2013].

Similar to many other fields, some ambiguity within terminology concerning sentiment analysis, opinion mining and subjectivity analysis exists. To clarify these matters we will establish definitions that will be used further on in this work. Term *Sentiment Analysis* was first recorded to be used in year 2001 in [Tong, 2001, Das and Chen, 2001] and was intended to be "used in reference to the automatic analysis of evaluative text and tracking of the predictive judgments" [Pang and Lee, 2008]. Currently, the term is used to describe a superclass of tasks including analysis of sentiment, opinions and subjectivity in texts. The first use of *Opinion Mining* was found in [Dave et al., 2003] and is directly associated with web-search and web-crawling, i.e. is in information retrieval field. *Subjectivity Analysis* is the recognition of opinion-oriented language in order to distinguish it from objective language [Pang and Lee, 2008].

In this thesis we utilize polarity classification of sentiment, meaning the sentiment is classified according to a one-dimensional (aspect, scale, axis?). More specifically, binary classification will be used for experimental evaluation. As for granularity aspect of classification, it can be chosen to vary from word level classification upto document level classification.

Further in this chapter we give an overview of the two conceptual approaches in the field of sentiment analysis - lexical and machine learning. We outline basic principles of most popular "tools" for either approaches.

4.1. Lexical Approaches

Lexical approaches for sentiment analysis span from the knowledge acquired in the field of linguistics. This set of methods uses data received by a detailed syntactical and ontological analysis of natural languages. History of development as well as geographically-driven(?) differences in use are taken into account. This said, following approaches evaluate and rate specific words regarding their meaning, cases of use, applicability and their interdependence. To outline the difference with machine learning approaches - on one hand lexical way of analysis is more precise regarding specicfic word, at the same time is constrained in terms of available dictionaries (it has no way to keep up with the momentum at which new words (memes, abbreviations) emerge); on the other are able to handle virtually infinite in size dictionaries with sufficient training data (in case of supervized methods), but are limited in level of "understanding" the language, especially vast enormous amount of possible interdependencies cases. Eventually, in this thesis we try to combine the two paradigms by utilizing a SentiWordNet ontology base and emoticon phenomenon together with supervised machine learning tools.

4.1.1. WordNet

WordNet is a large lexical database of the English language, useful within many natural language processing applications. Main constituting units are *synsets* (cognitive synonyms), which in turn are comprised from nouns, verbs, adjectives and adverbs. Each synset represents a separate concept is conceptually interlinked with others by conceptual-semantic and lexical relations [Princeton.edu, 2013].

Language Definitions. Vocabulary of a language is defined as $W := (f, s)$, where form f is a string of elements over a finite alphabet, and sense s is an element from a given set of meanings. Forms are either utterances (smallest units of speech, with meaning) consisting of sequence of phonemes (very basic untis of a language, have no meaning alone) or inscriptions composed from characters [Miller, 1995]. Any form that has a sense within chosen language is called a *word* in this language. An alphabetical list of words is called a *dictionary. Synonyms* are the words that share some sense.

A set $C := (N, V, A, Adv)$, where elements are syntactic categories for nouns, verbs, adjectives and adverbs, respectively. Each syntactic category consists of sets of semantic contexts for each f, showing for which contexts it can be used to express a certain s.

The morphology of the language is defined in terms of a set M of relations between word forms. For example, the morphology of English is partitioned into inflectional, derivational, and compound morphological relations. Finally, the lexical semantics of the language is defined in terms of a set S of relations between word senses. The semantic relations into which a word enters deter- mine the definition of that word.

WordNet contains more than 118,000 different word forms and more than 90,000 different word senses, or more than 166,000 (f,s) pairs. Approximately 17% of the words in WordNet are polysemous; approximately 40% have one or more synonyms. The variety of *semantic relations* implemented in WordNet is limited compared to what is available, but is made for being easily understood by people not having advanced knowledge in linguistics. The included semantic relations are following:

- *Synonymy* - is WordNet's basic relation, because WordNet uses sets of synonyms (synsets) to represent word senses. Synonymy (*"syn": same*, *"onyma": name*) is a symmetric relation between word forms.
- *Antonymy* - (opposing-name) is also a symmetric semantic relation between word forms, especially important in organizing the meanings of adjectives and adverbs.
- *Hyponymy* - (sub-name) and its inverse, *hypernymy* (super-name), are transitive relations between synsets. Because there is usually only one hypernym, this semantic relation organizes the meanings of nouns into a hierarchical structure.
- *Meronymy* - (part-name) and its inverse, *holonymy* (whole-name), are complex semantic relations. WordNet distinguishes component parts, substantive parts, and member parts.
- *Troponymy* - (manner-name) is for verbs what hyponymy is for nouns, although the resulting hierarchies are much shallower.
- *Entailment* - relations between verbs.

Cross-POS relations. The four abovementioned syntactic categories have connections, but still are loosely connected. Meaning, very few cross-POS relations exist.

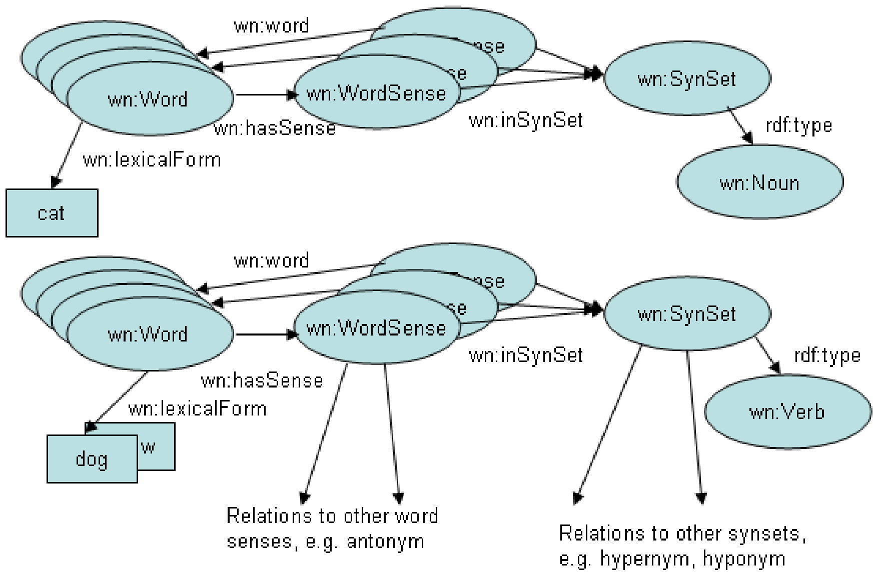


Figure 4.2.: The Structure of Wordnet.

[W3.org, 2013]

4.1.2. SentiWordNet

SentiWordNet is lexical resource specifically developed for supporting sentiment classification and opinion mining applications [Baccianella et al., 2010]. It is publicly available for research purposes. SentiWordNet sources from the WordNet. “SentiWordNet assigns to each synset of WordNet three sentiment scores: positivity, negativity, objectivity.” [Baccianella et al., 2010]. Values of positivity, negativity and objectivity are set in range [0.0, 1,0] and they sum up to 1.0. A SentiWordNet entry is a row with following attributes: part-of-speech tag, WordNet synset ID, posScore (positive), negScore (negative), synset terms, glossare (word meaning explanation with examples). Objectivity scores are derived as follows: *ObjScore = 1 - posScore - negScore.*

4.2. Machine Learning Approaches

There are three major types of Machine Learning algorithms: reinforcement learning, supervised and unsupervied learning. In this section we describe several supervised learning methods useful in the context of this thesis. In general, this type of algorithms involves a process of training. Training means feeding of labelled instances to the classifier. In the process of training, classifier’s parameters are tuned to fit the training data. Next, a testing phase is commonly performed. During a testing phase, a classifier’s performance is assessed. Depending on the kind of classifier, it may overcome adjustments of initial settings to further improve performance. The two phases of training and testing may be repeated in various configurations, again, depending on type of classifier, type and amount of available labelled data. A proper feature choice is crucial to ensure high classification accuracy rate. The set of features has to be representative and sufficient, yet minimal. Large feature vectors can reduce speed performances drastically. In text classification applications, feature vectors commonly consist of: uni- or n-grams; different kinds of metrics such as total number of words or number of negation particles. These features are extracted from raw text with help of test preprocessing tools (described in Chapter 3).

4.2.1. Naïve Bayes Classifier

Bayesian classifiers make class decision to an instance according to being the most likely for its feature vector. *Naïve Bayes Classifier (NBC)* is the simplest example of a Bayesian classifier. Within sentiment analysis it is known to have only marginally worse performance than Neural Networks or Support Vector Machines. But has a tangible advantage in terms of speed (both training and classification). Smoothing techniques have proven to useful for improving classification accuracy. At the core, NBC is based upon the *Bayes Theorem*:

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)} \tag{4.1}$$

where X is an unlabelled sample, C is a class this sample belongs to; $P(C|X)$ is a *posterior probability* of C, given X. In contrast, $P(C)$ is the *prior probability* of class C for known probability distribution of samples. Prior probability application helps compensate the unbalanced data. $P(X|C)$ is an orthogonal variant of posterior probability. It is the probability of sample X to occur, given it possesses features of class C.

Formally, the classification problem becomes a task to determine the probability of having class C, given the data X: $P(C|X)$.

Text Classification. As an input, each data sample is translated to a vector $X = (x_1, x_2, ..., x_n)$ of cardinality n, where x_i depicts a presence of the corresponding attribute (feture) in the sample. For the general case we denote arbitrary number m of classes: $C_1, C_2, ..., C_m$. Naive Bayes classifier will assign class C_i to a sample X only if the class has the highest posterior probability (given X) among all other classes. Formally, a sample X is classified as C_i if and only if:

$$P(C_i|X) > P(C_j|X)\ \forall j, \text{ such that } 1 \leq j \leq m, j \neq i$$

Evidently, we are looking for a *maximum posteriori hypothesis* C_i, while maximizing the $P(C_i|X)$. We compute this probability by using the Bayes Theorem (4.1), with adjusted form:

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)} \tag{4.2}$$

Since the probability of a sample is constant for all classes, the term $P(X)$ can be eliminated. Probabilities of class occurrence $P(C_i)$ can either be considered as equally probable or estimated from available examples:

$$P(C_i) = \tfrac{s_i}{s},$$

where s_i is the number of training examples for class C_i, and s is the total number of training examples.

Computing all posterior probabilites $P(X|C_i)$ is unfeasible. Due to vast(?) flexibility of language morphology, the number of attributes can be tremendous. Attributing to that is the high number of new forms added continuously. New words, abbreviations and ubiquitous slang usage do increase the complexity. The solution is to accept the *class conditional independence* condition. Meaning we are dramatically simplifying the computation under assumption that words (or n-grams) may occur independently of each other. Therefore, the $P(X|C_i)$ becomes a product of posterior probabilities of separate attributes given sample class:

$$P(X|C_i) = \prod_{k=1}^{n} P(x_k|C_i) \tag{4.3}$$

Where we compute $P(x_k|C_i)$ with *maximum likelihood estimation*:

$$P(x_k|C_i) = \frac{s_{ik}}{s_i} \tag{4.4}$$

with s_{ik} number of training samples if class C_i having attribute x_k, and s_i is the total number of training samples of class C_i. Evidently, our classification is simply a product of many small numbers within $[0, 1]$ interval. This form may lead to floating point errors during actual computations. *Log likelihood* trick is introduced to minimize such errors. The final classification function is:

$$\text{classify}(X) = \arg\max_{C_i}(\log P(C_i) + \sum_{k=1}^{n} \log \frac{s_{ik}}{s_i}) \tag{4.5}$$

Smoothing Technique. When we work with natural languages, many unknown features occur. This the result of a large active dictionary and also the sizes of training datasets are often limited. With current setup, when we spot an unknown word, the maximum likelihood estimation for 4.4 would prevent proper computation with 0 as a divisor. Eventually, the whole outcome of 4.3 could become 0. The so called *smoothing techniques* are used to resolve the issue. Further we describe the *Laplace* (or *laplacian*) *smoothing.*

Laplacian smoothing is one of the simplest approaches known. It works by simple addition of an extra 1 for word frequencies. The method works well in practice and is easily integrated in existing methods. Applied to the maxmum likelihood estimation (Equation 4.4), the classification function is shown in Equation 4.6 (note an extra 1 is added in the denominator as well). Another examples of smoothing techniques include *Jelinek-Mercer method* and *Absolute discounting.* All these techniques vary in implementation complexity and feasible applications (depending on the size of available training set, and size of each training exmaple).

$$\text{classify}(X) = \arg\max_{C_i}(\log P(C_i) + \sum_{k=1}^{n} \log \frac{s_{ik} + 1}{s_i + 1}) \tag{4.6}$$

4.2.2. Support Vector Machines

Support Vector Machines (SVM) is another example supervised machine learning algorithms. SVM is a non-probabilistic binary classifier, that could be used for classification and regression analysis. The standard SVM is a linear classifier, although it is often improved by using of a so called *kernel trick*. This allows to tackle non-linearly separable datasets as well. The underlying principle of an SVM is pretty basic within the field of machine learning. We say that we are searching for some hypothesis h within a hypothesis space H, such that it guarantees the lowest error probability $\text{Err}(h)$ for some sample S. For generic case, the input data is denoted as follows: $(\overrightarrow{x_1}, y_1), ..., (\overrightarrow{x_n}, y_n)$ are samples, where $\overrightarrow{x_i} \in \mathbb{R}^n$ is a feature vector (of cardinality n) and $y_i \in \{-1, +1\}$ is a binary label. The *true error* $\text{Err}(h)$ is defined to be within a certain bound of the *training error* (error rate on the training set after training on it) with probability $1 - \eta$. The bound itself depends on the complexity of chosen hypothesis, η value (represents an estimate for normal data points, i.e. controls number of samples contributing to the minimization objective) and type of data used (particularly concerned with the number of features n). Equation 4.7 demonstrates the true error formulation: here d denotes the VC-diension and, basically, describes the complexity of a hypothesis, and therefore it's ability to separate particular type of data (applicability to current instance space). By definition, the probability of the bound holding is $1 - \eta$. Thus, the equation shows a compromise to be made between training error rate and hypothesis complexity.

$$\text{Err}(h) \leq \text{Err}_{train}(h) + O(\frac{d\,ln(n/d) - ln(\eta)}{n}) \tag{4.7}$$

In general case SVM produces linear threshold functions. These functions represent separating hyperplanes within feature space. Figure 4.3 shows us graphical representation of the learned hyperplane h, given some set of samples. After optimal hyperplane is found, classification is done according to following function:

$$h(\vec{x}) = sign(\vec{w} \cdot \vec{x} + b) = \begin{cases} +1, & \text{if } \vec{w} \cdot \vec{x} + b > 0 \\ -1, & \text{otherwise} \end{cases} \tag{4.8}$$

Here $\vec{w}$ is a normal vector that represents the hyperplane and b is an offset from the origin of the normal vector. Additional to the standard method, δ is a margin magnitude. All the samples lying closest to the hyperplane (minimum distance δ) are called *support vectors.* A standard SVM that allows no samples within a margin is called *linear hard-margin SVM.* A hyperplane for such variation is found by solving an optimization problem given in Equation 4.9. We define $\delta = \frac{1}{||\vec{w}||}$, where $||\vec{w}||$ is the L_2-norm of the vector $\vec{w}$. Therefore, while minimizing the $\vec{w} \cdot \vec{w}$ part, we maximize the δ value [Joachims, 2002].

$$minimize: \ V(\vec{w}, b) = \frac{1}{2}\vec{w} \cdot \vec{w} \qquad subject\,to: \ \forall_{i=1}^{n} : y_i(\vec{w} \cdot x_i \vec{+} b) \geq 1 \tag{4.9}$$

Kernel Trick. The above given setup is feasible for linearly separable data only. To overcome this limitation it is common to use a so called *kernel trick*. In order to architect a non-linear SVM, feature vectors $\vec{x_i}$ have to mapped to higher dimensions (where the data becomes linearly separable). The vectors are mapped to a feature space X' by a non-linear function $\Phi(\vec{x_i})$. The mapping algorithm is of high complexity (if computed explicitly), and therefore, infeasible for large scale applications. An efficient solution was discovered in [Boser et al., 1992], it consists of two parts. The first part of the solution states that it is sufficient to compute a dot product $\Phi(\vec{x_i}) \cdot \Phi(\vec{x_j})$. The second part states that for some cases (if they satisfy Mercer's Theorem) kernel functions $k(\vec{x_1}, \vec{x_2}) = \Phi(\vec{x_1}) \cdot \Phi(\vec{x_2})$ can be used to avoid explicit computations of dot products. To conclude, these functions drastically improve performance. Among the most frequently used kernel functions are: sigmoid function, radial basis function and polynomial kernels - see Equations 4.10, 4.11, 4.12. The rest of SVM steps are the same, but specific methods of solving this problem are beyond the scope of this work.

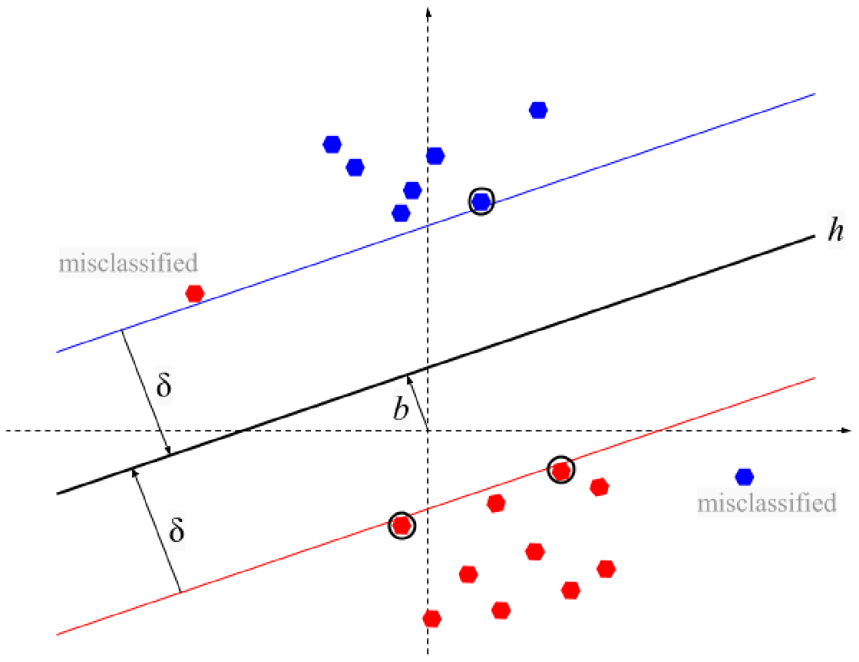


Figure 4.3.: The SVM learns a hyperplane, that divides samples into two classes. [research.microsoft.com, 2013]

$$k_{sigm}(\vec{x_1},\vec{x_2}) = tanh(s(\vec{x_1}\cdot\vec{x_2}) + c) \tag{4.10}$$

$$k_{RBF}(\vec{x_1},\vec{x_2}) = exp(-\gamma(\vec{x_1 - x_2})^2) \tag{4.11}$$

$$k_{poly}(\vec{x_1},\vec{x_2}) = (\vec{x_1}\cdot\vec{x_2} + 1)^d \tag{4.12}$$

Naive Bayes based classifiers are more suitable for mobile devices. In terms of performance for sentiment analysis applications, SVM only marginally outperforms Naive Bayes classifiers by 1-2% [Huang et al., 2003].

4.2.3. Evaluation

Overfitting and Underfitting. When an underlying classificator model becomes too complex, the situation is called overfitting. In this case a classifier is generally performing very well on the training set, i.e. the training error rate is low. But, at the same time, the classifier performs poorly on unseen samples, i.e. the true error rate is relatively high. Depending on a machine learning approach, different methods are used to suppress overfitting. For Naive Bayes or Support Vector Machine a general suggestion is to limit the size of a dictionary of accepted words, e.g., by setting a frequency threshold or using a predefined vocabulary. For Decision Trees overfitting can be avoided by ignoring outliers (post-training tree pruning). Underfitting is not exactly the opposite of overfitting. In the sense that classifier's underlying model is rather simplistic, compared to the overfitting case. But in turn, both training error and true error have high values.

4.3. Emoticon Classification

Emoticons are a useful feature of modern languages. They are both popular and indicative, thereofore helpful within sentiment analysis aplications. Emoticons became ubiquitous due to two factors: phones have small keyboards (people were trying to minimize the number of keys pressed for sending a message) and messages have limited length (as in SMS, due to technical restrictions; or Twitter, due to platform restrictions). Emoticons are mainly a 2-3 symbol combination, allowing to express an emotion in highly compact way. This characteristic makes emoticons easy to identify in text. Most emoticons are used to express a particular emotion, i.e. their meaning is unambiguous. This attributes them to being highly indicative.

Beyond classification of messages that contain emoticons with pre-trained classifier models, such messages can be used to train a classifier directly. Our system allows such mode. A pre-trained Naive Bayes classifier can be further improved on the mobile device, with messages gathered from it. This essentially helps save effort in gathering large training sets labelled manually.

5
Android Mobile Platform

Android Operating System is the best choice as a platform for the application we develop. The installation base is tremendous across the world - more than 1 billion devices activated so far with more than 1.5 million new activations per day rate [Techcrunch.com, 2013] - guarantees an unbeatable reach to users(?). The system is being developed by Google Inc. and is integrated with it's various services like web-search and youtube. Regardless of the fact, Android continues to be an open-source software with major releases provided every six to nine months. The code is partially licensed under Apache and BSD licenses. Although, there is no guarantee for this to continue, the developer community has the ability to proceed with the development process at any time.

Android is much different from it's biggest competitor - iOS. The platform gives developers opportunity to completely change the look and feel of the system. Yet empowers the experience by including the best-in-class services from Google - web-search, maps, email and video.

5.1. Ecosystem

Inclusion of new features as well as bug fixes are only limited by developer's imagination and interest. As of current state, Google provides the source code for all major releases. So anyone can "fork" the repository and alter the code according to his needs. The source code is located in the central repository [Googlesource.com, 2013]. If a developer spots a bug in the code, he is able to influence the fast fixing of it for other users by submitting a bug report to Google [Android.com, 2013]. Moreover, the open-minded community in the open-source ecosystem is proactive. Open-source centric approach results in a wide variety of available to developers tools.

5.2. Architecture

Android OS has a layered structure, where components of each layer interact within their layer or the one below. A schematic representation of available layers can be seen on Figure 5.1. As an introduction to the existing terminology within Android ecosystem, we describe the most crucial ones.

Application Layer. Application layer is the one user interacts with directly, while using the device. Whatever a normal user does, goes through this layer. Besides what user installs, the layer includes built-in applications: Alarm Clock, Browser, Calculator, Camera, Contacts, Email, Gallery, Messaging, Music, Phone, Settings. It is common for various phone manufacturers to include their own applications.

Application Framework Layer. Application Framework layer unites a number of services. These services provide access to the underlying hardware utilities - cam be seen as an API. Applications from the Application layer access provided services and control different functions of the mobile device, e.g., wi-fi connectifity, bluetooth connectivity, file storage management, telephony and location. Characterizing parts of the layer are: *Activity Manager* - provides interaction for all activitise running in the system; *Resource Manager* - provides access to embedded resources of applications (defined in XML configuration files); *Package Manager* - gives access to package data over all installed applications; *Content Providers* - are

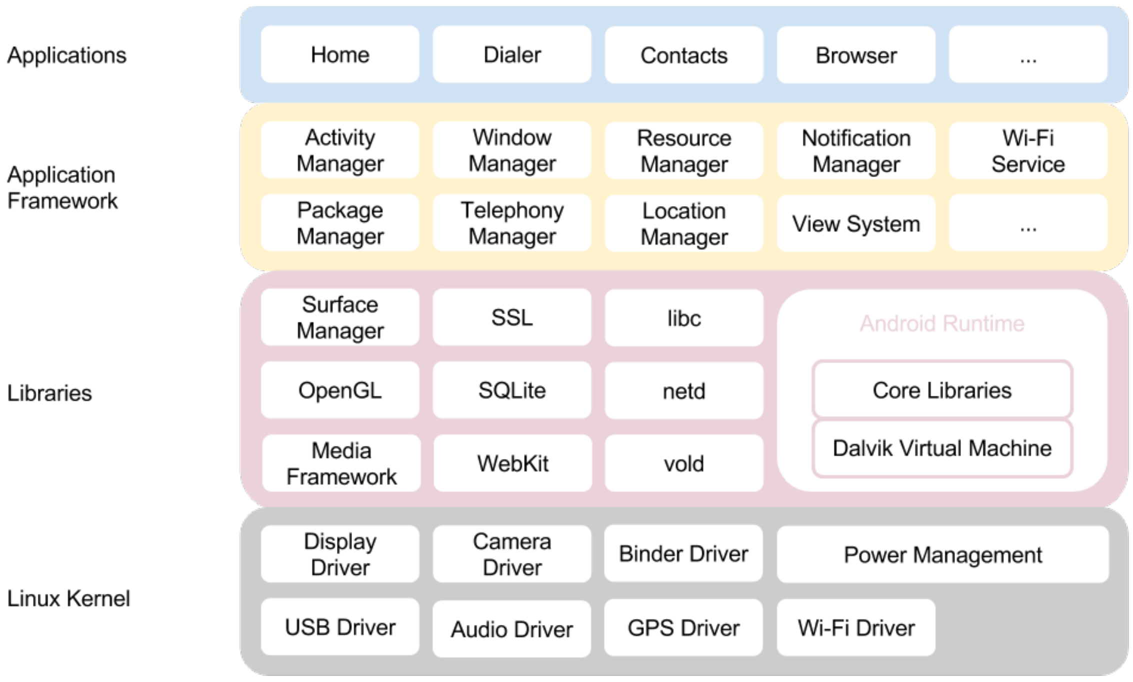


Figure 5.1.: Layer Architecture of Android OS.

interfaces that connect data from one process with code executed in another process [developer.android.com, 2013f]; *Notification Manager* - provides methods to notify users from within background activities (available methods are - persistent icon in the status bar, through device's LEDs, backlight flashing, playing sounds, vibrating) [developer.android.com, 2013d].

Libraries Layer. The Libraries layer includes standard stack of libraries, required to realize all expected use-cases of the system. These libraries allow efficient storage, processing and transaction of application data. These libraries are native, and therefore, written in C/C++ for best performance and small memory footprint. The libraries are comprised of some of the industry's well-known products: *SQLite* - an embedded database management system with effortless configuration [SQLite, 2013]; *WebKit* - is an open-source web browser engine, used in Safari and Chrome browsers [WebKit, 2013]; *OpenGL* - a multi-platform graphics library for computer graphics rendering [OpenGL, 2013].

Android Runtime. A sub-layer within the Libraries/Native layer responsible for Android applications execution. As of Android version 4.4 ("KitKat"), a whole new, yet experimental version of a virtual machine is introduced - ART virtual machine. Active developers have a way to switch either Dalvik (libdvm.so) for production version testing or ART (libart.so) for early adoption of the new runtime. Additionally to a virtual machine, modified Java SE libraries are bundled in the layer. These libraries provide a common Java API for data structures, file and network access, graphics and other utilities. Structurally, the runtime consists of three parts: daemons, service manager, zygote. *Daemons* are special programs that execute certain methods numerous times, but with relatively large intervals. They include clocks and timers. A *service manager* is a background application that operates (launches and terminates) the services of the Application Framework layer (Telephony service, Connectivity service ...). In Android, every application is run in a separate virtual machine, so for better system performance the VM has to be able to start fast and use as little memory as possible. This is the task for Zygote. *Zygote* is a special process within a VM that guarantees an efficient way to do this. It optimizes memory usage by extractingcommon heap structures and libraries and sharing them across all processes. The initial Zygote process is created during system boot. Upon request, Zygote process spawns new VM instances with minimized startup times. By default, Zygote assumes all shared data is immutable. In case an event of change occurs within some VM instance, the memory from the shared Zygote process is copied to the corresponding VM process. This strategy results in high(?) memory saving without surrendering security.

Dalvik Virtual Machine (DVM). DVM is the core part of Android Runtime layer. It is responsible for application execution. DVM is Android's special version of a virtual machine that brings the power of Java language platform independence. In contrast to *Java Virtual Machine* (JVM), Dalvik VM executes files in the Dalvik Executable (.dex, DEX) format. The special type of bytecode is adjusted for best fit across various mobile devices - it is more compact and efficient. The Dalvik VM is not compatible with the standard JVM. The JVM was taken as a reference point (with universal bytecode and broad hardware compatibility), but with scarce computational resources in mind. As a result, Dalvik is capable of running with little RAM, slow CPU and limited speed buses (no swap is needed). Also limited power supply (battery) and sandboxed application runtime are important features of the VM that influence both user experience and security. A substantial performance increase is achieved by using of a register-based VM architecture, instead of traditional stack-based. Register-based code is around 25% larger than stack-based, but is overriden by the 50% space savings with .dex file structure. DEX files are generated from .class (Java bytecode) files [Ehringer, 2010]. In contrast to .class files, DEX files include multiple classes within one file. DEX files use shared, type-specific constant pools to preserve valuable memory. Four separate pools are devoted for constants: strings, type/class, field and method constant pools. They use more space if only one class is considered, but since DEX files include several classes - space savings are substantial. All the constants in the pool did not necessarily have the “final” modifier in raw code, but also large portion of such constants is extracted after static analysis by the javac java compiler.

Kernel Layer. The *Kernel* layer is the most low-level component of an operating system. It provides required drivers to control underlying hardware. It possesses the main hardware resources managing logic - keeps track of utilized resources, controls power usage, delivers execution process threading and low-level memory management functionality. All four major hardware subsystems are controlled by the kernel - central processing unit (CPU) + graphical processing unit (GPU), permanent memory (hard disk drive or flash-based drive), random access memory (RAM), peripheral input-output (IO) interfaces (such as USB or integrated Wi-Fi/Bluetooth wireless adapters).

Android’s kernel is based on a robust and open-source Linux kernel. Until Android version 4.0, Linux kernel versions 2.6.x were used, afterwards, the version 3.4.10 is used. Even though Linux kernel is appears to be highly secure and robust compared to other architectures, enthusiasts manage to find security loopholes. This is often done to circumvent manufacturer’s limitations, i.e., for full control of the system and ability to customize device’s software to their needs.

6 SentiMenthal Architecture

Architecture is decisive to success. This means we have to be “prepared” before actual implementation starts. Preparation consists of establishing the requirements and making architecture decision phases. The two phases could run in parallel, although some requirement elicitation has to be done in the first place. This chapter starts with software requirements overview and continues with architectural decisions description. In the end we describe most of the third-party tools used during the development and give comments regarding particular choices made.

Sentiment analysis of social media messages is a complicated task. Classification of short messages is a particularly difficult task, taking into account the fact, that many messages contain numerous abbreviations, misspellings, grammar mistakes, and, of course, ever evolving “memes”. To adjust the performance and classification accuracy of our application various text pre-processing configurations had to be tested. To conclude, while developing this software we concentrate on a primary objective to produce an application with the following characteristics:

- It is lightweight and runs well on mobile devices
- Allows users to collect textual information from several messaging applications
- Analyzes and presents results in a comprehensible way

A general process pipeline of SentiMenthal was defined, prior to implementation (see Figure ?). At the first stage comes text preprocessing. Preprocessing routines have to be executed in the same order and configuration for both training and classification phases. An important decision is made upon the fact of presence of emiticons in the message. We take responsibility to assume, that emoticons if used, are mostly without sarcasm or irony, leading us unamguosly classify the message. In this way, if an emoticon (or more than on) is present in a message, the algorithm proceeds to classify message through a simplistic emoticon classifier. Such classifier is manually preconfigured, needs no training, and is potentially modifyable on-the-fly.

6.1. Software Requirements

It is a good practice to design functional and non-functional requirements before the actual implementation of such system. However, for large projects this scenario is quite optimistic. It is impossible to predict and design all the requirements beforehand - drastic changes are often needed during the development process due to practical evidence of infeasibility of implementing of some decisions.

Functional requirements define functions of software developed. They describe what kinds of input data is received, what manipulations are done over it and, eventually, what kinds of output data is produced.

Non-functional requirements define criteria and metrics for system operation evaluation (e.g., performance evaluation or input data classification accuracy). They describe the "quality of the outcome" rather than means of getting the result.

Although, most of functional requirements are implemented through the GUI subsystem, it is essentially dependent on other subsystems. We define a list of functional requirements:

- Gather messages from one of the supported applications (Skype, Facebook Messenger, WhatsApp);
- Store messages in embedded database for later use;
- View gathered messages. Filtered by application/sender/timeframe;
- Select one of available classifiers (in application preferences);
- Classify a selected messages set with preselected classifier;
- Train selected classifier with manually classified messages. Store updated classifier model;
- View unclassified messages;
- Classify unclassified messages manually;
- Automatically or manually delete non-English or unclassified messages;
- Store classification results for separate classifiers;
- View message sentiment fluctuation charts with selected parameters - particular timeframe, source application, sender/senders, classifier;
- View classifier performance comparison charts;
- For each of the applications - show number of gathered messages, number positively and negatively classified messages;

- View estimated vocabulary size fluctuation chart;
- View database word frequency chart;
- View parameters of classifier's model.

For the list of non-functional requirements we have picked the most crucial ones. Selected categories may seem vague and blurry, but this is the nature of non-functional requirements. The list is the following, sorted in importance descending order:

- Swift extraction, processing and analysis of messages
- Protection of ensitive data
- Ability to run the application on multiple hardware platforms
- User-friendly graphical interface
- Message classification accuracy

6.1.1. Security

SentiMenthal software works with sensitive information and should comply with strict security requirements. Very few people would trust a program to track their private messages and transmit anywhere for analysis. It is the primary reason we irrevocably agreed to perform any kinds of data analysis on the source device only. The decision is crucial for application's architecture. From one point of view, there is no need for secure transmissions of "toxic" data; from another - data storage and processing has to be done on the mobile device with reduced performance.

Android OS alows us an easy and robust way to manage data through the Android Application Sandbox mechanism [developer.android.com, 2013h]. All application data and code execution are isolated from other programs. Additionally, if one would want to be able to store data on external storage devices (e.g. flash memory cards), built-in cryptography tools are available. Such functionality is not envisaged within our application, to further assure our users of safety. To reiterate, as of current state, none of the data is sent by our application from the device. If this situation is changed, naturally, a proper warning will be given.

6.2. System Architecture

Separation of concerns is a powerful method for productive software development. Our software product is composed of four subsystems: graphical user interface (GUI), database (DB), Text Extraction logic and Text Analysis module. A schematic presentation is shown on Figure 6.1. Naturally, GUI controls most of the functionality and is responsible for data presentation. GUI subsystem has bidirectional

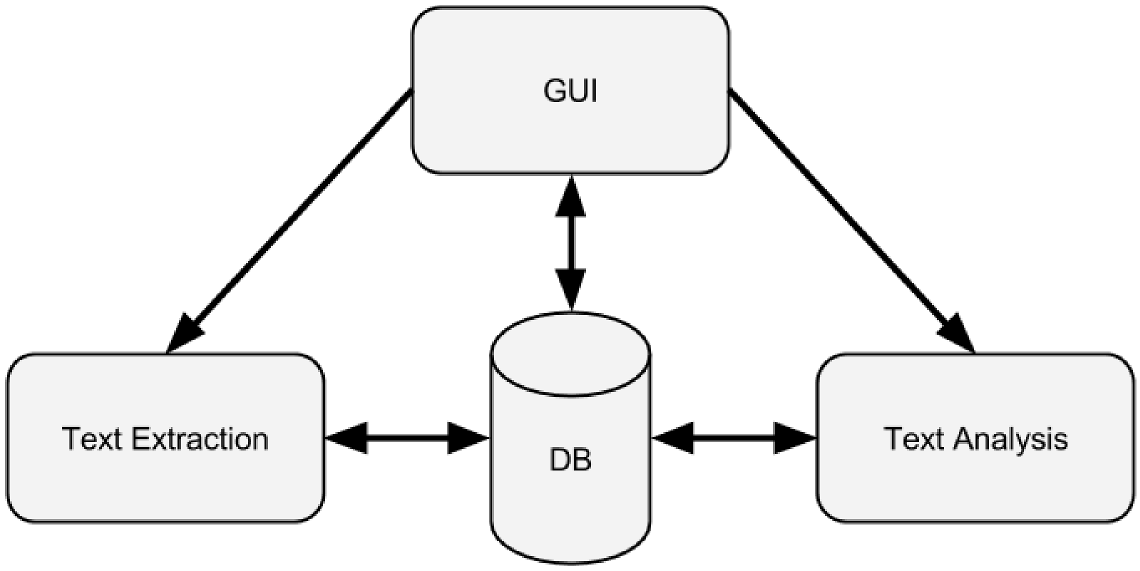


Figure 6.1.: Separation of concerns in the SentiMenthal application.

relations with DB, since it performs both “reads” and “writes” with the database. The relations are directed from GUI to Text Extraction and Text Analysis, since it invokes methods of these subsystems; the opposite direction is absent, since GUI is not influenced directly by the data produced in TE and TA. The latter two subsystems have bidirectional relations with the database, as the data flows both ways there.

6.2.1. GUI

The GUI subsystem is responsible for inner application navigation and presentation of data. Navigation has to be intuitive and seamless. Data presentation has to be concise and self-descriptive. Both raw and processed data can be viewed and edited. A proper attention has to be devoted to the matter of separation of different types of data presented. To increase the percepted value of our application for users, presentation subsystem has to be adjustable and modifyable. In our application we attempted to find a balance out number of options and user-friendliness - application has to stay intuitive, yet useful.

Apart from analysis results presentation, two important parts are: classifier settings and manual classification. For the purpose of empirical comparison, we have added a feature to switch between several modifications of classifiers. The selection is made in a special settings view. Classifier choice and other settings are saved, to give opportunity to continue analysis with those after application realunch. Another aspect of our GUI design is a set of windows for manual classification and editing of gathered messages. The feature is important for refining messages corpus and improving classification models.

Fast prototyping and implementation are crucial for prompt development of such systems. This is reason we used standard and unmodified ADT components for the GUI. Therefore, we were able to start implementing and testing fucntional features soon. A choice of well-supported and an easy to use chartin library is also essential. All in all a good user interface results in improved application utility and overall user satisfaction.

An integral part of an appealing user interface is informativity. We have encountered such a case during the message crawling process. Since the process relies on the AccessibilityService "trick", a user may be confused if he sees numerous actions performed "by the device" in one of the applications. Also, the remaining time and number of gathered messages so far may be useful to display. To solve this task, we implemented a semi-transparent overlay button with useful information. This is realized with a method recently used to create Facebook Chat Heads. ...blablabla, give a code examplea and some description...

6.2.2. DB

DB subsystem is responsible for raw and processed data storage. Stored data has to be compact and easily accessible and modifiable.

In our case, to spare development type, a choice to use only two database tables has been made. One for raw messages and one for processed and classified ones. This approach is far from optimal, but is required for faster development. In ideal case, we would require two tables per supported application, plus additional tables for intermediate results and other needs.

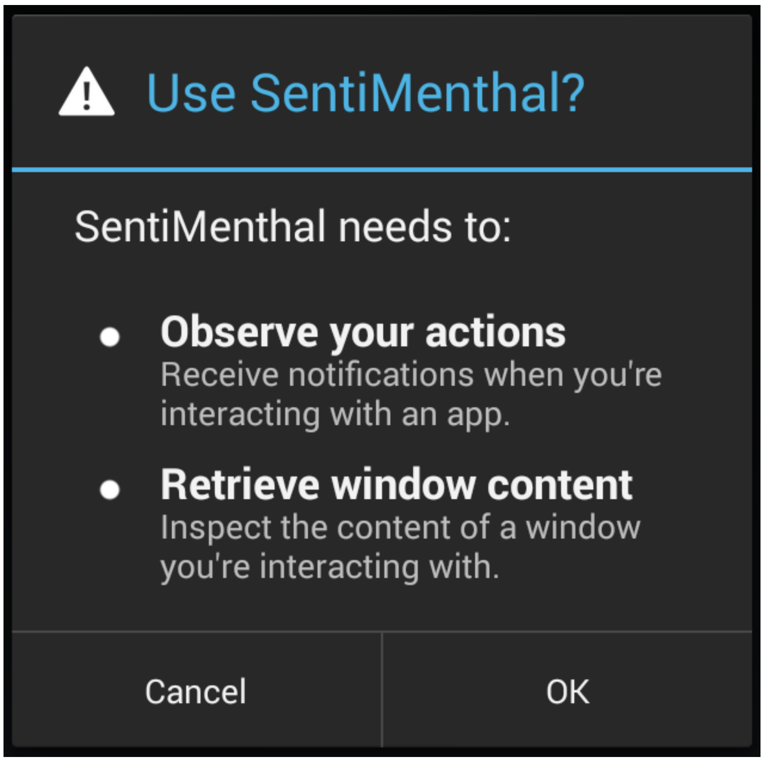
6.2.3. Extraction

Text Extraction subsystem is responsible for data gathering and sending to the storage. It has to be robust, fast and able to gather data from multiple sources. The subsystem represents a package "extraction" consisting of an interface class and three unified classes - for each supported application. Interface class contains abstract methods. It guides the three implementing classes for correct and unified structure. A benefit of such approach is expandability - support for more message source applications could be added without great changes in architecture of SentiMenthal.

```
<service
    android:name="org.menthal.app.MyAccessibilityService"
    android:enabled="true"
    android:exported="false"
    android:permission="android.permission.
        BIND_ACCESSIBILITY_SERVICE" >
    <intent-filter>
        <action android:name="android.accessibilityservice.
            AccessibilityService" />
    </intent-filter>

    <meta-data
        android:name="android.accessibilityservice"
        android:resource="@xml/accessibilityservice" />
</service>
```

Listing 6.1: Accessibility Service Configuration

The subsystem uses a rather unusual way of interaction with the mobile device. 6.1 depicts part of application's XML configuration file, where the "MyAccessibilityService" is registered. It can be seen, that it requires a "BIND_ACCESSIBILITY_SERVICE" permission. This particular permission allows an application to access display state of any running application on the device. Therefore, it can only be granted by the user, manually. This has to be done before SentiMenthal is run. The walkthrough is the following: navigate to system settings -> scroll down to the "SYSTEM" section -> Accessibility -> "SERVICES" section -> SentiMenthal -> toggle "ON" -> a warning will appear (see ...), press "OK".


Use SentiMenthal?

SentiMenthal needs to:

- **Observe your actions**
  Receive notifications when you're interacting with an app.
- **Retrieve window content**
  Inspect the content of a window you're interacting with.

Figure 6.2.: A warning for accessiblity service permissions.

6.2.4. Analysis

Text Analysis subsystem is the crucial component of the application. It is responsible for text pre-processing and classification. The subsystem is dependent upon hardware performance, therefore, it has to be optimized in order to account for limited resources. Several classifier implementations are be offered as an option. Two versions of Naive Bayes classifiers are implemented. One version is able to infer a dicionary of arbitrary size - any words from input messages (pre-processed) are added to the classifier model. The second version uses a predefined dictionary, using the SentiWordNet as a reference point. Another version available for option is a simplistic classifier with sentiment values taken from SentiWordNet source.

6.3. Tools

Choice for developer's tools is crucial for success of the whole project. Badly designed, scarce in functionality or slow tools can demotivate the whole development team. As a result, negligence at tools selection may draw development cost and effort infeasible, and, eventually, a project failure.

At its core, development process starts by not the programming language selection, but by selection of an ecosystem. Although, ecosystem, in most cases, dictates the programming language to be used. In our case, the Android ecosystem is chosen as the only one satisfying all functional requirements for the project. Finally, development tools can be split into these categories: prototyping tools (for GUI, requirements, architecture, API planning), development tools (IDEs, SDKs, third-party libraries, build tools, deployment/integration tools, emulators), product quality assurance tools, reference libraries and data sources, team development organization tools. Incidentally, not all categories mentioned were used during development of SentiMenthal. The software project is of smaller size and was implemented by only one person. For instance, prototyping, code testing and team-directed tools were used minimally if not at all. Thereofore, they are not worth mentioning here further.

6.3.1. Software Development Kit (SDK)

As mentioned before, the choice is obvious and is in favor of Android SDK. It is an essential part of the Android ecosystem, one could not imagine Android to be as it is today without it's SDK. An SDK is a coherent collection of software libraries and development tools. The main advantage to have an SDK elsewhere is simplicity and orderliness of a development process. Unification and high level of integration across development tools allows well established development pipelines and methodologies to be created. In turn, development becomes predictable and streamlined. Historically, since large corporations tend to support their own ecosystems, they compete for the number of developers under their platforms. And potentially many developers are attracted by the quality of tools (especially free or even open source) available.

Android SDK is a bundle of common libraries with extensively documented APIs (Application Programming Interface) and developer tools. The tools provide means for building, testing and debugging Android applications. The SDK consists of modular packages, which are distributed through the Android SDK Manager [developer.android.com, 2013e]. The manager envisions multiple supported versions of the API and keeps required components up-to-date. Vast majority of applications require a fracture of available packages. These packages are: SDK Tools (tools for debugging and testing and other utilities that are required to develop an app), SDK Platform-tools (platform-dependent tools for developing and debugging your application, they support the latest features of the Android platform and are typically updated only when a new platform becomes available), SDK Platform (there is one SDK Platform (android.jar) available for each version of Android; a platform version is specified for every aaplication), System Images (a set of Android OS images for various hardware architectures; the package is required for the emulator for robust test/debug), Android Support (includes popular components - Fragment and ViewPaper with older APIs support) [developer.android.com, 2013g]. Among most common SDK Tools are: AVD Manager - Android Virtual Device Manager (GUI tool to create/delete virtual devices and configure hardware specifications along with the highest supported API version); adb - Android Debug Bridge (provides means of communication with virtual of physical, connected device) [developer.android.com, 2013a]; Hierarchy Viewer - a debug tool, helps with user interface optimizations (screenshots can be made from a running device along with auxiliary tree data structure with all of current presentation state) [developer.android.com, 2013c].

6.3.2. Emulators

Emulators are software environments that allow in-place (developer's computer) testing/debugging. Various hardware configuration emulation allows to predict and eliminate future errors, when the application is released. Although, Android SDK comes with a default mobile device emulator, there are alternative implementations [developer.android.com, 2013b]. In practice, the bundled emulator is has poor performance, despite complete API support. Several alternative emulators are available for developers. Notable examples are - BlueStacks [BlueStacks, 2013] and GenyMotion Emulator [GenyMotion, 2013]. BlueStacks emulator provides high performance with hardware (GPU+CPU) acceleration, and an exhaustive support for multiple OS configurations. The software is able to virtualize and run Android OS on Windows (x86 and ARM), Mac OS X (x86) and Chrome OS (x86). The powerful architecture behind allows to run Windows OS under Android, for faster startup times and longer battery life. The product in has over 10 million downloads in total and is distributed free of charge. The GenyMotion Emulator all latest Android releases and emulates under Windows. The software is distributed with a "freemium" strategy - both free and paid versions with more features and better support are available.

6.3.3. Integrated Development Environment (IDE)

IDEs are software products made for programmers to speed up, simplify the process of development. This result achieved by placing all of the used tools in one place. An important quality is that all the tools are connected trhough a single API (the one defined by particular IDE) and do work seamlessly together. Most popular modern IDEs support multiple programming languages. This is realized through syntax highlighting, automated code completion and availability of specialized tools (compilers, plugins for common frameworks). An IDE ecosystem is often augmented by popular multi-platform tools, such as version control, modeling, team coordination tools, testing suits, report generation and visualization tools etc. The four of the most aknowledged IDEs come with well-established communities around them. Communities provide great support for open-source plugins along with swift development of newer ones. The four aforementioned IDEs are: Eclipse, IntelliJ IDEA and NetBeans. We will describe in detail the first two, since Eclipse was used for SentiMenthal development and IntelliJ IDEA is in contrast a commercial product with outstanding features.

Eclipse is an open-source, cross-platform integrated development environment, developed by Eclipse Foundation. The IDE is written mostly in Java language, therefore, it is the supported by default. Although this is by for not limited to it, as special plugins are available for other languages support. To address the IDE within the Android development context, until May 2013 the officially supported Android Development Toolkit (ADT) was based on Eclipse platform. This played favor of Eclipse as a default and best choice for Android development. ADT is a powerful plugin, bundled with predefined wizards for mobile application projects and components creation; and multiple GUI wrappers for essential Android tools. All in all, Eclipse IDE has found great simpathy of developer community for its opennes, great extension features and large library of integratable tools.

IntelliJ IDEA is in contrast a commercially oriented software product. It supports a wide variety of programming languages and technologies connected to them. The software is praised for its advanced code editing features, such as fast and precise code completion; pleasant and user-intuitive user interface and sophisticated graphical interface building tools. Professional, yet paid customer support also comes advantageous for enterprises. Worth mentioning, IDEA also has a thought-through API for new plugins to be created. As of current this IDE comes with built-in Android development support, morevover, since May 2012, Google has announced to support a specialized and free version of it (called Android Studio) as the primary choice for Android developers.

6.3.4. Sentiment Reference Dictionary

POS	ID	PosScore	NegScore	SynsetTerms	Gloss
a	00001740	0.125	0	able#1	(usually followed by 'to') having the necessary means or skill or know-how or authority to do something; "able to swim"; "she was able to program her computer"; "we were at last able to buy a car"; "able to get a grant for the project"
a	00002098	0	0.75	unable#1	(usually followed by 'to') not having the necessary means or skill or know-how; "unable to get to town without a car"; "unable to obtain funds"
a	00002312	0	0	dorsal#2 abaxial#1	facing away from the axis of an organ or organism; "the abaxial surface of a leaf is the underside or side facing away from the stem"
a	00002527	0	0	ventral#2 adaxial#1	nearest to or facing toward the axis of an organ or organism; "the upper side of a leaf is known as the adaxial surface"

Table 6.1.: Sample data lines from a raw SentiWordNet resource.

Language detection task can be accomplished in two possible ways. One way is to use an realization, where dictionary for classifiers is arbitrary and constructed from training data. Another way is to utilize an existing dictionary, for instance, from SentiWordNet. It can later be used either to constrain a classifier's dictionary (supervised learning case) or as a basis dictionary with predetermined sentiment values in a simplistic classifier. For illustrative purposes, we compare both versions with SentiWordNet and get a total of three core variants of text classification. SentiWordNet is distributed in a form a table formatted text file. Initially, the library contains entries with more data than required (see 6.1). For our purposes, we extract unique terms with corresponding positive and negative scores. There are cases, where we omit the terms with undefined scores (0). Also, the library contains terms consisting from several words connected with an underline ("_"). Such terms require additional processing.

6.3.5. Other Tools

To present results of text analysis, SentiMenthal uses a specialized charting library - AChartEngine. It is a native charting library for Android, distributed free of charge under Apache 2.0 license [AChartEngine, 2013]. The library is praised for it's wide variety of available charting types, good documentation and high performance. It is essentially faster due to being native to the OS (does not use in-browser rendering with JavaScript configurations as some of the alternatives), yet is thought to be harder to configure for pleasing aesthetics. The library is highly customizable and supports a list of chart types: line chart, area chart, scatter chart, time chart, bar chart, pie chart, bubble chart, doughnut chart, range (high-low) bar chart, dial chart / gauge, combined (any combination of line, cubic line, scatter, bar, range bar, bubble) chart, cubic line chart. “All the above supported chart types can contain multiple series, can be displayed with the X axis horizontally (default) or vertically and support many other custom features. The charts can be built as a view that can be added to a view group or as an intent, such as it can be used to start an activity. The model and the graphing code is well optimized such as it can handle and display huge number of values.” [AChartEngine Source, 2013].

A Bloom Filter structure is used to save valuable system resources. Therefore, the qualilty of implementation of this instrument is crucial. To achieve good results, a sophisticated manipulations with device’s memory have to be executed at runtime. A suitable implementation was found in the Apache Cassandra - a distributed database management system. It is an open-source software, distributed under Apache License 2.0. The four useful classes are located in the “org.apache.cassandra.utils”, and they are: “BloomFilter”, “IFilter”, “Murmur2BloomFilter” and “MurmurHash”. The last two classes are responsible for generating a sufficient number of special hash functions. To generate a Bloom Filter bitstring object, an instance of Murmur2BloomFilter has to be created with provided number of hashes and an of BitSet of chosen size. Full routine code of bitstring population is demonstrated in 6.2.

```java
public BloomFilter generateBitString(InputStream is, BitSet bitset
    ) {

  BloomFilter bf = new Murmur2BloomFilter(hashes, bitset);

  BufferedReader br = new BufferedReader(new InputStreamReader(is)
      );
  try {
    String line;
    while ((line = br.readLine()) != null) {
      bf.add(line);
    }

    br.close();

  } catch (FileNotFoundException e) {
    e.printStackTrace();
  } catch (IOException e) {
    e.printStackTrace();
  }

  return bf;
}
```

Listing 6.2: Method for Bloom Filter configuration and population

Word stemming is another integral part of the natural language analysis process. Various libraries are available with with stemming functionality, but our goal was to find a well-developed one, with open source code and a viable license type. Programming language is also critical, of course. The most used type of stemming approach is a Porter stemmer, so the choice was limited to it as well. An implementation Java code samples are available on the Web and, therefore, open-source, provided free of charge. To have a stable and supported version, we have opted for one provided by Apache Software Foundation. Sources can be found in Lucene or OpenNLP software products [Apache, 2013a, Apache, 2013b].

7 SentiMenthal - Data Extraction, Preprocessing and Storage

Technical Implementation Details

7.1. Data Extraction

Message extraction task is dedicated to the three exctraction classes - “MessengerExtractor”, “SkypeExtractor”, “WhatsAppExtractor”. The extraction process is fired from application GUI and is based on the Accessibility Service trick. The idea behind is to emulate user actions, while in a selected application, and grab all messages available on device’s display.

The approach is flawed in terms of speed and accuracy, but was the only viable solution at the time. As of current state, Facebook provides full-featured API to access messages. And WhatsApp’s messages can be retrieved from device’s filesystem - which is a serious security problem, by our judgement.

7.1.1. Utilizing Accessibility Service

The cornerstone here is the “MyAccessibilityService” class. Using an Inversion-of-Control (IOC) it is subscribed to Android’s event propagation system. This is done by extending of abstract “AccessibilityService” and overriding its methods. If the service registers suceesssfully within the system, it will receive broadcasted events through the “onAccessibilityEvent” method for further manipulations. We are interested in events of three types:

- AccessibilityEvent.TYPE_WINDOW_STATE_CHANGED
- AccessibilityEvent.TYPE_WINDOW_CONTENT_CHANGED

- AccessibilityEvent.TYPE_VIEW_SCROLLED

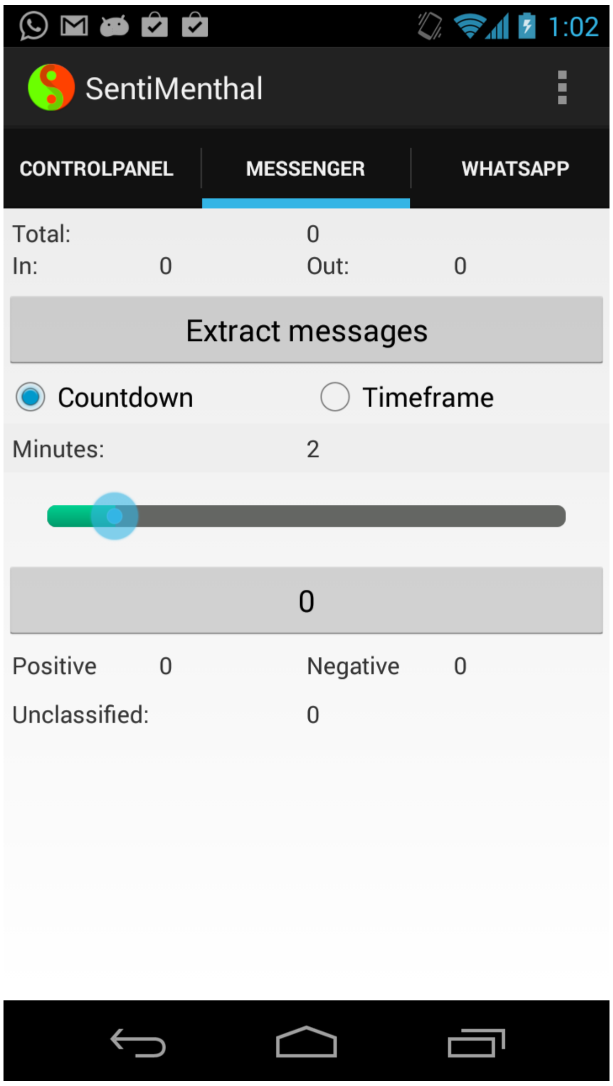
As soon as we spot an event of one of these types, we proceed with message extraction procedure. An event object provides means to access instance of "AccessibilityNodeInfo", that eventually contains information about device's display state in a form of a tree. To assure that event comes from one of the supported applications, we check package signature, which is unique to each app. After an event passes two checks, it is directed to one of the extraction classes. Extraction classes on their part, traverse the node tree to extract messages and also fire new actions, which lead to new events with updated display data. In 7.1 is the code sample for event handling (logging and error handling from original program code are omitted):

```java
@Override
public void onAccessibilityEvent(final AccessibilityEvent event) {

  int eventType = event.getEventType();
  if (eventType == AccessibilityEvent.TYPE_WINDOW_STATE_CHANGED
      || eventType == AccessibilityEvent.
       TYPE_WINDOW_CONTENT_CHANGED
      || eventType == AccessibilityEvent.TYPE_VIEW_SCROLLED) {

    final AccessibilityNodeInfo node = event.getSource();

    CharSequence packageName = node.getPackageName();
    if (packageName == null) {
      return;
    }

    if (collectMessengerData && packageName.equals(Apps.MESSENGER.
       packageName)) {
      me.handleEvent(this, node);
    } else if (collectWhatsappData && packageName.equals(Apps.
       WHATSAPP.packageName)) {
      wa.handleEvent(this, node);
    } else if (collectSkypeData && packageName.equals(Apps.SKYPE.
       packageName)) {
      se.handleEvent(this, node);
    }
  }
}
```

Listing 7.1: Event capture and filtering

7.1.2. Extraction from Applications

Every application we support differs in a way messages are exrtacted. Although, there is a generic way to gather messages (goto conversations list -> select next



Total: 0
In: 0 Out: 0

[✓ Countdown] Timeframe

Minutes: 2

Positive 0 Negative 0
Unclassified: 0

Figure 7.1.: User interface for each of supported applications.

conversation -> scroll up for older messages -> ... repeat all over), there are nuances in implementation for each particular app. Message extraction is initiated from SentiMenthal application's GUI (see 7.1). A user selects an application by sliding windows, decides whether he wants to limit extraction process with a countdown way or by selecting a timeframe (last day, week or a month). The process itself is time-consuming and averages at around 100 messages/minute. During extraction, a user is notified through an overlay button (put on top of application window) and may not interact with the device. In case, when user registers an interface action ducring extaction, accessibility service may behave unexpectedly upto a crash of both source application and SentiMenthal app. In case a user is willing to wait no more, he may click the overlay button to abort extraction and return to SentiMenthal (see 7.2).

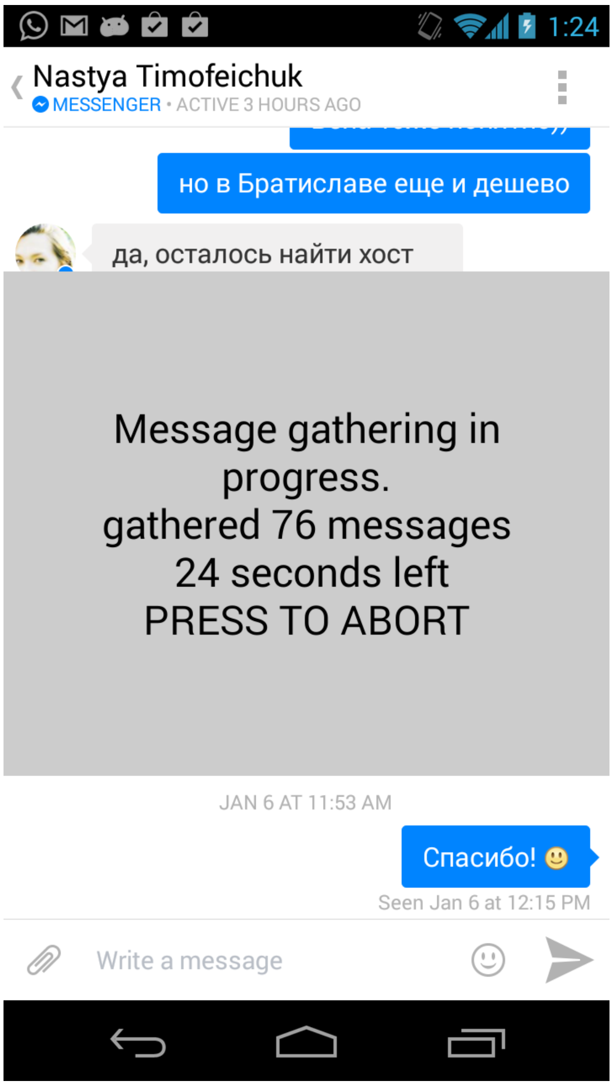


Figure 7.2.: Messages extraction in progress, with abort button overlay.

7.2. Data Storage

- sqlite database - implemented schema description - performance tests

Message storage is realized using an embedded SQLite database management system. The architecture employs two tables - one for storing raw messages, another for classified ones. The two tables are - “MessageSQLHelper” and “SentimentSQLHelper” located in the package “org.menthal.db” along with corresponding data access objects for controlling - “MessageDao” and “SentimentDao”.

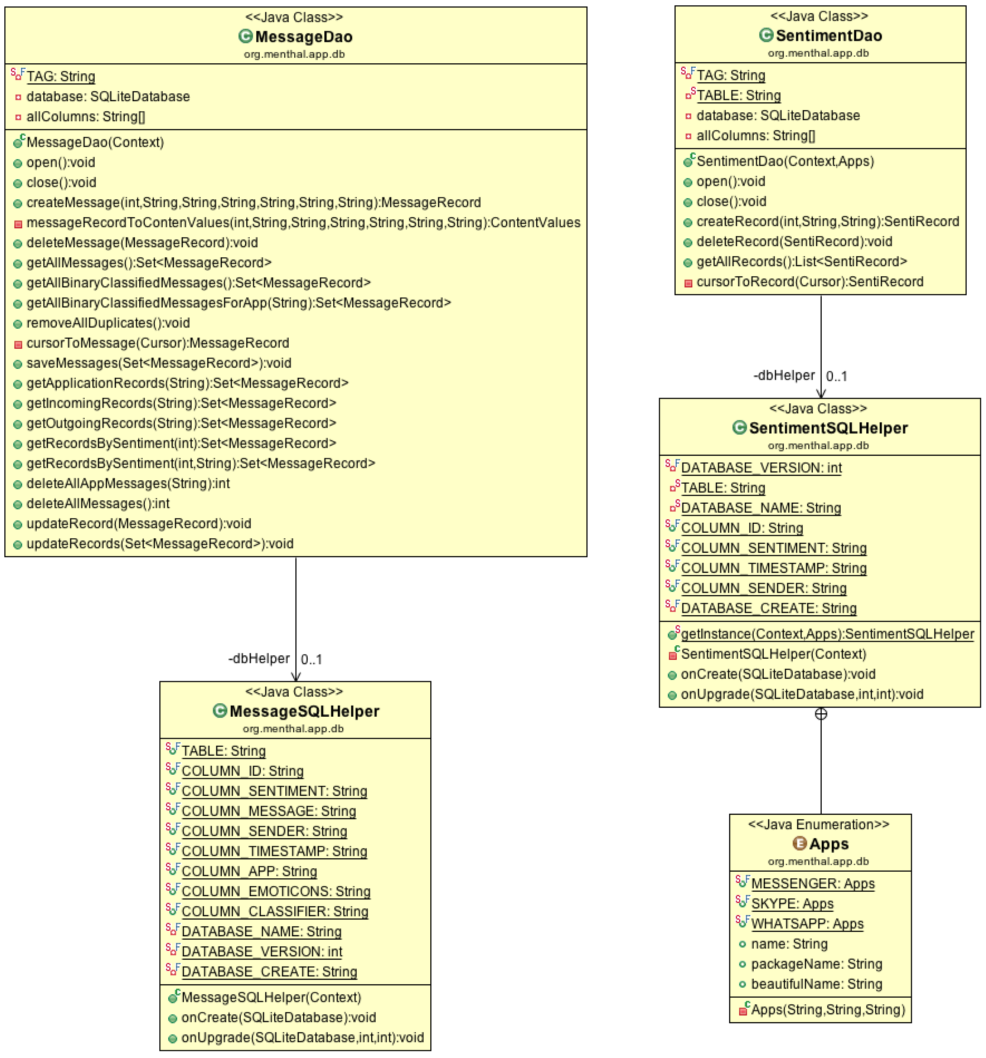

<<Java Class>>
MessageDao
org.menthal.app.db
TAG: String
database: SQLiteDatabase
allColumns: String[]
MessageDao(Context)
open():void
close():void
createMessage(int,String,String,String,String,String,String):MessageRecord
messageRecordToContenValues(int,String,String,String,String,String):ContentValues
deleteMessage(MessageRecord):void
getAllMessages():Set<MessageRecord>
getAllBinaryClassifiedMessages():Set<MessageRecord>
getAllBinaryClassifiedMessagesForApp(String):Set<MessageRecord>
removeAllDuplicates():void
cursorToMessage(Cursor):MessageRecord
saveMessages(Set<MessageRecord>):void
getApplicationRecords(String):Set<MessageRecord>
getIncomingRecords(String):Set<MessageRecord>
getOutgoingRecords(String):Set<MessageRecord>
getRecordsBySentiment(int):Set<MessageRecord>
getRecordsBySentiment(int,String):Set<MessageRecord>
deleteAllAppMessages(String):int
deleteAllMessages():int
updateRecord(MessageRecord):void
updateRecords(Set<MessageRecord>):void
-dbHelper
0..1
<<Java Class>>
MessageSQLHelper
org.menthal.app.db
TABLE: String
COLUMN_ID: String
COLUMN_SENTIMENT: String
COLUMN_MESSAGE: String
COLUMN_SENDER: String
COLUMN_TIMESTAMP: String
COLUMN_APP: String
COLUMN_EMOTICONS: String
COLUMN_CLASSIFIER: String
DATABASE_NAME: String
DATABASE_VERSION: int
DATABASE_CREATE: String
MessageSQLHelper(Context)
onCreate(SQLiteDatabase):void
onUpgrade(SQLiteDatabase,int,int):void
<<Java Class>>
SentimentDao
org.menthal.app.db
TAG: String
TABLE: String
database: SQLiteDatabase
allColumns: String[]
SentimentDao(Context,Apps)
open():void
close():void
createRecord(int,String,String):SentiRecord
deleteRecord(SentiRecord):void
getAllRecords():List<SentiRecord>
cursorToRecord(Cursor):SentiRecord
-dbHelper
0..1
<<Java Class>>
SentimentSQLHelper
org.menthal.app.db
DATABASE_VERSION: int
TABLE: String
DATABASE_NAME: String
COLUMN_ID: String
COLUMN_SENTIMENT: String
COLUMN_TIMESTAMP: String
COLUMN_SENDER: String
DATABASE_CREATE: String
getInstance(Context,Apps):SentimentSQLHelper
SentimentSQLHelper(Context)
onCreate(SQLiteDatabase):void
onUpgrade(SQLiteDatabase,int,int):void
<<Java Enumeration>>
Apps
org.menthal.app.db
MESSENGER: Apps
SKYPE: Apps
WHATSAPP: Apps
name: String
packageName: String
beautifulName: String
Apps(String,String,String)

Figure 7.3.: Data storage UML diagram

7.3. Text Preprocessing

In order to normalize textual data, a preprocessing subroutine has to be defined. An important rule for correct classification, is that a preprocessing procedure has to match for both input text message and the one used to normalize classifier's training dataset. Among the most useful techniques for textual information refinement are: letter case unification, stripping off of special characters, removal of user identification and other irrelevant information (usernames, hashtags, email, urls), language detection and others. All of the aforementioned methods are applied in our application and are discussed in the next subsections.

7.3.1. Normalized Text Consistency

In the vast number of text preprocessing configurations possible, a tool to control proper usage is required. Such mechanism is integrated in our application. A class “PreprocessingEnum” is created to store labels to most common configurations used, with detailed descriptions of a pipeline (see 7.2). Next, for each trained classificator model, a we add an instance variable with a link to the enum. This way, during the classification process, we are able to identify exactly which combination of preprocessing techniques was used. This data, of course, is preserved after serialization-deserialiazation processes.

```java
package org.menthal.analysis.text;

public enum PreprocessingEnum {
  NOSTEM("Lower case; filter: username, hashtags, urls, emails,
      numbers; collapse repeated symbols", 1),
  STEM("Lower case; filter: username, hashtags, urls, emails,
      numbers, stem; collapse repeated symbols", 2),
  NOSTEMNEG("Lower case; filter: username, hashtags, urls, emails,
       numbers; collapse repeated symbols; create negated pairs",
      3),
  STEMNEG("Lower case; filter: username, hashtags, urls, emails,
      numbers; stem; collapse repeated symbols; create negated
      pairs", 4);
  private int value;

  private PreprocessingEnum(String description, int value) {
    this.value = value;
  }

  public int getValue() {
    return value;
  }
}
```

Listing 7.2: Text normalization consistency control

7.3.2. Normalization Methods

The enum structure is further utilized in the dedicated to text preprocessing routines class - “Processor”. A method making decision on the shape of preprocessing pipeline is given in the 7.3. In the second method provided, are five common filtering routines. All five rely on regular expression search patterns. We provide aforementioned expressions in 7.4. Naturally, it is hard (some times even impossible) to predict and account for all extreme cases to be matched properly. Given patterns are partially found on the Internet, and have been praised for accuracy. A powerful service for testing regular expressions can be found at “http://www.rubular.com”.

```java
private static String[] normalize(String message,
    PreprocessingEnum type) {
  String result = message.toLowerCase(Locale.ENGLISH);
  String[] tokens;

  switch (type) {
  case NOSTEM:
    result = removeCommonStructures(result);

    tokens = SU.splitToWords(result);
    tokens = SU.toWordPairsNegationContractions(tokens);
    tokens = TextProcessor.removeStopWords(tokens);
    tokens = SU.collapseRepeatedCharacters(tokens);
    return tokens;
  case STEM:
    result = removeCommonStructures(result);

    tokens = SU.splitToWords(result);
    tokens = TextProcessor.stemTokens(tokens);
    tokens = SU.toWordPairsNegationContractions(tokens);
    tokens = TextProcessor.removeStopWords(tokens);
    tokens = SU.collapseRepeatedCharacters(tokens);
    return tokens;
  case NOSTEMNEG:
    result = removeCommonStructures(result);

    tokens = SU.splitToWords(result);
    tokens = SU.toWordPairsNegationContractions(tokens);
    tokens = TextProcessor.removeStopWords(tokens);
    tokens = SU.collapseRepeatedCharacters(tokens);
    return tokens;
  case STEMNEG:
    result = removeCommonStructures(result);

    tokens = SU.splitToWords(result);
    tokens = TextProcessor.stemTokens(tokens);
    tokens = SU.toWordPairsNegationContractions(tokens);
    tokens = TextProcessor.removeStopWords(tokens);
    tokens = SU.collapseRepeatedCharacters(tokens);
    return tokens;
  default:
    return normalize(message, PreprocessingEnum.STEM);
  }
}

private static String removeCommonStructures(String result) {
  result = TextProcessor.removeUserTags(result);
  result = TextProcessor.removeHashTags(result);
  result = TextProcessor.removeUrls(result);
  result = TextProcessor.removeEmails(result);
  result = TextProcessor.removeNumbers(result);
  return result;
}
```

Listing 7.3: Methods controlling text normalization

```java
public static String removeUserTags(String result) {
  String regex = "/@([A-Za-z0-9_]{1,15})/";
  return result.replaceAll(regex, "");
}

public static String removeNumbers(String result) {
  String regex = "[0-9]";
  return result.replaceAll(regex, "");
}

public static String removeHashTags(String result) {
  String regex = "\\B#((([a-zA-Z]+[a-zA-Z0-9_-]*)|([a-zA-Z0-9_-]*[
      a-zA-Z]+))[a-zA -Z0-9_-]*)";
  return result.replaceAll(regex, "");
}

public static String removeEmails(String result) {
  String regex = "[_A-Za-z0-9-\\+]+(\\.[_A-Za-z0-9-]+)*@[A-Za-z0
      -9-]+(\\.[A-Za-z0-9]+)*(\\.[A-Za-z]{2,})";
  return result.replaceAll(regex, "");
}

public static String filterUrls(String str) {
  String regex = "/((([A-Za-z]{3,9}:(?:\\/\\/)?)(?:[-;:&=\\+\\$,\\
      w]+@)?[A-Za-z0-9.-]+|(?:www.|[-;:&=\\+\\$,\\w]+@)[A-Za-z0
      -9.-]+)((?:\\/[\\+~%\\/.\\w-_]*)?\\??(?:[-\\+=&;%@.\\w_]*)
      #?(?:[\\w]*))?)/";
  return str.replaceAll(regex, "");
}
```

Listing 7.4: Regular expressions based methods for common text preprocessing routines

7.3.3. Emoticon Extraction

Emoticons are handled in two ways - first, if we need to remove them, this can be done with help of regular expressions efficiently; second, if we want to extract and preserve emoticons, a better approach is to use predefined lists of emoticons. SentiMenthal provides a binary only classification of text, therefore it is sufficient for us to define two lists of smileys - negative and positive ones. Complete lists are provided in the next chapter along with implementation details.

8 SentiMenthal - Text Mining

...one paragraph...

8.1. Classifiers Setup

We employ a two-layered approach for classification. A general workflow is presented in 8.1. After preprocessing phase textual data can be used for classification. Emoticon detection has an advantage, and if any are found - the message is classified by the emoticon classifier. In case message contains no emoji, a language detection routine is executed. A message is passed further to classifiers in case English language was detected, otherwise it is discarded. Then, one of the three classificator types (apart from emoticons) is chosen. To sum up, we can get a classification result (negative / positive) from one of the four classifier types.

Twitter Corpora. Twitter platform is a useful source of training data. We selected Twitter as a source for its availability, high volume and data actuality. Although, Twitter prohibits distribution of their message corpuses by third parties, the data can be acquired directly crawling the web-site or through API. Even so, large databases of Twitter messages are freely available on the internet. After having tested several message sets, we have chosen the largest one. The final training set contains over 1,600,000 tweets. The corpus was annonated based on emoticons contained in text [Sentiment140.com, 2013]. Such approach outputs high classification accuracy and it is efficient in terms of effort required. It gives opportunuity to keep our classifier up-to-date with virtually no effort. Twitter API allows to search recent messages with various emoticons. Therefore, an automatic function to retrieve new tweets and improve the model can be added in the future. As a

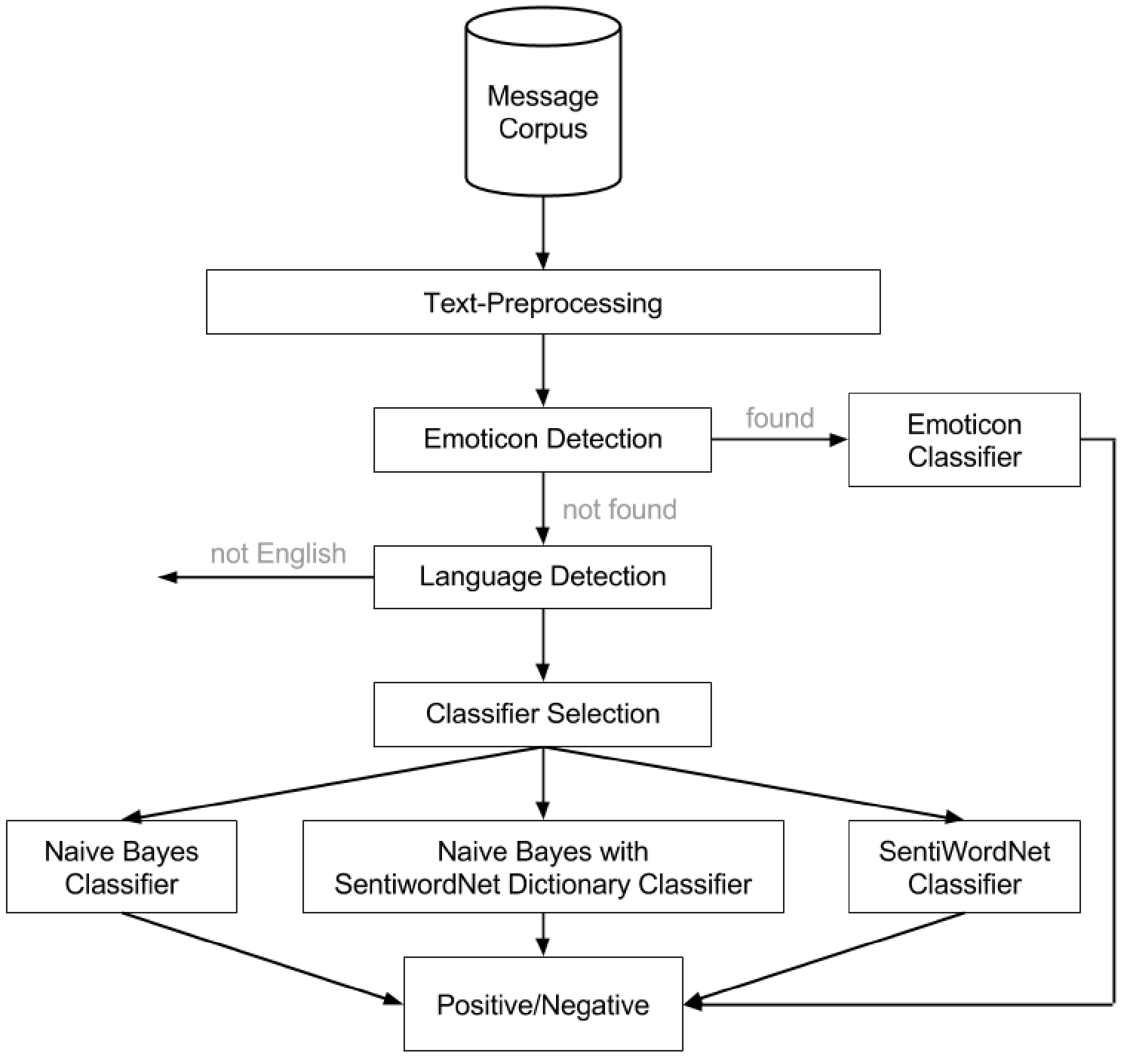


Figure 8.1.: Classification Module Workflow.

temporary solution to augment existing corpora with recent Twitter messages, we have developed a simplistic Python script (see Appendix A).

8.1.1. Negation handling (one paragraph only)

8.1.2. Emoticon Classification

As mentioned in previous chapters, emoticons are highly indicative artifacts in communication and, therefore, useful for sentiment analysis applications. We have implemented a primitive, yet powerful classifier for emoticons. In the chosen configuration, this classifier dominates over other variants of classifiers, due to its low requirements and high accuracy. Classification decision is made based on fact of inclusion of the found emoticon in on of the two classes - negative/positive (see

8.1). The classifier can also be used for emoticon detection and sending messages to Naive Bayes implementations for further improvement of their models. Another useful application is possible for the problem irony detection - we can detect when an emoticon classifier's output differs from the machine learning classifier. Such cases are indication, that with high probability, the message contains irony. Such approach would eliminate the need for sophisticated context analysis and allow collection of irony messages corpus automatically.

8.1.3. Naive Bayes Training Setup

Larger part of the corpus are messages without emoticons. Evaluation tests during development have indicated that simplistic classifiers that rely on a labeled dictionary do not profide sufficient accuracy in real-world applications. This happens due to a vast syntactical variety in social networks with limited length of a message. In contrast, classifiers based on Naive Bayes algorithm, show comparatively high accuracy (73-88% for implementations in SentiMenthal). 8.2 presents a general workflow for such classifiers. It is applicable to both variants of Naive Bayes classifiers, with one distinction, that one version omits the part with SentiWordDictionary. The chart depicts two sources of input data for classifiers - training dataset from Twitter and a SentiWordNet reference library. The messages corpus overcomes more text-preprocessing procedures than the labeled dictionary. The cloud on the figure shows a common virtual lexical base for the two sources. It has two meanings: first - in case when SentiWordNet dictionary is used, that the results of preprocessing of the two sources stays consistent to one another; second - all the messages received as a training data overcome exactly the same preprocessing pipeline. Moreover, the testing dataset has to be processed equally.

Positive	Negative
:) :] :} :o) :o] :o} :-] :-) :-} =) =] =} =^] =^) =^} :B :-D :-B :^D :^B =B =^B =^D :') :'] :'} =') ='] ='} <3 ^.^ ^-^ ^_^ ^__^ ^___^ ^____^ ^_____^ ^^ :* =* :-* ;) ;] ;} :-p :-P :-b :^p :^P :^b =P =p \o\ /o/ :P :p :b =b =^p =^P =^b \o/ (y)))))))	D: D= D-: D^: D^= :(:[:{ :o(:o[:^(:^[:^{ =^(=^{ >=(>=[>={ >=(>:-{ >:-[>:-(>=^[>:-(:-[:-(=(=[={ =^[>:-=(>=[>=^(:'(:'[:'{ ='{ ='(='[=\ :\ =/ :/ =$ o.O O_o Oo :$:-{ >:-{ >=^{ :o{ (((((

Table 8.1.: Two classes of emoticons.

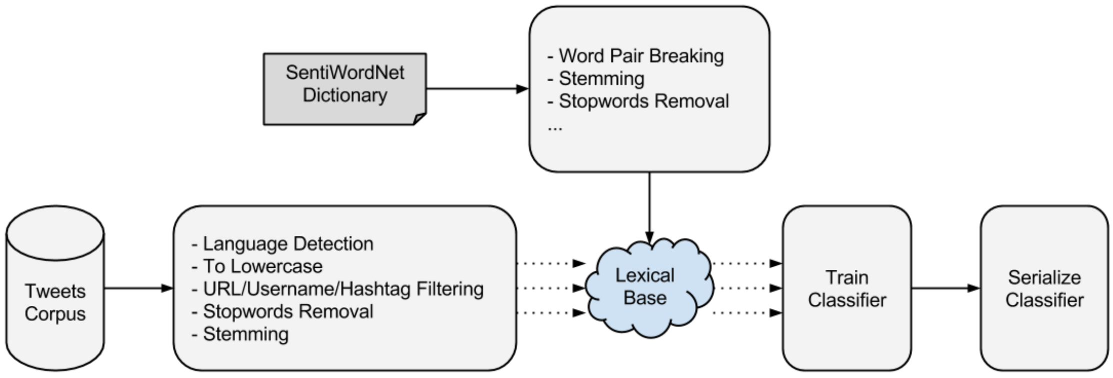


Figure 8.2.: Naive Bayes classifier training workflow.

Classifier training is a computationally exhausting process. Also, in our configuration, classification model would not be changing frequently. Therefore, training is executed on a full-featured computer. Afterwards, the classifier is serialized for later use on smartphone. A reference implementation (with best results) is provided in Appendix B. For better results, it is good to have a large training dataset. As is evident from the previous part of this chapter, we chose Twitter as a source of such data. The final training set consisted of more than 1590000 messages, after uncompatible rows were filtered.

8.1.4. Classification Module

According to modular organization of SentiMenthal, all classes related to text classification are located in a dedicated package - “org.menthal.analysis.classification”. Naive Bayes based classifiers are standardized through the “IClassifier” interface (see 8.3); others are put independently (see 8.4). The “ClassifierRunner” class provides complimentary methods (data manipulation, data structures conversion, performance testing) for classifiers’ operation. The “DataVector” class is a helper data-structure for convenience of Naive Bayes based classifiers. A classifier called “SimpleClassifier” uses SentiWordNet labeled dictionary for sentiment calculation; and a Bloom Filter structure for fast language detection.

8.2. Known Issues

Although, we were able to reach classification accuracies of as high as 88% on a test dataset, the results are expected to be 5-15% lower in real-world applications.

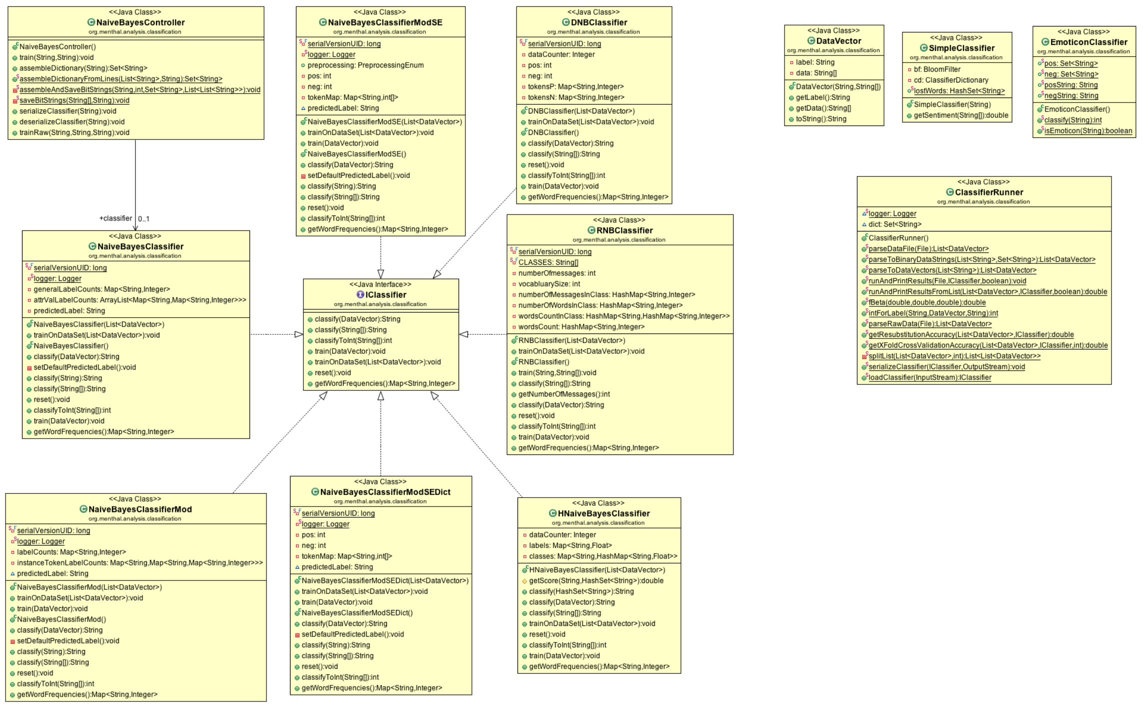
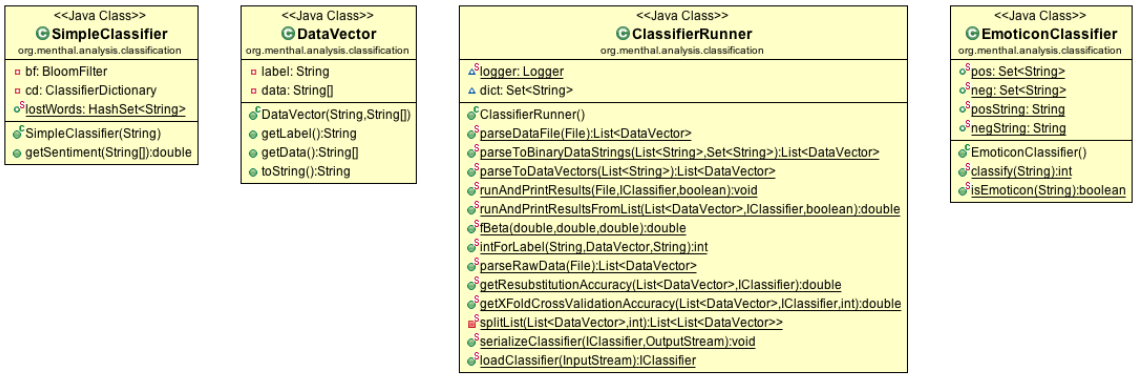


Figure 8.3.: Classification module structure (part 1).


<<Java Class>>
SimpleClassifier
org.menthal.analysis.classification
bf: BloomFilter
cd: ClassifierDictionary
lostWords: HashSet<String>
SimpleClassifier(String)
getSentiment(String[]):double
<<Java Class>>
DataVector
org.menthal.analysis.classification
label: String
data: String[]
DataVector(String,String[])
getLabel():String
getData():String[]
toString():String
<<Java Class>>
ClassifierRunner
org.menthal.analysis.classification
logger: Logger
dict: Set<String>
ClassifierRunner()
parseDataFile(File):List<DataVector>
parseToBinaryDataStrings(List<String>,Set<String>):List<DataVector>
parseToDataVectors(List<String>):List<DataVector>
runAndPrintResults(File,IClassifier,boolean):void
runAndPrintResultsFromList(List<DataVector>,IClassifier,boolean):double
fBeta(double,double,double):double
intForLabel(String,DataVector,String):int
parseRawData(File):List<DataVector>
getResubstitutionAccuracy(List<DataVector>,IClassifier):double
getXFoldCrossValidationAccuracy(List<DataVector>,IClassifier,int):double
splitList(List<DataVector>,int):List<List<DataVector>>
serializeClassifier(IClassifier,OutputStream):void
loadClassifier(InputStream):IClassifier
<<Java Class>>
EmoticonClassifier
org.menthal.analysis.classification
pos: Set<String>
neg: Set<String>
posString: String
negString: String
EmoticonClassifier()
classify(String):int
isEmoticon(String):boolean
Figure 8.4.: Classification module structure (part 2).

First of all, the main problem is with message length and vocabulary used. Short messages may be left with zero tokens after preprocessing, therefore, are simply ignored. Another aspect is with available dictionaries within classifiers - they are limited. Even though, the resulting dictionary from SentiWordNet is about 120000 tokens, this still does not account the vast variety of slang words, misspellings, shortenings etc. The second limitation may be attenuated manually, by including of the most frequent words from the message corpus.

9 Experimental Evaluation

Experimental evaluation is an important part of effective development. We have run a series of tests to estimate quality of our software and to reveal limitations along with possible ways of improvement for the future. Evaluation routine includes experiments for machine learning based approaches as well as lexical-based ones. Two types of Naive Bayes based classifier and a classifier based purely on SentiWordNet dictionary are tested. Accuracy measurements are performed with varying number of training epochs for machine learning classifiers. Preprocessing times are measured for the four selected configurations. We also provide a sample chart (sentiment over time) from the SentiMenthal application.

9.1. Preprocessing

Our text-preprocessing configuration involves 5 to 6 filtering routines based on regular expressions, which are known to have performance issues. On 9.1 is the comparison for the four configurations. Evidently, text preparation requires substantial resources, compared to classifier training. Configurations with stemming procedure perform worse, but not by much.

SentiMenthal performance evaluation

Charts comparing

- Classifiers performance (accuracy)
- Classifiers performance (speed)
- Naive Bayes with varying text pre-processing setups

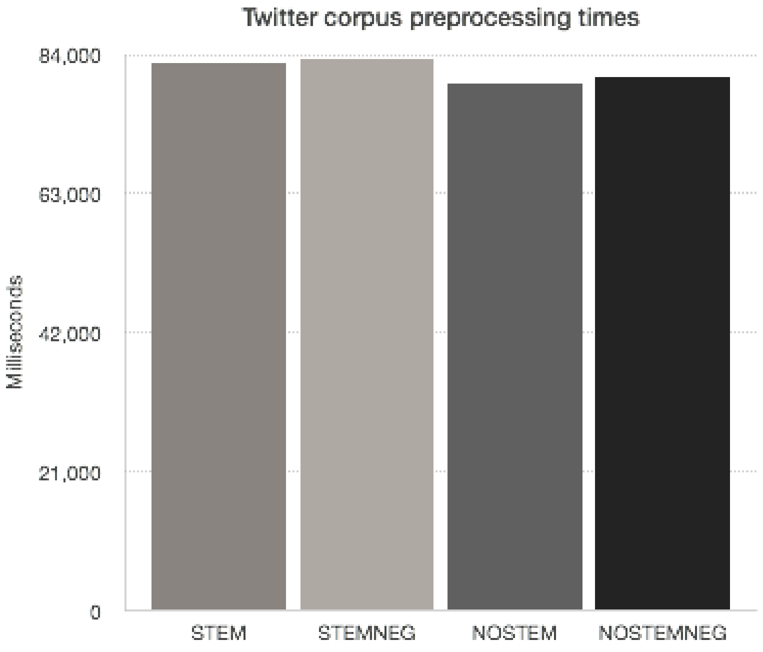


Figure 9.1.: Twitter corpus preprocessing times.

- Naive Bayes with full dictionary vs. high-frequency cut dictionary vs. unstemmed SentiWordNet as dict. vs. stemmed SentiWordNet as dict.
- average response time vs. date

9.2. Classifier Training

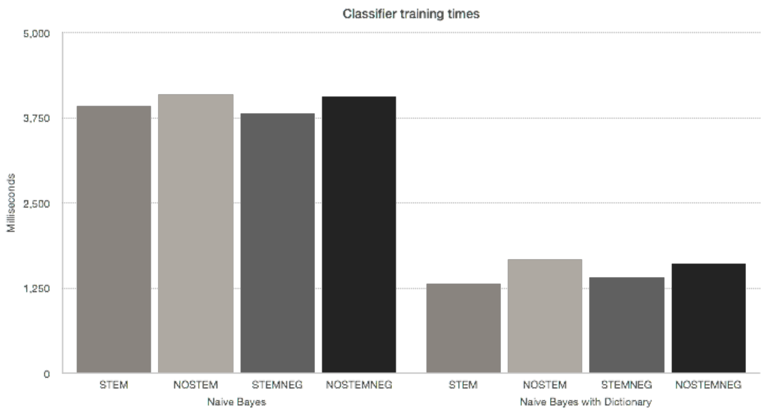


Figure 9.2.: Classifier Training Times comparison.

Naive Bayes classifier require previous training to create a model. The procedure is computationally demanding. The fact that the model does not have to changed frequently is advantageous - we are able to train our classifiers on a standard personal computer and use it in SentiMenthal app later. For our test we have used a corpus of 1,600,000 tweets mentioned previously. The result is shown on 9.2, it is reasonably good as this result was produced on a regular laptop machine and the training times are negligible for such amount of data. Due to a fast Bloom Filter structure in the Naive Bayes with Dictionary classifier, it performs 3 to 4 times faster.

9.3. Classifier Accuracy

The set of figures: Figure 9.3, Figure 9.4, Figure 9.5, Figure 9.6 we show accuracy plots for all major versions of Naive Bayes classifiers - both types (unbounded dictionary and SentiWordNet bounded versions) and all four preprocessing workflows - STEM, NOSTEM, NOSTEMNEG, STEMNEG. The performance differs marginally, but the version with STEM outperforms others.

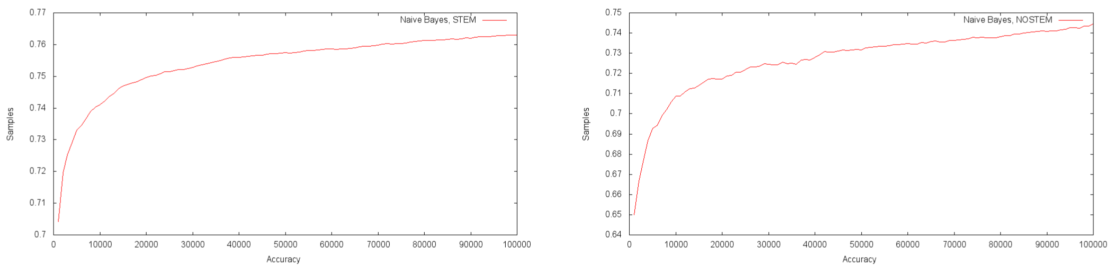


Figure 9.3.: Naive Bayes classifier (STEM, NOSTEM) accuracy.

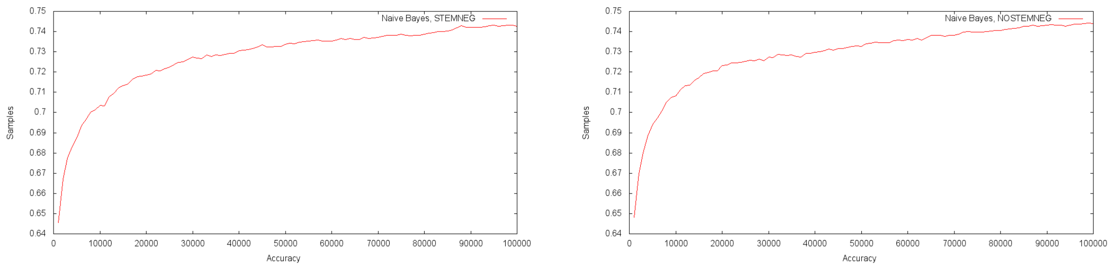


Figure 9.4.: Naive Bayes classifier (STEMNEG, NOSTEMNEG) accuracy.

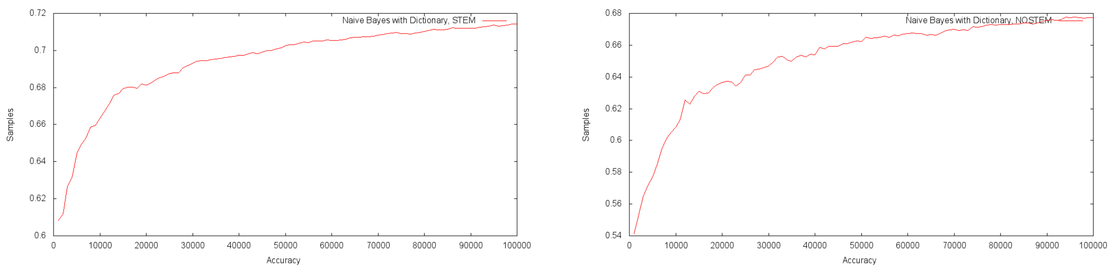


Figure 9.5.: Naive Bayes classifier with SentiWordNet (STEM, NOSTEM) accuracy.

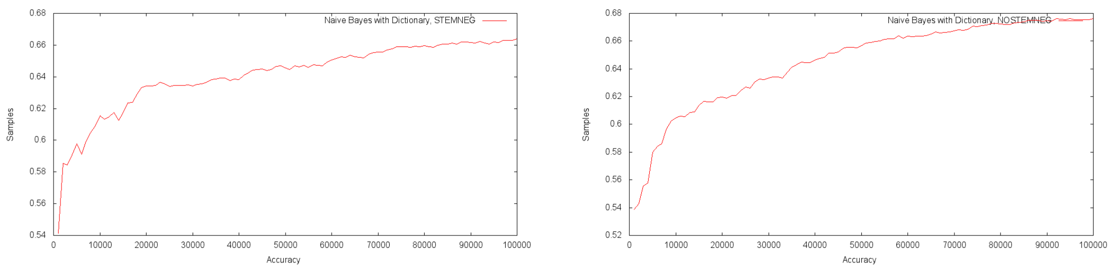


Figure 9.6.: Naive Bayes classifier with SentiWordNet (STEMNEG, NOSTEMNEG) accuracy.

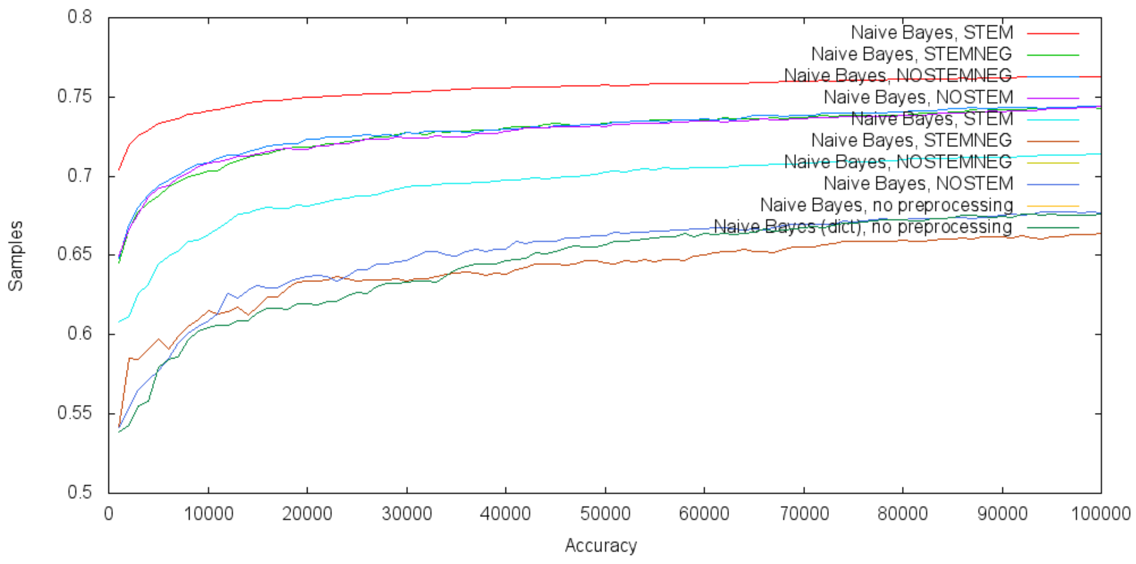


Figure 9.7.: Compared accuracy of 10 Naive Bayes classifiers.

10 Conclusion and Future Work

Stress and mental diseases are important problems in developed countries. Since traditional psychology methods are not able to present a viable solution, we need to develop new strategies to overcome the issue. Spread of open-source mobile operating systems, rapid increase in computing power of mobile devices, and popularization of wearable gadgets allow new applications for human behaviour tracking to be developed. This thesis has addressed only one possible aspect of research through mobile devices - sentiment analysis of user messages. The approach has proven to be robust and useful.

This work opens a perspective on what is possible with the help of modern mobile devices. To achieve our goals, we had to use unconventional data extraction methods (Accessibility Service in Android OS) as well as search for new solutions and optimizing of old ones. Text mining on a mobile device has turned out to be a challenging task. Throughout our development process, with error and trial we have tested, accepted and discarded numerous configurations and architectural decisions.

To prove the chosen strategies, an experimental evaluation has been executed. We summarized received results and outlined directions for future improvements. The best overall performance was achieved with a hybrid message classification pipeline, which employed a preset emoticon classifier on the first level, and a Naive Bayes classifier with a preprocessing workflow that excluded a stemming part. Bloom Filter data structure have proven to be extremely useful for given circumstances. In the next section, we give our understanding and vision on what could be improved in our analysis.

10.1. Future Work

Natural Language Processing is a tedious technical problem. The potential for improvement is still enormous. Processing power of mobile devices grows at extreme rates, guaranteeing for improvements in quality of human languages understanding. More powerful machine learning techniques will become applicable on smartphones, and this will open new horizons for both commercial and personal applications for human behaviour analysis.

How SentiMenthal could be improved in the future? Naturally, more advanced machine learning techniques can be plugged-in to the application, and tested for performance. Apart from continuous improvements in text analysis methods, we can focus on different kinds of data to extract knowledge regarding user's mental state. One could think of tracking Facebook activities of a user, such as number of likes clicked, number and sentiment of statuses announced, number of links clicked and their content's sentiments.

Various commercial opportunities are available for this kinds of applications. Ranging from sedative pills advertisment to deals propositions (expecting for impulsive buying). Another application we can imagine is for large corporations, where employee productivity is of high importance and the company would track their mental state and make appropriate business trips decisions.

A
Extraction of Twitter Messages with Emoticons

```python
from TwitterSearch import *
try:
    tso = TwitterSearchOrder() # create a TwitterSearchOrder
        object
    tso.setKeywords([':)', ':(', ':D', 'D:'])
    tso.setLanguage('en')
    tso.setCount(10)
    tso.setIncludeEntities(False) # avoid redundant entity
        information

    ts = TwitterSearch(
        consumer_key = '1ylfGVll4CFrgwAQBTLQ',
        consumer_secret = '4
            noCSolIG4r7MJlWg8x0Wo7kp0zd5oyOy7CsD8NfyJo',
        access_token = '337112301-
            sRjh9GLvu9svihYgaZnRdDlPmqugs6NjJw2loL1D',
        access_token_secret = '
            lwo0AAjvdT525mq7ZIBSgaP0kX6i5DkICl6L8Cxu90k3V'
     )

    f = open('emoticon_tweets.txt', 'w')
    for tweet in ts.searchTweetsIterable(tso): # extraction...
        f.write(tweet['text'].encode('utf-8') )
        f.write('\n')
    f.close()

except TwitterSearchException as e: # error output
    print(e)
```

Listing A.1: Simplistic Python script for extraction of Twitter messages with emoticons.

B
Reference Naive Bayes Classifier Implementation

```java
package org.menthal.analysis.classification;

import java.io.Serializable;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.logging.ConsoleHandler;
import java.util.logging.Logger;
import java.util.logging.SimpleFormatter;
import org.menthal.analysis.text.PreprocessingEnum;

public class NaiveBayesClassifierModSE implements IClassifier,
    Serializable {

  private static final long serialVersionUID =
      -5808977468958125765L;

  private static Logger logger = Logger.getLogger(
      NaiveBayesClassifierModSE.class.getPackage().getName());
  static {
    ConsoleHandler ch = new ConsoleHandler();
    ch.setFormatter(new SimpleFormatter());
    logger.addHandler(ch);
  }

  public PreprocessingEnum preprocessing = PreprocessingEnum.
      NOSTEM;

  private int pos = 0;
  private int neg = 0;
  private Map<String, int[]> tokenMap = new HashMap<String, int
      []>();

  String predictedLabel = null;

  public NaiveBayesClassifierModSE(List<DataVector> trainingData)
      {
    trainOnDataSet(trainingData);
  }

  public void trainOnDataSet(List<DataVector> trainingData) {
    if (trainingData == null || trainingData.size() < 1) {
      return;
    }

    for (DataVector vector : trainingData) {
      train(vector);
    }
  }

  public void train(DataVector vector) {
    String label = vector.getLabel();

    if (label == null || label.length() == 0) {
      System.out.println("Skip instance due to incorrect label.");
      return;
    }

    int classIndex = 0;

    if (label.equals("+1")) {
      pos++;
      classIndex = 1;
    } else if (label.equals("-1")) {
      neg++;
      classIndex = 0;
    } else {
      return;
    }
    setDefaultPredictedLabel();

    for (String token : vector.getData()) {
      int[] arr = tokenMap.get(token);
      if (arr == null) {
        arr = new int[] { 0, 0 };
        tokenMap.put(token, arr);
      }
      arr[classIndex]++;
    }
  }

  public NaiveBayesClassifierModSE() {
  }

  @Override
  public String classify(DataVector vector) {
    double posProduct = 1; // Assumed both classes equally likely
        ...
    double negProduct = 1; // Assumed both classes equally likely
        ...

    for (String token : vector.getData()) {
      int[] arr = tokenMap.get(token);
      if (arr != null) {
        posProduct *= ((double) arr[1] + 0.2) / pos;
        negProduct *= ((double) arr[0] + 0.2) / neg;
      }
    }

    if (posProduct >= negProduct) {
      return "+1";
    } else {
      return "-1";
    }
  }

  private void setDefaultPredictedLabel() {
    if (pos >= neg) {
      predictedLabel = "+1";
    } else {
      predictedLabel = "-1";
    }
  }

  public String classify(String string) {
    DataVector vector = new DataVector("", string.split(" "));
    return classify(vector);
  }

  @Override
  public String classify(String[] data) {
    return classify(new DataVector("", data));
  }

  @Override
  public void reset() {
    pos = 0;
    neg = 0;
    tokenMap.clear();
  }

  @Override
  public int classifyToInt(String[] data) {
    double posProduct = 1; // Assume both classes equally likely
        ...
    double negProduct = 1; // Assume both classes equally likely
        ...

    for (String token : data) {
      int[] arr = tokenMap.get(token);
      if (arr != null) {
        posProduct *= ((double) arr[1] + 0.2) / pos;
        negProduct *= ((double) arr[0] + 0.2) / neg;
      }
    }

    if (posProduct >= negProduct) {
      return 1;
    } else {
      return -1;
    }
  }

  @Override
  public Map<String, Integer> getWordFrequencies() {
    Map<String, Integer> map = new HashMap<String, Integer>();

    for (String key : tokenMap.keySet()) {
      map.put(key, tokenMap.get(key)[1] + tokenMap.get(key)[0]);
    }

    return map;
  }
}
```

Listing B.1: **Naive Bayes classifier, Java implementation.**

C
Chart Generation Method

```java
public Intent executeTime(Context context, Map<Date, int[]>
    frequencies) {
  String[] titles = new String[] { "Positive", "Negative", "
      Unknown" };

  List<double[]> values = new ArrayList<double[]>();
  List<Date[]> dates = new ArrayList<Date[]>();
  Date[] dateArr = new Date[frequencies.size()];
  double[] pos = new double[frequencies.size()];
  double[] neg = new double[frequencies.size()];
  double[] unkn = new double[frequencies.size()];
  int i = 0;
  for (Date d : frequencies.keySet()) {
    dateArr[i] = d;
    pos[i] = frequencies.get(d)[0];
    unkn[i] = frequencies.get(d)[1];
    neg[i] = frequencies.get(d)[2];
    i++;
  }

  dates.add(dateArr);
  dates.add(dateArr);
  dates.add(dateArr);
  values.add(pos);
  values.add(neg);
  values.add(unkn);

  int[] colors = new int[] { Color.GREEN, Color.RED, Color.YELLOW
      };
  PointStyle[] styles = new PointStyle[] { PointStyle.CIRCLE,
      PointStyle.CIRCLE, PointStyle.POINT };

  double[] bounds = getMinMax(dateArr);
  double maxFrequency = getMax(pos, neg, unkn);

  XYMultipleSeriesRenderer renderer = buildRenderer(colors, styles
      );
  setChartSettings(renderer, "Sentiment change", "Date", "Number
      of messages", bounds[0], bounds[1], -10, maxFrequency + 10,
      Color.LTGRAY, Color.LTGRAY);
  renderer.setYLabels(10);
  renderer.setXRoundedLabels(false);
  renderer.setZoomButtonsVisible(true);
  renderer.setApplyBackgroundColor(true);
  renderer.setBackgroundColor(Color.parseColor("#111111"));

  Intent intent = ChartFactory.getTimeChartIntent(context,
      buildDateDataset(titles, dates, values), renderer, "yyyy MMM
      dd");
  return intent;
}
```

Listing C.1: **Sample code for chart generation AChartEngine.**

Bibliography

[AChartEngine, 2013] AChartEngine (2013). AChartEngine. http://www.achartengine.org/content/download.html. 02.12.2013. (cited on page 41)

[AChartEngine Source, 2013] AChartEngine Source (2013). AChartEngine. https://code.google.com/p/achartengine/. 02.12.2013. (cited on page 42)

[Ahern and Schwartz, 1979] Ahern, G. L. and Schwartz, G. E. (1979). Differential lateralization for positive versus negative emotion. *Neuropsychologia*, 17(6):693–698. (cited on page 9)

[allpsych.com, 2013] allpsych.com (2013). Motivation and emotion. http://allpsych.com/psychology101/emotion.html. (cited on page 5)

[Android.com, 2013] Android.com (2013). Android: Report bugs. http://source.android.com/source/report-bugs.html. (cited on page 28)

[Apache, 2013a] Apache (2013a). Apache Lucene. http://lucene.apache.org/. 02.12.2013. (cited on page 43)

[Apache, 2013b] Apache (2013b). OpenNLP. http://opennlp.apache.org/. 02.12.2013. (cited on page 43)

[Apache.org, 2013] Apache.org (2013). Apache mahout. http://mahout.apache.org/. (cited on page 18)

[Baccianella et al., 2010] Baccianella, S., Esuli, A., and Sebastiani, F. (2010). Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. In *LREC*, volume 10, pages 2200–2204. (cited on page 21)

[Bloom, 1970] Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426. (cited on page 13)

[BlueStacks, 2013] BlueStacks (2013). BlueStacks. http://www.bluestacks.com/technology.html. 02.12.2013. (cited on page 39)

[Boser et al., 1992] Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM. (cited on page 25)

[CMU.edu, 2013] CMU.edu (2013). Emoticon use proposed. http://www.cs.cmu.edu/~sef/Orig-Smiley.htm. (cited on page 11)

[cs.waikato.ac.nz, 2013] cs.waikato.ac.nz (2013). Weka 3: Data mining software in java. http://www.cs.waikato.ac.nz/ml/weka/. (cited on page 18)

[Das and Chen, 2001] Das, S. and Chen, M. (2001). Yahoo! for amazon: Extracting market sentiment from stock message boards. In *Proceedings of the Asia Pacific Finance Association Annual Conference (APFA)*, volume 35, page 43. (cited on page 18)

[Dave et al., 2003] Dave, K., Lawrence, S., and Pennock, D. M. (2003). Mining the peanut gallery: Opinion extraction and semantic classification of product reviews. In *Proceedings of the 12th international conference on World Wide Web*, pages 519–528. ACM. (cited on page 18)

[Demaree et al., 2005] Demaree, H. A., Everhart, D. E., Youngstrom, E. A., and Harrison, D. W. (2005). Brain lateralization of emotional processing: historical roots and a future incorporating "dominance". *Behavioral and cognitive neuroscience reviews*, 4(1):3–20. (cited on page 9)

[developer.android.com, 2013a] developer.android.com (2013a). Android Debug Bridge. http://developer.android.com/tools/help/adb.html. 02.12.2013. (cited on page 39)

[developer.android.com, 2013b] developer.android.com (2013b). Android Emulator. http://developer.android.com/tools/help/emulator.html. 02.12.2013. (cited on page 39)

[developer.android.com, 2013c] developer.android.com (2013c). Android Hierarchy Viewer. http://developer.android.com/tools/help/hierarchy-viewer.html. 02.12.2013. (cited on page 39)

[developer.android.com, 2013d] developer.android.com (2013d). Android NotificationManager. http://developer.android.com/reference/android/app/NotificationManager.html. 02.12.2013. (cited on page 30)

[developer.android.com, 2013e] developer.android.com (2013e). Android SDK. http://developer.android.com/sdk/index.html. 02.12.2013. (cited on page 39)

[developer.android.com, 2013f] developer.android.com (2013f). App Components. http://developer.android.com/guide/components/index.html. 02.12.2013. (cited on page 30)

[developer.android.com, 2013g] developer.android.com (2013g). SDK Exploring. http://developer.android.com/sdk/exploring.html. 02.12.2013. (cited on page 39)

[developer.android.com, 2013h] developer.android.com (2013h). Security Tips. http://developer.android.com/training/articles/security-tips.html. 02.12.2013. (cited on page 34)

[Ehringer, 2010] Ehringer, D. (2010). The dalvik virtual machine architecture. *Techn. report (March 2010)*. (cited on page 31)

[Ekman and Davidson, 1994] Ekman, P. E. and Davidson, R. J. (1994). *The nature of emotion: Fundamental questions*. Oxford University Press. (cited on page 8)

[Ethnologue.com, 2013a] Ethnologue.com (2013a). Chinese, mandarin. http://archive.ethnologue.com/16/show_language.asp?code=cmn. (cited on page 12)

[Ethnologue.com, 2013b] Ethnologue.com (2013b). English. http://archive.ethnologue.com/16/show_language.asp?code=eng. (cited on page 12)

[Fernández-Tobías et al., 2013] Fernández-Tobías, I., Cantador, I., and Plaza, L. (2013). An emotion dimensional model based on social tags: Crossing folksonomies and enhancing recommendations. In *E-Commerce and Web Technologies*, pages 88–100. Springer. (cited on page 3)

[GenyMotion, 2013] GenyMotion (2013). GenyMotion Emulator. http://www.genymotion.com/features/. 02.12.2013. (cited on page 39)

[Googlesource.com, 2013] Googlesource.com (2013). Android git repositories. https://android.googlesource.com. (cited on page 28)

[Huang et al., 2003] Huang, J., Lu, J., and Ling, C. X. (2003). Comparing naive bayes, decision trees, and svm with auc and accuracy. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 553–556. IEEE. (cited on page 26)

[Joachims, 2002] Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM. (cited on page 24)

[Kahneman, 2011] Kahneman, D. (2011). *Thinking, Fast and Slow*. Farrar, Straus and Giroux. (cited on page 6)

[ling.gu.se, 2013] ling.gu.se (2013). Languages of the world. http://www.ling.gu.se/projekt/sprakfrageladan/english/sprakfakta/eng-sprak-i-varlden.html. (cited on page 12)

[Manning et al., 2008] Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge. (cited on page 16)

[Miller, 1995] Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41. (cited on page 19)

[MLOSS.org, 2013] MLOSS.org (2013). Mloss platform. http://mloss.org/. (cited on page 18)

[NLTK.org, 2013] NLTK.org (2013). Nltk 3.0. http://nltk.org/. (cited on page 18)

[OpenGL, 2013] OpenGL (2013). OpenGL. http://www.opengl.org. 02.12.2013. (cited on page 30)

[Pang and Lee, 2008] Pang, B. and Lee, L. (2008). Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, 2(1-2):1–135. (cited on page 18)

[Pelachaud, 2013] Pelachaud, C. (2013). *Emotion-oriented systems*. John Wiley & Sons. (cited on page 6)

[Plutchik, 1991] Plutchik, R. (1991). *The emotions*. University Press of America. (cited on page 7)

[Princeton.edu, 2013] Princeton.edu (2013). What is wordnet? http://wordnet.princeton.edu/. (cited on page 19)

[Ptaszynski et al., 2011] Ptaszynski, M., Rzepka, R., Araki, K., and Momouchi, Y. (2011). Research on emoticons: Review of the field and proposal of research framework. (cited on page 11)

[research.microsoft.com, 2013] research.microsoft.com (2013). The standard svm formulation. http://research.microsoft.com/en-us/um/people/manik/projects/trade-off/svm.html. (cited on page 25)

[Russell, 2003] Russell, J. A. (2003). Core affect and the psychological construction of emotion. *Psychological review*, 110(1):145. (cited on page 7)

[Russell and Barrett, 1999] Russell, J. A. and Barrett, L. F. (1999). Core affect, prototypical emotional episodes, and other things called emotion: dissecting the elephant. *Journal of personality and social psychology*, 76(5):805. (cited on page 8)

[Scherer, 2005] Scherer, K. R. (2005). What are emotions? and how can they be measured? *Social science information*, 44(4):695–729. (cited on pages 4 and 7)

[Schwartz et al., 1979] Schwartz, G. E., Ahern, G. L., and Brown, S.-L. (1979). Lateralized facial muscle response to positive and negative emotional stimuli. *Psychophysiology*, 16(6):561–571. (cited on page 9)

[Sentiment140.com, 2013] Sentiment140.com (2013). Sentiment140. http://help.sentiment140.com/for-students. 12.12.2013. (cited on page 52)

[Solomon and Stone, 2002] Solomon, R. C. and Stone, L. D. (2002). On "positive" and "negative" emotions. *Journal for the Theory of Social Behaviour*, 32(4):417–435. (cited on page 9)

[SQLite, 2013] SQLite (2013). SQLite. http://www.sqlite.org. 02.12.2013. (cited on page 30)

[Statisticbrain.com, 2013a] Statisticbrain.com (2013a). Facebook statistics. http://www.statisticbrain.com/facebook-statistics/. (cited on page 17)

[Statisticbrain.com, 2013b] Statisticbrain.com (2013b). Twitter statistics. http://www.statisticbrain.com/twitter-statistics/. (cited on page 17)

[Techcrunch.com, 2013] Techcrunch.com (2013). Google announces 1b total android activations. http://techcrunch.com/2013/09/03/google-announces-1b-total-android-activations-names-next-version-kitkat/. (cited on page 28)

[Tong, 2001] Tong, R. M. (2001). An operational system for detecting and tracking opinions in on-line discussion. In *Working Notes of the ACM SIGIR 2001 Workshop on Operational Text Classification*, volume 1, page 6. (cited on page 18)

[W3.org, 2013] W3.org (2013). The structure of wordnet. http://www.w3.org. http://www.w3.org/2001/sw/BestPractices/WNET/wordnet-sw-20040713.html. (cited on pages 14 and 20)

[Wager et al., 2003] Wager, T. D., Phan, K. L., Liberzon, I., and Taylor, S. F. (2003). Valence, gender, and lateralization of functional brain anatomy in emotion: a meta-analysis of findings from neuroimaging. *Neuroimage*, 19(3):513–531. (cited on page 9)

[WebKit, 2013] WebKit (2013). WebKit. http://www.webkit.org. 02.12.2013. (cited on page 30)

[Zimmermann et al., 2006] Zimmermann, P., Gomez, P., Danuser, B., and Schär, S. (2006). Extending usability: putting affect into the user-experience. *Proceedings of NordiCHI'06*, pages 27–32. (cited on page 7)

List of Figures

List of Tables

Listings