BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

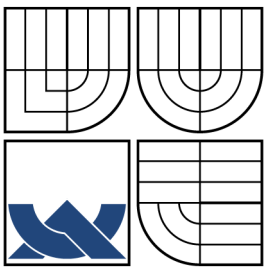
SOCIAL MEDIA ANALYSIS USING PATTERN RECOGNITION

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

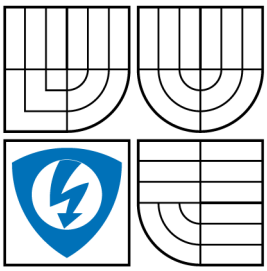
AUTHOR
AUTOR PRÁCE

Bc. VILIAM KRIŽAN

Brno 2015



BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

SOCIAL MEDIA ANALYSIS USING PATTERN RECOGNITION

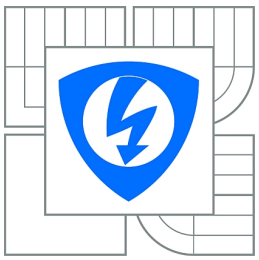
ANALÝZA SOCIÁLNÍCH SÍTÍ VYUŽITÍM METOD ROZPOZNÁNÍ VZORU

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE
Bc. VILIAM KRIŽAN

SUPERVISOR
VEDOUCÍ PRÁCE
Ing. HICHAM ATASSI

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Viliam Križan	***ID:***	134529
Ročník:	2	***Akademický rok:***	2014/2015

NÁZEV TÉMATU:

Analýza sociálních sítí využitím metod rozpoznání vzoru

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte současné metody rozpoznání emocí z textu. Zaměřte se na metody využívající algoritmy strojového učení. Navrhněte a v jazyce Python vytvořte nástroj, který umožní rozpoznat emocionální stav uživatelů ze zpráv sociální sítě Twitter v určité geografické lokalitě. Navržený nástroj by měl být vybaven vhodným grafickým rozhraním.

DOPORUČENÁ LITERATURA:

[1] DUDA, Richard O.; HART, Peter E.; STORK, David G. Pattern classification. John Wiley & Sons, 2012.

[2] FELDMAN, Ronen; SANGER, James (ed.). The text mining handbook: advanced approaches in analyzing unstructured data. Cambridge University Press, 2007.

[3] FRANCIS, Louise; FLYNN, Matt. Text mining handbook. In: Casualty Actuarial Society E-Forum, Spring 2010. 2010. p. 1.

Termín zadání:	9.2.2015	***Termín odevzdání:***	26.5.2015

Vedoucí práce: Ing. Hicham Atassi, Ph.D.
Konzultanti diplomové práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRACT

The diploma thesis deals with emotion recognition from texts on social media. The state-of-the-art methods of feature extraction, corpora and classifiers are described in the first section. Emotions are recognized by a classifier trained on annotated data from the microblog network Twitter. The advantage of using Twitter was the possibility to specify data collection to a certain geographical location. Geographical data allows to monitor emotional variations of population, for e.g. in different cities. The first task was to propose and develop a Baseline algorithm which classifies data to emotional classes. The classification accuracy is improved by employing a more complex SVM classifier. SVM classifiers, feature vectorizers and feature selectors are used from the Scikit library, which is written in Python. The data for classifier training were collected from the USA by the own developed mining application. The classifier are trained on data automatically annotated in the collection process. Two implementations of SVM classifiers are used. Final classified emotions that appear in different cities and in different time intervals are displayed as color markers on a map.

KEYWORDS

emotions, text recognition, emotional analysis, data mining, twitter, tweets, social media, Python, SVM, Scikit

ABSTRAKT

Diplomová práca sa zaoberá rozpoznávaním emócií z textu v sociálnych sieťach. Práca popisuje súčasné metódy extrakcie príznakov, používané lexikóny, korpusy a klasifikátory. Emócie boli rozpoznávané na základe klasifikátoru, netrénovaného na anotovaných dátach z mikroblogovacej siete Twitter. Výhodou použitia služby Twitter, bolo geografické vymedzenie dát, ktoré umožňuje sledovanie zmien emócií populácie v rôznych mestách. Prvým prístupom klasifikácie bolo vytvorenie Baseline algoritmu. Pre zlepšenie klasifikácie sme v druhom bode použili komplexnejší SVM klasifikátor. SVM klasifikátory, extrakcie a selekcie príznakov boli použité z dostupnej Python knižnice Scikit. Dáta pre natrénovanie klasifikátoru boli zhromažďované z oblasti USA, a to s pomocou vytvorenej aplikácie. Klasifikátor bol natrénovaný na dátach, označených pri ich zhromažďovaní – bez manuálnej anotácie. Boli použité dve rôzne implantácie SVM klasifikátorov. Výsledné klasifikované emócie, v rôznych mestách a dňoch, boli zobrazené v podobe farebných značiek na mape.

KLÍČOVÁ SLOVA

emoce, rozpoznávaní textu, analýza emócii, data mining, twitter, tweety, sociální sítě, Python, SVM, Scikit

KRIŽAN, Viliam *Social Media Analysis using Pattern Recognition*: master's thesis. Brno: Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Telecommunications, 2015. 67 p. Supervised by Ing. Hicham Atassi

DECLARATION

I declare that I have written my master's thesis on the theme of "Social Media Analysis using Pattern Recognition" independently, under the guidance of the master's thesis supervisor and using the technical literature and other sources of information which are all quoted in the thesis and detailed in the list of literature at the end of the thesis.

As the author of the master's thesis I furthermore declare that, as regards the creation of this master's thesis, I have not infringed any copyright. In particular, I have not unlawfully encroached on anyone's personal and/or ownership rights and I am fully aware of the consequences in the case of breaking Regulation § 11 and the following of the Copyright Act No 121/2000 Sb., and of the rights related to intellectual property right and changes in some Acts (Intellectual Property Act) and formulated in later regulations, inclusive of the possible consequences resulting from the provisions of Criminal Act No 40/2009 Sb., Section 2, Head VI, Part 4.

Brno ....

(author's signature)

ACKNOWLEDGEMENT

I am grateful to Hicham Atassi for frequent consultations and for valuable advices.

Brno . . . .

(author's signature)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

ACKNOWLEDGEMENT

Research described in this master's thesis has been implemented in the laboratories supported byt the SIX project; reg. no. CZ.1.05/2.1.00/03.0072, operational program Výzkum a vývoj pro inovace.

Brno . . . . . .

. . . . . .
(author's signature)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Emotions have strong impact on human's life. Emotions are expressed either by a facial expressions, gesture, voice, or they are reflected by a written text. Text can be in the form of document, correspondence list, short message, blog, or as a review of product or service. Nowadays, there is a huge amount of data available on the Internet. Social media are a good example of environments, where people share their life events or ideas. The posts in social media are usually represented by text, which can be analyzed by automatic systems.

I have exploited the well-known Social Network Twitter. Among the variety of Social Media on the Internet, the data from Twitter were uniquely represented as short messages known as tweets.

The major aim of this thesis was to create an application capable of classifying tweets into emotional classes in certain geographic locations. The motivation for this work was to observe emotional variations in predefined cities. The overview of emotional content of tweets can be used for monitoring of how the population reacts to certain situations. The reactions of population can dramatically change due to political, ethnic or natural inconsistencies. The emotional monitoring can also capture the population reaction based on a defined topic. Automatic systems may react on a detected emotional fluctuations in the geographic area, and find a reason.

In first part of the thesis I proposed and developed an application that implemented a Baseline algorithm for classifying tweets in a user defined location. Then, I figured out that this algorithm needed to be improved hence a more complex classifier, the Support Vector Machine classifier, was employed. Data for classification were collected by own proposed mining application. The classifier trained model was built and trained on collected tweets from the United States of America. Classification was done from tweets from several American cities. The statistical results of temporal emotional changes were presented in the own proposed visualizing application.

This thesis is organized as follows:

Chapter 1 describes corpora and lexicons used usually for emotion recognition and for sentiment analysis. Next, the state-of-the-art for feature extraction and selection, that encode the emotional state of text data, is presented. The chapter also presents various types of classifiers, used in text mining domain.

Chapter 2 introduces the Twitter APIs used for extracting data from Twitter. The Search API is also described in detail, including described parameters defining the search. Finally, the Streaming API which can be used for data mining is also presented.

Chapter 3 specifies technologies used for developing all own applications. In addition to Twitter API, this chapter describes the Google APIs used for geocoding and visualization, Scikit Learn and NLTK used for the classification, and GTK+ for developing the GUI. Basic usage and examples are also included.

Chapter 4 deals with the design and implementation of the Baseline application. The application is written in Python, and is equipped by a graphical user interface.

The application analyzes tweets and recognizes their emotional content within a user defined geographical location.

Chapter 5 discusses the acquisition of data from Twitter, which are used for the classifier training and evaluation. Moreover, a server application was developed to collect tweets using the Twitter API. Collected tweets were splitted into two groups: first group for classifier training, testing and validating, and the second group is left for final evaluation. The chapter also presents the collection scheme and the obtained statistical data of collected tweets.

Chapter 6 describes the steps of classifier training and testing. Two classifiers were trained, one for valence classification and the other for the emotional classification. The classifier pipeline uses components from the Scikit toolkit[24].

Finally, Chapter 7 reports results of experiments made on tweets, that have been collected in the mining process. A client application with a graphical interface was proposed and developed. The developed application classified the collected tweets into an emotional classes, and visualized the results as a statistical data on the map of USA.

1 STATE OF THE ART

Text mining is a significant area of machine learning. Texts are mostly written in natural language, in a contrast of computer or programming languages. That gives us the opportunity to take the advantage of the language rules and grammar, for making the content understandable by computers. Processing of natural language have variety of applications, such as question answering (IBM's Watson) or information extraction.

The most attractive area of text processing is sentiment analysis and emotion recognition. Sentiment is "a personal blief or judgment that is not founded on proof or certainty" [5]. Sentiment may express subjectivity, attitude, opinion, orientation, emotion, etc. Systems based on sentiment analysis can be used for a review summarization of products or services. For example, the classification of movie reviews to either positive or negative categories. One of the promising applications of sentiment analysis and recognition is public media evaluation, especially those from social media.

Human emotions are a key part of human interaction, which is also valid for to human computer interaction, where the machines have difficulty to understand the emotions. Most of the researchers have categorized the emotions into different classes. The six categories defined by Ekman [2] are the most known among them. These basic Ekman's emotions are *happiness*, *sadness*, *fear*, *anger*, *disgust* and *surprise*.

The later sections will describe available corpora, lexicons and techniques for feature extraction, feature selection and classification.

Fig. 1.1: Base Scheme of Text Recognition

1.1 Corpora and Lexicons

Having annotated data is crucial for any experiment with text mining. Corpus is made from the acquired data, that are labeled either by human or machine.

1.1.1 MPQA Corpora

The MPQAs are Multi-Perspective Question Answering corpora. The *MPQA Opinion Corpus* was created as a part of corpus annotation project by Wiebe, Wilson and Cardie [9]. The lexicon was constructed by manual annotation of a 10 000 sentences from the world news. Annotators classified a big amount of words and constituents, such as adjectives, modals, adverbs, verbs and nouns. Several types of sentiment and private states were labeled. The emotion labels were ambiguous and hard to distinguish from the other types of sentiment [1].

The *Subjectivity Lexicon* was built for sentiment analysis, analyzing the polarity of expressions. Lexicon contains a set of positive, negative and neutral words and phrases, which are tagged with their prior polarity. The corpus was constructed by annotating almost 16 thousand expressions from 425 documents. [11]

All MPQA resources are available at `http://mpqa.cs.pitt.edu`.

1.1.2 General Inquirer

General Inquirer (GI) is a Harvard's lexicon, which tags almost 12 thousand words into 182 categories. Categories includes: valence categories, syntactic and semantic markers, words of pleasures or pain, words reflecting the language of particular institutions, etc. The emotion-related tags can be used as features for identification in emotion recognition systems. More about the GI at `http://www.wjh.harvard.edu/~inquirer/`.

1.1.3 LIWC

Linguistic Inquiry and Word Count (LIWC) is a proprietary text analysis software. LWIC includes LWIWC2007 dictionary, which was composed of almost 4 500 words and word stems. Words are classified into categories, for example the word "cried" is classified into sadness, negative emotion, overall effect, verb and past tense verb. The results of LIWC classification are highly correlated with the General Inquirer's. For more information see `http://www.liwc.net`.

1.1.4 WordNet

WordNet is a lexical reference system, where nouns, verbs and adjectives are organized into synonym sets, called *synsets*. The synsets are interlinked by conceptual-semantic and lexical relations. Wordnet links the word forms, specific senses of words and the semantic relations. The most frequent synset relation is supersubordinate relation (hyperonymy). For example a subset of object: "bed" and "bunkbed".

WordNet Domains is a lexicon based on Princeton's WordNet[1], with an expansion of domain labels and multilingual support. Domain labels are a semantic way for categorization into areas of human knowledge, for example sports, finances, news, etc. Domains provides an natural way of semantic relations between word senses. Each synset is a member of at least one semantic domain.

WordNet-Affect is an extension of WordNet Domains, proposed by Strapparava and Valitutti [8]. Affective (emotional) labels were added to the lexicon, as a representation of emotional states: mood, emotional responses, attitude, or situation eliciting an emotions. Four extra labels were created as a support, besides the emotion tags, hence *positive*, *negative*, *ambiguous*, and *neutral*. The six basic (Ekman's [2]) emotions are part of the lexicon. The main purpose of the lexicon was to describe affective meanings in natural language. See the WordNet Domains homepage `http://wndomains.fbk.eu` for more details.

1.2 Feature Extraction

Text contains words, letters, punctuation or other symbols, and may be arranged into patterns. These characteristics are known features (or attributes). Features are a part of the learning and evaluation process, and they are used by classifiers for finding the differences between classes. Frequently used features by researchers are n-grams (incl. unigrams), patterns, punctuation characters, emoticons (especially in blogs and chats), uppercases and count features[2].

1.2.1 *n*-grams

n-grams are sequences of n items, such as words, letters, syllables, etc. The purpose of the n-grams is to compute the joint of sequence probabilities or probability of an upcoming item in the sequence. Simplest n-gram model is the unigram model, where n is 1. The next most used n-grams are bigrams, which are suitable for two-word phrases, including negative phrases, like "not good". The disadvantage of n-gram model is that most languages have long-distant dependencies. This drawback makes the n-gram model unsuitable for text data information extraction when using it as the only model.

Davidov et al. [5] purposed a binary n-gram features ($n \in 1$–5) with a weight, equal to inverted count of word/sequence in their corpus. The weight led to prioritizing the rare words/sequences over the common ones.

The n-grams might not be efficient for sentiment analysis. Unigrams can be supported by part-of-speech (POS) tags as meta features, like presented by Barbosa et al. [15]. Besides of the POS tags, they mapped the words to their prior subjectivity and polarity. The prior polarity is switched from positive to negative, or vice-versa, by the occurrence of negative word (not) or negative expression. They derived the subjectivity and polarity of the tag

[1] `http://wordnet.princeton.edu`

[2] Counted feature is considered as counts of a specific feature.

from the subjectivity lexicon. Go et al. [14], on the other hand, found the POS tags as improper features.

1.2.2 Punctuation and Emoticons

Punctuations are a part of sentences, and can be used as generic features. The exclamation marks "!" and question marks "?" can emphasize the textual emotional content. For example, the repetition of exclamation marks significantly indicates anger. The number of these punctuations in a sentence is a possible feature. Capital letters of words can increase their weight or express a certain emotion.

Emoticons (smileys) are textual representations of emotions. They are usually written with basic ASCII characters. For example :-) express happiness whereas :-(reflects sadness. Nowadays, ideograms, represented by pictograph, are getting more popular in electronic messages and social networks. These pictographs are called *emojis*, and are represented by a UTF-8 encoded character. Characters are rendered using a special font. As an analogy to the ASCII emoticons, the emojis ☺ and ☹ represent their emotions analogously to ASCII characters. Emoticons and emojis are considered as the most significant carriers of information in the textual content.

1.2.3 Pattern-based features

One of the pattern based feature is based on the arrangement of texts into *high-frequency words (HFW)* and *content words (CW)*. The pattern-based feature was presented by Davidov et al. [6] and implemented in [5]. They defined the HFW as words appearing more than T_H times, per million words, and CW as words appearing more than T_C times, per million words. T_H and T_C are thresholds, which were set to 1 000 words per million for T_C, and 100 words per million for T_H. The pattern is constructed by 2–6 HFWs and 1–5 CWs. Avoidance of capturing only a part of multi-word expression is done by constraining the pattern to begin and end with an HFW.

Class sequential rules (CSR) is the next approach of pattern-based features, proposed by Wen et al. [3]. CSR is a sentence-level pattern for the sentiment/emotion classification. Two emotion labels are obtained for each sentence, by lexicon-based and SVM-based methods. The derived CSR feature is subsequently used for SVM-based emotion classification. Sentences with conjunction words, such as "but" or "where", are separated before the feature extraction. These conjunction words are added into the sequence, since they reflect relationship between the sub-sentences. The same emotion labels, for one (sub)sentence collected by classifiers, are written as one into the sequence. The advantage of this approach is the usage of several emotional labels. An example of a CSR from sentence with conjunction word:

$$< \{none, sadness\}\{sadness\}\{but\}\{happiness\} > . \tag{1.1}$$

1.3 Feature Selection/Reduction

Extraction of all possible features from the training set is not an effective approach, since it can cause overfitting or misclassification due to the inclusion of redundant or irrelevant features. To solve these issues, the number of input features should be reduced by either selecting a certain number of them or by transforming the whole feature space to a new space with smaller number of dimensions. Regarding text data, the feature selection is performed by applying a set of rules as it is reported in the following paragraphs.

Davidov et al. [5] defined an 0.5 % rule. All words (unigrams) and n-grams, appearing in less than 0.5 % of the training set, were discarded.

Go et al. [14] took the advantage of Twitter syntax. They replaced the Twitter's special properties by tokens. Usernames, which starts with a @ (at) symbol, were replaced by `USERNAME` token. URL links were converted into `URL` tokens. Repeated letters, which often occurs in Twitter messages, were shrunk to a maximum of two occurrences. For example the word "huuuuungry", was replaced by "huugry" hence still preserving the stronger emphasis (two "u"s). This type of feature selection reduced the set to 54 %, from the original feature set.

1.4 Text Classification

Classification, as the last step of the pattern recognition process, assigns the input feature vector to a certain class. There are several algorithms for classification, the most known among them for emotion recognition are the *Naive Bayes*, *Maximum Entropy*, *Support Vector Machines (SVM)* and *k-nearest neighbors*.

1.4.1 Naive Bayes

The Naive Bayes classification method is based on Bayes rule, and relies on a simple representation of document, called *bag of words*. The aim of this method is to choose the class with a highest probability. This can be formalized as follows:

$$c_{NB} = \arg\max_{c_j \in C} P(c_j) \prod_{i \in positions} P(w_i|c_j), \tag{1.2}$$

where c_j represents the class from all C classes and w_i represents the word on position i. The $P(c_j)$ is the prior probability of the class c_j and $P(w_i|c_j)$ is the probability of word w_i for the given class c_j.

Aman [1] in his research used the Naive Bayes classification for emotion recognition. The accuracy of the emotion/non-emotion classification with ten-fold cross validation was 72.08 %, combining the General Inquirer (section 1.1.2) and WordNet-Affect (section 1.1.4) lexicons altogether and separately. It is worth mentioning that non-lexical features increased the overall accuracy.

Go et al. [14] in the Twitter sentiment analysis obtained the best accuracy of Naive Bayes using unigrams and bigrams as features. Sentiment was represented as positive, negative or neutral. The resulting accuracy was 82.7 %.

1.4.2 *k*-nearest neighbors

Davidov et al. [5] used an k-nearest neighbors (kNN) method for classification. In their approach, they constructed a feature vector V. For each vector the Euclidean distance was computed between the trained and the testing vectors. The constant k was set to 10.

1.4.3 MaxEnt

Maximum Entropy (MaxEnt) is a feature-based model, that uses search-based optimization to find weights for the features that maximize the likelihood of the training data. The MaxEnt model does not make an independence assumption of features, like in the Naive Bayes, and therefore overlapping features have no effect on the classification.

Go et al. [14] used Standford Classifier for preforming MaxEnt classification. They used conjugate gradient ascent for the training weights. The reached accuracy was 83.0 %, using unigrams and bigrams – higher accuracy comparing to the Naive Bayes.

1.4.4 Support Vector Machines

Support Vector Machines (SVMs) preprocesses data in high dimensions, much higher than the original feature space. The main idea behind the SVMs, is to find a separating hyperplane, with the largest margin in the learning process. SVMs are defined by vectors of points closest to the separating hyperplane. The best hyperplane is defined by the largest margin between two classes. An example of linear separation can be seen on Fig. 1.2. [16]

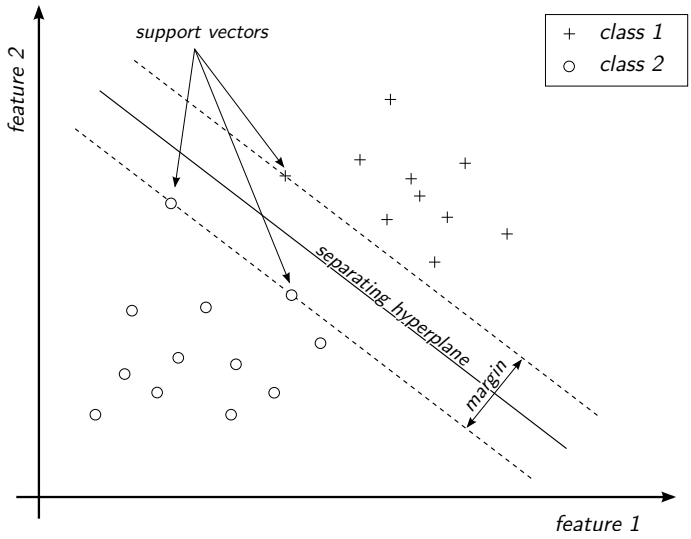


Fig. 1.2: SVM linear separation

The SVMs can only separate two classes. There are several methods, that overcomes this drawback, such as "one-against-one" (one-vs-one) approach and "one-against-all" (one-vs-rest) approach. The "one-against-one" approach creates $C_2(n)$ two-class combinations (n is the number of classes) of SVMs, comparing each class to the other. The "one-against-all" approach compares each class against the concatenation of the rest classes.

SVM have shown a slightly better results over Naive Bayes in emotion recognition presented by Aman [1]. The accuracy of SVM reached 73.89 %, where the non-lexical features had no affect on the result.

Wen et al. [3] compared the use of three SVMs on classifying a Chinese microblog texts, for both sentence-level and document-level emotion classification.

Go et al. [14] used an SVM with linear kernel. Sentiment got the best accuracy using only unigrams as features.

Milgram et al. [13], compared the two multiclass approaches: "one-against-one" and "one-against-all". The "one-against-all" strategy was more accurate for classification with fewer classes.

2 TWITTER AND APIS

Twitter is a microblog site, where users share short messages known as tweets. Tweets are limited to 140 characters, and may contain various characters. Character `@`, followed by a username, is used to refer, include, or address the message to a certain user, for e.g. `@stanfordnlp`. The target user may be notified about the tweet. Referring to the particular user is the prime key of direct (private) messages. The `#` (hash) character has a role of tagging the message with a certain keyword(s). A tweet may contain couple of *hashtags*. Hashtags are good for other users, who are following or searching for a defined topic or subject, for e.g. `#finances`.

Twitter posts may be retweeted (reposted) by other users to share the information to their present or future followers. Retweeted (reposted) tweets are indicated by the retweet keyword "RT" at the beginning of message. Due to the length limitation, the tweets often have URLs, which may point to the entire article or story. URLs are also suitable for linking to a picture or a short video.

Twitter introduces different types of APIs to read and write Twitter data (tweets). The Twitter data are accessible through the Streaming APIs or the REST APIs. REST APIs provide an access in query-response context. Streaming APIs allows to continuously deliver tweets in real-time using a persistent connection. It should be mentioned here that authentication is required in order to receive or edit Twitter data.

2.1 Authentication and Authorization

The own application have to be registered, in order to use the Twitter APIs. Application registration form is available on `https://apps.twitter.com`, where basic application details as name, description and website are filled.[1] After the registration, two keys (tokens) are generated for authentication: *Consumer Key* and *Consumer Secret*.

Twitter APIs uses OAuth[2] mechanism to authenticate the application. Two types of authentication are available: *Application–user authentication* and *Application-only authentication*.

Application–user authentication uses access tokens (besides of consumer tokens), specified for each twitter user. Access tokens are generated in the user authorization. The user has to be signed in on Twitter, to confirm the authorization for the requested access level, depending on API's endpoint (connection). If the application wants to make requests on user's behalf, the user has to grant the permission for this action. The user can manage the authorized applications in Twitter application management, for e.g. to remove the application grants.

Application-only authentication is used for making an API requests on the application level, without a user context. This type of authentication does not approve the application

[1] Twitter account is required for registering a new application.
[2] `http://oauth.net`

to manipulate with twitter data, or to read private user's informations, incl. private tweets, emails, etc. Not all Twitter APIs support the Application-only authentication (for e.g. the Streaming APIs does not).

More about the Twitter OAuth authentication and authorization is available in [17].

2.2 Twitter REST APIs

Twitter REST APIs (version 1.1) provides RESTful accessibility to the Twitter data. For e.g., reading tweets, looking into the user's profiles, creating new tweets, etc. Actions, which are requested to be processed or done, are defined by API endpoints. An application connects to the certain endpoint, to get (GET) or write (POST) data, using the request-response protocol (HTTP). All endpoints requires an OAuth authentication to be built upon a request. The response from the server is encoded in JSON format.

2.2.1 Search API

The search endpoint `search/tweets` is available for searching tweets from Twitter domain. The Search API is focused on relevance, rather than completeness. Note that not all tweets are indexed by the Twitter. The completeness of tweets is the main feature of Streaming API.

The Search API offers various operators in search query, to clearly specify results that should be retrieved. The default operator is AND, which is defined by spaces in-between the words. The OR operator is declared by putting the `OR` keyword, surrounded by spaces between the two words. The NOT operator is represented by a minus character, perpending the word that should be excluded from the search results.

Search can be also refined by other additional parameters. One of the parameters is the Geolocalization (`geocode`), which gives the ability to restrict tweets by a given location. The position is specified by latitude, longitude and radius, where the tweets have been constructed at the time of writing. A tweet location is determined by the GPS locators, available in mobile handhelds, or by estimation, using reverse geocoding of the user's profile. Second significant parameter is the result type (`result_type`), where the tweets are narrowed into `popular`, `recent` or `mixed`. Other additional parameters, as for iteration, specification of language or a number of tweets are also available.

2.2.2 Limitations

REST APIs are subjected to the rate limitation. The limitation is based on a number of requests that are sent to the Twitter. For the user authentication, the restriction is considered on a per-user basis and for each endpoint separately. There is a 15 minute window, where maximum of requests can be sent, depending on the endpoint. Most of the endpoints have maximum of 15 requests in the window, again for each user (access token) independently. When using the application-only authentication, the rate limitation

is restricted to the whole application globally. That means, that the request restriction in a window is considered per application only, and not for per user, due to absence of user authorization.

The search endpoint (`search/tweets`) limitation is 180 requests for user-based authentication and 450 requests for app-based authentication. The maximum number of tweets per request is 100. See the Rate Limits Chart in [18] for full limit description.

2.3 Twitter Streaming APIs

The Streaming APIs are based on persistent connection to the providing server. Streaming APIs offer low latency access to Twitter's global stream. The data are pushed in real time and in a single connection. APIs provide three types of streams:

- Public streams – suitable for data mining or following certain user or topic,
- User streams – suitable for stream and events of one authorized user,
- Site streams – suitable for receiving real-time data of multiple authorized users.

Disadvantage of using the Twitter Streaming API is the requirement of *Application–user authentication*. A Twitter user need to be specified in order to get the Stream data. Each user may get a different set of data, based on its behavior on Twitter (followers, statuses, searches), even when the same parameters for Streaming API are used.

2.3.1 Public Streams

The public REST APIs, as presented above, offered to read and search the tweets in Twitter. Public Streams provide the same thing, except the tweets are received in real time.

The version 1.1 of APIs implements the `statuses/filter` endpoint, which allows the application to set the filter of receiving tweets. Filter is specified by (at least one) parameter.

3 TECHNOLOGIES

The implementation which is be presented in Section 4, uses various technologies. Python 3 is used as a base programming language. The library modules and APIs are presented in this chapter.

3.1 Twitter API

The Twitter's REST and Streaming APIs (described in Chapter 2) are handled by Python's TwitterAPI module[1]. The connection to API requires a `CONSUMER_KEY` and a `CONSUMER_SECRET`, when using an Application-only authentication. For Application-user authentication, the `ACCESS_TOKENT_KEY` and `ACCESS_TOKEN_SECRET`, have to be defined additionally.

Requests are made by `request()` method, defining the endpoint and the addressed data. The data are either GET data, for requesting, or POST data, for posting onto the Twitter. The next example shows a basic search request of tweets, containing the word "pizza". Note that the `auth_type='oAuth2'` sings the usage of Application-only authentication.

```
from TwitterAPI import TwitterAPI
api = TwitterAPI(CONSUMER_KEY, CONSUMER_SECRET,
                 auth_type='oAuth2')
r = api.request('search/tweets', {'q':'pizza'})
```

3.2 Google APIs

The geographic location, represented in numerical form, is hard to obtain and remember by humans. People are used to write the addresses or names of locations or cities. For this purpose, the *Google Geocoding API* is available. *Google Map APIs* are a good candidate for visualization of the selected location or area.

3.2.1 Geocoding API

The Geocoding API[2] is used for converting an address (or name of the location) into the geographic coordinates, latitude and longitude. The reverse geocoding, for translating the geographic coordinates into the human-readable address, is also available.

The geographic code is retrieved by making a HTTP request, to the constructed URL `https://maps.googleapis.com/maps/api/geocode/json?address=query`.
The `query` is considered as desired address. The output is received in JSON format, where all possible results are presented. Exact geographic coordinates, of the location, are found in the `location` key under the `geomerty`.

[1] Available at `https://github.com/geduldig/TwitterAPI`
[2] Documentation at `https://developers.google.com/maps/documentation/geocoding`

Limit of the Google Geocoding API (for free) is maximum of 2 500 requests per 24-hours, or maximum of 5 requests per second. This limitation is restricted for each API key or for each IP address (when not using an API key) separately.

3.2.2 Static Maps API

Google provides Map APIs for presentation of the world map, or a selected area. The API is intended for inclusion on the web pages or mobile applications. The *Static Maps API v2*[3] generates a static map image of a defined location, zoom and/or size. To request the static map the address `https://maps.googleapis.com/maps/api/staticmap` is used. The URL address requires some parameters to be declared. The basic available parameters are:

- `center` – for specifying the exact geographic coordinates or address,
- `zoom` – for defining the zoom level of the map (number from 0 to 21),
- `size` – which defines the dimension of a produced image in pixels (for eg. 450x300)
- `maptype` – which specifies the type of map to be constructed (`roadmap`, `satellite`, `hybrid`, or `terrain`)

Limitations of Static Maps API are 1 000 static map requests per IP address per 24 hour period, and 50 requests per IP address per minute. When using the authentication, by defining the Google API token key with the request, the limitation is 25 000 images per 24-hours for one token key.

3.2.3 Usage

The HTTP requests, for accessing and receiving the Google API data, can be made by using a `urllib.request` module. `urllib` is a standard Python library. The request is made by calling the `urllib.request.urlopen()` method with the specified URL. The GET data, included in the URL, should be properly encoded. Encoding is done by the `urllib.parse.urlencode()` method.

Example of usage:

```
import urllib.request
url = 'https://maps.googleapis.com/maps/api/geocode/json?'
url += urllib.parse.urlencode({'address': 'london'}) # GET data
response = urllib.request.urlopen(url)
rawdata = response.read()
text = rawdata.decode('utf-8')
```

3.3 GTK+

GTK+ (GIMP toolkit)[4] is a cross-platform library for creating an GUI application. GTK+ supports basic widgets, such as Windows, Boxes, Buttons, Text inputs, etc. The Python's bindings for the GTK+ can be found in the module `gi.repository`.

[3] Documentation at `https://developers.google.com/maps/documentation/staticmaps/index`
[4] `http://www.gtk.org/`

The GUI can be constructed in WYSIWYG Glade editor, as an XML `.glade` file. The glade file is loaded by the GTK+, where each widget is identified and referenced by its ID.

3.4 Scikit Learn

Scikit Learn is an open source set of tools for Machine Learning written in Python. Tools are supported by different low-level C/C++ libraries, such as libSVM or liblinear. Scikit includes algorithms for supervised and unsupervised learning. Mechanisms included in the bundle are Classification, Regression, Clustering, Model selection, Dimensionality reduction and methods for Data preprocessing. Each estimator implements a `fit` method for training and `predict` method for predicting a class or a value in case of regression tasks. Some transformer estimators use `transform` method for transforming the data or reducing the dimensionality, for e.g. in Feature Selection. [23]

3.5 Natural Language Toolkit

Natural Language Toolkit (NLTK) is a Python based library for working with human language data. NLTK provides interfaces to corpora and lexical resources. Toolkit includes methods for text tokenization, stemming, parsing, semantic reasoning and tagging.

Available corporas are WordNet, The Penn Treeban Corpus, The Inaugural Address Corpus, and many more. Each corpus or dictionary has to be downloaded using a `nltk.download()` interactive command in Python's interpreter.

4 BASELINE APPROACH

The first aim of this thesis was to propose and develop a baseline approach which classifies tweets within a certain geographical location according to their emotional content. Tweets were collected by the Twitter REST API, using the search mechanism presented in Section 2.2. The application returns a statistical overview of tweets emotions within the selected area. Note that only English tweets are processed.

4.1 Baseline Algorithm

The task of Baseline Algorithm is to classify tweets into 6 basic emotional states (Ekman [2]; Chapter 1) and into mixed emotion or no emotion. These emotional categories are labeled by two character labels, as shown in Table 4.1. Unigrams are used as an emotion indicators, including words and emoticons. These indicators are written into a CSV file, as a database. Lexicon database is built by a few words (collected from Aman's thesis [1]) and most frequent emoticons.

Tab. 4.1: Labels of Emotion Categories [1]

Emotion Category	**Label**
Happiness	hp
Sadness	sd
Anger	ag
Disgust	dg
Surprise	sp
Fear	fr
Mixed emotion	me
No emotion	ne

Regular expressions are created for each emotion category, consisting of unigrams loaded from the database. The regular expressions are composed to match at the word or emoticon borders, as shown in Figure 4.1. Start of an unigram is considered at a word border (`\b`) or at start of a hashtag (`#`). End of the unigram is matched at the border or before an non-word character (`\W`). Unigrams, of one emotional label, are escaped and concatenated by a *or* (pipe, `|`) character. Note that `(?:)` is a non-capturing group sequence, used only for matching alternatives.

`(?:\b|#)(`*`joined-unigrams`*`)(?:\b|\W)`

Fig. 4.1: Regular Expression Composition

Feature extraction is done by applying the regular expressions, for each emotional category of one tweet. The expression matches the emotional indicators. The classification is evaluated by counting the occurrence of indicators, and returning the corresponding emotion label. Only one label is returned per tweet. If there are no indicators in each

category, the result is *no emotion* (`ne`). When two or more emotion labels occurs in one tweet message, the *mixed emotion* (`me`) is considered.

Module `Baseline` was developed, for the purpose of emotion classification. Module loads the emotion indicators, compiles the regular expressions, and classifies the presented data. Tweets are classified by calling an `Baseline.process()` method. The iterable object (list) of tweets is given as an argument for the method. The method returns a counts of emotional labels, for a statistical calculations.

4.2 Application

The application is designed (Fig. 4.2) to take an user input, by typing the address of desired location. Location is translated into geographic coordinates by Google Geocoding (Section 3.2.1). Coordinates are used in Twitter Search API (Section 2.2.1), to define the location of tweets in specified radius[1].

Classification, of founded tweets, is done by the Baseline algorithm. The result of classification is represented by pie chart, where each color represents its own emotional category. The map is downloaded by Google Static Maps (Section 3.2.2) as a temporary PNG image, to show the area of interest.

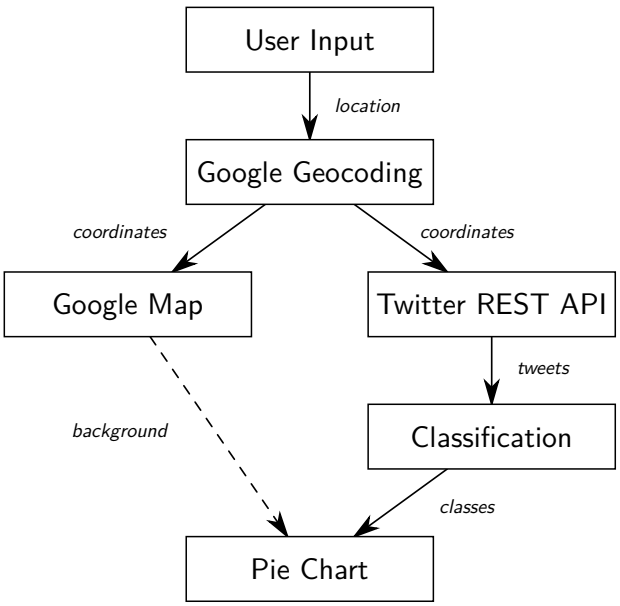


Fig. 4.2: Baseline Application Flow Design

Gnome's GTK+ was used for building the graphical user interface (GUI) in the application. The skeleton of GUI is shown on Fig. 4.3. *Menu bar* have a function of controlling the interface and showing the About dialog. *Search entry* have a purpose of user inputs, to type the location's address. After the entry activation (by hitting the Enter), the application flow starts to begin (Fig. 4.2). If the input is a known address and geographic

[1] Radius was set to 15 km.

coordinates are found, the tweets are downloaded and analyzed. All progress is shown in the *Status bar*. The resulting pie chart is rendered as an overlay on a map. The percentage of emotion labels (emotions) are written in the Status bar.

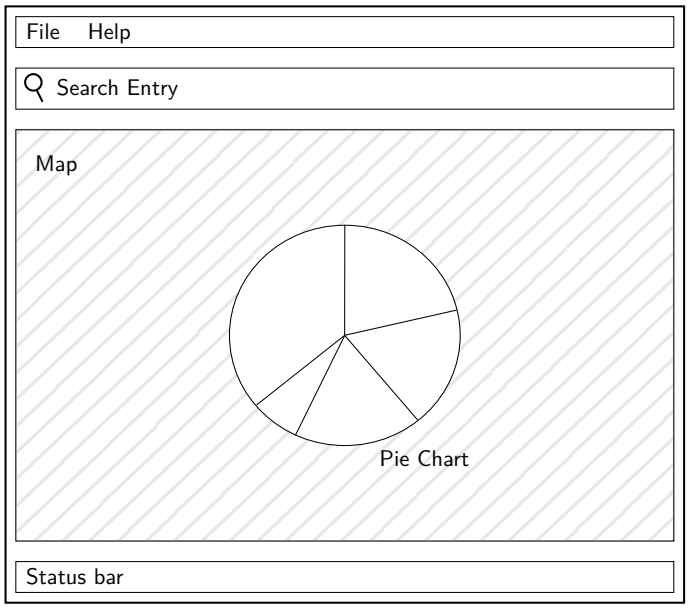


Fig. 4.3: Baseline Graphical User Interface Design

An example of usage and results are illustrated on Fig. 4.4. “London” was used as an input of search entry. 50 tweets were analyzed, resulting 19 % of happiness, 5 % of sadness, 7 % of mixed emotion, and rest 69 % of no emotion.

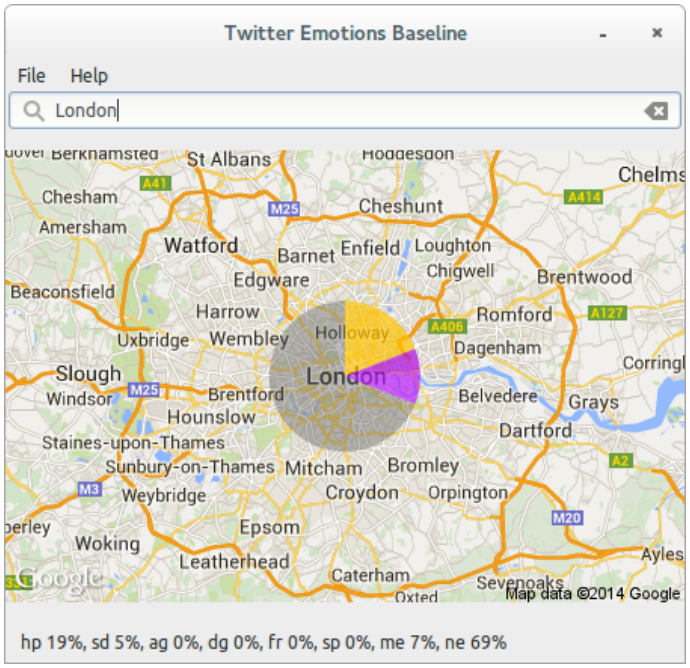


Fig. 4.4: Screenshot of the Baseline Application

5 DATA ACQUISITION

This chapter focuses on data collection from Twitter. Data were selected using a search mechanism with specified search terms. Each tweet found by the search term is annotated with labels, that the search term have assigned. The objective was to target on English speaking countries, namely on the United States of America. The application proposed for data collection and annotation is described below.

5.1 Mining Application

Application *Twitter Miner* was developed for the purpose of tweets collection, and therefore to build the Twitter corpus. Data from Twitter are collected using the Twitter REST API (as presented in Section 2.2). Selection of data is provided by the Search API, where GEO location option is used. Tweets are saved into two groups. The first group of tweets is for training, testing, validating the classifier. The second group of tweets is for final evaluation using a trained classifier. Data for training are gathered and annotated using a search terms. Search terms have assigned an emotional label and/or valence label, that is added to the collected tweet. Data designated for final evaluation are collected in the same way as data for training. Filters may be defined in a place of a search term, for e.g. to filter only non-retweeted (reposted) tweets (`-filter:retweets`). Final evaluation is based on the trained model, where each tweet is classified to an emotional class. The classified data can be visualized on map, to show a emotional changes in time in different cities.

Tweets are stored in a relational (SQL) database, specifically to the `tweets` table. The `tweets` table consists of tweet ID, text, user ID, number or retweets (reposts), flags and other columns that are essential for the mining process. Each tweet may have couple of labels, which are saved into the `tweet_labels` table. Each label have an attribute, that indicates an used classifier, from which the label was classified or annotated.[1] The indicator with the value of "search" express, that the label was annotated by using one of the search terms, from a Table 5.1.

The database have two other tables, namely `locations` and `search_terms`, which are used for specifying the data collection. The `locations` table stores a geographic locations, including a location name, geographic code, radius, and a type of mining group. The mining group tells whether data collected from the location are intended for training (testing and validation) or for final evaluation. The second table `search_terms` holds the search queries, that are used to build a corpus. The full specification of table structures can be found in Appendix A. The connection and the table collation for text storage was set to `utf8mb4` (in MySQL), for the purpose of storing the 4 Byte UTF-8 characters, such as emojis.

Twitter Miner application is executed by `miner.py`, using a Python 3 interpreter. The application has several options:

- `-c COUNT` – to set the maximum number of tweets per request,

[1] A label and class are interchangeable.

- `-i INTERVAL` – to set the time interval between a set of requests (in seconds),
- `-g train/classify` – to specify the data collection for training or for classification
- `--classify` – to use a trained emotional classifier and classify collected tweets.

Three main Python classes were developed for the mining application, namely the `Database`, `TwitterAPI`, and `MinerBase`. The `Database` class handles a connection to the database and provides methods for fetching and storing tweets, locations and search terms. The `TwitterAPI` is a class from the external library, which was described in Section 3.1.

The main class of the application is `MinerBase`. The `MinerBase` handles several *subminers*, where each *subminer* (`TweetMiner` class) have its own geographic location coordinates and search term (only in training) parameters. The parameters for a *subminer* are used to build a REST request query for the Twitter API. Note that only one instance of `TwitterAPI` is needed, and therefore it is shared among all "subminers". The miner uses the *subminers* to pool Twitter for new tweets in defined intervals.

The subminer stores an indicator of location (`location_id`) and an indicator of used search term (`search_`category). The indicators help the subminers to find a maximum tweet ID, for next requests – for each location and search terms separately. The maximum tweet ID is used to iterate over a timeline, that Twitter responds with.

Getting data for training is done by iterating over a search timeline and setting an initial cursor. Tweets are collected based on their IDs, starting from the cursor to the least non-collected tweet. The cursor is set to the min. tweet ID from the tweets set after each request, and therefore to continue the iteration (from top to bottom). The cursor is reset after all tweets are collected in the timeline iteration. Resetting cursors causes the next pooling to start the iteration from the very top, and therefore to get newest data. The iteration is done by setting `max_id` and `since_id` as the request parameters. The `max_id` parameter is the cursor with a value of last tweet ID from previous request minus one (preventing the last tweet to be fetched again). The `since_id` parameter is set to a max tweet ID, that is stored in the database for that location and search term.

Collection of tweets for final evaluation uses a principle of getting a newest Twitter data for each request – newest as the newest submitted tweets. The `since_id` is used to set the max tweet ID from the last request or from the database, preventing to fetching a duplicate data. For more information about the iteration process, please refer to *Working with Timelines* in [18].

Twitter Miner application can be started also by executing the `runminer.sh` script, which will ensure a non-stop running (restarts the miner after a failure).

5.2 Collecting Data from Twitter

Data collection from Twitter was performed within the United States. The option `lang` was set to `en` in the requests, to ensure that all tweets were written only in English only.

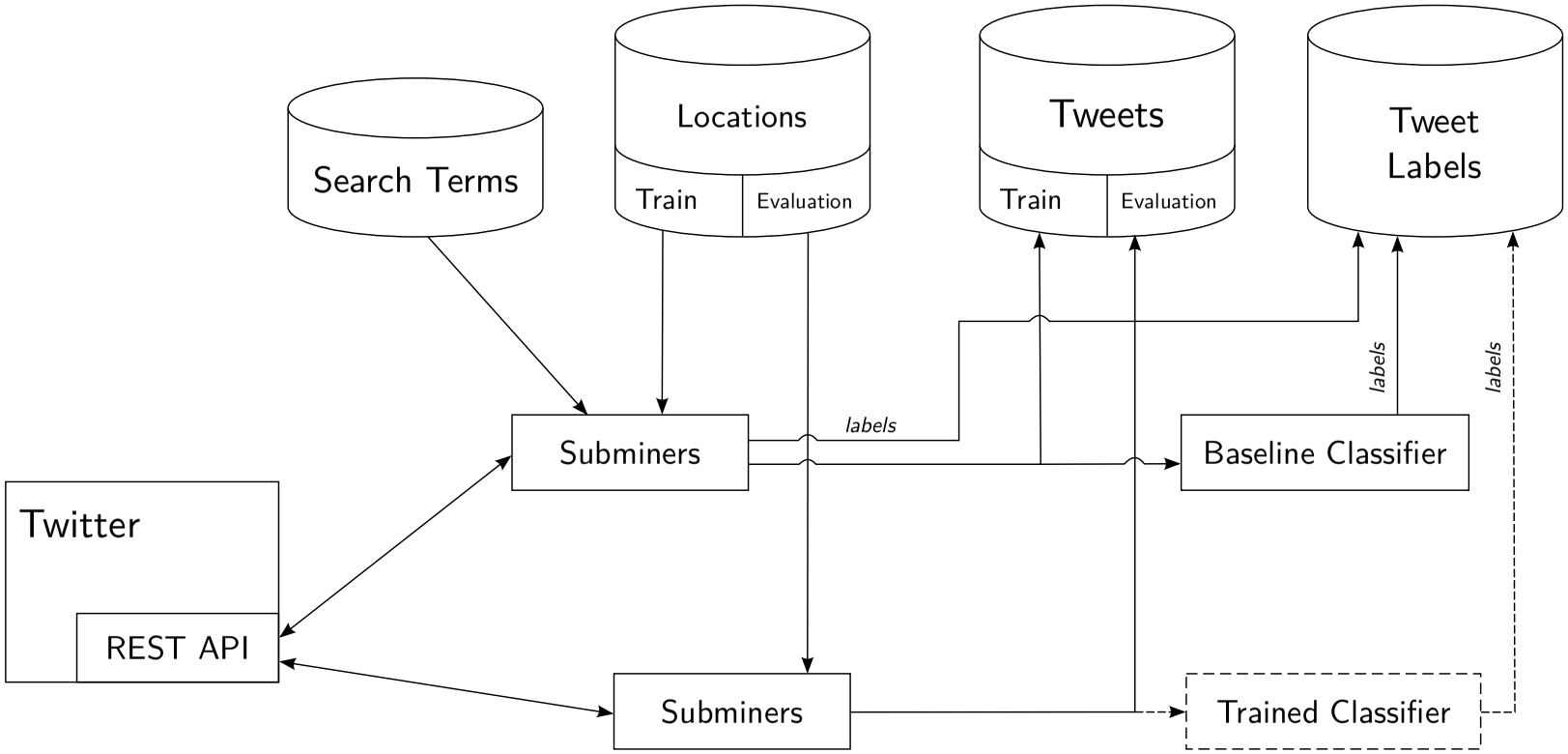


Fig. 5.1: Data collection flow

5.2.1 Data of Training Corpus

The amount of data flowing on Twitter takes a grater amount of time to be properly labeled by human annotators. Using search terms, such as hashtags [21] and emoticons, overcome the need of manual annotation. The search terms were mapped to proper labels, as seen on Table 5.1. The positive (pos) and the negative (neg) labels were added to support emotional labels. Hashtags were selected to represent an emotional states with mainly used words (unigrams), from the Baseline approach, using the same principles as in [22].

Tab. 5.1: Search terms with appropriate labels added

Search terms	**Labels added**
`#angry OR #annoyed OR #pissed`	ag, neg
`#amazing`	sp, pos
`#crap`	dg, neg
`#disgusted`	dg, neg
`#frustrated`	fr, neg
`#happy OR #awesome`	hp, pos
`#sad OR #lonely OR #unhappy`	sd, neg
`#surprised OR #suddenly`	sp
`:)`	pos
`:(`	neg

Each collected tweet was classified and annotated with the Baseline algorithm, indicated by "baseline" string value in the labels table. This approach was done to find emotional states even in tweets annotated only to valence labels (pos/neg) in search process.

Data collection started on 3rd of March 2015, targeting on 30 US cities. Focusing on areas with about 12 km radiuses and using a not frequently used hashtags, led to a insufficient amount of data for emotional annotation and for the learning process. Around 1 200 tweets were collected per day, where the *disgust* and *fear* were presented only in few tweets (1 to 10 per day). The area of data collection was extended to the whole USA (excl. Hawaii and Alaska), resulting in more data. The final areas of Twitter data collection can be seen on Figure 5.2.

Tweets were finally acquired from a 6 overlaying areas, covering most of the land of United States. Uniqueness of tweet identifiers ensured storing non-duplicated data. Total of 222 480 tweets were collected, intended for building the Twitter Corpora. The emotional label *fear* has been annotated only on about 800 tweets. The low number of tweets indicating the *fear* emotion was insufficient for training, and therefore this label was not considered.

The positive valence was found in 158 316 tweets and the negative valence in 64 827 tweets. The emotional labels classified by the Baseline algorithm and assigned to valence category (in search process) increased the total number of emotional labels 10 times. The final count of labels, combined with the baseline emotional annotations, for training the model

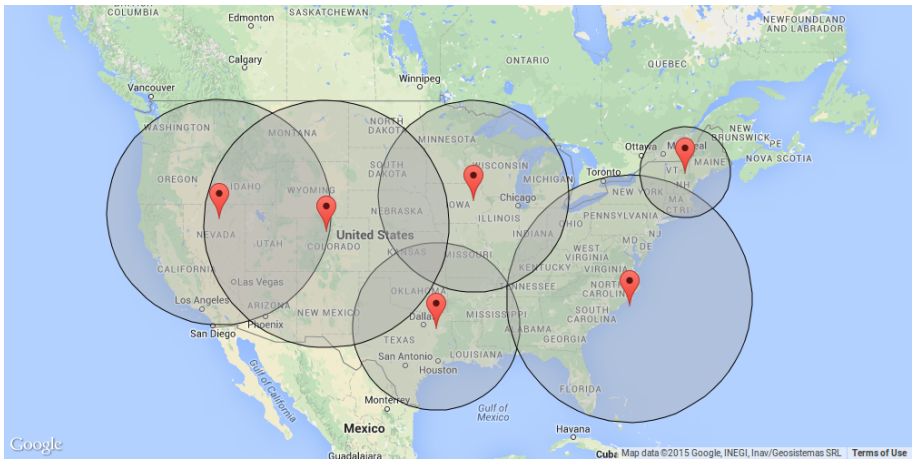
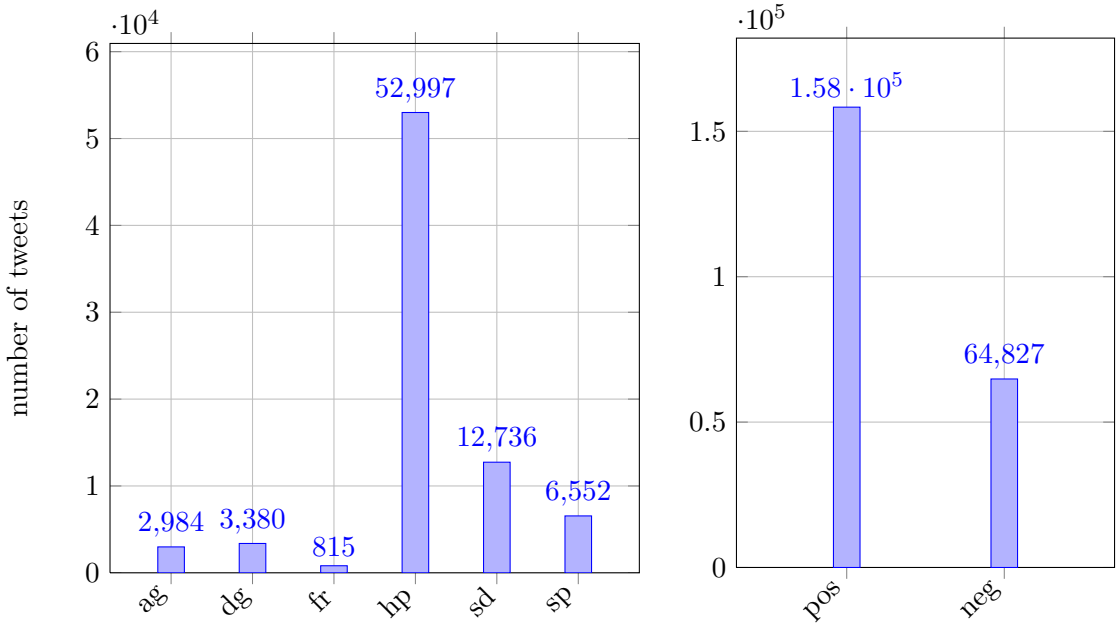


Fig. 5.2: Areas of Twitter data collection

can be seen on Figure 5.3.



(a) emotional labels

(b) polarity labels

Fig. 5.3: Collected tweets by labels

5.2.2 Data for Final Evaluation

Collection of tweets for the final evaluation was done to show changes of emotional content within predefined geographical areas. Collected data in this step will be subsequently classified by the classifier trained on data from Twitter corpus.

Tweets for final evaluation were collected from 32 cities. Only tweets that were not retweeted (reposted) and were not replies to other tweets, were selected. The filtering was performed using Twitter in-query filter flags, as seen on Figure 5.4. The map (Fig. 5.5) shows the geographical locations and radiuses where the tweets were collected. Radiuses vary from 5 km to 23 km. The cities were selected based on their population or another special attributes, such as criminality or unemployment. The full list of the cities can be found in the table of Appendix B.

```
-filter:retweets AND -filter:replies
```

Fig. 5.4: Twitter in-query filtering flags

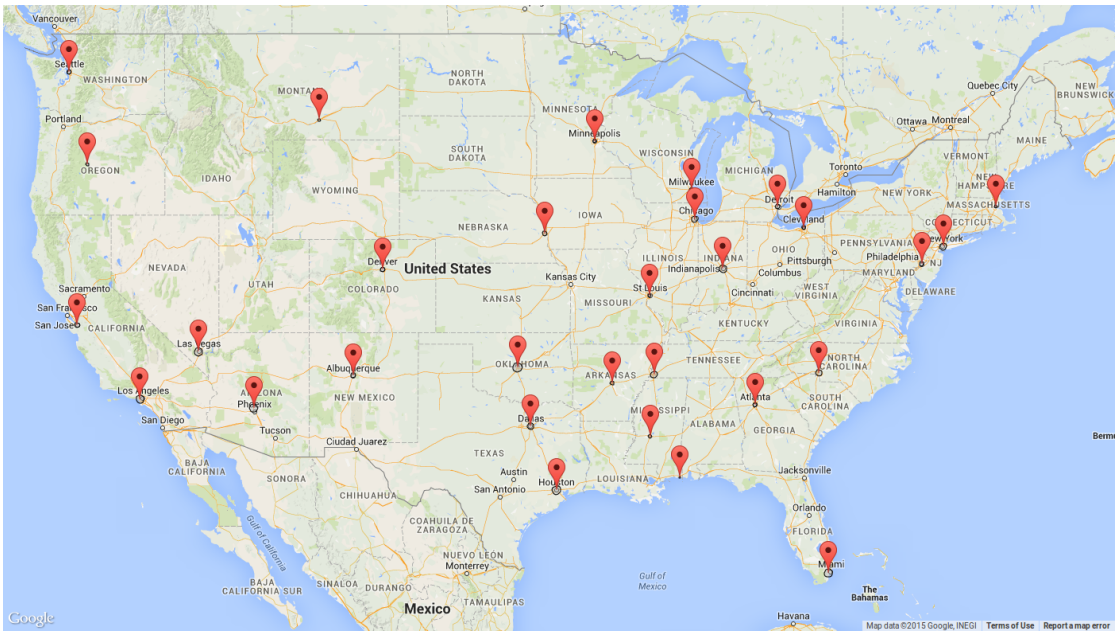


Fig. 5.5: Area of Twitter data collection for evaluation

The mining application was set to collect the tweets regularly by a time interval of one hour. The application is executed as follows:

```
runminer.sh -g classify -i 3600 -c 40
```

The tweets collection, for final evaluation, started on 26th of April 2015 and continued until 22nd of May 2015. The collection rate was 40 tweets per hour per city, resulting of max. 960 tweets per day for a city and max. of 30 720 total tweets per day. Figure 5.6 shows the number of tweets collected for each day, ordered by a creation date. Note, that UTC time was considered. On average 19 497 tweets were collected per day.

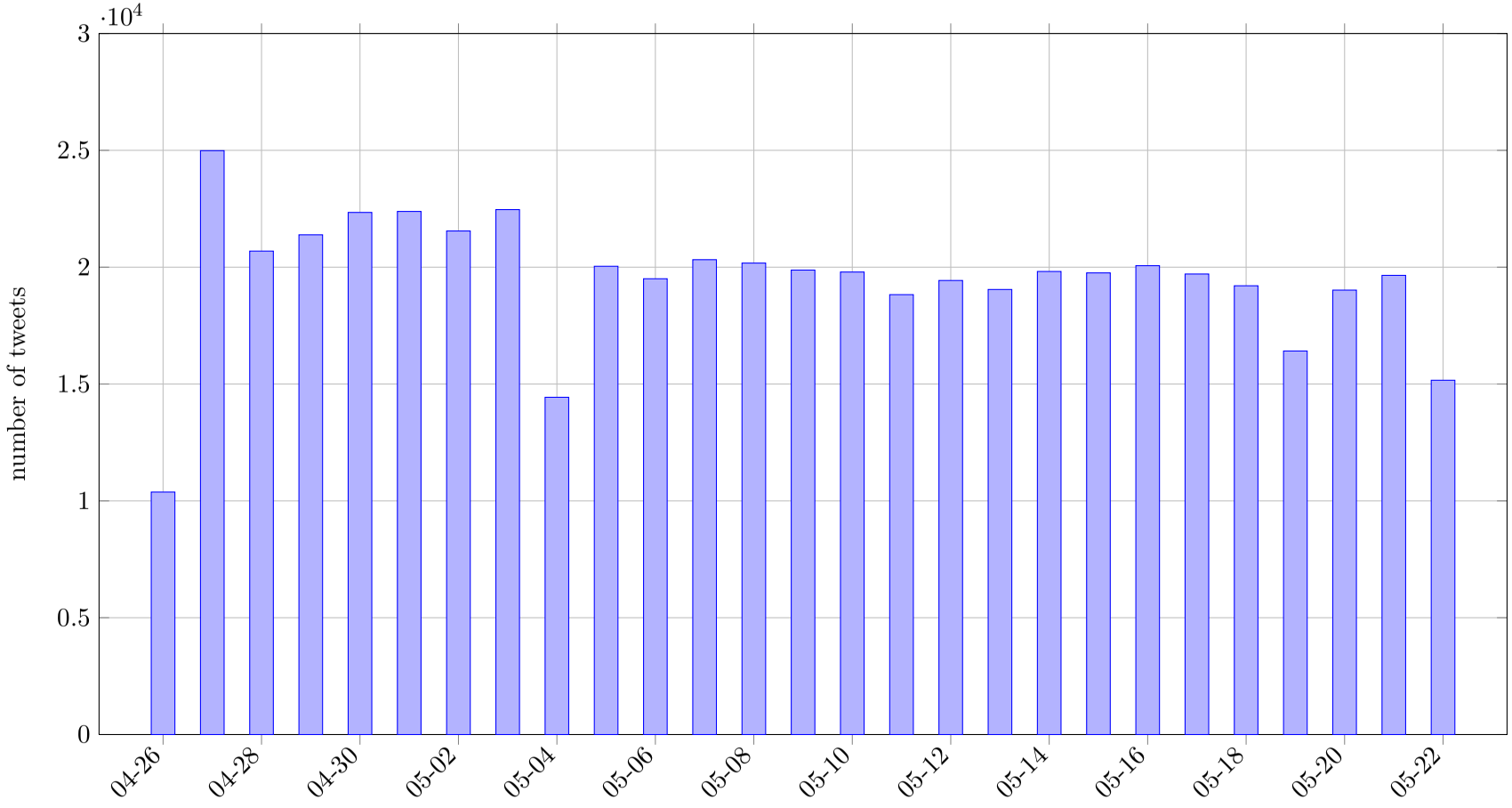


Fig. 5.6: Tweets collected for final evaluation by date of creation

6 CLASSIFICATION

Data collected for training were used to train a SVM classifier. Each tweet was tokenized to tokens, such as words, emoticons and hashtags. Non-valuable tweets, that did not contained at least 2 English words, were filtered, and therefore were not used in the training process. Features were extracted using a Hash Vectorizer for n-grams and a Count Vectorizer (with Tf-idf weighting) for emoticons [24]. The features from each vectorizer were concatenated using the Feature Union method [24]. The Variance threshold [24] was used as a basic method for a Feature Selection.

The process of building and training a classifier is described below. The building steps are visualized on Fig. 6.1.

6.1 Data Fetching and Preprocessing

Tweets for training, testing and validation are fetched from the `tweets` table. Tweets with emotional labels (*ag, sp, hp, sd, dg*) assigned in the search process and by the Baseline algorithm are selected. Only non-retweeted (non-reposted) and non-reply tweets are fetched, by setting a condition in two table columns (Appendix A):
`retweeted = 0 AND is_reply = 0`.

6.1.1 Tokenization

Tokenization of text from tweets is done by Christopher Potts's Twitter-aware tokenizer[7]. Tokenizer splits the text into a tokens of ASCII emoticons (incl. the reverse order), phone numbers, HTML tags, Twitter hashtags, Twitter user handles and words. The Twitter-aware tokenizer also replaces the HTML entities into a proper UTF-8 Unicode characters. The tokenization is done by a regular expression, compiled from all named token possibilities.

The Twitter-aware tokenizer was refactored to run on Python 3. UTF-8 emoticons and emojis were added to the list of token possibilities, to the regular expression of a tokenizer, specifically these sets:

- U+2600 – U+26FF — symbols (♀, ☺),
- U+2700 – U+27BF — symbols and arrows (⋆),
- U+10080 – U+100FF — linear ideograms,
- U+1F300 – U+1F5FF — pictographs (☾),
- U+1F600 – U+1F64F — emojis.

The tokenizer is implemented in the `TweetUtils` module as the `TweetTokenizer` class. The class have the `tokenize()` method for tokenizing a single text string and the `tokenize_batch()` method for tokenizing an iterable object (such as list).

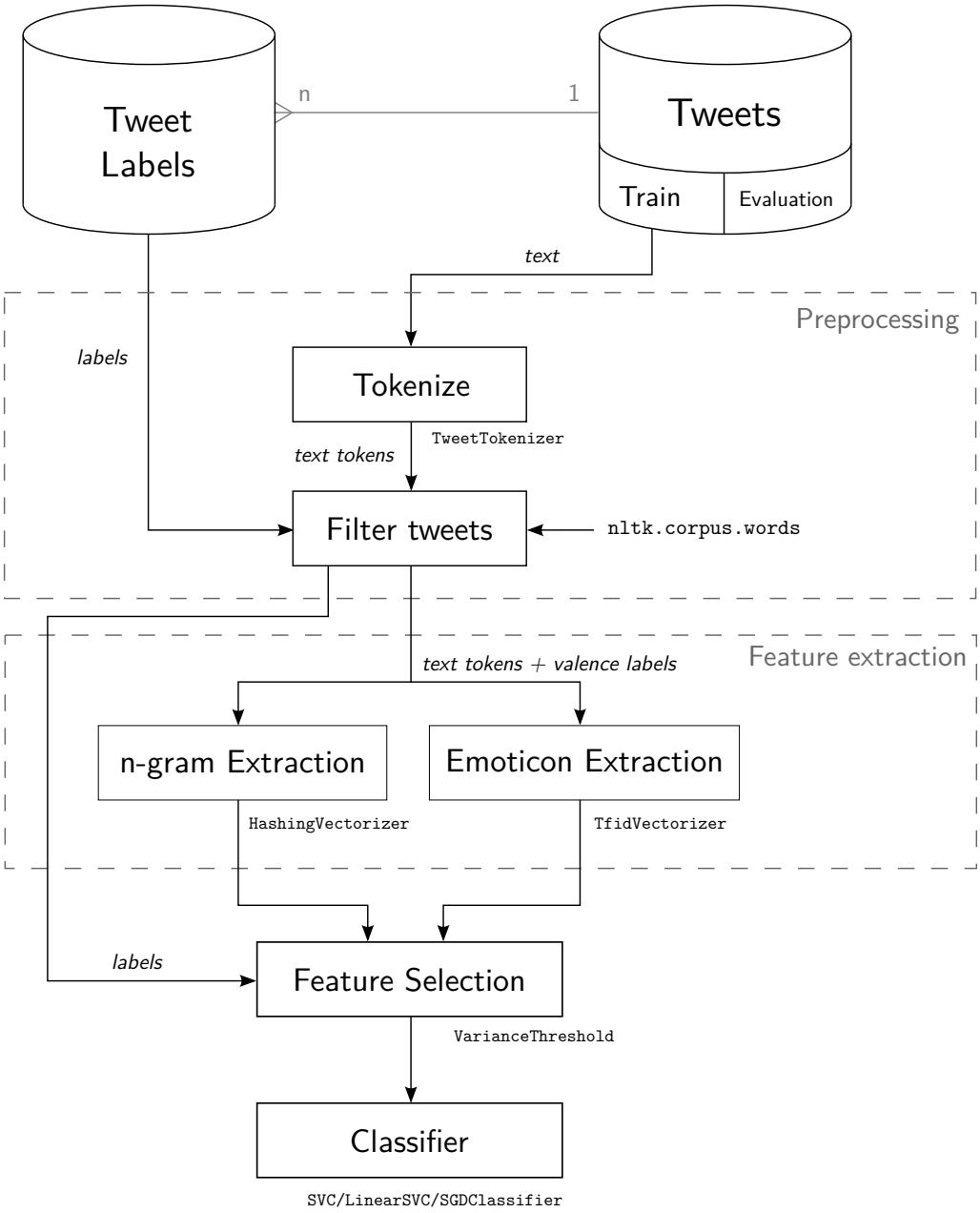


Fig. 6.1: Classifier model build process

6.1.2 Filtering

Filtering tweets for training purposes consists of these steps:

1. Tweets which contain less than 2 English words are discarded.
2. Removing hashtags that have been used for searching.
3. Replacing URLs and user handles with special tokens.

Fetched tweets are filtered before passing them into the learning process. Tweets that have less than 2 English words are not passed to the vectorizer, and therefore discarded from the learning process. Words are selected from the tokens, after the tokenization, and compared with the word dictionary.

The word dictionary is retrieved from the NLTK (Section 3.5). The module containing the dictionary is imported from `nltk.corpus.words`.[1] The module contains dictionaries of various languages, and therefore the desired language must be specified. The words returned from the dictionary are returned in Python's list format, by default. The `list` format has time complexity of O(n), which slows down the searching for a proper word. The reference variable of words was transformed to the Python's `set` format. The time complexity of searching and value reading of the `set` format is O(1). Transformation from `list` to `set` reduced the searching process from second to nanoseconds. The Python's `set` format is an implementation of Hash Set, however it does not support indexing and slicing.

Example of using the word dictionary:

```
from nltk.corpus import words

words = words.words('en')   # Get English Words
words = set(words)          # Transform to set

# Test if words is in the~dictionary
'word' in words  # Returns: True or False
```

The hashtags from search terms (Tab. 5.1) are removed before training. The absence of removal would lead classifier to be focused primary on these hashtags.

Tokenized tweets are filtered for Feature Extraction, to include only the words, emoticons, special tokens and other characters, based on the Feature Extractor. The filters for each Feature Extractor will be presented in the following section.

An example of tokenization and n-gram filtering is shown on Fig. 6.2.

6.2 Feature Extraction

Two feature extraction schemes were used, namely the *Hashing Vectorizer* and the *Count Vectorizer with Tf-idf* weighting. The feature extraction schemes were build using the Scikit library [23], located in a module `sklearn.feature_extraction.text`. Each vectorizer

[1] Note, that the corpus needs to be downloaded using a `nltk.download()`.

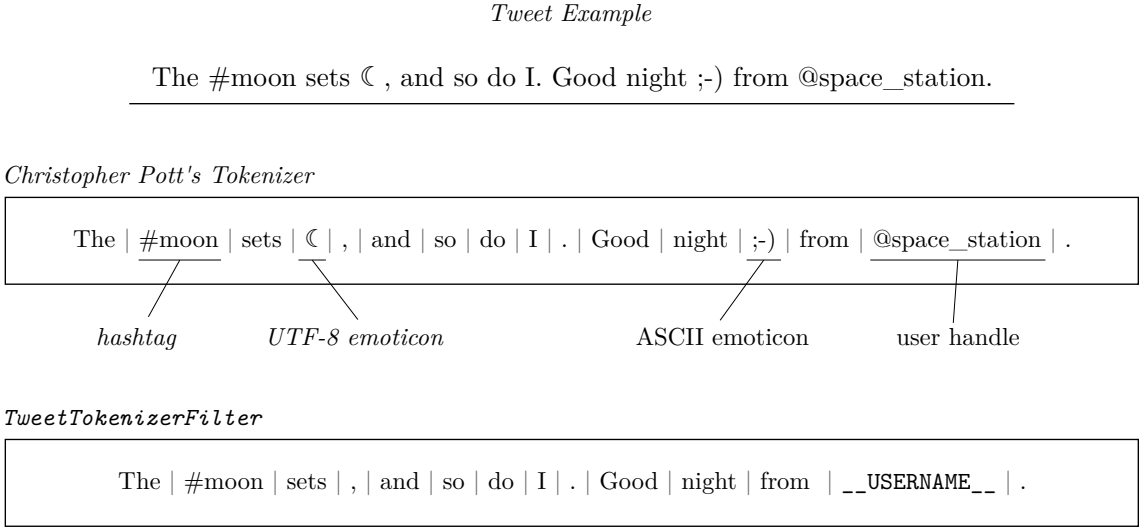


Fig. 6.2: Example of tokenization and filtration

had to implement `fit()` and `transform()` methods. Features from vectorizers were concatenated by a Feature Union [24].

6.2.1 *n*-gram Extraction

Hashing Vectorizer (`HashingVectorizer`) was used to extract n-grams, where $n \in \{1, 2, 3\}$. Therefore only unigrams, bigrams and trigrams were considered as features.

Feature hashing is based on a "hashing trick"[25], where each feature is hashed into a number. The hash value corresponds to a column in the hash table. The hash table should be adequately big enough, to be able to store several features and minimize hashing collisions. Most of the hash table values are zeros, and therefore the hash table is represented by a sparse matrix. The signed 32 bit version of *Murmurhash3* is used in the Scikit's implementation of `HashingVectorizer`. The hash value is shortened by a modulo of number of desired features (number of hash table's columns).

The feature vectorizer is stateless, and therefore it does not need to be fitted before the transformation. The statelessness have a disadvantage, that the already hashed feature cannot be reversely computed for purposes of real text n-gram feature determination. The advantage, on the other hand, is low memory consumption, because of an absence of bag-of-words dictionary. The feature hashing is suitable for large scale data.

The tokens filtration, for the n-gram feature extractor, is done by implemented `TweetTokenizerFilter` (`TweetUtils` module). The tokenizer filter removes the emoticons and emojis, and passes only words and other set of characters to the vectorizer. The Twitter user handles are replaced by the special token `__USERNAME__`, to represent the username placement. The URL links are replaced by the `__URL__` token. The replacement of user handles and URLs helps to remove insignificant data, while keeping an information of placement and count.

The valence labels were added to the tokens for the n-gram feature extractor, to support the decision making. The positive valence was identified by a `__pos__` token and the negative valence by a `__neg__` token.

The Scikit's `HashingVectorizer` parameters were set as follows:

```
HashingVectorizer(lowercase=False,
                  preprocessor=None,
                  tokenizer=tok_filter.filter,
                  non_negative=True,
                  ngram_range=(1, 3),
                  n_features=2 ** 22,
)
```

The lowercasing of characters and preprocessing was done by the `TweetTokenizer`. Tweets passed to the vectorizer were already tokenized, and therefore the `tokenizer` parameter was set to the reference of filtering method of an `TweetTokenizerFilter` instance. Maximum number of features (table columns) was set to $2^{20} = 1,048,576$ features.

6.2.2 Emoticon Extraction

The *Count Vectorizer with Tf-idf* weighting was used to extract the emoticons and emojis from tweets. The data for feature extraction were filtered to include only the emoticons. The filtration was done on tokens, from an already tokenized tweets. The instance of `TweetTokenizerEmoticons` (`TweetUtils` module) provided the filtration mechanism.

The Scikit's *Tf-idf Vectorizer* consist of a *Count Vectorizer* followed by a *Tf-idf Transformer*. The Tf-idf, term-frequency times inverse document-frequency, is a common weighting scheme, used especially in a document classification. The Tf-idf weighting scales down the impact of tokens, that occur frequently in the given corpus, and therefore making the less frequently occurred tokens to be more valued. More about the Tf-idf can be found in [26].

The `TfidfVectorizer` paramters were set as follows:

```
TfidfVectorizer(lowercase=False,
                preprocessor=None,
                # TweetTokenizerEmoticons instance
                tokenizer=tok_emoticons.filter,
                max_df=1.0,
                min_df=1,
                max_features=None,  # All emoticons
                use_idf=True,
                smooth_idf=True,
)
```

The `tok_emoticons` was an instance of `TweetTokenizerEmoticons`.

6.3 Feature Selection

Features, from the Hashing Vectorizer and the Tf-idf Vectorizer, were concatenated by a Feature Union [24]. The constructed Feature Union is an instance of `sklearn.pipeline.FeatureUnion`, which holds all vectorizers. Each vectorizer (transformer) in the Feature Union is identified by its name. Optionally, the vectorizer can be wighted, by setting the `transformer_weights` parameter. The weight parameter is defined in a Python's dictionary format, where the keys are the identifiers of vectorizers.

The first feature selection is done by a *Variance Threshold* method, using the instance of `sklearn.feature_selection.VarianceThreshold`. The threshold was set to 0.000004.

The `LinearSVC` and the `SGDClassifier` classifiers (will be presented in Secion 6.4) implements the loss function, and have the penalty type option. The penalty can be set to the L1 (Least Absolute Deviation, Eq. 6.1) or to the L2 (Least Squares Error, Eq. 6.2). One of the loss function, that is available in both classifier, is *Squared Hinge-loss* (L2) [27].

$$S = \sum_{i}^{m} |y_i - \mathrm{f}(x_i)| \tag{6.1}$$

$$S = \sum_{i}^{m} (y_i - \mathrm{f}(x_i))^2 \tag{6.2}$$

6.4 Classifiers

SVM classifiers (Section 1.4.4) were used for the Twitter emotional and valence classification. The linear kernel was used for SVM class separation, since it was proven to be efficient for textual and document classification ([14]). The "one-vs-rest" approach was used for the multiclass classification.

6.4.1 SVM Classifiers in Scikit

Scikit implements these SVM classifiers for supervised learning, that have a linear kernel:

- `SVC` (`kernel='linear'`),
- `NuSVC` (`kernel='linear'`),
- `LinearSVC`,
- `SGDClassifier`.

Each instance of Scikit's SVM classifier have a `fit()` method for training and a `predict()` method for classification. The `fit(X, y)` method takes two arrays as arguments. The `X` is a matrix of vectorized data, where each row represents a tweet (document, or a sample) and each column represents one feature. The `y` is a row vector of annotated classes for each tweet (document). The classification method `predict(X)` takes the same type of matrix as the fit method.

The `SVC` and the `NuSVC` are an implementations of *libSVM* [28]. These implementations supports various kernels, namely the RBF, linear, polynomial, sigmoid and the precomputed kernel. The time complexity is higher than quadratic, which is unsatisfactory for training on a higher number of tweets (documents). The multiclass support is done only by a "one-vs-one" scheme, but can be also done with "one-vs-rest" scheme using a `OneVsRestClassifier` wrapper [24]. The `NuSVC` is an Nu-Support Vector Classifier, which can control the number of support vectors.

The `LinearSVC` is an implementation of *liblinear* [29]. The liblinear is more flexible, than *libSVM*, and supports various penalties and loss functions. The liblinear is very efficient for scaling to large number of data (tweets), than the *libSVM*. The multiclass support is done by "one-vs-rest" scheme. The disadvantage of *liblinear* is the usage of internal sparse data representation, where a memory copying is required.

The `SGDClassifier` is a set of linear classifiers with Stochastic Gradient Descent training, which also includes the SVM. The SGD classifier overcomes the need of coping data to the internal representation, as occurring in the `LinearSVC` (*liblinear*). The advantage of `SGDClassifier` is a much fast scaling to very large data sets, comparing against the `LinearSVC`.

6.4.2 Classification Settings

The `LinearSVC` classifier parameters, for classification to the emotional and valence classes, were set as follows:

```
LinearSVC(C=1.0,
          loss='squared_hinge',
          penalty='l2',
          dual=False,
          tol=1e-3,  # tolerance
          class_weight='auto',
          multi_class='ovr',
)
```

The `class_weight='auto'` sets the class weighting based on a number of tweets (samples) for each class – done for a reason of unbalanced number of tweets in each class. The `multi_class='ovr'` selects the "one-vs-rest" scheme for multiclass classification.

The optional `SGDClassifier` was used to show the differences against the `LinearSVC`. The parameters were set to the same values as for the `LinearSVC` shown above. Training of the `LinearSVC` was done in a tens of seconds, however the training of `SGDClassifier` took only milliseconds, but with a less precise classification.

6.5 Building a Trained Model

Two models were trained and built, namely the Emotional classifier model and the Valence classifier model. The valence classifier, with positive and negative classes, was used to support the classification of emotional classes. Each model was saved (serialized) for a later

classification, for the final evaluation process. An own application was developed to do the building and training the model.

6.5.1 Training Application

The application *Build Model* (`build_model.py`) was implemented for building and training a model for Twitter emotional and valence classification. Tweets annotated with *ag*, *sp*, *hp*, *sd* and *dg* labels were fetched for the training of emotional classifier. The *pos* and *neg* labeled tweets (annotated by search process) were fetched for the valence classifier training. Each model had to be build and trained separately. Training of each classifier was done by running the application with a different arguments.

The application have these arguments available:

- `MODELTYPE` – type of the model: *emotional* or *valence* (required),
- `-c CLF` – classifier type: svc, linearsvc (default) or sgdc,
- `-f FILENAME` – path to a filename, where the model will be saved to,
- `-x` – do the cross-validation.

Tweets are fetched from the database using the `Database.get_tweets_for_clf()` method. The fetched tweets are randomly shuffled and separated into a training (90 %) and a testing group (10 %)[2], before getting into the Feature Extractor. The separated tweets are split into tweets texts and annotated labels, using a `make_bunch()` function (TweetUtils). The tweets for training are cleared from the searched hashtags.

Each steps for training a classifier (and later for classification) (Fig. 6.1) are pipelined together using the `sklearn.pipeline.make_pipeline()` function. The pipeline comprises of the Feature Union of two vectorizers (`HashingVectorizer` and `TfidfVectorizer`), the Feature Selector `VarianceThreshold` and the SVM classifier. The created pipeline is shown on the Figure 6.3.

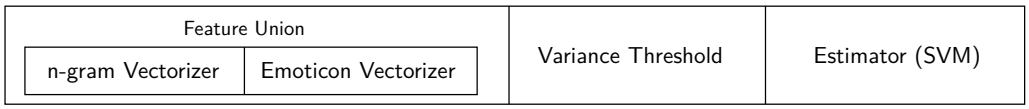


Fig. 6.3: Classification pipeline

The instance of a class `TrainedModel` (TweetUtils) holds the pipeline and the names (text representation) of classes (`target_names`). The trained model is saved and serialized by the Python's pickle module, if the *Build Model* application have been started with the `-f` option. Examples of steps for fetching, shuffling, splitting, making and saving the trained model are shown in the following code:

```
# Fetching tweets from database
tweets = db.get_tweets_for_clf(labels, limit)
n_tweets = len(tweets)
```

[2]Splitting is done before the filtration, and therefore may not be accurate.

```
tweets = shuffle(tweets)  # from sklearn.utils

# Separation to training and testing groups
ratio = .9
tweets_train = tweets[ : round(ratio * n_tweets)]
tweets_test = tweets[round(ratio * n_tweets) : ]

# Spliting the~data and labels
remove = ( ...searched hashtags... )
tweets_train, y_train = make_bunch(tweets_train, labels,
                                   tokenizer, remove=remove,
                                   min_words=2)
tweets_test, y_test = make_bunch(tweets_test, labels, tokenizer,
                                 min_words=0)

# Making a~pipeline
pipeline = make_pipeline(feature_union, feature_selection, clf)
model = TrainedModel(pipeline=pipeline, target_names=labels)

# Saving the~model
import pickle
with open(filename, 'wb') as fh:
    pickle.dump(model, fh)
```

A report, consisting of precision (Eq. 6.3), recall (Eq. 6.4), and f_1 score (Eq. 6.5) [30], is presented after the training process. The average precision is weighted by the number of tweets in each class.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6.3}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{6.4}$$

$$f_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6.5}$$

The 5-fold cross-validation is done by the `sklearn.cross_validation.cross_val_score()` function.

6.5.2 Evaluation of Emotional Classifier

Emotional classifier was trained on 39 947 tweets, from a total of 47 461 filtered tweets. Vectorization and tokenization took around 6 s, making a 1 048 757 sparse features. Feature Selection process identified 16, 979 features for the classifier.

The training time of `LinearSVC` classifier was around 32 s, and the test time about 3 ms[3]. The 5-fold Cross-validation gave an accuracy of 0.8910 ($\pm$0.01) and f_1 score 0.8972 ($\pm$0.01). The precision, recall and f1 score for each class can be seen on Table 6.1.

The training time of `Stochastic Gradient Descent` classifier (`SGDClassifier`) was only around 310 ms. The 5-fold Cross-validation accuracy was 0.8707 ($\pm$ 0.02) and the f_1

[3]Features were already vectorized (extracted) and selected.

Tab. 6.1: Classification report of Emotional classifier – type `LinearSVC`

Class	Precision	Recall	f_1	Support
ag	0.72	0.87	0.79	181
sp	0.56	0.87	0.68	470
hp	0.98	0.89	0.93	3110
sd	0.95	0.92	0.94	745
dg	0.94	0.95	0.95	240
average	0.92	0.90	0.90	total 4746

score 0.8764 (± 0.01). The precision, recall and f1 score for each class can be seen on Table 6.2.

Tab. 6.2: Classification report of Emotional classifier – type `SGDClassifier`

Class	Precision	Recall	f_1	Support
ag	0.70	0.74	0.72	202
sp	0.49	0.73	0.59	460
hp	0.95	0.88	0.91	3103
sd	0.92	0.91	0.92	742
dg	0.98	0.89	0.93	239
average	0.89	0.87	0.88	total 4746

The classifier of type `LinearSVC` was selected, since it was more precise than the SGD classifier, and had taken the same classification time. The trained models of emotional classifiers were saved to the `SVM-linear-model-emotional.pkl` for `LinearSVC` and to `SVM-sgd-model-emotional.pkl` for `SGDClassifier` (see Appendix C).

The surprise emotion had an precision of around 50 % for each classifier. This means, that the surprise emotions were correctly classified only in the 1/2 of cases – same as using a randomly generated surprise emotion flag.

6.5.3 Evaluation of Valence Classifier

The valence classifier (pos/neg) was trained on 123 509 tweets, from a total of 139 174 unfiltered tweets. Vectorization and tokenization took around 34 s, making a 1 048 820 sparse features. Feature Selection process identified 16, 590 features for the classifier.

The training time of `LinearSVC` classifier was around 2.6 s, and the test time about 3 ms[4]. The 5-fold Cross-validation gave an accuracy of 0.9843 (±0.00) and f_1 score 0.9843 (±0.00). The precision, recall and f1 score for each class can be seen on Table 6.4.

The training time of `Stochastic Gradient Descent` classifier (`SGDClassifier`) was only around 261 ms. The 5-fold Cross-validation accuracy was 0.9757 (± 0.02) and the f_1 score 0.9758 (± 0.01). The precision, recall and f1 score for each class can be seen on Table 6.2.

[4]Features were already vectorized (extracted) and selected.

Tab. 6.3: Classification report of Valence classifier – type `LinearSVC`

Class	**Precision**	**Recall**	f_1	**Support**
pos	0.99	0.99	0.99	10876
neg	0.97	0.99	0.98	4789
average	0.99	0.99	0.99	total 15665

Tab. 6.4: Classification report of Valence classifier – type `SGDClassifier`

Class	**Precision**	**Recall**	f_1	**Support**
pos	0.99	0.97	0.98	10930
neg	0.94	0.98	0.96	4735
average	0.97	0.97	0.97	total 15665

The classifier of type `LinearSVC` was selected, since it had taken the same classification time. The trained models of valence classifiers were saved to the `SVM-linear-model--valence.pkl` for `LinearSVC` and to `SVM-sgd-model-valence.pkl` for `SGDClassifier` (see Appendix C).

7 FINAL EVALUATION

This chapter discuss the final evaluation of classified data, using the trained classifier. The tweets were collected in a time period of one month, as presented in Section 5.2.2. The classification of collected tweets was done in the application with GUI – *Geo Emotions*. The results of classified emotional classes were saved to the database, for later visualization. Visualization was done in the same application, where the emotional view were presented on the map of USA.

7.1 Fetching and Classifying Tweets

Classification of collected tweets can be done either in the visualizing application *Geo Emotions* or right after collection in the *Mining Application*. The option "reclassify" must be checked for classification in the *Geo Emotions* application. If the tweets need to be classified right after they are fetch from Twitter, the argument `--classify` must be used, when starting the *Mining Application*.

The *Geo Emotions* application takes tweets collected in the collection process (Section 5.2.2), when classification is requested. Only tweets from locations, that have the `mining_group` equal to "classify", are selected. Fetching tweets for the classification is done by the method (instance) `Database.get_tweets()`, for which the location ID of tweets is specified. The classification is done using trained classifiers, as presented in Section 6.5.

The `Classification` module was created to load and hold the Emotional classifier (`EmotionalClassifier`) and the Valence classifier (`ValenceClassifier`). The `ValenceClassifier` loads the valence `LinearSVM` model from the file `SVM-linear-model-valence.pkl`. The `EmotionalClassifier` loads the emotional `LinearSVM` model from the file `SVM-linear-model-emotional.pkl` and creates an instance of `ValenceClassifier`. The data classified by the instance of `EmotionalClassifier` are classified at first to a valence class by the `ValenceClassifier`, which is appended as an additional token to the rest of tokens (of tokenized tweets).

Each Python's classes, that are used for holding the classifier, implements the `classify()` method. The method accepts a list of raw data for the first argument. Raw data are tokenized and then passed to the `predict()` method of the pipelined classifier (Fig. 6.3). The pipeline of the classifier is located in the instance of `TrainedModel`. Trained model instance is loaded from the file, using the Python's pickle data serializer: `pickle.load(filehandle)`.

The *Geo Emotions* application uses multiprocessing approach, to reduce the time of classification. Several worker threads are created, based on a number of CPU cores. Two threads are created for each CPU core. Fetched tweets are separated into smaller parts, called chunks, before creation of worker threads. Each chunk contains a maximum of 250 tweets, that are waiting to be classified. The chunk is passed to the task queue, from which the worker threads read. The results of classification (classes for each tweet) are passed to the result queue by a worker thread. The `task_done()` is called on the task

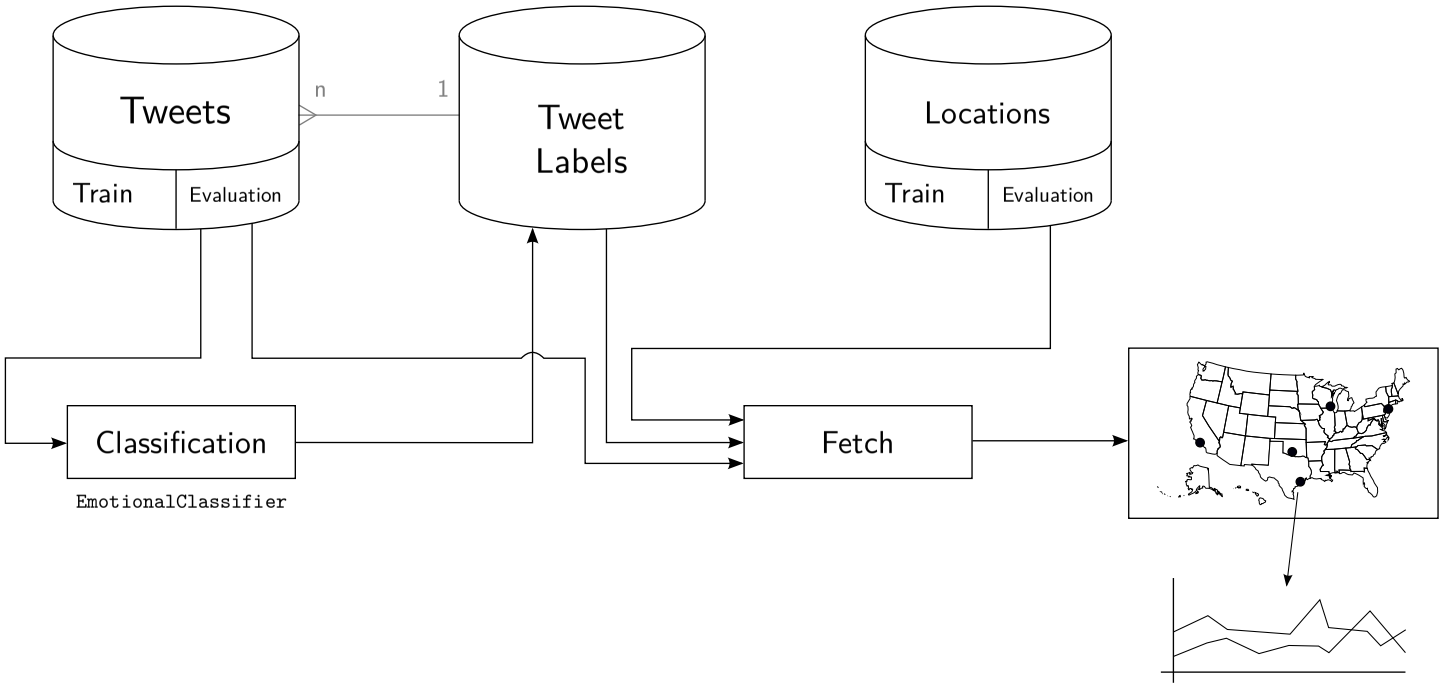


Fig. 7.1: Classification and Visualization

queue by a worker thread, when the chunk classification is finished. The result queue is read by the main classification thread, which concatenates the results (classes) and updates the graphical progress bar. The final concatenated classes of tweets are saved to the database, after all jobs from the task queue are done. The classification takes around 2 minutes, when using 8 thread utilization.

7.2 Visualization

The *Geo Emotions* application (`geo_emotions.py`) was created to visualize emotional changes over time in a certain geographical location. The application also does the classification of unclassified tweets, in case they are not already classified by the mining application.

7.2.1 Graphical User Interface

The Graphical User Interface of the application was implemented using a GTK+ toolkit. The design of the GUI is shown on the Fig. 7.2. The GUI consists of a button, checkbox, status line, progress bar and a drawing window. The drawing window is a WebKit's WebView component, which is basically a small web browser.

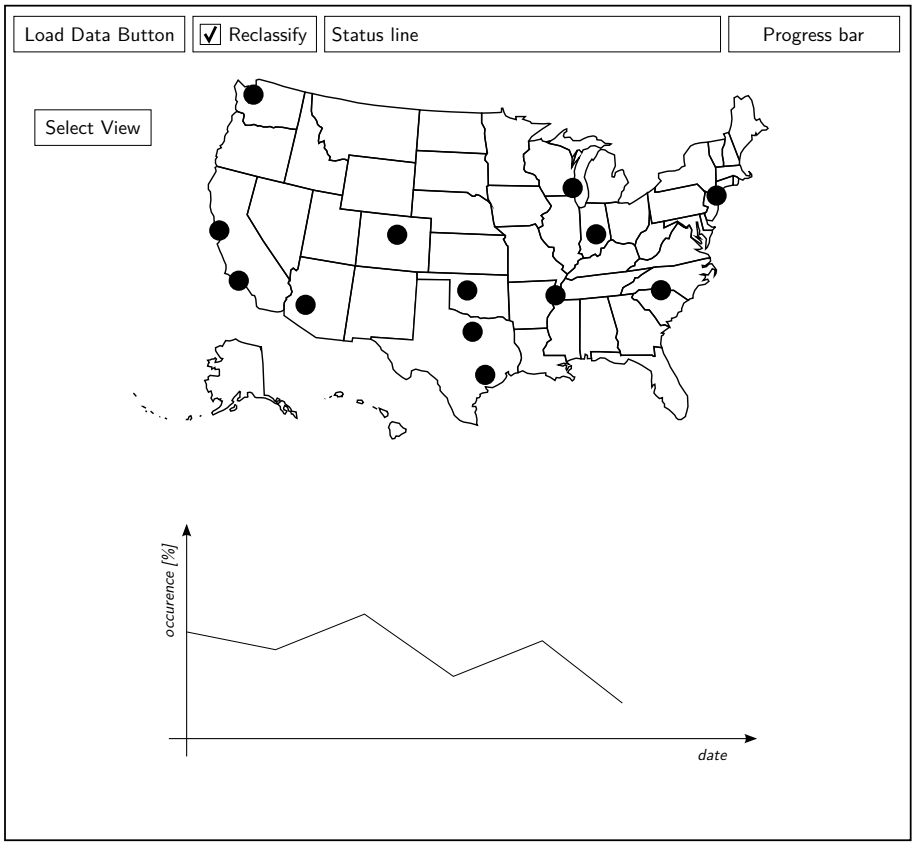


Fig. 7.2: Geo Emotions GUI Design

7.2.2 Generation of Visualization

Loading and visualizing the data is done by clicking on the "Load Data" button. The button's action spawns a new thread of a `GeoEmotionsThread`, which handles the classification and generates data for visualization. The application does emotional classification (Section 7.1) of tweets from geographical locations intended for classification, if the "Reclassify" option is checked.

Statistical data for visualization of already classified tweets are selected from the database. The statistics of an emotional occurrences are calculated from the emotional classes, based on the geographical location and date of tweet submission. A mixed value is computed to visualize all emotions at once. Calculation of mixed emotional value is done as follows:

$$mixed\_value = ag * 0 + sd * 30 + dg * 40 + sp * 50 + hp * 100, \tag{7.1}$$

where the ag, sd, dg, sp, and the hp represents a percentage of emotional occurrence, that they express. The percentage is calculated for each geographical location and each date separately. The equation 7.1 was proposed based on trial-error process. The mixed value is shown as a color gradient, as seen on Fig. 7.3.

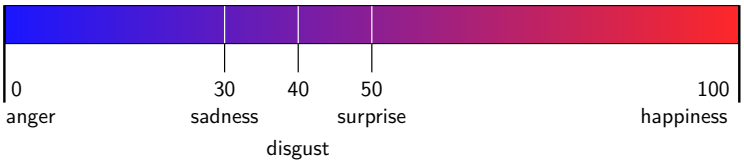


Fig. 7.3: Gradient for showing mixed emotions on the map

The visualization is generated into the HTML content, and after that shown within the WebView component. The HTML is generated by an instance of `HtmlDataVisualizer`, using a HTML template. The HTML template consists of common HTML5 skeleton, Google Charts API[1] and JavaScript. The JavaScript handles the charts data and user's interaction in the WebView GUI component. The computed statistics are visualized on the map for each geographical location for a selected date, and also on the graph, for the desired geographical location.

The map in the application (Fig. 7.4) shows markers on the cities where the tweets were collected. The radius of markers depends on the tweets collection radius, in that particular geographical area. The color of the marker is dependent on the mixed emotional value (Eq. 7.1) of the selected date, if the mixed emotional view is selected. When a specified emotion is selected to be viewed, then the color of marker represents the percentage of selected emotion. The color gradient is dynamically scaled, based on the minimums and maximums of percentages for all geographical locations for selected date. Scaling insures better representation, that can be visually viewed. Date is selected using slider under the map.

[1] `https://developers.google.com/chart/`

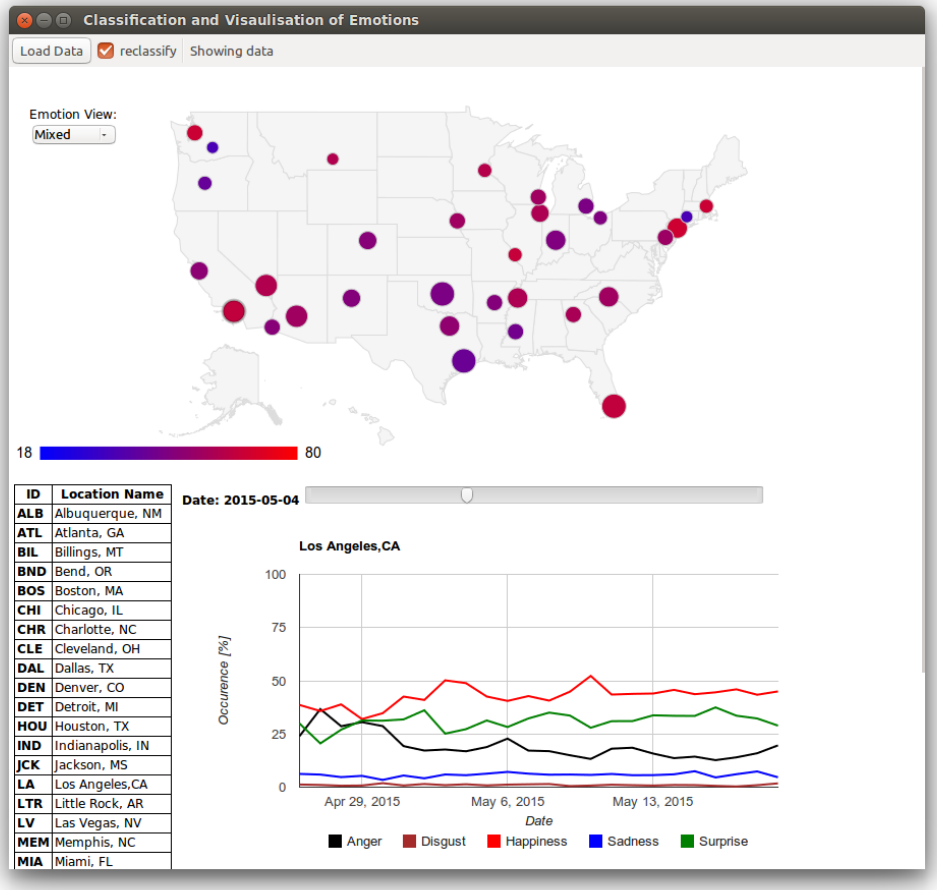


Fig. 7.4: Geo Emotions Application

The line chart is drawn after selection of desired city by clicking on the marker. The chart shows the temporal changes of all emotions, where the x-axis represents the date and y-axis represents the percentual occurrence of emotions.

7.3 Statistics

Table 7.1 shows the statical information about the tweet collection process and emotions that have been reflected in the tweets.

Tab. 7.1: Final Statistics

Tweets collected for training	222,476	
Tweets collected for final evaluation	526,421	510,448 effective
Tweets total		748,897
Time period of collection for evaluation		27 days
Total cities		32 cities
Average tweets per day		19,497 tweets
Classification time (8 threads, all tweets)		96 s
Most appearing emotion		Happiness (*hp*)
Least appearing emotion		Disgust (*dg*)
Anger	114,243 tweets	22 %
Disgust	6,822 tweets	1 %
Happiness	198,881 tweets	39 %
Sadness	50,641 tweets	10 %
Surprise	139,861 tweets	28 %

8 CONCLUSION

Emotions reflected in text documents and messages lack of facial, voice or mimic expressions. This drawback makes it a difficult task for text recognition systems. Emotions were defined, among other researchers, by Ekman [2] to *anger*, *disgust*, *fear*, *happiness*, *sadness*, and *surprise*.

There are variety of classifier algorithms available, such as the Naive Byes, k-nearest neighbors, MaxEnt and Support Vector Machines. The SVM classifier has proven to be the most efficient for text classification.

From Social Networks available nowadays, such as Facebook, Myspace, Twitter or hi5, Twitter has the most amount of data available for public. Not only the Twitter data are mostly text oriented, but they are also limited to the 140 characters. The limitation to a short messages indirectly enforce users to express their emotions intensively. Twitter messages, called tweets, contain emoticons and hashtags among the words. Emoticons themselves express emotions. Tweets often include slang words, which makes the task of emotion recognition even more difficult when using a dictionary-based approach.

Twitter provides an APIs for reading and posting tweets. The Search API provides an access to the data, by applying search queries in the requests. One of the parameters in the search query request is the `geocode`, which limits the tweets a certain radius within a geographical location. Maximum of 100 tweets can be acquired per request and maximum of 480 requests can be sent within a 15 minute window.

First I proposed and developed an application that used the Baseline algorithm for tweets classification into a 8 emotional labels. The application uses a location coordinates, based on user input (address), to fetch the tweets. The fetched tweets are classified one by one using unigrams as features. The Baseline corpus consist of most frequently used words and emoticons that strongly express the emotions. The result is represented by a pie chart drawn on a map.

Next I decided to separate the task into two parts. The first part was an server application, which acquired tweets from Twitter – either for testing, training and validating the classifier or for final data evaluation. The second part was a client application, that classified already collected tweets and visualized the results. The aim of the thesis was to target on tweets within the United States of America.

Data from Twitter were acquired using proposed and developed application, called *Twitter Miner*. The application used the Twitter API library for pooling Twitter in regular time intervals using a search query. Fetched tweets were stored into the MySQL database.

Tweets intended for training and testing the classifier were fetched using selected search terms. The search terms were constructed by an emotions or hashtags, of words from the Baseline lexicon. Each fetched tweet was annotated by a labels attached to a search term, that the tweet was collected by. The tweets collection was targeted on the continental part of USA. 222 480 tweets were collected in total for training and testing. The happiness emotion prevailed over the others. The fear emotion was reflected by only 800 tweets, which

was insufficient for the classifier training, and therefore this emotion was not considered.

Tweets for final evaluation were collected from 32 cities in a 27 day period. The cities were selected by their population or special feature, such as higher criminality or burglary. Tweet collection was running non-stop on a server, collecting (max.) 40 tweets per city per hour. 510 448 were collected in total for the classification and evaluation. The collection average per day was 19 497 tweets.

Two classifiers were trained and built by own proposed *Build Model* application. The application used labeled tweets from the database. The tweets were preprocessed by the tokenizer and filter. The trained model contains two feature vectorizers, namely the Hashing Vectorizer and Tf-idf Vectorizer. Two implementations of the SVM's from the Scikit's toolkit were used: Linear SVC and Stochastic Gradient Descend SVC. Two classifiers were trained by the application, namely the Valence classifier – classifying into pos/neg classes, and the Emotional classifier, which considered five emotions.

An emotional classifier classified only into emotional classes, excluding the fear emotion. The valence label was added as a feature, classified by the Valence classifier. Only the `LinearSVC` was used for final evaluation, since it showed better classification accuracy comparing to `SGDClassifier`. The accuracy given by 5-fold cross-validation was around 89 %. The precision of classification for *surprise* emotion was only 56 %.

Application *Geo Emotions* was developed for visualization and classification of collected tweets. Classification process was performed by multithreading, using worker threads. Classification of half million tweets by 8 threads took 96 s. Tokenization and vectorization of features were the most time consuming upon the classification process. The results of emotions for each city and for each date were visualized on a map of USA. Each city had an marker, which color was dependent on the occurred emotion or mixed emotions. The date could be selected by slider under the map. The line chart of an emotion progress for all days, that have been mined, was shown after selecting the desired city.

The classification can be improved by collecting more data, that are rightfully annotated. New features can be used, such as Class Sequential Rules, to improve the model. An option of manual human annotation is possible, using a web interface for the annotators (not included).

The Python was used as the main programming language, with the support of other libraries. Classifiers, feature extraction and feature selection were used from the Scikit toolkit. The HTML5 and JavaScript was used for visualization of classified data. The SQL was used for the communication between the applications and the database. The mining application is highly dependent on the data provided by the Twitter API.

The progress of the research, that was done for the diploma thesis, is shown on the timeline (Fig. 8.1). The list of all created application is shown on a Table 8.1.

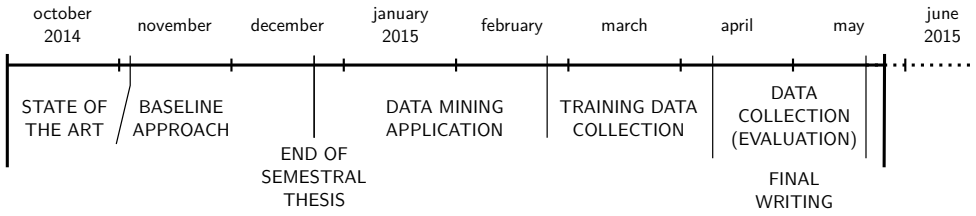


Fig. 8.1: Research Timeline

Tab. 8.1: Proposed and Developed Applications

Application Name	**Executable**	**Type**	**GUI**	**Purpose**
Twitter Emotion Baseline	`emotions.py`	Client	✓	Classification of tweets using Baseline algorithm
Twitter Miner	`miner.py`	Server	–	Tweets collection (and classification)
Build Model	`build_model.py`	Both	–	Training and testing classifiers
Geo Emotions	`geo_emotions.py`	Client	✓	Visualization and classification

BIBLIOGRAPHY

[1] AMAN, Saima. *Recognizing emotions in text.* 2007. PhD Thesis. University of Ottawa.

[2] EKMAN, Paul. An argument for basic emotions. *Cognition & Emotion*, 1992, 6.3-4: 169-200.

[3] WEN, Shiyang; WAN, Xiaojun. *Emotion Classification in Microblog Texts Using Class Sequential Rules.* Institute of Computer Science and Technology, Peking University, Beijing, China. In: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence. Quebec. 2014. [online] [cit. 2014-11-16]. ISBN 2159-5399. Available at: <`http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/download/8209/8419`>

[4] Natural Language Processing. JURAFSKY, Dan; Christopher MANNING. [online]. Standford. Coursera. [cit. 2014-11-16]. Dostupné z: <`https://class.coursera.org/nlp/lecture`>

[5] DAVIDOV, Dmitry; TSUR, Oren; RAPPOPORT, Ari. Enhanced sentiment learning using twitter hashtags and smileys. In: *Proceedings of the 23rd International Conference on Computational Linguistics: Posters.* Association for Computational Linguistics, 2010. p. 241-249.

[6] DAVIDOV, Dmitry; RAPPOPORT, Ari. Efficient unsupervised discovery of word categories using symmetric patterns and high frequency words. In: *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics.* Association for Computational Linguistics, 2006. p. 297-304.

[7] POTTS, Christopher. STANFORD LINGUISTICS. Sentiment Symposium Tutorial [online]. 2011 [cit. 2014-12-02]. Available at: <`http://sentiment.christopherpotts.net`>

[8] STRAPPARAVA, Carlo; VALITUTTI, Alessandro. WordNet Affect: an Affective Extension of WordNet. In: *LREC.* 2004. p. 1083-1086.

[9] WIEBE, Janyce; WILSON, Theresa; CARDIE, Claire. Annotating expressions of opinions and emotions in language. *Language resources and evaluation*, 2005, 39.2-3: 165-210.

[10] WILSON, Theresa Ann. *Fine-grained subjectivity and sentiment analysis: recognizing the intensity, polarity, and attitudes of private states.* ProQuest, 2008.

[11] WILSON, Theresa; WIEBE, Janyce; HOFFMANN, Paul. Recognizing contextual polarity in phrase-level sentiment analysis. In: *Proceedings of the conference on human language technology and empirical methods in natural language processing. Association for Computational Linguistics*, 2005. p. 347-354.

[12] MILLER, George A., et al. Introduction to wordnet: An on-line lexical database. *International journal of lexicography*, 1990, 3.4: 235-244.

[13] MILGRAM, Jonathan; CHERIET, Mohamed; SABOURIN, Robert. "One Against One" or "One Against All": Which One is Better for Handwriting Recognition with SVMs?. In: *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006.

[14] GO, Alec; BHAYANI, Richa; HUANG, Lei. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, 2009, 1-12.

[15] BARBOSA, Luciano; FENG, Junlan. Robust sentiment detection on twitter from biased and noisy data. In: *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*. Association for Computational Linguistics, 2010. p. 36-44.

[16] DUDA, Richard O.; HART, Peter E.; STORK, David G. *Pattern classification*. John Wiley & Sons, 2012.

[17] OAuth: Send secure authorized requests to the Twitter API. *Twitter Developers*. [cit. 2014-12-01]. Available at: <`https://dev.twitter.com/oauth`>

[18] REST APIs. *Twitter Developers*. [cit. 2014-12-01]. Available at: <`https://dev.twitter.com/rest/public`>

[19] The Streaming APIs *Twitter Developers*. [cit. 2014-12-01]. Available at: <`https://dev.twitter.com/streaming/overview`>

[20] PILGRIM, Mark. *Dive into Python 3*. New York: Apress, 2009, xlix, 360 p. Expert's voice in open source. ISBN 14-302-2415-0.

[21] ROBERTS, Kirk, et al. EmpaTweet: Annotating and Detecting Emotions on Twitter. In: *LREC*. 2012. p. 3806-3813.

[22] MOHAMMAD, Saif M. #Emotional tweets. In: *Proceedings of the First Joint Conference on Lexical and Computational Semantics-Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation*. Association for Computational Linguistics, 2012. p. 246-255.

[23] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011. Available at: <`http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf`>

[24] Documentation scikit-learn: machine learning in Python. [cit. 2015-05-02]. Available at: <`http://scikit-learn.org/stable/documentation.html`>

[25] WEINBERGER, Kilian, et al. Feature hashing for large scale multitask learning. In: *Proceedings of the 26th Annual International Conference on Machine Learning.* ACM, 2009. p. 1113-1120. Available at: <http://arxiv.org/pdf/0902.2206>

[26] AIZAWA, Akiko. An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, 2003, 39.1: 45-65.

[27] LEE, Ching-Pei; LIN, Chih-Jen. A study on L2-loss (squared hinge-loss) multiclass SVM. *Neural computation*, 2013, 25.5: 1302-1323.

[28] CHANG, Chih-Chung; LIN Chin-Jen. LIBSVM: A Library for Support Vector Machines. 2013. Available at: <http://csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf>

[29] FAN, Rong-En, et. al. LIBLINEAR: A Library for Lange Linear Classification. 2014. Available at: <http://csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf>

[30] POWERS, David Martin. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. 2011.

LIST OF SYMBOLS, PHYSICAL CONSTANTS AND ABBREVIATIONS

API	Application Programming Interface
CSV	Comma-separated values
GPS	Global Positioning System
GUI	Graphical User Interface
HTML	HyperText Markup Language
JSON	JavaScript Object Notation
kNN	k-nearest neighbors
MaxEnt	Maximum Entropy
NLP	Natural Language Processing
POS	part of speech
RBF	Radial Basis Function
REST	Representational State Transfer
SQL	Structured Query Language
SVC	Support Vector Classifier
SVM	Support Vector Machines
Tf-idf	Term Frequency–Inverse document frequency
UTC	Coordinated Universal Time
UTF-8	Universal Coded Character Set + Transformation Format–8-bit
WYSIWYG	What You See Is What You Get
XML	Extensible Markup Language

LIST OF APPENDICES

A SQL TABLE STRUCTURE

Note: Column in bold italic font represents a PRIMARY index.

Tab. A.1: `search_terms` Table

Column	**Type**	**Note**
search_category	`varchar(8)`	Internal identification
search	`varchar(255)`	Search query
labels	`varchar(255)`	Labels to add (separated by comma)
mine	`tinyint(1)`	Whether to use in mining

Tab. A.2: `locations` Table

Column	**Type**	**Note**
location_id	`varchar(8)`	
location_name	`varchar(64)`	Full location name
geocode	`varchar(64)`	Geo location in decimal format
radius	`int(11)`	Radius in km
mine	`tinyint(1)`	Whether to use in mining
mining_group	`enum('train', 'classify')`	For training or classification?

Tab. A.3: `tweets` Table

Column	**Type**	**Note**
id	`bigint(20)`	Real Twitter Tweet ID
location_id	`varchar(8)`	Location id of miner (refers to locations)
created_at	`datetime`	Tweet time and date of creation
text	`varchar(255)`	Text of the tweet
retweet_count	`int(11)`	Number of retweets
is_reply	`tinyint(1)`	Whether is an reply
retweeted	`tinyint(1)`	Whether the text is retweeted from another user
user_id	`bigint(64)`	Real Twitter User ID
iso_language	`char(2)`	Language of the text in ISO
search_category	`varchar(8)`	Search category
mined_at	`timestamp`	Time and date of mining

Tab. A.4: `tweet_labels` Table

Column	**Type**	**Note**
tweet_id	`bigint(20)`	Refers to tweets
label	`varchar(8)`	
classifier	`varchar(32)`	

B CITIES OF TWEETS COLLECTION

Population was retrieved from the U.S Census Bureau and is stated as of 2014. Population is rounded to thousands.

Tab. B.1: Cities of tweets collection

ID	City	State	Population	Radius	Note
ALB	Albuquerque	NM	554,000	13 km	higher criminality
ATL	Atlanta	GA	448,000	11 km	
BIL	Billings	MT	109,000	6 km	
BND	Bend	OR	166,000	7 km	higher unemployment
BOS	Boston	MA	646,000	7 km	
CHI	Chicago	IL	2,700,000	14 km	
CHR	Charlotte	NC	793,000	16 km	
CLE	Cleveland	OH	390,000	8 km	higher burglary and robbery
DAL	Dallas	TX	1,300,000	17 km	
DEN	Denver	CO	649,000	12 km	
DET	Detroit	MI	689,000	11 km	higher criminality
HOU	Houston	TX	2,200,000	23 km	
IND	Indianapolis	IN	843,000	18 km	
JCK	Jackson	MS	173,000	10 km	
LA	Los Angeles	CA	3,900,000	20 km	
LTR	Little Rock	AR	179,000	10 km	
LV	Las Vegas	NV	584,000	20 km	
MEM	Memphis	NC	647,000	17 km	
MIA	Miami	FL	399,000	23 km	
MIL	Milwaukee	WI	599,000	9 km	
MIN	Minneapolis	MN	400,000	7 km	
NYC	New York	NY	8,500,000	16 km	
OKC	Oklahoma City	OK	646,000	23 km	
OMA	Omaha	NE	424,000	11 km	
PHIL	Philadelphia	PA	1,600,000	11 km	
PHX	Phoenix	AZ	1,500,000	21 km	
SEA	Seattle	WA	652,000	9 km	
SJ	San Jose	CA	1,000,000	13 km	
STL	St. Louis	MO	318,000	8 km	higher property crime
WTR	Waterbury	CT	110,000	6 km	
YAK	Yakima	WA	247,000	6 km	
YUM	Yuma	AZ	201,000	9 km	

C CONTENTS OF THE DVD

```
/
├── Applications/
│   ├── gui/
│   │   ├── GUI.py
│   │   ├── main.glade
│   │   └── PieChart.py
│   ├── libs/
│   │   ├── scikit-learn-0.15.2/
│   │   ├── TwitterAPI/
│   │   └── MySQL-for-Python-3/
│   ├── Baseline.py
│   ├── config.py
│   ├── Database.py
│   ├── Ekman.py
│   ├── emotions.py
│   ├── MinerBase.py
│   ├── miner.py
│   ├── runminer.sh
│   ├── TweetMiner.py
│   ├── TweetUtils.py
│   └── TwitterEmotions.py
├── Data/
│   ├── mined_tweets.sql.gz
│   ├── mysql_database_structure.sql
│   ├── mysql_initial_data.sql
│   ├── SVM-linear-model-emotional.pkl
│   ├── SVM-linear-model-valence.pkl
│   ├── tweet_labels_annotated.sql.gz
│   ├── tweet_labels_classified.sql.gz
│   └── VirtualMachine.ova
└── Documents/
    └── Social Media Analysis using Pattern Recognition.pdf
```

D INSTALLATION & REQUIREMENTS

D.1 Requirements

- Python 3+
- MySQL 5.5.3+

Python libraries:

- setuptools[1]
- urllib3
- requests_oauthlib
- numpy
- scipy
- MySQLdb[2]
- sklearn (Scikit) 0.16+[2]
- ntlk[3] [2]
- *python3-gi*
- *python3-gi-cairo*

D.2 Installation

Installation was tested on Ubuntu 14.10 and Debian Wheezy.

```
# apt-get install python3 python3-setuptools python3-dev\
python3-urllib3 python3-pip python3-numpy python3-scipy python3-gi \
python3-gi-cairo
# pip3 install requests_oauthlib
```

```
# apt-get install mysql-server libmysql-dev
```

```
$ cd {DVDroot}/Application/libs/MySQL-for-Python-3
$ python3 setup.py build
$ sudo python3 setup.py install
```

Use the same steps to install the `TwitterAPI`, `scikit-learn` and `nltk`.

D.3 Configuration

1. Open the `config.py` file, located at `Application/`.
2. Set the Consumer key and secret, obtained from Twitter in the application registration.
3. Configure the database credentials.
4. Import the database structure from file `mysql_database_structure.sql`, located in `Data/`.
5. Import initial data to the database from file `mysql_initial_data.sql`
6. Copy SVM models from `Data/` to `Application/models/`.

[1] for building and installing other python's libraries
[2] bundled
[3] Corporas are downloaded separately.