Deusto

Facultad de Ingeniería
Universidad de Deusto

Ingeniaritza Fakultatea
Deustuko Unibertsitatea

Grado en Ingeniería Informática
Informatikako Ingeniaritzako Gradua

Proyecto fin de grado
Gradu amaierako proiektua

Automatic detection of repressed anger from text messages

Jesús María Sesma Solance

Director: Takashi Yukawa

Nagaoka, Mayo de 2017

Abstract

The growth in the usage of social media, microblog and review platforms has resulted in a significant increase of the access to short text messages that reflect individual's opinion and feelings. Automatic detection of people's emotions has a wide range of applications such as producing systems that measure the satisfaction of customers and thus help companies to improve their products or services. This research project focuses on detecting anger, an emotion that is relative difficult to detect compared to other sentiments due to the usage of linguistics figurative language techniques, such as irony, that intends to communicate the opposite of what it is literally said. To this purpose, a review of the state of the art has been made and an experiment using not only traditional machine learning techniques, such as Neural Networks, K-Nearest Neighbor and Support Vector Machines, but also Deep learning algorithms has been conducted in an open social network like Twitter. The proposed methods define the repressed anger detection as a classification problem and to solve it, the task is divided into two subtasks that complement each other. The first focuses on explicit anger detection, while the second subtask's goal is to detect irony. The system make use of selected featured based on characteristics properties of English language and studies emotion and irony in psychology. The model is composed on features such as: frequency of words, style in written and spoken languages, intensity of adverbs and adjectives, structure of the document, use of emoticons, synonymy, ambiguity and the contrast of sentiment and negative situation.

Keywords

Emotion Analysis, Supervised Classification, One-vs-one Classification, Convolutional Neural Networks, Repressed Anger Detection

Contents

List of Figures

Chapter 2

Chapter 3

List of Tables

Chapter 3

Algorithm Index

1. INTRODUCTION

2. PLANNING AND METHODOLOGY

In order to accomplish the Master's thesis during the established period of eight months, the following plan will be followed:

1. **Exploratory phase:** this period of time will consist in searching and studying the literature related to the research field in order to build a solid theoretical framework on which support the rest of the investigation. Even though this process is presented as a isolated task, it is an fundamental activity that will be executed continuously throughout the whole project.
2. **First contact with the technology:** after having reviewed the first phase of the state of art and checked the advantages or limitations of each architecture, the first empirical tests will be done. These experiments will serve to for a better understanding of the literature read in the exploratory phase and to observe how they behave with the gathered sample data.
3. **Specification and design of the solution:** in this phase, based on the knowledge gathered from the empirical tests in the last point and the relevant literature that has been constantly read, the requirements of the solution that best results can achieve for this research topic will be specified.
4. **Design of the functional prototype and evaluation:** after the requirements gathering, during this task, a prototype that justify all the steps followed this far will be developed, and evaluate its performance.
5. **Thesis document writing:** Finally all the relevant knowledge that has been required to complete this research will be gathered, the procedure to develop the system will be described and the conclusion and future work on this topic will be written down altogether, resulting in this master's thesis document, that will be refined until its laster submission and defense.

Previously defined tasks will be executed according to the next schedule diagram:

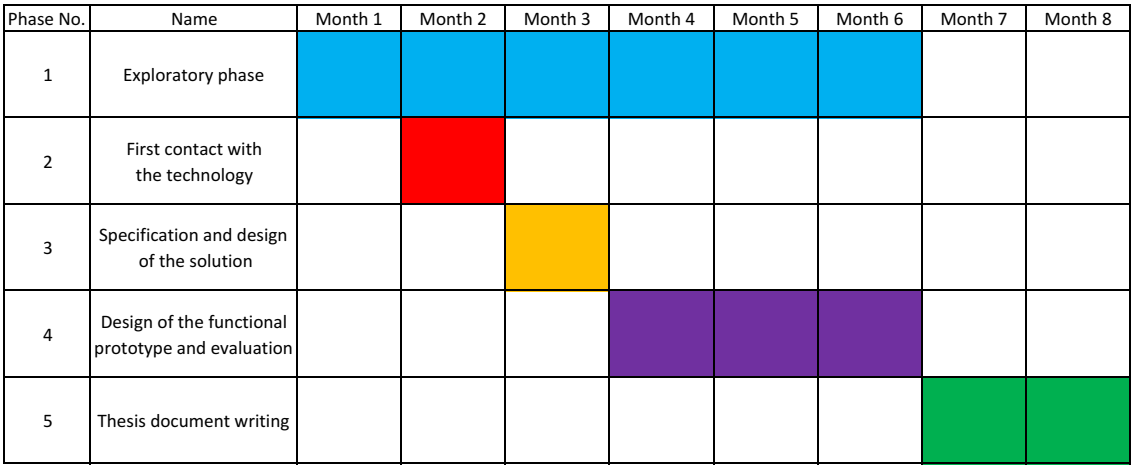

Figure 2.1: Schedule

The figure 2.2 illustrates the methodology used during the investigation process, in which the results obtained from the prototypes might not be satisfactory or could lead to formulate new questions. To solve those doubts a redesign of the system may be needed. In addition, after reporting the results to project supervisor, new relevant information might be provided. This information has to be added to the knowledge obtained from previous state of the art review and reanalyze if the proposed solution requirements are still valid.

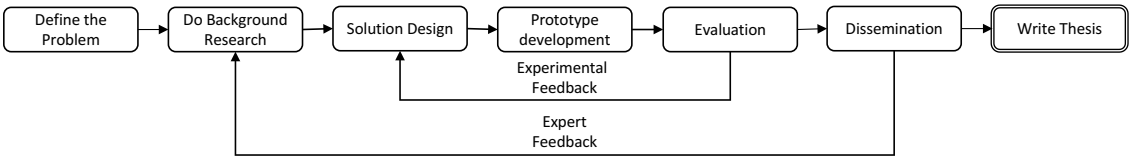

Figure 2.2: Methodology

3. STATE OF ART

Main activity: Paper reading

Introduction to Sentiment Analysis. Relevant International Workshop found: SemEval Datasets repositories: Twitter as corpus, Amazon products reviews Dataset labeling: CrowdFlower, Amazon Mechanical Turk. Word embedding: Word2vec, GloVe. Text classification features: N-grams based on Sentiment lexicon Learning Methods: Supervised, Unsupervised, Distant Deep Learning techniques: Convolutional NN, Recurrent NN, Deep Autoencoders Classifiers: Bag of Words, Naive Bayes, Logistic Regression, SVM, NN, Clustering Accuracy evaluation: Precision, Recall, F1-Score, Cross-validation Trends: Best scoring teams used Deep Learning techniques.

Differences between Sentiment and Emotion Analysis

SemEval works on sentiment analysis, which focuses on binary or multiclass classification, being Positive, Neutral and Negative the most common one. This means: None of the SemEval preprocessed datasets are valid for me. (Raw data may work for manual annotation) The need of procedure adaptation to fit my research domain. E.g. Use Emotion lexicon, such as EmoLex or LIWC, instead of those related to Sentiment. Need to focus the research on discover word features that help to classify anger in texts. E.g. Heavy Punctuation, Quotation Marks, Emoticons; Unigrams, among others.

Next week work: Deep reading and prepare a Playground with small sample data.

Read fundamental theory about classifiers such as SVN, Bayesian, KNN, Decision Tree. Urge test what has been learned in order to set the basic knowledge. Download and test WEKAs basic functionalities for educational and research purpose. (Includes a No-Code-GUI, which makes it user friendly for initial testings) Use scikit-learn for following iterations of the classifier (More modern algorithms, better memory management, Python development environment...) Further reading about Emotion classification techniques and investigate about repressed anger in English texts. Multiclass angry classification?

Future work: Deep Learning

Depending on the generated dataset (size and labeled/unlabeled) the project could opt for a DL based classifier. Could score better results in the benchmarking. To gather large amount of data for each class is compulsory solve the overfitting problem. (Detection and prevention by using Cross-validation) If the dataset is not completely labeled weakly (distant) supervised learning could be applied. Theano and Keras have proved a DL frameworks to perform well.

3.1. CLASSIFICATION TECHNIQUES

The aim of this section has two purposes. The first one, is to make an introduction of the basic concepts of classification, which is essential for the detection of repressed anger. The second one,

is to explain how the algorithms used in this study work.

3.1.1 Fundamentals of Classification

According to [23], classification can be defined as the task of predicting an outcome from a given input. This outcome is produced by the process of mapping a group of characteristics present in the input to a certain category. In other words, it consists in assigning objects (the input) to one of several predefined classes (the outcome) [19]. Examples of classification can be found in everyday life, such as e-mail spam detection, news classifiers, Optical Character Recognition (OCR), animal kingdom classification (see Figure 3.1), among many others.

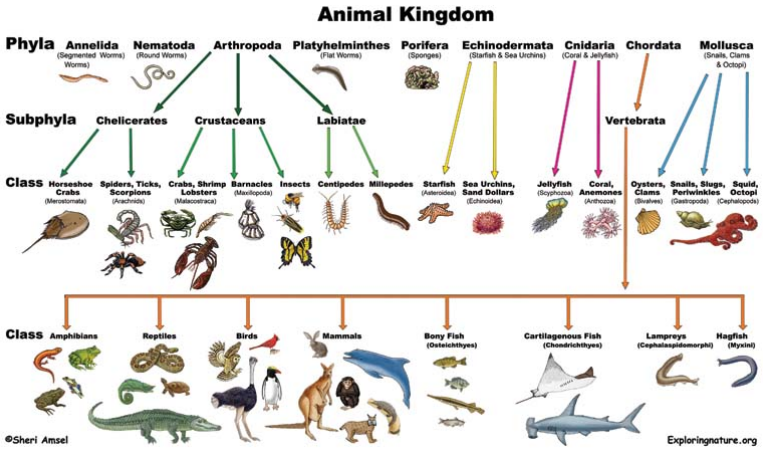

Figure 3.1: Classification of animals. The image is extracted from Exploring Nature.

The input data for a classification task is composed by a collection of records, the dataset. In the same time, each record, also known as an instance, is composed by a set attributes. From all these attributes there is one considered special, which is called the target attribute or the class label. Regular attributes can be both discrete or continuous values. For the values signed for the class label, however, they must be discrete. This characteristic is what distinguishes classification form regression. Table 3.1 shows a sample dataset for animal classification into the following categories: amphibian, bird, fish or mammal.

Tan Pang-Ning et al. propose a more mathematical definition of classification stating that it is the process of learning a target function f, also known as classification model, that maps each attribute set x to one of the predefined class labels y.

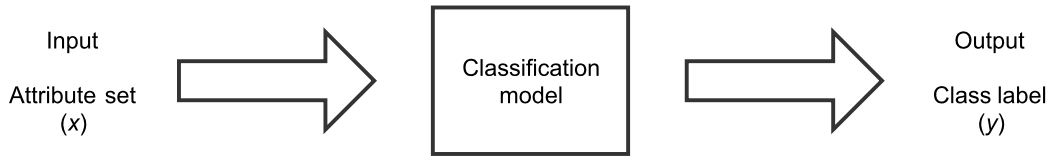

Figure 3.2: Classification as a task of mapping a set attributes x into its fitting class label y.

A classification model is useful for the following purposes [19]:

Common Name	Hair	Feathers	Eggs	Milk	Aquatic	Legs	Class Label
antelope	Yes	No	No	No	No	4	mammal
catfish	No	No	Yes	No	Yes	0	fish
dolphin	No	No	No	Yes	Yes	0	mammal
dove	No	Yes	Yes	No	No	2	bird
duck	No	Yes	Yes	No	Yes	2	bird
elephant	Yes	Yes	No	Yes	No	4	mammal
flamingo	Yes	Yes	Yes	No	No	2	bird
frog	No	No	Yes	No	Yes	4	amphibian
fruit bat	Yes	No	No	Yes	No	2	mammal
gull	No	Yes	Yes	No	Yes	2	bird
herring	No	No	Yes	No	Yes	0	fish
kiwi	No	No	Yes	No	No	2	bird
lark	No	Yes	Yes	No	No	2	bird
lynx	Yes	No	No	Yes	No	4	mammal
mole	Yes	No	No	Yes	No	4	mammal
mongoose	Yes	No	No	Yes	No	4	mammal
newt	No	No	Yes	No	Yes	4	amphibian

Table 3.1: Animal kingdom dataset.

- **Descriptive Modeling:** Since a classification model presents the main features of the data, it can serve as an explanatory tool to distinguish between instances of different categories [18].
- **Predictive Modeling:** A classification model can also be used to predict the class label of an unknown new instance. As shown in Figure 3.2, a classification model can be represented as a black box that automatically assigns a class label to an instance by providing its attribute set.

It is important to remark that classification techniques perform their best when used for predicting or describing datasets which its class label is binary or nominal, Since they no consider properties such ordinality or the implicit order among the categories, they become ineffective with ordinal class labels [7].

3.1.2 General classification problem solving

For general classification problems solving, popular techniques consists on a process that starts with building classification models from a sample dataset [27]. Each technique depends on a learning algorithm witch is in charge of generating the classification model. A good model should define the relationship between the input attribute set and its belonging category that suits the best. Therefore the model should be valid for both, the sample data used to generate the model and also for new unknown instances. Among popular classification techniques Support Vector Machine (SVM), Neural Networks (NNs), Naive Bayes or Decision Trees (DTs) can be found [9].

As shown in the Figure 3.3, to solve a classification problem a sample dataset must be provided as a training set. This sample is used to build the classification model according to the learning algorithm. After the model is built, it is applied to unlabeled dataset, also called the test set, to predict the categories of each instances of the records. To measure how good the model is, there is only need to count the number of instances have been correctly and incorrectly classified from the test set. Usually, to represent system's performance values, a confusion matrix is used [11].

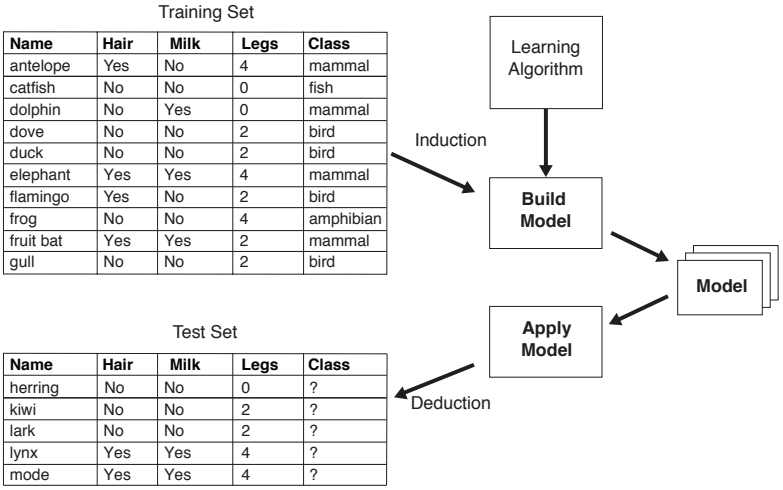

Figure 3.3: General approach for classification model building and new instance category prediction.

		Predicted Class	
		$Class = Yes$	$Class = No$
Actual Class	$Class = Yes$	a	b
	$Class = No$	c	d

Table 3.2: Confusion matrix of a binary classification.

As a example, Table 3.2 represents the confusion matrix of a binary classification problem.

4. DEVELOPMENT

SMILE Twitter emotion dataset[24]

"If you provide Content to third parties, including downloadable datasets of Content or an API that returns Content, you will only distribute or allow download of Tweet IDs and/or User IDs."[22]

EMOTEX[12]

Twitter sentiment datasets are publicly available in CrowdFlower's "Data for Everyone" collection[5].

Wang dataset[25]

As stated on last sub-meeting, SemEval Workgroup's[21] labeled dataset doesn't work for the scope of this research. They focus on classifying polarity of texts messages (positive, negative and neutral classes) Time spending activity: Search for emotion related dataset. Found SMILE dataset, containing 3085 Twitter messages. Classifies tweets into the following 7 classes: Anger, Disgust, Happy, Sad, Surprise, No-code and Not-relevant.

SMILE Dataset problems

Very small dataset. More that 50% of Neutral/"Garbage" data. Multilingual tweets found (English, French, Japanese, among others.) Almost 2% of the data refers to Anger

Conclusions: Not appropriate or enough for a decent supervised learning approach.

However English Angry tweets will be extracted for further use.

[Dataset class distribution image]

Research: Building automatic labeled corpus

EMOTEX & SocialCom: search for tweets with Emotion hashtags from Circumplex model and WordNet Synsets.

[image of EMOTEX system to automatically obtain labeled]

Common Anger Hashtags: #angry, #annoyed, #annoying, #bothered, #frustrate, #fury, #furious, #irritating, #mad, among many others.

Twitter has made very specific restrictions on redistributing Twitter data, which means that available datasets don't include Tweet text, just a unique identification number.

Retrieve tweet text: python Tweepy[1] API's .get_status(id) method should do the work.

CrowdFlower public emotion dataset found!

[Data for everyone image]

Crowdflower emotion dataset statistics

40,000 rows of data. Each row represents a tweet. 13 labels (classes) Classification: 110 as Anger and 1311 as Hate. Sum: 1421 (3%)

Conclusions: Might be a good sample to do a binary classification test:

E.g. 1421 Anger / 1421 Not Anger (others)

Understanding WEKA ARFF file format

Up to now, downloaded datasets were stored as JSON, CSV or Plain Text files. However, WEKA works with a proprietary format file called Attribute-Relation File Format (ARFF).

Completed Task: Convert downloaded CSV and JSON datasets to ARFF and load them into WEKA to check for possible conversion errors.

(python liac-arff module)

[ARFF file format explanation]

Some simple tests done

Testing WEKA's StringToWordVector to convert ARFF file's tweet messages into more machine learning friendly numerical attributes. (Combined with rainbow stopwords, stemmers, n-gram tokenizers) Reading about how Random Forest algorithm works and use it to make a rough training and cross-validation classification with random selected features. (5 fold cross-validation results 28.49Resample (Splitting original dataset into, training, validation and testing sets) Weka's Attribute selection to predict best feature subset (wrappers and rankings) without overfitting.

Next week work

As Crowdflower dataset is composed of 40,000 instances, the resulting matrix from executing WEKA's StringToWordVector tool generates over sparse 10,000 features/attributes without filtering. Classification of a such a big dataset using SVM or such algorithms takes over 2 hours in generating just the model (on a macbook pro laptop). Therefore previously proposed 2,841 instances (1421 Anger / 1421 Not Anger) dataset is going to be built in order to be able to test those algorithms in reasonable time.

At the same time while testing new algorithms, information about how they work is going to be read, to understand if the obtained result are the expected.

Continue investigating WEKA and the useful tools it provides for text classification.

Crowdflower emotion dataset is unbalanced

Binary Classification Dataset Generation

Review Anger and Hate classes and analyse the viability of merging them into a single class. Checked both classes messages, they contain similar information. Therefore the merge is feasible. Transform the leftover classes into no anger class to perform binary classification. To solve the unbalanced dataset problem there 2 possibilities were studied: Use Stratified K-Folds. Undersampling the dataset to a more suitable class appearance ratio. (Random permutation of [No. angry tweets] without replacement) Merge both classes randomly. Text preprocessing (remove URLs, user mentions, etc.)

Last reviews result: 40,000 instances. Word2vector, 1-3 n-grams[3] ($\tilde{7}$000 attrs.) Multi-class classification. Random Forest (5 CV): 28.49This reviews results: 2,866 instances. Word2vector 1-3 n-grams ($\tilde{5}$000 attrs.) Binary classification. Random Forest (10 CV): 63.89SVM
SMO (10 CV): 65.42K-NN
IBK (10 CV): 55.76NN (10 CV): Too long (cancelled)

[binary anger dataset distribution image]

Read literature about Convolutional Neural Networks for NLT[2]

Use case of CNN for Computer Vision and its translation into NLP. What is Convolution. What are Convolutional Neural Networks. Application into NLP. CNN Hyperparameters: Narrow/wide convolution. Stride size. Pooling layers. Channels.

IMPORTANT: The input layer of CNN is a sentence composed of concatenated word2vec word embeddings.

Word2Vec on ksl-xe5: Preparing for DL

[Remainder] Up to know all the experiments have been done with WEKAs GUI client and its automatic WordToVector tool. Now a test using gensim (python word2vec module) has been done for the crowdflower multilabel dataset. For the trained Word2Vec model, the GoogleNews-Vector(300)[10] has been used.

publicly available in: GoogleNews-vectors-negative300.bin.gz. Output: 40,000 instances of arrays 31 elements (length of largest sentence for dataset) of 300 features. $\tilde{1}$.27Gb (Untabbed) Spent time on research how to optimize serialization. JSON default module Avg. time: $\tilde{1}$6 mins. UJSON module Avg. time: < 2 mins.

Expanding the dataset

Found another dataset (Wenbo Wang) that contains labels and tweet id. Created a Twitter Application to have access to Twitters API. Created a Python module that download tweets based on an id and save all the relevant data into a CSV. (Havent tested serialization yet, still downloading.) This module can be expanded to become a streamer that download tweets based on emotion related hashtags, and get automatic labeled dataset as EMOTEX and SocialCom suggests.

Future Work: Create a CNN

Make a first test for a Deep Learning technique on the ksl-xe5 server. Since the server does not have graphic cards to speed up the process, I will try to establish connection with my Home server in Spain and replicate the same environment and analyze if there is any performance improvement compared to CPU only computation. Continue rating the chapter about how traditional classification algorithms work.

Some authors define sarcasm as a sub-set of irony.

Specs: CPU: Intel I7-5930K (6 core/ 12 threads) GPU: x2 GTX 980TI (5.63 TFLOPS) Storage: 1TB Samsung 950 PCI-E SSD Raid 0

Tweet downloader: Wang Dataset

Successfully tested programed module to download Wang dataset. Download tweets into CSV with the following information: tweet-id, label, author, text-content However, since dataset is formed with tweets collected from 2011, some of the are unavailable. E.g. From the test.txt, which originally contains 250,000 tweets only 65,324 has been able to be downloaded. (Took 30 hours to download)

TODO: Implement a logger to check if the tweet could not be downloaded due to unavailability or a network failure, for a download retrial.

Started implementing the CNN

Started implementing the Convolution Neural Network (CNN) by using Keras[14] python module for Deep Learning. Just need to make some adaptations to the way it parses the word2vec values to be able to test it. I also investigate how to make full use of the dual graphic card setup of my home server. According to literature I have read there are two techniques to distribute the workflow: Data parallelization Model parallelization The technologies used are based on Message Passing Interface (MPI) architecture. Apparently is not a simple task and the speed up rounds x1.3 x1.6 (last percentage being very unlikely)

Work for next week

Implement the tweet downloader logger to understand why downloads fail, and try to re-download them if network failure. Finish implementing the CNN. Do a single GPU test with home server and measure required time to perform the classification.

Tweet downloader

Implemented a basic logger that stores in real time the tweet_id, error_code and message on a file. With this simple logger I found these possible errors: Error 34: Sorry, that page does not exist. Error 63: User has been suspended. Error 88: Rate limit exceeded. Error 114: No status found with that ID. Error 179: Sorry, you are not authorized to see this status. Error 500: Internal Server Error.

Error handling (1)

For the previously presented error list there are only two errors that can be handled: Error 88: Rate limit exceeded. Error 500: Internal Server Error. Twitter API enables to download tweets for an interval of 15 minutes, after that it stops providing data for a server request cold down. To solve this error, I make the tweet downloader to wait until the server sends a download acknowledge to resume the download process and avoid getting tweets lost during the cold down interval.

Error handling (2) and logger optimization

Since the Internal Server Error can happen randomly and its duration of the error is unpredictable, the error is recorded for posterior tweet redownload. As real time loading involves serialization of small chunks of information multiple time, and the error code and message gets repeated it become inefficient and slow down the tweet download. The solution proposed consists on producing a dictionary in RAM and save it into a JSON file with the following structure after the download finishes:

$...code-error(key) : tweets : [tweet\_id, tweet\_id...] message : messagetext., No-code-error-message : tweets : [tweet\_id]...$

Measuring performance

First downloading test consist on downloading tweets from labeled tweet id list. Originally it contained 250,000 tweets. The system was able to download 65,324 tweet text in 30 hours. After handling the server cold down error the system was able to download 153,387 tweets in 69 hours. The system maintained almost the same performance after adding the having added the error logger comparing the time spent with the amount of tweets downloaded.

Deep Learning

First basic Deep Learning test implemented using Convolutional Neural Networks (CNN) on Crowdflower multilabel dataset (Keras over TensorFlow). From 40,000 tweets 70% has been used for

training and the remaining 30% for validation. Has is the first test, 3-grams, 4-grams and 5-grams have been used for the training over the 300 attribute word2vec. Every convolution generates a set of features, but they are discarded for now. This week I have been able to test on ksl-xe5 server. The results obtained on the training validation accuracy disappointing for 13 label classification, about 26.33%. It spent 4 hours 30 mins for the training task for 40,000 tweets.

Model characteristics: Model size: 300 Number of filters: 200 Batch size: 50 Number of epochs: 1000 Evaluation period: 12

Work for the next week

Run the CNN test on home server and try to run on spain university for speed up checking. Spent some time investigating test preprocessing such as English contraction expansion and misspelled words correction. Investigate how to improve accuracy in classical and deep learning technique by exploiting features.

Last notes

I have contacted the vice-dean of University of Deusto and ask about how much time I can spend in NUT for the research. Their answer is that I can stay as much time needed, following NUTs evaluation schedule. Meaning that if NUT accept it I could extend my stay in Japan for research as much I NUT allows me.

Deep Learning CPU only vs GPGPU

Last week I was able to test the implemented Convolutional Neural Network on Crowdflower multilabel dataset, based on a TensorFlow and Keras modules. Using a CPU only configuration on ksl-xe5 server it took 4 hours and 30 minutes to complete the training and validation over the 40,000 tweets. On the home server running the same test on a GPGPU configuration in conjunction with CUDA 8 and CUDA Deep Neural Network (cuDNN) v5[4], it took 2 hours 9 minutes to perform the same task. Proving that GPGPU can speed up 52.59% the process.

Model characteristics: Model size: 300 Number of filters: 200 Batch size: 50 Number of epochs: 1000 Evaluation period: 12

Test processing Investigated about expanding unambiguous English contractions to process them into word embeddings. After using a dictionary of contractions and being able to expand them I found that NLTK Stopwords would remove them anyways. A future task would be to investigate if negation is a good feature to extract and if so, remove them from the Stopwords list to the able to process them. I also investigated Oxfords Common Misspellings and PyEnchant, a python module that uses OpenOffice dictionaries to correct spelling errors. I will implement a test to check if I can correct misspellings and compare corrected words appearance with the pre-trained Word2Vect model.

Feature extraction

This week Ive been focused investigating about emotional words lexicon and emoticon lexicon. I have download available lexicons from NRC Emotion Lexicon (EmoLex) and used Wikipedias and EMOTEXs list of emoticons to create an emoticon lexicon. I have used Regular Expressions (regex) to find emoticons on tweets, before the punctuation removal. The next step would be storing them into the word embeddings to make use of them on CNN, and as a feature for classical classificators.

Future work

During the training phase on the CNN the weights are now static, therefore I will investigate how to make this weights variable. CNN only work with word embeddings, and do not support the use of other features as with classical classifiers. Due to this I will implement a SVM classifier and use the emoticons, appearance of emotion words, and other features apart from word embeddings.

Spelling check (1)

Last week I talked about a python module called Pyenchant, which provides a function suggest(word) returning an ordered list with the possible corrected words. However its accuracy is not very high. Therefore I researched new approaches and found an basic idea proposed from Peter Norvig[13], Director of Research at Google Inc. He explained that Google search engine uses probability theory to choose the most likely spelling correction for a word (argmax). argmaxc candidates P(c) P(w|c) C candidates: Candidates model. P(c): Language model. P(w|c): Error model.

Spelling check (2)

Candidate model is limited to words with up to 2 editions, correcting splits, deletes, transposes, replaces and insertion errors. The language model estimates the probability of a words by counting the number of appearance. The tests runs on a dictionary that uses Project Gutenbergs free eBooks, Wiktionary most frequents words[26] and British National Corpus (BNC) [15] (Model side 6Mb.) The error models prioritizes the original word if it is known, the the list of words known with one modification, followed by the list of words known with two modifications and finally the original word if it is unknown.

Spell checker result using the Big.txt Language Model that Norving Provides. 51% of 270 correct (35% unknown) at 37 words per second 53% of 400 correct (32% unknown) at 35 words per second

Using the Language Model I generated. 75% of 270 correct (5% unknown) at 64 words per second 69% of 400 correct (7% unknown) at 63 words per second

Making some tests

I tested spell correction with the crowdflower dataset. Checking the word that were not numerical. Whenever the spell checker modified a word I printed into the screen so I could do a manual comparison. I found that the accuracy was better than the enchant module but it continue failing in some cases and all the slang that people uses on the internet was missing in the model, therefore those kind of words were being replaced by a totally unrelated words. The next step I made was investigating about english slang used on twitter and improving the model accuracy.

Improving the model and twitter slangs (1)

Since the model does not contemplate slang terms found on tweets I have searched for a dictionary of slang with their meaning, so that it can be replaced from the original text during the preprocessing. To improve the model first I searched for some n-gram database. Google has a huge that includes 5-grams (22Gb), However the access to this resource costs 150 USD for non-members[17]. Therefore, I downloaded all the English Ebooks from Project Gutenberg as for 2014 from Kiwix harvester[16] (42Gb) publicly available from `http://download.kiwix.org/zim/gutenberg/gutenberg_en_all_10_2014.zim`. After solving encoding errors in some file names I start the eBook preprocessing.

Improving the model and twitter slangs (2)

Since all the eBooks are stored as HTML document, I removed all the HTML tags from the texts, to improve later on model generation performance, it took 1 hour 38 mins to remove all the HTML tags. I programmed a python module that take the requested model size as argument and concatenates as many eBooks until reaching requested size. The Spell checker now read the model generated from this book concatenation, but next step would be to generate a word counter and store this document as a model instead to improve even more spell checker performance.

Future work

Put all spelling related modules and slang dictionaries together and test on crowdflower dataset to check if performance improves. Continue emoticon and emotion words feature extraction. Continue implementing weight variable Convolutional Neural Network training. Continue implementing SVM classifier.

Spell checking model redesigned

On the previous I stated that I programmed a python module that removes the HTML tags from Project Gutenberg's eBooks and concatenates it into a single document to. Last week I redesigned this module. Now, the program search for word frequency files from eBooks and generates a single dictionary with the word occurrences from each document. If the word frequency file does not exists for a certain eBook it process the document by checking if it contains HTML tags and if so remove them, overwriting the original file in the process. Calculates the word frequency for that eBook and saves it into a file while at the same time updating the global dictionary that contains the words for all documents. This way if the program has to run a second time it does not have to start all the process from the beginning.

Adding twitter scope words to spelling model

After downloading 36,173 English eBook from Project Gutenberg[8], and the most general used word from wiktionary and oxford corpus I decided to add more content that its form could fit better to the written style used on twitter messages. Therefore I decided to add to the model the following list of most used words: TV programs words from Wiktionary. Film series from Opensubtitles extracted by Hermit Dave[6]. Contemporary fiction from Wiktionary.

Merge Datasets

Programmed a python module that receives as input a list of dataset paths and a sentiment target label mapper file. With this information perform a dataset concatenation, and transforms labels as the mapper file states and filter tweets that contains non-english characters, typical from Spanish, Portuguese and French, found on Wang dataset. It can also transform the dataset labels into anger and no_anger classes, used for binary classification.

Generalize Python Modules

Since all the programmed modules inputs and outputs were hardcoded, I spent time on making them variable arguments introduced on console line. This would ease me testing the programmed modules with different datasets, such as word2vec transformation and Convolutional Neural Network (CNN) classification.

Convolutional Neural Networks Binary Classification Test

My last test with CNN I used Crowdflowers dataset, consisting in 40,000 tweets with 13 sentiment target labels. The dataset was divided in 70% training 30% validation. After training the validation accuracy the system scored was 26.33%. Then I made a second test using the spelling checker

on the word2vec process. If a word could be found in the model it would check its spelling and replaced with the suggested word correction. The accuracy the system scored after apply this simple modification was 27%. For my last test I decided to change the dataset. I merged Crowdflowers dataset and Wangs test.csv and transform it into a binary class label dataset, representing 33,752 anger and 33,752 no_anger tweets. Again, using the spellchecker on word2vec process I run the CNN classification and obtained a 99.73% validation accuracy.

Convolutional Neural Networks Binary Classification Test

Since the result were so good I thought I did something wrong, and check that the merged dataset was formed correctly without repeated data. But apparently it is OK. Therefore, since I didnt have 21,000 tweets for creating proper size test dataset and I did not have time to retrain the model again and re-split the dataset, I took my remaining small train dataset I had from Wang dataset, containing 282 tweets. I made a binary dataset with that and used as test data. By using the model weight I got from the training and validation process I test the new unclassified 282 binary labeled tweets. The system achieved a 100% accuracy on the testing phase over that small dataset.

Future work

Add slang dictionaries and test crowdflower dataset to check if performance improves. Continue emoticon and emotion words feature extraction. Continue implementing SVM classifier. Split the dataset into 70% train 15% validation 15% test and re-run the CNN module and recheck the results over binary classification. Calculate Precision, Recall and F1-score.

Last week I made the first CNN accuracy evaluation that achieved 99.73%, but the test size was 282 tweets, compared to 21,200 tweets used in the evaluation, which was not enough. Due to that, I made modifications into the module that transform the dataset into word embeddings to be able to split the dataset into 70% training, 15% validation and 15% testing. Right away, I made another CNN training, and validation and testing, and confirmed that the accuracy was really of 99.73%

CNN: Speed up training phase

During last test I made I realise that most of the time of the training phase, the model was not learning anything new, after the maximum validation accuracy was reached. Therefore I spent time or investigating how to stop the training phase when no more gain was scored. I found the following relevant classes: EarlyStopping: Stop training when a monitored quantity has stopped improving (val_loss) ModelCheckpoint: Save the model after every epoch. (Save only best model) Spent time comparison: Before modifications: $\tilde{9}$ hours 30 mins. After modifications: $\tilde{1}$0 mins. (Stopped after 22 Epoch)

Confusion Matrix and Scikit evaluation

To evaluate that what I was doing was right I created a confusion matrix with the method Keras library provided, plus what I researched about how to make diagrams with matplotlibs pyplot. The result was this:

[image of confusion matrix]

Scikit evaluation[20]

Since I cannot the data I got from Keras I used scikit to check if the predictions CNN make are correct. I for the confusion matrix I was also able to calculate the following evaluation metrics: Accuracy: 0.9974323524 Recall (macro): 0.9974325863 Precision (macro): 0.997434915 F1-score (macro): 0.9974323499

Cuda installation on arrived server (1)

I tried to follow my own installation guide I made several weeks ago and try to install Fedora 24. However, the combination of Skylake and Pascal architecture has made the more difficult. Skylake requires Linux kernel 4.5.5 to work. Fedora 24s open-source video drivers dont work with Pascal architecture, so after grub menu in the OS installation process the screen becomes black. Therefore I tried to Install Fedora 25, that runs wayland by default instead of X server. Nevertheless, I found that Nvidia does not support the kernel source that packs Fedora 25 and makes difficult its installation.

Cuda installation on arrived server (2)

So I decided that I would keep to my original idea to install Fedora 24. To do so I read that first I had to disconnect the Nvidia GPU out of the PCI slot. After almost 1 hour trying to pull out the GPU out of the board (It was stuck on the PCI port) I was able to run the Fedora 24 live USB and install it. This week, I will check if there is any problem with kernel version and Nvidia. But if I find any problem that makes difficult the cuda toolkit installation, my last attempt will be to install Ubuntu 16.04

Future work

Add slang dictionaries and test crowdflower dataset to check if performance improves. Continue emoticon and emotion words feature extraction. Continue implementing SVM classifier. Fix CNN accuracy lost after serializing model. Install all CUDA Servers dependencies. Search for an irony dataset to make test with CNN.

Download Irony and Sarcasm tweets Previously I read some literature about Irony and Sarcasm. Some authors define sarcasm as part of Irony. Others state that is a figurative language completely different. For my research due to the reduced amount of data I have found so far, I will deal with these term as if they were equal. With the tweet-downloader module I have been able to download tweet that contain the #irony, #ironic or #sarcasm hashtags in tweets. Up to now the system has downloaded 18,398 tweets, which I still need to preprocess and filter.

Preprocess Irony Dataset I used a Google Sheets extension called Twitter Archiver that remotely downloads tweets and saves it into a Google Sheets document. As I said on last group meeting, I was downloading English Tweets that contained #irony, #ironic and #sarcasm hashtags. I have managed to download a total of 34.000 tweets (still downloading) I need to remove useless columns from the CSV, rename columns names to be able to use with my python modules, remove duplicate tweets documents (Retweets) and remove these 3 hashtags from the text. I created a python module that performed all the tasks from last point.

Perform Irony Dataset CNN Test

Dataset characteristics: Instances: 56,264 tweets. Balanced 50% irony (28,132) 2 target labels, (binary) Split: 70% train. 30% val. Val. Acc: 91.88% (w/u spell) Val. Acc: 91.86% (w. spell. chckr)

Recall (macro): 0.9188087708 Precision (macro): 0.9188862312 F1 (macro): 0.9188298366

Note: With the spell checker the CNN losses accuracy for this dataset. To implement a more complex spelling checker would be needed, + other techniques (stemming, lemmatization, synonyms).

Model characteristics: Model size: 300 Number of filters: 200 Batch size: 50 Number of epochs: 1000 Evaluation period: 12

Found a bug in Anger Dataset

Anger dataset is generated by merging the Crowdflower dataset and Wang test dataset. When I created the dataset I checked that Crowdflower tweets didnt contain hashtags such as #anger, #rage, etc. However I forgot to check that on the Wang dataset, which DO contain them. I modified the dataset merger module to find those hashtags and remove them from the tweet content. After doing it a performed again the CNN to recheck the results:

Dataset characteristics: Instances: 67,504* tweets. 2 target labels Acc: 99.64$\tilde{9}$9.74%

Dataset characteristics: Instances: 70,108 tweets. 2 target labels Acc: 95.92% (w/o spell) Acc: 96.15% (w. spell chkr)

Note: Used less tweets on first test because I removed all the duplicated tweets. Now the merger maintains the first instance. (fixed)

Future work Investigate how to combine both anger and irony classificators and do a final test with that classifier. (Ensemble Learning?)

Continue implementing SVM classifier. Add slang dictionaries and test crowdflower dataset to check if performance improves. Continue emoticon and emotion words feature extraction. Fix CNN accuracy lost after serializing model.

CNN module After training and validating the model, serializing it and loading the weights for posterior executions would produce a considerable accuracy lost on the classifier ($\tilde{5}$1%). This issue has been already fixed. The possibility of storing the models and make the weights reusable enables to program successive module that uses all the pre-trained classifiers to combine their results to detect repressed anger tweets. Fixed a bug with the Confusion Matrix Drawer, not rendering the normalized version of the matrix properly. Standardized the output to easily be able to use the results into the next module that combines all classifiers.

Read literature (Model combination)

Since I have two classifiers that can detect anger and irony independently, I have read some papers about the techniques used to combine the classifiers to be able to use the to detect repressed anger. First I thought about Ensembled Learnings Divide and Conquer technique, with a voting combination rule. However after Asking to Yukawa sensei he proposed to build a 2x2 matrix to classify the output as following (right):

[Image of divide and conquer technique]

[image of 2x2 combination matrix]

I started programming the module that combines the classifiers. It already loads the classifiers. However I have to preprocess the dataset on a different way to proceed with the prediction and combination.

[image of the merge to word2vec process]

Note: Combined test dataset fragment is used in the final steps prediction phase.

Question: Can the binary anger and binary irony datasets no_irony and no_anger have tweets in common?

Finished dataset preparation

[image of the merge to word2vec process]

Implemented the dataset preparation system that I presented last week. Around unique 220,000 gathered after merging all datasets. Re-splitted dataset into anger and irony binary dataset with no common tweets with 67,746 and 56,264 tweets respectively. (removing key hashtags) Resulting dataset were divided into 73% train, 18% validation and 9% test. Anger and ironys tests dataset were merged generating a the final prediction file of 11,060 tweets.

Pre-training the classifiers and prediction

Resulting train and evaluation word embeddings are used for classification learning process, generating model structure and model weights files. Test word embeddings are used to generate an approximate performance record sample. With the model structure and weights and the merged test dataset, a CSV with the prediction result is generated. A evaluation method for this last step is still missing.

[image from word2vec to final prediction process]

Searching for evaluation method

[image of excel with the classification labels, half of the prediction can be done]

To validate the prediction, Yukawa sensei suggested to perform a manual classification of the dataset

Dataset manual classification

As doing a single classification is not reliable, I created a program with google app script that automatically creates google forms based on the dataset data.

I will send to some friends to help me with this task, in order to improve the reliability of the manual classification.

[various images of google tech, code, forms and results]

Doubt: Using hashtags or remove them

Example 1:

I #hate when I have to meet my neighbors in the morning. (in sentence natural usage)

Example 2:

I had to explain my parents once again how to use the refrigerator due to Alzheimer #sad (out sentence usage for classification)

The second usage case alters the results of the classification. But since I cannot differentiate between all the tweets of the dataset, I think that I should remove all the dataset, (even if some of the sentences lose meaning) because the second case (majority) is faking the result of the learning process.

Re-run the learning process (no hashtags)

After re-running the learning process of both classifiers and check their individual accuracy score it when down to 78% and 84% for anger and irony respectively

Dataset manual classification

Since I generate a new random dataset with no hashtags I had to start the manual classification again.

Manually classified 300 tweets by myself.

Collaborators have classified 100 tweets (validation).

My goal is to have at least 1000 tweets classified and validated.

Processing the form results (I)

Background: In the forms I asked 2 questions: (I) to classify the sentence and (II) if the sentence was from the irony tweet downloader task I also ask them if the tweet was really ironic instead of talking about irony. This would help me to correct the misclassified tweets in the automatic process. Difficulties found by the classifiers: Some of the tweets are answers to other messages, without that context is difficult to classify current tweet properly. To check if a tweet is ironic, on some tweets they lack information about the topic to judge properly. After finished my part of the manual classification, while the collaborators keep sending me more form responses, I started programming the module that processes the results and incorporates them into my CSV for later use in the validation of the system.

[image of generated excel from google form responses]

Results

From the 11,074 tweets dedicated for the prediction phase 1,000 were manually classified. From the obtained responses only those tweets that more than half of the submitters were chosen, the rest (176) were considered ambiguous. In the end the amount of tweets available for the evaluation process were 824 with the following distribution: Explicit anger: 193, Irony: 47, Repressed anger: 44 and Normal: 546

[CHECK WHY AMBIGOUOUS NUMBER IS 176 BUT 830 TWEETS ARE IN DATASET INSTEAD 824]

Accuracy: 0.5144578313 Recall(macro): 0.4695591165 Precision(macro): 0.3846361992 F1-Score(macro): 0.3871343406

Trying a new pre-trained word2vec model (I)

Up to know I have been using Googles publicly available 300 feature, pre-trained word2vec model that has been trained from the english text extracted from Google News. However, this news tend to be written in a style that differs from how tweet are. Usually tweets contain terms used in spoken English that do not appear in the news. This is the reason why a important amount of theses words are not presented in the Googles model. Apart from trying to correct the word spelling and use a slang dictionary, using a model that have been trained with actual tweets would increase the chances of improving the amount of words used on the dataset presented in the model. I found a new model, from Fréderic Goding[**godin2015multimedi**] that has been trained with 400 millions of tweets and uses 400 features.

Not enough RAM Memory

I encounter a few problems to run the whole process with the new model in two steps: (I) generating binary anger datasets word embeddings and (II) compiling the CNN for the first time to train the anger classifier. In both processes the ram of the current GPGPU system is enough to complete these tasks. For the word2vec module, before I the GPGPU system was ordered I was working with xe5 server with 128GB of RAM. Therefore, I focus the implementation of the module in execution speed, not on memory consumption. But for the GPGPU system, since the memory is limited I optimize it as much as I could without sacrificing the initial speed design. I managed to make it work with Google word2vec pre-trained module, but for twitter it is not enough.

Memory consumption and xe5 execution time I run the word2vec over the binary anger dataset on xe5 and check its memory consumption check the amount of memory need to complete that task. The maximum peak was around 43GB of RAM. Since in this process only CPU and RAM are involved the execution time is more or less the same as in the GPGPU system. For the CNN compilation and training the peak was 30GB of RAM on xe5. Even if the GPGPU does have 31.3GB of memory available, it does still crash. Forums on the internet refer to this error to not having enough RAM or powerful CPU. xe5 expent almost 2 hours to finish the training of the CNN. When the same task with binary irony dataset, that luckily does work under the amount of RAM ($\tilde{8}$,000 tweet less) available usually takes around 6 minutes.

Results comparison Accuracy: 0.5626506024 Recall(macro): 0.4403748582 Precision(macro): 0.3916296856 F1-Score(macro): 0.390081225

[images of normalized confusion matrix from google and twitter model]

5. RESEARCH FRAMEWORK

6. RESULTS

7. CONCLUSION AND FUTURE WORK

8. SELF-ASSESSMENT

Bibliography

[1] bliti. *An easy-to-use Python library for accessing the Twitter API*. URL: http://www.tweepy.org (visited on 12/10/2016).

[2] Denny Britz. *UNDERSTANDING CONVOLUTIONAL NEURAL NETWORKS FOR NLP*. 2015. URL: http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/ (visited on 02/12/2016).

[3] William B Cavnar, John M Trenkle et al. 'N-gram-based text categorization'. In: *Ann Arbor MI* 48113.2 (1994), pages 161–175.

[4] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro and Evan Shelhamer. 'cudnn: Efficient primitives for deep learning'. In: *arXiv preprint arXiv:1410.0759* (2014).

[5] *CrowdFlower: Data for Everyone*. URL: https://www.crowdflower.com/data-for-everyone/ (visited on 06/03/2016).

[6] Hermit Dave. *Invoke IT Limited: Frequency Word Lists*. URL: https://invokeit.wordpress.com/frequency-word-lists/ (visited on 28/11/2016).

[7] Eibe Frank and Mark Hall. 'A simple approach to ordinal classification'. In: *European Conference on Machine Learning*. Springer. 2001, pages 145–156.

[8] *Free ebooks by Project Gutenberg*. URL: http://www.gutenberg.org (visited on 01/12/2016).

[9] GV Garje, Apoorva Inamdar, Apeksha Bhansali, Saif Ali Khan and Harsha Mahajan. 'SENTIMENT ANALYSIS: CLASSIFICATION AND SEARCHING TECHNIQUES'. In: (2016).

[10] *Google Code: word2vec*. 2013. URL: https://code.google.com/archive/p/word2vec/#Pre-trained_word_and_phrase_vectors (visited on 23/11/2016).

[11] Howard Hamilton. *Confusion Matrix*. 2000. URL: http://www2.cs.uregina.ca/~dbd/cs831/notes/confusion_matrix/confusion_matrix.html (visited on 21/10/2016).

[12] Maryam Hasan, Elke Rundensteiner and Emmanuel Agu. 'Emotex: Detecting emotions in twitter messages'. In: *Academy of Science and Engineering (ASE)* (2014).

[13] *How to Write a Spelling Corrector*. URL: http://norvig.com/spell-correct.html (visited on 27/11/2016).

[14] *Keras: Deep Learning library for Theano and TensorFlow*. URL: https://keras.io (visited on 06/12/2016).

[15] Adam Kilgarriff. *BNC database and word frequency lists*. 1995. URL: http://www.kilgarriff.co.uk/bnc-readme.html (visited on 27/11/2016).

[16] *Kiwix: Project Gutenberg*. URL: http://wiki.kiwix.org/wiki/Project_Gutenberg (visited on 01/12/2016).

[17] *Linguistic Data Consortium: Web 1T 5-gram Version 1*. 2006. URL: https://catalog.ldc.upenn.edu/LDC2006T13 (visited on 28/11/2016).

[18] David Madigan. *Descriptive Modeling*. 2002.

[19] Tan Pang-Ning, Michael Steinbach, Vipin Kumar et al. 'Introduction to data mining'. In: *Library of congress*. Volume 74. 2006.

[20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay. 'Scikit-learn: Machine Learning in Python'. In: *Journal of Machine Learning Research* 12 (2011), pages 2825–2830.

[21] *SemEval Portal*. URL: http://www.aclweb.org/aclwiki/index.php?title=SemEval_Portal (visited on 03/09/2016).

[22] *Twitter Developer Documentation: Developer Agreement & Policy*. URL: https://dev.twitter.com/overview/terms/agreement-and-policy (visited on 21/10/2016).

[23] Fabricio Voznika and Leonardo Viana. *Data Mining Classification*. 2007.

[24] Bo Wang, Maria Liakata, Arkaitz Zubiaga, Rob Procter and Eric Jensen. 'SMILE: Twitter emotion classification using domain adaptation'. In: *CEUR Workshop Proceedings*. Volume 1619. Sun SITE Central Europe. 2016, pages 15–21.

[25] Wenbo Wang, Lu Chen, Krishnaprasad Thirunarayan and Amit P Sheth. 'Harnessing twitter" big data" for automatic emotion identification'. In: *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom)*. IEEE. 2012, pages 587–592.

[26] *Wiktionary:Frequency lists*. URL: https://en.wiktionary.org/wiki/Wiktionary:Frequency_lists (visited on 28/11/2016).

[27] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

Acronyms

BNC British National Corpus. 14

DT Decision Tree. 7

NN Neural Network. 7

OCR Optical Character Recognition. 6

SVM Support Vector Machine. 7