CMPUT 551 - Project Report
Sentiment Analysis In Figurative Language
(Using Twitter Data)

Adam St. Arnaud	-	ajstarna@ualberta.ca
Alexandr Petcovici	-	apetcovi@ualberta.ca
Janek Goergens	-	gorgens@ualberta.ca
Robert Post	-	rpost@ualberta.ca
Sankalp Prabhakar	-	sankalp@ualberta.ca

Abstract

In this project we address the problem of sentiment analysis in figurative language using data from the popular microblogging service Twitter. We process the Twitter data and extract features to build a model which predicts a real-valued sentiment score for a tweet. The data for this task was made available from *SemEval 2015 task 11*[3], which contains around 8000 figurative language tweets.

1 Introduction

1.1 The Problem

Sentiment analysis is the area of natural language processing (NLP) concerned with computationally identifying the emotion expressed by an author of a piece of text (positive, negative, neutral). Determining the sentiment of a piece of text has important implications for opinion mining, parsing users reviews and recommendation systems. Humans communicate in complicated ways and often don't strictly use literal language. Figurative language, such as sarcasm, irony, and metaphors, are quite prevalent in standard human communication. Hence, in order to create better representations of human language, systems must take figurative language into account.

1.2 Related Work

In [7] there is an effort to classify the sentiment (positive or negative) on movie reviews. However, there is no specific treatment of a figurative language. The authors of [6] propose a method to automatically collect a corpus that can be used to train a sentiment classifier which can be used to classify sentiments in tweets (positive, neutral, negative). Their classifier is based on the multinomial Naive Bayes classifier that uses N-grams and part-of-speech (POS) tagging as features. Moreover in [10] the authors study the contextual elements in discourse that could modify the meaning of words bearing opinion, thus affecting the overall polarity of a sentence. Typically, words having irony and metaphors affect the overall implied meaning of a sentence.

1.3 Motivation

Figurative language is very different from a literal language in the context that the actual meaning differs from the literal meaning of a given text or communication. Hence, the direct approaches based on words and their lexical semantics are often inadequate to predict the implied meanings. It has become quite common to use figurative language on the texts of the web, social media and general human communication. Therefore, to better our understanding, its really important to decipher the actual meaning behind a text or communication of figurative language.

2 Problem Formulation

2.1 Twitter Data

Twitter is a web platform where users can post short messages with up to 140 characters to broadcast things they want the world to know. These messages are called *tweets* and the whole Twitter system became very famous in the last years. That's the reason why there's a high interest in performing sentiment analysis on twitter data. We are provided with a set of 8000 tweets where each tweet was annotated by 7 people. Each one scored the given tweet on a range from -5 to 5 (where -5 indicates a very negative, 0 a neutral and 5 a positive sentiment). The highest and lowest scores were ignored and the average of the remaining 5 scores is given for each tweet. This data is provided by *SemEval-2015 Task 11*.

2.2 Histogram

Figure 1 shows the distribution of the sentiment scores of the tweets. Clearly most of the tweets have a negative score, so the mean score equals -1.992 with a standard deviation of 1.377.

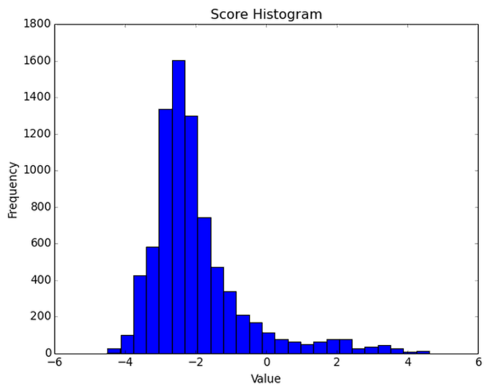

Figure 1: Histogram showing the score distribution for the given data

2.3 Pipeline

In order to have a running system where we can use our raw twitter data as an input and get an error value as output, we need to implement a couple of steps. This ends up as a chain of processes, which we define in a system pipeline model. This model is shown in figure 2.

2.4 Evaluation

We evaluate our model by calculating the L2 error between the predictions and the annotated data. The advantage of using L2 error over L1 is providing a harsher penalty for tweets that are incorrectly predicted by a large amount.
Using this method we calculated the error of random guessing to be 14.203 and predicting a neutral value of 0 for each tweet gives us 5.862. Our baseline is defined by predicting the mean value (section 2.2) of -1.992 for each tweet. This results in a baseline error of 1.896129.

3 Preprocessing

Twitter data contains a variety of linguistic phenomena such as abbreviations, slang, emoticons, spelling errors, elongated words, etc. As a result, in NLP systems which use this data to train prediction models, the same word can have multiple representations. For example the word "please" can be also represented as: "PLEASE", "pleeeese" or "plizzzz". This situation leads to feature duplication, under-fitting and to a poorly trained model. In order to solve this problem NLP systems

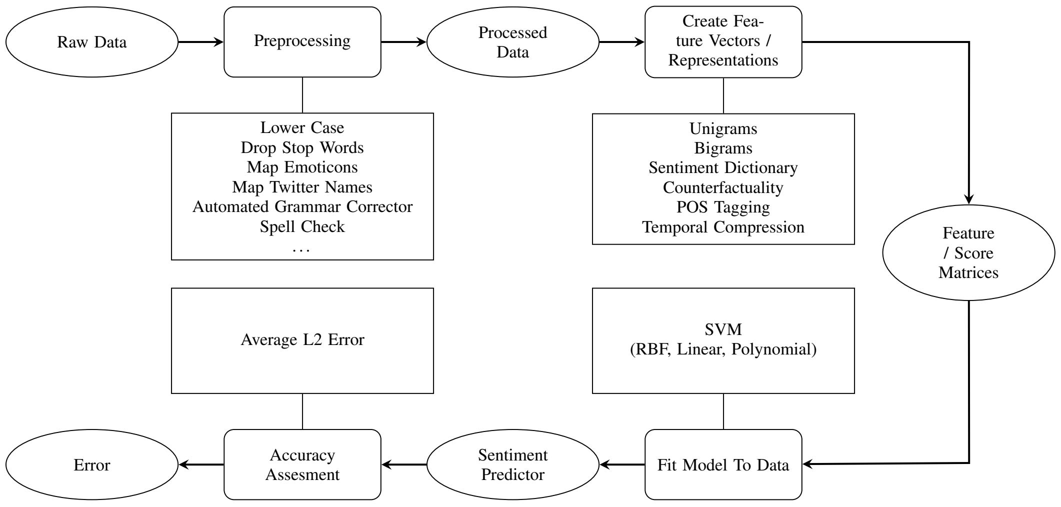

Figure 2: Pipeline Model

often perform preprocessing. The goal of preprocessing is to modify the original text in such a way as to improve feature extraction, while preserving its linguistic, semantic and sentiment characteristics. For example, after preprocessing, the words "PLEASE", "pleeeese" and "plizzzz" will all become the same word "please." At the same time, preprocessing may discard parts of a tweet which actually contain meaningful information about the tweets sentiment. For example the word "PLEASE" with all uppercase characters might express anger, and by mapping it to "please," this sentiment would be lost.
In our NLP System, we created two groups of preprocessing steps: mandatory preprocessing steps and optional preprocessing steps and each step is applied individually to each meaningful token. Tokens are words, emoticons, twitter hashtags, links, etc. In order to extract meaningful tokens from each tweet we used Ark_twokenize [5].
Next sections of the report describe preprocessing steps within each preprocessing category.

3.1 Mandatory Preprocessing Steps

Analysis of the Twitter data indicated that three mandatory preprocessing steps help improve the quality of data without potential negative side effects. These steps are: removing HTML specific characters (for example "&nbsp;"), removing non-English characters and mapping all links from Twitter text to a special token understood by our developed system. After performing these steps, the training corpus of our data was reduced from 23536 to 22774 unique tokens.

3.2 Optional Preprocessing Steps

The effect of optional preprocessing steps is definitive and can have both positive and negative effects on the original data. Full description of optional preprocessing steps is shown in table 1.

Table 1: Preprocessing steps

NAME	DESCRIPTION	EXAMPLE		UNIQUE TOKENS COUNT
		BEFORE	AFTER	
Convert to lowercase	Converts all characters from a word to lowercase	PLEASE	please	20092
Remove apostrophe	Removes apostrophe and return the original words for most common cases where apostrophe is used as vowel substitution.	you're	you are	22718
		I've	I have	
De-elongate words	Convert elongated words to normal version (works in combination with automated grammar corrector)	pleeeeeeese	pleease (please - after grammar corrector)	22640
Remove stop words	There does not exist a single definition of stop words. For our task stop words are words which do not have any sentimental load. We used the stop word list from nltk corpus except words "no" and "not"	under		22647
		before		
		very		
		to		
Map emoticons	Convert emoticons to a special token which preserves the sentimental load of the original emoticon. There are three special tokens for negative, positive and neutral sentiment.	;) =D (48 emoticons)	token_em_ pos	22722
		:(;[(18 emoticons)	token_em_neg	
		;| (54 emoticons)	token_em_neutral	
Map twitter name	Convert twitter names to a special token	@alex139	token_twitter_name	19381
Map slangs	Resolve slags to original words (5200 slangs in slang dictionary [4])	youre	you are	2059
Automated Grammar Corrector	Automatically corrects spelling mistakes.	mispelled	misspelled	18651
Stemming	Stemming is the process of reducing inflated words to their original form. For this task we used Lancaster stemmer [9]	computers	comput	16449
		computing	comput	
		computing	computerization	
Lemmatization	Similar with stemming, except return a valid and correctly spelled word	computers	compute	21814
Morphing	Returns the morphological root of the word and converts most of the verbs from any tense to their Present Simple Tense Form	were	be	21048
		had	have	
Hyponyms	Maps a given word to the category in which it belongs.	love	emotion	16859
		hate	emotion	
		dog	canine	
		wolf	canine	

Some preprocessing steps have a risk in that they can alter the sentiment of a word. For example, the Lancaster Stemmer will transform the word “fill,” which has a positive sentiment, and the word “filth,” which has a negative sentiment, to the same word “fil” [8].The same applies to Lemmatization, Morphing and Hyponyms.
One goal of our NLP System is to find the best combination of preprocessing steps which minimizes the prediction error.

4 Features

4.1 N-Grams

N-Grams are one of the most used features in NLP as they can be used to describe meaningful collections of words - a desirable trait for many contexts [7] [6]. Formally an n-gram is a contiguous sequence of n items taken from a larger sequence. In our application n-grams are built out of words.

A unigram is a n-gram for $n = 1$. In our application unigrams form the base features, and are actually the most important features in our pipeline. As an example, a tweet such as: I hate work would generate three features, (I), (hate), and (work). In the context of sentiment analysis the individual words can carry sentiment, as such the unigram (hate) can be an important feature to predict the sentiment of the whole tweet.

Bigrams are n-grams where $n = 2$. These features are all sequence pairings of words in a tweet. Again with the example: I hate work, we would generate two bigrams: (I hate) and (hate work). The intuition behind using bigrams as a feature is that bigrams like (I hate) carry a different sentiment value than the words separately. Additionally, words that dont necessarily communicate sentiment by themselves can carry sentiment when considered together, such as (this sucks). N-grams with $n \geq 3$ are not considered as the number of unique trigrams and above combinatorially explode and produce poor results.

To build a feature vector from a set of documents all containing a set of n-grams we add each gram to a bag of words dictionary. Then when a feature vector is created for a tweet we look up the unique place within that dictionary for each gram in order to determine which dimension to increment the count of that gram.

In building our feature vectors we only considered the frequency of the grams, instead of the just the 0/1 presence or the td-idf (term frequency-inverse document frequency). In doing this our n-gram implementation can also be used to pick up and count the number of special tokens we have inserted such as special mapped tokens from preprocessing, and POS tags.

4.2 Part of Speech (POS) Tagging

One of the most commonly used techniques in Natural Language Processing (NLP) is part-of-speech (POS) tagging. POS tagging can be used to determine sentiment based on the number of tagged tokens. Typically, a tweet with more nouns tends to have a more neutral sentiment than ones with more adjectives or verbs.

We studied several POS tagging tools & settled on using Tweet NLP [2], a twitter specific POS tagger. Every word or a token in a given tweet is tagged based on its part-of-speech (noun, verb, adverb, adjectives etc), some of which are twitter specific: # (hashtags), @ (at-mentions), ˜(discourse markers: RTs for retweets), U (URLs), E (Emoticons). The table 2 lists all the used tags along with their descriptions.

Table 2: List Of Tags

TAG	DESCRIPTION	EXAMPLES
O	pronoun	it you u me
N	common noun	central park
V	verb	amuse attach
A	adjective	good fav lil
R	adverb	2 (i.e. too)
D	determiner	the the it's its
P	pre or postposition	4 (i.e. for)
#	hashtags	#ualberta, #ml
@	at-mentions	@USA @Ualberta
˜	discourse markers	RT for retweets
U	URL or email address	http://bit.ly/abc
E	emoticons	:-) :(:P :O
,	punctuations	!!! ... ?!?!
!	interjections	lol haha ftw yea
ˆ	proper noun	John USA Ipad

We selected 15 of the most common tags in linguistics including the twitter specific tags. We create a feature corresponding to each tag which is the count of the number of instances of that tag in a given tweet. Only those tags with a confidence score greater than 0.9 are included in the count.

4.3 Sentiment

We used *SentiWordNet* to add features specifically related to the sentiment of the tweets. This is a dictionary, designed for opinion mining, where each of the over 100,000 words is assigned a positive and negative sentiment score. For a given tweet, we calculate a positive sum feature: $p_{sum} = \sum_i^n p_i$, where piis the positive sentiment score from SentiWordNet for the ith word in a tweet with n words. We similarly calculate a negative sum feature. If a word is not found in the dictionary, then it does not contribute to either of the two sentiment features. These two features are added to the bag of words feature vector as a real number. It is possible that different preprocessing steps could affect the sentiment score of a tweet by modifying said tweet (such as by stemming); future experimentation is required to determine the effects of different combinations of preprocessing on sentiment score.

4.4 Irony

To account for possible irony in the tweets we implement two features based on the work of Reyes et. al [10]. The features are created to detect the so-called *counter-factuality* and *temporal compression* of a tweet. Reyes et. al. determined that ironic tweets were more likely to have a high level of these two measures [10].

Counter-factuality: The first measure, counter-factuality, is focused on "discursive terms that hint at opposition or contradiction in a text, such as about, nevertheless, nonetheless, and yet." [10]. The full list of counter-factual words includes 41 entrees; there are a total of 4187 occurrences of these words in the data set, and 3123 tweets contain at least one counter-factual word.

Temporal Compression: The second measure of tweet irony that we considered is temporal compression, which focuses on words related to an opposition in time, thus indicating an abrupt change in narrative [10]. The list of temporal compression words contains 13 words such as suddenly, abruptly, and now. There are only 170 instances of a temporal compression word in the dataset, with only 103 tweets even containing a single instance.

Feature Creation: For each measure, we create a real-numbered feature based on the ratio of how many words in a given tweet possess the characteristic we are analyzing. That is, we have two ratio features $r_t = \frac{n_t}{n}$, where $t \in \{\texttt{counterFactuality}, \texttt{temporalCompression}\}$, n_t is the number of words in a given tweet that fit into category t, and n is the total number of words in the tweet.

5 Experiment Design

Our experimental test plan involved two recursive steps. An intuition phase, of selecting preprocessing steps and features that we felt would be beneficial in general, and an experimental phase where we ran our model to verify our predictions. First we ran our pipeline with several different selections of features, preprocessing steps, and SVM kernels in order to get an initial idea of the search space and produce baseline results that we could compare future iterations with. Next we took our baseline and switched each possible step on and off in order to see the effect of changing a single feature or preprocessing step. This verified our prediction that some steps would be more useful, and we concluded that several preprocessing steps would be included in every other iteration. This made sense as we felt changing the characters to lowercase, removing non ASCII, and mapping twitter names and website links to a special token would help reduce noise within our features. Once we had selected mandatory preprocessing steps we tried many different combinations of features in order to see the effects of including or excluding any one of them. In exploring this feature space, we found that using a very restrictive set unigrams produced the best results. At this stage we were able to generate every possible combination of the remaining preprocessing steps and features, over 1000 test cases. We exhaustively searched this space then we explored certain hyper parameters such as SVMs kernels, degree, and C parameter to find our optimal solution reported below.

6 Results

Table 3: Subset of our complete Result List[1]

PARAM	EXP 1	EXP 2	EXP 3	EXP 4	EXP 8	EXP 6	EXP 7	EXP 8
Expand Apostrophe	1	0	0	1	0	0	1	1
Char Repetitions	1	0	0	0	0	0	0	0
Rem. Tw. Names	1	0	0	0	0	0	0	0
Normalize to ASCII	1	0	0	1	1	1	1	1
Drop Stop Words	0	0	0	0	0	0	1	1
Tokenize	1	0	0	0	0	0	1	1
Map Emoticons	0	0	0	0	0	0	1	1
Map URLs	0	0	0	1	1	1	1	1
Map Annotations	0	0	0	1	1	1	1	1
Wordnet Morphing	0	0	0	0	0	0	0	1
Num of Unigrams	17500	0	20000	20000	15000	20000	200	193
Num of Bigrams	2500	0	0	0	5000	0	5	4
Sentiment Dict	1	1	0	0	1	0	1	1
Counter Fact	1	1	0	0	1	0	1	1
Temporal Comp	1	1	0	0	1	0	1	0
POS Tagging	0	0	0	0	1	0	0	0
Result	2.187	1.998	1.952	2.171	1.902	1.949	1.633	1.624

As a result of our extensive feature space search we found our best result listed in table 3, which had an average 5 fold cross validation error of 1.624. The best result used a very restrictive number of unigrams and bigrams, 194 and 4 respectively, and only a subset of our features and preprocessing steps. We also explored the kernel and C hyperparameters of the SVM. Table 3 reports selected examples from our extensive list of tests[1], demonstrating that our scores in general got better over time. Many preprocessing steps have been omitted as they are not descriptive for the examples provided. The majority of our results show that the number of unigrams is the most important feature,all others have minimal impact on the results. Compared to the baseline error of 1.892 our result shows an improvement over guessing the mean by approximately 14%. Our model also provides a significant improvement over random guessing or guessing neutral for all tweets.

7 Conclusion and Future Work

We found experimentally that using a restrictive set of unigrams provided the best results for figurative sentiment analysis. Many other features had little impact. Our pipeline allowed us to run many models in order to reach this conclusion. Furthermore it seems sentiment analysis of figurative language is a very difficult task, and our results are only marginally better than our baseline. Future work could analyse the unigrams that provided the best result as we used a very restrictive model. Perhaps hand selecting these unigrams and bigrams would provide better results compared to automatically selecting the ones with the highest frequency. Additionally, exploring more models such as neural nets and incorporating more parameters and features could improve our results. We anticipate the results of SemEval task 11 2015 in order to see how other teams have tackled this problem.

Acknowledgments

We would like to thank Dr. Russell Greiner and Dr. Greg Kondrak for their invaluable guidance.

References

[1] Complete final results. `https://docs.google.com/spreadsheets/d/1kWKhhIHFZuzi5Xtg_jOHE81rfpeK6b9g40oHf3JazIM/pubhtml`. Accessed: 2014-12-12.

[2] Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A Smith. Part-of-speech tagging for twitter: Annotation, features, and experiments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 42–47. Association for Computational Linguistics, 2011.

[3] UK John Barnden (J.A.Barndencs.bham.ac.uk) University of Birmingham. Semeval-2015 task 11: Sentiment analysis of figurative language in twitter. 2015.

[4] MultiMedia LLC. Internet slang translator dictionary, 2005.

[5] Myle Ott. ark-twokenize-py. `https://github.com/charlespwd/project-title`, 2013.

[6] Er Pak and Patrick Paroubek. Twitter as a corpus for sentiment analysis and opinion mining. In *In Proceedings of the Seventh Conference on International Language Resources and Evaluation*.

[7] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques. In *IN PROCEEDINGS OF EMNLP*, pages 79–86, 2002.

[8] Christopher Potts. Sentiment symposium tutorial: Stemming, 2001.

[9] NLTK Project. Source code for nltk.stem.lancaster, 2001.

[10] Antonio Reyes, Paolo Rosso, and Tony Veale. A multidimensional approach for detecting irony in twitter. *Language Resources and Evaluation*, 47(1):239–268, 2013.

Group Contribution

Robert I developed the code for using the bag of words dictionary and n-gram features within the pipeline. Adam and I designed the general flow of data in order to make adding future features simple. I ran the exploratory data analysis on the twitter and score data. I did lots of refactoring and redesigning of the general pipeline tool developed by Janek. Alex and I designed the experiment and test plan by picking the test cases, distributing them among team members, and ran them on the cluster provided by Steve - as well as compiling the results.

Adam I developed the code for error evaluation and cross validation. With Rob, I designed the general flow of data in order to make adding future features simple. I developed the code for creating sentiment features based on the sentiWordNet dictionary. I developed the code for creating irony features based on counter-factuality and temporal compression.

Sankalp I did literature surveys on different tools for part-of-speech tagging. I used the Tweet NLP POS tagger and developed code to select and map tweet tokens to their respective tags using the 15 most used tags in linguistics (including some of the twitter specific tags).

Alexandr I developed the code for preprocessing step (the complete list in the table 1 of the report). I also tested preprocessing step and made sure that they do not affect each other. Robert and I developed the testing plan, automated a part of it, performed tests on cluster, distributed test cases over team members. I performed the search for optimal parameter C of the error term in SVM regression model.

Janek I developed code to wrap all the different pipeline parts and run them. This includes a routine to read **.ini* files where we can defines switches for each preprocessing step and some other configurations. With this we were able to run multiple different combinations to compare them.
I also cared about the LaTeXfiles for this report.