UPPSALA
UNIVERSITET

UPTEC F 16012
Examensarbete 30 hp
Mars 2016

Sentiment and topic classification of messages on Twitter

and using the results to interact with Twitter users

David Jäderberg

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Sentiment and topic classification of messages on Twitter

David Jäderberg

We classify messages posted to social media network Twitter based on the sentiment and topic of the messages. We use the results of the classification to sometimes generate responses that are sent to the original user and their network on Twitter using natural language processing. A network of users who post science related content is used as the sources of data. The classifications of the dataset show worse results than others have achieved for sentiment analysis of content on Twitter, possibly due to the data sets that were used.

Handledare: Stamatios Nikolis
Ämnesgranskare: David Sumpter
Examinator: Tomas Nyberg
ISSN: 1401-5757, UPTEC F 16012

Populärvetenskaplig sammanfattning

Inom maskininlärning används ofta så kallad “sentiment analysis”, sentimentanalys, för att utvärdera de åsikter som uttrycks i olika texter. Texterna placeras i en av tre olika klasser för att beskriva åsikterna. Klasserna som används är oftast “positiv”, “negativ” och “objektiv”. Men den informationen ger endast mycket begränsad information om texten, så det här arbetet undersöker om det går att använda samma metoder för att också utvärdera vilken typ av ämne som texten handlar om.

Texterna som klassificeras samlas in från en specifik grupp användare på det sociala nätverket Twitter. Eftersom alla texter på Twitter är mycket korta bör det gälla att varje text endast diskuterar ett ämne, vilket förenklar klassificeringsprocessen. Den specifika gruppen användare består till största delen av vetenskapsmän och konton som till stor del skickar meddelanden om olika vetenskapsrelaterade ämnen. Detta betyder att de texter som samlas in handlar om ett fåtal ämnen. De klasser som används är “vetenskapliga nyheter”, “vetenskaplig åsikt”, “populärvetenskap”, “politik” och “annat”.

Informationen om texternas ämne och sentiment används sedan för att interagera med användarna på Twitter genom att använda olika metoder inom språkteknologi och artificiell intelligens.

Texterna klassificeras genom att träna naiva Bayesianska klassificerare, en enkel men ofta effektiv metod som baseras på Bayes sats. Varje text klassificeras baserat på vilka ord och par av ord som texten innehåller. Resultaten från klassificeringen visar att klassificerarna inte lyckas klasificera texterna bättre än slumpen, men vi visar också att metoden som använts klarar av att klassificera sentiment för ett annat dataset av texter från Twitter. Detta kan tolkas som att problemet ligger i att de texter som smalats in till detta arbete inte är speciellt tillgängliga för klassificering av sentiment. För klassificering av texternas ämne finns inget passande dataset att jämföra med.

Interaktionen med användarna på Twitter beror i så pass stor del på att klassificeringen av texterna fungerar korrekt att det år svårt att utvärdera hur bra interaktionen fungerar.

Evenetuellt framtida arbete skulle kunna fokusera på att utvärdera metoderna med ett dataset som är större och med tydligare kategorier, eller på att använda andra metoder för att klassificera texterna.

Contents

1 Introduction

With the growth of interactive websites, such as social media, forums, and crowd-sourced product review, it is now possible for people to spread their opinion on different subjects to strangers and many people choose to do so [1]. The creation of such content opens up many possibilities to apply sentiment analysis and natural language processing on the data that is made available. Sentiment analysis can be very effective at identifying different sentiments on web pages [2], most commonly determining if a text, or part of text expresses a positive or negative sentiment. Typically, sentiment analysis does not consider the context of the sentiment, but some work exists where the causes of the sentiment are determined as well [3]. This paper considers the context of sentiments by determining the topic of texts as well as the sentiment.

The topic and sentiment analysis is performed on messages posted to the social media network Twitter. One of the defining characteristics of Twitter is the limitation that all messages must be short, no more than 140 characters long. Any such short message is called a tweet. The number of different topics that are analysed is by necessity limited, which is done by using only tweets from a certain group of users, with the common theme of discussing mainly scientific news and other issues relating to scientific research. The sentiment analysis as well as the determination of the topic of tweets is performed using multinomial naïve Bayes classifiers, since that method offers a combination of accuracy and simplicity [4].

The results of the sentiment and topic analysis are used as the basis to interact with human users on Twitter, using natural language processing with an approach similar to the ELIZA program [5] which has been extended to work with parts-of-speech as well as with specific keywords.

2 Tools

The program was written in the programming language Python [6], due to the availability of relevant libraries. The most used library in the program is the Natural Language Toolkit (NLTK) [7], which is written in Python and is therefore easy to use within Python. Another library used is the Python Twitter Tools [8], which offers a simple way to interact with the Twitter API from Python and was chosen over other options for its ease of use and compatibility with version 3 of Python.

2.1 Twitter and the Twitter API

Twitter is used as the source of data and also as an outlet for the content that is created by the program. All communication with Twitter is made possible by using the Twitter API (Application programming interface). The most important concept used is that of a message posted to Twitter, a so called tweet. Each tweet must be posted by some user on Twitter and will then be shown to other users if they are "following" the posting user on Twitter. After a tweet has been posted, any user can perform some actions in relation to the tweet, such replying to the tweet, which will notify the user who posted the tweet of the reply. Another action is to "retweet" a tweet, which posts the tweet as if the retweeting user had posted a new tweet (i.e. shows it to users who follow the retweeting user), while still retaining the information of which user originally posted the tweet.

The Twitter API is offered as a way to programmatically access the data available on Twitter. In terms of accessing the tweets that users post, the API offers two different approaches, both with different advantages and limitations. The two options are the REST (Representational state transfer) API and the streaming API. The main limitations of each option is that the REST API only allows a certain number of requests in a given time period, while the streaming API only allows access to new data.

Accessing the Twitter API is done by making use of the Python Twitter Tools [8], which allows to use functions calls in Python rather direct access to websites. The Python Twitter Tools also transform the replies from the Twitter API into data structures native to Python, e.g. lists or dictionaries. The library also offers an automated process for authenticating with Twitter, which allows the program written in this project to be more easily set up for different users.

2.1.1 The REST API

The Twitter representational state transfer (REST) API [9] allows access to read and write Twitter data in batches. It offers many different options for accessing the different parts of the data, with methods to access, for example, the newest tweets sent by a certain user, any specific tweet, and if you have access to the user's authentication details, the newest tweets that would be shown to the user. The REST API also offers the only way to post new tweets and other updates to the Twitter service.

The main limitation of the REST API is that it is rate limited, i.e. there is a limit to how many requests a program is allowed to send in a given time period. These limits can be problematic when trying to collect larger amounts of data or when repeatedly fetching small amounts of data.

The REST API offers the possibility of posting content to Twitter, which is not possible through the streaming API.

2.1.2 Streaming API

The streaming API [10] offers fewer, more specialized actions than the REST API, but can be used to collect larger amounts of data. The streaming API continuously delivers new data from some subset of the data posted to Twitter. It can be used to fetch all new tweets posted for a certain user to see, or a randomly selected subset of all tweets posted to Twitter. It is also possible to to receive tweets that match some filter, e.g. containing a certain word or having been posted by a certain group of users.

2.2 The Natural Language Toolkit

The Natural Language Toolkit (NLTK) is a library for natural language processing written in Python [7]. NLTK contains functions to perform a wide variety of tasks in the filed of natural language processing, we discuss the functions used in this project. One function often used as the first natural language processing stage is tokenization, available in NLTK using the function `word_tokenize`, which takes as input a string of text and returns a list of strings, where each element of the list is a word or a piece of punctuation.

A tokenized text, i.e. a list of strings, can be passed to NLTK for part-of-speech tagging, either using a specialized tagger or the default offering. The default tagger offered by NLTK can be called using `pos_tag`, which uses the Penn Treebank corpus [11]. The tagger will

assign one of 36 part-of-speech tags (or one of 12 punctuation tags) to each element of the input list. The function `pos_tag` will return a list of tuples where each tuple has two elements, the first being the corresponding string from the input and the second being the tag assigned to the string. The full list of possible tags can be found in Table 2 in [11].

NLTK contains a module, `classify`, which contains a framework for interacting with a classifiers in NLTK, as well as some built-in classification algorithms. The module also contains an interface that allows for seamless integration of classifiers offered by the Python library Scikit-learn [12], a larger selection than what is available in NLTK itself. The integration of Scikit-learn classifiers into NLTK is done using `classify.scikitlearn.SklearnClassifier`, which takes as input a Scikit-learn pipeline. For example, the set up to use a multinomial naïve Bayes classifier from Scikit-learn using the NLTK interface is shown in Algorithm 1.

```
import nltk
import sklearn
pipeline = sklearn.pipeline.Pipeline([
('nb', MultinomialNB()),
])
classifier = nltk.classifier.scikitlearn.SklearnClassifier(pipeline)
```

Algorithm 1: How to set up a multinomial naïve Bayes classifier using the Scikit-learn pipeline that can be accessed using the standard NLTK interface for classifiers.

2.3 Datasets

When running, the program automatically downloads and processes new tweets from a limited group of Twitter users, a group containing in large part scientists and some science-related news sources. Gathering this data relies on the Python Twitter Tools to fetch data from Twitter. Some tweets that are not relevant are removed from the dataset, using metadata attributes supplied by Twitter to determine the relevance of each tweet. The removed tweets have not spread much through the network of Twitter users, or are tagged as not being written in English. The spread of each tweet is measured by considering how many times the tweet has been marked as a favorite or has been retweeted by other Twitter users in relation to the age of the tweet and the number of followers that the posting user has. The language tag is supplied by Twitter as a piece of metadata, so the method used to determine the language is not known, may change at any point and should not be relied on to be perfectly accurate.

While running, the program makes use of pre-trained naïve Bayes classifiers, which have been trained using a dataset of tweets. The dataset used for training the naïve Bayesian classifiers was collected from the same group of Twitter users, with a science-related focus. The dataset was collected during sessions downloading all new tweets posted by a user in the group while the collection program was running. The collection program was executed several times, taking breaks between executions, from February to April of 2015, resulting in a dataset containing all tweets from some time periods but wholly missing others. The dataset contains 7597 tweets.

The dataset has not been made available in full as per the Twitter terms of service agreement, which does not allow distributing a copy of tweets (including for example usernames

and the full text of the tweets). One reason for this is the intention that the user who originally posted the tweet should be able to erase the tweet, which can't be done if it is allowable to make copies of tweets.

Another smaller dataset has been collected, using the same users on Twitter as the sources. Each tweet in the smaller dataset has been manually tagged with a tag describing the sentiment of the tweet and a tag describing the content of the tweet. The sentiment of a tweet is described with one of the tags "positive", "negative", or "objective". The content of a tweet is described with one of the tags "scientific news", "scientific opinion", "popular science", "political", or "other". The tagged dataset contains 1308 tweets.

A dataset of sentiment tagged tweets regarding four different topics was used to compare classification accuracy with the manually tagged dataset collected for this project. The external dataset is available at [13]. This dataset is available as a list of identification numbers of tweets, meaning that the full tweets (including the text of the tweets) must be downloaded from Twitter, which may not work if the tweet has been removed or is no longer publicly available. This limitation meant that only 4617 tweets from the dataset could be downloaded, rather than the 5513 tweets originally available in the dataset. The dataset tags the sentiment of each tweets as one of the following options: "positive", "negative", "neutral", and "irrelevant". This is similar to how the dataset collected for this project tags the sentiment of tweet, but has the addition of "irrelevant" tags. The dataset also contains tags of the subject discussed in the tweet but these tags do not consider the same idea of topic as considered in this project, as the topic of tweets is only based on direct mention of a Twitter user.

3 Classification

The goal of the classification stage is to assign two classes to each tweet, one describing the sentiment of the tweet and one describing the subject matter discussed in the tweet. The assigned classes are later used as the basis of the response to the tweet that is generated by the program. The sentiment of a tweet can be classified as either positive, negative, or objective. While the classes for the sentiment classification are rather general, the classes for the classification of subject matter must be chosen with consideration of the data sources, since the number of classes is otherwise almost unlimited. With this in mind, the subject matter of a tweet can be classified as scientific news, scientific opinion, popular science, political, or other.

Both classifications are performed using the same process, which begins by dividing each tweet into words using the tokenization method `word_tokenize` in NLTK, with added support for some patterns that are common on Twitter. The adjustments made are:

- Any word beginning with '@' until the next space or punctuation mark is considered a single word, which NLTK would otherwise not do. This is because such a word will refer to a user on Twitter, most likely a person.
- Any word beginning with '#' until the next space or punctuation mark is considered a single word. This is a common pattern on Twitter called a hashtag, usually used to denote that a tweet is relevant to a certain topic.
- Some common emoticons are are considered a single word. Any pairing of ':', or ';' followed by '(', ')', 'D', 'P', or 'p'; such as ':)' or ';D', is considered an emoticon.

The tokenization is performed on any tweet that is processed in the classification stage of the program.

Taking the tokenized text of each tweet as input data, two multinomial naïve Bayes classifiers are constructed to perform classification of the data. The multinomial classifiers are used since they have shown better performance than other naïve Bayes classifiers [14] and naïve Bayes classifiers are a good fit in general as they are relatively simple and well tested implementations are already available. The constructed classifiers consider all unigrams and bigrams, i.e. all words and word pairs, in the input text as features of the input text, to create a better understanding of context when compared with only using unigrams. A very basic example showing the need for bigrams is negation, e.g. the bigram "not good" is likely indicative of a negative sentiment, while the unigrams "not" and "good" may be misconstrued as positive. The use of bigrams has been shown to give minor accuracy improvements compared to using uni- or trigrams when classifying the sentiment of tweets [15].

To construct and train the classifiers, the Python module Scikit-learn [12] is used, due to its support of multinomial naïve Bayes classifiers, whereas NLTK only has built-in support for naïve Bayes classifiers based on the Gaussian distribution. Introducing Scikit-learn to the program is simple since NLTK offers an API wrapper for Scikit-learn (in the module `classify.scikitlearn`) with the method `Skclassifier`, which allows access to a scikit-learn classifier with the same interface. This allows for using the scikit-learn classifiers as drop-in replacements for NLTK classifiers.

The classifiers are trained using a manually classified dataset of tweets from the same users that the program will be reading tweets from (see section 2.3). Any new tweet is then classified by the naïve Bayes classifiers to determine the subject matter in the tweet as well as the sentiment of the tweet and is passed to the response generation stage with these characteristics known.

3.1 Results

The classification accuracy of the naïve Bayes classifiers was tested using the manually tagged dataset of tweets (section 2.3). The dataset was randomly divided into two parts, with 70% of the dataset was used to train the classifiers and 30% used to evaluate the classifiers. This process was repeated 20 times and the accuracy was calculated both for classification of content and sentiment of each tweet. Using this method, the accuracy when classifying the content of a tweet was measured at 29.9% and the accuracy when classifying the sentiment of a tweet was measured at 33.5%. Considering that there are three possible tags for the sentiment of a tweet, 33.5% accuracy is no better than a random choice. While there are five possible tags for classification of content, and so, 29.9% accuracy is better than a random choice, one must consider that there is not an equal distribution of all tags in the dataset. The tag "other" is assigned the largest number of tweets, with 32.2% of the dataset being tagged as such, meaning that tagging each tweet with "other" would in most cases be more accurate than using a trained naïve Bayes classifier. This argument also holds true for the sentiment classification, where the tag "objective" is more common than the others.

A more detailed view of how the naïve Bayes classifiers are behaving can be shown using the confusion matrices for content and sentiment classification. The confusion matrix for content classification is shown in Table 1. From the table, it can be seen that every class has a high probability of being classified as scientific opinion by the classifier, meaning that a large percentage of incorrect classifications are from tweet being given the tag "scientific opinion". The confusion matrix for sentiment classification is shown in Table 2, which shows

Table 1: Confusion matrix for content classification of the manually tagged dataset using randomly chosen 70% of the dataset for training and the rest for evaluation, with the process repeated 20 times. Each tweet is added to the matrix by letting the manually set tag determine the row and letting the tag set by the classifier determine the column. Each element of the matrix shows how many tweets were classified with a certain pair of tags. The part of the name of the tag making it explicit that the tag refers to science related topics has been left out for the tags scientific news, scientific opinion and popular science.

		Class assigned by classifier				
		News	Opinion	Other	Political	Popular
Manually assigned class	News	585	913	88	308	376
	Opinion	54	493	17	102	123
	Other	179	1010	407	426	367
	Political	41	373	25	494	82
	Popular	156	598	58	197	368

Table 2: Confusion matrix for sentiment classification of the manually tagged dataset using randomly chosen 70% of the dataset for training and the rest for evaluation, with the process repeated 20 times. Each tweet is added to the matrix by letting the manually set tag determine the row and letting the tag set by the classifier determine the column. Each element of the matrix shows how many tweets were classified with a certain pair of tags.

		Class assigned by classifier		
		Negative	Objective	Positive
Manually assigned class	Negative	716	84	162
	Objective	2565	1038	1438
	Positive	734	136	876

that the classifier very often incorrectly classifies tweets that are given the "objective" tag and also shows that it is common for tweets with the tag "positive" to be classified as "negative".

The classification accuracy can also be evaluated by studying if the accuracy changes with the size of the training set. In Figure 1, this is shown for the manually tagged dataset and the external dataset. The accuracy for the manually tagged dataset does not change much, while the external dataset has a much classifier even when using the maximum training size for the manually tagged dataset, when compared to the minimum training set size.

4 Response generation

The response generation stage takes as input a tokenized and classified tweet, tries to identify special properties of the tweet (in terms of keywords or language properties) and generates a response based on the properties of the tweet. This uses natural language processing, using both NLTK tools and specialized methods. Several different approaches are used, with the goal to should create a larger variety in the responses.

The program implements an approach for response generation that is similar to the ELIZA program [5]. Such approaches can use keywords in the tweet to base the response around or other more general methods to identify parts of the original tweets that can be used as the basis of a response. One of those approaches is to identify sequences of words from different parts of speech, i.e. verbs or nouns. With this approach, the sequences

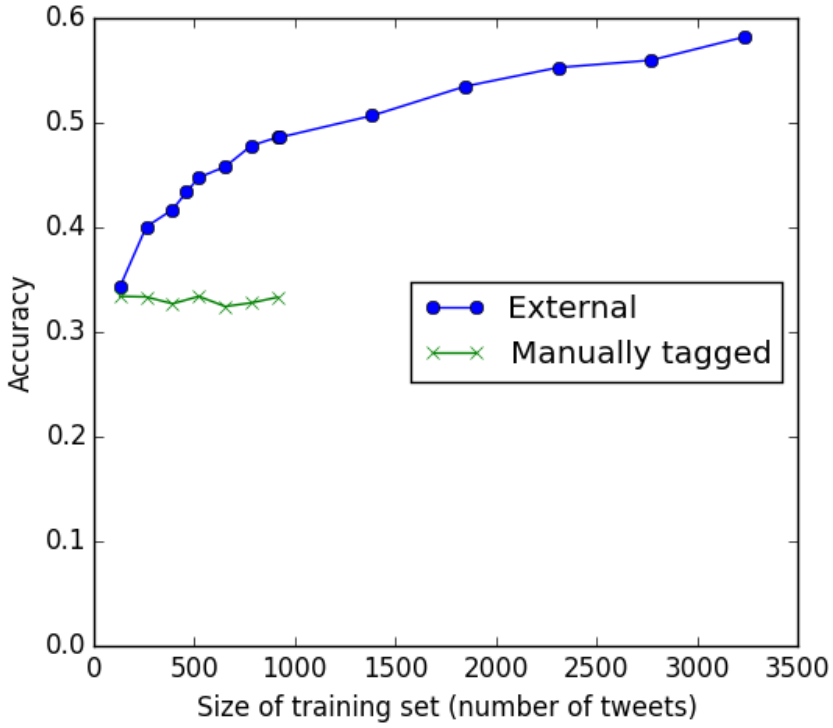

Figure 1: The classification accuracy as a function of the size of the training set using the same naïve Bayes approach for training the classifiers. The accuracy of the manually tagged dataset is the accuracy of topic classification, while the accuracy shown for the external dataset is the accuracy when classifying the sentiment.

describe general patterns but can still be used to generate grammatically correct responses. To do this, the part-of-speech tagger functionalities of NLTK are used to assign each word in each tweet a tag that identifies the part-of-speech that the word belongs to. The program then searches for pre-programmed patterns of tags, i.e. different sequences of words with certain part-of-speech tags, and use the occurrence of such patterns to generate a response.

This approach is a generalization the approach used by the ELIZA program, which uses specific words instead of parts-of-speech. This approach can be mixed with searching for certain words. An example pattern that uses both parts-of-speech and words to search for matching tweets is "**noun** is **adjective**", which will match and tweet that contains a sequence of words where the first word is a noun, the second word is "is" and the third word is an adjective. To specify such a pattern in the program, one can construct a tuple in python, `((None, noun), ('is',None), (None, adjective))`. This tuple can then be searched for in every tweet processed by the response generation stage of the program, which also requires specifying a function that is to be called to generate a response, if the pattern is found. The function will be passed the tweet as an input variable and can be used to perform a follow up action, such a replying to the original tweet or retweeting with an added comment. The program can choose to send a retweet of a tweet, in particular if no pattern can be identified in the tweet.

The program keeps track of a set of the most common nouns and adjectives that appear in tweets in the dataset and any correlation between occurrences of any such words. For example, if a tweet contains a common noun, **n**, and a common adjective, **a**, the program might generate the response "Isn't **n** quite **a**?". The program can also generate new tweets, i.e. not responses, that find correlated words and make statements such as "What does **a** really have to do with **b**?", where **a** and **b** refer to nouns that commonly used together in tweets in the data set.

The program does not run response generation for each tweet, but rather uses an estimation of if other users have already found the tweet interesting by taking into account the number of retweets and favorites of a tweet in relation to how many followers the poster of the tweet has. The number of responses that are generated is also limited to 10 per day, and the program will only actively post tweets during the afternoon. These limitations are put in place to mimic the behaviour of human users. A further limitation is that program will not directly respond to users who do not already follow the Twitter account of the program.

4.1 Results

The program has been allowed to run, i.e. manually download and process all new tweets from the group of users that it aims to interact with for a period of a few weeks, being limited to generating 10 responses per day. The program has been posting messages to the Twitter account @SubjectBot[1]. The program has been inactive on some of the days due to bugfixes and other minor changes having been performed.

An example of the kind of responses generated by the program is shown in Figure 2, which contains two tweets where the program has retweeted content from other users and one tweet that is a question regarding some message that another user posted. The question asked has not made use of any particular occurrence of a keyword or similar in the original tweet, it is a generic response based on the topic classification of the tweet, e.g. "scientific opinion".

[1] `www.twitter.com/SubjectBot`

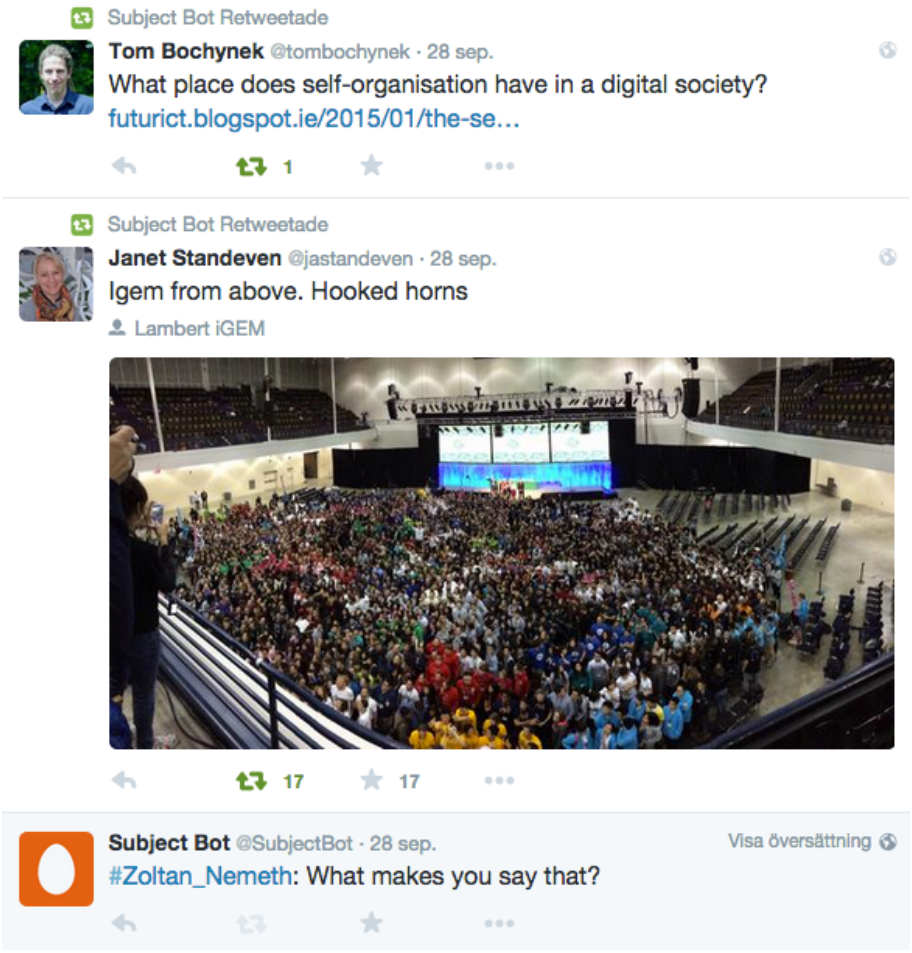

Figure 2: An excerpt from the stream of tweets posted by the program as shown when visiting the program's account on Twitter. This shows an example of the type responses generated by the program.

5 Discussion

The poor classification accuracy of the trained naïve multinomial Bayes classifiers is a limiting factor for the performance of the program, as the response generation heavily depends on the classifications of tweets. The simplest approach to improving the accuracy would be to increase the size of the training dataset, but there is also no indication that using a larger training set would improve the classification accuracy, as can be seen in Figure 1. It may still be possible to achieve higher accuracy by adding training data, in the case that the used dataset is too small to allow the classifiers to learn any significant pattern. On the other hand, the external dataset shows improvements in classification accuracy for training sizes that are available for the manually tagged dataset. This could indicate that in some sense the external dataset is easier to classify. A possible reason might be that the manually tagged dataset contains tweets mostly from scientists discussing science-related subjects. This may mean that tweets in the manually tagged dataset use a larger, more specialized vocabulary, making any patterns in the dataset less pronounced. Other sentiment classifications of Twitter data [15–17] also show higher accuracies using multinomial naïve Bayes classifiers with similar feature extraction, further indicating that the manually tagged dataset is not classified as well by naïve Bayes classifiers as other datasets using data from Twitter.

Another possibility is that the topic tags are assigned, by humans, in a manner that uses properties not described by the extracted features, i.e. other properties than which unigrams and bigrams occur in the tweet determine the topic of the tweet. This is unlikely to be the only reason for the poor classification accuracy, since if it were, the classifiers trained for evaluating sentiment of the manually tagged dataset should show better results, but they also have poor accuracy. Still, it might be possible to find features that would improve accuracy, but the necessary features may also be dependent on the dataset, e.g. using part-of-speech tags as features have been found to be both helpful [18] and not [17] for classifying the sentiment of tweets.

The confusion matrix for the classification of sentiment for the manually tagged dataset, Table 2, shows that the trained classifier performs well at classifying negative tweets correctly, being almost 75% accurate. The accuracy is also reasonable for positive tweets at more than 50%, while the classifier very often assigns an incorrect tag to the objective tweets. A reason for this may be that the tweets given the objective tag are more varied, while there may be clearer connections between tweets that are positive or negative.

A similar issue can be seen in Table 1, where tweets that have manually been assigned the "other" tag are often misclassified by the classifier, a tag which is intended to contain a much larger variety of tweets since the purpose of the tag is to cover everything not included in any other tag. So it may be possible to improve the classification accuracy by introducing explicit classes for other types of content in the dataset. But this also introduces other problems, such as simply having more options leads to there existing more incorrect choices. It may also introduce unbalanced classes, i.e. classes that contain many fewer examples in the training set, which can be problematic for multinomial naïve Bayes classifiers [19]. On the other hand, when the content classifier does classify something with the other tag, it has quite a high accuracy. The classifier also has relatively few false positives for tweets it has classified as scientific news. This property could possibly be exploited for response generation; any tweet the has been classified as news is quite likely to actually be news. The only other class with few false positives is "other", which is much harder to make use of for

response generation, as a tweet can still be about many different topics, even if it has been correctly classified as “other”.

In Table 1 it can be seen that the classifiers will assign one of the tags scientific opinion, political, or popular science to tweets without much correlation with the manually assigned tag. This would indicate that the classifier has a poor understanding of what characteristics define a tweet in these classes.

In response generation, the program can make use of part-of-speech tags to identify parts of sentences that it can generate a response to. While this approach is more generic and allows for writing more generic rules, that also introduces other issues that are not fully handled in this implementation. This first issue is requiring correct part-of-speech tags, but as shown in [20], it is possible to achieve high accuracy, also showing the possibility of improvement using more specialized tagging over using the Penn Treebank corpus.

A second issue is that of dealing with different forms of words, as created by different inflections. The implementation makes no such considerations and can therefore create sentences containing grammar errors that humans would not commit. A possible approach to deal with this is to stem the word, reducing the word to its base form, and then inflecting the word correctly using the base word. Several algorithms used for stemming are included in NLTK, while tools to inflect a base word are not included and there are no well-known software tools that can generally perform this task.

While the program is fully working as is, it does not produce results that are particularly convincing or engaging. The main limitation of this is likely the poor classification accuracy of the trained classifiers, which also makes it harder to evaluate the quality of the responses generated by the program, but to significantly improve the function of the program, there will likely need to be improvements in both areas. It should be possible to improve the classification accuracy, as there is other work [15–18] showing much better results, but it could be that this may only be possible on by applying the program to other sources of data. As for response generation, the main options for improving the performance would be to create a larger variety of responses that can be created using the current methods or using other natural language processing methods to create responses.

References

[1] B. Pang and L. Lee, "Opinion mining and sentiment analysis," *Found. Trends Inf. Retr.*, vol. 2, pp. 1–135, Jan. 2008.

[2] T. Nasukawa and J. Yi, "Sentiment analysis: Capturing favorability using natural language processing," in *Proceedings of the 2nd International Conference on Knowledge Capture*, K-CAP '03, (New York, NY, USA), pp. 70–77, ACM, 2003.

[3] K. Cai, S. Spangler, Y. Chen, and L. Zhang, "Leveraging sentiment analysis for topic detection," *Web Intelli. and Agent Sys.*, vol. 8, pp. 291–302, Aug. 2010.

[4] T. Chalothorn and J. Ellman, "Tjp: Using twitter to analyze the polarity of contexts," *Atlanta, Georgia, USA*, p. 375, 2013.

[5] J. Weizenbaum, "Eliza - a computer program for the study of natural language communication between man and machine," *Communications of the ACM*, vol. 9, no. 1, pp. 36–45, 1966.

[6] G. Van Rossum *et al.*, "Python programming language.," in *USENIX Annual Technical Conference*, vol. 41, 2007.

[7] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python*. O'Reilly Media, Inc., 2009.

[8] "Python twitter tools." `http://mike.verdone.ca/twitter/`. Accessed: 2015-06-02.

[9] "Rest apis." `https://dev.twitter.com/rest/public`. Accessed: 2015-09-24.

[10] "The streaming apis." `https://dev.twitter.com/streaming/overview`. Accessed: 2015-09-24.

[11] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: The penn treebank," *Computational linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[12] 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," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[13] N. Sanders, "Twitter sentiment corpus." `http://sananalytics.com/lab/twitter-sentiment/`. Accessed: 2015-09-03.

[14] S. Eyheramendy, D. D. Lewis, and D. Madigan, "On the naive bayes model for text categorization," 2003.

[15] A. Pak and P. Paroubek, "Twitter as a corpus for sentiment analysis and opinion mining.," in *LREC*, vol. 10, pp. 1320–1326, 2010.

[16] A. Go, R. Bhayani, and L. Huang, "Twitter sentiment classification using distant supervision," *CS224N Project Report, Stanford*, vol. 1, p. 12, 2009.

[17] A. Agarwal, B. Xie, I. Vovsha, O. Rambow, and R. Passonneau, "Sentiment analysis of twitter data," in *Proceedings of the Workshop on Languages in Social Media*, pp. 30–38, Association for Computational Linguistics, 2011.

[18] L. Barbosa and J. Feng, "Robust sentiment detection on twitter from biased and noisy data," in *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pp. 36–44, Association for Computational Linguistics, 2010.

[19] E. Frank and R. R. Bouckaert, "Naive bayes for text classification with unbalanced classes," in *Knowledge Discovery in Databases: PKDD 2006*, pp. 503–510, Springer, 2006.

[20] K. Gimpel, N. Schneider, B. O'Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yogatama, J. Flanigan, and N. 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*, pp. 42–47, Association for Computational Linguistics, 2011.

A Program structure

The program is written in python 3. The program is constructed as a python package using setuptools, the package is named `subjectBot` and defines the dependencies `nltk`, `twitter`, `numpy`, `scikit-learn`, `tabulate` and `matplotlib`. The `tabulate` and `matplotlib` packages are non-essential, used only for visualizations of data.

The `subjectBot` package itself is divided into 9 different modules, covering the different tasks that the program performs. The `main` module defines mostly higher-level functions, including the main function itself, which downloads, processes and generates new responses to tweets. It also defines some functions to help with these tasks, such as `get_newest_tweets` which fetches the most recent tweets posted by the group of users that the bot follows.

It also defines functions for other general tasks that the program can perform, such as `train`, `test` and `full_test`, which cover the training and testing of classifiers. To train a classifier, `train()` is called with the first argument as the name of a file that is a correctly formatted, i.e. containing "id", "text", "sentiment", and "content" columns. The classifiers are stored in a binary format in the file whose name is given as the second argument. The classifiers can then be tested using `test` or be used during the normal operation of the program by setting the correct filename as the default in the `action.init` function. The `full_test` function incorporates both training and testing into one function.

The main module also defines the functions `common_words` and `co_occurence` which respectively find the most common words in a corpus and finds and correlations between such words in some (usually the same) corpus. These functions are included since the most common words and their correlations can be used by the program during response generation. Both functions are used by `common` which is the canonical way to generate sets of common words that can be used by the program and should be used to generate new lists of common words every once in a while (when new tweets have been collected).

Several other functions are available in the `main` module, which perform more convenience-oriented functions, e.g. pretty-printing confusion matrices of common words in the dataset with `confusion_matrix`.

A.1 Setting up the program

Before it is possible to successfully run the program, it is necessary to perform some steps to set up the program. The first step is to install python 3 and afterwards the program can be installed. The program can be installed, for example, using the command `pip -e ~/SubjectBot/` where `~/SubjectBot` is the location of the source code. Since the program is written in python 3, it should be installed using tools for python 3, i.e. the correct version of `pip` may be called `pip3` on some systems. Executing this command successfully will install all the necessary dependencies making it possible to perform the following steps of the setup.

When all required dependencies are installed, it is necessary to provide some data to set up the program. To create the needed data files, first load the `main` module, e.g. using `python -i subjectBot/main.py` and then call the function `train()` where the input file should be a .csv-file with the first line `"id","text","sentiment","content"`. Each following line should contain, separated by a single comma, first an integer followed by three strings. Each string should be surrounded by `"`, with any occurrences of the character in the string itself being replaced by `""`. The integer should not be surrounded by anything. This will train the necessary classifiers and store them in a binary format in the file `data/classifiers` (relative path). It is also possible to construct a database of non-classified tweets, but this will also be automatically created during normal operation of the program, all tweets that are processed are stored. But to do this manually, one should store each tweet using the `shelve` module in python, e.g. with `shelf[tweet["id_str"]] = bot.dehydrate(tweet)`, if the file "user.db" has been opened to the variable `shelf`. The function `dehydrate` removes unneeded attributes of the tweets and is found in the `bot` module.

With the dependencies installed and the data files in place, it is possible to run the program, by loading the main module into an interactive python prompt and calling the function `main`. The first time this is done, the program will ask the user to authenticate the program use to a Twitter account. The configuration will then be stored in the file `config.txt`. The program will then run until the process is interrupted.