Tell me a good joke!

Evelina Dumitrescu, master SCPD

April 24, 2015

1 Introduction

In this project, we want to determine what makes a good joke and as relevant data for telling the quality of the joke we considered the users' comments. This turns out to be a problem of sentiment analysis and classification. Based on the emotions extracted from the comments, we can classify them as positive, negative or neutral and tell how funny a joke is. For the moment, we chose to limit to the comments from the joke pages available on Facebook (ie: 9GAG, Joke of the Day). In the second section we present current techniques used for sentiment analysis and classification and in the third section we present our approach.

2 Related work

In the last decade, the subject of sentiment analysis and classification for social media networks has received increased attention. We have identified three major techniques relevant for our project: lexicon based, noisy labels and machine learning methods.

The first approach consists in using a dictionary of words or concepts annotated with their semantic orientation or polarity. In order to achieve a better precision and performance, domain-specific lexicons should be used, providing better word polarities for a given domain. However, building such lexicons is extremely expensive, so many systems use a general purpose lexicon. It is common to extract Part of Speech (POS) information for overcoming word-sense ambiguity and to consider some parts to be more relevant than others (ie: adjective and adverbs can provide more intense emotions than verbs or nouns). Examples of domain-independent dictionaries are SenticNet [1] and SentiWordNet [2]. SenticNet uses the bag of concepts approach, inferring the polarity of common sense concepts from natural language text at a semantic level, rather than at the syntactic level. It integrates approximative 15000 entries. On the other side, the SentiWordNet lexicon uses the "bag of words" technique and is based on an English dictionary called WordNet. It groups grammatical categories into synonym sets called synsets. SentiWordNet associates three scores for a synset to indicate the sentiment from the text: positive, negative and neutral. It has over 100000 entries. Lexicon based techniques are proposed in [1], [2], [3].

[1] http://sentic.net/

[2] http://sentiwordnet.isti.cnr.it/

Due to the fact that in social media platforms the majority of messages are short sized (ie: Facebook status messages, Twitter reviews, some comments), the noisy label technique can lead also to good results. [4] and [5] use emoticons as noisy labels to train classifiers on a dataset collected from the Usenet newsgroup or from Twitter reviews. One can assume that an emoticon represents an emotion and all the words from the message are related to this emotion.

Alternatively, supervised machine learning techniques are used to build models or discriminators for different classes. For instance, [6] uses the Naive Bayes algorithm to separate positive reviews from the negative ones. Other research articles [7] realise a comparative approach between several machine learning algorithms, such as Naive Bayes, Support Vector Machines and Maximum Entropy.

3 Proposed solution

In this section we describe our approach for classifying jokes and for evaluation their perceived level of amusement. Our approach can be structured in five stages: corpus creation, composing the training data set, building the classfier, evaluate new jokes and determine the accuracy of our system. For creating the corpus, we extract comments from the joke pages available on Facbook using the Facebook Graphs API [3]. Using the Natural Language Toolkit(NLTK) [4], we tokenize the sentences and apply stemmatization algorithms on the resulting words. For the begining, we will work with "bag of words". We consider emoticons as standalone words. Increased attention should be given for repteated letters (ie: "happy"), negations, contractions and stopwords elimination.

In order to form the training data set, we split the processed comments into positive and negative/neutral categories. For every word from a comment, compute the relevance for the category that the specific comment belongs. In order to do this, we use the term frequency-inverse document frequency(TF-IDF) score. This is a numerical statistic metric that increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus; this helps to adjust to the fact that some words appear more frequent in general. For instance, "laugh", "funny", ":D" are relevant for a positive comment and "cry", "horrible", "disgusting" and ";(" are relevant for a negative comment.

For classifying new comments, we intend to use the Transformed Weight-normalized Complement Naive Bayes [8]. We need to take into account also the TF-IDF score, not just the probabilities, because it is important to know its relevance, besides how frequent the term occurs in a specific class. The Scikit-learn [5] framework already has this implemented.

To evaluate the quality of the joke, we need to count the number of positive/negative comments. In order to tell how funny is the joke considered by a specific user, we can use a score weighted by the positive/negative polarity, but also by the relevance for the positive category $\sum_{word \in Words} Polarity_{word} * TFIDF_{word}$. For the polarity of words, we can use the SenticNet database [6] and include additional polarities for emoticons. For

[3]https://developers.facebook.com/docs/graph-api
[4]http://www.nltk.org/
[5]scikit-learn.org/
[6]view-source:http://sentic.net/api/en/concept/

instance, “think” and “funny” have positive polarities of 0.061 and of 0.619; however, “think” won’t be as relevant as “funny”. On the other hand, “ugly” has a negative polarity of -0.581.

It is essential to know how accurate our classifier is. To achieve this, we should use testing samples for comments and see if classifier’s prediction corresponds to our judgement. We need to identify false positives (FP), false negatives (FN), true positives (TP) and true negative (TN) cases. Furthermore, the F1, precision (positive predictive value), recall (sensitivity, true positive rate) scores have to be computed for measuring the efficiency.

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

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

$F1 = 2 * \frac{Precision*Recall}{Precision+Recall}$

Crossvalidation aproaches might help improve the evaluation.

4 Current state of the project

In this section we are going to describe the steps that we followed for developing this stage of the project: building the corpus, preprocessing the text and selecting the most important features that could reflect whether a user perceives a joke as amusing or not.

4.1 Building the corpus

For building the corpus, we created a custom crawler to extract posts and comments from Facebook public pages. We choose pages that could give us as much relevant information for the user’s opinion as possible (e.g: we tried to avoid pages with a lot of spam posts of not enough comments). We selected the most recent 200 posts and at most 200 comments for each post. Further, we classified the comments into positive, negative and spam ones.

For interacting with Facebook data, we used the Graph API[9]. Facebook Graph API presents an interface of the Facebook social graph, uniformly representing objects in the graph(e.g.: people, photos, pages, events, comments) and the connection between them (e.g. relationships, content sharing, tagging). We extract the data in JSON format.

For each post, we store the post id, the created time, number of likes and shares, comments and information that if relevant for the content of the post, such as the description, message or picture. For each comment we store the identity of the user that posted the comment, the message, message tags, the number of likes and the time of creation.

Facebook Graph API uses cursor pagination and at most 25 elements can be extracted at a time, for accessing more content the next field from the cursor should be used.

4.2 Preprocessing the text

Social media text processing can raise a lot of problems, due to the content that uses special characters(e.g: unicode characters for Arabic and Chinese language or emojis), the presence of emoticons, pictures, message tags and URL links. Therefore, in order to manage the extraction of features from a text, all this elements should be removed or substituted.

Firstly, we removed all unicode characters and message tags(person names from comments). Secondly, we classified emoticons commonly used on Facebook in more than 40 catagories an subsituted each emoticon with this label: happy face(:-) :) :D :o) :] :3 :c) =] 8) =)), crying(:'-(:'(), angel(O:-) 0:-3 0:3 0:-) 0:) 0;^), devilish(}:-) }:) 3:-) 3:)). Further, we expanded contractions with their expanded forms, such as replacing "can't" with "cannot", or "would've" with "would have". This approach can be applied for emojis also, but we will be consider it for the next milestone.

We used NLTK (Natural Language Toolkit) for splitting each comment into a list of sentences and each sentence into a list of words. In addition, we removed all punctuation signs and stopwords(e.g. a, about, above, across, after, afterwards, again, against). We used the list of stopwords included in the NLTK toolkit and Scikit-learn, but without words that express negation like not, nor, neither, nothing, nobody). Words that express negations are extremely important because they reverse the semantic and polarity of the words.

The next step of text preprocessing is spell checking. On social media, people are often not strictly grammatical. They will write things like "I looooooove it!" to emphasize their feelings. Unfortunately, computers don't know that "looooooove" is a variation of "love", unless they are told so. We had to remove all these repeating characters in order to end up with a gramatically correct English word. As an enhancement, we recursively remove the characters until the word is found in the WordNet[7] dictionary. We also try to remove repeating sequences of letters, like the ones that express happiness and laughing(e.g lololol becomes lol and hahaha becomes haha).

Each token generated by the NLTK tokenizer is verified with the PyEnchant[8] spell checker. PyEnchant is a spellchecking library that is based on the Enchant[9] library. The Enchant library provides spell checking (e.g whether a specific word appears in the specified dictionary) for various languages, and additionally can suggest a list of words in case the spell checker returns false. We first check wheather the given word is present in the dictionary or not. If it is, then no spelling correction is necessary and the word is returned. But if the word is not found, it looks up a list of suggestions and returns the first suggestion, as long as the edit distance(the number of character changes necessary to transform the given word into the suggested word) is less or equal to 2. This is to ensure that no unlikely replacement words are returned.

When communicating on social media, people use a lot of abbreviations and jargons. Therefore, we extracted acronyms and jargon terms from Netlingo [10]. Netlingo is an online dictionary with a comprehensive list of text message shorthand and Internet acronyms,

[7]https://wordnet.princeton.edu/
[8]http://pythonhosted.org/pyenchant/
[9]http://www.abisource.com/projects/enchant/
[10]http://www.netlingo.com

grouped in several categories: chat shorthand, online jargon, net technology, hardware, software, web sites etc. Enchant also supports custom word lists for spell checking. These can be combined with an existing dictionary, allowing you to augment the dictionary with your own words.

The final step of text preprocessing is word lemmatization using NLTK WordnetLemmatizer. Unlike stemmatization which leaves the root stem, lemmatization consists in finding the root word. The lemma (basic dictionary form) of a word is used as a feature instead of the actual word from the text. For example best becomes good, dogs becomes dog and so on. The reason behind this idea is to make the features more general for easier classification of the document.

As a optimization considered for the next milestone, we want to replace each expression consisting in a negation (e.g no, not, neither) and any other word with the antonym of the word. Let's say we have a sentence such as "I do not like the joke". With antonym replacement, one can replace "not like" with "dislike", resulting in the sentence "I do dislike the joke".

4.3 Selecting the most important features

A common mistake when choosing a good feature set is trying to provide too many features based on the training set, which makes the resulting classifier prone to the idiosyncrasies of the training set and therefore fails to generalise well to examples that were not encountered before (i.e new to the classifier). This is known as overfitting.

We used unigrams, bigrams and trigrams as features. Unigrams features are known as representing documents as a feature vector with the elements showing if a word appears in the document. Each word has a frequency of occurrences in the text. Ngrams are a sequence of nwords from a given sequence of text or speech. This allows the features to be a pair of words (bigrams), triples(trigrams) and so on. This allows capturing of more context.

We used as a feature weighting method TF-IDF(Term Frequency Inverse Document Frequency). Term Frequency(TF) reffers to the term frequency of each feature in the sentence it belongs to.For each term, the Inverse Document Frequency(IDF)is calculated by the logarithmic value of the number of documents in the corpus divided by the number of documents containing the specific term. The TF-IDF is computed as the TF multiplied by the IDF.

For avoiding problems when a specific word is frequently used in a specific document, the normalisation technique is used when dealing with TF-IDF. We used L2 normalization(Euclidean distance). Sometimes, it is better not to use all the features of the document but only those highly informative. If the features do not add any value or worsen the model, it is better not to include them. Chi square feature selection allows the selection of the K best features, the most informative ones.

Scikit-learn has all this techniques implemented in the TfidfVectorizer and SelectKBest classes. We applied them on the processed comments and extracted the most 100 relevant features, as shown in Most relevant 100 features selection compared to UClassify polarities. We verified our selected features for the negative and positive categories and

verified if their polarities match with the ones retrieved by Uclassify[11].

```
tfidf = TfidfVectorizer(
  ngram_range=(1,3),
  max_features=3000,
  norm='l2',
  sublinear_tf=True)
tfidf_feature = tfidf.fit_transform(corpus)
feature_names = tfidf.get_feature_names()
ch2 = SelectKBest(chi2, k=100)
chi2_feature = ch2.fit_transform(tfidf, [1,2])
feature_names = [feature_names[i] for i in ch2.get_support(indices=True)]
scores = np.array(chi2_feature.toarray()).T
```

4.4 Alternative tools

We consider that text processing can be done more easily using a scripting language such as Python. Moreover, there are already a lot of mature open source projects written in Python, that can help improving research for NLP, such as PyEnchant, Numpy, Scipy, NLTK or Scikit-learn. Alternatives for NLTK are OpenNLP[12] and Stanford NLP tools[13]. Besides PyEnchant, Whoosh[14] could have been an alternative. Other tools that do similar things related to out project are VaderSentiment[15] and TwitterNLP[16].

[11] http://uclassify.com
[12] https://opennlp.apache.org/
[13] http://nlp.stanford.edu/software/
[14] https://pypi.python.org/pypi/Whoosh/
[15] https://pypi.python.org/pypi/vaderSentiment
[16] http://www.ark.cs.cmu.edu/TweetNLP/

A Most relevant 100 features selection compared to UClassify polarities

Feature	KBest+	KBest-	Polarity+	Polarity-
aha	[0.103958]	[0.018791]	[0.5]	[0.5]
aha aha	[0.091618]	[0.]	[0.5]	[0.5]
aha lmao	[0.070359]	[0.]	[0.5]	[0.5]
aha lol	[0.086207]	[0.]	[0.99103]	[0.00897]
aha lol aha	[0.064342]	[0.]	[0.99103]	[0.00897]
aha nice	[0.064342]	[0.]	[0.998848]	[0.001152]
bastard	[0.]	[0.066917]	[0.]	[1]
bitch	[0.]	[0.052578]	[1.00000000e-06]	[0.999999]
break	[0.]	[0.050897]	[0.125791]	[0.874209]
break heart	[0.]	[0.049638]	[0.778643]	[0.221357]
brilliant	[0.056585]	[0.]	[0.98534]	[0.01466]
crazy love	[0.056585]	[0.]	[0.999005]	[0.000995]
cruel	[0.]	[0.060765]	[0.029257]	[0.970743]
cruelty	[0.]	[0.051483]	[0.000727]	[0.999273]
disgusting	[0.]	[0.064109]	[0.]	[1]
emoticonbrokenheart beautiful	[0.056585]	[0.]	[0.999865]	[0.000135]
emoticoncrying	[0.]	[0.069892]	[0.5]	[0.5]
emoticoncrying emoticoncrying	[0.]	[0.046663]	[0.5]	[0.5]
emoticonfrown	[0.]	[0.078746]	[0.5]	[0.5]
emoticonfrown emoticoncrying	[0.]	[0.045795]	[0.5]	[0.5]
emoticonfrown emoticonfrown	[0.]	[0.050897]	[0.5]	[0.5]
emoticonkiss emoticonkiss	[0.]	[0.054518]	[0.5]	[0.5]
emoticonkiss emoticonkiss emoticonkiss	[0.]	[0.046663]	[0.5]	[0.5]
emoticonlaughing	[0.079431]	[0.]	[0.5]	[0.5]
emoticonsmiley aha	[0.070359]	[0.]	[0.5]	[0.5]
emoticonsmiley lol	[0.070359]	[0.]	[0.99103]	[0.00897]
emoticonsmiley yes	[0.064342]	[0.]	[0.375703]	[0.624297]
evil	[0.]	[0.065868]	[0.294442]	[0.705558]
funny aha	[0.056585]	[0.]	[0.728604]	[0.271396]
funny lol	[0.064342]	[0.]	[0.962574]	[0.037426]
hahaha	[0.056585]	[0.]	[1]	[0.]
hate	[0.]	[0.048958]	[0.000268]	[0.999732]
heart	[0.]	[0.062308]	[0.985097]	[0.014903]
heartbreaking	[0.]	[0.064921]	[0.999997]	[3.00000000e-0
heartless	[0.]	[0.045795]	[0.039128]	[0.960872]
hell	[0.]	[0.053585]	[0.000139]	[0.999861]
hilarious	[0.056585]	[0.]	[0.997942]	[0.002058]
hope	[0.]	[0.061036]	[0.237972]	[0.762028]
horrible	[0.]	[0.06156]	[0.]	[1]
horrific	[0.]	[0.046663]	[0.023352]	[0.976648]
idiot	[0.]	[0.051483]	[0.000185]	[0.999815]
joke lol	[0.070359]	[0.]	[0.200838]	[0.799162]
karma	[0.]	[0.052042]	[0.002846]	[0.997154]

Feature	KBest+	KBest-	Polarity+	Polarity-
knock knock	[0.075275]	[0.]	[0.002424]	[0.997575]
knock knock knock	[0.056585]	[0.]	[0.00211]	[0.99789]
knock knock penny	[0.056585]	[0.]	[0.036057]	[0.963943]
knock penny	[0.056585]	[0.]	[0.139876]	[0.860124]
lmao	[0.113754]	[0.]	[0.5]	[0.5]
lmao aha	[0.070359]	[0.]	[0.5]	[0.5]
lmao lol	[0.070359]	[0.]	[0.99103]	[0.00897]
lmao lol lol	[0.056585]	[0.]	[0.993255]	[0.006745]
lmfao	[0.056585]	[0.]	[0.5]	[0.5]
lmfaohaha	[0.056585]	[0.]	[0.5]	[0.5]
lol	[0.109877]	[0.008954]	[0.99103]	[0.00897]
lol aha	[0.101721]	[0.]	[0.99103]	[0.00897]
lol aha lol	[0.056585]	[0.]	[0.993255]	[0.006745]
lol emoticonkiss	[0.056585]	[0.]	[0.99103]	[0.00897]
lol emoticonlaughing	[0.056585]	[0.]	[0.99103]	[0.00897]
lol lmao	[0.070359]	[0.]	[0.99103]	[0.00897]
lol lol	[0.106355]	[0.]	[0.993255]	[0.006745]
lol lol aha	[0.056585]	[0.]	[0.993255]	[0.006745]
lol lol lol	[0.056585]	[0.]	[0.993915]	[0.006085]
lol true	[0.064342]	[0.]	[0.988542]	[0.011458]
lolz	[0.075275]	[0.]	[0.5]	[0.5]
lool	[0.070359]	[0.]	[0.5]	[0.5]
monster	[0.]	[0.047475]	[0.2148]	[0.7852]
penny	[0.056585]	[0.]	[0.896307]	[0.103693]
poor baby	[0.]	[0.045795]	[0.000401]	[0.999599]
really funny	[0.056585]	[0.]	[0.83291]	[0.16709]
sad emoticonfrown	[0.]	[0.052042]	[0.002303]	[0.997697]
sad sad	[0.]	[0.048238]	[0.001492]	[0.998508]
saving	[0.056585]	[0.]	[0.180845]	[0.819155]
scum	[0.]	[0.052042]	[5.90000000e-05]	[0.999941]
seeing post	[0.045653]	[0.]	[0.199581]	[0.800419]
shame	[0.]	[0.048958]	[0.00036]	[0.99964]
sick	[0.]	[0.075971]	[0.006602]	[0.993398]
sick sick	[0.]	[0.046663]	[0.004634]	[0.995366]
smart	[0.045653]	[0.]	[0.98547]	[0.01453]
soi	[0.]	[0.045795]	[0.5]	[0.5]
stupid	[0.]	[0.058361]	[2.60000000e-05]	[0.999974]
suarez	[0.045653]	[0.]	[0.5]	[0.5]
suck	[0.]	[0.048238]	[0.006315]	[0.993685]
superb	[0.056585]	[0.]	[0.999986]	[1.40000000e-05]
terrible	[0.]	[0.052578]	[1.00000000e-06]	[0.999999]
thy	[0.045653]	[0.]	[0.949888]	[0.050112]
true aha	[0.056585]	[0.]	[0.970427]	[0.029573]
true lol lol	[0.045653]	[0.]	[0.991465]	[0.008535]
true love	[0.045653]	[0.]	[0.999669]	[0.000331]

Feature	KBest+	KBest-	Polarity+	Polarity-
woo	[0.045653]	[0.]	[0.999893]	[0.000107]
wrong	[0.]	[0.054518]	[0.01236]	[0.98764]
wtf	[0.]	[0.046663]	[1]	[0.]
yes	[0.103725]	[0.024997]	[0.375703]	[0.624297]
yes emoticonbrokenheart	[0.045653]	[0.]	[0.375703]	[0.624297]
yes emoticonsmiley	[0.045653]	[0.]	[0.375703]	[0.624297]
yes emoticonsmiley yes	[0.045653]	[0.]	[0.36733]	[0.63267]
yes want	[0.064342]	[0.]	[0.334579]	[0.665421]
yes yes	[0.135977]	[0.]	[0.36733]	[0.63267]
yes yes emoticonbrokenheart	[0.045653]	[0.]	[0.36733]	[0.63267]
yes yes want	[0.064342]	[0.]	[0.34244]	[0.65756]
yes yes yes	[0.129603]	[0.]	[0.36429]	[0.63571]

References

[1] P. Gonçalves, M. Araújo, F. Benevenuto, and M. Cha, "Comparing and combining sentiment analysis methods," in *Proceedings of the first ACM conference on online social networks*, pp. 27–38, ACM, 2013.

[2] M. Taboada, J. Brooke, M. Tofiloski, K. Voll, and M. Stede, "Lexicon-based methods for sentiment analysis," *Computational linguistics*, vol. 37, no. 2, pp. 267–307, 2011.

[3] G. Gezici, R. Dehkharghani, B. Yanikoglu, D. Tapucu, and Y. Saygin, "Su-sentilab: A classification system for sentiment analysis in twitter," in *Proceedings of the International Workshop on Semantic Evaluation*, pp. 471–477, 2013.

[4] J. Read, "Using emoticons to reduce dependency in machine learning techniques for sentiment classification," in *Proceedings of the ACL Student Research Workshop*, pp. 43–48, Association for Computational Linguistics, 2005.

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

[6] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up?: sentiment classification using machine learning techniques," in *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pp. 79–86, Association for Computational Linguistics, 2002.

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

[8] J. D. Rennie, L. Shih, J. Teevan, D. R. Karger, *et al.*, "Tackling the poor assumptions of naive bayes text classifiers," in *ICML*, vol. 3, pp. 616–623, Washington DC), 2003.

[9] J. Weaver and P. Tarjan, "Facebook linked data via the graph api," *Semantic Web*, vol. 4, no. 3, pp. 245–250, 2013.