Facebook Emotion Prediction

Project Plan

Krebs, Florian
Lubascher, Bruno G.
Moers, Tobias
Schaap, Pieter

March 24, 2017

Contents

1 Introduction

Natural Language Processing (NLP) is a field where the interaction of computers and human languages is studied. Within this field, many tasks are studied. These tasks are usually related to identifying syntax in the text or recognising its semantics.

By correctly identifying the syntax of the text, the program can go on to complete more complex tasks, since it already found the building blocks of the text. This tasks can include distinguishing nouns, verbs, adjectives and so on, which is called part-of-speech tagging. Another task could be sentence breaking, which tries to identify separate sentences. A *period* can be used to end a sentence, but also in abbreviations, so the program needs to look for further clues to what creates a sentence boundary. All tasks in this category can be used to help in the representation of text in computer friendly form.

The other side of NLP is understanding the information carried within the text. By understanding the semantics, one can have programs that answer questions, perform translations, capture the sentiment of a sentence, among many others.

The user case that we will be working with, is to perform NLP tasks on Facebook posts on a customer service page. Here, NLP can help the page owners to flag posts that require special attention. Sentiment analysis for example, could differentiate between positive and negative post, allowing the page owners to reply faster to unsatisfied customers. Also, it would be possible to figure out how the page owners reply affect the reaction from customers reading it.

1.1 Data set

Our data set consists out of Facebook posts on the customer service page of two British supermarket chains, namely Tesco and Sainsbury. These posts are mainly initiated by the supermarkets' customers. They contain written positive feedback and negative complaints. The written replies to the initial posts are also present, with the additional information of whether the reply is written by the page owner or by another user.

In addition to the written text of the posts, we have Facebook's reaction matrix [1]. Such reactions belong only to the initial post, and not to the post replies. These reactions include *like*, *love*, *wow*, *haha*, *sad*, *angry* as shown in Figure 1. These reactions were introduced by Facebook on February 24th, 2016 and allows users to express an emotion towards the post. The post replies contain a reduced matrix, where only the *like* reaction is present.

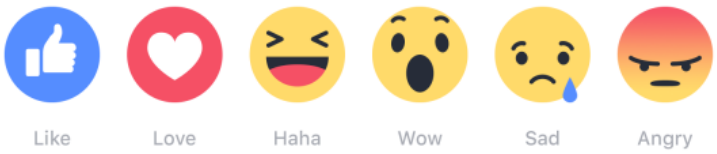

Figure 1: The Facebook reaction icons that users are able to select for an original post. Figure from Facebook (2016).

[1] http://newsroom.fb.com/news/2016/02/reactions-now-available-globally/

2 Problem Statement

In this research project the primary goal is to utilise NLP techniques in combination with deep learning in order to predict the ratio of the *reaction matrix* for a Facebook post.

NLP would be used to extract features from the original post in order to feed them into the prediction model. These features can be related to the syntax, such as the number of adjectives and verbs used, or the diversity of the vocabulary used. Features can also be related to the semantics of the post. One such feature is the post's sentiment, where based on what is written, the post gets classified as either a positive or negative review.

After processing the text the task of reaction prediction takes comes into play. For this, a Neural Network (NN) can be built, where the input is the processed and transformed text, and the output is a prediction on the number of reactions that the post will receive.

2.1 Hypothesis

The expectation is that the classifier is able to give a reasonable estimation on the ratio based on the contents of a post. Furthermore it is hypothesised that indications on absolute amount of likes are unlikely to function well as these numbers highly rely on external factors such as the amount of followers and friends that the posting person or page has. Other external factors that could influence these rates are situations where the author advertised his or her post. Such advertisements may be targeted at specific audiences and thus influence both ratios and absolute like counts.

2.2 Research goals

There are two main goals for this research project.

1. Creating a text classifier that can decide whether a post is negative or positive
2. Prediction of motions of a Facebook post of either Sansbury or Tesco page

The first goal describes the idea of creating a classifier that can decide whether a Facebook post is negative or positive. We want to achieve this by using a CNN network for text classification motivated by Kim (2014). Furthermore, an additional task will be the prediction of reactions in the form of reaction distribution. In this context, we could also try to find the impact of the page owner's response quality on the reactions by the customers.

Through these goals, the following research questions will be investigated:

1. What are the possible and efficient ways to transform text in order to feed it into a NN?
2. Can a NN be trained to predict the ratios of reaction-likes?

   How accurate are the predictions?
3. If the ratios can be predicted well, can the application be extended to give expected (absolute) counts reaction-likes?
4. How well does the application deal with posts that do not appear in the data-set?

3 Related Work

Sentiment analysis in the field of NLP is currently a popular research paradigm. An overview of state of the art techniques regarding the analysis of opinions in texts can be found in the paper of Sun et al. (2016). Various NLP approaches are discussed as well as challenges and open problems related to opinion mining.

Classical text classifications rely on human-designed features (knowledge bases and special tree kernels). Lai et al. (2015) introduces a recurrent Convolutional Neural Network (CNN) for text classification that does not rely on human-designed features. They use a Skip-gram model to pre-train the word embeddings. The recurrent part of the neural network is combined with a convolutional part containing a convolutional layer and a max-pooling layer. Lai et al. (2015) claim that their network outperforms traditional techniques and even a normal CNN.

Furthermore, Kim (2014) provides a CNN for sentence classification that also uses pre-trained word vectors. The model they use can be observed in Figure 2. Their model consists of three main layers: a convolutional layer with multiple filter widths and feature maps, a max-over-time pooling and a fully connected layer with dropout and softmax output. They use different models: the first one is the baseline model that has randomly initialized words which are then modified during training, the second one is a static model that use pre-trained vectors from word2vec, the third one is a non-static model that fine tunes the trained vectors for each task and the last model is a multichannel model that use two sets of word vectors. Their approaches outperform some techniques but Kim (2014) claim that there need to be made further adjustments and regularisations to make the best out of this models.

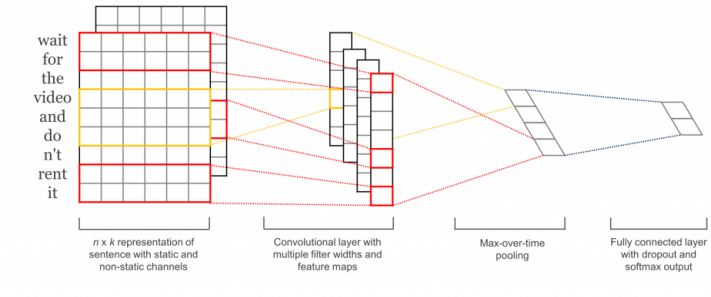

Figure 2: CNN model proposed by Kim (2014)

For the application of a Recurrent Neural Network (RNN) on textdata the paper of Perez-Ortiz et al. (2001) may prove an interesting read. In this paper a RNN is used for online text prediction. This text prediction is then used to compress texts and may even beat the classical use of a Finite State Machine (FSM).

As sentiment analysis is an important aspect of NLP, more accurate results are always welcome. The paper by Kleć (2012) shows how sparse auto-encoders can benefit the prediction of text sentiment on a NN. There, he sees the benefit on reconstructing the vectorised text input to feed it into a normal layered NN. The results of his experimentation is that auto-encoders do benefit the sentiment analysis classification task for NNs.

Lastly Arathi discusses how text imputation can be solved by the usage of a RNN. The performance of the RNN is measured by computing the Levenshtein distance between the output and the actual data.

4 Tools

4.1 Word2Vec

In difference to former approaches in which words are presented as unique ids the new approach uses vectors containing features. This representation of words as vectors is called "word embeddings". The unique ids do not provide any useful information about their dependencies to other words. Moreover, the usage of unique ids leads to data sparsity which is bad for the performance of neural networks. In difference to that the vectorised representation groups words that are similar to each other because those words will get similar feature vectors. One of the most famous examples for word embeddings is the following:

```
king - man + woman = queen
```

This simple example already shows the power of word embeddings. By subtracting the features of "man" from "king" and adding the features of "woman" the resulting vector will be close to "queen".
Word2Vec is a two-layer neural net that transforms text into vectors. It can find dependencies between words by scanning big text corpora. The output after processing such a text corpus is a vocabulary mapping a feature vector to each unique word contained in the corpus. This can be done by two different approaches: continuous bag of words (CBOW) or skip-gram. While CBOW uses the context to predict a target word, skip-gram does it the other way round and tries to predict the context given a certain word. In general CBOW performs well on small datasets while skip-gram is in favour for big datasets (Abadi et al. (2015)).

4.2 Keras

Keras is a deep learning library for Python. It provides a high-level neural net API to program modular and extensible networks. It facilitates the use of convolutional networks and recurrent networks. Both neural network configurations are widely applicable to text processing.

Keras also runs on top of TensorFlow. It has been decided to use it over NLTK, because other user's experience show that TensorFlow performs faster text mining.

Another key aspect of Keras, is its easy integration to GPU processing, which we will be using.

4.3 Ubuntu

Since some deep learning libraries require a Unix based operating system the project will be designed to run on Ubuntu 16.04 (LTS). Additionally support is widely available for Ubuntu with many guides facilitating trouble shooting if libraries do not work as expected as well as avoiding other compatibility issues.

5 State of the project

5.1 Filtering data

One step of the pre-processing of the data is the data filtering of the raw data set. The raw data set is given in so called tab-files, which are files that separate the columns

by tabs. Furthermore, the data set is split up in twelve folders (january until december 2016) for each supermarket (Tesco, Sainsbury). Each folder has two tab-files: the fullstats-file that contains the main Facebook post and the general information about this post and the comments-file that contains all the comments on that original Facebook post.
These tab files are converted into common csv-files in which the columns are separated by semicolons. Moreover, in this converting step a filter is added, which ignores all posts with pictures because one may lose important information when taking the text of those posts with pictures inside. For instance, one could think of a customer complaining about a specific vegetable and the reason why he is complaining is completely hidden in the picture. Therefore, all posts containing pictures are ignored. In addition to that, the posts that have less than twenty characters in the main post are also ignored because they may contain too less information. So, the two main filters for the first pre-processing step are: posts with a picture in it will be ignored and posts with less than twenty characters will be ignored.
One can see in Table 5.1 the reduction of the raw data, concerning only the main posts. The data sets are reduced heavily. The Sainsbury data set shrinked from 17431 Facebook posts to 10362 and the Tesco data set shrinked from 31425 Facebook posts to 17815. After filtering the raw data in a more handy file format,simple statis-

Data set	**Raw size**	**Survived size**	**Removed size**
Sainsbury	17,431	10,362	7,069
Tesco	31,245	17,815	13,430

Table 1: Reduction of raw data after filtering

tics is performed with the data. The information about how many likes, different reactions, shares and comments there are in general can be observed in Table 5.1. As

Data set	**Likes**	**Comments**	**Reactions**	**Shares**
Sainsbury	63,881	47,667	68,202	5,482
Tesco	267,705	80,203	293,968	22,532

Table 2: Main statistic of the filtered data

the data also contains the count of each Facebook reaction for each post, one can see the results of the evaluation in Table 5.1. The most used reaction is the *like reaction* and the *love reaction*.

Data set	**Like**	**Love**	**Wow**	**Haha**	**Sad**	**Angry**
Sainsbury	63,106	2,707	395	365	199	598
Tesco	264,871	20,941	1,177	1,468	677	1,641

Table 3: Reactions statistic of the filtered data

5.2 Pre-processing in Python

Before the data can be added into a word2vec model it needs to be loaded into memory to be further filtered. The current implementation provides two ways of loading the data into memory.
The first way is to load the filtered Facebook data from the generated csv files using the “DataImporter” class. This class receives the location of a zip file containing the filtered csv files and a target location at which the zip will be extracted. After the extraction of the zip files the importer will iterate over every csv file in the target directory and import it into memory.

The other way is to specify a list containing titles of Wikipedia pages which will be downloaded, pre-processed and saved using the "WikipediaPageProcessor".

The first step of pre-processing is done using several regular expression which are the following:

1. Convert string to lower case (e.g. "Word" → "word")
2. Add blanks between punctuation and words (e.g. "word." → "word . ")
3. Replace URLs with "__URL__" (e.g. "http://www.google.com" → "__URL__")
4. Replace user/profile links with "__AT_USER__" (e.g. "@User1" → "__AT_USER__")
5. Remove the hash from a hash tag (e.g. "#hashtag" → "hashtag")
6. Replace any white space with blanks (e.g. "\t" → " ")
7. Replace three or more occurrences of one character in a row with the character itself (e.g. "looooove" → "love")
8. Remove short words with less than 3 characters (e.g. "at" → "")
9. Remove sequences containing numbers (e.g. "gr34t" → "")

Afterwards the algorithm downloads a list of English stopwords to which it adds all punctuations. Then the sentence is split at every blank creating an array of words. Every word in that array that is not present in the stopwords will be added to our list of tokens. At the end this list of tokens is joined with a blank between each word.

5.3 word2vec

The current code uses gensim which is a free to use python library for unsupervised semantic modelling from plain text (Řehůřek and Sojka (2010)). It starts by loading and pre-processing a list of Wikipedia pages with the help of the "WikipediaPageProcessor" and the Facebook dataset with the "DataImporter" (see 5.2). It then creates the word2vec model with a vocabulary containing every word of the Facebook dataset that survived the filtering and pre-processing and that is used at least twice. The code loads a given pretrained model and uses its word embeddings for the created vocabulary. After the pretrained model has been loaded we continue training the model with the help of the loaded Wikipedia articles. At the end the model gets saved for later usage.
Since the pretrained model that we are loading is using vectors with a dimension of 300 we are also bound to that number. There are other models which are collected in the Github repository by Scharffe (2017) that might be worth considering. Especially the two models created by Google with a dimension of 1000 seem to be interesting and might replace the currently used GoogleNews model with a dimension of 300. Their only disadvantage is the smaller vocabulary size which is with 1.4 million only half of the GoogleNews model's size.

5.4 Graphical representation

After the word2vec model has been build, it is possible to visualise the word embeddings. This happens with the help of TensorFlow and TensorBoard. First, the model gets loaded into TensorFlow which will then create the needed files for TensorBoard. After the data was created successfully one can see a three dimensional representation in TensorBoard's "Embeddings"-tab which allows the user to see dependencies

and nearest neighbours of selected words. One can see two examples for the words “supermarket” and “Sainsbury” in Figures 3 and 4.

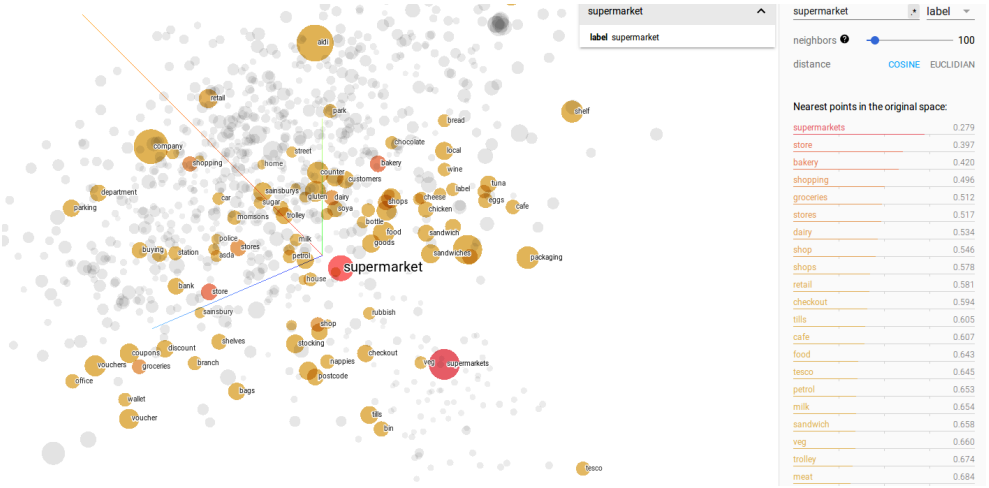

Figure 3: Nearest neighbours for the word “supermarket”

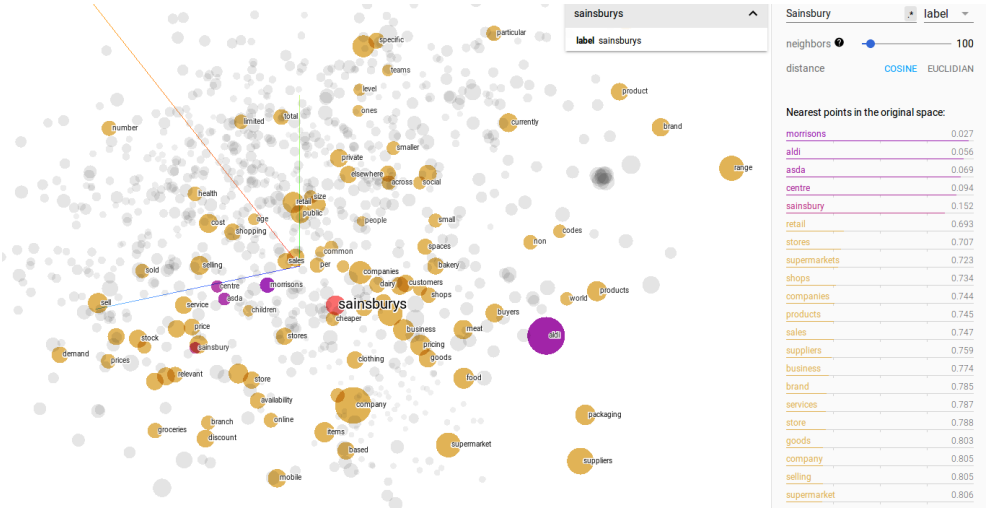

Figure 4: Nearest neighbours for the word “Sainsbury”

6 Planning

The research is broken down into several subtasks

1. Data Sanitation
   1.1. Convert the raw data to a suitable format. **Done**.
   1.2. Filter the data to keep only relevant information. **Done**.
2. Sentiment Analysis
   2.1. Perform sentiment analysis on the user post to determine whether it was a positive or negative post.
   2.2. Correlate the sentiment of the post with the reply of the page owner.

3. Reaction prediction

   3.1. Implement a deep learning network using Keras. The network should take the original post and the page owner's reply to determine the amount and types of Facebook reactions the post will receive.

4. Fine Tuning. All the models will need to be tested and experimented on to determine adequate parameter values to be used.

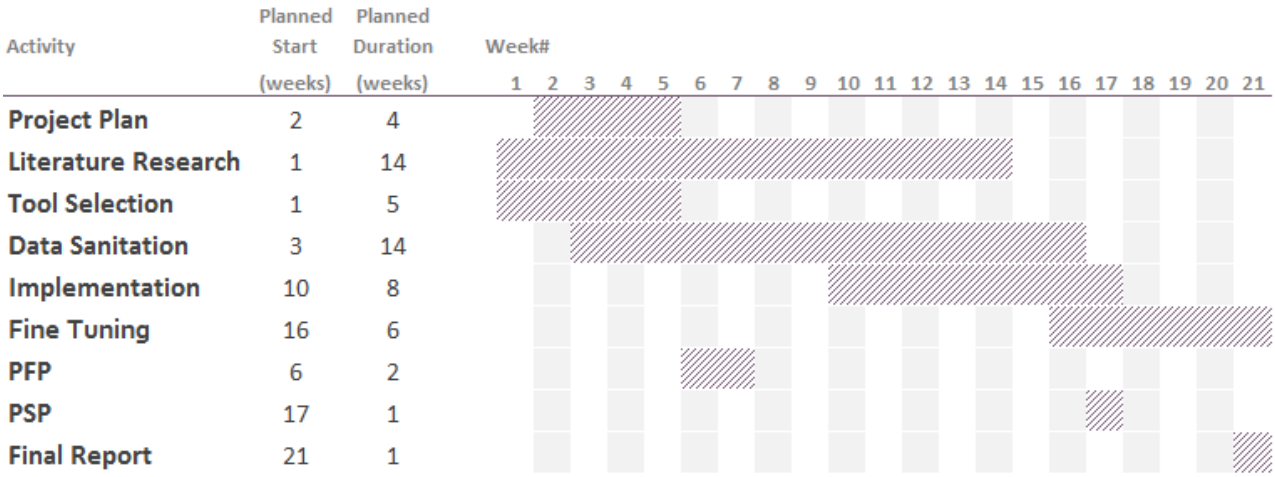

Figure 5: Planning of the project

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Arathi, M. Solving text imputation using recurrent neural networks.

Facebook (2016). Reactions now available globally. http://newsroom.fb.com/news/2016/02/reactions-now-available-globally/. Accessed: 2017-03-14.

Fan, W. and Gordon, M. D. (2014). The power of social media analytics. *Commun. ACM*, 57(6):74–81.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. *CoRR*, abs/1404.2188.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882.

Kleć, M. (2012). Sparse autoencoders in sentiment analysis. PolTAL '14, Warsaw, Poland.

Lai, S., Xu, L., Liu, K., and Zhao, J. (2015). Recurrent convolutional neural networks for text classification.

Perez-Ortiz, J. A., Calera-Rubio, J., and Forcada, M. L. (2001). Online text prediction with recurrent neural networks. *Neural Processing Letters*, 12:127–140.

Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. http://is.muni.cz/publication/884893/en.

Scharffe, F. (2017). Word2Vec pretrained models.

Sun, S., Luo, C., and Chen, J. (2016). A review of natural language processing techniques for opinion mining systems. *Elsevier*, 34:10–25.

Zhang, L. and Liu, B. (2014). *Aspect and Entity Extraction for Opinion Mining*, pages 1–40. Springer Berlin Heidelberg, Berlin, Heidelberg.