Sentiment Analysis in Twitter

Elliot Bartholme*
Korea Advanced Institute of Science and Technology
Daejeon, Korea
elliot.bartholme@gmail.com

Nam Ung Kim†
Korea Advanced Institute of Science and Technology
Daejeon, Korea
knw1121@kaist.ac.kr

Jung Choi‡
Korea Advanced Institute of Science and Technology
Daejeon, Korea
kaka00@kaist.ac.kr

ABSTRACT

Sentiment analysis is part of the various Natural Language Processing tasks and Text Mining applications, and can be summarized as *systematically identifying, classifying or extracting opinions, subjective information and affective states*. In this paper, we present a project on Twitter sentimental analysis based on the SemEval-2016 task 4. Three subtasks are considered, all belonging to sentiment classification. The problem and tasks are first stated, then a description of overall ideas and approaches is given, and finally we give explanations of issues encountered as well as results and further discussion.[1]

KEYWORDS

Sentiment Analysis, Natural Language Processing, Text Mining, Twitter, Python, Machine Learning

1 INTRODUCTION

Recently, with the rise of social networks and new forms of communication, expressing opinion and sentiments with online text has become common and ubiquitous. Among medias and social networks, Twitter is ideal as users post short and limited-length messages — a process called microblogging — that usually conveys a sentiment or an opinion about news or events around the world. Sentiment analysis is useful for automatically assessing the opinion of messages, and understand deeper how sentiment is conveyed throughout this social media.

However, one of the drawbacks about Twitter's data is its *cleanness*. As messages are limited in size, slang words and abbreviation use is common, and misspellings are often abundant. Twitter has a special and inherent use for quickly expressing opinions which leads, compared to the message length, to a large vocabulary when handled by automation processes, even though this vocabulary is in reality small when corrected or brought back to its stem (which is done mindlessly by the human brain when this latter reads a text). The preprocessing steps therefore have to be handled with care to benefit from machine learning approaches which need a systematic processing of text and features to build efficient models on top of labeled data.

The three subtasks in scope are drawn from SemEval contest and they are **(A)** single-label multi-class classification (SLMC), **(B)** binary classification and **(C)** ordinal classification (OC). To perform on these, two main approaches will be used, that are common in sentiment analysis : (i) knowledge-based techniques, often referred as lexicon-based approach, and (ii) statistical methods, often referred as machine learning approach. While the former needs domain-based facts such as lexical resources and dictionaries for predefined emotions, the latter requires a lot of data for training that must be manually labeled — for supervised learning, which can be time-prohibitive or cost-prohibitive.

Both approaches have been used and combined in a hybrid technique based on a large set of features, intended to increase the expressiveness of simple machine learning models which are often too systematic for complex tasks such as sentiment analysis.

2 TASKS DESCRIPTION

We present here each of the tasks we have in scope.

2.1 Subtask A : Message polarity classification

The aim of this task is to predict the polarity of a tweet, i.e the sentiment it carries overall. The polarity is defined by a three-class scale of sentiments: $C = \{positive, neutral, negative\}$. Different measures for evaluation can be defined, but the one adopted officially by SemEval is the average F_1 score for positive and negative class as defined in [13]. It allows better weighting for imbalanced classes, which accuracy does not perceive well, both are however reported for testing.

2.2 Subtask B : Tweet classification according to a two-point scale

This time, the goal is to make a binary classification of tweets, but knowing the topic it is about. Thus the task is to predict whether the tweet conveys a positive or negative sentiment towards the topic : $C = \{positive, negative\}$. The evaluation measure chosen here is the *macro-averaged recall* over *positive* and *negative* classes[13].

This task has been approached as a simple binary classification, and no distinction between topics has been made. This has been supported by previous work and promising results as soon as the trials began, as well as the fact that the dataset comprises only of one label for each tweet and thus does not make a strict distinction between aspect-level and overall sentiment.

*In charge of python development, documentation, model selection, model testing, lexicon aggregation, statistics and data analysis, results reporting, doc2vec training and writing

†In charge of preprocessing with cleaning twitter specific elements and negated contexts, feature engineering with added Bing Liu lexicon features and elongated words, model testing, and writing

‡In charge of investigating previous researches on WSD, evaluating possible WSD methodologies, designing/testing graph models, designing/testing Word2Vec-based approaches, model traning and writing

[1]The design paper as well as source code can be found in https://github.com/elliotbart/SemTweet

2.3 Subtask C : Tweet classification according to a five-point scale

Once again, the aim is to know which sentiment a tweet communicates on a given topic, but this time with a five-point scale, ranging from highly negative to highly positive : $C = \{highlypositive, positive, neutral, negative, highlynegative\}$, or $C = \{2, 1, 0, -1, -2\}$ in a numerical translation. The evaluation measure is the *macro-averaged mean absolute error* (*MAE*) [13] :

$$MAE^M(h, Te) = \frac{1}{|C|}\sum_{j=1}^{|C|}\frac{1}{|Te_j|}\sum_{\mathbf{x}_i \in Te_j}|h(\mathbf{x}_i) - y_i| \qquad (1)$$

where Te_j is the set of documents with class j and h($\mathbf{x}_i$) and y_i respectively are the predicted label and the true label of tweet i. Macro-averaged is robust to class imbalance compared to standard *mean absolute error* and with perfectly balanced datasets it becomes equivalent, so it appears to be a better method of error scoring.

Furthermore, as it is an error measure, lower values are logically better. Its advantage is to well take into account the differences of classification errors compared to *SLMC* task : all do not weigh equally, for instance a tweet labeled *highly positive* instead of *highly negative* should be given special attention compared to a tweet labeled *highly positive* instead of *positive*, and penalized more. Finally, as for subtask B, topics have been carried out without distinction, and the problem has been handled as a conventional multi-class approach.

3 DATA ANALYSIS

The dataset consists in different text files with either format ***text***<tab>***label*** or ***text***<tab>***topic***<tab>***label***. Following is a summary table of repartition of tweets across training and testing data across tasks. No validation data has been provided.

	Subtask A	Subtask B	Subtask C
Training tweets	11,692	7,860	8,048
Testing tweets	16,398	8,335	16,400

Table 1: Number of tweets provided for each task

3.1 Class distribution

The first analysis of these datasets reveal a high class imbalance for each task in the training data, see figures 1, 2, 3.

It should be noted that training set of task B contains neutral tweets (contrary to the expected subtask), but the test set doesn't so those tweets can be used for training on a further binary evaluation.

Negative class is always under-represented and should therefore be granted special attention. Any simple machine learning model would by nature have preponderance to label any tweet as positive once the system is trained. This is for that matter that averaged F_1 score is much more suitable than accuracy for evaluation.

3.2 Topics

If we inspect deeper the tweets, we see that topics are various : 274 for training set B and 100 for training set C. They represent

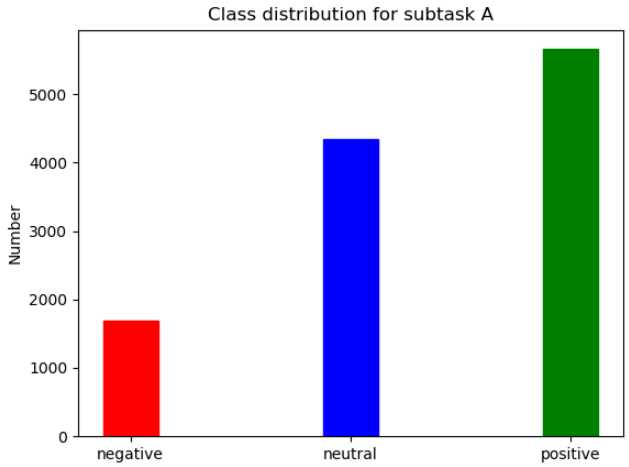

Figure 1: Class distribution in training set A

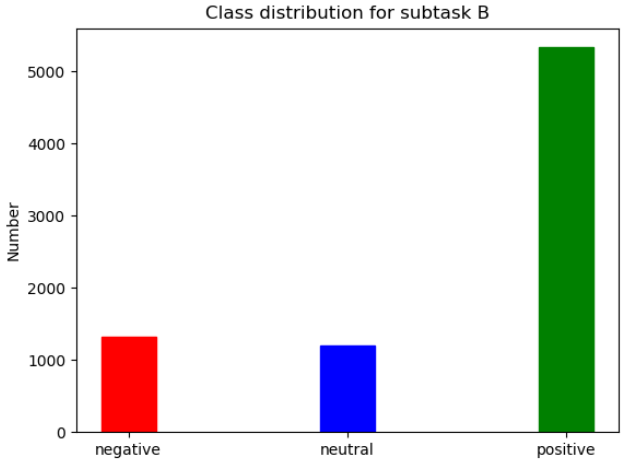

Figure 2: Class distribution in training set B

celebrities, political personalities, brands, artists and others. Most common topics are presented in table 2:

Data	Topics		
Train B	*tgif*	*bob marley*	*ac/dc*
Train C	*tgif*	*google+*	*tiger woods*

Table 2: Three most popular topics for tasks B and C

Those topics are often highly related to a specific label as the tweets have been retrieved in the same period, where the overall thoughts for an entity are more or less the same for the majority of Twitter users. The tables 2 and 3 demonstrate it.

We see that dealing with celebrities in general brings more positive opinions than controversial topics such as Monsanto. The issue is that positive topics are much more numerous than the others, which brings the imbalance we saw previously.

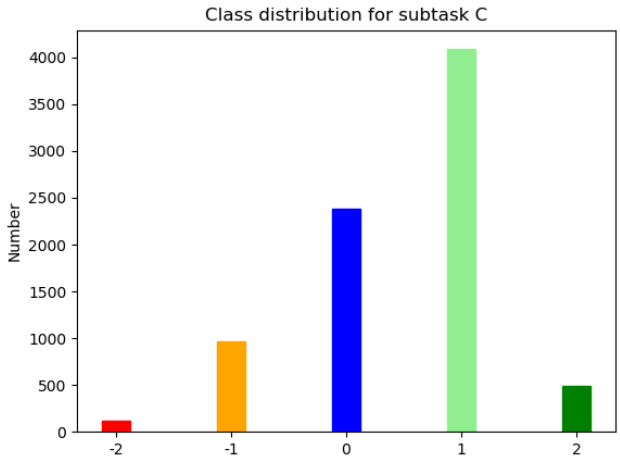

Figure 3: Class distribution in training set C

Topic	Positive	Negative
tgif	79	2
bob marley	80	1
ac/dc	77	2
monsanto	7	49

Table 3: Polarity distribution of most common topics in task B

4 FEATURE EXTRACTION

In this section, we will describe the details of the features we chose in order to achieve better performance in the classification task. A good set of informative features provides more interpretation, is less time-consuming and computation-costing than building the most sophisticated machine learning model. Our approach was to experiment many existing approaches and hopefully come up with the best combination among those, using some logic and intuition around sentiment expression in the language.

We combined classic features in text classification such as bag of words with sentiment specific features like lexicon based scores which are expected to give fair and better clues to discriminate sentences into each sentiment category.

4.1 Tokenizing process

Our tokenizer is based on word extraction, with specific features inherent to Twitter. We chose to keep punctuations symbol and to include emoticon tokens as they are shown to be used in different sentiment contexts. We normalized users mentions (id) and urls to the same token. Hashtags are extracted, and english contractions are retained thanks to useful regex. The tokenizer also proposes different options including stemming, lemming, lowercasing, and is easily adaptable. We also chose to handle negation context to benefit from their informativeness in sentiment analysis. A negated context is defined between any negation mark (*don't, can't, not, no...*) and a punctuation mark.

Different tokenizing approaches have been tried, but they do not yield significant results maybe due to the limited size of training data, and are highly correlated to the data itself, meaning that no proceeds gives universally the best results for each task and set of tweets.

Once text is tokenized, we construct a bag of words on top of the training data, and we used three ways of weighting each word : raw term frequency, binary presence or absence and *tf-idf* (which generally performs best).

4.2 Custom features

A set of *handmade* and Twitter specific features have been designed according to our intuition and previous work on sentiment analysis. They include the number of exclamation marks, interrogation marks, quotes, uppercases letters, positive and negative emoticons, ellipsis, mentions, hashtags, urls and elongated words. We also counted the number of each of the four pos tags used by WordNet (noun, verb, adjective, adverb).

Many of them have been shown to be helpful in the discrimination process, see figures 4, 5, 6. However, what is striking is the difference between discriminative features in favor of positive class and negative class (counts are over-sized for the positive label) which will not help that much given that the negative class is the one that presents the most challenge.

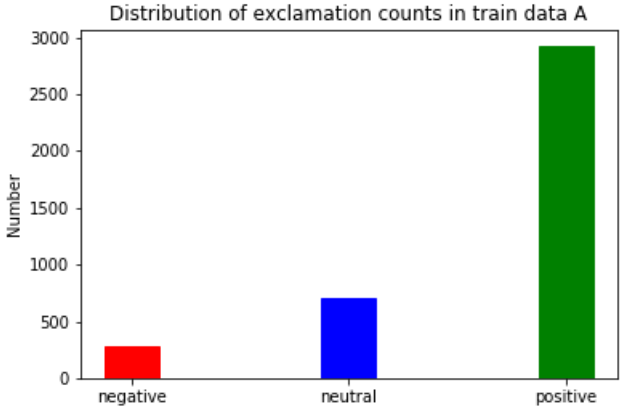

Figure 4: Distribution of exclamations marks in train data A

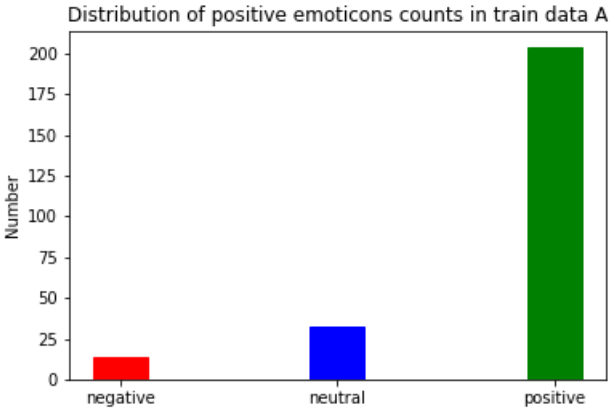

Figure 5: Distribution of positive emoticons in train data A

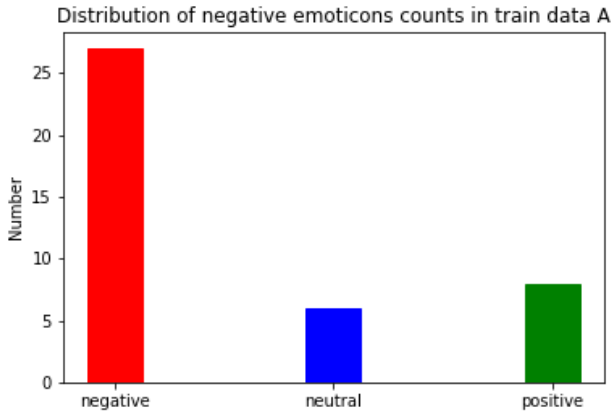

Figure 6: Distribution of negative emoticons in train data A

Finally, we present a graph in figure 7 showing the 15 best tokens according to chi2 score selection, which is highly dominated again by words in favor of the positive class.

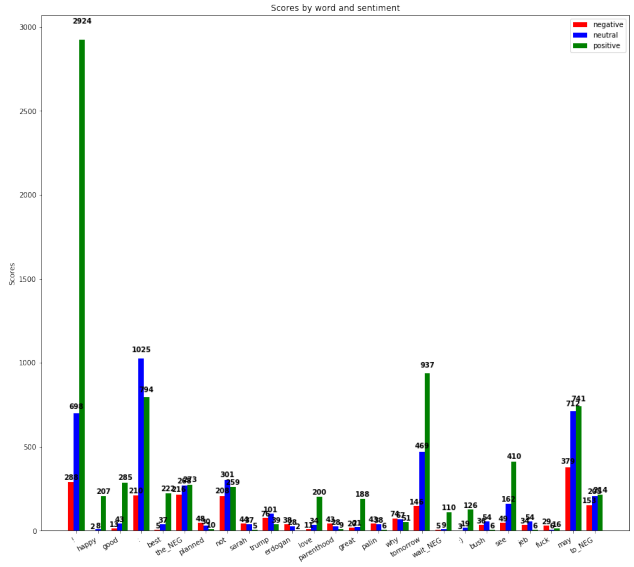

Figure 7: Distribution of best tokens in train data A

4.3 Lexicon-based sentiment score computation

We experimented the SentiWordNet lexicon, which is a vocabulary set generated with both manual selection and semi-supervised data augmentation. SentiWordNet offers polarity scores (positiveness, objectiveness, negativeness) that are connected to each word synsets in the WordNet data corpus. So, we had to first disambiguate each word tokens and figure out the scores of the corresponding synsets.

However, the pre-defined positive, negative, and objective scores of SentiWordNet did not seem to give reasonable semantic clues for each word used in tweets, so we decided to try a better solution to come up with sentiment scoring. We describe more of this procedure in the upcoming section.

Apart from unsatisfying quality of SentiWordNet scores, we tried splitting the positive and negative score features into more detailed subsets. We split them into four categories, which are POS-POS/POS-NEG/NEG-POS/NEG-NEG. Each respectively corresponds to the positive sentiment in a unnegated context, positive sentiment in a negated context, negative sentiment in a unnegated context, and negative sentiment in a negated context. These subsets are expected to give better clue to the learning models. For each word synset, we first checked if the word is negated, and looked up the three SentiWordNet scores(positive/negative/objective) to add the value to the corresponding feature subset.

Secondly, we used Bing Liu opinion lexicon, which consists in an list of English positive and negative words. The process is very similar excepted that counts instead of scores are provided. Here, we also used the same sub-categorizing technique to get four features (POS-POS/POS-NEG/NEG-POS/NEG-NEG).

Another used lexicon is the Sentiment140 unigram (62,468 unigrams) and bigrams (677,698 bigrams) which provides negative and positive scores, and which was automatically designed on Twitter data by extracting emoticon-labeled tweets [11].

Other lexicons include NRC Word-Emotion Association Lexicon [12], MPQA [17], ANEW [2] and AFINN [14]. The main idea with using all these different resources is to extract more powerful indicators about the polarity of a tweet, with varied ways to aggregate scores or counts along with different formats and scores.

The figure 8 shows that some of our lexicon based features successfully bring discrimination across classes. However those results must be treated cautiously because chi2 score is biased by high occurrences and the counts on the graph are not scaled.

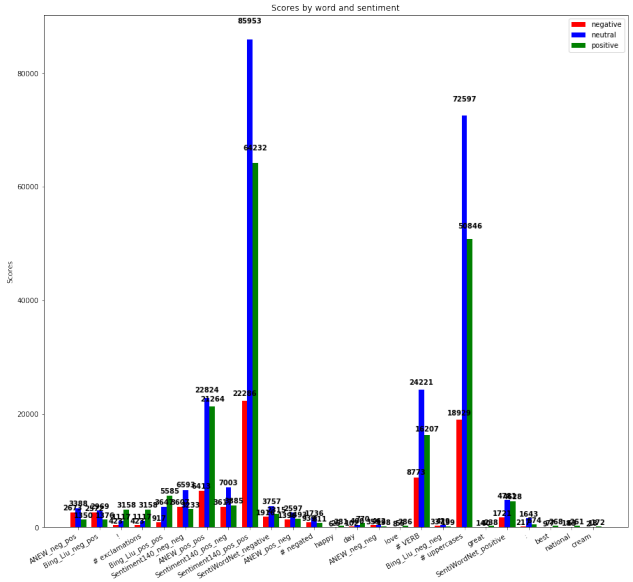

Figure 8: Distribution of best features and tokens in train data A

4.4 Distributed word representations

Tomas Mikolov et al. [9] showed that distributed word representation reconstruct linguistic contexts of words better than any other representation, and allow state-of-the art in several Natural Language Processing applications. We therefore used pretrained word embeddings on twitter data in order to use them. We computed average representation between each vector of each word for a tweet, and use the result as feature for a linear classifier. It did not yield good results.

Quoc Le and Mikolov [7] came with distributed representations of sentences and documents, finding directly embeddings at a sentence or paragraph level and thus showing better results than the bag-of-words model which fails at capturing relationship and ordering between words. We therefore trained their model on our data and used these embeddings as straight features for a linear classifier. Once again, it did not produce good results.

Doc2vec limitations in our case are our training data which is clearly insufficient to train accurate embeddings. We confirmed this result be increasing of 3% the accuracy and f-score each time we doubled our training data (using the other tasks). However, the negative class stayed at a disappointing 0 recall and precision. Using word2vec, results were similar.

Using word embeddings for sentiment analysis has been showed to often fails because close embedded and semantic words are sometimes distant in terms of sentiment polarity (this is notably the case for *good* and *bad* which are extremely correlated in terms of cosine similarity but obviously contradictory for sentiment analysis). Maas et al. [8] and Yu et al. [18] developed models to either directly learn better word representation for sentiment analysis or to refine the representation, moving sentiment opposed words in the other direction by an clever objective function. We though did not have sufficient time to put them into practice.

4.5 Limitation of using basic word sense disambiguation function

To get the correct semantic property of each word, we need to use a process named *word sense disambiguation*. A basic algorithm implemented in NLTK library to do so is the Lesk algorithm. We set different distance functions for the algorithm and compared each. Although none of them gave satisfactory results, we found that adapted distance based Lesk algorithm [1] was the best among all.

However, since it is a crucial point to infer accurate meaning of each word, we decided to use a more advanced algorithms that can assign proper *synset* for each word by analyzing at the sentence level.

There have been a large number of researches that performed better on this task, using supervised learning methods, LSA, LDA, graphical models, recurrent neural networks, or skip gram models. However, we chose to use a traditional method based on maximizing similarity between synsets in WordNet. We experimented 2 similarity functions, which are Resnik similarity[16] and JiangâĂŞConrath similarity[4]. The former one is based on shared information between synsets and gives poor precision and good recall results. We chose this feature because a good recall result would produce better input to the following sentiment analysis step because it would simply give a larger number of data that can be used as features.[3] The latter one uses the hierarchical information if WordNet and computes the conditional probability of encountering a child-synset given a parent synset. It was chosen because it performed the best among all similarity measures in correlation with the human rating measure.

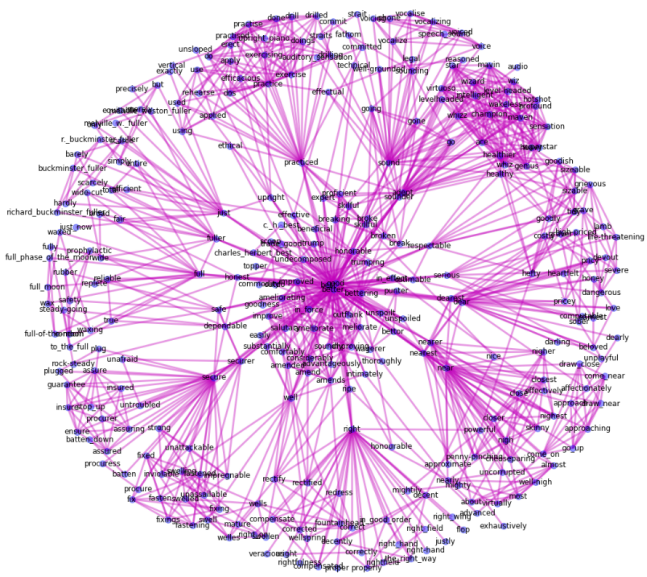

Figure 9: A subgraph connected to word 'good' in the basic WordNet graph based on taxonomy structure.

4.6 Computing sentiment score derived from semantic relation inside WordNet corpus

In hope of getting better feature other than simply counting the appearance of each synset in any of the sentiment lexicon datasets we listed, we tried a new approach using a graphical model we built with a 'dictionary' concept. In WordNet corpus, there is a dictionary definition sentence for each synset. This information can extend the relationship between all the synsets. The original semantic distance functions used in shortes path detection are based on *'hypernym-hyponym'*, *'antonym'*, or 'word-all corresponding synsets' relationships which are already defined in the WordNet corpus. However, this relationship is very limited and cannot discover rich possible semantic relationships between word synsets.

A basic graphical model built with WordNet data was introduced in [5]. It uses the basic synset relationship structure defined in WordNet. That is, each text word in the corpus forms a node, and is connected to all other words that share any synset. The distance algorithms of NLTK library used in the word disambiguation phase are based on this graphical model. We first tried the same simple graphical model based on the plain word-synset relationship which was suggested in the paper.[5] The result is shown in Figure 7.

They constructed relations at the level of plain words. Basically, what they tried was to set up a graph of words where all words are connected with their synonyms by edges. To get synonyms, they

used the WordNet taxonomy structure and simply connected all words that share the same synset.

However, this approach can only give ambiguous relationship between words since every word has multiple synsets that can possibly have totally different definitions. So our first approach was to build a graph based on synset list. All synsets will be connected with any other synsets that share the same lemmas(the most specific unit that have the plain word and synset information).

As described above, the default path similarity measure in NLTK library uses the taxonomy tree structure which does not strongly cover the semantic relationship between synsets. Therefore, its ability to distinguish the semantic difference is not strong. For our Tweet sentiment classification task, we tried computing each tweet word's shortest path distance to manually selected set of positive words and negative words, though it did not give a reasonable outcome.

To address this problem, we benchmarked a dictionary-based word embedding approach and tried making up a solution that fit our task. The idea was to use the definition of each word synset to discover richer relationship between the words in the corpus, so that any word could be better figured out to have either positive or negative semantic characteristics.

In a more recent paper[6], the dictionary definitions were efficiently used to guide the word sense disambiguation problem while computing the word embedding simultaneously. While most current algorithms heavily depend on the corpora they use and often have out-of-vocabulary problem, this dictionary-based model is free from those issues and expected to have more precise and authoritative description of each word's semantics.

Their idea is to create a vector embedding of synsets in WordNet corpora by simultaneously disambiguating definition sentence and training a recurrent neural network for embeddings computation.

They model a disambiguation phase for each word in the definition sentence by message-passing algorithm that is based on the conditional probabilities. They seek to find a new similarity metric that is used for co-occurring synsets in a single definition sentence, and these similarity values will eventually be passed to the conditional probability of synsets for each plain word in the definition sentence. This computation is injected inside the training of a recurrent neural network model with some additional sophisticated terms for a better computational result.

We have not reimplemented the whole system, but acquired the idea of using definition sentence to catch better relationship between all synsets. Instead of using neural network to figure out a vector space which is out of the scope of our class, we chose to rely on existing word sense disambiguation functions to set up the basic proper semantics of words from any sentences. That way, we hoped to build a new graph model that connects semantically related synsets.

In our experiment, we simply disambiguated all the words in the definition sentence using adapted lesk algorithm, and applied to the graph structure of the whole WordNet synsets. That is, we created nodes for every synset and connected them by *'synset-definition'* relationship. Negated words were replaced by their antonyms. Our assumption was that any words that appear in the definition sentence of a certain word will be semantically close to it. However, as we will mention later in this section, results did not come out as we expected.

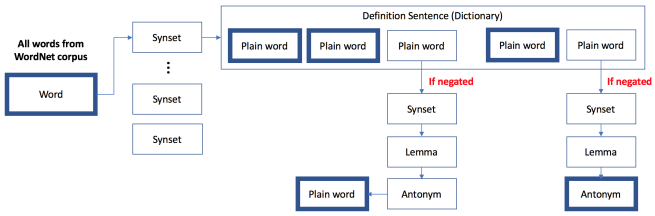

Figure 10: Input prepared for a *Definition2Vec model*. A word and words from its definition sentence were gathered together to be used as inputs for a Word2Vec architecture model.

Therefore, we switched from graph model to the Word2Vec approach.[10] We came up with two different approaches. One was to simply put a word and its definition sentence together and train a *Definition2Vec* model on them. (Figure 10) Since Word2Vec models are based on the assumption that closely located words are to be related semantically, all the words in the definition sentence and the word will have closer location in the extracted vector space.

For the second approach, we used the disambiguated synsets as inputs to the model. For every definition sentence of a synset in the WordNet corpus, we performed the word sense disambiguation process, and converted each word to a synset that is expected to have a correctly related semantic. Then we used them as input to the *Synset2Vec*, which seek to figure out a vector space where related synsets are close together. (Figure 11)

To compute the sentiment score of a sentence using these models, we selected a set of positive words and a set of negative words which we expected to have strong signal for sentiment classification. 69 positive words and 74 negative words were selected to be used as references for semantic location inside the vector space. The antonyms for each word are also added to the reference list. Also, we manually selected the correct meaning of each word and gathered them as a reference set of synsets(both positive and negative synsets). For WordNet graph model, we simply computed the average distance from each of these reference sets and subtracted from each other to get the final sentiment score. In *Definition2Vec* model, all words were converted to a certain vector using the model, and the vector similarity from each of reference set was used as a sentiment measure. For *Synset2Vec* model, we first disambiguated every word in the tweet, and calculated the vector similarity from the reference sets of positive/negative synsets. However, the result was still not satisfying. Logistic regression using the score from each of four models as a feature showed 48.38% of accuracy which is actually the same number we can get from simply choosing 'positive' all the time. Therefore, we decided to drop these as features.

5 CLASSIFIERS

To build sentiment classifiers for the 3 tasks we were tackling, we tried various different supervised learning algorithms. We tried out

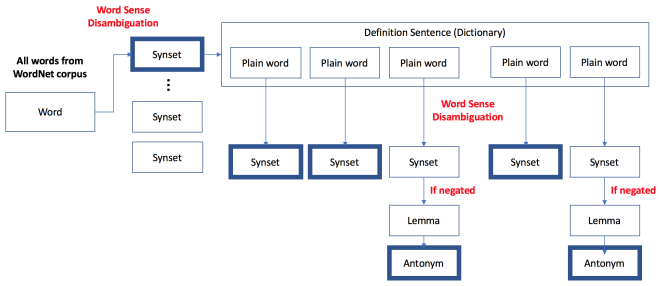

Figure 11: Input prepared for a *Synset2Vec model*. A word and words disambiguated word(synset) and all the synsets disambiguated from its definition sentence were gathered together to be used as inputs for a Word2Vec architecture model.

Naive Bayes (Multinomial and Bernoulli), Logistic Regression, Support Vector Machines, k-Nearest Neighbors, and Neural Networks.[15] Our approach was to try different models to train on the features we extracted, and experiment which gives out the best results. In this section, we briefly discuss the classifier we experimented.

5.1 Naive Bayes

The concept behind the Naive Bayes approach is to apply Bayes' theorem while naively assuming the independence between the features considering classes. When we are trying to classify a certain tweet to class C_k with the given vector of features $\mathbf{x}$, we would be looking for the probability of $P(C_k|\mathbf{x})$. For this probability, Bayes' theorem gives the following equation.

$$P(C_k|\mathbf{x}) = \frac{P(C_k)P(\mathbf{x}|C_k)}{P(\mathbf{x})}$$

Now, if we assume naively that every pair of features is independent,

$$P(x_i|C_k, x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) = P(x_i|C_k)$$

the above equation can be simplified like this:

$$P(C_k|x_1, ..., x_n) = \frac{P(C_k)\prod_{i=1}^{n} P(x_i|C_k)}{P(x_1, ..., x_n)}$$

Once given an input, the $P(x_1, ..., x_n)$ is constant, so we can come up with the following :

$$P(C_k|x_1, ..., x_n) \propto P(C_k)\prod_{i=1}^{n} P(x_i|C_k)$$

The above is the Naive Bayes probability model, to turn this in to classifier, we pick the hypothesis that is most probable with the Maximum a Posteriori (MAP) decision rule. Below is the Naive Bayes classifier which assigns $\hat{y} = C_k$ for some k.

$$\hat{y} = \underset{k\in\{1,...,K\}}{\text{argmax}}\ P(C_k)\prod_{i=1}^{n} P(x_i|C_k)$$

Different Naive Bayes classifiers exist regarding different assumptions of the distribution of $P(x_i|C_k)$. These variations are Gaussian Naive Bayes, Multinomial Naive Bayes, and Bernoulli Naive Bayes. We dismiss the Gaussian Naive Bayes approach for it concerns dealing with continuous data, because our features have discrete values (mainly counted numbers). Although the assumption for Naive Bayes shows significant weakness by over-simplification, it is known to work well especially with document classification.

5.2 Logistic Regression

Logistic regression is a linear model for classification in which the probabilities of possible outcomes from a single trial are described in by a logistic function of a linear combination of features. The implementation of logistic regression either handle binary (logit odds), one-vs-rest (multi logit odds), and multinomial (MaxEnt) classification. We mainly used L2 regularization and one-vs-rest for our experiments with logistic regression, as they produced the best results. The L2 penalized logistic regression model minimizes the cost function:

$$\min_{w,c} \frac{1}{2}w^T w + C\sum_{i=1}^{n} log(exp(-y_i(X_i^T w + c)) + 1)$$

The solver used is *liblinear*, which is a solver using a coordinate descent algorithm along with decomposing the problem into separate *one-vs-rest* binary classifiers. So for Task A and Task C there would be 3 and 5 of such under the hood classifiers at work. Newton-Raphson's iterative algorithm is used when modeling probabilities in a multinomial way (max entropy assumption). Logistic regression is a highly interpretable model and generally offers very good results.

5.3 Support Vector Machines

A Support Vector Machine (SVM) model represents the given data as points in space, the resulting map of different points is then divided by a margin as wide as possible to distinguish classes. New examples provided will then fall into either side of the gap and be classified accordingly. SVC supports multi-class classification with the *one-versus-one* approach, which means that given n classes, all possible pairs are considered, so $\frac{n(n-1)}{2}$ classifiers are constructed.

Below, we briefly discuss the mathematical formulation of the Support Vector Classifier implemented in *scikit-learn* library.

When given training vectors $x_i \in \mathbb{R}^p$, i=1,...,n, for two classes, with the vector of corresponding labels $y \in \{1, -1\}^n$, SVM solve the primal optimization below.

$$\min_{w,b,\xi} \frac{1}{2}w^T w + C\sum_{i=1}^{n} \xi_i$$

$$subject\ to\ y_i(w^T\phi(x_i) + b) \geq 1 - \xi_i,$$
$$\xi_i \geq 0, i = 1, ..., n$$

The dual problem is written below, with e being a vector of all ones. $C > 0$ being the upper bound, and Q an $n \times n$ positive semidefinite matrix, $Q_{ij} \equiv y_i y_j K(x_i, x_j)$, where $K(x_i, x_j) = \phi(x_i)^T\phi(x_j)$ is the kernel. Function ϕ maps the training vectors into higher dimensions.

$$\min_{a} \frac{1}{2}\alpha^T Q\alpha - e^T\alpha$$

$$subject\ to\ y^T\alpha = 0$$
$$0 \leq \alpha_i \leq C, i = 1, ..., n$$

This is the decision function.

$$sgn(\sum_{i=1}^{n} y_i \alpha_i K(x_i, x) + \rho)$$

SVM have been shown to work very well in text classification as they can handle high dimensional feature spaces, and are therefore well suited for our tasks. We used only linear kernels as gaussian gave really low scores.

5.4 k-Nearest Neighbors

The Nearest Neighbors approach in classification is to output a class membership for a given input data according to a majority vote by its nearest neighbors. A new input is assigned to the dominant classe from its k nearest neighbors. Optimal choice for the value of k differs between data, so we tried out a range from 1 to 10 as our candidates for the value k.

5.5 Neural Networks

We also tried out a multi-layer perceptron neural network. A multi-layer perceptron network learns a function $f(\cdot) : R^m \rightarrow R^o$ from the training data. It starts with an input layer and ends with an output layer, and in between the two are non-linear layers which are called hidden layers. Thanks to these hidden layers, such a network is able to learn non-linear function of its input. MLP requires hyper-parameters such as the number of hidden layers, which we tried out with different sizes to find the best results.

One has however to be careful that neural networks need important scaling in features, especially when they are highly different as in our model where we combine custom modeled features with words term frequencies, *tf-idf* or binary indicators.

5.6 Model

Our model consists in taking a list of n tweets in input, tokenizing each of them into a list of tokens, and then create a bag-of-words (often *tf-idf* in our final model). In parallel, we compute the other features that we designed, and call the lexicons using well designed data structures to extract scores.

We produce features vectors that have the size n of the input data and sequentially stack them to the bag of words. Everything is done in a pipeline, which offers two main benefits : we obtain a useful modularity that allows to quickly design new features extractors and put them into the pipeline whenever desired, and secondly it optimizes our cross validation processes.

Indeed, it is important to extract features only based on the training data, and it comes really important in our model to be careful not to leak some vocabulary or information from the test folds by creating features before the splits. This is often underestimated in validation processes but is really important to avoid bias and overoptimistic metrics.

6 RESULTS

Our goal was to maximize macro-averaged F_1 score of positive and negative class for subtask A, macro-averaged recall for subtask B, and macro-averaged mean absolute error for subtask C. In this section we present the results we obtained and compare some models.

To obtain results of table 4, we use a stratified cross validation strategy of 3 folds randomly shuffled and report the mean and variance of the macro-averaged F_1 score for 4 iterations. The only features used were unigrams bag of words using different weighting schemes. LR, SVM, KNN, NBB and NBM refer respectively to logistic regression, support vector machine, k-nearest-neighbors classifiers, bernoulli naive bayes and multinomial naive bayes, *bow* refer to bag-of-words. Procedure is the same for tasks B and C, tables 5 and 6, but reporting the appropriate measures: ρ^{PN} and *MAE*.

The aim of this first approach is to construct a *baseline* using the simplest feature model possible and default parameters for all classifiers. We also reported scores of naives classifiers, familiarly called *dummy*, which classify with simple rules like the most frequent label, randomly or with a lexicon score : *dmf* refer to most frequent classifier, *dmr* refer to random classifier, *dml* to lexicon classifier[2]. The results of these special classifiers are intended to provide a really low baseline to absolutely come over. The type of bag-of-words has no influence on their results as it is not used, so we provided the same score for the random *dummy* classifier even though its output change (explaining a small variance for it).

	tf bow	*binary bow*	*tf-idf bow*
LR	0.531 (+/- 0.010)	0.513 (+/- 0.007)	0.446 (+/- 0.006)
SVM	**0.532** (+/- 0.005)	0.53 (+/- 0.009)	0.494 (+/- 0.009)
KNN	0.36 (+/- 0.009)	0.35 (+/- 0.01)	0.4696 (+/- 0.013)
NBB	0.401 (+/- 0.002)	0.400 (+/- 0.001)	0.401 (+/- 0.003)
NBM	0.481 (+/- 0.003)	0.483 (+/- 0.007)	0.348 (+/- 0.001)
dmf	0.326	0.326	0.326
dmr	0.299 (+/- 0.012)	0.299 (+/- 0.012)	0.299 (+/- 0.012)
dml	0.341	0.341	0.341

Table 4: Cross validation mean averaged F_1 score for task A using basic features

	tf bow	*binary bow*	*tf-idf bow*
LR	**0.689** (+/- 0.009)	0.687 (+/- 0.004)	0.593 (+/- 0.007)
SVM	0.681 (+/- 0.008)	0.682 (+/- 0.013)	0.667 (+/- 0.001)
KNN	0.482 (+/- 0.008)	0.484 (+/- 0.007)	0.614 (+/- 0.011)
NBB	0.544 (+/- 0.003)	0.543 (+/- 0.009)	0.545 (+/- 0.002)
NBM	0.640 (+/- 0.006)	0.641 (+/- 0.002)	0.500 (+/- 0.000)
dmf	0.5	0.5	0.5
dmr	0.335 (+/- 0.022)	0.335 (+/- 0.022)	0.335 (+/- 0.022)
dml	0.372	0.372	0.372

Table 5: Cross validation mean averaged *recall* score for task B using basic features

Having only train data, we had to develop a strict evaluation and selection process based on cross validation. We used nested cross validation to get a good estimate of each model performance and cross validation grid search to perform fine-tuning of the hyper-parameters for the selected best models. Logistic regression, naive bayes and svm have shown to perform the best on our train data. This is why we selected these models to report the result of our final scores using the full set of features and tuned classifiers.

[2]the lexicon classifier counts the number of words of each label according to ANEW and gives the most represented class to the tweet

	tf bow	*binary bow*	*tf-idf bow*
LR	0.992 (+/- 0.017)	0.984 (+/- 0.011)	1.135 (+/- 0.017)
SVM	**0.948** (+/- 0.008)	0.955 (+/- 0.018)	1.031 (+/- 0.014)
KNN	1.286 (+/- 0.017)	1.278 (+/- 0.006)	1.034 (+/- 0.017)
NBB	1.200 (+/- 0.010)	1.197 (+/- 0.005)	1.204 (+/- 0.008)
NBM	1.105 (+/- 0.014)	1.110 (+/- 0.013)	1.351 (+/- 0.005)
dmf	1.4	1.4	1.4
dmr	1.606 (+/- 0.053)	1.606 (+/- 0.053)	1.606 (+/- 0.053)
dml	1.133	1.133	1.133

Table 6: Cross validation mean averaged *MAE* score for task C using basic features

Test data has been released two days between the deadline, but we only used it to report our results with the three classifiers chosen by cross validation during training, and their parameters. In no case, these results reported on test data allow to select a model as the best one, since it would be a completely biased process and the model would become overfitted. Normally, we should have released only the result of our best system to avoid tempting deductions on test data, but we did it to present a larger panel of models.

	Train A	*Train B*	*Train C*
LR	**0.589** (+/-0.010)	**0.728** (+/- 0.0011)	**0.907** (+/- 0.016)
SVM	0.569 (+/- 0.008	0.698 (+/- 0.008)	0.923 (+/- 0.015)
NBM	0.519 (+- 0.002)	0.654 (+/- 0.001)	1.127 (+/- 0.004)

Table 7: Cross validation scores (F_1^{PN}, ρ^{PN}, MAE) for associated tasks on train data, using full features set and tuned models

	Test A	*Test B*	*Test C*
LR	**0.506**	0.638	0.987
	0.552	0.735	0.465
SVM	0.505	**0.674**	**0.945**
	0.54	0.722	0.456
NBM	0.43	0.58	1.245
	0.488	0.789	0.434
dmf	0.0[3]	0.5	1.2
	0.505	0.786	0.492
dmr	0.271	0.342	1.620
	0.334	0.339	0.2
dml	0.326	0.598	1.133
	0.504	**0.794**	**0.502**

Table 8: Scores for each task on test data, using full features set and tuned models used in table 7. The score inherent to the task (F_1^{PN}, ρ^{PN}, MAE) is reported on the first line and accuracy on the second

We report on table 8 the results of our three best models, along with three naive classifiers, using bigrams. Finally, our highest results are in table 9, using full model with trigrams. These final results give interesting thoughts. Some are deceiving while some are rather good. We see that in general *dummy* classifiers produce good accuracy especially for task B where we obtain an impressing 0.794 accuracy. However, our models perform best to increase the metric which matters. Our efforts to increase the metrics of negative class show good results especially with logistic regression on task B.

	Test A	*Test B*	*Test C*
LR	0.5634	**0.6917**	0.9707
	0.629	**0.823**	0.502
SVM	**0.5667**	0.6774	**0.9273**
	0.62	0.811	0.511

Table 9: Highest results on test data using trigrams. The score inherent to the task (F_1^{PN}, ρ^{PN}, MAE) is reported on the first line and accuracy on the second

Nevertheless we note that the lexicon classifier yields really respectable scores even for F_1 recall and *MAE*, where it outperforms multinomial naive bayes classification and produces a high 59.8% for ρ^{PN}. It shows that the ANEW lexicon is efficient and that naive scoring classification can still be competitive. Regarding deceiving naive bayes scores, we may assume that given our highly correlated and dependent features the model cannot benefit from them.

We finally report on tables 10, 11 and 12 the performance on the test set of the classifiers used in table 8, for each task, using only a simple bag-of-words and no parameters optimization from the train data. It is provided to show how our features and model-tuning have allowed to increase the results also on test data (what is already showed by table 7 but on train data).

	tf bow	*binary bow*	*tf-idf bow*
LR	**0.478**	0.474	0.397
	0.573	0.578	**0.581**
SVM	0.457	0.456	0.453
	0.50	0.519	0.534
NBM	0.435	0.432	0.2808
	0.531	0.527	0.436

Table 10: Scores for task A on test data, using *bow* features and default parameterized models. F_1^{PN} is reported on the first line and accuracy on the second

	tf bow	*binary bow*	*tf-idf bow*
LR	**0.59**	0.582	0.532
	0.764	0.76	0.794
SVM	0.569	0.565	0.567
	0.679	0.68	0.789
NBM	0.588	0.584	0.5
	0.79	**0.792**	0.786

Table 11: Scores for task B on test data, using *bow* features and default parameterized models. ρ^{PN} is reported on the first line and accuracy on the second

The results between tables 4, 5, 6 and 7 as well as between tables 10, 11, 12 and 8 (and especially 9!) show to what extend our model increases the performance on the training and the test set compared to a simple bag-of-words approach. Although some of the results are not convincing, others show a big step and the overall behavior is an increase of the importance of negative class, which was the objective. We increased the objective scores between 4% and 8% for each subtask thanks to our model.

	tf bow	*binary bow*	*tf-idf bow*
LR	1.105	1.108	1.25
	0.475	0.475	0.455
SVM	1.043	**1.035**	1.169
	0.449	0.45	**0.482**
NBM	1.19	1.178	1.381
	0.475	0.473	0.393

Table 12: Scores for task C on test data, using *bow* features and default parameterized models. *MAE* is reported on the first line and accuracy on the second

We also remark that though *tf-idf* produced relatively low score when used as the only features for a classifier, when combined with our set, it yields much better results, as it is used in the majority of our best models. It remains very poor with naive bayes, because this model uses term frequencies to compute probabilities of words and therefore does not connect well with this weighting metric.

7 CONCLUSION

In order to achieve the best score possible for subtasks A, B and C we had to increase the importance of the negative class (respectively -2 for task C). Highly dominated by positive and/or neutral labels, it suffered a lot when using a simple machine learning approach. Finding good discriminative features with highly imbalanced classes is tough, and a large part of our efforts has been put for it.

We followed strict development processes to avoid any bias and knowledge leak with test data. Repetitive cross validation to have good et stable estimate of different metrics have been used. We did not give a lot of importance to accuracy as a score due to the obvious imbalance between labels. Increasing accuracy in such conditions is much easier and does not reflect how well behave the system in a multiclass context.

We strictly used only the training sets that were given for each task. The result are better using more training, but part of the challenge was to stick to the SemEval data sets.

We developed a set of hybrid and *handmade* features, using lexicons and Twitter specific insights, that have been shown to increase the score we obtained both on train data by cross validation and on test data.

Results also show that the train and test sets are significantly different and thus it is hard to predict a behavior in our models. It is probable that using validation sets and test development sets may have increased generalization of our system and its stability. Though, we didn't consider having enough data to do so.

Furthermore, using advanced machine learning techniques such as stacking ensemble models or deep learning approaches, might improve our results. We also think that the key lies in the way of computing features thanks to the lexicons. Their wide variety maybe lead us to over-evaluate the problem and miss out on the best way to use them. We want to furthermore investigate more on distributed word representation and sentiment disambiguation to a finer level of analysis with entity and aspect-level work.

REFERENCES

[1] Satanjeev Banerjee and Ted Pedersen. 2002. An Adapted Lesk Algorithm for Word Sense Disambiguation Using WordNet. In *Proceedings of the Third International Conference on Computational Linguistics and Intelligent Text Processing (CICLing '02)*. Springer-Verlag, London, UK, UK, 136–145. http://dl.acm.org/citation.cfm?id=647344.724142

[2] Margaret M. Bradley, Peter J. Lang, Margaret M. Bradley, and Peter J. Lang. 1999. Affective Norms for English Words (ANEW): Instruction manual and affective ratings. (1999).

[3] Alexander Budanitsky and Graeme Hirst. 2001. Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures. In *Workshop on WordNet and other lexical resources*, Vol. 2. 2–2.

[4] Jay J Jiang and David W Conrath. 1997. Semantic similarity based on corpus statistics and lexical taxonomy. *arXiv preprint cmp-lg/9709008* (1997).

[5] Jaap Kamps, Maarten Marx, Robert J Mokken, Maarten De Rijke, et al. 2004. Using WordNet to Measure Semantic Orientations of Adjectives.. In *LREC*, Vol. 4. Citeseer, 1115–1118.

[6] Byungkon Kang and Kyung-Ah Sohn. [n. d.]. Embedding Senses via Dictionary Bootstrapping. ([n. d.]).

[7] Quoc Le and Tomas Mikolov. 2014. Distributed Representations of Sentences and Documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32 (ICML'14)*. JMLR.org, II–1188–II–1196. http://dl.acm.org/citation.cfm?id=3044805.3045025

[8] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1 (HLT '11)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 142–150. http://dl.acm.org/citation.cfm?id=2002472.2002491

[9] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 3111–3119. http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

[10] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[11] Saif M. Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu. 2013. NRC-Canada: Building the State-of-the-Art in Sentiment Analysis of Tweets. In *Proceedings of the seventh international workshop on Semantic Evaluation Exercises (SemEval-2013)*. Atlanta, Georgia, USA.

[12] Saif M. Mohammad and Peter D. Turney. 2013. Crowdsourcing a Word-Emotion Association Lexicon. 29, 3 (2013), 436–465.

[13] Preslav Nakov, Alan Ritter, Sara Rosenthal, Fabrizio Sebastiani, and Veselin Stoyanov. 2016. SemEval-2016 Task 4: Sentiment Analysis in Twitter. In *Proceedings of SemEval-2016 (ACL '16)*. 18. https://aclweb.org/anthology/S/S16/S16-1001.pdf

[14] F. Å. Nielsen. 2011. AFINN : A new ANEW: Evaluation of a word list for sentiment analysis in microblogs. (mar 2011). http://www2.imm.dtu.dk/pubdb/p.php?6010

[15] 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. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[16] Philip Resnik et al. 1999. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *J. Artif. Intell. Res.(JAIR)* 11 (1999), 95–130.

[17] Theresa Wilson, Janyce Wiebe, and PaulHoffmann. 2005. Recognizing Contextual Polarity in Phrase-Level Sentiment Analysis.. In *Proc. of HLT-EMNLP*.

[18] Liang-Chih Yu, Jin Wang, K. Robert Lai, and Xue-Jie Zhang. 2017. Refining Word Embeddings for Sentiment Analysis. In *EMNLP*.