

Structural Feature Extraction and Engineering for Sentiment Analysis

Eva Perontsi

Thesis submitted in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science and Engineering

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisor: Associate Prof. *Polyvios Pratikakis*

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

The work has been supported by the Foundation for Research and Technology - Hellas (FORTH), Institute of Computer Science (ICS).

UNIVERSITY OF CRETE
COMPUTER SCIENCE DEPARTMENT

**Structural Feature Extraction and Engineering for Sentiment
Analysis**

Thesis submitted by
Eva Perontsi
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author:




Eva Perontsi

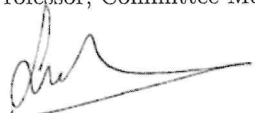
Committee approvals:



Polyvios Pratikakis
Associate Professor, Thesis Supervisor, Committee Member



Ioannis Tsamardinos
Professor, Committee Member



Sotiris Ioannidis
Associate Professor, Committee Member

Departmental approval:



Polyvios Pratikakis
Associate Professor, Director of Graduate Studies

Heraklion, October 2022

Structural Feature Extraction and Engineering for Sentiment Analysis

Abstract

The subject of this work is the sentiment analysis of Greek-speaking tweets. We use natural language processing (NLP) methods and neural networks to create three different classification models. The first model processes single, independent tweets and decides if their sentiment is positive or not, or if it is negative or not. The second model considers a tweet paired with its textual context, meaning the tweet that it responds to. With the third neural model we attempt to do sentiment analysis with the tweet, textual context and some additional, structural features, as input. These structural features are extracted from the Twitter graph, and give us information about the authors of the tweet and textual context. Our experiments show that the additional text context improves our prediction by a small percentage in some cases. However, we find no correlation between the predicted tweet sentiment and the Twitter graph structural features.

Εξαγωγή και Αξιοποίηση Δομικών Χαρακτηριστικών για Ανάλυση Συναισθήματος

Περίληψη

Το θέμα αυτής της εργασίας είναι η ανάλυση συναισθήματος (sentiment analysis) ελληνόφωνων tweets. Χρησιμοποιούμε τεχνικές ανάλυσης φυσικής γλώσσας (natural language processing, NLP) και νευρωνικών δικτύων για να δημιουργήσουμε τρία διαφορετικά μοντέλα κατηγοριοποίησης. Το πρώτο μοντέλο επεξεργάζεται μεμονωμένα tweets και καθορίζει αν έχουν ή όχι θετικό sentiment ή αντίστοιχα αρνητικό sentiment. Το δεύτερο μοντέλο λαμβάνει υπ' όψιν ένα tweet μαζί με τα συμφραζόμενά του, δηλαδή το κείμενο στο οποίο απαντάει. Στο τρίτο μοντέλο δοκιμάζουμε να πραγματοποιήσουμε sentiment analysis έχοντας ως δεδομένα το tweet, τα συμφραζόμενα, και ένα σύνολο από δομικά χαρακτηριστικά των συγγραφέων αυτών των tweets. Τα δομικά αυτά χαρακτηριστικά τα συγκεντρώνουμε μελετώντας το γράφο του Twitter. Τα πειράματά μας δείχνουν ότι τα επιπλέον συμφραζόμενα σε μορφή κειμένου βελτιώνουν τις προβλέψεις μας κατά ένα μικρό ποσοστό σε κάποιες περιπτώσεις. Ωστόσο, δε βρίσκουμε κάποια συσχέτιση ανάμεσα στην πρόβλεψη ενός tweet sentiment και τα δομικά στοιχεία του γράφου του Twitter.

Ευχαριστίες

Στην οικογένειά μου

Contents

1	Introduction	1
2	Related work	3
3	Dataset	6
1	Data collection and features	6
2	Labeling	9
3	Preprocessing	11
4	Design	14
1	Base model	15
2	Conversations model	17
3	Conversations and structural features model	18
5	Experiments and Results	20
1	Base model	22
2	Conversations model	22
3	Conversations and structural features model	23
6	Conclusion and Discussion	24
1	Conclusion	24
2	Limitations	25
3	Discussion	26
4	Future work	26
	Bibliography	28
1	APPENDIX	30

Chapter 1

Introduction

Online posts are a widespread way of communication. Classifying them as positive or negative gives us a great outlook about the views of the author on a subject. Many companies and applications use sentiment analysis to take a thorough look at their users' preferences and satisfaction. In this work, we determine if the additional text context in the form of previous parts of the online conversation, and structural context in the form of graph features of a twitter post affect the accuracy of its sentiment prediction.

Twitter is a widely used online social network (OSN) where millions of users interact every day by writing small text posts called "tweets". We choose Twitter for this work, because its popularity (about 500 million tweets per day) combined with its microblogging nature, leads to very large, continuously growing sets of small, easy to handle, text data. Also, tweets tend to be more opinionated as they comment on political and social issues, such as elections[7] and trending events. We use a custom Twitter crawler, twAwler [19], and gather ≈ 6.7 GB of Greek-speaking tweets in the period between January 1st 2021 and February 1st 2021.

In order to discover the sentiment of the Greek speaking part of Twitter, we firstly build our base case. The base case of our study is the sentiment analysis on a single tweet using a long short-term memory (LSTM) [10] neural network. Given that a tweet might be the response to another tweet as a reply, we extend our model to consider the latter as textual context for our analysis. To further improve the sentiment predictions, we modify our model anew to also utilize the graph properties between the involved users as structural context.

Our first attempt to improve our sentiment predictions for our analysis, is to get additional context for the tweet we want to evaluate. We call **base tweet** a crawled Twitter post that someone has replied to, and **target tweet** the reply to the base tweet. We predict the sentiment of the target tweet, given our knowledge of its textual context (the base tweet) and its content, and find a small increase in accuracy.

In our second attempt to better improve our predictive model, we make the hypothesis that information about the structure of the Twitter graph will help us

better our predictions. Thus, we further augment our analysis by adding structural features to complement our tweet conversations. These features are derived from the Twitter graph, and illustrate the role of the base and target tweets' authors. Nodes in the Twitter graph represent users and posts, and edges represent different interactions between them. There are three types of interactions: 1) between posts, such as "replies-to-tweet", that connects a reply node with the tweet it replies to, 2) between users, such as "follow", which connects a user with another user they follow, and 3) between a post and a user, such as "favorite", which connects a user with a tweet they favorite.

We use the "replies-to-tweet" relationship to find pairs of base tweets and replies, called conversations. We also create a graph embedding, a vector that summarizes the structural features of the conversing users. The embedding consists of but is not limited to the mention, quote, favourite and reply in and out degrees, the followers count, the friend/follow ratio etc. These features give us a new perspective about a user's role in the twitter graph; and increase the accuracy of our prediction.

We build our models with the keras[6] API in Python. The first model processes individual tweets, the second model processes conversations and the third and last model processes conversations and graph features. We train our three models with different sets of parameters, and keep our best performing model version for each category. The best model that processes tweets has an accuracy of 72.2%. The best model that takes tweet conversations as input has similar performance as its predecessor, with 72.78% accuracy. All the models that take tweet conversations and graph features as input, unfortunately have base line accuracy, with the best performing model at 50.85% accuracy.

The first model's performance is comparable with other sentiment analysis models in the bibliography, like Wang *et al.* [21] . We interpret the second model's slightly better accuracy as the result of our labeling process; we do not take into consideration the base tweet when we label the target tweets in our dataset. Likewise, the graph features do not contribute to our initial labeling of the dataset, and the third model fails to discover any correlation between these features and the tweet conversations.

Chapter 2

Related work

Sentiment analysis on social media posts is a very popular topic right now, as different platforms take advantage of the users' preferences to tune their recommendation algorithms and target their advertisements. There are many different tools for sentiment analysis using NLP, from libraries like python's NLTK[2] and Stanford CoreNLP[15], to pre-trained models like BERT[9]. In our work, we focus on Greek language texts, which are not supported by many of the existing tools, and we use neural networks for our analysis.

There are many works on neural networks and sentiment analysis on online social media posts, some of which are presented below.

Huang *et al.* [11] implement a recurrent neural network that uses LSTM layers in order to do a sentiment classification of short texts. They gather data from Weibo, a Chinese microblogging platform similar to Twitter. Their work is the main influence for our decision to create a base model and then "build" upon it to add the additional context. More specifically, at first they create an LSTM model that takes single posts as input and then they extend it in order to process process conversations by adding a second LSTM layer (Hierarchical LSTM model).

Wang *et al.* [21] use an LSTM model in combination with word embeddings to perform sentiment analysis on short text from various online sources. They train their model with two different datasets, an English movie reviews dataset extracted from IMDB and a Chinese movie reviews dataset from Douban. They test their model with a dataset of Chinese forum posts extracted from the PTT bulletin board. They use the Word2Vec Skip-gram model[17] to create the word embeddings for their input and then feed these representations in an LSTM model. In their experiments they achieve over 80% accuracy with the LSTM model for the IMBD reviews dataset, around 75% accuracy for the Douban reviews dataset and around 70% accuracy for the PTT posts dataset.

Yin *et al.* [22] compare the performance of convolutional neural networks (CNNs) with the performance of recurrent neural networks (RNNs) on six different

NLP tasks; sentiment classification, relation classification, textual entailment, answer selection, question relation match, path query answering and part-of-speech tagging. Their experiments for the sentiment analysis task show that RNNs perform better than CNNs.

Ratkiewicz *et al.* [20] search for astroturfing, or orchestrated campaigns appearing to be grass-roots movements in order to influence opinion. They use hashtags, mentions, URLs and the entire text of every tweet as “memes” and look into propagation patterns in diffusion networks. They use a set of features per meme to classify memes, and assign six GPOMS sentiment dimensions [4]: calm, alert, sure, vital, kind, happy.

Conover *et al.* [8] found that the retweet graph is better correlated to political affiliation clustering than the mention graph. Mentions can be interactions between disagreeing parties but retweets are almost always endorsements. Retweet cliques seem to be politically homophilic.

Narr *et al.* [18] develop a sentiment analysis that is agnostic to language and can be trained to recognize sentiment-heavy n-grams starting from tweets containing emoticons, before it is applied to tweets in each language. The authors use both twitter API’s language recognition and Chrome’s language recognition system to partition tweets of many languages per language, and train their analysis on each language. The method’s precision depends on the language and local culture, writing style, etc., and was able to perform with up to 80% accuracy for large training sets in some languages.

Even though Greek is a language spoken for a tiny percentage of the world population, there are some teams that do sentiment analysis on Greek text extracted from social networks.

Antonakaki *et al.* [1] present an analysis of the Greek 2015 referendum and parliamentary elections. They use a stemmer for word matching, and a lexicon-based sentiment analysis to assign sentiment to LDA topics. They produced an accurate prediction of the ballot results by counting number of tweets and tweet sentiment per topic, and assigning topics to outcomes. TwAwler uses the same sentiment dictionary, kindly provided by the authors, to extract sentiment features per user and per entity.

Markopoulos *et al.* [16] gather hotel reviews written in Greek from TripAdvisor and perform sentiment analysis using a Support Vector Machine (SVM). In order to represent the greek text as data that are readable by the SVM, they use two different traditional NLP approaches; the TF-IDF Bag-of-Words approach, and the Term Occurrence approach. The model that follows the TF-IDF Bag-of-Words approach achieves 95.78% accuracy, whereas the model that uses Term Occurrence in the data processing achieves 71.76% accuracy in predicting positive and negative reviews.

Kydros *et al.* [13] analyze the Greek part of the Twitter graph during the first wave of the COVID-19 pandemic. They focus on COVID-19 related keywords and gather four datasets tweets that contain these words, each one for the duration of a month, from March 2020 to June 2020. They visualize the connections between

these keywords and also use a Greek sentiment lexicon to find the sentiment during these time periods. They find that negative moods such as fear and anger are more prominent than positive moods during these months.

Kalamatianos *et al.* [12] use a Greek sentiment lexicon to evaluate Greek tweets on six different sentiment values; anger, disgust, fear, happiness, sadness and surprise. They also get the most used hashtags in a specific time period and use the sentiment lexicon to produce their sentiment in the same six different values.

Chapter 3

Dataset

1 Data collection and features

We gather ~ 7 million Greek conversations from Twitter, in the time period of January 2021, using twAwler [19]. From the collected conversations, we create two balanced datasets of 100.000 conversations each. An example datapoint is shown in 3.1.

The "base_tweet" is the base tweet as described in chapter 1, it creates the context of the conversation. The "reply" is the "target_tweet" as described in chapter 1. It is the tweet we want to do sentiment analysis on. The "author_id" and "replier_id" are the Twitter ID's of the author of the base tweet and target tweet respectively. We use these ID's to match the participants of each conversation to their structural features. The "author_features" and "replier_features" fields represent the structural features of the authors of the base and target tweet. These vectors contain numerous features extracted by the Twitter graph. The complete list of the structural features we use is in 3.2 and 3.3. These structural features are categorized in seven groups that are separated with horizontal lines in the tables. The first group contains general metrics about the user, like the number of tweets they have posted, how many friends and followers they have and how many tweets they have deleted. The second group contains all the information related to the "mention" relationship regarding this user. For example, there is the number of times that the user has been mentioned by other users, the number of times he mentions other users etc. Likewise, the third group contains the information about the "quote" relationship, the fourth group consists of the "retweet" features and the fifth group of the "reply" features. The sixth group has more information about the user's friends and followers, like the number of the user's followers that are Greek, the number of friends that are tracked by the twAwler API, and more. Finally, in the last group there are some statistics about the tweets the user posts, such as the average number of words in their tweet and the number of "favorite" tweets.

```

"base_tweet": "Καλημέρα και καλό μήνα!",
"reply": "Δεν μας παρατάς κι εσύ.",
"author_id": 111111111,
"replier_id": 222222222,
"author_features": [tweet_count, favourites_count, followers_count,
friends_count, friends_count, fr_fo_ratio, seen_total, total_inferred, seen_greek_total,
deleted_tweets, seen_top_tweets, top_tweets_pcmt, mention_indegree, men-
tion_outdegree, mention_inweight, mention_outweight, mention_avg_inweight,
mention_avg_outweight, mention_out_in_ratio, mention_pcmt, quote_indegree,
quote_outdegree, quote_inweight, quote_outweight, quote_avg_inweight,
quote_avg_outweight, quote_out_in_ratio, quote_pcmt, retweet_indegree,
retweet_outdegree, retweet_inweight, retweet_outweight, retweet_avg_inweight,
retweet_avg_outweight, retweet_out_in_ratio, retweet_pcmt, reply_indegree,
reply_outdegree, reply_inweight, reply_outweight, reply_avg_inweight, re-
ply_avg_outweight, reply_out_in_ratio, reply_pcmt, seen_replied_to, seen_fr,
gr_fr, gr_fr_pcmt, tr_fr, tr_fr_pcmt, seen_fo, gr_fo, gr_fo_pcmt, tr_fo, tr_fo_pcmt,
tr_fo_jaccard, fr_and_fo, fr_or_fo, gr_fr_fo, gr_fr_fo_pcmt, total_words, min_wptw,
avg_wptw, med_wptw, std_wptw, favoriters, favorited],
"replier_features": [tweet_count, favourites_count, followers_count,
friends_count, friends_count, fr_fo_ratio, seen_total, total_inferred, seen_greek_total,
deleted_tweets, seen_top_tweets, top_tweets_pcmt, mention_indegree, men-
tion_outdegree, mention_inweight, mention_outweight, mention_avg_inweight,
mention_avg_outweight, mention_out_in_ratio, mention_pcmt, quote_indegree,
quote_outdegree, quote_inweight, quote_outweight, quote_avg_inweight,
quote_avg_outweight, quote_out_in_ratio, quote_pcmt, retweet_indegree,
retweet_outdegree, retweet_inweight, retweet_outweight, retweet_avg_inweight,
retweet_avg_outweight, retweet_out_in_ratio, retweet_pcmt, reply_indegree,
reply_outdegree, reply_inweight, reply_outweight, reply_avg_inweight, re-
ply_avg_outweight, reply_out_in_ratio, reply_pcmt, seen_replied_to, seen_fr,
gr_fr, gr_fr_pcmt, tr_fr, tr_fr_pcmt, seen_fo, gr_fo, gr_fo_pcmt, tr_fo, tr_fo_pcmt,
tr_fo_jaccard, fr_and_fo, fr_or_fo, gr_fr_fo, gr_fr_fo_pcmt, total_words, min_wptw,
avg_wptw, med_wptw, std_wptw, favoriters, favorited],
"pos": 0.5,
"neg": 2

```

Figure 3.1: Example of a datapoint

tweet_count	Total tweet count, as reported by twitter.
favourites_count	Number of tweets this user has favorited, as reported by twitter.
followers_count	Number of users that follow this user, as reported by twitter.
friends_count	Number of users this user follows, as reported by twitter.
fr_fo_ratio	Ratio of friends to followers.
seen_total	Total number of this user's tweets used in computing this vector.
total_inferred	Total number of this user's tweets including encountered tweet IDs that could not be crawled and are probably deleted.
seen_greek_total	Total number of this user's tweets marked as being in Greek by the twitter API.
deleted_tweets	Total number of this user's tweets that were seen and then were deleted.
seen_top_tweets	Seen tweets that were not retweets or replies.
top_tweets_pcmt	Percentage of seen tweets that were not retweets or replies.
mention_indegree	Number of users that mention this user.
mention_outdegree	Number of users mentioned by this user.
mention_inweight	Number of tweets that mention this user.
mention_outweight	Number of mentions by this user.
mention_avg_inweight	Average mentions of this user per mentioner.
mention_avg_outweight	Average mentions per mentioned user.
mention_out_in_ratio	Out-degree/in-degree ratio (mention reciprocity).
mention_pcmt	Percentage of seen tweets that are mentions.
quote_indegree	Number of users that quote this user.
quote_outdegree	Number of users quoted by this user.
quote_inweight	Number of tweets that quote this user.
quote_outweight	Number of quotes by this user.
quote_avg_inweight	Average mentions of this user per mentioner.
quote_avg_outweight	Average mentions per quoted user.
quote_out_in_ratio	Out-degree/in-degree ratio (quote reciprocity).
quote_pcmt	Percentage of seen tweets that are mentions.
retweet_indegree	Number of users that have retweeted this user.
retweet_outdegree	Number of users this user retweeted.
retweet_inweight	Number of retweets of this user's tweets.
retweet_outweight	Number of retweets by this user.
retweet_avg_inweight	Average retweets per retweeter.
retweet_avg_outweight	Average retweets per retweeted user.
retweet_out_in_ratio	Out-degree/in-degree ratio (retweet reciprocity).
retweet_pcmt	Percentage of retweets in tweets seen by this user.

Figure 3.2: Table of structural features (1/2)

reply_indegree	Number of users that have replied to this user at least once.
reply_outdegree	Number of users to which this user replied at least once.
reply_inweight	Number of replies to this user.
reply_outweight	Number of replies tweeted by this user.
reply_avg_inweight	In-Replies per in-degree.
reply_avg_outweight	Out-Replies per out-degree.
reply_out_in_ratio	Out-degree/in-degree ratio (replies sent for each received).
replies_pcmt	Percentage of replies in tweets seen by this user.
seen_replied_to	Number of tweets that received replies from other users.
seen_fr	Total number of twitter users seen to be followed by this user at least once.
gr_fr	Number of friends marked Greek-speaking.
gr_fr_pcmt	Number of friends marked Greek-speaking.
tr_fr	Number of friends tracked.
tr_fr_pcmt	Percentage of friends tracked.
seen_fo	Total number of twitter users seen to follow this user at least once.
gr_fo	Number of followers marked Greek-speaking.
gr_fo_pcmt	Percentage of followers marked Greek-speaking.
tr_fo	Number of followers currently tracked.
tr_fo_pcmt	Percentage of followers currently tracked.
fr_fo_jaccard	Jaccard similarity between friend and follower sets.
fr_and_fo	Size of the intersection of the friend and follower sets, <i>i.e.</i> , all reciprocal follow edges.
fr_or_fo	Size of the union of the friend and follower sets, <i>i.e.</i> , all neighbors in the follow graph.
gr_fr_fo	Number of all friends and followers marked as Greek-speaking.
gr_fr_fo_pcmt	Percent of Greek-speaking neighbors (friends and followers).
total_words	Total number of words in all seen tweets.
min_wptw	Minimum number of words per seen tweet.
avg_wptw	Average number of words per seen tweet.
med_wptw	Median number of words per seen tweet.
std_wptw	Standard deviation of the number of words per tweet.
favoriters	Number of users seen to have liked a tweet by this user.
favorited	Number of users whose any tweet this user has liked.

Figure 3.3: Table of structural features (2/2)

2 Labeling

In order to use our crawled tweets for our analysis, we need to create a training dataset, that consists of labeled datapoints. We use a sentiment analysis dictionary [1] for the Greek language to label our collected tweets. A sentiment analysis dictionary is a dictionary that maps a word or phrase to a sentiment. In our case, the dictionary maps each Greek word to a tuple of numbers that represent the positive and negative sentiment of this word. For the positive sentiment, the number is between 0 and 4, where 4 corresponds to "very positive" and 0 to "not positive at all". Likewise, for the negative sentiment, -4 implies a "very negative" and 0 a "not negative at all" sentiment.

In the gathered conversations we can see a clear tendency towards negativity.

In figure 3.4 we show the summation of the absolute values for each class (positive and negative).

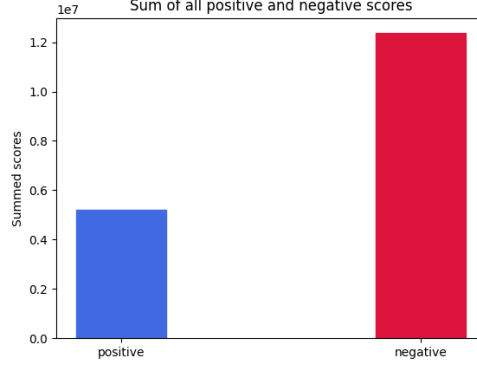


Figure 3.4: Sum of positive and negative sentiment of Greek tweets

From this point on, we will refer to sentiment with an absolute value greater or equal than 2 as **strong** (positive or negative) and to sentiment with an absolute value less than 2 as **weak**.

In figures 3.5 and 3.6 we show that it is more common for negative sentiment to be strong, than it is for positive sentiment. Figure 3.5 shows the number of conversations with a specific positive and negative score in absolute value. The chart in 3.6 shows the number of conversations that have a strong positive and/or a strong negative sentiment score. It is obvious that in the Greek speaking community on Twitter for the specific period, negativity is much more strongly expressed than positivity.

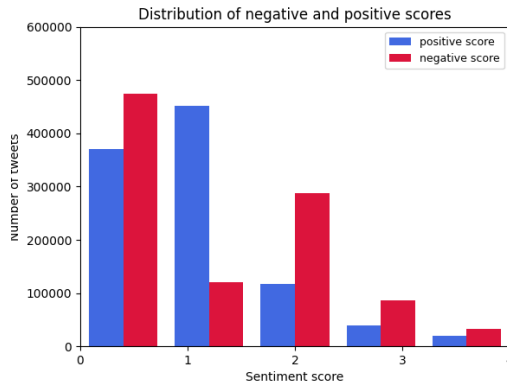


Figure 3.5: Positive and negative sentiment of Greek tweets

In the scope of this work, we treat this sentiment analysis problem as a binary classification problem. Therefore, we need to convert the $[-4, 0]$ and $[0, 4]$ labels to

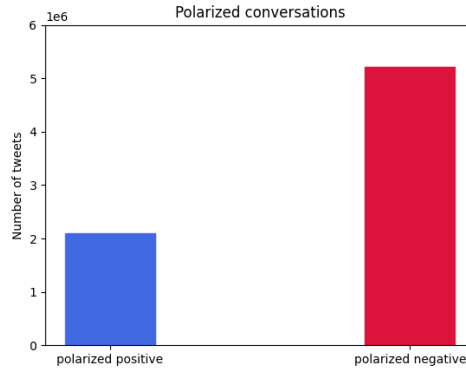


Figure 3.6: Polarized positive and negative Greek tweets

binary ones for each score. We use a binary 0,1 label for each sentiment (positive and negative), based on whether it is weak or strong. For example, a conversation with labels $(-3, 1)$ now has a negative label 1 and a positive label 0.

We make the decision to keep two separate labels for positive and negative, instead of using a sole 0/1 label for positive/negative sentiment, because many conversations have mixed sentiment.

As shown above, the original dataset is heavily skewed in favor of negative conversations. To eliminate this type of negative bias in our model, we randomly select 50,000 conversations of each one of our labeled classes; weak positive, strong positive, weak negative and strong negative. With these four sets we create two datasets; a negative dataset, that contains the weak and strong negative conversations, and a positive one, that contains the weak and strong positive conversations.

3 Preprocessing

In order to pass our data through the training process, we first need to clean up the text data from noise. We split the tweet and remove URLs, mentions, HTML tags, punctuation, emojis, single letter words and white characters. In order to feed our data to our neural models, we have to decide on a fixed length for all our tweets. This is essential because the tensorflow neural networks we work with use tensors. Tensors are multi-dimensional matrices that need to be rectangular, that means along each axis every element has the same size.

Therefore, we need to decide on a fixed length for all our tweets. We decide that all tweets must have length of 40 words. We choose this number because only a small percentage of tweets has more than 40 words. More specifically, in figure 3.7 there is the ECDF graph of the lengths of 200,000 randomly selected tweets and replies. More specifically, over 80% of tweets contain only 20 words or less. So, we truncate longer tweets and keep the first 40 words, and we pad shorter tweets with empty words until they reach length 40.

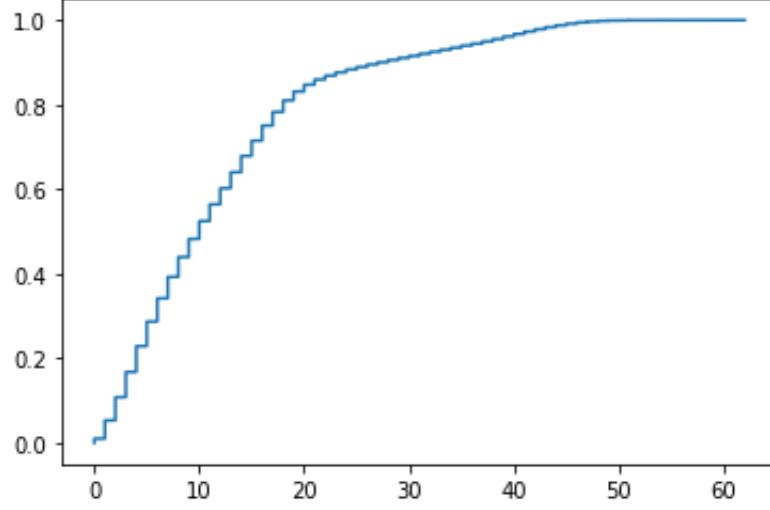


Figure 3.7: Tweet length ECDF

Neural models cannot read text as input, they understand numerical values. A common method to transform text data to numerical values are word embeddings. These are vectors of numbers where each vector represents one word. There are different pre-trained models that produce these types of representations, such as Word2Vec [17] (used in Wang *et al.* [21]) and Fasttext [3]. In our work we use the Fasttext pre-trained model, that creates the vector representations for our text data. We replace each word in a tweet that exists in the Fasttext model with the equivalent vector and each empty padding word with a zero vector.

In figure 3.8 there are the three stages of text preprocessing in detail. In the first stage, we have the original tweet, “@user123 θυμάσαι τι ωραία που είχαμε περάσει σε αυτή τη συναυλία: <https://www.youtube.com/watch?v=184LefO27ys>”. In the second stage we want to keep only the useful words for our analysis. To do that, we remove the user mention, the punctuation and the video URL. Additionally, we split the tweet into word tokens, and truncate (or pad it at the front with empty words) in order for it to be 40 words long. When padding is necessary, we add it to the start of the tweet, rather than the end, because our models propagate the information of each tweet to its last word. Thus, it is better to keep the information of the datapoint near the end of the sequence. At the final stage, we replace each word with its Fasttext embedding. The empty padding words are replaced with a zero vector.

The structural data are represented as vectors of length 66, where each feature is a numerical value extracted from the Twitter graph. These features are shown in detail in tables 3.2 and 3.3. These features are already numerical by nature, so we do not process them any further.

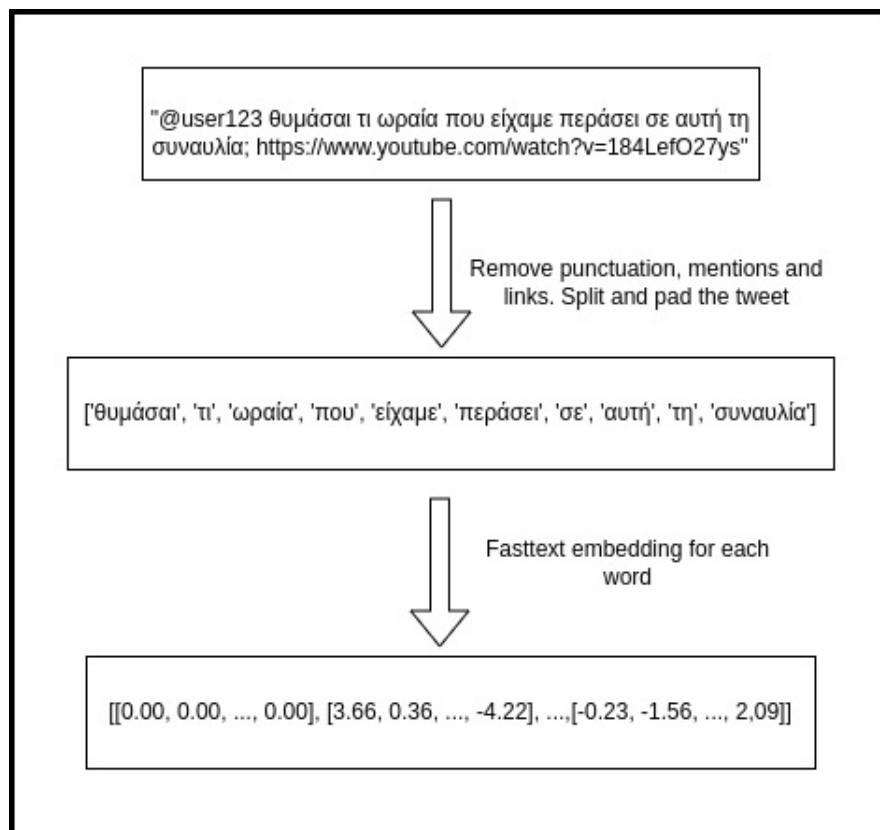


Figure 3.8: Data preprocessing

Chapter 4

Design

Neural networks are used for a variety of tasks, such as image classification, speech recognition and other predictive models. Recurrent neural networks are a subclass of neural networks that utilize internal memory to process sequences of data points. Examples of these types of sequences are speech (sequence of sounds), text (sequence of words), and video (sequence of images). We decide to use RNNs instead of other types of neural network because they perform well in tasks that use sequences, like our dataset that consists of sequences of words (tweets and conversations).

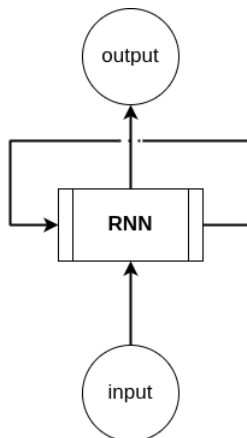


Figure 4.1: Example of a recurrent neural network

Of various RNN architectures we choose long short-term memory or LSTM networks. LSTMs are a type of recurrent model that tackle the problem of the "vanishing" gradient. The vanishing gradient problem comes up when gradients tend to zero after being multiplied multiple times with very small numbers during back-propagation. LSTMs tackle this problem with the use of three components called "gates". The input, output and forget gate of the LSTM decide what information is to be remembered by the network, what information needs to go

through as the output and what information has to be forgotten. This way, if a gradient is zero will be forgotten by the forget gate and will not be propagated furthermore. Figure 4.2 shows an LSTM cell. The three aforementioned gates are the sigmoid functions at i_t (input gate), o_t (output gate) and f_t (forget gate).

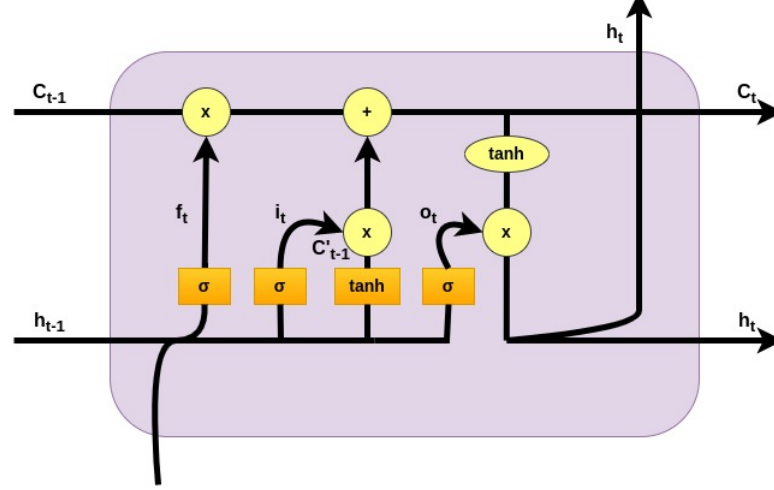


Figure 4.2: Structure of the LSTM cell

In the following sections we present our models. We start with the "base model" which is the base for our next two. This first model takes as input a single tweet. Next, we modify the base model and create the "conversations model", which analyzes tweet conversations (text plus context). Finally, we extend the conversations model and create the "conversations and structural features model", which analyzes tweet conversations paired with structural features.

1 Base model

In order to test our hypothesis, that text and structural context improves the accuracy of the sentiment analysis of tweets, we first have to create our base model as a point of reference. The architecture of the model is shown in figure 4.3.

The base model consists of an input layer that takes a tweet as an input, a lambda layer, an LSTM layer and a dense layer, that produces a number as the output. This output is the positive (or negative, depending on the training dataset) sentiment, that ranges from 0 to 1.

The input layer is responsible to feed the model with batches of input data. The base model takes tweets (sequences of words) as input. The LSTM layer is the recurrent neural layer that efficiently works with sequence data. This layer essentially does the processing of the input; it remembers the gist of each word in a tweet and passes it to the next word, up until the end of the sentence. Now, the very end of the sentence has all the information that represents the whole tweet.

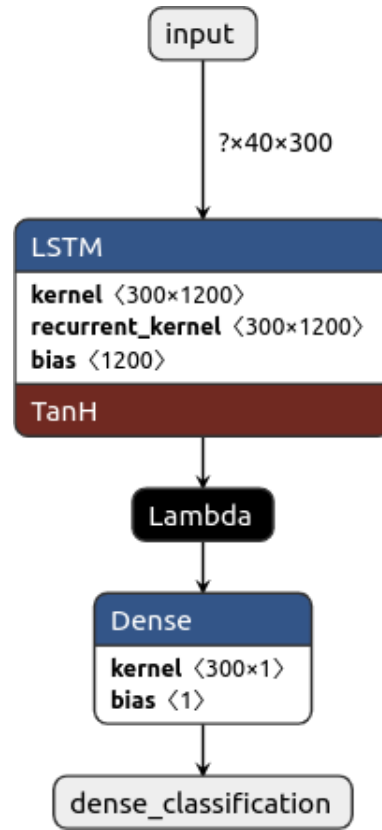


Figure 4.3: The base model

In order to keep this piece of information, we use the lambda layer.

The lambda layer is a layer that implements a custom lambda function. We use it to only keep the output of the last word of each sentence, instead of keeping one output per word. This way we only get the return sequences that represent the tweet, before we move to the next datapoint.

Finally, the fully connected (dense) layer uses the binary crossentropy loss function to perform the final step of regression. This layer is the classifier, and creates the model's output. For the experiments we use the negative dataset, which contains 50.000 weak negative and 50.000 strong negative tweets. The output prediction is a number between 0 and 1; which translates to how weak (closer to zero) or strong (closer to one) is the negative sentiment of the input.

Intuitively, we expect that information about the tweets of the same conversation will give us a clearer view of the tweet of interest. We test this intuition with the next model, that gets a tweet A as input paired with the tweet B that it replies to. We still want to predict the sentiment of tweet A, which is the reply in this scenario.

2 Conversations model

In order to test the hypothesis that additional text context improves the sentiment analysis we do on tweets, we utilize what we call conversational context. The conversational context, as presented above, is the tweet that our tweet of interest replies to. We call this extra tweet the "trigger" tweet, because it triggers the conversation. We integrate this conversational context in our analysis by feeding the trigger (or base) tweet to our model alongside the tweet of interest. In order to do that, we concatenate the trigger tweet and the reply tweet in a **conversation** of length 80 (two padded tweets of maximum length 40).

The model's architecture is very similar to the base model, as we can see in figure 4.4. We choose to only make minimal alterations to the base model to convert it to the conversations model. We think this way our comparison is fair, and the performance of the conversation model will improve only if the additional data help our objective, and not because we switched the model's architecture. The only adjustment here is the input's dimensions and the custom function in the lambda layer. Here we want to keep the returned sequences of each conversation (trigger tweet and reply tweet), so we modify our lambda function in order to return the last word of the whole conversation.

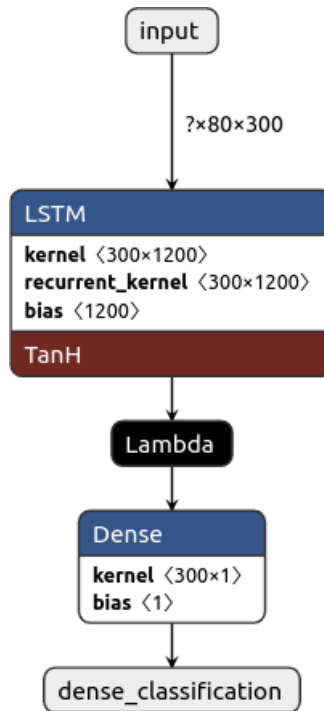


Figure 4.4: The conversations model

Now that we have a whole conversation as input, we go a step further and think about the users that participate in this conversation. Who are these users

and what can their role in the Twitter graph say about the intended sentiment of their posts? To answer these questions, we integrate some structural features that we extract from the Twitter graph to our data, and modify the conversations model in order to take the conversations and these features as input.

3 Conversations and structural features model

We want to test our last hypothesis; that if we add some structural features alongside the conversations, our model’s accuracy improves. The features that we use are extracted from the Twitter graph using twAwler [19]. These features give us information about the users that participate in each conversation of our dataset and are presented in detail in tables 3.2, 3.3.

These 66 features of our choice constitute a numerical vector of length 66 for each user. In contrast with the additional text context, in this case the structural features are not of the same nature with the tweet we want to analyze. Therefore, we cannot concatenate the structural features vectors and the conversations to create the input data. For this reason, we decide to feed the structural features to the model from a separate input layer.

We analyze the text data (conversations) in the same way as in section 2 up until the lambda layer. The text data pass through an LSTM and a lambda layer as before, and continue to a combination of dense layers and a flatten layer. The structural data go straight to a combination of dense layers and then a flatten layer. The purpose of these dense layers for both branches of the model is to reduce the dimensions of our data. We use these flatten layers to flatten the processed text and structural data to one dimensional vectors. For the final step of the classification, we concatenate these flattened intermediate results and pass them through a dense layer that predicts the output.

The conversations and structural features model is shown in figure 4.5.

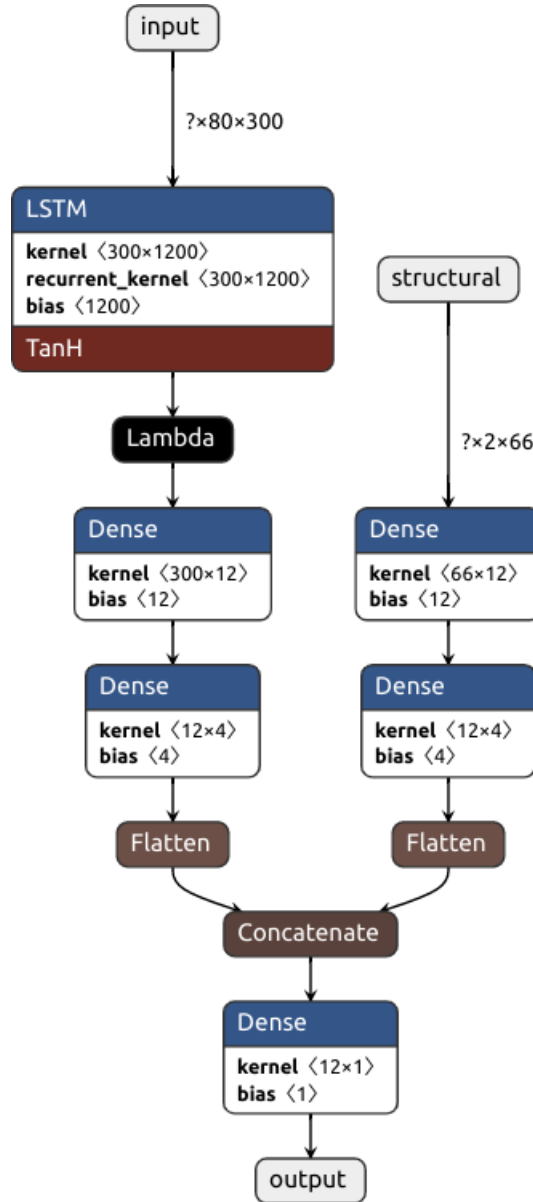


Figure 4.5: The conversations and structural features model

Chapter 5

Experiments and Results

After we gather and process our data and build our models, we construct the experiments that will either support or reject our hypothesis. The goal of these experiments is to see whether our sentiment analysis model performs better when we give additional context as input. More specifically, first we construct experiments for our base model, which is our point of reference for the other, extended, models.

In all experiments we use a balanced dataset of 100,000 tweets. This dataset is balanced in regards to the negative score, and our models predict only the negative score of the datapoints, with 0 annotating a weak negative score and 1 a strong negative score.. More specifically, the dataset contains 50,000 weak negative tweets and 50,000 strong negative tweets. As it can be seen in figure 3.1, a sample datapoint of our dataset already contains the target tweet (reply), the trigger (base) tweet and the structural features. Depending on which model we use, only some of these data are used. The base model only sees the target tweet, the conversations model sees both the target and trigger tweets and the conversations and structural features model sees all three of these types of data.

After we conclude the experiments for our base model, we run experiments with the same parameters and dataset for the conversations model and the conversations and structural features model.

In each of these experiments we split our data to 80% training and 20% validation sets. The validation set for each model uses data that are labeled by the sentiment dictionary, and are not seen by the model in the training process.

In all these experiments we use log loss (binary cross entropy) as a loss function and SGD as our optimizer. In figure 5.1 we can see the loss function. In this figure the true class is 1, and the more we approach the true class with our classifier, the loss tends to zero. Ideally the loss would be 0.0 if we could predict everything correctly, so we generally look for smaller numbers. On the other hand, we want our accuracy to be high, where accuracy equal to 1 means all predictions are correct.

The parameters we tune in our experiments are three; *the learning rate*, which

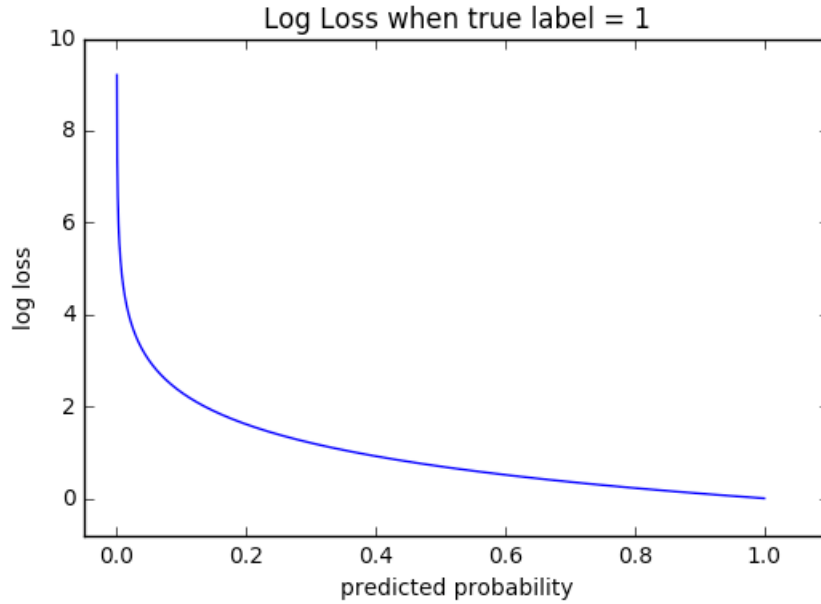


Figure 5.1: Log loss (binary cross entropy loss)

is the rate that our optimizer adjusts the weights of the model, the *batch size*, which defines the number of samples that are processed before the model weights are updated and the number of *epochs*, which defines how many times we will pass through the whole dataset.

- We try 0.0001, 0.0005 and 0.001 as learning rates. We settle for these learning rates because anything larger leads our loss to infinity, and anything smaller is very slow.
- For the batch sizes we try 30, 60, 120 and for some experiments, 1. Generally batch sizes 30, 60 and 120 work well, with the main difference between them being the duration of the training process. The experiments with batch size 1 have poor results and take several hours to finish.
- For the number of epochs we choose 6 and 10. In the majority of the experiments we train for 6 epochs because after the 6th epoch the accuracy stops increasing. In the experiments where we use 10 epochs, the maximum accuracy is still achieved on epoch 6 and at the best case scenario stays the same for the following 4 epochs. In some experiments it actually decreases after the 5th-7th epoch.

1 Base model

In table 5.1 we show the training time, training loss, validation accuracy and validation loss for each experiment of the base model. The parameters of the experiments that we tune are on the left side of the table; the learning rate, the batch size and the number of epochs.

The most accurate model has a validation accuracy of 72.3%, in the case of 0.001 as the learning rate, batches of size 60 and training for 10 epochs. However, this model has a bit higher training and validation losses. The second best experiment achieved 72.2% accuracy, was trained for 6 epochs with batch size equal to 120 and the learning rate set to 0.0005.

Learning rate	Batch size	Epochs	Training time	Training Accuracy	Training Loss	Validation Accuracy	Validation loss
0.0001	60	6	0:31:51	0.6571	0.654	0.6628	0.6513
0.0001	60	10	0:54:07	0.6868	0.6111	0.6878	0.6105
0.0001	1	6	13:40:42	0.501	7.7	0.496	7.7738
0.0001	30	6	0:48:07	0.501	7.6969	0.496	7.7746
0.0001	120	6	0:20:49	0.501	7.6969	0.496	7.7746
0.0005	60	6	0:34:15	0.6965	0.6003	0.7018	0.5997
0.0005	60	10	0:58:27	0.501	7.6969	0.496	7.7746
0.0005	30	6	0:54:42	0.711	0.5929	0.7216	0.5916
0.0005	120	6	0:26:51	0.7192	0.5843	0.722	0.5851
0.001	60	6	0:35:09	0.591	6.2858	0.6055	6.069
0.001	60	10	0:54:02	0.7091	0.6022	0.723	0.5784

Table 5.1: Experiments using the base model

2 Conversations model

The conversations model has very similar performance. The best performing model concerning both accuracy and loss achieves 72.78% accuracy, where the learning rate is set at 0.001, and the model trained with batches of size 60 for 10 epochs.

The results of the different experiments are in table 5.2.

Learning rate	Batch size	Epochs	Training time	Training Accuracy	Training Loss	Validation Accuracy	Validation loss
0.0001	60	6	1:22:25	0.6657	0.6524	0.6694	0.6505
0.0001	60	10	1:39:51	0.6719	0.6317	0.6747	0.6298
0.0001	1	6	25:10:32	0.6991	0.6232	0.7219	0.5894
0.0001	30	6	2:52:54	0.682	0.6164	0.6824	0.6154
0.0001	120	10	1:19:27	0.6851	0.6081	0.6849	0.6092
0.0005	60	6	1:27:11	0.698	0.5999	0.7093	0.5952
0.0005	60	10	2:26:38	0.7176	0.5832	0.7264	0.5778
0.0005	30	6	1:48:06	0.6767	0.9145	0.703	0.5957
0.0005	120	6	0:34:46	0.712	0.5882	0.7128	0.5973
0.0005	120	10	0:54:49	0.6871	1.6579	0.4997	7.7178
0.001	60	6	1:23:36	0.7115	0.5947	0.7133	0.5895
0.001	60	10	2:04:08	0.723	0.5832	0.7278	0.5989

Table 5.2: Experiments using the conversations model

We think these results are justified, because we do not take the additional text context into account when we label our dataset. So, if the target tweet is strongly negative and the trigger tweet is weakly negative, the label of this pair

will be strong negative. However, if the two tweets of the conversation agree sentiment-wise, the label will be correctly associated with this datapoint.

3 Conversations and structural features model

The experiments for the conversations and structural features model result in baseline accuracy and high losses, with the highest accuracy at 50.85%. Figure 5.3 presents the experiments in detail.

Learning rate	Batch size	Epochs	Training time	Training Accuracy	Training Loss	Validation Accuracy	Validation loss
0.0001	60	6	1:21:57	0.5002	7.7085	0.4996	7.7205
0.0001	60	10	1:56:28	0.5108	7.5394	0.5085	7.5754
0.0001	30	6	1:30:24	0.4997	7.6298	0.5012	7.6069
0.0001	120	6	0:34:29	0.4997	7.6296	0.5012	7.607
0.0005	60	6	1:14:56	0.5003	7.7079	0.4988	7.7311
0.0005	60	10	1:50:27	0.4995	7.6323	0.5013	7.6059
0.0005	30	6	1:29:12	0.4985	7.6477	0.5015	7.6022
0.0005	120	6	0:42:19	0.4988	7.7309	0.5048	7.6392
0.001	60	6	1:34:08	0.5005	7.7049	0.4998	7.7153
0.001	60	10	1:50:43	0.496	7.6881	0.4973	7.6681

Table 5.3: Experiments using the conversation and structural features model

We can see that in all our experiments we get very high losses and baseline accuracy. It seems that adding the structural features reduce our model's capabilities and make it equivalent to a coin toss.

In order to make sure that it is not our architecture that results into poor performance, we tested the same architecture with the dense and flatten layers with text data only. This resulted to comparable results to the experiments of the original base and conversations model. Therefore, we think that the structural features themselves are the problem. It seems that the conversations and structural features model perceives these features as noise. This can be explained by two main points. First, the limitation we have in the experiments of the conversations model stands in this model as well. The structural features are not taken into consideration in the labeling process. Second, the structural features themselves do not describe the relationship between the two conversing users, but they give us more general information about the participating users. This means that if user A participates in the conversation of the 1st datapoint and the 101st, the structural features of A will be the same in both of these datapoints.

Chapter 6

Conclusion and Discussion

In this chapter we discuss the conclusion of our results, the importance and limitations of sentiment analysis on text, and especially short texts like tweets, and we present some points for future work.

1 Conclusion

In this work, we aim to investigate whether additional contextual or structural information increases the accuracy of our sentiment classification model. We design three neural network models that predict the sentiment of Greek tweets. First, we build the first (base) model, that does sentiment analysis on single tweets. This model achieves an accuracy of 72.2%. Next, we test if additional context in the form of text increases the base model's accuracy. For this second model, we keep the original design of our base model, in order to compare our results in a fair manner. We call this the conversations model, and its difference from the base model is that it takes pairs of conversing tweets as input. We observe only a slight improvement ($< 0.5\%$) compared to the base model's accuracy, as it goes up to 72.78%. Finally, we extend the conversations model to take into consideration the additional structural context. This context is derived from the Greek Twitter graph, and it is fed into the model in the form of two structural features vectors, one for each user that participates in a conversation. Unfortunately, this model achieves only around base-line accuracy at best, in all experiments, with the best performing model at 50.85% accuracy.

From this work we conclude that additional text features be of assistance in the sentiment analysis of Greek tweets, especially in cases where they multiply the sentiment of our target tweet. This means that when the two participating tweets in a conversation agree sentiment-wise (both positive or both negative), we get a better prediction. When the participating tweets disagree on their sentiments, our prediction does not improve because the labeling of the dataset does not consider the additional context.

Furthermore, we deduce that the structural features we provide to our models

work as noise. These additional data not only seem to not improve our model but they hinder the training process and result to a base-line accuracy. We think that the nature of these features is responsible for the poor performance and that better constructed structural features may produce better results.

2 Limitations

Below are the most significant limitations of our work:

- **Text data:** The text data we work with are extracted from Twitter. Twitter has a word limit for tweets, and it is also shown in figure 3.7 that tweets tend to be short. In order to feed our tweets (and conversations) in our neural models, we have to pad them to a fixed dimension. The padding with zero vectors in combination with the short length of tweets by nature, lead to datapoints with little information. For example, a tweet with three words: "Θέλουμε εκλογές αύριο!", transforms into 40 vectors of length 300, but 297 of them are zero filled vectors used for padding.
- **Structural data:** The structural data we extract from the Twitter graph are shown in tables 3.2 and 3.3. We can see in these tables that the features we have for the author of a tweet are somewhat generic, they outline the role of this user in relation to the whole Greek Twitter graph. We think that structural features that highlight the author relationship with the other user in the conversation carry more important information about the relationship between the users, and therefore, the sentiment of their conversation. Unfortunately, the feature extraction for each and every pair of users that interact in our dataset, would be impractical for Twitter conversations.
- **Labeling:** In our work, we use a sentiment dictionary [1] to label our datapoints. There are limitations to this type of labeling. First, words tend to have different meaning and sentiment in different cases. Second, we do not take into account negations, because we get the sentiment by word. For example, a word with a strong negative sentiment, combined with the equally strongly negative particle "δεν", leads to a strongly negative sentence, instead of a positive one which would be the sentiment us humans understand when we combine a negation with a negative connotated word. Lastly, the dictionary itself has some biases that carry on to our work, because it is made by humans, which inevitably project their own beliefs and experiences to the sentiment score they give to each word. A very interesting example is that the Greek word for "male" has a very positive sentiment in the dictionary, but the word for "female" has a negative sentiment, although both words describe something neutral.
- **Sentiment analysis:** Sentiment analysis as a problem itself, also has limitations. A main open problem is to detect sarcasm in text. Even as humans,

when we read a text we cannot always decide if the author is sarcastic or not. Of course, the context can give us more information to an extent, but it is not always easily detectable. Another problem, in a more macroscopic view, is that languages always evolve, and new slang words are created every day. This means that sentiment analysis dictionaries need to be updated continuously in order to stay up to date. Additionally, each space (or medium) uses language differently. For example, a word can have a very different connotation in a tweet and in a news article. Lastly, every person's experiences, upbringing and personality can affect the way they produce and understand written and spoken language. The problem of accurately interpretation of such texts is by nature hard, even if it is performed by humans.

3 Discussion

Our hypothesis for this work was that each time we add more context in the form of additional text and structural features, our model's predictions would be more accurate. For the conversation model we believe that additional context, especially the text to which our tweet of interest replies to, gives us the "feeling" of the whole conversation. However, the conversations model achieves similar accuracy as the base model. This can be attributed to the method we follow for the labeling; we do not take this context in account, when we create the dataset.

For the third model, we thought that underlying connections that exist in the Twitter graph between the users that exchange tweets would reveal the true sentiment of these exchanges. For example, two users that follow each other and belong to the same communities would overall have more positive conversations than people who are not "friends" on Twitter, even if they write something that seems negative out of context. As mentioned in the previous section, getting this structural information about each pair of users that interact in our dataset would be very slow. The structural features that we are able to extract and use give us a more macroscopic view of a users place in the Greek-speaking Twitter graph, and we cannot directly link this information with a specific relationship with another user. Furthermore, the issue with the labeling process remains; we do not include these structural features in the labeling process, and our model cannot identify the relationship between the structural features vector and the conversation vectors. For our model, the structural features are just noise, that's why its accuracy is around 50%, which is basically classifying our data based on a coin toss.

4 Future work

Even though our experiments could not support our hypothesis, we think there are many steps we can take in the future in order to examine the sentiment analysis problem from a different view.

The pre-trained Fasttext embedding model is trained with Wikipedia data, which are very different than Twitter data. We think that if we re-train the Fasttext with Twitter data, the vector embeddings will be more accurate for our tweets dataset. We can also try to use (and re-train) another embedding model, like BERT [9] or Word2Vec [17].

In order to create a more defined objective for our models, we can gather a dataset of Tweets that are on a specific topic or follow a specific hashtag. By focusing on a topic and give more emphasis to topic-relevant words we can limit the vagueness in the usage of certain words, and have higher accuracy as a result. In this approach it may also be useful to re-train the text embedding model with topic-specific tweets.

Regarding the model design, we can also perform our analysis using the stacked LSTM architecture [11] or a deeper model in general. We can also use Gated Recursive Convolutional neural network (grConv) [5], that are more efficient than LSTMs in sentiment analysis tasks, as seen in Yin *et al.* [22].

To create more representative structural features, we can focus on the most significant ones and extract information that describes the relationship between the two conversing users. This is not time efficient with the volume of structural features we extract in this work, but if the number of features is limited the extraction process will be faster.

Lastly, we can even try to feed the whole Greek Twitter graph to a graph embedding model such as pytorch BigGraph [14] to create more representative structural features for our users.

Bibliography

- [1] Despoina Antonakaki, Dimitris Spiliotopoulos, Christos V. Samaras, Sotiris Ioannidis, and Paraskevi Fragopoulou. Investigating the complete corpus of referendum and elections tweets. In *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 100–105, 2016.
- [2] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. ” O’Reilly Media, Inc.”, 2009.
- [3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.
- [4] Johan Bollen, Huina Mao, and Xiaojun Zeng. Twitter mood predicts the stock market. *Journal of computational science*, 2(1):1–8, 2011.
- [5] Kyunghyung Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv:1409.1259*, 2014.
- [6] François Chollet et al. Keras. <https://keras.io>, 2015.
- [7] M.D. Conover, J. Ratkiewicz, M. Francisco, B. Gonçalves, A. Flammini, and F. Menczer. Political polarization on twitter. *Association for the Advancement of Artificial Intelligence*, 2011.
- [8] Michael Conover, Jacob Ratkiewicz, Matthew R Francisco, Bruno Gonçalves, Filippo Menczer, and Alessandro Flammini. Political polarization on twitter. *ICWSM*, 133:89–96, 2011.
- [9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*, 2018.
- [10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, <https://doi.org/10.1162/neco.1997.9.8.1735>, 1997.

- [11] Minlie Huang, Yujie Cao, and Chao Dong. Modeling rich contexts for sentiment classification with lstm. *arXiv:1605.01478v1*, 2016.
- [12] Georgios Kalamatianos, Dimitrios Mallis, Symeon Symeonidis, and Avi Arampatzis. Sentiment analysis of greek tweets and hashtags using a sentiment lexicon. *19th Panhellenic Conference on Informatics* <https://dl.acm.org/doi/10.1145/2801948.2802010>, 2015.
- [13] Dimitrios Kydros, Maria Argyropoulou, and Vasiliki Vrana. A content and sentiment analysis of greek tweets during the pandemic. *Sustainability* 2021, 13, 6150. <https://doi.org/10.3390/su13116150>, 2021.
- [14] Adam Lerer, Ledell Wu, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. Pytorch-biggraph: A large-scale graph embedding system. *arXiv:1903.12287*, 2019.
- [15] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *ACL (System Demonstrations)*, pages 55–60. The Association for Computer Linguistics, 2014.
- [16] George Markopoulos, George Mikros, Anastasia Iliadi, and Michalis Liontos. Sentiment analysis of hotel reviews in greek: A comparison of unigram features. <http://users.uoa.gr/~gmikros/Pdf/Sentiment%20Analysis%20of%20Hotel%20Reviews%20in%20Greek.pdf>, 2015.
- [17] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv:1301.3781*, 2013.
- [18] Sascha Narr, Michael Hulfenhaus, and Sahin Albayrak. Language-independent twitter sentiment analysis. *Knowledge discovery and machine learning (KDML), LWA*, pages 12–14, 2012.
- [19] Polyvios Pratikakis. twawler: A lightweight twitter crawler. *arXiv:1804.07748*, 2018.
- [20] Jacob Ratkiewicz, Michael Conover, Mark R Meiss, Bruno Gonçalves, Alessandro Flammini, and Filippo Menczer. Detecting and tracking political abuse in social media. *ICWSM*, 11:297–304, 2011.
- [21] Jenq-Haur Wang, Ting-Wei Liu, and Xiong Luo Long Wang. An lstm approach to short text sentiment classification with word embeddings. *The 2018 Conference on Computational Linguistics and Speech Processing*, pages 214–223, 2018.
- [22] Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. Comparative study of cnn and rnn for natural language processing. *arXiv:1702.01923*, 2017.

1 APPENDIX

In this section we present some of our failed attempts at creating models. First, we create the base model as shown in figure 1

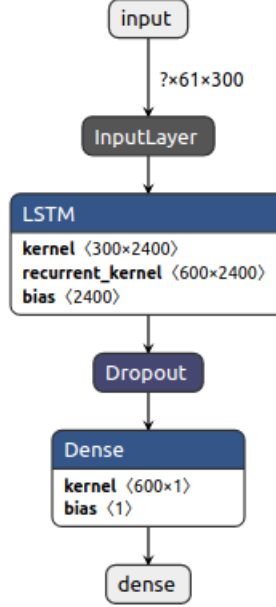


Figure 1: Base model using LSTM and dropout

We tried using dropout to only keep a small piece of information that represents the whole tweet (what we achieve with the usage of the lambda layer in our final model). This architecture did not work and the model had very low accuracy, as seen in figure 2

Next, we tried using the stacked LSTM architecture as presented in Huang *et al.* [11]. We tried using this architecture for the conversations model, in figure 3. In these attempts, we give the two tweets of the conversations in two separate input layers. We figure that in the following three model versions, the input is misunderstood as two different entities.

We also used this architecture for the structural features model, with two versions, one with a dense layer and one without. See figures 4 and 5

These models, feel wrong semantically, because our data are disconnected. Also, it seems that we keep irrelevant information that makes the training of these models very time consuming. The HLSTM with the dense layer (figure 5) took eighteen hours per epoch to train with this architecture and a heavier version of our data (more samples and more data per sample, that we used at first).

```

Epoch 1/10
552/552 [=====] - 24s 44ms/step - loss: 0.0000e+00 - accuracy: 0.3184 - val_loss: 0.0000e
+00 - val_accuracy: 0.3180
Epoch 2/10
552/552 [=====] - 23s 42ms/step - loss: 0.0000e+00 - accuracy: 0.3184 - val_loss: 0.0000e
+00 - val_accuracy: 0.3180
Epoch 3/10
552/552 [=====] - 23s 42ms/step - loss: 0.0000e+00 - accuracy: 0.3184 - val_loss: 0.0000e
+00 - val_accuracy: 0.3180
Epoch 4/10
552/552 [=====] - 23s 42ms/step - loss: 0.0000e+00 - accuracy: 0.3184 - val_loss: 0.0000e
+00 - val_accuracy: 0.3180
Epoch 5/10
552/552 [=====] - 23s 42ms/step - loss: 0.0000e+00 - accuracy: 0.3184 - val_loss: 0.0000e
+00 - val_accuracy: 0.3180
Epoch 6/10
552/552 [=====] - 23s 42ms/step - loss: 0.0000e+00 - accuracy: 0.3184 - val_loss: 0.0000e
+00 - val_accuracy: 0.3180
Epoch 7/10
552/552 [=====] - 23s 41ms/step - loss: 0.0000e+00 - accuracy: 0.3184 - val_loss: 0.0000e
+00 - val_accuracy: 0.3180
Epoch 8/10
552/552 [=====] - 23s 42ms/step - loss: 0.0000e+00 - accuracy: 0.3184 - val_loss: 0.0000e
+00 - val_accuracy: 0.3180
Epoch 9/10
552/552 [=====] - 23s 42ms/step - loss: 0.0000e+00 - accuracy: 0.3184 - val_loss: 0.0000e
+00 - val_accuracy: 0.3180
Epoch 10/10
552/552 [=====] - 23s 42ms/step - loss: 0.0000e+00 - accuracy: 0.3184 - val_loss: 0.0000e
+00 - val_accuracy: 0.3180

ut[17]: <tensorflow.python.keras.callbacks.History at 0x7f9bf94ccf28>

n [18]: ss = sentence_matrix_df.head(len(y_val))
score, acc = model.evaluate(ss, y_val)

print('Test score:', score)
print('Test accuracy:', acc)

550/550 [=====] - 6s 12ms/step - loss: 0.0000e+00 - accuracy: 0.3180
Test score: 0.0
Test accuracy: 0.31804490089416504

```

Figure 2: Training of base model with dropout layer

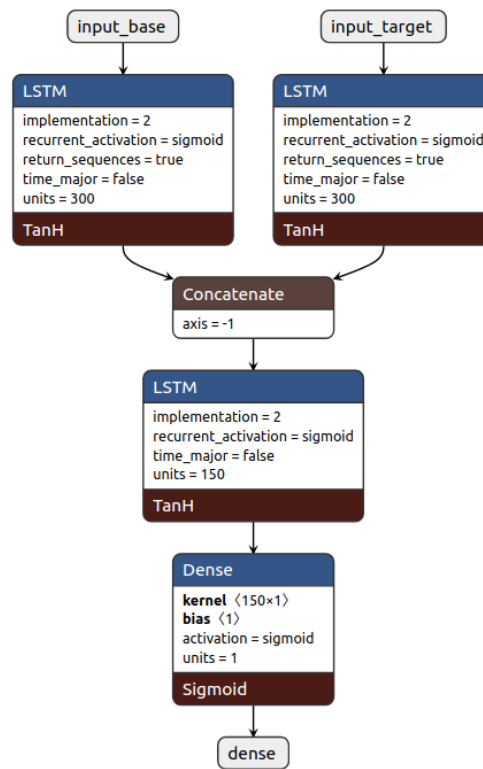


Figure 3: HLSTM architecture for conversations model

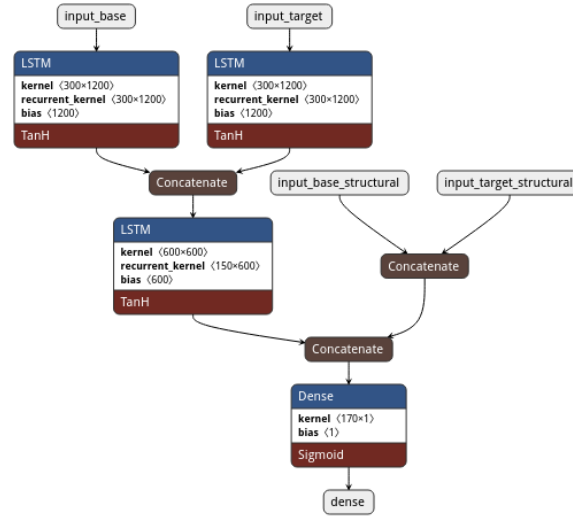


Figure 4: HLSTM architecture for structural model

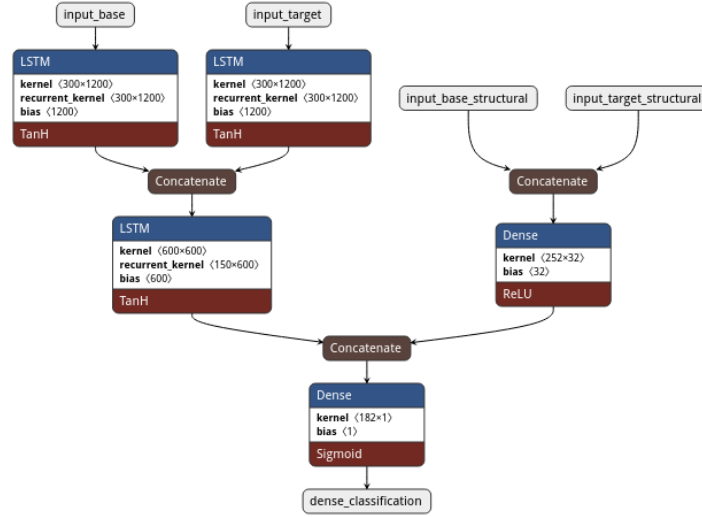


Figure 5: HLSTM architecture for structural model with an additional dense layer

In our final and working models, we use the lambda layer which is absent in our failed attempts. This layer allows us to keep a smaller representation of the tweet (or conversation) and makes the number of parameters for the model smaller. Also, we choose not to separate the conversation nor the structural features, because intuitively it makes more sense to process them together. Finally, the second level of LSTM layers is not needed anymore in our architecture because of the lambda layer. Without the lambda layer we still get a sequence of representations, but with the lambda layer we keep only one representation for each datapoint.