

# Project Report: Sarcasm Detection <sup>★</sup>

by Sakina Hajiyevea

ELTE University, Computer Science, Data Science Specialization, Budapest, Hungary

**Abstract.** Sentiment analysis is often used in NLP to understand people's subjective opinions. However, the analysis results may be biased if people use sarcasm in their statements. In order to correctly understand people's true intention, being able to detect sarcasm is critical, especially in its application in many areas of interest of NLP applications, including marketing research, opinion mining and information categorization, which are very important for companies and their prosperity. This paper addresses a key NLP problem known as sarcasm detection using several models and experimenting on each for further insights and deeper understanding of the topic.

**Keywords:** Sarcasm Detection on Reddit Dataset · Sentiment Analysis · NLP · Convolutional Neural Networks (CNN) · Deep Learning

## 1 A General Introduction to Sarcasm Detection

### 1.1 Explanation of a notion

Sarcasm can be considered as expressing a bitter gibe or taunt. In brief, if we address the dictionary we would have the following definition: "Sarcasm is a sharp and often satirical or ironic utterance designed to cut or give pain". In a simple language it is the use of remarks that clearly mean the opposite of what is said, made in order to hurt someone's feelings or to criticize something in a humorous way.

It can be easier to understand the term with some examples: "Do you want help? - No, thanks, I am really enjoying the challenge", "I work 40 hours a week to be this poor", or when something bad happens "That's just what I needed today!".

### 1.2 The Problem of Sarcasm Detection

As human beings we have a social nature and interacting with each other involves not only sentences, but also emotions, the tone of voice and mimics, which makes it easier to understand each other. Sarcasm is both positively funny and negatively nasty and plays an important role in human social interaction. However, sometimes it is difficult to detect whether someone is making fun of us with some irony. This is why it is important to create something which helps us

---

<sup>★</sup> Supported by ELTE University.

to detect sarcasm. For computers, which do not know any emotions or humour, it is challenging to differentiate between sarcastic and non-sarcastic statements, however, it is not impossible.

**How Computers Detect Sarcasm?** This question interested me from the beginning of the project. Of course, we do have several models predicting sarcasm, however, I was interested in the background processes in how computers understand and differentiate sarcasm. I am sharing my findings regarding this question. In any effort to detect sarcasm, a key ingredient to check for is "incongruity" - ,that is, is there a disparity between the literal meaning of the sentence and the author's underlying intention? In simpler terms, the computer is trying to find conflicting sentiments. One way for computers to detect incongruity is to compare words within a sentence, looking for positive expressions accompanying a negative scenario or vice versa.

However, simply identifying conflicting sentiments together in a sentence is not enough to prove that the sentence is sarcastic. The composition of the sentence is another key factor. For example, this sentence also contains a positive description and a negative concept, but obviously cannot be counted as sarcastic:

" I like all birds except my neighbour's screaming cockatoo."

Other challenges that exist in understanding sarcastic statements is the reference to multiple events and the need to extract a large amount of facts, commonsense knowledge, anaphora resolution (or pronoun resolution- problem of resolving references to earlier or later items in discourse, which are usually represented by noun phrases referring to objects in the real world, however, can also be verb phrases, whole sentences or paragraphs), and logical reasoning. Another way of detecting whether a statement is sarcastic is to look at personality-based features (people, who have the tendency to post something sarcastic will continue doing so, which makes it easier to determine irony quickly just by the personal habits). There are some other features as sentiment and emotion-based features incorporating into the sarcasm detection framework. Each set of features are learned by separate models, becoming pre-trained models for extracting sarcasm-related features from a dataset.

## 2 Exploratory Data Analysis and Approaches

### 2.1 Taken Steps in Data Cleaning

A data science or machine learning project can only be as good as the data it's fed, so the first step of any project is to examine the dataset. We'll use the

train-balanced-sarcasm.csv file provided. The next step that goes hand-in-hand with examining is cleaning the dataset, but since this one is already pre-cleaned we won't have to worry about that.

What is the goal of examining the data?

1. Understand what the features represent.
2. Observe what type of data you are working with (structured vs. unstructured? text, numbers, dates, categories, etc?).
3. Determine if there are missing values.
4. Identify any anomalies or outliers.
5. Examine the target variable in particular and check if the classes are balanced or imbalanced.

Our Reddit sarcasm dataset

(the link for downloading is here <https://dl.dropboxusercontent.com/s/qpkodtmj5a6p4z3/sarcasm.zip?dl=0>) contains of 1010826 rows and 10 columns. This dataset contains 1.3 million Sarcastic comments from the Internet commentary website Reddit. The dataset was generated by scraping comments from Reddit containing the ( sarcasm) tag. This tag is often used by Redditors to indicate that their comment is in jest and not meant to be taken seriously, and is generally a reliable indicator of sarcastic comment content.

Data has balanced and imbalanced (i.e true distribution) versions. (True ratio is about 1:100). The corpus has 1.3 million sarcastic statements, along with what they responded to as well as many non-sarcastic comments from the same source. The data was gathered by: Mikhail Khodak and Nikunj Saunshi and Kiran Vodrahalli for their article "A Large Self-Annotated Corpus for Sarcasm". Labelled comments are in the train-balanced-sarcasm.csv file. As column names we have label, comment, author, subreddit, score, ups, downs, date, created\_utc, parent\_comment. After clarifying all the columns, their meaning and impact on first glance, we should determine the missing values. When we check the missing values we see that our dataset contains 53 empty comments, the best way here is just to remove those, we will not need them in our analysis. In addition, our data is balanced, which means the number of 0 and 1 labeled data is equal. Then, I created Wordclouds to see the frequency of mostly used words, which did not really help at that moment, but can be kept for future analysis. Next, I imported the NLTK (Natural Language Toolkit) package, which is a very useful package for text mining and analysis. It is great for sentiment analysis and implementing such algorithms as Named Entity Recognition, tokenizing, part-of-speech tagging (POS) and topic segmentation. As the next step, I switched my attention towards subreddits column, because it also may contain some sarcastic comments. Unique subreddits are 14876, which is much. In addition, I checked the authors list for the sarcastic subreddits they posted, and tried to group them, which was just done out of curiosity to see if I can get some insights from this piece of data. After all these steps were taken, I split the data into train and test sets with the ratio as follows test\_size=0.25, which makes our test part in the form of (252694, 10).

Computers do not understand the text data as we humans do, this is why we need

to vectorize the data, which basically means to represent text with numbers. We will do this by creating a bag of words representation for each comment. We take a look at all the words in all of the comments and make those the columns of our new feature matrix. Then, we count how many times each word appears in each comment and place those values in the appropriate spots in the matrix. I used the CountVectorizer to accomplish this. There are several different parameters that we can play with, but for now I set max\_features=20000, so that I can save some computational time. As a result, it is clear that the matrix is very sparse, meaning that it has a lot of 0 values. This makes sense, because each comment is only going to have a few out of those 20,000 words.

Another useful step in text mining is to remove stopwords (the repeated words that add no meaning to the context of the text, usually are pronouns, articles, conjunctions and etc.). It is important to also make sure all the punctuation and unnecessary symbols (such as hashtags and etc.) are all eliminated and all words are in lower case.

The next step suggested is to use a TfidfTransformer to transform the matrix of counts into a matrix of scaled term frequencies. This piece of our pipeline also has some parameters that could vary, but I will use the defaults. It is worth noting that the transformation is performed on the bag of words matrix, not on the original text data.

Next, we'll start with models, to be precise, a Logistic regression model.

### 3 Models

#### 3.1 Logistic Regression Attempt

**Logistic Regression Algorithm and Formula** Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. It can help us to predict whether the statement is sarcastic or not sarcastic. Logistic regression predictions are discrete (only specific values or categories are allowed). Given a data(X,Y), X being a matrix of values with m examples and n features and Y being a vector with m examples. The objective is to train the model to predict which class the future values belong to. Primarily, we create a weight matrix with random initialization. Then we multiply it by features.

$$a = w_0 + w_1x_1 + w_2x_2 + \dots w_nx_n$$

Eq 1.

We then pass the output obtained from Eq 1. to a link function. This is followed

$$\hat{y}_i = 1/(1 + e^{-a})$$

Link Function

$$cost(w) = (-1/m) \sum_{i=1}^{i=m} y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

cost function

by calculating the cost for that iteration whose formula is as follows in cost function formula.

The derivative of this cost is calculated following which the weights are updated.

$$dw_j = \sum_{i=1}^{i=n} (\hat{y} - y) x_j^i$$

Gradient

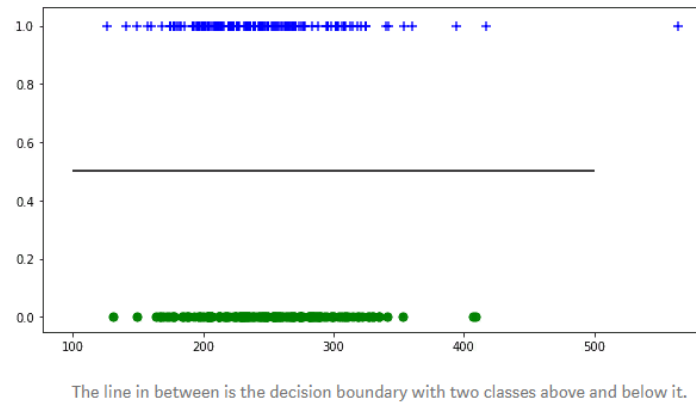
$$w_i = w_j - (\alpha * dw_j)$$

Update

The goal of the logistic regression algorithm is to create a linear decision boundary separating two classes from one another. This decision boundary is given by a conditional probability.

After going a little into deep of logistic regression, I started with an untuned LogisticRegression model, so I cross validated it with 3 folds to get some accuracy values. As a result of waiting for approximately 20 minutes, Logistic Regression model achieved **69 %** accuracy. This is a nice result for such a short time we can get, however, for improving it I take several steps as follows.

First, let's start by taking a closer look at some of the parameters in our CountVectorizer. The first parameter I want to set is `strip_accents='unicode'`, which will remove accented letters from the comments are replace them with unaccented ones.



Next, we can see that the vectorizer has built-in functionality to remove stopwords, or words that are so common in the English language so as to be useless in classification. Though I can set the stopwords list to be the pre-built one, it may remove words that might be useful to us; therefore, I created my own. The next useful feature is `ngram_range`, which allows us to use not only single words, but also multi-word phrases (called ngrams) as features. By default, it's set to `(1, 1)`, which means we only use ngrams of length 1 (i.e. single words, or unigrams). We can set it to `(1, 2)` to use unigrams and bigrams, or higher numbers if they are useful. I will save this for cross validation.

We then have two closely-related parameters we can play with: `min_df` and `max_df`. These parameters allow us to automatically disregard words that occur in a certain number of comments. For example, words that appear in only a tiny fraction of the comments would not be very representative and may not be useful for classification. Likewise, words that appear in most comments will not be that useful either.

I will set `max_df=0.70`, which will throw out all words that occur in more than 70 % of comments. This makes sense to me because about 50 % are sarcastic and 50% are not, so a word will be useful if it mostly occurs only in sarcastic comments or only in non-sarcastic ones, hence a maximum percentage close to 50%. I will set `min_df=0.0001`, because words that occur only in 75 out of over 758,000 comments do not have a high chance of being useful.

The last useful parameter I will consider is `max_features`, which controls the maximum number of words we will retain as features in our new feature matrix. By setting `max_df` and `min_df` we will be able to limit the number of features we have. However, once we start considering bigrams, trigrams, and (maybe) beyond, we will once again have an enormous list. I will limit the number of features we keep to 50,000 or less. This is another parameter we can potentially configure with cross validation. In this part I used cross validation with `GridSearchCV`. After this grid search finally finishes running (10 hours of awaiting), we can see that the best model it found had 15,000 max features, an

n-gram range of (1, 3), and an ll\_ratio of 0.75. This model achieved an accuracy of **0.70**. We can possibly improve upon this by trying more different values for the parameters in a more granular range. However, we also have other models to attempt.

### 3.2 Naive Bayes Attempt

The Naive Bayes classifier is one that is based on probability using Bayes' theorem. It calculates the probability of a sample to be in a certain category, based on prior knowledge. I want to use Multinomial Naive Bayes, where multinomial means there are more than two possible outcomes. Let me quickly go through some basics :

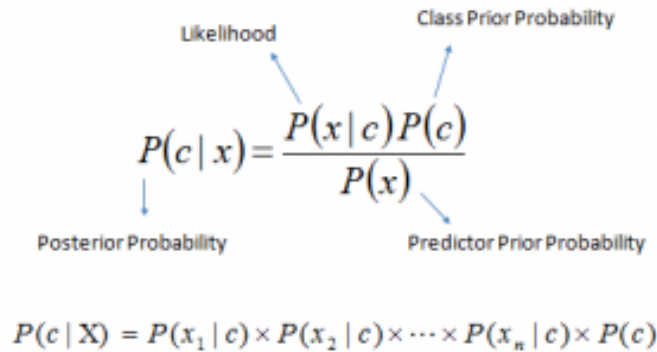
Here

$P(c|x)$  is the posterior probability of class (c, target) given predictor (x, attributes).

$P(c)$  is the prior probability of class.

$P(x|c)$  is the likelihood which is the probability of predictor given class.

$P(x)$  is the prior probability of predictor.



The diagram shows the Naive Bayes formula with labels for its components. The formula is  $P(c|x) = \frac{P(x|c)P(c)}{P(x)}$ . Arrows point from the labels to the corresponding parts of the formula: 'Likelihood' points to  $P(x|c)$ , 'Class Prior Probability' points to  $P(c)$ , 'Posterior Probability' points to  $P(c|x)$ , and 'Predictor Prior Probability' points to  $P(x)$ .

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \dots \times P(x_n|c) \times P(c)$$

Naive Bayes classifiers are mostly used in text classification (due to better result in multi class problems and independence rule) and have higher success rate as compared to other algorithms. As a result, it is widely used in Spam filtering (identify spam e-mail) and Sentiment Analysis (in social media analysis, to identify positive and negative customer sentiments).

The particular type of Naive Bayes I am using is Multinomial Naive Bayes, which is good for working with text datasets like the one we have.

The only parameter to modify here would be alpha, which controls the smoothing of probabilities, but I left it at the default value of 1. The process took approximately an hour and a half.

The results show that the best classifier achieved a maximum accuracy of **69%** with 15,000 features that include uni-, bi-, and trigrams. This is just about the

```

vect = CountVectorizer(strip_accents='unicode', stop_words=stopwords, min_df=0.0001, max_df=0.70)
tf_trans = TfidfTransformer()

# creating the NB model
nb_model = MultinomialNB()

pipeline = Pipeline([
    ('vect', vect),
    ('tftrans', tf_trans),
    ('model', nb_model)
])

param_grid = {
    'vect_ngram_range': [(1, 1), (1, 2), (1, 3)],
    'vect_max_features': (5000, 15000, 30000)
}

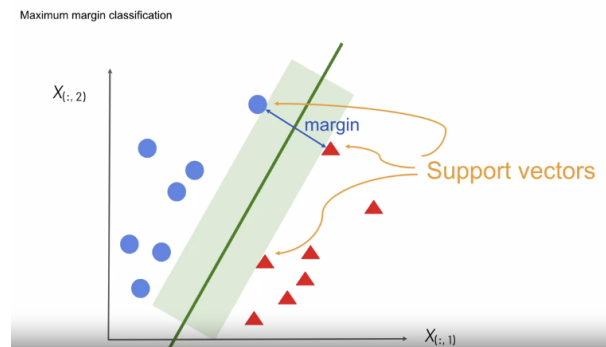
grid_nb = GridSearchCV(pipeline, param_grid, scoring="accuracy", cv=3, n_jobs=-1)
grid_nb.fit(train_comments, train_df["label"])

```

same as the Logistic Regression classifier above. One option we have is to add some sort of feature selection step to reduce the number of features we train on. The Logistic Regression classifier did this as part of the training process (due to the `l1_ratio` allowing for some Lasso regularization), but the Naive Bayes classifier would need this to be done explicitly. But first, we can try the Support Vector Machine.

### 3.3 SVM or Support Vector Machine Attempt

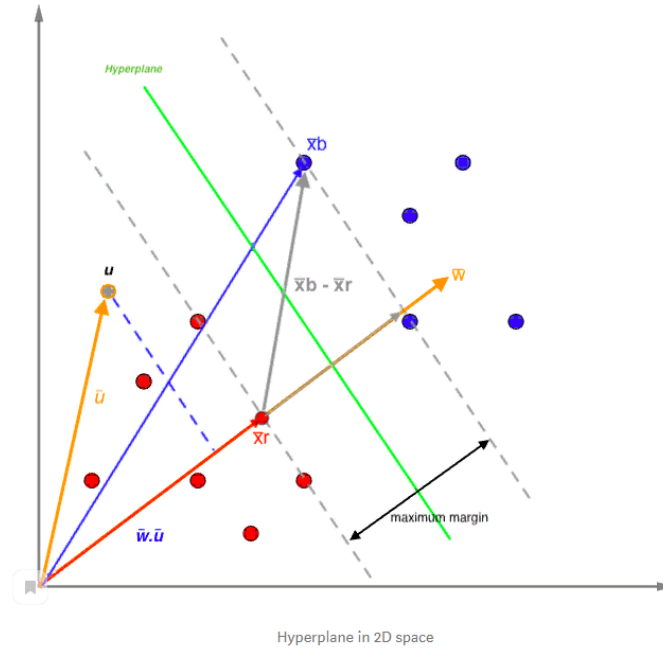
Support Vector Machine models are basically drawing a line to divide one class of data points from the other. The trick is finding the best line. SVMs are one of the most powerful tools in Supervised Machine Learning for both classification and regression, they are also called Large Margin Classifiers, since we try to find the best line that maximizes the distance (also margins) between itself and the closest data points from both classes (see the picture below).



Let's consider a line (hyperplane) which separates the points optimally as shown



below. The points on the inner side of the line are red and on the outer side is the blue ones. This line is optimal hyperplane, which means it has a maximum margin. We have the point  $u$  which is unknown to us and we want to decide whether it is red or blue. A vector  $\bar{w}$  has drawn such that it is perpendicular to our line while the vector  $u$  is point vector to  $u$ .  $\bar{x}r$  and  $\bar{x}b$  are vectors for a red and blue point respectively. The dot product of  $\bar{w}$  and  $\bar{u}$  decides whether the



point  $u$  is red or blue. If it is greater than certain value then it is blue else it is red. We have constraints about our point as shown in Equation 1. Next thing we want to do is to find the width of our margin.

We want our points to be as separated as possible, so we want this width to be maximum. Here,  $\bar{w}$  can be minimised to increase the width. In other words we can write this as below

$$\bar{w} \cdot \bar{x}_b + b \geq 1 \quad \bar{w} \cdot \bar{x}_r + b \leq 1$$

Equation 1

To simplify things we are introducing a new symbol  $y$  in the equation so that these two conditions can be merged. Let us consider  $y = +1$  for blue and  $-1$  for red. After multiplying  $y$  with both equations we get the following equation.

$$y (\bar{w} \cdot \bar{x}_{r|b} + b) - 1 \geq 0$$

Equation 2

To find the width, we can use the dot product of unit vector in direction of  $\bar{w}$  and  $(\bar{x}_b - \bar{x}_r)$  as shown in our 2D space.

$$(\bar{x}_b - \bar{x}_r) \cdot \frac{\bar{w}}{\|\bar{w}\|}$$

This equation exactly represents the margin. We can fit the *Equation 2* into this equation and we will get the following result.

$$\overset{1-b}{\curvearrowright} (\bar{x}_b - \bar{x}_r) \overset{1-b}{\curvearrowright} \cdot \frac{\bar{w}}{\|\bar{w}\|} = \frac{2}{\|\bar{w}\|}$$

Obtaining the margin

$$\min \left( \frac{1}{2} \|\bar{w}\|^2 \right)$$

Equation 3

In Equation 4,  $\alpha_i$  represents the lagrange multiplier and the  $b$  represents the constant. In our case, as the constraint applies to each point in space, we take summation of constraint on each point.

$$L = \frac{1}{2} \|\bar{w}\|^2 - \sum_i \alpha_i [y_i (\bar{w} \cdot \bar{x}_i + b) - 1]$$

Equation 4

To maximise the width we need to prove that  $\nabla L = 0$ . After doing some calculation we can get the following interesting things

$$\bar{w} = \sum_i \alpha_i y_i x_i - \sum_i \alpha_i y_i = 0$$

Equation 5 and Equation 6

Great! after putting the corresponding values in equation 4 we can get the following equation

$$L = \sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \bar{x}_i \bar{x}_j$$

The final equation

In particular, I will be using an SVM solved by Stochastic Gradient Descent, which is an approach used to train linear classifiers with a convex cost function. There are lots of parameters to modify in the SGDClassifier. We can change the type of loss function and type of penalty term we use. We can change the `l1_ratio` that has to do with the regularization penalty as well, the number of iterations to permit and the tolerance for the stopping condition, we can also change how to determine the learning rate and various constants that are relevant to the mathematical formulae that define the SVM. I kept it the same and trained the model for almost 3 hours.

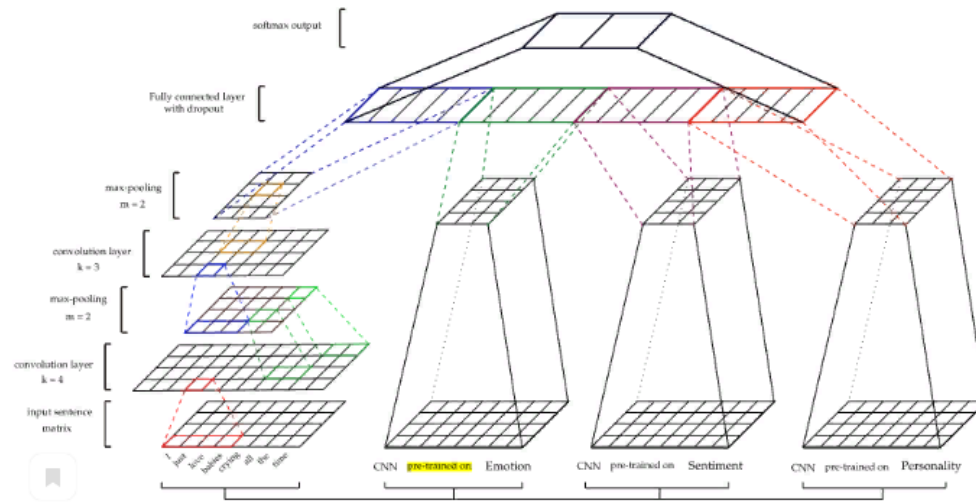
Our results show that the best model as determined by our grid search achieved **0.68** accuracy, with `l1_ratio=0` (indicating Ridge regularization) and a list of 15,000 uni-, bi-, and trigrams. This is below the accuracy of the other models, but since the difference is very slight it may not be significant.

All three of the models we tried agreed upon 15,000 features and the inclusion of bi- and trigrams. If we were to continue running grid search to select a model, we could try running it with `max_feature` values of 10,000, 15,000, 20,000, and 25,000. We may also decide to try 4-grams.

### 3.4 Deep Learning Approach

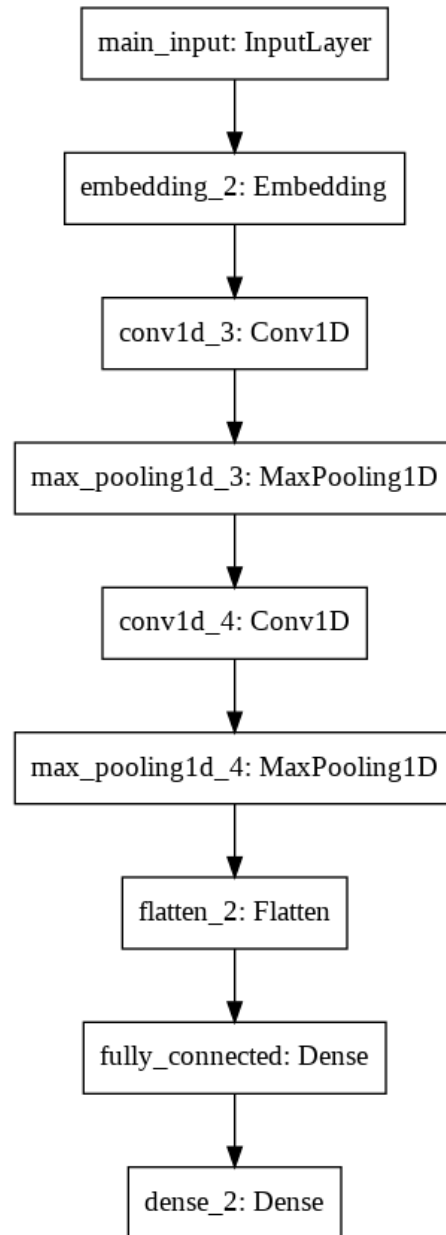
In this part of my report I finally discuss Deep Learning Model I implemented, to be precise, the Convolutional Neural Networks. CNNs are effective at modeling hierarchy of local features to learn more global features, which is essential to learn context. Sentences are represented using word vectors (embeddings) and

provided as input. Google's word2vec vectors are employed as input. Non-static representations are used, therefore, parameters for these word vectors are learned during the training phase. Max pooling is then applied to the feature maps to generate features. A fully connected layer is applied followed by a softmax layer for outputting the final prediction. (See diagram of the CNN-based architecture below)



Here I would like to show the parameters I set for this model: Fully connected with 2 hidden layers of 300 nodes and 4 nodes. ReLU function for activation in hidden layers, and Softmax in output layer. Trained for 50 epochs.

- epochs = 50
- learning\_rate = 0.001
- embed\_size = 300
- hidden\_size = 128, 256
- output\_size = 2 (binary classification)
- batch\_size = 64
- activation = 'relu', 'softmax'
- loss\_fuction = categorical\_crossentropy
- optimizer = optim.Adam



The baseline model I trained appeared to take 3-4 hours to train and gave the results with the accuracy rate varying from **71-74%**, depending on its mood I guess. Of course, if I change some parameters the results and number of epochs may vary, however, Deep Learning model once again proved its worth in use terms. Although, deep learning models tend to overfit, the results I got appear to be solid and, in the future, if I work more on feature engineering and parameters tuning the accuracy may hit 80% and even more. Unfortunately, because of time limitatins and the fact that training models takes long enough, I could not experiment as much as I wanted, however, during the summer I plan to play with this and other sarcasm and sentiment related data for the sake of experience and knowledge.

On the previous page you can see the process, or workflow of the model.

## 4 Conclusion

Let us generalize and compare the results by the models trained and the respective accuracy I got bt training those models:

- Logistic Regression: Accuracy of 69%
- Logistic Regression + GridSearch: Accuracy of 70%
- Naive Bayes (Multinomial): Accuracy of 69%
- Support Vector Machines: Accuracy of 68%
- Convolutional Neural Networks: Accuracy of 71-74%

From the following results we can deduce that there is still a room for improvement and the accuracy rate can get higher if we spend some more time with trying out new models,doing some parameter tuning and in-depth feature engineering.

In conclusion, although I chose this topic without prior research, just out of curiosity in sarcasm topic itself and somehow because of my personality, I am very happy that I took this challenging project and will try to continue pursuing more of my time going deeper into research. This was a great opportunity for me to grasp some basic knowledge related to NLP, sentiment analysis and Neural Networks, and to improve my skills and knowledge even further on this path, maybe for my thesis work.

## References

1. Some articles were reviewed from the internet related to NLP, Sentiment Analysis, Deep Learning, Models I trained and the understanding of Mathematics behind each.
2. Mikhail Khodak, Nikunj Saunshi, Kiran Vodrahalli. 2017. A Large Self-Annotated Corpus for Sarcasm
3. Silvio Amir, Byron C. Wallace, Hao Lyu, Paula Carvalho, Mario J. Silva. 2016. Modelling Context with User Embeddings for Sarcasm Detection in Social Media