CSC 461 - Intro. to Machine Learning

Final Project

Sentiment Analysis

06/12/2024

Instructor: Dr. Seifedine Kadry

Mohamad Hijazi 202207795

Kamal Dbouk 202203415

I. Project Proposal

Sentiment analysis is the process of digitally determining the attitude of the writer's text. In this project, the primary goal is to analyze the sentiment of data and classify them by "Positive sentiment" or "Negative sentiment". The focus is on Logistic Regression (LR) and k-Nearest Neighbours (kNN).

Objective: The project focuses on sentiment analysis, aiming to classify tweets as positive or negative. This could be impactful for social media monitoring, customer feedback analysis, and sentiment-driven business decisions.

Data: For the first dataset, we utilized tweets from the Natural Language Toolkit, specifically the collection called "twitter_samples". This collection contains pre-labeled tweets making this model fall under the category of supervised learning. This dataset consists of a total of 10,000 tweets. In order to avoid biases, the data is split into 5,000 positive tweets and 5,000 negative tweets. The reason using tweets can be effective is because they are written using everyday language, reflecting real-world cases which also include slang.

For the second dataset we opted for a bigger and more randomized collection of tweets for further training of the model. We extracted 10,000 positive tweets and 10,000 negative tweets for a total of 20,000 (https://github.com/cblancac/SentimentAnalysisBert/blob/main/data).

Approach: We applied LR and kNN classifiers to predict the sentiment as they are simple to integrate and effective in handling such tasks. LR is widely used for binary classification and performs well on linearly separable data. Thus, using it to classify "Positive V.S. Negative" was a good idea. On the other hand, kNN works by storing all available data locally and classifying new data according to similarity. We use both LR and kNN in two different instances where we edit the preprocessing segment of the code which will be discussed further below. We also developed a simple Django app using the LR model to classify text input from the user. Below are screenshots of that application.

In order to run this code, you must unzip the file attached with this submission and open it using PyCharm. In order to run the code, you must run python manage.py runserver in the terminal. Paste http://127.0.0.1:8000/polls/ in the URL in order to access the application. In order to find the code with detailed documentation, you must open the .ipynb file in Google Colab.

II. Data Exploration and Preprocessing

For both models, we prepare the data using two different preprocessing functions to study the behaviour of each. The first one is more basic, it removes any special characters from the tweets such as dollar signs ($), URLs, hashtags (#), etc. Common stop words like "the", "and", "is", etc. are also removed according to the NLTK's stop word list and a function is used to take out numbers. Punctuation is also disregarded. We use NLTK's stemming function to return words to their root form (e.g. running becomes run). Finally, we tokenize tweets, meaning we turn them into an array of individual words. Thus, the output of this function would be an array of individual stemmed words excluding the irrelevant data. While working on this, we became aware of the fact that negating a word does not return an accurate result. This is due to words like "not", "never", and "-n't" being disregarded. Due to this, we implemented a different preprocessing function that took into consideration negating words. The way this worked is when a common negating word was identified, a boolean value "negation" was set to true, and that led to the next word in the sequence to be processed as "NEG_word". This meant a word preceded by "not" would have a negative tag to identify it correctly.

We also implemented a function called "build_freq" that builds a frequency dictionary of the pair of pre-labeled words and their sentiment with how many times they were repeated. An example of this would be the following:

{ ('happi', 1) : 1, ('sad', 0) : 2 } → "happi" representing the stemmed version of "happy", "happier", "happiest", "happily", etc. with the corresponding sentiment for positive (1). This is followed by a colon with a number that indicates how many times it was repeated, in this case 1. Similarly, we can infer that "sad" is of negative sentiment (0) and was mentioned twice in the data in all of its forms.

The data is already divided into positive and negative tweets. In terms of training, we used 80% by using 4,000 tweets from the positive and 4,000 from the negative (8000 per negative and positive for the second dataset). We keep them equal to avoid creating a bias. The remaining tweets (20%) will be used for testing. We then use the previously defined function "build_freq" to create the frequency dictionary.

III. Model Selection and Training

In order to classify sentiment of tweets, we implemented two models, LR and kNN. For the LR, which predicts the probability of a class based on a logistic function, , we chose to use it because it performs well on linearly separable data. kNN , which classifies the based on the majority label of its closest neighbors, was implemented for the opposite reason, for its ability to model non-linear patterns which is needed when the data preprocessing adds the layer of negations. For logistic regression, there were no specific hyperparameters focused on as much as the preprocessing was. However, we did play around with the learning rate and number of iterations for the gradient descent function and found that a learning rate of 1e-9 coupled with 15,000 was yielding the best results. For the kNN model, we looped over the number of

k-neighbours and ultimately found that k=20 was yielding the best results.We also used “n_jobs” ,Setting it to -1 to use all available cores, speeding up the algorithm when dealing with the larger dataset.

We evaluated the accuracy by the following:

$$Accuracy = \frac{Correct\ Predictions}{Total\ Predictions}$$

We also evaluated the accuracy between the two preprocessing functions differently by using obvious sentiments to test the predictions like “I am not happy”.

In terms of training, either the data was processed with negation handling or without, then in both situations, converted into feature vectors to be used as weights for the LR sigmoid function or to be scattered locally to be prepared for prediction in the case of kNN. Both models were trained on these frequency based features.

IV. Evaluation and Analysis

Initially, the original model did not handle negation in the preprocessing part and on Dataset 1.We were able to see that the model performed well with a 92% accuracy for both LR and kNN. We were able to attribute this success to the fact that the data was very clearly linearly separable between “Positive” and “Negative”.They can identify strong correlations between individual words like "happy" and "sad". The success can also be attributed to the data being small. Since tweets are actually limited to 280 characters maximum, the data we are training and testing on is relatively small.

This brings us to the main limitation in the model: Both LR and kNN classified statements that had negations incorrectly. This is because the initial pre-processing function disregards stopwords like "not".This means that statements such as " I am not happy" will be considered to have positive sentiment as the model will only consider "happy" => "happi".

For this reason we re-implemented the models, this time with negation handling. We tested this with Dataset1, but the model was still not able to detect "not happy" as a negative sentiment. After analysis, we realized that Dataset 1 is insufficient to train the model to handle negation. Thus, we opted for Dataset 2 with twice the positive and negative tweets, and after training the model , it was able to correctly identify "not happy"=>"NEG_happi" as a negative sentiment.

However,The accuracy of the models saw a significant drop, with logistic regression dropping to around 60% and KNN to 70%. With consideration to the algorithm used by the model for assessing sentiment and prediction, we deduced that the algorithm is inherently simple, meaning it does not work well with more complex data as well as varying tweets as in the case with Dataset 2 and handling negation.

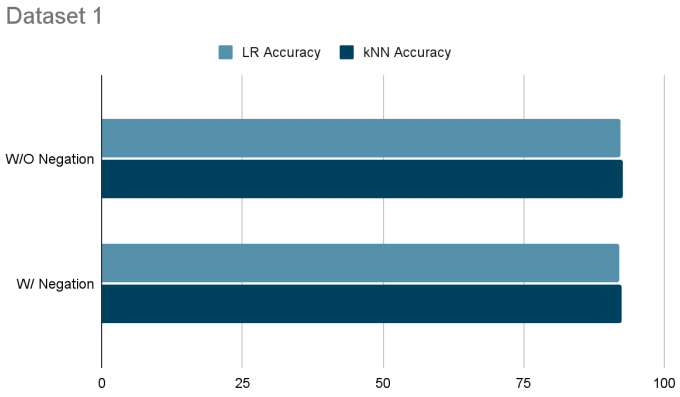
Diving further into how the algorithm functions, let's take the statement “ I do not like the weather today” which has an overall negative sentiment. With preprocessing extraction of features , the model will end up with the frequency list:

{ (Neg_like, 0) : 1, ('weather, 0) : 1,(today, 0) : 1}

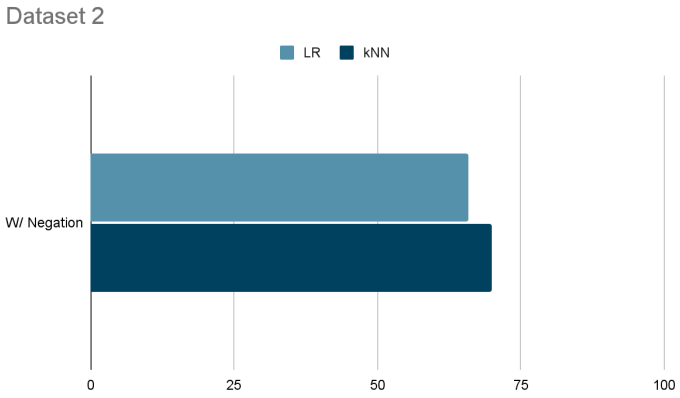
Therefore, associating words such as “weather” and “today” with negative sentiments, even though the words do not imply a specific sentiment without context. With that, it can be said that the model’s main strategy, of assigning sentiment according to the frequency of which words are associated with positive or negative sentiments, makes it highly susceptible to error, with results

varying significantly based on the data provided. This explains why the model performed with a lower accuracy on a bigger dataset with a more randomized selection.

In all scenarios discussed so far, the KNN algorithm performed slightly better than the logistic regression model as can be seen in the below statistics.



Small Data - No negation- High accuracy



Large Data - Negation- lower accuracy

V. Conclusion and Future Work

Overall, the model performed rather well on Dataset1. We were able to infer that this data was rather simple which is why it needed a simple model to match it. The analysis of these two models led us to learn how preprocessing plays a significant role in the model's effectiveness. Everything revolving around processing, tokenization, removing stopwords, and stemming helped the model focus on the data that mattered the most within the tweet itself. However, it still faced a challenge of ambiguity due to the negations it was not able to classify at first. We encountered several limitations in this implementation, most significantly being the handling of negations. On the other hand, kNN mainly faced issues due to the computational costs that can be rather expensive as kNN relies on distance calculations (e.g. euclidean distance). Another limitation was the short length of tweets that can be vague and could not be properly predicted. In terms of future work, we believe that in order to tackle the issue of the negation handling, we must augment the training data properly with examples that contain negations. This will help LR learn more meaningful weights. We may also integrate elements of deep learning and utilize different pre-trained models like the Hugging Face API which excels in sentiment analysis.