Deep Learning for Natural Language Processing

Project I

Chalkias Spyridon

November 2021

Contents

1 Vaccine Sentiment Classifier **2**
1.1 Task . . . 2
1.2 Data Preprocessing . . . 2
1.2.1 Data Cleaning . . . 2
1.2.2 Data Reduction . . . 3
1.2.3 Data Transformation . . . 3
1.3 Softmax Regression . . . 4
1.4 Learning Curve . . . 4
1.5 Hyperparameter tuning (opt.) . . . 4
1.6 Evaluating the models . . . 5
1.7 Results . . . 5
1.7.1 Choosing the best vectorizer . . . 5
1.7.2 Unigrams, Bigrams or both? . . . 6
1.7.3 Overall best performing model . . . 7

2 Sources **10**

1 Vaccine Sentiment Classifier

1.1 Task

The task is to perform sentiment analysis on a dataset consisting of tweets and be able to infer whether a new tweet is:

1. Neutral
2. Anti-vax
3. Pro-vax

1.2 Data Preprocessing

Real-world data tend to be incomplete, noisy, and inconsistent. This can lead to a poor quality of collected data and further to a low quality of models built on such data. In order to address these issues, Data Preprocessing provides operations which can organise the data into a proper form for better understanding in data mining process.
Some of the methods applied in order to prepare the data for analysis are:

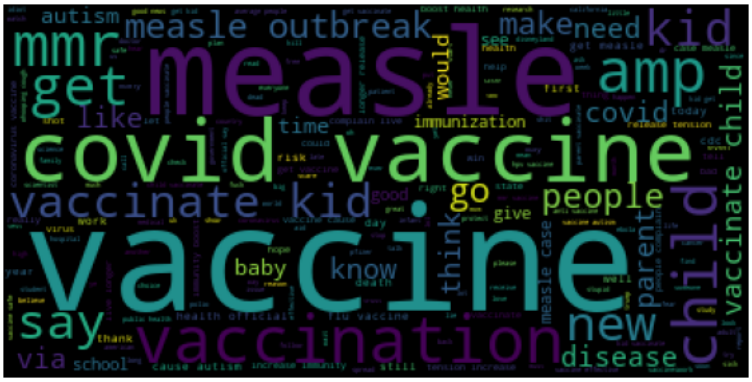
- Data Cleaning
- Data Reduction
- Data Transformation

Note that the dataset provided is imbalanced, since the *Neutral* and *Pro-Vax* tweets are far more than the *Anti-Vax* onces. However, this does not stop us from searching for the overall best performing model, since the **weighted f1-score** is able to represent a multiclass model's performance with confidence.

1.2.1 Data Cleaning

The cleaning process involved three (3) operations:

- Visualizing the tweets using **Wordclouds**, in order to expose the dataset's keywords. An indicative wordcloud is shown below:



- Checking whether there's need to drop data rows that contain *null* values. Luckily, the data had no such rows.
- Cleaning tweets from
  - URLs.
  - Noise contained in '< >'parenthesis.
  - The phrase 'RT' (meaning retweet).
  - '@' and the following twitter account name.
  - Any punctuation.
  - '\r\n' which is present in some strings.
  - Numbers, capitalization and white space.

  which are totally useless and add noise to the data.

1.2.2 Data Reduction

Data had a significant reduction process through *Tokenization and Lemmatization*, in order to group together the inflected forms of the words so they can be analysed in less items which are more independent and have greater semantic content. Also, all the stopwords were removed, so that data delivers more semantic content with as few words as possible. The stopwords container used was provided by *NLTK* library and the lemmatization process was supported by *spaCy* library.

1.2.3 Data Transformation

The data was transformed by testing three (3) representations:

1. *Bag of Words (BoW)* representation using sklearn's **CountVectorizer()**.

2. *Term Trequency-Inverse Document Frequency (TF-IDF)* representation using sklearn's **TfidfVectorizer()**.

3. *Hashing Vectorizer* representation using sklearn's **HashingVectorizer()**.

The parameters used in the aforementioned models were chosen experimentally by using sklearn's Grid Search.

Note that the computational procedure for finding the best vectorizer is not displayed in the exam's notebook, because it is considered optional; only the estimator's hyperparameter tuning is displayed because of its significant importance.

1.3 Softmax Regression

In order to construct the model, sklearn's *LogisticRegression()* function was used, with:

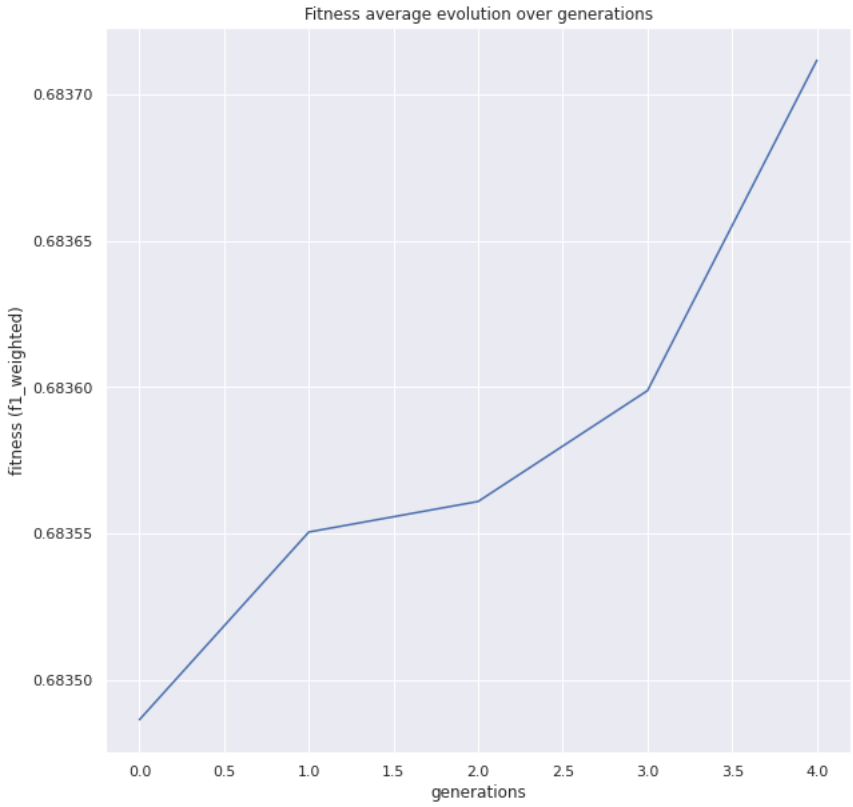
- Parameters chosen by running hyperparameter tuning, as shown in subsection **2.5**.
- **multi_class** parameter equal to *multinomial* in order for the model to support multiple classes.

1.4 Learning Curve

In order to visualize the model's progress, a learning curve was used as shown in this link. The code written in the previous link was modified in order to match to the needs of the model.

1.5 Hyperparameter tuning (opt.)

In order to fine-tune the model's hyperparameters and therefore produce the best performing model for the given dataset, the project uses evolutionary algorithms and more specifically, sklearn's *GASearchCV()* function. A Genetic algorithm is a metaheuristic inspired by the process of natural selection, and is used in optimization and search problems in general, and usually is based on a set of functions such as mutation, crossover, and selection. An indicative diagram showing the overall fitness of a model through 4 generations is shown below:



1.6 Evaluating the models

In order to evaluate the model's performance, sklearn's classification_report was used. As a result, the model was evaluated over its:

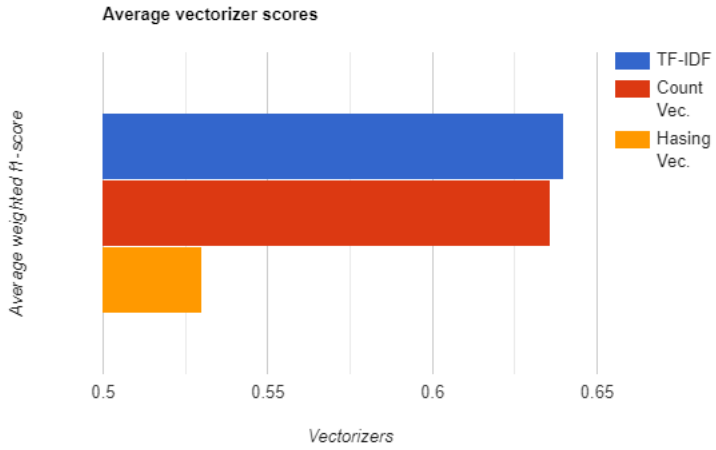
1. Precision
2. Recall
3. F1-Score
4. Support

1.7 Results

In this section, the multiple produced results are being diagrammatically shown and commented, in order to end up with the overall best tuned model.

1.7.1 Choosing the best vectorizer

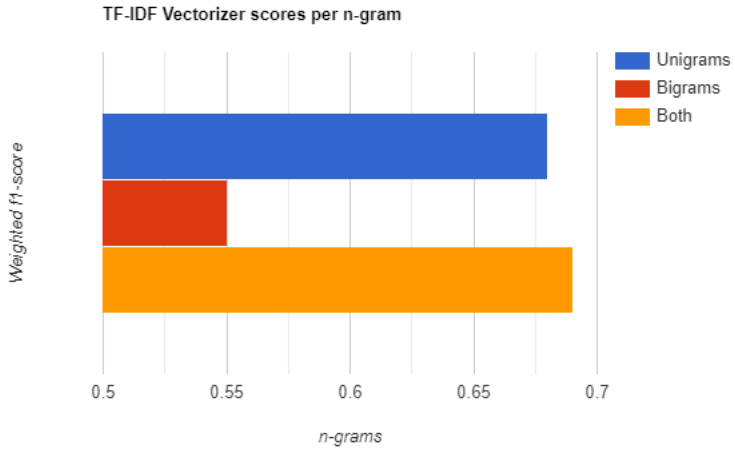
The diagrams below, depict the average weighted f1-score of each vectorizer, for all three (3) classes:

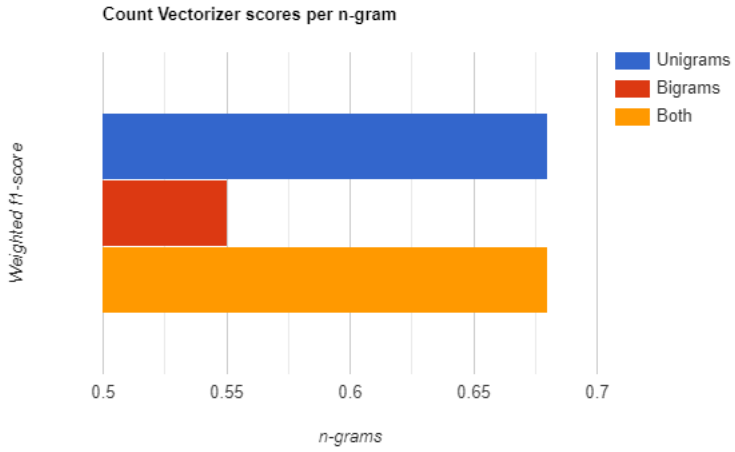


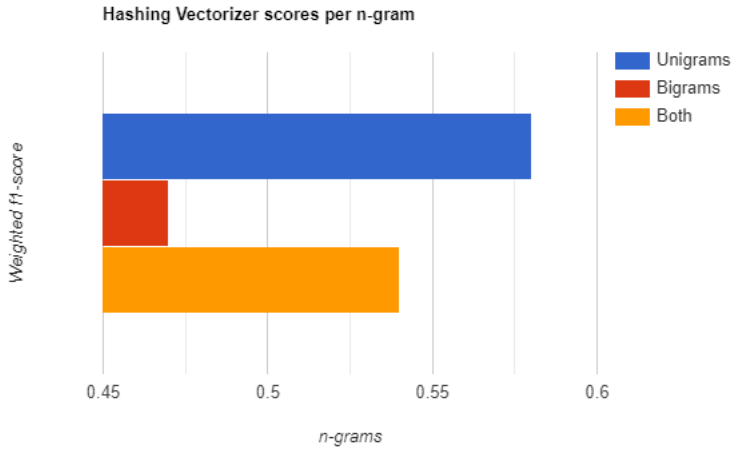
The overall best performing vectorizer seems to be *TfidfVectorizer*, by scoring an average score of 64%.

1.7.2 Unigrams, Bigrams or both?

The diagrams below, depict the weighted f1-score of each vectorizer, for all three (3) classes, for every pair of n-grams (unigrams, bigrams and both):





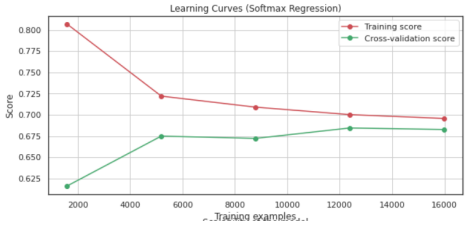


It is obvious that TfidfVectorizer fed with both unigams and bigrams, performed the best compared to the rest options, by producing a score of 69%.

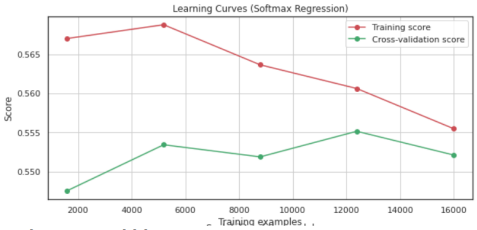
1.7.3 Overall best performing model

For every pair *(vectorizer, n-grams)*, a learning curve has been designed based on the **weighted f1-score**, in order to inspect the corresponding model's learning procedure and infer whether or not the model overfits or underfits.

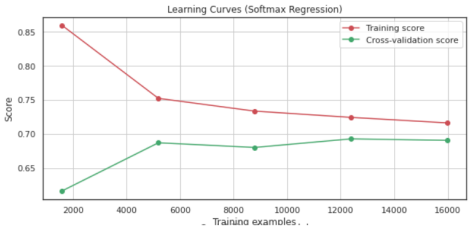
TF-IDF using only unigrams



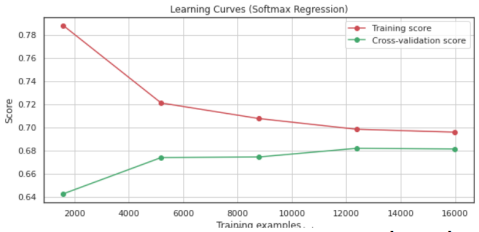
TF-IDF using only bigrams



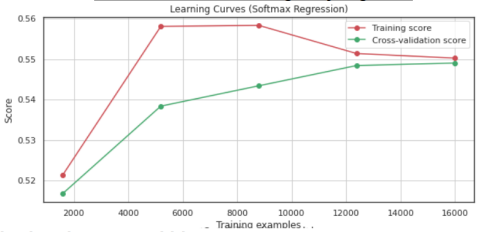
TF-IDF using both unigrams and bigrams



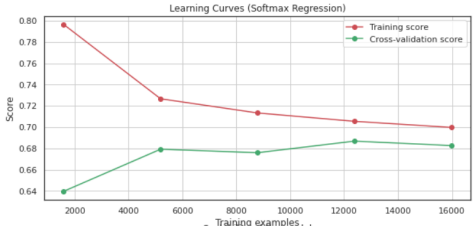
Count Vectorizer using only unigrams

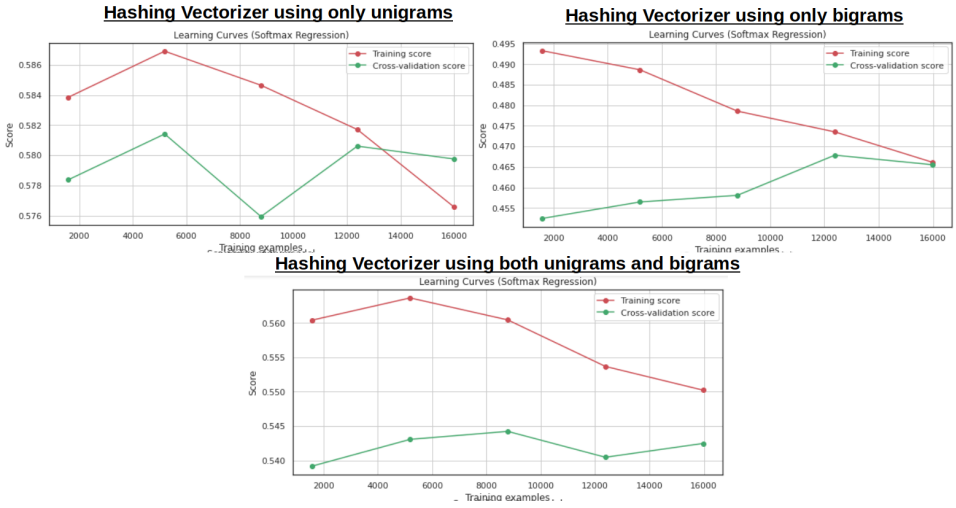


Count Vectorizer using only bigrams



Count Vectorizer using both unigrams and bigrams





As we can see, both Count and TD-IDF Vectorizers perform very well when working with:

- Only unigrams
- Both unigrams and bigrams

That is, because as shown in the diagrams, the training set **begins with a high score**, and gradually **reduces** it, while the validation set **begins with a fairly low score**, and gradually **increases** it, until the two aforementioned scores converge to values that have a small generalization gap. That way, the previous models do not overfit or underfit.

On the other hand, when using bigrams, the previous 2 vectorizers seem to be able to generalize but their respective curves are not as stable as in the previous representations and in addition they converge to a lower score.

Focusing on the Hashing Vectorizer, we can see that when working with exclusively bigrams, the training and learning curves converge to a point with very little, almost no generalization gap, meaning that the respective model does not overfit or underfit.

On the other hand, when working with exclusively unigrams, it seems that there is not enough and sufficient information to evaluate the ability of the model to generalize.

However, that does not happen when Hashing Vectorizer works both with unigrams and bigrams. If we were to decide whether the model overfit or underfit, we can obviously see that the training score is far better than the

testing score, which leads us to say that the model does overfit. Also, this is evident due to the fact that the learning curves have a huge generalization gap.

To conclude, after preprocessing and experimenting with a wide variety of representations and hyperparameter grids, the best performing model resulted to be a **TF-IDF** represented model, made up with **both unigams and bigrams**. It scored a weighted f1-score of **69%**.

2 Sources

- *Fake News Detection System* written **in collaboration with Charalampos Maraziaris - 1115201800105**
- Scikit Learn
- Stack Overflow
- Medium
- Towards Data Science