Sentiment Analysis Using Deep Learning

Priyal Nile
New York University
NY, USA
pan303@stern.nyu.edu

ABSTRACT

Sentiment Analysis is being used widely across all industrial sectors to understand the customer's opinion on the products and services industries are offering. In this project, we are exploring different machine learning models on US airline Data using Machine Learning and Deep Learning Technologies in the AI context. For the predictive model implementation, we are using RapidMiner and Python Programming Language. In the end, we are also dealing with imbalanced data using the Text augmentation technique, showing the comparative analysis of different types of Modelling techniques. In this final project, we are deploying Deep Learning models to build a Multi-Class classification model to predict the sentiment of consumers towards US airlines based on their reviews expressed in the form of tweets.

KEYWORDS

Sentiment Analysis, Deep Learning, RapidMiner, Python, Class Imbalance, Multi-Class Classification, Text Augmentation Technique, NLP

1. INTRODUCTION

There has been a tremendous amount of growth in user-generated data since the last decade. With cutting-edge technologies like machine learning, deep learning, and artificial intelligence, it is possible to extract useful information from past data and predict future data. Businesses are relying on their decision making based on data analysis and predictive modeling.

In this project, we are using the dataset of problems of each major U.S. airline in the form of Tweets. The customer sentiments are either Positive (1), Negative (-1), or Neutral (0).

For the Predictive Model Development purpose, we are using RapidMiner (earlier known as YALE, i.e., Yet Another Learning Environment(wiki)), a data science software platform developed by RapidMiner organization. RapidMiner uses a client/server model with server offered either on-premises or in public/private cloud. RapidMiner provides a GUI to design and execute analytical workflows. The workflows are known as 'processes' & those consist of 'Operators.'

In addition to this, we are also implementing the predictive model using Python Programming language and illustrating the results along with the high accuracy of the multi-class classification problem. We are implementing Standard Machine Learning Models like Naïve Bayes Model and Deep Learning Model.

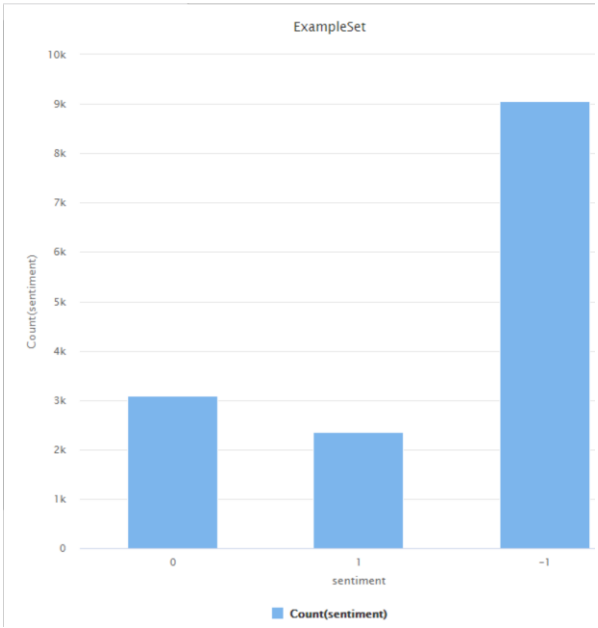
In the end, we are showing, which model is performing better in terms of accuracy, and the reason for model selection along with some additional screenshots of the RapidMiner Designs will be explained in a separate ReadMe Documentation.

2. SENTIMENT ANALYSIS USING RAPIDMINER

2.1 Data Exploration and Preparation:

For RapidMiner Implementation, we used the Preprocessed Dataset along with the '0' padded columns.

We used the retrieve operator to view the data & it's distribution in RapidMiner Visualization Module. This data is the input to our Design and the output if obtained through Performance Operator, as shown in the design diagrams.



We implemented the following four models using RapidMiner:

Traditional Machine Learning Models such as

- Naive Bayes Classifier and
- Multinomial Logistic Regression

And Deep learning Models such as

- Fully Connected Layers with 80/20 Train/Test Split
- Fully Connected Layers with Cross-Validation

Note: The Comparative analysis among these models have been tabulated in the next section.

2.2 Simple ML Naïve Bayes Model:

- It is a high bias low variance classifier that is used as a baseline model in many of the text classification problems. The best thing is that it gives better results even for a small dataset.
- Since we have the three types of output labels, i.e., +1, 0 & -1, this is a multi-class classification problem.
- We achieved that using setting the **Label 'sentiment' as Polynomial.**
- Also, we used Numerical to Polynomial Operator, which is used for changing the type of numeric attributes to polynomial type.
- There are no options to tune the model in the RapidMiner further. Based on the standard configuration, we got an accuracy of ***60.20*** using the Naïve Bayes Model.

2.3 Simple ML Baseline Model (Multinomial Logistic Regression)

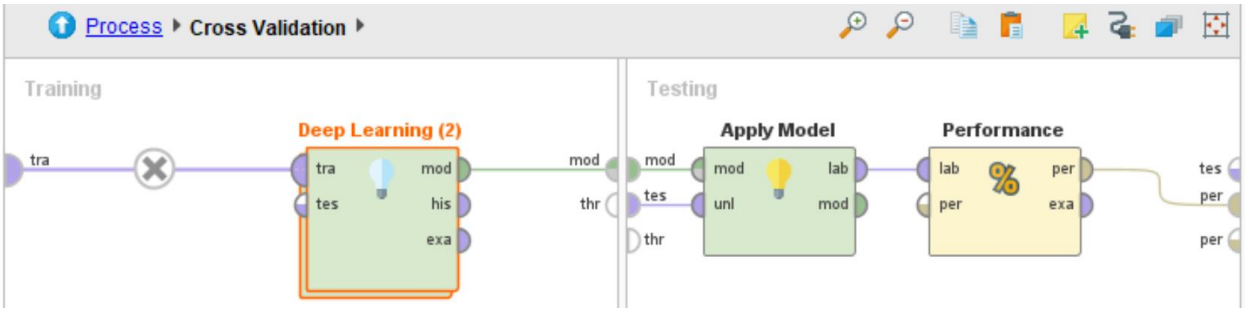
- RapidMiner does not support the direct implementation of the Logistic Regression for Multi-class Classification. So, we used the 'Polynomial to Binomial Classification' operator, which builds the polynomial classification model through the given binomial classification learner. It works in the One-Vs-All methodology.
- It works like a nested operator where we can stack up the Logistic Regression model.
- Using this model, we got an accuracy of ***61.69***

2.4 Deep Learning Fully Connected Network with 80/20 Train-Test Split:

- Using 4 Fully Connected Layers with neurons 16-8-4-3, activation functions as ReLU, Softmax and learning rate 0.001, we got an accuracy of ***63.30***
- Deep Learning works on the principle of Artificial Neural Networks, which is inspired by the biological neural network of humans. Hence, The accuracy is better than the traditional machine learning models.
- The deep learning model requires lots of training, and we set the epochs parameter to 10000, which gave the best accuracy. Since the learning rate is 0.001

2.5 Deep Learning Fully Connected Network with Cross-Validation: (Best Performance)

- To estimate the statistical performance of the learning model, we used Cross-Validation
- It is the best resampling procedure that is useful for limited data samples. It also helps to reduce overfitting.
- With K folds in cross-validation, K-1 chunks of data are the training dataset, and remaining chunks of data become the part of the testing dataset.
- Since we want to maintain 80-20 Train-Test Split, we used five-fold cross-validation, and it gave the best result for the 80% training dataset & 20% test dataset.
- Cross-Validation in the RapidMiner has two partitions separately for Training & Testing, as shown below:



With limited number of epochs=100, learning rate=0.01 & optimizer='Adam'

These optimizers are used as updaters in RapidMiner to adjust the weights and bias values to reduce the loss.

Adam Optimizer can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum. It uses the squared gradients to scale the learning rate like RMSprop, and it takes advantage of momentum by using the moving average of the gradient instead of the gradient itself like SGD with momentum.

We used the Loss Function 'Cross-Entropy Loss.'

The cross-entropy loss is useful in the case of two distributions, true-distribution, and the estimated distribution.

Mathematically,

$$-(y\log(p)+(1-y)\log(1-p))$$

If $M > 2$ (i.e. multiclass classification), we calculate a separate loss for each class label per observation and sum the result.

$$-\sum_{c=1}^{M} y_{o,c}\log(p_{o,c})$$

2.6 Comparison between Models:

M No	Model	Cross Validation (Folds)	Epochs	Learning Rate	Regularization	Fully Connected Layers (Neurons)	Activation Function	Time to run the model	Accuracy
1	Naïve Bayes Model	No (0)	Default	-	-	-	-	1 Second	60.20
2	Multinomial Logistic Regression	No (0)	Default	0.01	Yes	-		2 Seconds	61.69
3	DL: Fully Connected Layers with 80/20 Train/Test Split	No (0)	10000	0.001	Yes	4 (16;8;4;3)	3 Layers (ReLU); Last Layer (Softmax)	2 Minutes 22 Seconds	63.30
4	DL: Fully Connected Layers with Cross Validation	Yes(5)	1000	0.01	Yes	4 (32;16;8;3)	3 Layers (Sigmoid); Last Layer (Softmax)	3 Minutes 33 Seconds	64.64

- Model 4 gives the best accuracy of 64.64 in RapidMiner
- We achieved the best accuracy using Sigmoid as an activation function over ReLU. The possible reason includes
  - Sigmoid does not suffer through a Dead Neuron Problem like ReLU
  - Although, the sigmoid function may suffer due to the problem of vanishing gradient problem.
  - Since we have only two hidden layers in our model design, we did not observe the problem of vanishing gradients for Sigmoid function and got the best performance than ReLU.
- Computationally, the deep learning model with Cross-Validation takes the maximum amount of time.

In all the models, we have used Softmax as the activation function in the output layer. Softmax or normalized exponential function has Resulting values in a range between 0 and 1 while adding up to one. Hence this function can be used to map values to probability like values.

Note: We also implemented the models using CNN. However, there was no significant improvement observed for the CNN Models. The Maximum accuracy we got using CNN was **64.07**

Model Number	MiniBatch	Cross Validation (Folds)	Epochs	Learning Rate	Regularization for DL	Total CNN Hidden Layers	Total Fully Connected Layers	Activation Function	Time to run the model	Accuracy
1	No	Yes (5)	900	0.01	yes	0	4	Sigmoid	2min	63.99
2	No	Yes (5)	1200	0.01	yes	0	4(16;8;4;3)	Sigmoid	2min	64.41
3	No	Yes (5)	1200	0.01	Yes	0	4 (32;16;8;3)	Sigmoid	5 min	64.42
4	**No**	**Yes (5)**	**1000**	**0.01**	**Yes**	**0**	**4 (32;16;8;3)**	**Sigmoid**	**3 min 33 Seconds**	**64.64**
5	No	Yes (5)	950	0.01	Yes	0	4 (32;16;8;3)	Sigmoid	3min	64.56
6	**No**	**Yes (5)**	**1200**	**0.01**	**Yes**	**2**	**4 (32;32;16;3)**	**Sigmoid**	**3 min**	**63.69**
7	**No**	**Yes (5)**	**1200**	**0.01**	**Yes**	**2**	**4 (64;32;16;3)**	**Sigmoid**	**8 min**	**64.07**

To optimize the performance and fine-tune the dataset and parameters, we also implemented the project using Python Programming. We addressed the issue of Imbalanced Data, Implemented traditional machine learning models, and Deep Learning Models in Python.

3. SENTIMENT ANALYSIS USING PYTHON

3.1 Data Exploration and Preparation:

For the Python Implementation, we are using the Original Dataset without any '0' padded and top 30 words.

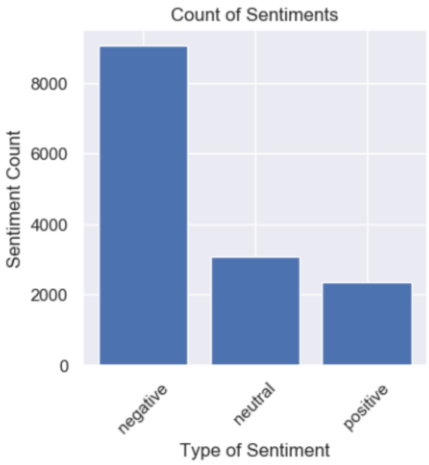
The US Airlines Dataset has two columns:

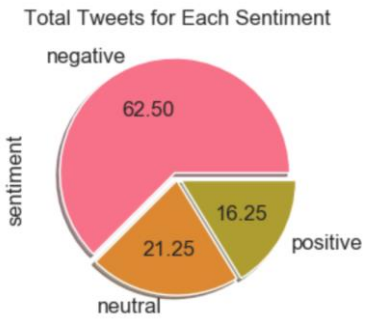
- ***'sentiment'*** => output label
  - +1 => Positive
  - -1 => Negative
  - 0 => Neutral
- ***'processed_text'*** => Features in the form of tweets

We are using some of the Natural Language Processing Toolkits for the implementation of this project. Since the efficiency of NLP would benefit from developing a better predictive model.

e.g., NLTK, NLP Augmenter

3.2 Original Data:



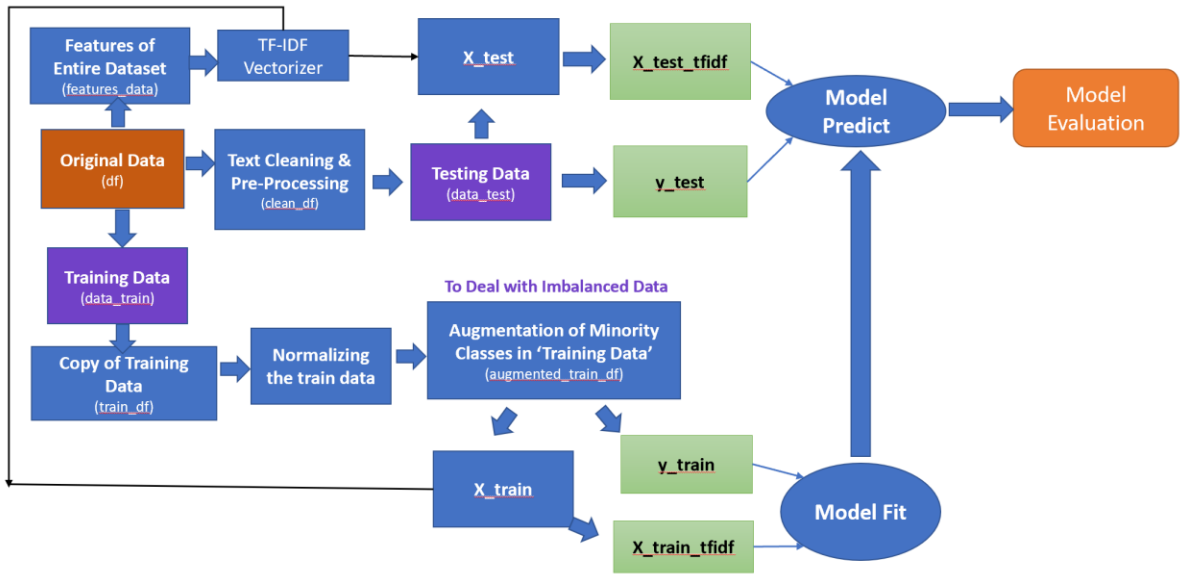


3.3 Class-wise Data Distribution:

The given dataset shows that 62.50 % of the data has Negative Sentiments. So, our predictive model may suffer from the class imbalance problem. To address the Class Imbalance Issue, we are doing oversampling of minority classes using text augmentation techniques. E.g., nlpaug. It will oversample the neutral & positive classes by applying random words variations with the same meanings using high cosine similarities.

We are also using the Lemmatization, which groups the infected forms of a word together to analyze them as a single item followed by Stemming for the text normalization.

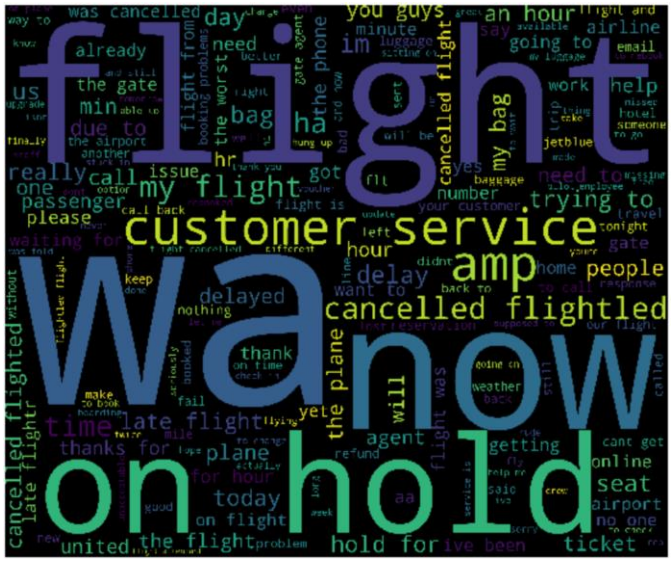
Note: To avoid data leakage, it is essential to do the oversampling only on the training dataset. To make sure that our model does not encounter the issue of data leakage, we followed our code development cycle, as illustrated in the self-explanatory diagram below:



The dataset consists of mainly two columns 'Processed_text' and 'sentiments.' To understand the positive, negative, and neutral sentiments in the given dataset, we incorporated the following word clouds for each output class.

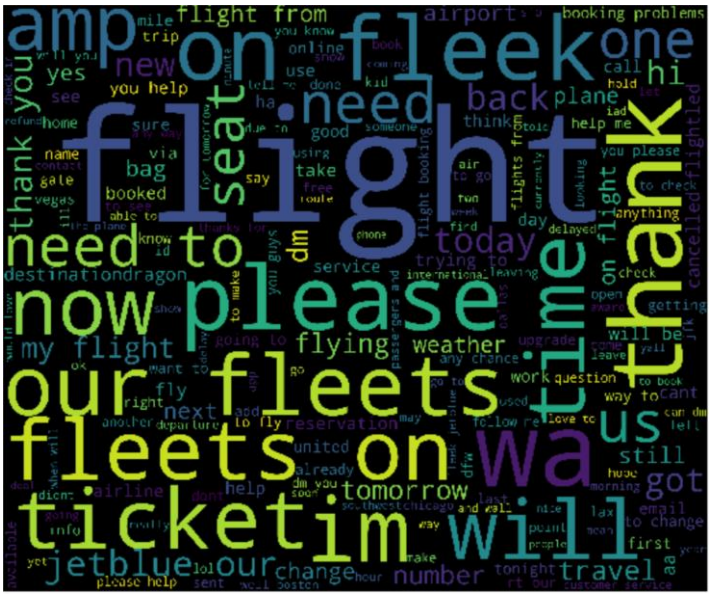
3.4 Exploring the Sentiments in the data:

- Negative Sentiments:



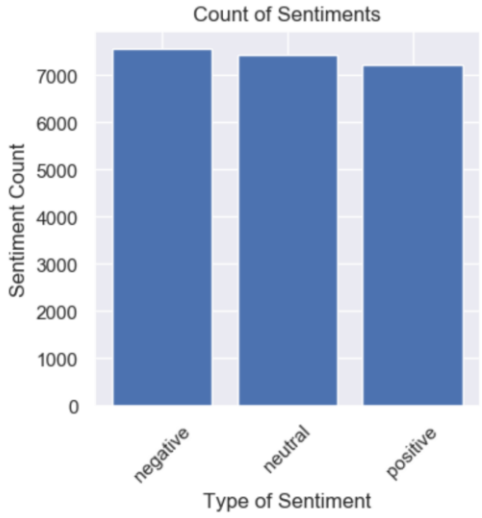
- Positive Sentiments:

- Neutral Sentiments:



3.5 Handling Class Imbalance Problem:

As explained earlier, the class imbalance problem needs to be addressed to achieve better performance for our model. We used NLP tools to manage this issue & oversampled the minority classes (neutral and positive). The following distribution shows that the data is balanced now. The Python Notebook attached with the submission document can drive the step by step process that we did to achieve this balanced data.



3.6 Baseline Model:

Bernoulli's Naïve Bayes & Logistic Regression using (Non-Binary Count Vectorizer) are our baseline models.

Mainly following four models have been implemented:

Model Number	Model	Accuracy	Run Time
1	Random Forest Classifier	70.81	1 Minute
2	Bernoulli's Naïve Bayes	76.13	3 Seconds
3	Non Binary Count Vectorizer	76.44	5 Seconds
4	Deep Learning	77.61	1 Minute 8 Seconds

Bernoulli's Naïve Bayes Model:

- Based on the Bayes' Theorem of Conditional probabilities, the Bernoulli's Naïve model has been implemented. We used binary count vectorizer for converting the collection of text documents to a matrix of token counts.
- We achieved an accuracy of 76.13%

Logistic Regression Using Non-Binary Count Vectorizer:

- It gives better results than Bernoulli's Naïve Bayes Model since here precise counts of words are considered.
- Using the Logistic Regression from Scikit learn with multi_class=auto, we achieved good accuracy of 76.85 %

Random Forest Classifier:

- We also used the Random Forest Classifier, which is an ensemble tree-based learning algorithm and which is a set of decision trees.
- It aggregates the votes from different decision trees.
- For the Random forest, we train a model with a relatively small number of samples and can get better results. However, here the accuracy we achieved is less than the Bernoulli's Model. Since Random forest can feel like a black box approach for a statistical modeler, we have very little control over what the model does.

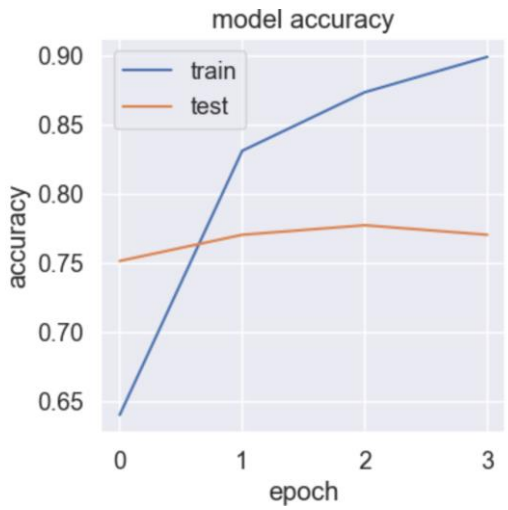
Deep Learning Model:

- As explained in the implementation of Deep Learning using RapidMiner, we also got high accuracy for the Deep Learning Model using Python.
- We used Tokenizer. In Python, tokenization refers to splitting up a larger body of text into smaller lines, words, or even creating words for a non-English language. The various tokenization functions in-built into the nltk module.
- We also have achieved Balanced data through augmented_traindf using NLP Toolkits.
- And we used one-hot encoding for the Output label 'sentiment.'
- We used embeddings with 32, 16, and 3 neurons along with 'Sigmoid' and 'softmax' as activation functions.
- As explained in RapidMiner Section, Sigmoid & Softmax gives the best results for the deep learning model associated with this airline dataset.

- Here, we used RMSProp Optimizer since it gave the best results. The gist of using RMSProp is that:
    - It Maintains a moving (discounted) average of the square of gradients
    - It Divides the gradient by the root of this average
    - This implementation of RMSprop uses plain momentum, not Nesterov momentum

3.7 Model Evaluation:

- The Train Accuracy is very high up to 90% with significantly less loss of 10%
- The Test Accuracy is up to 78% and loss of 22



4. CONCLUSIONS:

- **The maximum accuracy of 64.64** is obtained for Fully Connected Neural Networks with Cross-validation **using RapidMiner**
- **The maximum accuracy of 77.61** is obtained for Deep Learning Model **using Python** Programming Approach
- **Python Programming works better** for model development **than AutoML Solutions like RapidMiner**. Since we can parameterize the hyperparameters effectively in Python. However, RapidMiner is super easy to use with the drag-drop mechanism of the operators and does not require a programming background to implement machine learning models.
- **Deep Learning Model gives higher accuracy** than the **traditional machine learning models**. However, a significant difference can be observed if the number of data points are vast since Deep Learning Model works best when the data points are large enough.

5. ACKNOWLEDGMENT

We would like to thank Professor Alexander Tuzhilin sincerely' for his excellent guidance for the course 'Introduction to AI & It's Application in Business.' We would also like to thank the Teaching Fellow 'Pan Li' and 'Kefei Wu' for helping us to clear the project queries and RapidMiner issues in their office hours.

We would also like to thank the RapidMiner Community, which helped to solve some of the challenges that we faced during the implementation of this project.

6. REFERENCES

Some of the references that were useful while implementing this project:

https://community.rapidminer.com/discussion/26572/multi-class-labels
https://docs.rapidminer.com/latest/studio/operators/modeling/predictive/ensembles/polynomial_by_binomial_classification.html
https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html#cross-entropy
https://www.reddit.com/r/learnmachinelearning/comments/9kn8p0/what_are_the_advantages_of_sigmoid_over_relu_as_a/
https://pypi.org/project/nlpaug/
https://www.quora.com/What-are-the-advantages-and-disadvantages-for-a-random-forest-algorithm
https://www.tutorialspoint.com/python_text_processing/python_tokenization.htm