NOVA IMS Information Management School

Text Mining Project Handout 2023

MASTER DEGREE PROGRAM IN DATA SCIENCE AND ADVANCED ANALYTICS

Predicting Airbnb Unlisting

Group 16

Felix Gaber, number: 20221385

Josephine Lutter, number: 20220546

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

Index

1. Data Exploration 2

Non-Semantic Part 3

2. Preprocessing 3

3. Feature Engineering 3

4. Classification Models and Evaluation 4

Semantic Part 5

2. Preprocessing 5

3. Feature Engineering 5

A. Word Embeddings - Skip-Gram 5

B. Word Embeddings - GloVe 6

4. Classification Models and Evaluation 6

A. LSTM (Long Short-Term Memory) 6

B. DistilBERT 7

Annex 10

References 0

1. Data Exploration

Extra Work: Sentiment analysis and linguistic analysis.

The foundation of our analysis is built upon four integrated Excel files, which we further split into X_train, X_val, and X_test. After the data partitioning, we had 9996 observations in the X_train dataset and 2500 observations in X_val. First, we grouped the reviews from guests, the so-called *comments,* and associated them with the unique property index. The exploratory analysis reveals that the variable *comments* in the train dataset contains 3,215 missing values, while the validation dataset contains 814 missing values. Most unique values are registered in the variable *description* with 9486 values and least in the variable *host_about* containing 3810 values. No duplicate values are found in the datasets. The *unlisted* variable in our dataset exhibits an imbalance, with 7,244 occurrences of 0 and 2,752 occurrences of 1. This indicates an unbalanced distribution of the target variable (Figure 1). By analyzing the distribution of text lengths using **bar charts**, we can identify patterns and outliers in the *comments* variable that contain a more significant number of words than others (Figure 2).

To further gain insights into the vocabulary and extract repetitive keywords, the **frequency of words** in the corpus is analyzed and plotted for listed and unlisted properties. Considering that the text data will be transformed in the following preprocessing, the frequency distributions will change, ending up being more significant and valuable within the second exploratory analysis.

By performing **linguistic analysis**, multiple languages were identified (Figure 3). English and Portuguese are the most frequent languages in the *host_about* and *description* columns. The two most often languages within the *comments* column are English and French. This mismatch raises concerns about potential bias that may influence the property listing status.

In addition, **sentiment analysis** was conducted on the guest comments to assess overall property satisfaction to draw conclusions about the property listing status (Figure 4). This analysis plays a significant role in understanding the guest experience. Three lexicon-based approaches, AFINN, SentiWordNet, and NLTK, were utilized to assign sentiment labels ('positive', 'negative', and 'neutral') to each comment based on sentiment scores obtained from these approaches. It was observed that NLTK and AFINN yielded similar results in sentiment analysis, while SentiWordNet showed slight variations in its distribution. Most comments are associated with positive sentiments, indicating no distinct pattern and general high customer satisfaction.

The second exploratory analysis of the preprocessed data provides valuable insights into the text's characteristics. To enhance our understanding of the text data **frequency distributions, n-grams, and word clouds** were analyzed. Pyramid plots were generated, highlighting words with the highest relative frequency differences between listed and unlisted properties for each lemmatized variable. Pyramid plots with n-grams visualize the most frequent words and reveal differences in word usage between listed and unlisted properties for unigrams, bigrams, and trigrams (Figure 5). The word clouds further supported these findings, highlighting the distinctive word choices based on the property listing status.

After conducting an exploratory analysis, our project was divided into two parts: non-semantic and semantic analysis.

The **non-semantic** part concentrates on syntactic structures and patterns within the text data. It focuses on extracting structural and grammatical information rather than an in-depth understanding of the text's underlying meaning. For this part, we will use traditional machine learning algorithms. The **semantic** part considers the meaning and interpretation of language. This involves capturing the contextual information, semantic relationships, and deeper meanings by employing two-word embedding techniques and implementing advanced models such as Long Short-Term Memory (LSTM) and BERT (Bidirectional Encoder Representations from Transformers). The reader is kindly invited to note that the **extra points earned from the preprocessing work** should be accounted for and added in **both the semantic and non-semantic parts** of the subsequent evaluation.

Non-Semantic Part

2. Preprocessing

Extra Work: Reducing stop words based on language (all supported languages in the nltk library), Transform Treebank part-of-speech tags into WordNet POS tags to ensure accurate lemmatization of words based on their context, sentiment analysis on the text data to extract emotional content, which would be further utilized in the modeling process.

In the preparation, the unstructured text data is transformed into a more standardized format to extract valuable insights by enhancing accuracy. The corpus is transformed by converting the text to **lowercase** to ensure consistency and **stop words** which are common words that do not carry significant meaning. Further text cleaning involves **regex** to remove **punctuation** and **numerical data,** as well as the elimination of **tags** and **whitespace. Tokenization** consists in splitting the text into individual words or tokens. This facilitates subsequent analyses by examining text at the word level. In the next step, we performed **lemmatization** and **stemming**. Lemmatization reduces words to their base or root form while stemming truncates words by removing suffixes. Both techniques help to consolidate words with similar meanings and reduce the overall vocabulary size. These preprocessing steps prepare the text data for feature extraction techniques like bag-of-words and TF-IDF, making it more suitable for machine learning models.

3. Feature Engineering

Extra Work: Implementing four feature engineering techniques (BoW, TF-IDF, Word Embeddings: Glove, and Skip-Gram using Word2Vec).

In feature engineering, we combined the three relevant columns to implement a single data point in the text vectorization techniques. Text vectorization is a process that converts textual data into a numerical format, allowing machine learning algorithms to comprehend it.

Before combining the variables, we filled the missing values with a placeholder string called *missing*. Next, we applied two techniques for text vectorization, Bag-of-Words (BoW) and TF-IDF (Term Frequency-Inverse Document Frequency). **BoW** represents each document by counting the frequency of words it contains, disregarding their order. On the other hand, **TF-IDF** assigns weights to words based on their importance within the document and across the entire corpus. These techniques assist in capturing the significance of words and their impact on the text's overall meaning.

4. Classification Models and Evaluation

After performing vectorization on the text data, we trained and evaluated machine learning models that require BoW or TF-IDF representations. The five models tested were Naive Bayes, Random Forest, SVM, Logistic Regression, and KNN, using both TF-IDF and BoW vectorization techniques. The training data was used to fit each model, and predictions were made on the validation data. Classification reports and confusion matrices were generated for each model to assess performance.

The significance of the project lies in its ability to provide insights for resource planning and platform enhancement by generating a business forecast and analyzing market trends. Based on the business reasons and the project goal, the F1 score may be the best metric for evaluating the model.

The F1 score provides a balanced assessment of model performance as it calculates the harmonic mean of precision and recall. Because class balance is unbalanced in this project, the F1 score can help reduce potential bias from unbalanced classes, as opposed to using precision alone. A superior F1 score shows that our final model has good precision and recall, which can correctly predict unlisted and listed properties.

Among the models tested without hyperparameter tuning, the following two performed the best:

- Logistic Regression utilizing TF-IDF achieved an accuracy of 0.89. The weighted F1 score was 0.89, with an F1 score of 0.92 for class 0 and 0.80 for class 1.
- Random Forest utilizing BoW achieved an accuracy of 0.89. The weighted F1 score was 0.89, with an F1 score of 0.92 for class 0 and 0.81 for class 1.

Learning curves were plotted to enhance the visualization of Logistic Regression and Random Forest performance as the training set size increases. These curves provide valuable insights into how the models perform with varying amounts of training data, aiding in a better understanding of their behavior. Notably, the Random Forest model exhibited a distinct trend of overfitting, which could have also been supported through the BoW approach (Figure 6). We decided not to deepen the analysis by implementing hyperparameter tuning and instead shifted our focus to more advanced algorithms such as LSTM and Bert.

Semantic Part

2. Preprocessing

Extra Work: BeautifulSoup library to parse and clean up the HTML markup in the text data.

Within the second part of the notebook, we implemented Skip-Gram embedding to conduct natural language processing (NLP) tasks on text data while reducing the dimensionality of the representations. At first, we initiated a coarser cleaning procedure to prepare the text data. **Stop words** were not used in the cleaning process, ensuring they did not hinder subsequent analysis. In addition, we transformed the text to **lowercase**, eliminated all **non-word characters, tags**, and **digits**, and split the text into individual words using a **tokenizer**. Empty values were replaced with the term *missing*. We purposefully did not perform deeper cleaning at this stage. Instead, we focused on more complex models recognizing semantic relationships between words.

3. Feature Engineering

A. Word Embeddings - Skip-Gram

After performing text cleaning, we implemented Skip-Gram embedding to conduct NLP tasks on the cleaned text data while reducing the dimensionality of the word representations. The Skip-Gram model is particularly effective in capturing the meaning and associations of words within their local context. It achieves this by predicting the context words based on a given target word. This approach enables the model to learn high-quality word representations encapsulating semantic relationships between words.

To train the Skip-Gram model, we utilized the Word2Vec method available in the *gensim* library. This method is well-suited for learning word embeddings from large text corpora. The parameters were chosen as follows:

- vector_size=100: The size of the vector generated by the model for each word was set to 100. This means that each word is embedded into a 100-dimensional space.
- window=5: The window size determines how many words on each side of a given word are considered when determining the word's context. In this case, five words were considered on each side of the given word.
- min_count=1: Words that occur less than once in the entire text corpus are ignored. Since min_count was set to 1, all words were considered.
- sg=1: The training was performed using the Skip-Gram approach (sg stands for skip-gram).

After training the Word2Vec models, they were visualized to gain insights into the distribution of words in the embedded space. The code of the interactive 3D visualizations (Figure 7) is attached notebook. Nevertheless, the outputs of this cell needed to be cleared due to the maximum file size of 50MB in Moodle.

To summarize, in this part of the project, text processing was performed using the Skip-Gram Word2Vec model to embed words into a vector space and capture their semantic similarities. The results were visualized to provide an understanding of the structure and relationships in the word space. Modeling was done using the following word embedding approach GloVe.

B. Word Embeddings - GloVe

In this part of the project, we employed pre-trained GloVe (Global Vectors for Word Representation) models to vectorize the text data. We chose to utilize GloVe models instead of Skip-Gram due to their training on larger text datasets. These models were trained on extensive text corpora, making them a valuable resource for our later project stages.

Two GloVe models were loaded from the *gensim* library: 'glove-twitter-100', a model trained on Twitter data, and 'glove-wiki-gigaword-100', a model trained on Wikipedia and Gigaword data.

We created separate corpora for training and validation for each text column (in this case, *host_about*, *description*, *comments*, and *all*). Subsequently, each of the corpora was vectorized using both GloVe models. To accomplish this, we utilized the *corpus2vec* function, which iterated through each document in the corpus and each word in each document. If a word was present in the key indices of the respective pre-trained model, its vector was retrieved from the model and added to the document's vectors. The resulting vectors were then saved for later use. For each combination of text column and GloVe model, a corresponding set of document vectors was created for both training and validation data.

In summary, in this part of the project, the power of pre-trained GloVe models was utilized to transform the text data into vector representations. Various document vectors were created using two different GloVe models, which can be used for further analysis and modeling.

4. Classification Models and Evaluation

Extra Work: By meticulously applying multiple transformer models (DistilBERT base multilingual cased) and utilizing transformed-based embeddings across diverse corpora, we successfully generated 38 distinct transformed-based embeddings and corresponding models.

A. LSTM (Long Short-Term Memory)

To effectively work with the data, the initial step involved converting it into a format compatible with TensorFlow. Subsequently, the corpus underwent sequential processing to ascertain the lengths of the sequences and perform data padding accordingly. This vital padding step standardized the sequence lengths, which is essential for effectively training the LSTM model.

The next step involved constructing a bidirectional LSTM model using TensorFlow's Keras API. The model architecture includes an input layer, a masking layer, an LSTM layer, and a densely connected output layer. The input layer is responsible for accepting the prepared text data, while the masking layer ensures that only relevant information is passed to the LSTM. Being bidirectional, the LSTM layer is capable of learning and retaining information from both the past and future within the sequences.

Finally, the output layer utilized a sigmoid activation function and comprised two neurons, aligning with our binary classification objective.

Given memory limitations, we decided to train and validate the model solely on the *description* column. The comparison of the different performances is visualized in Figure 8.

For the model trained on the **Twitter** dataset, we achieved a binary accuracy of 79.19% and an F1 score of 79.19% after 20 training epochs. The validation loss was 0.5529, the validation accuracy was 73.92%, and the validation F1 score was 74.57%. When training on the **Wikipedia** dataset, the model reached a binary accuracy of 78.33% and an F1 score of 78.33% after 20 epochs. The validation loss was 0.5599, the validation accuracy was 73.20%, and the validation F1 score was 73.66%.
Our results demonstrate that the chosen model can extract relevant information from the *description* column, leading to reasonable accuracy in classification. In conclusion, the model trained on the Twitter data performed slightly better than the one trained on Wikipedia data. Subsequently, we decided to move on to more promising models like DistilBert and stopped further investigation at this point.

B. DistilBERT

A. Train each column separately (*host_about*, *description*, *comments*)

During the implementation of Bert, we followed a systematic approach to determine the best training procedure for the DistilBert model. At first, we trained separate models using different datasets, namely the *host_about*, *description*, and *comments* columns.

We used consistent training arguments for each training step, including five training epochs, a training batch size of 16 per device, and an evaluation batch size of 64 per device. Additionally, we set a learning rate of 0.00002. After completing the training, we evaluated and compared the performance of each model. The results are the following:

- The model trained on the *host_about* data achieved an accuracy of 79%. The weighted F1 score was 0.77, with an F1 score of 0.86 for class 0 and 0.55 for class 1.
- The model trained on the *description* data achieved an accuracy of 77%. The weighted F1 score was 0.76, with an F1 score of 0.85 for class 0 and 0.55 for class 1.
- The model trained on the *comments* data achieved an accuracy of 92%. However, this model exhibited a low weighted F1 score of 0.90. It obtained a high F1 score of 0.96 for class 0 but a meager F1 score of only 0.04 for class 1.

Considering these results, the model trained on the *comments* data performed the best in accuracy. However, it showed significant weakness in classifying class 1.

B. Comparing different combinations

We explored an additional procedure for implementation, in which we used a hierarchical approach of a combination of the comment and description columns for classification.

We devised a new approach to address the dataset's imbalance and enhance the overall performance. Threshold shifting was not a viable solution in our case since it would have led to significant prediction losses for the opposing class. Instead, we leveraged the existence of missing values in the *comments* column to improve the classification. Our strategy involved utilizing the comments for classification in observations where they are present while using the *comments* column for the remaining observations. To facilitate this, we have introduced a new column called *text_to_use*, which contains either the comments or the description, depending on the observation.

Considering that the maximum token limit of 512 was often exceeded, particularly in the comments, we deliberated whether it would be beneficial to randomly select a subset, considering the different languages. Ultimately, we concluded that utilizing the most recent comments would be more appropriate, as they should have the most significant impact on whether a property is listed in the next quarter. However, we opted against this approach due to the absence of timestamps for the comments and our utilization of multilingual DistilBert, which should not be significantly affected by language differences or only exhibit minimal discrepancies. Therefore, we simply utilized the first 512 tokens from each observation in every column. This approach allowed us to handle the token limit constraint and ensured consistency across the dataset.

With this method, we further explored different hyperparameters to determine the most effective approach. Specifically, we investigated various learning rates and train/eval batch sizes. The learning rates considered were 0.0001, 0.0005, and 0.001, while the train batch sizes were 16 and 32, and the eval batch sizes were 32 and 64.

The results of the experiments indicate that the choice of learning rate significantly influenced the F1 score, while the batch sizes had little impact. Among the different learning rates tested (0.0001, 0.0005, and 0.001), the best performance was consistently achieved with a learning rate of 0.0001. On the other hand, the precision, recall, and F1 score were consistently low, around 0.357800 for learning rates of 0.0005 and 0.001, indicating poor performance across all batch sizes tested.

C. Testing different parameters

Drawing from the findings that lower learning rates yield superior results, we executed another experiment using learning rates smaller than previously tried. We narrowed our focus to the following range of learning rates for the new experiment (0.00001, 0.0001, 0.0005, 0.001), while maintaining the training batch sizes at 16 and 32, and evaluation batch sizes at 32 and 64.

In this experiment, we observed that the highest learning rate (0.0001) consistently produced the most favorable outcomes. Based on this finding, we focused our investigation on learning rates closer to this optimal value of 0.0001. As a result, we conducted tests using the following learning rates 0.00025, 0.000125, 0.0001, 0.0000875, and 0.000075, with each test consisting of 5 epochs. The training batch size was kept constant at 32, while the evaluation batch size was set to 64.

The results indicated that learning rates of 0.00025 and 0.000125 achieved the highest global F1 scores of 0.842332 and 0.842566, respectively. However, considering the potential impact of varying the number of epochs on the score, we decided to further evaluate the learning rate of 0.0001 with a higher number of epochs.

Surprisingly, in the subsequent run using 20 epochs, a per-device training batch size of 32, a per-device evaluation batch size of 64, and a learning rate of 0.0001, the model achieved an F1 score of 0.92 for class 0 and 0.77 for class 1, which was the best result obtained. The second-best model achieved an F1 score of 0.91 for class 0 and 0.77 for class 1, using a learning rate of 0.000125. Consequently, we concluded that training the model on the entire training dataset using the parameter of the best model and predicting on the test dataset would be the most appropriate final course of action. While we initially contemplated incorporating another classifier from the non-semantic analysis (specifically Random Forest in conjunction with Bag of Words), we ultimately decided against it due to the pronounced overfitting observed with the Random Forest model.

Finally, we obtained a mean of the classes of 0.25198 on our test data, having 350 instances for the listing property status 1 and 1039 for the property listing status 0 (Figure 9).

To summarize, the project's objective was to leverage NLP techniques and classification models to predict the likelihood of a property listed on Airbnb being unlisted in the next quarter. To achieve this, we utilized actual Airbnb property descriptions, host descriptions, and guest comments. The significance of the project lies in its ability to provide insights for resource planning and platform enhancement by generating a business forecast and analyzing market trends. To evaluate the performance of our models in alignment with the project's objectives, we utilized the F1 score as the evaluation metric. After conducting experiments, the DistilBert model obtained the best result with a training duration of 20 epochs, a per-device training batch size of 32, a per-device evaluation batch size of 64, and a learning rate of 0.0001. The model achieved an F1 score of 0.92 for class 0 (still listed) and 0.77 for class 1 (unlisted).

Annex

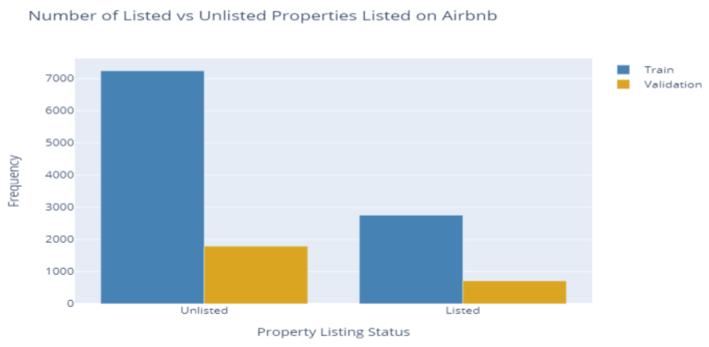

Figure 1: Representation of the unbalanced dataset

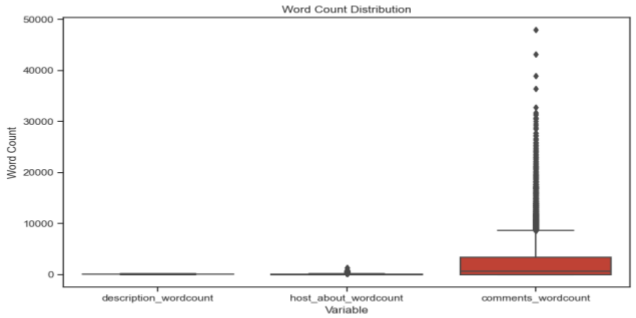

Figure 2: Word Count Distribution

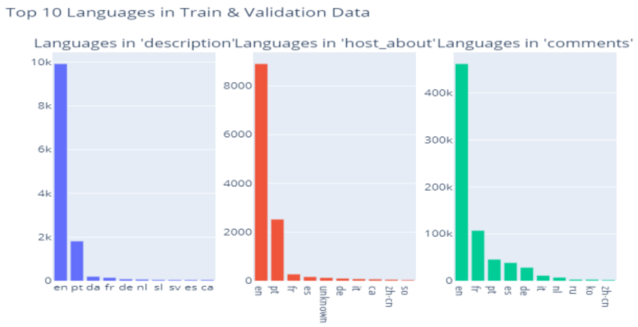

Figure 3: Top 10 languages in Train and Validation Data

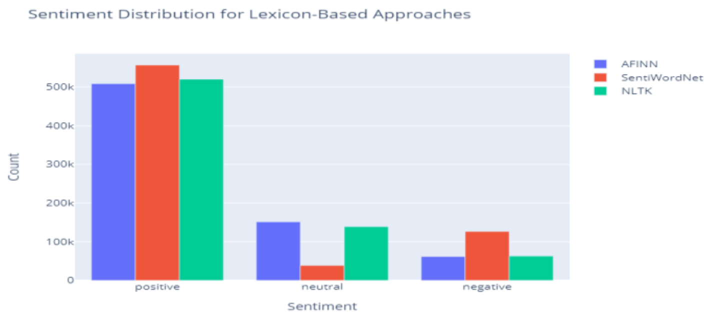

Figure 4: Sentiment Distribution for Lexicon-Based Approaches

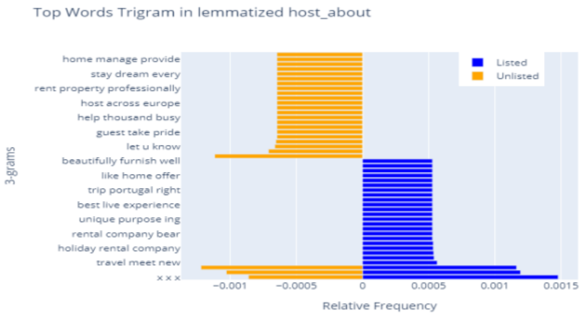

Figure 5: Pyramid plots with n-grams visualize the most frequent words and reveal differences in word usage

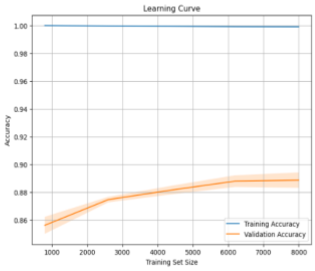

Figure 6: Learning Curve of Random Forest

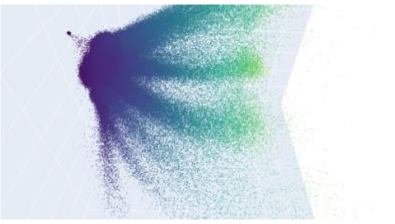

Figure 7: 3D Visualization of the semantic relationship in the comments column

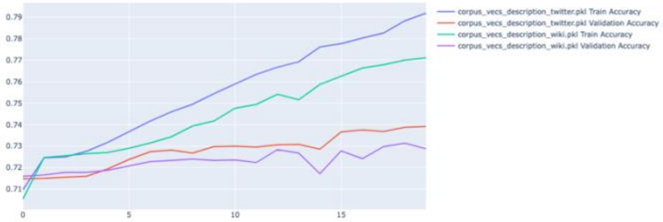

Figure 8: Twitter and Wiki GloVe F1 Comparison over the epochs of the LSTM

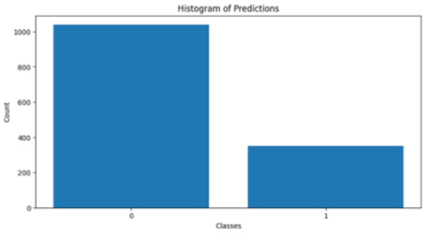

Figure 9: Histogram of the final prediction

References

1. Towards Data Science. (2021, Jan 28). Introduction to Word Embedding and Word2Vec. Retrieved from https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa
2. Maklin, C. (2020, Jun 8). Word2Vec Skip-Gram. Retrieved from https://medium.com/@corymaklin/word2vec-skip-gram-904775613b4c
3. Towards Data Science. (2019, July 11). Skip-Gram: NLP Context Words Prediction Algorithm. Retrieved from https://towardsdatascience.com/skip-gram-nlp-context-words-prediction-algorithm-5bbf34f84e0c#:~:text=Skip%2Dgram%20is%20one%20of,It's%20reverse%20of%20CBOW%20algorithm
4. Data Driven Investor. (2019, Mar 9). Word2Vec Skip-Gram Model Explained. Retrieved from https://medium.datadriveninvestor.com/word2vec-skip-gram-model-explained-383fa6ddc4ae
5. Japneet Singh Chawla. (2018, Aug 22). Customer Review Analytics using Text Mining. Retrieved from
6. https://medium.com/analytics-vidhya/customer-review-analytics-using-text-mining-cd1e17d6ee4e
7. Nidhaloff. (2020, Jul 1). How to Translate Text with Python. Retrieved from https://medium.com/analytics-vidhya/how-to-translate-text-with-python-9d203139dcf5
8. Towards Data Science. (2021, Jan 9). How to Detect and Translate Languages for NLP Project. Retrieved from https://towardsdatascience.com/how-to-detect-and-translate-languages-for-nlp-project-dfd52af0c3b5
9. Sebastian Theile, through Analytics Vidhya. (2019, Sept 7). Basics of Using Pre-trained GloVe Vectors in Python. Retrieved from https://medium.com/analytics-vidhya/basics-of-using-pre-trained-glove-vectors-in-python-d38905f356db
10. Japneet Singh Chawla. (2018, Apr 24). Word Vectorization Using GloVe. Retrieved from https://medium.com/analytics-vidhya/word-vectorization-using-glove-76919685ee0b