Content

Overview

This report provides an analysis and evaluation of text classification using data mining techniques such as collection, preprocessing, cleaning, conversion. For success in assignment was used Sklean, Keras, NLTK, Pandas, Numpy, Matplotlib, MinMaxHash libraries. Report consists of two main tasks related to categorization and duplication detection. For completing tasks related to text classification was analysis models like Support Vector Machine (SVM), Random Forest (RF), Logistic Regression (LR) with features Bag of Words (BoG) and Singular Value Decomposition (SVD) and evaluated Accuracy, Precision, Recall and F-Measure parameters. Tasks related to duplication detection required to familiarize with the Locality Sensitive Hashing (LSH) family and with two types of similarity Jaccard and Cosine with different permutations and K parameters.

1. Text Classification

Text classification also known as *text categorization* or *text tagging* – the task of assigning a set of predefined categories to open-ended. Text classifiers can be used to organize, structure, and categorize pretty much any kind of text – from documents, medical studies and files, and all over the web. For example, new articles can be organized by topics; support tickets can be organized by urgency; chat conversations can be organized by language; brand mentions can be organized by sentiment, and so on.

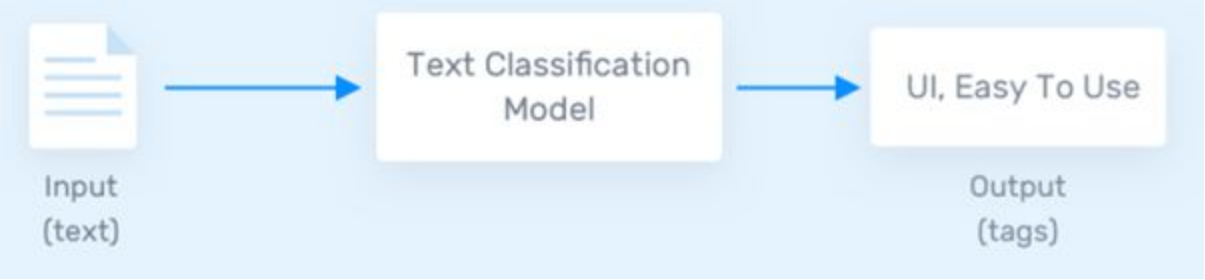

Figure 1. Text Classification process

Text classification can be done two different ways: manual or automatic classification. In the former, a human annotator interprets the content of text and categorizes it accordingly. This method can usually provide quality results but it's time-consuming and expensive. The latter applies Machine Learning, Natural Language Processing (NLP), and other AI-guided techniques to automatically classify text in a faster, more cost-effective, and more accurate manner. There are many approaches to automatic NLP text classification, which fall into three types of systems such as *Rule-based systems, Machine learning-based systems and Hybrid systems.*

Rule-based systems

Rule-based approaches classify text into organized groups by using a set of handcrafted linguistic rules. These rules instruct the system to use semantically relevant elements of a text to identify relevant categories based on its content. Each rule consists of an antecedent or pattern and a predicted category.

Machine learning based systems

Instead of relying on manually crafted rules, machine learning text classification learns to make classifications based on past observations. By using pre-labelled examples as training data, machine learning algorithms can learn the different associations between pieces of text, and that a particular output (i.e., tags) is expected for a particular input (i.e., text). A "tag" is the predetermined classification or category that any given text could fall into.

Hybrid Systems

Hybrid systems combine a machine learning-trained base classifier with a rule-based system, used to further improve the results. These hybrid systems can be easily fine-tuned by adding specific rules for those conflicting tags that haven't been correctly modelled by the base classifier.

In our assignment we used Machine Learning Approach for Text Classification. The Text Classification processing process using Machine Learning consists of feature extraction and a method is used to transform text into a numerical representation in the form of vectors. To accomplish this task we used the Bag of Words approach. Then, the machine learning algorithm is fed with training data that consists of pairs of feature sets – vectors for each text example and tags to produce a classification model as shown in figure 2.

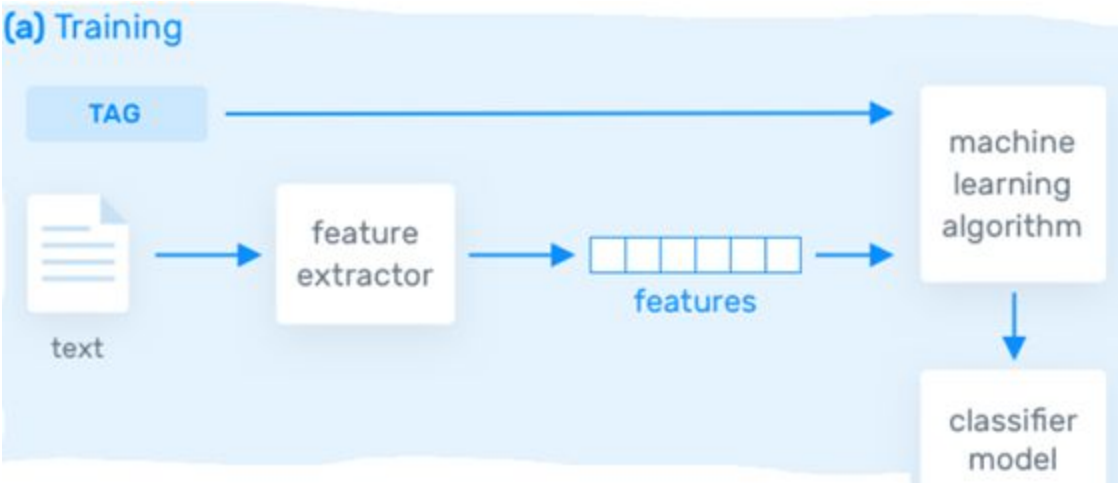

Figure 2. Training of Machine Learning Model for Text Classification

Once it's trained with enough training samples, the machine learning model can begin to make accurate predictions. Figure 3 presents the same feature extractor is used to transform unseen text to feature sets which can be fed into the classification model to get predictions.

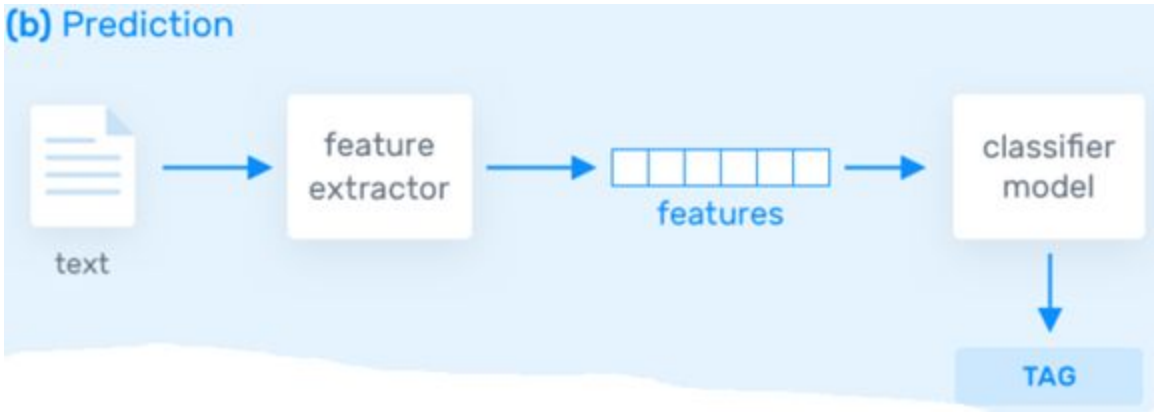

Figure 3. Prediction of Machine Learning Model for Text Classification

1.1 Word Cloud

Word clouds, also known as tag clouds or term clouds, are an emerging technology for analysing qualitative data. It's recommended to use word clouds as dashboards because they provide easy, quick, and meaningful analysis of qualitative data. Word clouds summarize or analyse text data and provide meaningful interpretations through text size and colour.

The assignment was given a "train_set.csv" file which contains news articles and each article belongs to one of categories "Business", "Entertainment", "Health" or "Technology". In paragraph 1.a the task is to build a Word Cloud and for completing this task we created a function "make_word_cloud" which takes as a parameter input text, the image to overlay words on its shape, list of Stop Words and name of output word cloud. More precisely, Stop Word is a set of commonly used words in any language, not just English. It removes the words that are very commonly used in a given language, so we can focus on the important words

instead. In our code we implemented three lists of words from NLTK library, Sklearn library and we found a list of stop words from the web page of Universite de Neuchatel. We compared these three Stop Words and decided to use the Stop Words list from Universite de Neuchatel. From figure 4 we can see that Universite de Neuchatel contains Stop Words which has the highest 571.

Stop Wrods

```
# Create stopword list:
nltk_sw = stopwords.words('english')
sklearn_stop_words = sk_sw.ENGLISH_STOP_WORDS
```

```
#Load from text file stop word for better performance
#the stop words file was taking from http://members.unine.ch/jacques.savoy/clef/englishST.txt
sw_list = []
with open('StopWords.txt', 'r') as f:
    [sw_list.append(word) for line in f for word in line.split()]

print("Lenghs of Stop Words from file ", len(sw_list))
print("Lenghs of Stop Words from NLTK Library ", len(nltk_sw))
print("Lenghs of Stop Words from Sklearn Library ", len(sklearn_stop_words))
```

```
Lenghs of Stop Words from file  571
Lenghs of Stop Words from NLTK Library  179
Lenghs of Stop Words from Sklearn Library  318
```

Figure 4. Comparison of Stop Words lists

For “make_word_cloud” function as an input text we filtered “train_set.csv” file and took text of each category then feeded to function for creating Word Cloud for “Business”, “Entertainment”, “Health” and “Technology”. However, before filtering we Preprocessed text to a more convenient way for future use in Machine Learning models. For Preprocessing we firstly removed empty rows then using NLTK library features such as tagging and “WordNetLemmatizer” for words. We sorted each word as “Noun”, “Verb” etc and gave each word token. Additionally, was removed all punctuations, numbers and converted all words to lowercase. After Text Processing data was filtered depending on each category and created Word Cloud.

Figure 5. Word Cloud for Technology category

Figure 5 illustrates the most commonly used words such as “apple”, “google”, “device” etc, in news articles for the Technology category. Same process was applied for all other categories.

Figure 6. Word Cloud for Entertainment category

Figure 6 shows Entertainment category used most common words from news articles. Words “photo”, “video”, “show” etc describes Entertainment category.

Figure 7. Word Cloud for Business and Health category

As can be seen from figure 7 Business and Health category Word Cloud was created and words such as “market”, “company”, “bank”, “percent” common for Business category, “health”, “patient”, “study”, “cancer” for Health category. Perhaps we should also point out the fact that the shape of Word Cloud by default is not like in our work. We decided to add some other shapes to make our work more unique and these changes can be modified by purpose. One must admit that the list of Stop Words can be expanded for better efficiency.

1.2 Classification Methods

Machine learning text classification models are working on the principle of learning to make classifications based on past observations. Trained data for models are pre-labeled and pre-processed with usage of feature extraction techniques. Machine learning algorithms can learn the various links between a piece of input text, and that a particular label is expected for a particular input.

The very first step of data classification is the data preprocessing part. The accuracy of the model is directly dependent on the quality of the training data. For the case of the text processing, a raw data (strings), where each sample represented as a text should be transformed into numerical representation in the form of a vector. In our work we will apply one of the most frequent approaches - Bag of Words (**BoW**) and Singular Value Decomposition (**SVD**).

1.2.1 Bag of Words (BoW)

A bag-of-words is a representation of text that describes the occurrence of words within a sample of a dataset. A output of BoW feature extraction model consisted of two things:

a. A vocabulary of known words.
b. A measure of the presence of known words.

It is called a “bag” of words, because any information regarding the order of words in the sample is discarded. The model is only taking into account the presence of known words in the document, not position of word in the document.

In our work, we used the ready function CountVectorizer() from the sklearn library to perform BoW, which converts a collection of text documents to a matrix of token counts. The input at the CountVectorizer is the preprocessed, tokenized data.

1.2.2 Singular Value Decomposition (SVD)

SVD is widely used both in the calculation of matrix operations, such as matrix inverse, but also as a data reduction method. The working principle of SVD is an input matrix decomposition method for reducing a matrix to its constituent parts in order to make subsequent matrix calculations easier.

$$A = U * Sigma * V^T$$

Where A is the real m x n matrix that we wish to decompose, U is an m x m matrix, Sigma (often represented by the uppercase Greek letter Sigma) is an m x n diagonal matrix, and V^T is the transpose of an n x n matrix where T is a superscript.

In our case, we used the output of the BoW function as an input to the SVD. SVD was performed by the TruncatedSVD() function from sklearn library. TruncatedSVD is preferable to work with sparse matrices, where it performs linear dimensionality reduction by means of truncated singular value decomposition. We set following parameters as follows:

a. N_components = 5, desired dimensionality of output data.
b. N_iter = 7, number of iterations for randomizer SVD solver.
c. Random_state = 42.

1.2.3 Classification models.

Due to the requirements the following models were used to achieve text classification: Support Vector Machine (SVM) and Random Forest. We used both BoW and SVD as input data for both models, and we decided to use the Logistic

Regression model with BoW. Before usage of the above mentioned models, the labels were encoded, this is done to transform Categorical data of string type in the data set into numerical values which the model can understand.

Raw Labels	Encoded Labels
Business	0
Entertainment	1
Health	2
Technology	3

Table 1. Label encoding

The dataset will be split into two data sets, Training and Test. The training data set will be used to fit the model and the predictions will be performed on the test data set.This can be done through the *train_test_split* from the *sklearn* library. The Training Data will have 70% of the corpus and Test data will have the remaining 30% as we have set the parameter *test_size=0.3*[6] .

SVM

Support-Vector machines construct a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class [7]. We considered C-Support Vector Classification model from sklearn library with linear kernel.

Random Forest

Random forests create decision trees on randomly selected data samples, get prediction from each tree and select the best solution by means of voting. It also provides a pretty good indicator of the feature importance. Random forests is considered as a highly accurate and robust method because of the number of

decision trees participating in the process.It does not suffer from the overfitting problem. The main reason is that it takes the average of all the predictions, which cancels out the biases. But Random forests are slow in generating predictions because they have multiple decision trees. Whenever it makes a prediction, all the trees in the forest have to make a prediction for the same given input and then perform voting on it. This whole process is time-consuming[9]. We set the number of decision trees equal to 100.

Logistic Regression

The **logistic regression classifier** uses the weighted combination of the input features and passes them through a sigmoid function. Sigmoid function transforms any real number input, to a number between 0 and 1.Logistic regression is fast and relatively uncomplicated, and it's convenient for you to interpret the results. Although it's essentially a method for binary classification, it can also be applied to multiclass problems like our task.

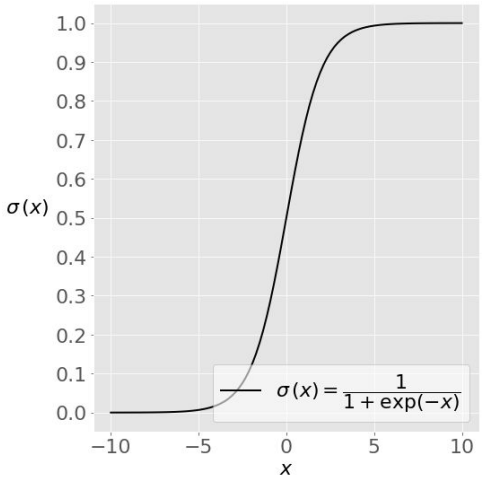

Figure 8. Sigmoid function

We put max_iter = 10000 that defines the maximum number of iterations by the solver during model fitting. Our solver is kept default, which is lbfgs(Limited memory BFGS).

1.2.4 Applying 5-folds and Evaluating results

K-folds cross-validation is commonly used in applied machine learning to compare and select a model for a given predictive modeling problem because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods. Cross-validation is a resampling procedure used to evaluate machine learning models. By the requirement we put k = 5 and used pipeline() for each model, where by default the input data was output of BoW and for models with SVD we included additional steps to perform SVD on the sparse matrix. For each model five epochs are performed, where one fold, which is 20% of the input dataset taken as a test dataset and rest as a train.

The evaluation table is consisted of the following metrics:

a. Accuracy - is the fraction of predictions our model got right.
b. Precision - the number of true positives divided by the number of true positives plus the number of false positives.
c. Recall - the number of true positives divided by the number of true positives plus the number of false negatives.
d. F - measure - is the harmonic mean of precision and recall

Statistic Measure	SVM (BoW)	Random Forest (BoW)	SVM (SVD)	Random Forest (SVD)	LR (BoW)
Accuracy	0.947135	0.939174	0.824366	0.845029	0.956214
Precision	0.942882	0.940146	0.817898	0.832741	0.953492
Recall	0.940326	0.926827	0.792269	0.82223	0.95015
F-Measure	0.941523	0.933147	0.803244	0.827028	0.951793

Table 2. Results of 5-folds CV for 5 models

From the table one can conclude that Our method which is Logistic Regression with a Bag of Words, provided the highest scores on all metrics. However, the combination of SVM, Random Forest model with SVD showed poor

results in comparison with the combination of the same models with Bag of Words. It can be explained that the matrix reduction is negatively affected on the total evaluation of the models. That is why we decided to apply only Bag of Words in our method and successfully overcame the nearest competitor SVM in all statistical metrics.

1.2.4 Predictions on the test set

For the test dataset without labels we decided to use LR with the BoW model to make predictions, since our model performed the highest scores. Before the usage of the model, the test set was gone through pre-processing steps, which included the same pre-processing calculations excluding undesirable symbols, words, tokenization and feature-extractions with usage BoW. After uploading the upload file to Kaggle, we got quite a high accuracy score which equals 93%. One can conclude that the better results can be achieved with optimizing the parameters of both BoW function as well as LR model, which will be done furthermore.

2. Nearest Neighbour Search and Duplicate Detection

Nearest Neighbor Search (NNS) is used to find objects that are similar to each other. The idea is that given an input, NNS finds the objects in the database that are similar to the input. As a simple example, database of news articles and target to retrieve news similar to given query then perform a nearest neighbors search for given input query against all articles in the database and return the top 10 results. In NNS, distance function is important. It decides how similar or dissimilar the objects are. Lower distance values indicate the objects are similar whereas higher indicate they are dissimilar. For example, if two objects have distance 0 then they are identical. In our assignment we used Cosine and Jaccard similarity to define distance functions. We need to represent our entities in a vector space in order to use nearest neighbours so we used TF-IDF for feature extraction techniques to convert text as vectors.

Various solutions to the NNS problem have been proposed. The quality and usefulness of the algorithms are determined by the time complexity of queries as well as the space complexity of any search data structures that must be maintained. The informal observation usually referred to as the curse of dimensionality states

that there is no general-purpose exact solution for NNS in high-dimensional Euclidean space using polynomial preprocessing and polylogarithmic search time. In our task was to apply Locality Sensitive Hashing to our data using Cosine and Jaccard Similarity.

Duplicate detection is a serious problem in many applications, including customer relationship management, personal information management. Whenever we want to consider a duplicate detection from a dataset which mostly comes under Data mining.

Duplicate detection is the process of identifying various representations of the same real-world objective in an information source. Nowadays duplicate detection methods need to process larger datasets in a shorter time. maintaining the quality of a dataset becomes increasingly difficult.Duplicate detection problem has two aspects. First one is multiple representations that are usually not the same but contain differences, such as changed addresses, misspellings, or missing values. This makes it difficult to detect these duplicates. Second duplicate detection is a very expensive operation, as it requires the comparison of every possible pair of duplicates typically complex to calculate similarity of every pair of records.

2.1. De-Duplication with Locality Sensitive Hashing

This task was done on the dataset consisting of Quora questions, where the main goal is to find the number of duplicate questions from the test-set that exist in the train-set. From the first glance we found out that train-datasets have plenty of empty samples, which were excluded in the dataset preprocessing part. We considered Cosine Similarity and Jaccard Similarity to solve the question, by the task requirement we chose a threshold of similarity equal 80%.

2.1.1 Jaccard Similarity

The method relies on two simple concepts, character shingles and the Jaccard similarity. ‘Shingle’ is a term that comes from roofing and refers to partially overlapping pieces of clay, stone, wood, asphalt or some such bits of roofing material. The idea for character shingles is similar, create a document

representation of consecutive overlapping character n-grams from each document. Notice that punctuation and whitespace are all part of the process. This representation preserves word order, to some extent, and allows comparing documents based on the sets of character shingles. The similarity of those documents can then simply be defined as the Jaccard similarity of the two sets of shingles; the number of elements (shingles) they have in common as a proportion of the combined size of the two sets, or the size of the intersection divided by the size of the union.

To perform Exact Jaccard Similarity we defined two functions: get_shingles() and jaccard(). The function get_shingles() will return us the set of lists, where each sample is given as a set of arrays with size of equal to shingle size equal to 5. And
The jaccard() function works as a division of the number of the intersection of the two sets to the total number of the sets.

2.1.2 Cosine Similarity

Cosine similarity measures the similarity between two vectors of an inner product space. It is measured by the cosine of the angle between two vectors and determines whether two vectors are pointing in roughly the same direction. It is often used to measure document similarity in text analysis.

$$sim(x, y) \ \frac{x*y}{||x||||y||}$$

where $||x||$ is the Euclidean norm of vector x=(x1,x2,…,xp) , defined as $\sqrt{x_1^2 + x_2^2 + ...x_n^2}$. Conceptually, it is the length of the vector. Similarly, $||y||$ is the Euclidean norm of vector y.

Both train and test dataset were gone though pre-processing part by extracting features using TfidfVectorizer()

For our function get_similarity_items we included the parameter topn, which returns the n-highest similarities to the certain sample. By default the 1st highest similar is the sample itself, then the function returns the other samples. We set topn = 3, considering that the test set could contain 2 copies of the certain sample.

2.1.3 Locality Sensitive Hashing (LSH)

LSH -Jaccard

Locality Sensitive Hashing (LSH) is a generic hashing technique that aims, as the name suggests, to preserve the local relations of the data while significantly reducing the dimensionality of the dataset. It can be used for computing the Jaccard similarities of elements as well as computing the cosine similarity depending on exactly which hashing function is selected, more on this later.

LSH is a slightly strange hashing technique as it tries to *ensure* hash collisions for similar items, something that hashing algorithms usually try to avoid. The overall aim is to reduce the number of comparisons needed to find similar items, the hash collisions come in handy here as similar documents have a high probability of having the same hash value. The hash values can be treated as an address for a bucket that contains likely duplicates, this reduces the number of comparisons needed as only the documents contained in a bucket, not every other document, need to be looked at to find the real duplicates. The probability of a hash collision for a minhash is exactly the Jaccard similarity of two sets. This can be seen by considering the two sets of shingles as a matrix.

In our work firstly created a LSH Index to generate a MinHashLSH table for each query and an encoded train set inside of that table. After that using build-in function of LSH compared each query MinHashLSH tables putting threshold equal to 0.8 and counted number of duplicated. Same process was applied to calculate LSH-Jaccard with different numbers of permutations 16,32,64, respectively. Build Time is time of creating LSH Index and generating MinHashLSH tables, Query Time is comparing time and Total time is sum of Build Time and Query Time.

LSH-Cosine

The idea behind LSH is to throw down random vectors. Where all points above the line will be considered as positive scores and assigned to bin 1 and points below the line will be considered as negative scores and assigned to bin 0. So we are translating the sign of the scores into a binary index. Since we worked on the random vectors equal to 16, we have 16 bits to represent the bin index.The first bit is given by the sign of the dot product between the first random vector and the document's TF-IDF vector, and so on.All documents that obtain exactly this vector will be assigned to the same bin. One further preprocessing step we'll perform is to convert this resulting bin into integer representation. This makes it convenient for us to refer to individual bins.We can repeat the identical operation on all documents in our dataset and compute the corresponding bin. We'll again leverage matrix operations so that no explicit loop is needed.Given the integer bin

indices for the documents, we would curate the list of document IDs that belong to each bin. Since a list is to be maintained for each unique bin index, a dictionary of lists is used.

After generating our LSH model, recall that during the background section, given a product's TF-IDF vector representation, we were able to find its similar product using standard cosine similarity. Here, we will look at these similar products' bins to see if the result matches intuition. The idea behind LSH is that similar data points will tend to fall into nearby bins.

Decide on the search radius r. This will determine the number of different bits between the two vectors. This parameter is given in the table 3 as K.

2.1.4 Results

Unfortunately, our measurements for Exact Jaccard, Exact Cosine and Cosine-LSH models were done only on the first 10000 samples of the train set and 1000 first samples of the test set. We could make measurements on the whole dataset due to lack of computational sources. Online services like Google colab and Kaggle notebook were also used, but even 3 days of computing on the Google colab did not succeed on our code. We are still trying to optimize our code and experimenting.

Type	Build Time	Query Time	Total Time	# Duplicates	Parameters
Exact-Cosine	0.62	14.45	15.08	2	-
Exact-Jaccard	0.34	373.84	374.1	2	-
LSH-Cosine	0.079	33.79	33.86	1	K = 2
LSH-Cosine	0.079	75.42	75.5	1	K = 3
LSH-Cosine	0.079	208.53	208.61	2	K = 4
LSH-Cosine	0.079	525.83	525.91	2	K = 5
LSH-Cosine	0.079	1130.02	1130.1	2	K = 6
LSH-Cosine	0.079	2335.46	2335.53	2	K = 7
LSH-Cosine	0.079	4671.07	4671.15	2	K = 8

LSH-Cosine	0.079	9342.31	9342.39	2	K = 9
LSH-Cosine	0.079	18686.2	18686.28	2	K = 10
LSH-Jaccard	223.61	2.12	225.74	617	Perm = 16
LSH-Jaccard	277.75	2.73	280.49	500	Perm = 32
LSH-Jaccard	422.53	3.79	426.32	541	Perm = 64

Table 3. Results of the Deduplication with LSH

From table 3. We can conclude that, the highest score was achieved with LSH-Jaccard model, where the #duplicates was 617 and the query time was the lowest. The worst model in terms of the time was Exact-Jaccard. Additionally, the total time for LSH-Cosine model is directly proportional to K parameter, the increase of the search radius involves comparing higher numbers of bins. Nevertheless, the proposed solution for the LSH-Cosine, Exact Cosine, Exact Jaccard showed operability only for the small datasets, while it was impossible to execute the whole dataset using available computational sources.

There two similar questions inside the first 10000 train set and 1000 test set found by models are same and provided below:

Train ID 2903	What hotel in Shillong Hill-station would be safe for unmarried couples, without the harassment of police, hotel staff, and moral police?
Test ID 316	What hotel in Mussoorie Hill-station would be safe for unmarried couples, without the harassment of police, hotel staff, and moral police?
Train ID 9700	What is the colour of water?
Test ID 299	What is colour of water?

As we can see, the difference between both sample is equal to one word.

2.2. Question Detection – ultimately identical

Question detection is a method for detecting identical or semi-identical questions to automatically answer questions posed by humans in a natural language which is used widely in marketing and customer services. Question detection refers to use of natural language processing (NLP), text analysis to systematically identify, extract, quantify and study affective states and subjective information. Part 2.b of the assignment is to make an identical question detection using the given "train_set.csv" file for training models and "test_set.csv" for predicting question detection.

First of all, we started with extracting data and some features such as length of questions, common words which are shared between question 1 and question 2, not frequently used words.

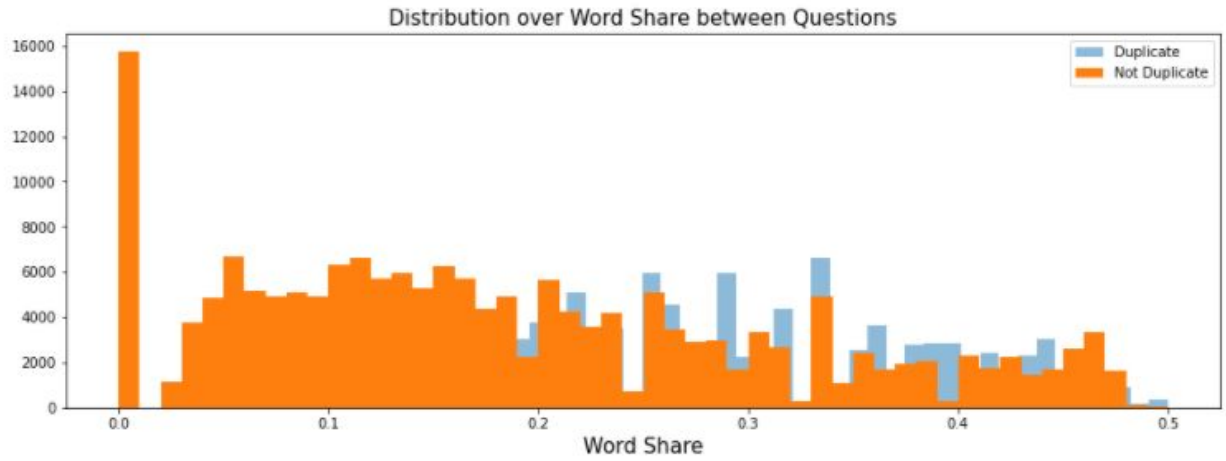

Figure 9. Common words - shared between two questions.

Term Frequency — Inverse Document Frequency (TFIDF) is a technique to quantify a word in documents, it generally computes a weight to each word which signifies the importance of the word in the document and corpus. This method is a widely used technique in Information Retrieval and Text Mining. Term Frequency (TF) measures the frequency of words in a document. This highly depends on the length of the document and the generality of word, for example a very common word such as "was" can appear multiple times in a document. but if we take two documents one which has 100 words and the other which have 10,000 words. There is a high probability that the common word such as "was" can be present more in the 10,000 worded document. But we cannot say that the longer document is more important than the shorter document. For this exact reason, we perform a

normalization on the frequency value. we divide the frequency with the total number of words in the document. Inverse Document Frequency (IDF) is the inverse of the document frequency which measures the informativeness. By calculating IDF we will give to words like stop words "is" gives very low value. This finally gives a relative weightage $idf(t) = N/df$. Cosine Similarity was used with TF-IDF to generate a metric that says how related are two documents by looking at the angle instead of magnitude. Cosine Similarity briefly is a measure that calculates the cosine of the angle between two vectors or two documents on the Vector Space[13].

$$cos\theta = \frac{\bar{a} * \bar{b}}{||\bar{a}|| * ||\bar{b}||}$$

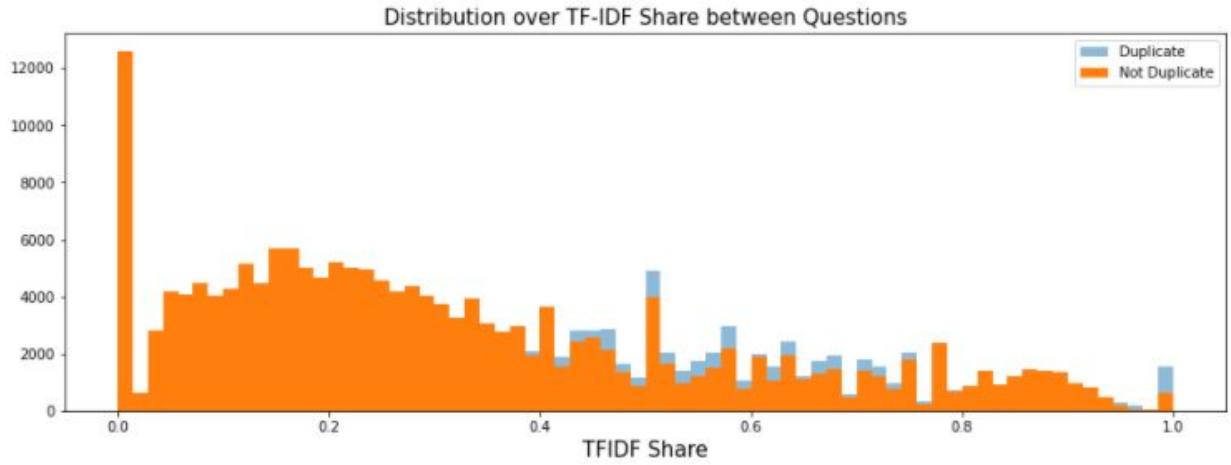

Figure 10. Not frequently used words between two question

For our task in 2.b we chose Random Forest and Logistic Regression models and also we applied 5th Cross Validation. We used Scaled Normalizer to better perform when numerical input variables are scaled to a standard range 0-1, which is the range for floating-point values where it has the most precision. Data for trains splitted as 70% as a train and 30% as test data for models. After splitting we run our models for classification. The results of classification is presented in Table 3 which describes such parameters as Precision, Recall, F-Measure and Accuracy.

Method	Precision	Recall	F-Measure	Accuracy
Random Forest	0.69	0.69	0.69	0.71

Logistic Regression	0.66	0.66	0.66	0.66

Table 3. Results of Classification

Last part of 2.b was choosing the best model and making a test set predictions of test "duplicate_predictions.csv" files which don't have an “IsDuplicate” label, "duplicate_predictions.csv" file contains Id and Predicted value.

Conclusion

To sum up, we have learned how to work with text datasets and understood the importance of the pre-processing part for each task. Moreover, we learned how to work with most common libraries in python and methods of feature extraction. Also, we are still researching the optimization of our results, by experimenting on the hyperparameters of the models. For the Deduplication task we found out that our proposed model is not compatible for high volume datasets, however we could understand the principle of the Jaccard and Cosine Similarities. The results can be upgraded by usage of fully-connected neural networks.

References

1. https://monkeylearn.com/text-classification/
2. https://journals.lww.com/nursingmanagement/fulltext/2018/10000/using_word_clouds_to_analyze_qualitative_data_in.10.aspx
3. https://machinelearningmastery.com/gentle-introduction-bag-words-model/
4. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
5. https://machinelearningmastery.com/singular-value-decomposition-for-machine-learning/
6. https://medium.com/@bedigunjit/simple-guide-to-text-classification-nlp-using-svm-and-naive-bayes-with-python-421db3a72d34
7. https://en.wikipedia.org/wiki/Support-vector_machine
8. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
9. https://www.datacamp.com/community/tutorials/random-forests-classifier-python
10. https://en.wikipedia.org/wiki/Question_answering
11. https://en.wikipedia.org/wiki/Sentiment_analysis
12. https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/
13. https://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/