PDAN8411 POE

Programming for data analytics

Arjun Dayar (st10300985)

2023

Table of Contents

[Text processing 2](#_Toc139374313)

[Why the chosen data set is appropriate for analysis 3](#_Toc139374314)

[Analysis that will be conducted 3](#_Toc139374315)

[Process used in data analysis. 4](#_Toc139374316)

[Step 1: 5](#_Toc139374317)

[Step 2: 5](#_Toc139374318)

[Step 3: 5](#_Toc139374319)

[Step 4: 5](#_Toc139374320)

[Step 5: 6](#_Toc139374321)

[Step 6: 6](#_Toc139374322)

[Step 7: 7](#_Toc139374323)

# Text processing

Text processing is an automated approach to analysing and manipulating unstructured text data, aiming to extract valuable insights and information. It involves employing various techniques and technologies, such as natural language processing (NLP) and machine learning, to handle and interpret textual data effectively (Gruetzemacher, 2022). By transforming unstructured text into a structured format, text processing enables machines to comprehend and make sense of the information contained within (Gruetzemacher, 2022).

The primary objective of text processing is to uncover meaningful patterns, sentiments, and trends hidden within large volumes of unstructured text data. By analysing and processing textual information, companies and organizations can gain valuable insights for decision-making, customer feedback analysis, and information retrieval purposes (Gruetzemacher, 2022). Text processing serves as a foundation for applications such as language translation, sentiment analysis, spam filtering, and information extraction (Gruetzemacher, 2022). These applications deal with massive amounts of text and require sophisticated backend processes to transform text into a format that algorithms can effectively analyse (Gruetzemacher, 2022).

Text processing techniques heavily rely on natural language processing (NLP) and machine learning algorithms. NLP enables machines to understand and process human language by utilizing computational linguistics and machine learning algorithms. It encompasses tasks such as tokenization, part-of-speech tagging, named entity recognition, sentiment analysis, and topic modelling (Nadkarni, Ohno-Machado and Chapman, 2011). Additionally, machine learning algorithms play a vital role in text processing by extracting meaningful information from textual data, classifying documents, performing sentiment analysis, and automating text-related tasks (Nadkarni, Ohno-Machado and Chapman, 2011).

Before analysis, text data often requires cleaning and pre-processing steps to ensure accurate and reliable results. Text cleaning involves removing punctuation, stop words, and irrelevant characters, while pre-processing encompasses tasks such as text normalization and handling misspellings or abbreviations. These steps ensure that the text is in a suitable format for analysis, improving the quality of insights extracted from the data (Gruetzemacher, 2022).

Text processing finds applications across various domains and industries. Language translation, facilitated by text processing techniques, enables the translation of text from one language to another, benefiting international communication and understanding. Sentiment analysis utilizes text processing to determine the sentiment or opinion expressed in a text corpus, allowing companies to gauge public opinion, customer feedback, and brand reputation. Another application is spam filtering, where text processing techniques automatically detect, and filter unsolicited and unwanted emails or messages based on their content. Additionally, text processing enables information extraction, extracting specific information or entities from text, such as named entities or key facts from news articles or documents (Nadkarni, Ohno-Machado and Chapman, 2011).

In conclusion, text processing, driven by techniques such as NLP and machine learning, automates the analysis and manipulation of unstructured text data. It plays a crucial role in uncovering valuable insights and information hidden within textual data, facilitating tasks such as language translation, sentiment analysis, spam filtering, and information extraction. By employing text processing techniques, companies and organizations can make informed decisions, gain a deeper understanding of customer feedback, and extract relevant information from large volumes of text data (Gruetzemacher, 2022).

# Why the chosen data set is appropriate for analysis

The SMS Spam Collection Dataset is an appropriate dataset for text analysis and building a prediction model to classify texts as spam or legitimate (ham). Here's why this dataset is suitable:

Dataset Composition: The SMS Spam Collection Dataset consists of 5,574 SMS messages in English, with each message labelled as either "ham" (legitimate) or "spam." The dataset provides a balanced representation of both ham and spam messages, which is essential for training a reliable prediction model.

Variety of Sources: The dataset is collected from various sources, including the Grumbletext website, NUS SMS Corpus, Caroline Tag's PhD Thesis, and the SMS Spam Corpus v.0.1 Big. These sources ensure a diverse range of messages from different contexts and origins, enhancing the dataset's representativeness.

Real-world Data: The messages in the dataset are real and non-encoded, reflecting the type of SMS messages users typically encounter in their daily lives. This real-world aspect makes the dataset relevant for practical text analysis applications.

Labelling Accuracy: The dataset has undergone manual labelling, ensuring that each message is correctly classified as ham or spam. The labelling process involved scanning hundreds of web pages and carefully identifying spam messages. This attention to labelling accuracy enhances the reliability of the dataset for training a prediction model.

Research and Usage: The SMS Spam Collection Dataset has been widely used in academic research, including studies on SMS spam filtering. It has been referenced in previous papers and research projects, establishing its credibility and usefulness.

By utilizing the SMS Spam Collection Dataset, researchers and data scientists can leverage its comprehensive and well-labelled data to develop accurate prediction models for SMS spam detection and analysis.

# Analysis that will be conducted

Based on the SMS Spam Collection Dataset, the analysis typically focuses on classifying SMS messages as either spam or legitimate (ham). The goal is to develop a model that can accurately distinguish between spam and ham messages based on the provided dataset. This analysis aims to answer the question of how well a machine learning model can classify SMS messages as spam or ham using the available data. To achieve this, the following analysis steps can be performed:

Data Exploration: Investigate the characteristics of the dataset, such as the distribution of spam and ham messages, the length of messages, and any patterns or trends that may emerge.

Text Pre-processing: Clean the text data by removing unnecessary characters, converting text to lowercase, removing stop words (commonly used words with little significance), and applying techniques like tokenization, stemming or lemmatization to reduce words to their root form.

Feature Engineering: Convert the text data into numerical features that machine learning algorithms can understand. Common techniques include using the Bag-of-Words model, TF-IDF (Term Frequency-Inverse Document Frequency) representation, or word embeddings like Word2Vec or GloVe. I’ll be using the TF-IDF technique.

Model Training and Evaluation: Split the dataset into training and testing sets. Train different classification models, such as Naive Bayes, logistic regression, or decision trees, on the training data. Evaluate the models' performance on the testing data using appropriate evaluation metrics like accuracy and precision score. This step helps identify the most effective model for spam detection.

Model Fine-tuning and Optimization: Experiment with different techniques to improve the model's performance, such as adjusting hyperparameters, using more advanced algorithms like Support Vector Machines (SVM) or ensemble methods (Random Forest, Gradient Boosting), or applying cross-validation to obtain more reliable performance estimates. For these models I’ll be using two classifiers, namely voting and stacking classifiers. I’ll then be picking one of the two classifiers for the final model testing.

Final Model testing: Select the best-performing model based on evaluation metrics and test it on unseen data or conduct additional cross-validation to estimate its performance more accurately. Report the final evaluation results and compare them with the initial performance.

# Process used in data analysis.

In this project, the type of analysis that's being conducted is text classification. Text classification is the process of assigning tags or categories to text according to its content. It's one of the fundamental tasks in Natural Language Processing (NLP) with broad applications such as sentiment analysis, topic labelling, spam detection, and intent detection. In this project, the goal is to classify text messages as either spam or ham (not spam). The dataset used in this project is the SMS Spam Collection dataset from the UCI Machine Learning Repository. The dataset consists of 5,169 SMS messages that are tagged as either spam or ham. The dataset is split into a training set and a test set. The training set consists of 4,134 messages, while the test set consists of 1,035 messages. The goal is to build a model that can accurately classify text messages as either spam or ham. The model will be trained using the training set, and then evaluated using the test set. The model will be evaluated based on its accuracy and precision scores. The accuracy score is the proportion of correct predictions over the total number of predictions. The precision score is the proportion of correct positive predictions over the total number of positive predictions. The model will be evaluated using the accuracy and precision scores because the dataset is imbalanced. Most of the messages in the dataset are ham messages, while only a small proportion of the messages are spam messages. Therefore, the accuracy and precision scores will provide a more accurate representation of the model's performance compared to other metrics such as the F1 score.

## Step 1:

The first part of the text analysis is to import the necessary libraries for data preprocessing, text processing, and modeling. The libraries are imported using the import keyword. Some libaries included for data processing include pandas, numpy, and sklearn. The panda’s library is used to read the dataset into a DataFrame. The numpy library is used to perform mathematical operations on the data. The sklearn library is used to split the dataset into training and test sets. Some libraries included for text processing include nltk, re, and sklearn. The nltk library is used to perform text preprocessing and text processing. The re library is used to perform regular expression operations on the text. The sklearn library is used to create a pipeline for text processing. Some libraries included for modeling include sklearn, nltk, and sklearn.ensemble. The sklearn library is used to create a pipeline for modeling. The nltk library is used to create a tokenizer for text processing. The sklearn.ensemble library is used to create an ensemble model for text classification.

## Step 2:

The next step is to perform data cleaning. The parts in this step are dropping duplicates, dropping null values, and resetting the index. The first step is to drop duplicates. This is done by calling the drop\_duplicates() method on the DataFrame df. The drop\_duplicates() method removes duplicate rows from the DataFrame. The next step is to drop null values. This is done by calling the dropna() method on the DataFrame df. The dropna() method removes rows with null values from the DataFrame. The next step is to reset the index. This is done by calling the reset\_index() method on the DataFrame df. The reset\_index() method resets the index of the DataFrame. The drop\_duplicates(), dropna(), and reset\_index() methods are called in a chain. The DataFrame df is assigned to the result of the chained methods. The first few rows of the DataFrame are printed using df.head() to see the results of the data cleaning.

## Step 3:

The following step involves Explorative Data Analysis (EDA). This is done to get a better understanding of the data. The EDA is done by creating a wordcloud for the spam and ham messages to visualize the most frequent words in each category. The type of EDA that will be used are pie charts, bar graphs, wordclouds, pairplot and a correlation matrix. The pie chart is used to show how much of the target variable is spam or ham as a percentage, the bar graph is used to show the distribution of the length of the text messages, and the wordcloud is used to visualize the most frequent words in each category. We also added 3 new columns namely num\_characters, counts the total number of characters in the text, num\_words, counts the total number of words in the text, and num\_sentences, counts the total number of sentences in the text. The correlation matrix is used to show the correlation between the target variable and the length of the text messages, the length of the text messages and the number of words, and the number of words and the number of sentences.

## Step 4:

In this step is focused on the main text classification used for data pre-processing. The first step is to create a pipeline for text processing. This is done by creating a pipeline object using the Pipeline class from the sklearn.pipeline module. The pipeline is initialized with a list of tuples, where each tuple consists of a step name and a corresponding data processing object or function. The pipeline consists of the following steps: tokenize, stem, lemmatize, and preprocess. The tokenize step uses the Tokenizer() object to tokenize the input text. The stem step uses the Stemmer() object to perform stemming on the tokenized text. The lemmatize step uses the Lemmatizer() object to perform lemmatization on the stemmed text. The preprocess step uses the Preprocessor() object to perform additional preprocessing on the lemmatized text. The next step is to apply the pipeline to the text. This is done by assigning a sample text message to the text variable. The pipeline is applied to the text by calling the transform() method on the pipeline object with the text as input. The preprocessed text is obtained from the transformed result, and it is printed to the console. The next step is to apply the pipeline to the 'text' column of a DataFrame called df. The transform() method is called on the pipeline object with the 'text' column as input. The transformed texts are assigned to a new column called 'transformed\_text' in the DataFrame df. Finally, the first few rows of the DataFrame are printed using df.head() to see the results of the transformation. Additionally, a histogram is included to show the distribution of the length of the text messages after the text processing.

## Step 5:

In this step of the project, we'll be building the machine learning model and evaluating different models to see which is best for this text classification. Firstly, we used TF-IDF on the transformed\_text. Then we'll be checking which of the 3 Naive Bayes models, GaussianNB,MultinomialNB, and BernoulliNB are best. Once we've picked the best model, we'll compare that to other models such as KNN, Random Forest, and SVC. We'll be using the accuracy and precision scores to evaluate the models. The accuracy score is the proportion of correct predictions over the total number of predictions. The precision score is the proportion of correct positive predictions over the total number of positive predictions. The model will be evaluated using the accuracy and precision scores because the dataset is imbalanced. Most of the messages in the dataset are ham messages, while only a small proportion of the messages are spam messages. Therefore, the accuracy and precision scores will provide a more accurate representation of the model's performance compared to other metrics such as the F1 score. The pipelines used are done by creating a dictionary called clfs. Each key-value pair in the dictionary represents a classifier name and its corresponding pipeline object. The pipeline object is created using the Pipeline class from the sklearn.pipeline module. The classifier and its parameters are defined within the pipeline as a tuple. The next step is to define a function called train\_classifier() that takes a classifier, training features (X\_train), training labels (y\_train), testing features (X\_test), and testing labels (y\_test) as parameters. This function fits the classifier to the training data, makes predictions on the testing data, and calculates the accuracy and precision scores. The accuracy and precision scores are then returned. The code then iterates over the items in the clfs dictionary using a for loop. For each classifier, it calls the train\_classifier() function with the appropriate parameters. The accuracy and precision scores are printed for each classifier, and then appended to the accuracy\_scores and precision\_scores list. Considering the accuracy and precision scores, Multinomial NB, Random Forest, and SVC appear to be the most promising algorithms for text classification in this evaluation. In closer inspection, overall, the MultinomialNB is the best model. Considering both accuracy and precision, it appears that Multinomial NB and KNN have the highest scores in both metrics.

## Step 6:

This second last step is model improvement. Firstly, we create a new data frame that changes the maximum parameters of the TF-IDF and see how each model compares to each other. Then we'll be using two types of classifiers, voting and stacking classifiers respectively. The Voting Classifier is a machine learning model that combines the predictions from multiple other models. It is an ensemble model that makes predictions by combining the predictions from multiple other models. The Stacking Classifier is a machine learning model that combines the predictions from multiple other models. It is an ensemble model that makes predictions by combining the predictions from multiple other models. The Voting Classifier and Stacking Classifier are promising models for text classification in this evaluation.

## Step 7:

This final step is testing the model with an input method that will either classify the text as spam or ham. A pipeline is used to fit tfidf and the model. Then the user will input a text message and the model will predict if it's spam or ham. Then the user will be asked if they want to input another text message. If they do, the process will repeat. If they don't, the program will end.

# References

UCI Machine Learning (2016). *SMS Spam Collection Dataset*. [Dataset]. Available at: https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset [Accessed date: 28 June of 2023].