TEXT MINING AND SEARCH

AMAZON FINE FOOD REVIEWS

AA 2022/2023

Merlo Fabrizio 847203 - Sanvito Simone 844794

Abstract

The aim of this project is to perform some NLP tasks on the Amazon Fine Food Reviews dataset, which consists of reviews of fine foods from Amazon.

The tasks performed on this dataset are text pre-processing, text representation, text clustering and text summarization. The models used for text clustering are k-means, hierarchical clustering, OPTICS, DBSCAN and Birch; for text summarization, there have been used models like TextRank, Txtai, T5, BART and Seq2Seq.
Both tasks are presented with relative evaluation metrics.

Introduction

The project aims to use the Amazon Fine Food Reviews dataset from Kaggle to perform text pre-processing, text representation, text clustering and text summarization tasks.

The dataset consists of over 500,000 food reviews from Amazon. The data includes information on the product, the user, and the review itself.

The dataset contains the following fields:

- ProductId: a unique identifier for the product
- UserId: a unique identifier for the user
- ProfileName: the user's name
- HelpfulnessNumerator: the number of users who found the review helpful
- HelpfulnessDenominator: the number of users who indicated whether they found the review helpful or not
- Score: the rating given by the user (ranging from 1 to 5)
- Time: the timestamp for the review
- Summary: a summary of the review
- Text: the full text of the review

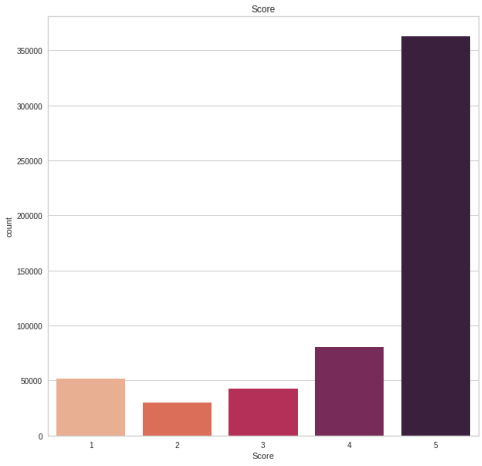
The data is provided in a single .csv file, with each row representing a single review.

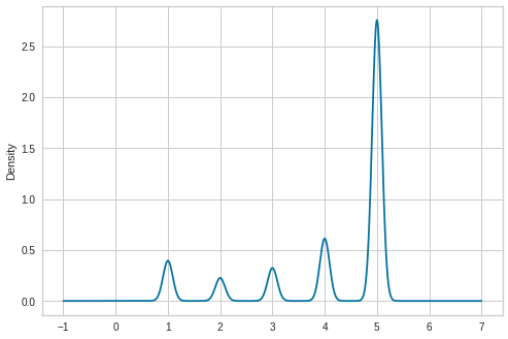
Exploratory Analysis

At first, there has been a phase of cleaning the dataset: first of all, it was performed the removal of Null/NaN values.

Subsequently, it has been made a procedure for duplicated values removal: the rows of the dataset have been considered duplicated if they had the same values for the columns ProductId, UserId, ProfileName, Time, Score, Summary and Text. With the drop duplicates method, rows that had the same values in the columns listed above were identified and removed.

The first visual analysis was made to represent the distribution of the reviews in the different categories of the Score column.





	Rating	Total	Percent
0	5	362684	63.900067
1	4	80598	14.200289
2	1	51988	9.159590
3	3	42575	7.501145
4	2	29735	5.238909

Figures 1 & 2 & 3: Distribution of values of the Score class

As we can see in the pictures, the variable Score is really very unbalanced, with most of the rows (~64%) that have a rating equal to 5.

Subsequently, a plot was created to see the average length of the reviews, dividing the reviews by the value of the score column.

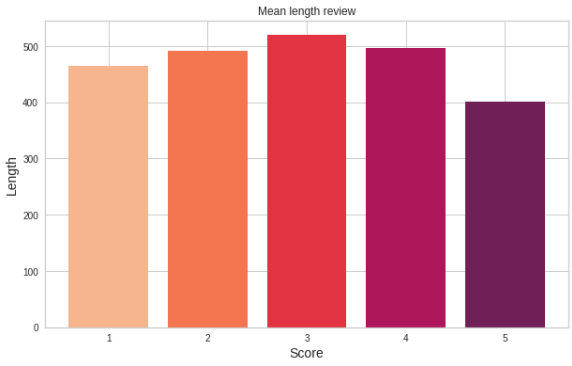


Figure 4: Mean length of the reviews by Score class

The length of a review is calculated as the number of characters that are contained in that review.

After having created a word cloud for the 20 most occurrent words in the text column, it was evident that the dataset needed a pre-processing phase before proceeding with text-mining tasks.

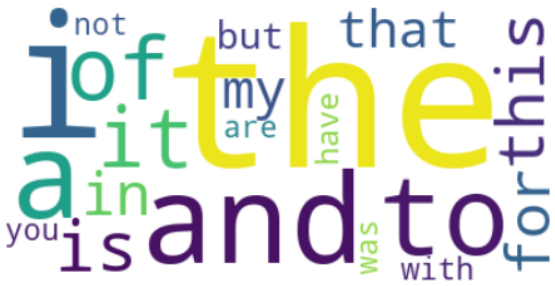


Figure 5: Wordcloud of the 20 most occurrent words in reviews' texts

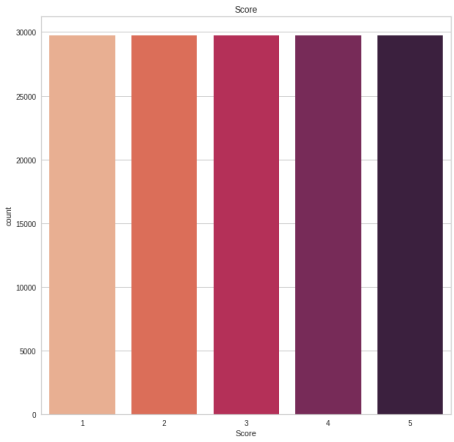
These words, in fact, are at most stop-words.

Balancing of Dataset

Since the dataset was unbalanced, before going on and starting with pre-processing phase, it was needed an intermediate step to balance the dataset, particularly for the values of the Score column.

It has been calculated the minimum number of rows across all unique values in the 'Score' column and that number has been used as the number of rows randomly kept for each group.

The results obtained are the following:



	Rating	Total	Percent
0	1	29735	20.0
1	2	29735	20.0
2	3	29735	20.0
3	4	29735	20.0
4	5	29735	20.0

Figures 6 & 7: Distribution of values of the Score class after balancing

So the dataset is perfectly balanced now.

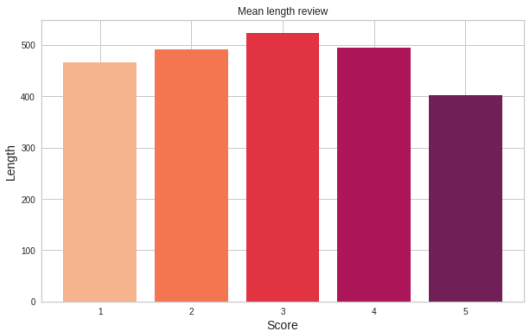


Figure 8: Mean length of the reviews by Score class after balancing

Even the distribution of the mean length of the reviews changes a bit. Now dataset is ready for pre-processing phase.

Pre-Processing

Prior to putting the data into the models, it was necessary to apply the following preprocessing steps.

First step: Normalization

This step is needed in order to be able to refer to a single word with different forms of it (ex. We want to match “U.S.A.” and “USA”).
To complete this first part of pre-processing of review texts, the following procedures were performed:

- conversion in lowercase;
- removal of links;
- removal of HTML tags;
- removal of special chars;
- removal of numbers;
- removal of emojis;
- removal of repeated letters: consent to the repetition of two letters maximum (ex. “gooood” -> “good”);
- removal of punctuation (except for the apex): did not remove the apex because if there is "didn't." it is not read as a contraction (because of the point). This allows for better recognition of contractions;
- expansion of contractions;
- removal of all punctuations;
- removal of whitespaces.

Second step: Stop words removal

This procedure involves identifying and removing stop words i.e. very common words (such as "a," "and," "the," and "in”) from a text in order to make the text easier to process.

Third step: Tokenization

Tokenization is the process of dividing a text document into smaller segments or units.

Fourth step: Lemmatization

Lemmatization brings every word to the root of that word.
We have chosen lemmatization because it is a less rough method (more accurate and more comprehensive).

Here there are some statistics after pre-processing.

It has been calculated the distribution of the number of words contained in a summary:

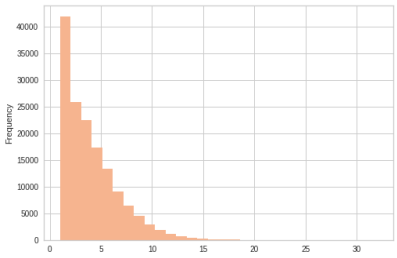


Figure 9: Distribution of the number of words contained in a summary

mean	4.385122
std	2.756634
min	1.000000
25%	2.000000
50%	4.000000
75%	6.000000
max	32.000000

Figure 10: Numerical distribution of the number of words contained in a summary

Summaries are composed of a mean of 4 words.

It has been calculated the distribution of the number of words contained in a review:

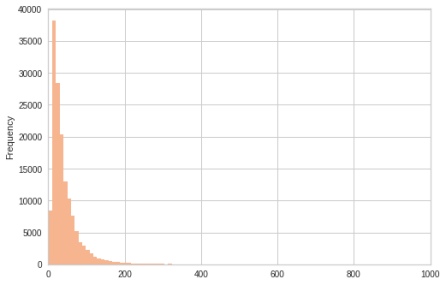


Figure 11: Distribution of the number of words contained in a review

mean	42.349965
std	41.401745
min	1.000000
25%	18.000000
50%	30.000000
75%	52.000000
max	1956.000000

Figure 12: Numerical distribution of the number of words contained in a review

A text has a mean of 42 words in it.

Text Representation

Sample

For computational and time reasons, a balanced data sample was used.

15% of the rows of balanced data were taken, keeping about 4400 occurrences for each Score class value, for a total of 22300 rows.

TF-IDF

For the text representation part, the choice of the technique to be used fell on TF-IDF.

TF-IDF is a numerical statistic that is used to reflect how important a word is to a document in a collection or corpus. It is composed of

- TF (Term Frequency): the number of times that a specific word appears in a document, normalized by dividing by the total number of words in the document;
- IDF (Inverse Document Frequency): a value which is obtained by dividing the total number of documents by the number of documents containing a specific word, and then taking the logarithm of that quotient.

TF-IDF is important because it weights the words in the vocabulary and makes it possible to filter out the noise and highlight the most important words in each document.
The usage of this technique can even make text clustering more effective by reducing the dimensionality of the data and focusing on the most relevant words.

By setting the ngram_range parameter to (1, 2), the TF-IDF vectorizer considers all single words and all two-word sequences when building the vocabulary (unigram and bigram). The parameter for the maximum number of features was set to 8 thousand, so the 8 thousand most occurrences of unigram and bigram were taken.

SVD for Dimensionality reduction

Next, to lighten the computational load, it was decided to proceed with a dimensionality reduction step using SVD.

Text data is often represented as matrices of word counts (in this case using TF-IDF), which can have a large number of columns. High dimensionality can make it difficult to apply clustering algorithms, as it can cause overlap between clusters. SVD decomposition allows for reducing the dimensionality of the data while preserving most of the information present in the original data.

Since TF-IDF has to deal with a sparse matrix it was decided to use SVD keeping 300 components in order to have computational advantages.

Text Clustering

Clustering is the process of grouping a set of objects into classes of similar objects. In particular, texts within a cluster should be similar and texts from different clusters should be dissimilar.

The score variable was used in the clustering evaluation to compare the clusters created by some algorithms with the original score of the reviews.

To carry out this task, different text clustering techniques were considered:

- **K-means clustering**: the texts are divided into a specified number k of clusters by means of the calculation of the distance between the texts and the centroids of the clusters.
- **Hierarchical clustering**: this method creates a hierarchy of clusters. It outputs a more informative structure than the unstructured set of clusters returned by flat clustering.
- **OPTICS**: finds a core sample of high density and expands clusters from them. OPTICS is a

density-based algorithm that can identify clusters of varying densities, by building a reachability plot which assigns a reachability distance to each point that represents the distance to the closest dense cluster.

- **DBSCAN**: DBSCAN is a density-based clustering algorithm that groups together points that are closely packed together, and marks points that lie alone in low-density regions as outliers.
- **Birch**: is a hierarchical density-based clustering algorithm that constructs a tree data structure with the cluster centroids being read off the leaf. These can be either the final cluster centroids or can be provided as input to another clustering algorithm.

For all text clustering procedures, 2 values were used for the number of clusters: 3 and 5. These values were calculated by the elbow method by computing the values with two different metrics: Shilouette and Distortion.

Silhouette
k=3

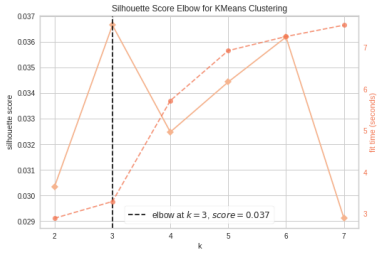


Figure 13: Elbow method with Silhouette metric

Distortion
k=5

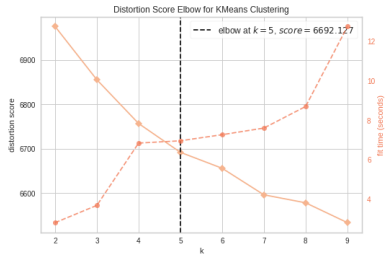


Figure 14: Elbow method with Distortion metric

Generally, for all implemented methods, the coefficient calculated via the "Distortion" metric provided (even if slightly) better results.

Both supervised and unsupervised clustering evaluation metrics were used. Supervised metrics provide an assessment of the quality of the generated clusters compared to true clusters, so they are useful in determining whether a clustering algorithm performs well for a given data set.
Unsupervised metrics, on the other hand, can provide information about the internal consistency of the generated clusters.

K-means Clustering

K-means k = 3

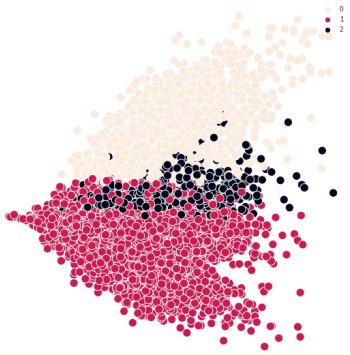


Figure 15: Plot for representation of the clusters for k-means with k=3

It seems that the clusters are balanced, (especially clusters 0 and 1). To be sure, it is needed to inspect more.

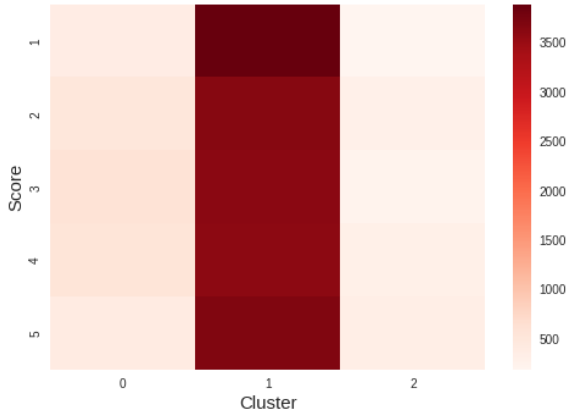


Figure 16: Confusion matrix for k-means with k=3

It turns out that almost all observations are in cluster 1.

K-means k=5

```
Rand index           : 0.4690558425572383
Adjusted Mutual Info : 0.0026645465947149434
Homogeneity          : 0.002331639741122631
Completeness         : 0.0039985272192034306
V measure            : 0.002945617399568574
Fowlkes Mallows      : 0.33278845761761544
Number of clusters 5

Silhouette           : 0.03442988052659058
Calinski Harabasz    : 352.3658491043245
Davies Bouldin       : 3.854574429053276
Number of clusters 5
```

Figure 17: Evaluation scores for k-means clustering with k=5

In general, these results indicate that the clustering model does not work very well, especially in terms of matching true clusters and the purity of the generated clusters.
For example, the Rand index (0.469) indicates a moderate match between the clusters assigned by the model and the "true" clusters.
The Silhouette (0.03) indicates that the elements within the same cluster are not very similar to each other. This could indicate that the clusters are overlapping with each other, that is, there is an area of overlap between the clusters where some elements could be assigned to each cluster. In other words, the clusters are not clearly separated from each other.

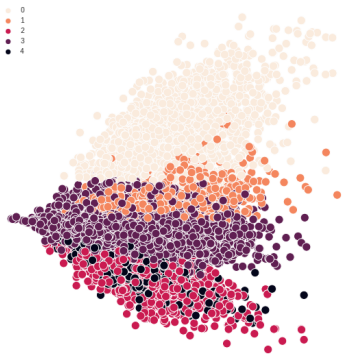


Figure 18: Plot for representation of the clusters for k-means with k=5

As seen in the k-means with 3 clusters, even in this graph it might appear that the clusters are well separated from each other (although not well separated).

After analysing the results of the k-means, we now know that we need to see the distribution of observations in the clusters in more detail.

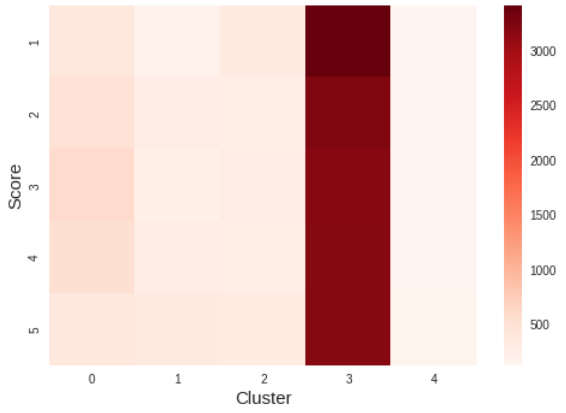


Figure 19: Confusion Matrix for k-means with k=5

In fact, as was the case with the k-means with 3 clusters, this algorithm does not identify the same groups that are identified through the score column values, but rather the observations are almost all concentrated in one cluster.

	Cluster 0	Cluster 1	Cluster 2	Cluster 3	Cluster 4
Score 1	381	184	334	3414	147
Score 2	510	298	275	3239	138
Score 3	602	237	297	3187	137
Score 4	549	300	280	3204	127
Score 5	403	354	330	3201	172

It can be seen from the table that most of the observations are contained in cluster 3, but the other clusters also contain a few hundred pieces of information per cluster.

Table 1: number of occurrences for every score value in each cluster

Below are the word clouds generated for the 5 clusters.

Cluster 0

Figure 20: Wordcloud for cluster 0 - k-means with k=5

Cluster 1

Figure 21: Wordcloud for cluster 1 - k-means with k=5

Cluster 2

Figure 22: Wordcloud for cluster 2 - k-means with k=5

Cluster 3

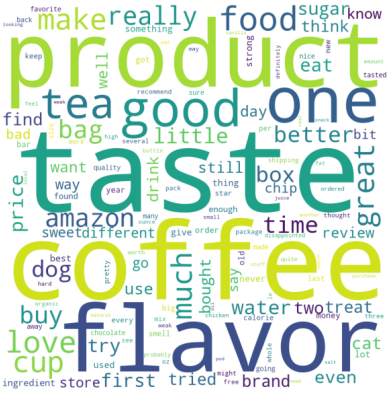


Figure 23: Wordcloud for cluster 3 - k-means with k=5

Cluster 4

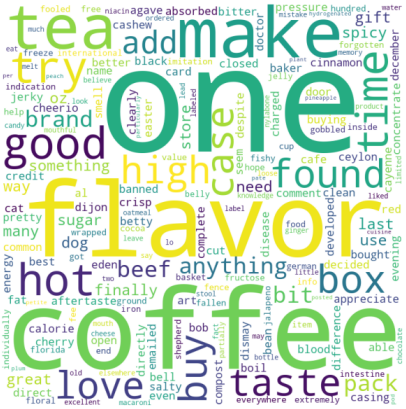


Figure 24: Wordcloud for cluster 4 - k-means with k=5

Word clouds are not very informative; they are difficult to interpret.

Hierarchical Clustering

Different types of linkage parameters (single or complete) and different types of affinity parameters (Euclidean and Manhattan) were used.

Single linkage is used to identify clusters with a small number of samples. Complete linkage is used to identify clusters with a large number of samples. The Euclidean distance between two points is the straight-line distance between them and is calculated as the square root of the sum of the squares of the differences between the coordinates of the points.
The Manhattan distance between two points is the sum of the absolute differences between their coordinates.

All hierarchical clustering algorithms have similar performance: the thing that stands out most is that these algorithms result in the association of all (or nearly all) observations within a single cluster.

So you get one cluster that contains almost all the observations and the others that are empty or contain a small subset of observations.
Compared with k-means clustering, the number of observations for the less populated clusters in these groups is significantly fewer: in fact, some clusters even have only 1 or even no observations per score class.

Below is an example, specifically hierarchical clustering with 5 clusters,

complete linkage approach and with Manhattan metric:

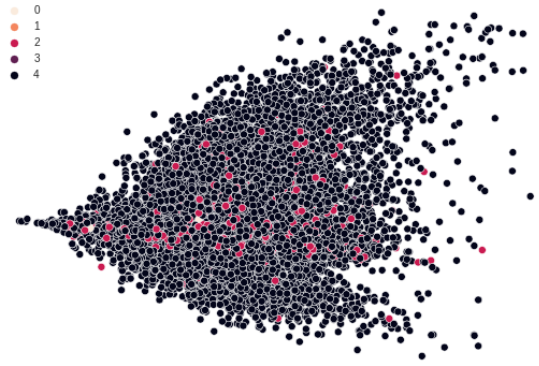


Figure 25: Plot for representation of the clusters for hierarchical clustering complete linkage and Manhattan metric with k=5

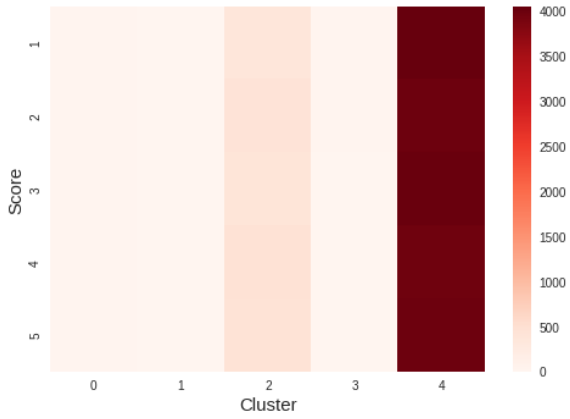


Figure 26: Confusion matrix for hierarchical clustering complete linkage and Manhattan metric with k=5

OPTICS, DBSCAN

OPTICS and DBSCAN do not require the number of clusters to be specified in advance. Instead, they automatically determine the number of clusters by identifying areas of the dataset with higher density.

These algorithms have low performances, similar to hierarchical clustering ones, so the automatic choice of the number of clusters does not benefit the analysis.

Birch

The Birch algorithm has slightly better performances: this is most visible for the method that uses 5 clusters.

In this case, it could seem that there are two clusters in which the observations are contained.

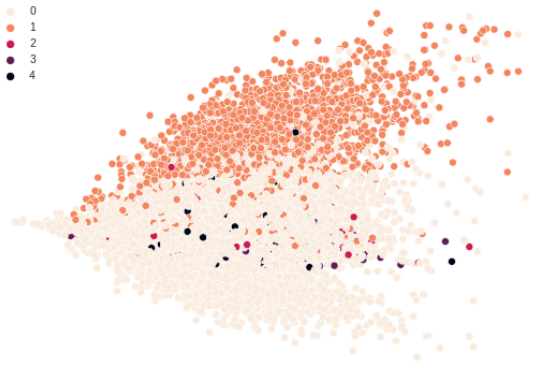


Figure 27: Plot for representation of the clusters for Birch with k=5

But in reality, the problem of clustering the observations into a single cluster persists, as is clearly seen from the confusion matrix.

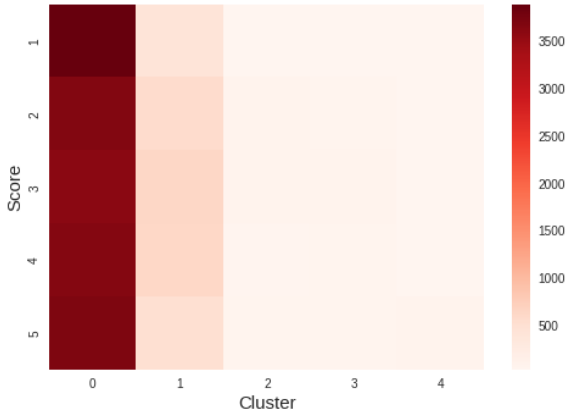


Figure 28: Confusion matrix for Birch with k=3

Final considerations

The models that perform best generally are those built using k-means as an

algorithm.
In second place are those built using Birch.
Models that use hierarchical clustering algorithms perform poorly, as do OPTICS and DBSCAN.
Generally, all these algorithms collect all values within a single class (and this is well evident from both the low-performance value and the confusion matrices).

Text Summarization

Text summarization is the process of reducing a text document to its most important points in order to provide a summary that retains the key ideas of the original text.

There are two main approaches to text summarization: extractive and abstractive.

- **Extractive**: involves identifying important parts of the original text and combining them to create a summary;
- **Abstractive**: involves generating new sentences from scratch to capture the main ideas in the original text.

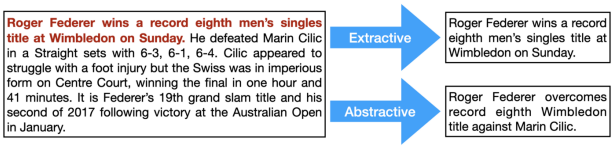


Figure 29: Example of difference between Extractive and Abstractive summarization

In our case, using preprocessing on the Summary column, we saw that approximately 50% of the words of the Summary were contained in the corresponding Text column cell.

For this reason, we decided to implement both methods (extractive and abstractive), but to focus mainly on the abstractive.

Pre-Processing

Before feeding the data into the models, it was essential to perform the following preprocessing procedures:

- Removing the lowest 25% (first quartile) of data from the "Text" category, so that the rows contained in the Text column will have more than 34 words.

count	567580.000000
mean	81.700340
std	79.631819
min	3.000000
25%	34.000000
50%	58.000000
75%	100.000000
max	3526.000000

Figure 30: Description of the number of words contained in the Text columns

- Removing the lowest 25% (first quartile) of data from the "Summary" column; so the rows that contain in the column Summary under 3 words were removed.

count	567580.000000
mean	4.125810
std	2.617353
min	1.000000
25%	2.000000
50%	4.000000
75%	5.000000
max	42.000000

Figure 31: Description of the number of words contained in the Summary columns

- Randomly select a subset of 50.000 data from the remaining in order to use those rows in the models.
The 50.000 data were used for the seq2seq model since it needs a large

amount of data to be trained, while for all other models, 2000 rows of these 50.000 were taken.
This was made because each prediction would take approximately 4 seconds to execute and with 50.000 rows it would have taken approximately 56 hours.

Extractive Summarization

For the extractive summarization part, we decided to use the preprocessed text and summary column using the same step defined in the pre-processing phase defined above (page 4), because we saw that by preprocessing both columns we could find more words that are present both in the summary column words and in the text column.

TextRank: It works by creating a word graph and using Google's PageRank algorithm to determine the relative importance of words in the graph.

The text is then summarized by selecting the sentences that contain the most important words.

We used the cleaned texts and a ratio (a number between 0 and 1 that determines the proportion of the number of sentences of the original text to be chosen for the summary) equal to the division between max number of words for column Summary and max number of words for column Text.

GPT3: is a neural network machine learning model trained using internet data to generate any type of text. Developed by OpenAI, it requires a small amount of input text to generate large volumes of relevant and sophisticated machine-generated text.
The results were not added because API requests are provided for a fee.

Abstractive Summarization

For the abstractive summarization part, we decided not to preprocess the text and summary column.

Txtai: executes machine-learning workflows to transform data and build AI-powered semantic search applications.

T5: is a transformer model that is trained in an end-to-end manner with text as input and modified text as output, in contrast to BERT-style models that can only output either a class label or a span of the input.

Seq2Seq UniDirectional Model:
These models map an input sequence to a target sequence and are composed of three main parts: an encoder, a decoder and an attention layer. The encoder reads the input sequence one step at a time and captures the context of the input, outputting a context vector. The decoder then takes this context vector and generates the output sequence, predicting the next word based on the previous one. These models typically use gated Recurrent Neural Networks (RNNs) such as Long Short-Term Memory (LSTM) units. The attention mechanism addresses the problem by having the model focus on specific parts of the sequence when predicting the output word, instead of considering the entire sequence.

We use a unidirectional LSTM encoder-decoder with randomly initialized

word embeddings and a global attention layer between the encoder and decoder. This results in the use of all hidden states of the encoder to generate the attended context vector.
During training, the model trains the encoder and the decoder simultaneously. The optimal hyperparameters for training were selected through multiple trials, resulting in

- Optimizer: RMSPROP: This was chosen because it performed better than other established optimizers like ADAM.
- Batch size: 32(Default Value): This was chosen as it yielded better results.
- Epochs: 10. The number of epochs was set to 10, but due to the long training time on a medium/high-end consumer PC, it was not practical to continue.

BART: is a transformer encoder-encoder (seq2seq) model that combines an autoregressive (GPT-like) decoder with a bidirectional (BERT-like) encoder.

Evaluation

The performances of the models were evaluated using two different methods:

Rouge: The first method compares the similarity between the system-generated summary and one or more human-written reference summaries by counting the number of matching n-grams. There are different variations of ROUGE depending on the length of the n-grams used. In this case, ROUGE-1 and ROUGE-2 scores were used because the summaries in the dataset were short. However, ROUGE scores only evaluate the similarity of the words used, not the coherence or fluency of the summary. To evaluate these aspects, a human evaluator must assess the results for fluency and correctness.

BLEU: BLEU (Bilingual Evaluation Understudy) is a score for comparing a candidate's translation of the text to one or more reference translations. Although developed for translation, it can be used to evaluate text generated for a suite of natural language processing tasks. It compares a generated summary to one or more reference summaries and calculates a score based on the number of matching n-grams between the two. A higher BLEU score indicates that the generated summary is more similar to the reference summaries. However, it's important to note that the BLEU score alone may not be a reliable indicator of the summary's quality, as it doesn't take into account the fluency, coherence or overall meaning of the generated summary.

Results

From the initial considerations, as we expected, the extractive model performs worse than the abstractive model.
This result was expected as it was noted that only 50% of the words in the summaries were contained in the related texts. Thus, since abstractive summarization uses new words not found in the texts to make the summary, the result obtained is consistent with expectations.

Model	Rogue 1	Rogue 2	Bleu
TextRank	0.90%	0.11%	0.96%
Txtai	7.30%	1.75%	11.5%
T5	5.70%	1.13%	9.98%
BART	5.42%	1.29%	10.5%
Seq2Seq	5.63%	0.07%	6.63%

Table 2: Model results

The results (Table 2) show that the Txtai model has the highest scores across all three metrics, indicating the best performance on the text summarization task among the models compared. Txtai also has relatively high scores, indicating good performance. The other models, TextRank, Seq2Seq and BART have more moderate scores, indicating that their performance on the text summarization task is not as strong as T5 and Txtai.

The performance of our model falls short of current top-performing summarization models, especially regarding Rogue 2. It is clear that Seq2Seq models need significant computational resources to generate high-quality results, as they have a large number of parameters to train. In this specific scenario, the models were trained for a relatively short period of time (10 epochs) using a limited number of rows, on a high-end consumer computer, in our particular case, Google Colab was used with the corresponding GPU. The training process took more or less 25 minutes for each epoch, so more than 4 hours to complete.

Conclusions and future developments

As for text clustering, the analyses performed do not bring much information. The k-means, is, in this case, the clustering algorithm that works best. All algorithms used have a common problem: in fact, all approaches group all observations into one cluster.

Regarding text summarization, the extractive model performs worse than the abstractive model.
Moreover, it is important to keep in mind that these metrics are not the only way to evaluate the performance of text summarization models and they may not capture all aspects of summary quality. For this reason, the values of the evaluation metrics must be combined with a human evaluation to evaluate the semantic goodness of the summaries

As possible future developments, surely one could be to use a powerful machine that can be able to process more data without computational capacity issues and time issues.
For both clustering and text summarization, another possibility could be to try different parameters in the methods already implemented and to try different new algorithms.
Also, regarding seq2seq, one could try increasing the number of epochs and try implementing the bi-directional method.