universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Text Analytics for Customer Relationship Management in Tourism“

verfasst von / submitted by

Sofiia PIVEN

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2023 / Vienna 2023

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 977
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Business Analytics
Betreut von / Supervisor:	ao. Univ.-Prof. Mag. Mag. Dr. Werner Winiwarter

Abstract

Travellers have become more demanding of accommodation as the hotel industry has become more competitive. Business owners are not always aware of the problems customers experience during their stay, which prevents small and medium sized businesses from gaining a competitive advantage in the market. In this thesis, I test and compare various supervised and unsupervised learning techniques and sentiment detection models on 500,000 Booking.com reviews. The results show that SpaCy offers the most compatible sentiment extraction method for deep learning. Neural networks tend to outperform classical algorithms, with Random Forest being the only exception, demonstrating excellent accuracy and performance comparable to BERT and RoBERTa. In addition, to improve overall customer satisfaction, I also propose three open-source web applications for hotel management, which include filtering, a self-developed prioritisation model with a focus on negative experiences, and an adjustable re-ranking system that covers individual characteristics of the accommodation.

Keywords: tourism, Booking.com, sentiment analysis, deep learning, neural networks, BERT, RoBERTa, SpaCy, customer prioritisation, customer re-ranking

Zusammenfassung

Aufgrund des stark gestiegenen Wettbewerbs im Hotelgewerbe sind die Ansprüche der Reisenden an ihre Unterkunft gestiegen. Unternehmer sind sich nicht immer der Probleme bewusst, die ihre Kunden während ihres Aufenthalts haben, was kleine und mittlere Unternehmen daran hindert, einen Wettbewerbsvorteil auf dem Markt zu erlangen. In dieser Forschungsarbeit teste und vergleiche ich verschiedene überwachte und unüberwachte Lerntechniken sowie Modelle zur Sentiment-Erkennung anhand von 500.000 Bewertungen auf Booking.com. Die Ergebnisse zeigen, dass SpaCy die am besten mit Deep Learning kompatible Sentiment-Extraktionsmethode bietet. Neuronale Netze übertreffen in der Regel klassische Algorithmen, mit der einzigen Ausnahme von Random Forest, das eine herausragende Genauigkeit und Leistung aufweist, die mit BERT und RoBERTa vergleichbar ist. Um die allgemeine Kundenzufriedenheit zu verbessern, schlage ich außerdem drei Open-Source-Webanwendungen für das Hotelmanagement vor: Filterung, ein selbst entwickeltes Priorisierungsmodell, das sich auf negative Erfahrungen konzentriert, und ein anpassbares Rankingsystem, das die individuellen Besonderheiten des Hotels abdeckt.

Schlüsselwörter: Tourismus, Booking.com, Stimmungsanalyse, Deep Learning, Neuronale Netze, BERT, RoBERTa, SpaCy, Kundenpriorisierung, Re-Ranking von Kunden

Contents

1 Introduction

Tourism is one of the fastest growing sectors of the economy and is now the world's third largest export industry [88]. This also means that quality standards and customer satisfaction requirements are becoming more demanding, especially for small and medium-sized companies, which not only suffered the most during the COVID period, but now need to gain unique competitive advantages in the marketplace in order to increase a market share and return to profitability. All of this means that every customer is valuable and it is within the hotels' power to make their experience positive.

Dealing with reviewer complaints is critical to T&T (travel and tourism) brand loyalty. In addition, comments have a major impact on the company's online reputation, Therefore, it is one of the most important tasks of every company to be able to handle and comfort dissatisfied customers. If they do not receive a response or feel not heard, they may escalate the conflict, for example, by publicly sharing their negative experience on social media. This is obviously damaging for any hotel.

In this thesis, I focus on techniques for identifying negative reviews left by tourists after their stay in the accommodation. It is a classification problem that involves many steps of text analysis: from tokenisation, lemmatisation, further vectorisation, to the detection of the review sentiment, the determination of the aspects, both direct and latent. My goal is to uncover most of the negative reviews for business owners, so that they can go back to the unhappy customers, try to resolve the problem and possibly fix it for future tourists.

This research makes a contribution that goes beyond a simple sentiment analysis, it includes the prioritisation and ranking of reviewers' feedback, simplifying the work of hotel owners who do not have to manually read through each comment and decide who to respond to first, and who also have to get the statistics of the bottlenecks in their business.

My work started with the exploration of the 500,000 reviews scraped from Booking.com. It then involved various vectorisations and four main sentiment detection tools specifically adapted for tourism purposes (see Section 4.2): Vader, TextBlob, AFINN and SpaCy. Text sentiment became the basis for both supervised and unsupervised machine learning algorithms, tailored to maximise the performance and identification of the positive class (negative reviews) from the text alone. Extracting the aspects and their sentiments from reviews was one of the ways to improve the performance of existing models or to create manually tailored algorithms with satisfactory results. Finally, the last and logical step was to create a prioritisation model for public use, with the possibility of adjusting the parameters according to the user's needs. The complex structure of the research techniques, with a cumulative effect, proved once again that hybrid methods, which harmoniously incorporate manually developed methods for specific domain characteristics, gave better results than generic algorithms.

The thesis has six chapters: Introduction, Literature Review, Data, Methods, Results and Conclusion. In Chapter 1, I provide a general overview of the topic, and highlight challenges and areas for development of the tourism domain. Chapter 2 is dedicated to the main researches on the related topics, on the basis of which I define the novelty of my work. In Chapter 3, I describe the dataset that I have used as a framework for my research and the exploratory analysis of the data. Chapter 4 presents the methodology, it is divided in four sections: Equipment Technical Characteristics, Sentiment Detection Models, Sentiment Prediction Models and Customer Prioritisation Models. At the end of the thesis, I discuss the results of the model performances in Chapter 5 and derive conclusions in Chapter 6 together with managerial implications and room for improvement of my work.

2 Literature Review

Dealing with customer reviews and complaints is an important task for any company operating in the tourism sector. However, in the age of digitalisation, it is becoming increasingly challenging. Tourists share their travel experiences on various social media and platforms, which has a massive impact on the reputation of hotels [77]. SMEs (small and medium-sized enterprises), which often have a conservative management style and inflexible policies towards modern issues in the digital world, are unable to effectively address them [25]. In addition, Covid-19 has also negatively influenced this trend, resulting in more than 50% revenue losses in US hotels alone [2], so there is a need for new innovative approaches to **customer relationship management (CRM)** that would correspond to tourism realities in order to make positive changes in the industry.

In addition, Presti and Maggiore [60] proved that immediate CRM strategies and quick responses to tourists improve overall customer satisfaction. Independently of this work, Aksoy and Yilmaz [3] came to the same conclusion that prompt handling of complaints can reduce the negative impact on tourism businesses. Another study in the field using the **Balance Score Card (BSC)** approach (a management system used to transform objectives into performance targets) [66] found a positive effect of using CRM techniques on the organisational performance of hotels.

However, most of the tools and applications involved in CRM, as well as the related researches, focus solely on the consumers of the service, rather than on hotel owners or management for analytical purposes. For instance, Huang and Chang [37] studied how the use of mobile apps influences the behavioural intentions of hotel consumers, while Stankov and Filimonau [74] examined the design perspectives of such apps and which of them could be more profitable in terms of capitalisation.

In contrast to tourism, there are many examples of analytics-related web applications built in **Streamlit** (open source application framework) in other domains [79]: Bioinformatics application built for exploratory data analysis, predictive modelling and data-driven decision making [52], sentiment analysis of Twitter microblogs [58] and electric vehicle related posts [78], movie sentiment analysis as a result of **Naive Bayes** based approach (supervised machine learning technique widely used for text classification) [63, 59].

Introducing suitable publicly available datasets with the reviews representing similar to real world scenarios for further research is another important branch of investigating the topic. Xu, Yang and Wu designed a new dataset **DMASTE (Diversified Multi-domain Dataset For Aspect Sentiment Triplet Extraction)** [89] giving an opportunity to experiment with the settings in-domain and cross-domain for better performance in aspect detection. Similarly, Chia, Chen and Han [19] introduced a new annotated benchmark dataset with two new domains based on hotel and cosmetics reviews.

In general, the topic of sentiment and emotion detection from text and aspect based sentiment analysis has also become extremely progressive recently. Many new or advanced and refined machine learning techniques have been introduced specifically to serve and improve the performance of NLP related tasks.

Yang, Dyer, He, Smola and Hovy proposed a new method for text classification using **hierarchical attention networks** [93]. Other researchers looked at transformer-based approaches and considered four different models to improve sentiment analysis performance: **Bidirectional Encoder Representations from Transformers (BERT)** [23], **Robustly Optimised BERT Pre-training Approach (RoBERTa)** [45], **distilled version of BERT (DistilBERT)** [67], and **large bidirectional neural network architecture (XLNet)** [57]. Similar research in the area of usefulness of customer reviews has been done by Boluki, Sharami, Shterionov [6] and another group Nayeem, Rafiei [55]. The former paper studied pre-trained models such as **RoBERTa** and **XLM-R** (transformer-based multilingual masked language model) [20] and proved that they significantly outperform the baseline of **Random Forest**, a general supervised learning technique that does not incorporate a language model and is purely predictive [59]. The latter additionally focused on the personality of the reviewer and the date of the review on e-commerce platforms .

Sentiment analysis can be extremely useful in some areas that are not immediately obvious, such as education or medicine. For example, a group of researchers from Australia and Hong Kong [70] have studied the effectiveness of opinion mining techniques in surveys and how they can improve the working relationship between students and professors. Another example is the classification of clinical notes. For this purpose, a general weak text classification model **KeyClass** has been proposed to perform the labelling without the need for human participation [31].

There are also many studies dedicated to the extraction of sentiment in social media, taking into account their growing popularity today. Nielsen, in his work on Twitter microblogs [56], developed **ANEW** (Affective Norms for English Words) [76], a labelled list of words with their sentiment scores against which the words in the post are compared and scored. Sally researched the sentiment of smartphone reviews on YouTube and tested them on **Naïve Bayes**, **Decision Tree** [7] and **Support Vector Machine** [65]. Yang, Fig and Sokolova in their research [92] analysed Reddit Covid-related posts using **Vader** [47] and **TextBlob** [69] sentiment extraction techniques.

Another Twitter and Facebook related research has been done in the direction of emotion detection from text. Gaind, Syal and Padgalwar [30] propose a new approach of first classifying the text into six different emotions and further combining them with the help of textual feature analysis or machine learning methods. Similar papers related to emotion extraction have introduced a context-specific model **REDAffectiveLM** [42], a combination of word-based and learning-based emotion detection algorithms [62], a **Bidirectional Long**

Short-Term Memory (BiLSTM) [9] and **Bidirectional Gated Recurrent Unit** (**BiGRU**) [95] ensemble model tested on Twitter posts [61], and **TM-Senti**, a Twitter dataset for better exploration of emotion trend changes in social media [94].

Apart from the overall sentiment detection of the text, one of its specialisations, **Aspect Based Sentiment Analysis** (**ABSA**), is also widely considered in modern studies. Some researchers from Tsinghua University have developed a new ABSA model called **SynGen** [96], which solves the problem of neighbouring dependency, when neighbouring words draw attention from the real aspects and their sentiments. **InstructABSA** [68] is another aspect-based and instruction-tuned modelling approach designed to effectively solve ABSA subtasks: **Aspect Term Extraction** (**ATE**), **Aspect Term Sentiment Classification** (**ATSC**) and **Joint Task Modelling**. Similar to the tools described above, Yang and Li presented a simple and easy-to-use **PyABSA** model [90], a modularised framework based on **PyTorch** (machine learning framework with the platform for building and training neural networks)[1] that can be implemented in a few lines of code.

Existing ABSA techniques are known for their high performance levels, which, however, is not the case for cross-domain analysis. Shi, Li, Bai, Yang and Jiang [72] addressed this problem by introducing a soft prompt-based joint model with external linguistic features, which allows to bridge the gap between the source and target domains. A more general solution to this problem was proposed in the form of a unified bidirectional generative algorithm [22], which trains the model not only in the text-to-label direction, but also in the label-to-text direction.

There are several ABSA challenges that have been fully or partially resolved thanks to recent publications. One of them is to improve the interpretability of the classification. Cheng, Zhou, Wu, Chen and He in their paper proposed a solution in the form of an **Interpretation-Enhanced Gradient-based model** (**IEGA**) [18] by incorporating small explanatory annotations. Another research introduced a contrastive variational information bottleneck framework (called **CVIB**) [14] to reduce spurious correlations for ABSA. Dasgupta and Sen [21] also contributed by developing an aspect-based opinion mining framework based on the concept of transfer learning to define the key aspects from the unstructured data formats. To address the problem of latent aspects without surface shape, Forouhesh and Mansouri [28] developed a new unsupervised learning method that outperforms **Latent Dirichlet Allocation** (**LDA**) [5], one of the best performing algorithms in this area.

As for customer prioritisation and ranking models, which I also consider in my research, the diversity of the available literature is much more sparse when compared to sentiment analysis. For example, in his paper, Cai [13] used the unsupervised learning method **k-means** [35] to first group all customers and then perform a ranking based on the weights assigned to the specific features that characterise the customer and are more important

[1] https://pytorch.org/

for the business model (in this case, the financial sector). Other research, unlike the previous study, favoured **recurrent neural networks** [91], but the main purpose was not simple grouping, but rather the detection of bot conversations that are more likely to lead to complaint escalation. Wang, Xie and Goh [85] took a completely different approach to customer prioritisation, which has nothing to do with machine learning. They compared the **House of Quality (HOQ)** [36] with the **Analytic Hierarchy Process (AHP)** [82] and concluded that if accuracy dominates over other parameters, AHP would be a preferable model, but if difficulty, time and cost are the main requirements for improvement, then it is more advisable to go for another method.

To the best of my knowledge, there is no study that offers a solution to the challenges in the tourism sector, provides a more advanced CRM system for analytics and management purposes, combines the tasks of overall sentiment detection (as well as classification), ABSA and prioritisation model. Furthermore, I found out that there is no free and open source system where users can create their own customisable ranking model, adjust the weights both positively and negatively, and not just follow someone else's pre-defined assumptions and guidelines, possibly even from a different domain, which may be of little interest to the parties involved.

3 Data

In my thesis, I used the Kaggle dataset[2] which contains more than 500,000 customer reviews about hotels in Europe scraped from Booking.com[3].

The dataset has undergone some preliminary cleaning. For example, punctuation marks were removed from the reviews. As the positive and negative parts of the comments were separated according to the structure of the booking reviews, the publisher of the data already calculated the number of words each part consisted of. All other features were original: hotel name and address, geographical location, average score and total number of reviews given to the hotel, some general information about the customer such as the country the reviewer is from and activity on Booking.com (measured by the total number of reviews written by that particular person), as well as trip-specific information, duration and purpose of stay, travel companions (if any) and date.

My personal preference for this data is based on the fact that it contains some additional details about the user and the hotel as the subject of the review, which could provide additional insight into the sentiment analysis and customer prioritisation system.

The first thing I did, before I got deeper into building models and deriving sentiments, was exploratory data analysis. It is an important task to understand the dataset and have a clear picture of what you are dealing with. As mentioned above, the data I have used only includes reviews of European hotels, so I took a closer look at the countries and cities where the hotels are located. In fact, there are 1492 unique hotels in 6 popular European tourist destinations: Italy, Austria, France, Spain, the Netherlands and the UK. As you can see in Figure 1, more than half of the reviews were written about British hotels, especially about hotels located in London. All other countries account for around 50,000 reviews each.

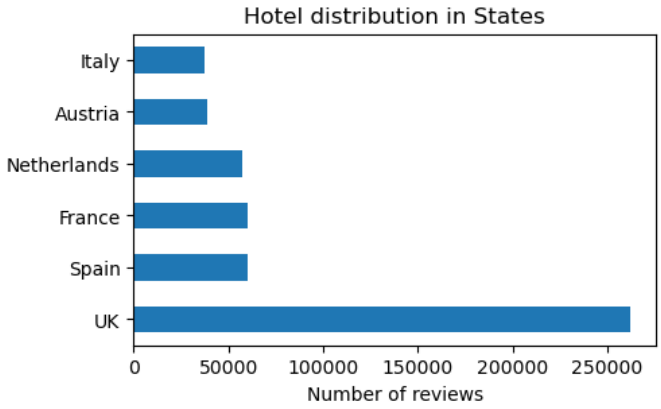


Figure 1: Hotel distribution in states

Since the dataset contains not only the address but also the exact geographical location of each hotel in

[2]https://www.kaggle.com/datasets/jiashenliu/515k-hotel-reviews-data-in-europe
[3]https://www.booking.com/

latitude and longitude, I was able to plot the resorts on a map using **plotly.express**[4] (data visualisation package allowing creation of interactive plots). One of the convenient features of this map is that it can be easily zoomed in and out to get a more precise or more general view of the data. In addition, as shown in Figure 2 zoomed in on Vienna, two other features besides the location, 'Average Score' and 'Additional Number of Scores' have been visualised in different colours and sizes.

Another location-related feature in the dataset is the 'nationality' of the reviewer, which is essentially the person's permanent place of residence. Although the hotels represented on Kaggle are all European, the reviews come from travellers from all over the world. The problem with visualising the countries was that there were only state names in the data, so I had to do additional work like extracting the country code from the name and then the central geographic location coordinates to be able to plot them on the map (Figure 3).

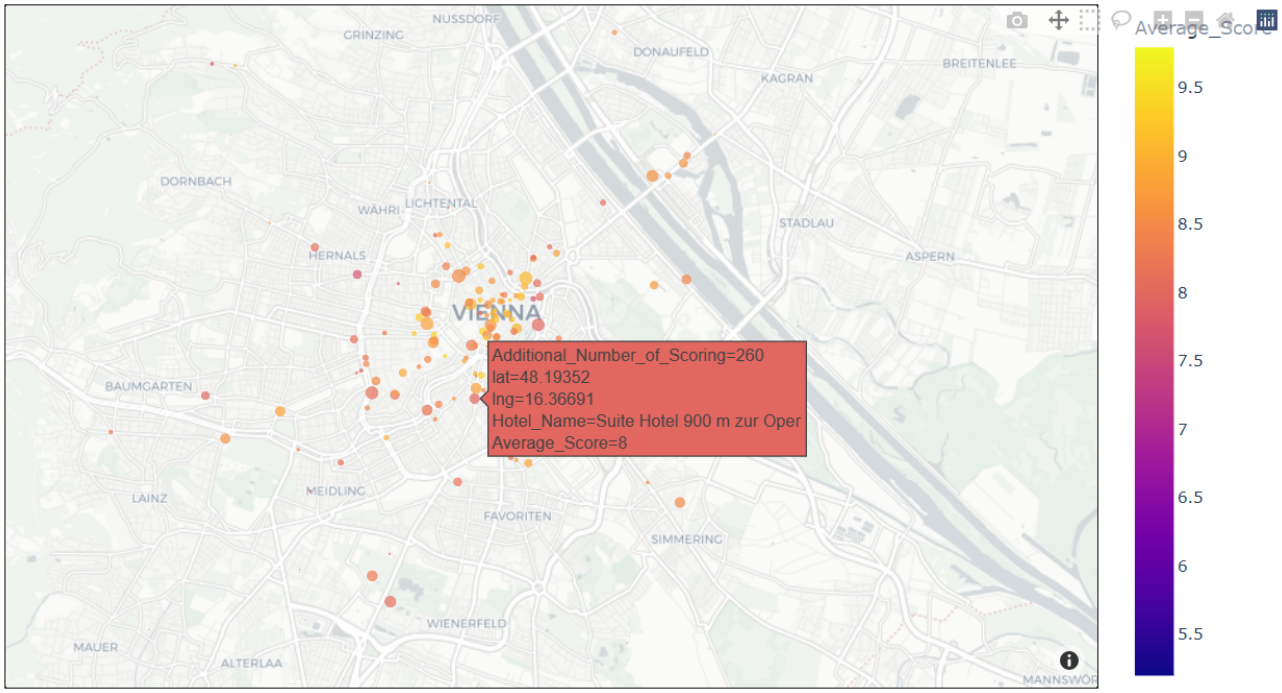


Figure 2: Map of the hotels in Vienna

Looking more closely at the reviewer and total scores, the highest average hotel score in the dataset is 9.8, which is the Ritz Paris, while the Liberty Hotel has the lowest total score of 5.2. Although the latter score is quite low, looking at the distribution of reviewer scores on the left hand side of Figure 4, it is clear that highly rated resorts (scores >7) are quite dominant in the market and account for the majority of the reviews given. This can also be seen in the hotel ratings. I have manually divided all ratings into 'Excellent' (>8.5), 'Very Good' (>= 8), 'Good' (>7), 'Average' (>5) and 'Bad' (everything else) to see if I am indeed dealing with mostly

[4]https://plotly.com/python/plotly-express/

highly rated hotels, which means that they are most likely to have predominantly positive reviews and aspects. As you can see in the pie chart on the right side of Figure 4, more than half of the hotels are rated positive overall (>7), which makes the work more difficult as the dataset is highly unbalanced.

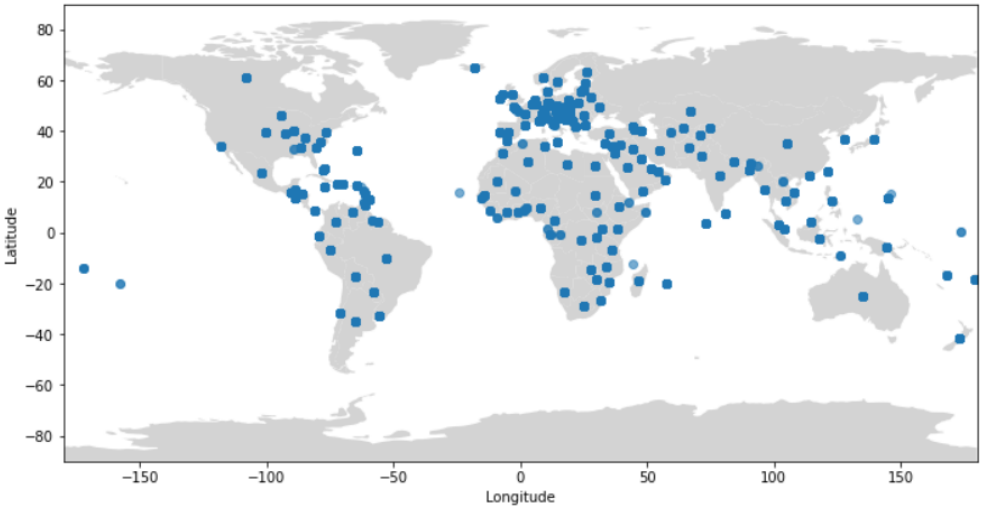


Figure 3: Reviewers' residency

After that, I wanted to look at other reviewer-related characteristics such as tags, which is an aggregated feature of length of stay, number of people travelling, purpose of trip, etc. In my case it is a string with all the details enumerated one after the other. To extract the meaningful features from it, I had to use the **re** Python library[5] and write a query to get the meaningful data and store it in a list format. This allowed me to use the **nltk.FreqDist** function, which is a part of **nltk** library (see details in Subsection 4.2.1) and computes a frequency distribution, to display the most common tags for users. As you can see in Figure 5, in the majority of cases the reviewers were on a leisure trip, with their spouse, spending 1 or 2 nights in a hotel and submitting the review from their mobile device. This data can be useful as it may have some influence on the review and can be used as one of the features in the prediction models if it proves to be meaningful enough during the research.

One of the ways to do this is to build a correlation matrix and identify any particularly salient connections between the features (especially something related to the reviews). This type of graph only processes numerical features, so the review itself cannot be used, but the number of positive and negative words can be used for this purpose. A column that I think could also be useful is 'Days Since Review', but it was originally stored in string format. It required some extra processing, such as deleting the alphabetical part and converting it to a number. It was then automatically processed and included in the correlation matrix. As I described above, I was able to derive 10 most common tags that I need to insert into the table (I use **pandas dataframe**[6] – 2 dimensional

[5]https://docs.python.org/3/library/re.html
[6]https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

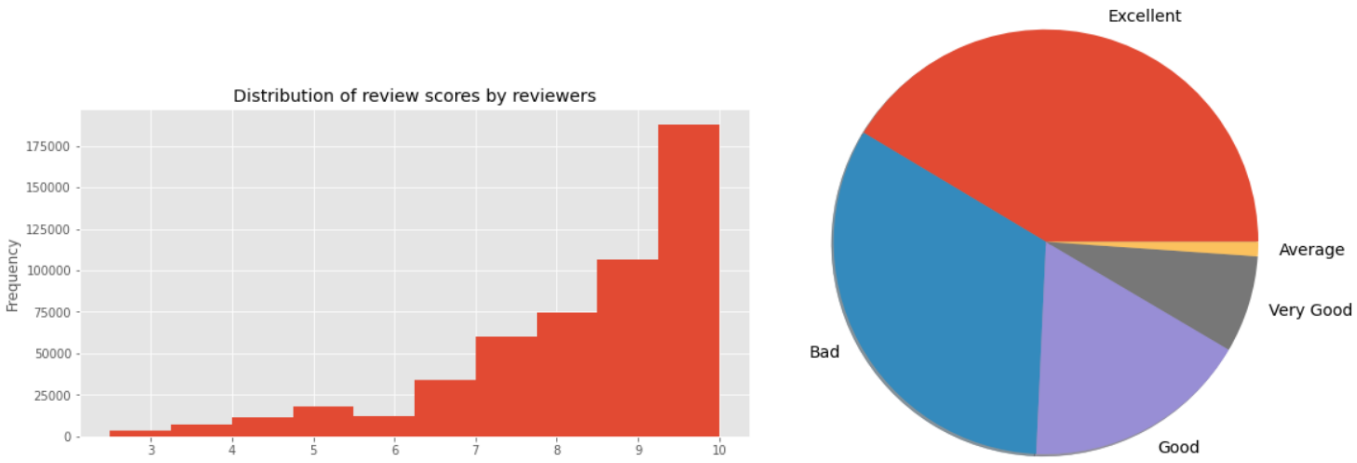


Figure 4: Distribution of reviewer scores by reviewers (left) and hotel ratings (right)

data structure, in my case table with rows and columns, for the exploration part) as True or False if the tags are originally included in the feature or not. The results of the correlation matrix are shown in Figure 6.

As the dataset is large, any work that was done directly in pandas tended to take a long time. In order to improve performance, it proved more efficient to work with more familiar Python data structures and convert everything to lists in this particular case. By doing this and using the **literal_eval** function from the **ast (Abstract Syntax Tree)** package[7] (a set of tools for working with abstract syntax trees or in other words hierarchical representation of the structure of a program), I was able to insert 10 additional columns with common tags much faster.

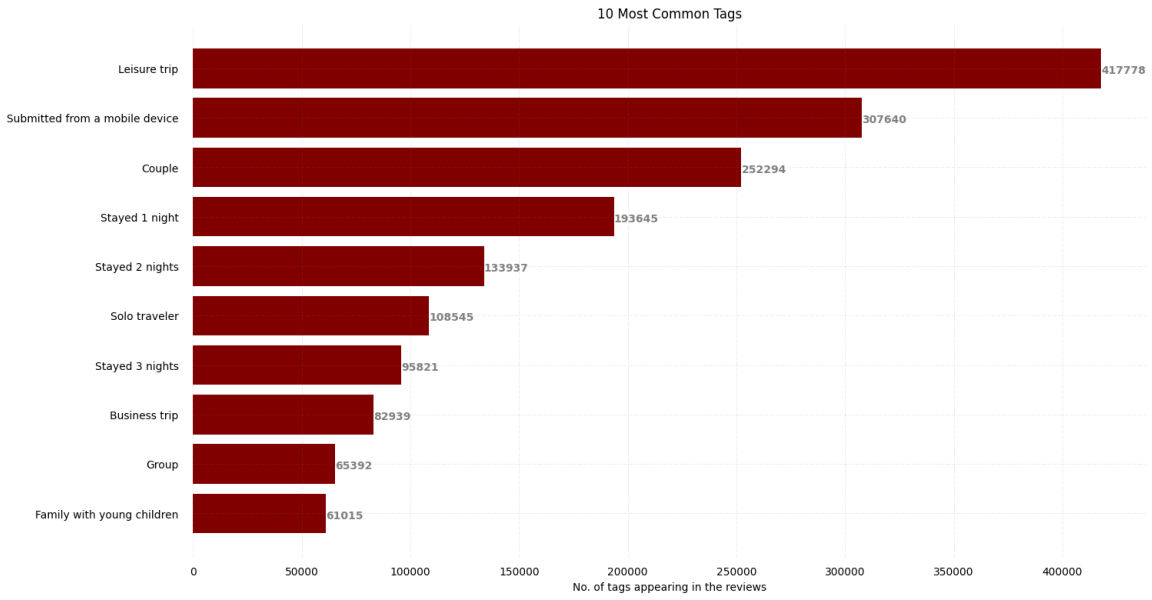


Figure 5: 10 Most common tags the reviewers specified while leaving the review

[7]https://docs.python.org/3/library/ast.html

There are some obvious correlations: the additional number of scores (pure score without written comments) and the total number of reviews have a strong connection with each other because they actually share almost the same information, average and reviewer score could also be a good example. Another similar case could be a negative correlation between leisure and business trips, as they represent completely opposite purposes of stay.

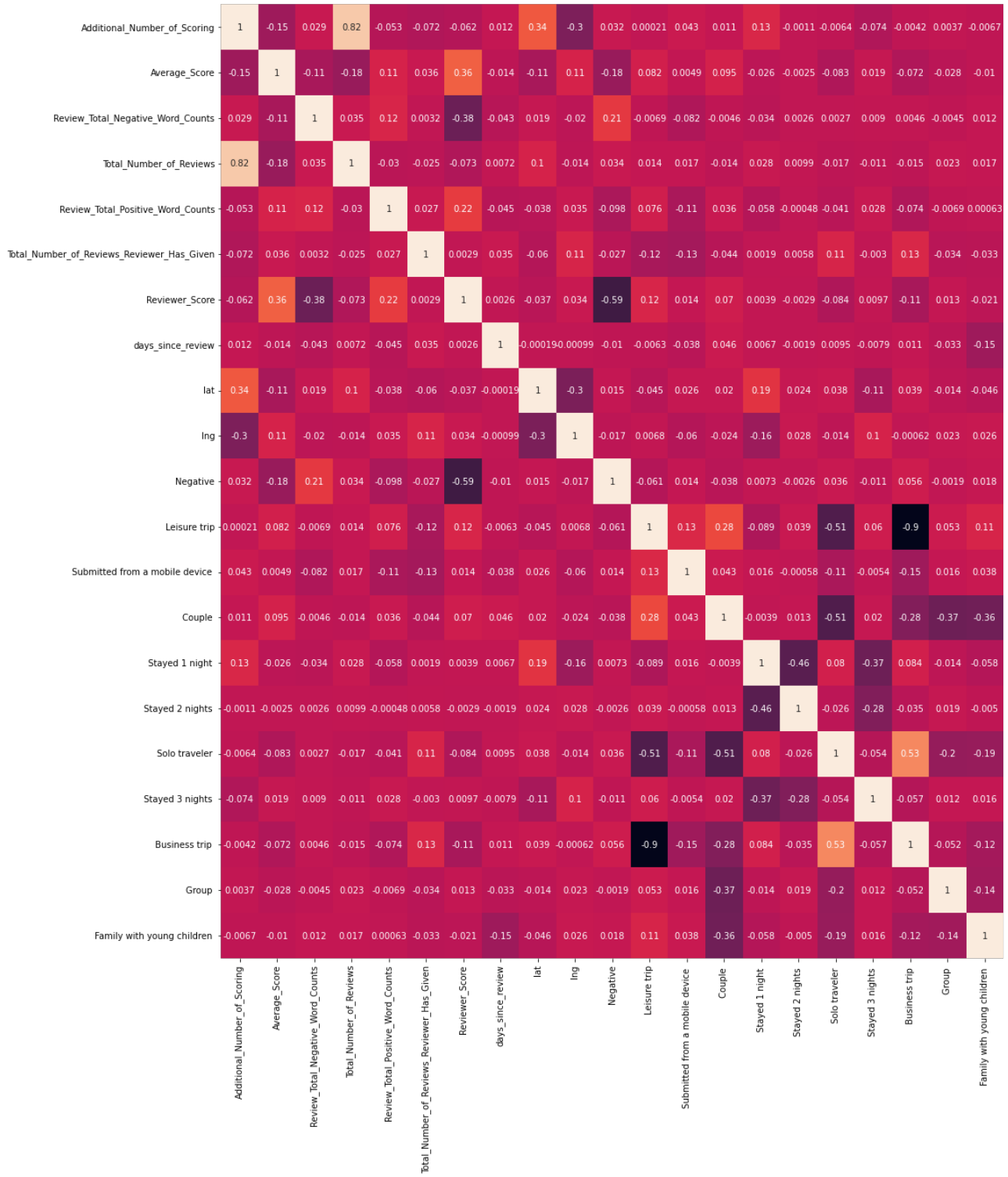


Figure 6: Correlation matrix

What I found interesting in Figure 6 are the following relationships of characteristics: the more words the negative part of the review contains, the more likely it is to be negative, but the same correlation between positive words and the review being positive is much weaker; people on business trips are more likely to leave reviews than tourists, but they tend to leave negative reviews more often than other travellers (in fact, the same is true for solo travellers, it can be explained by the fact that they are more likely to be on a business trip and have almost the same correlations).

In addition, reviewers who spend fewer nights at the hotel write more negative things about their stay, and the correlation increases steadily with the length of stay. Surprisingly, the same holds true for positive aspects, leading me to the possible conclusion that customers simply get to know the place better and are willing to write more the longer they stay. All other correlations seem to be totally insignificant or irrelevant to the work and therefore not worth further discussion.

The next thing I did is more closely related to the review. It is worth mentioning that for further sentiment analysis I had to concatenate negative and positive parts of the review. However, knowing both sides of the review separately helped in different situations.

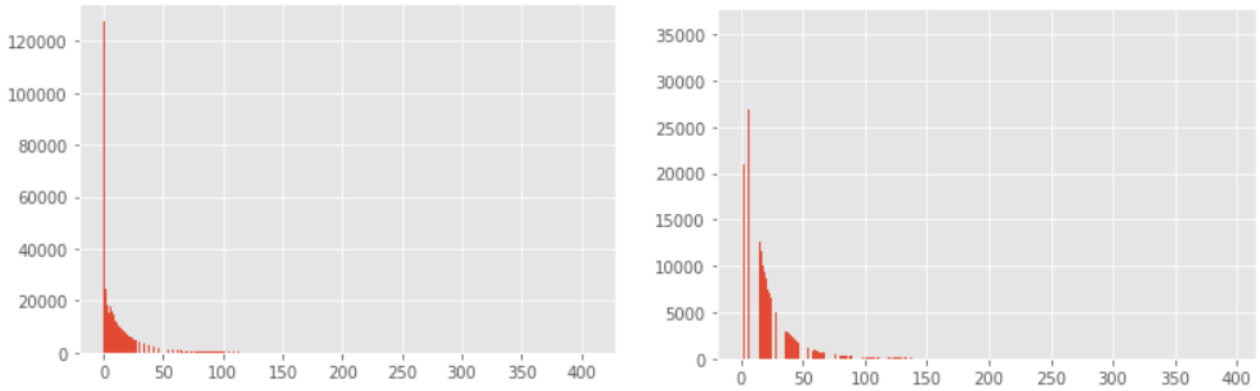


Figure 7: Distribution of negative (left) and positive (right) word counts

For example, I could plot the distribution of positive and negative word counts to get an idea of the overall tendency of the reviews, whether customers tend to write more about unpleasant experiences or praise the benefits. One of the issues that arose in visualising the distribution appropriately was the appropriate number of bins for histograms. According to the square root rule [24], one should take the square root of the total number of samples. Since I am dealing with a large data set, I ended up with 718 bins. From the histograms below (Figure 7) we can see that the number of '0-words' (nothing bad is written about the hotel) exceeds the number of non-positives by almost 6 times. And in general, once again, it is proven that the trend is that reviewers leave more comments about the advantages of their stay.

Figure 8: Wordcloud of the negative (left) and positive (right) aspects of the review

Furthermore, just by creating the wordmaps (Figure 8) for the pros and cons of the hotel, I was able to visualise the most frequently used words and phrases, which helped me to get an overview and further indications of aspect-based sentiments. If we compare the two graphs on the negative side, we can see the dominance of negations: *didn't*, *wasn't* and *don't*. Furthermore, among the recurring word aspects (*bathroom*, *room*, *hotel*, *breakfast*, *staff*, etc) that appear on both wordmaps, there are words with typically negative connotations: *small*, *expensive*, *noise*, *tiny*, *cold*, *nothing*, etc. Surprisingly, the positive words on the left (Figure 8) were no exception. *good*, *great*, *nice*, *helpful*, etc, albeit in a smaller font, but still clearly visible on the wordmap. One explanation for this may be that these words were originally used with negation (***not*** *good* or with a few more words in between ***not really as good as***), which may not be interpreted correctly by the **wordcloud**[8] Python package (which helps to determine frequency of a word in textual content through visualisation). On the positive side, there is obviously the wide use of positive aspects (often not just a single adjective, but rather in the form of 2-word collocations): *great/good location*, *helpful/friendly staff*, *comfortable/clean room*, as well as *loved*, *liked*, *lovely*, *perfect*, etc.

[8]https://pypi.org/project/wordcloud/

4 Methods

The aim of my work is to design a customer prioritisation model from scratch, based on the most efficient sentiment detection and prediction algorithms for the tourism domain, using reviews from Booking.com as training data. Taking into account the hardware limitations of my working laptop, I first tried different techniques that would encode the words of the review text, since most machine learning methods are not able to read pure text and require its numerical representation.

The next step would be an overall sentiment analysis of the whole comment. To do this, I experimented with four completely different methods and output techniques (see Subsections 4.2.6, 4.2.7, 4.2.8, 4.2.9): Vader, TextBlob, AFINN and SpaCy. Identifying the sentiment of the review based on the embeddings alone might be inaccurate or even biased by certain confusing word combinations that are easy for humans to interpret, but are completely puzzling for machines.

For this reason, as the next level of research, I also considered prediction models that would identify some patterns in the reviews, using features generated as a result of sentiment detection as input. Here I considered classic supervised and unsupervised learning methods as well as more advanced deep learning algorithms such as neural networks. I experimented and tried to find the combination of labeling and prediction technique with the highest model performance. The labeling was done based on the four sentiment analysis algorithms together with the score given to the hotel by the reviewer. I also did some fine tuning of the models and found the most efficient parameters that worked with my data.

I also considered models that focus on the review text and its aspects and do not require any preparation in the form of sentiment detection model results. I tried both widely used methods for NLP purposes and completely manually tailored ones. To improve the performance of the models, it is important to work with high quality data. Detecting the latent aspects turned out to be a good way to do this and to improve aspect detection and aspect-based sentiment detection respectively.

As for the prioritisation models that serve the business purposes of this study, here I decided to provide a variety of open source and free web applications for users to use according to their goals and business specificities. Therefore, I have developed three different models, which include: basic filtering, where hotel owners could easily sort out the reviews they are not interested in; a customer prioritisation model, specifically designed with the aim of prioritising negative experiences of travellers; and an adjustable re-ranking system, with the possibility of adjusting the weights of different features related to the reviewer and the review that would influence their overall score. Such applications could help hotel owners to identify bottlenecks in their business, to understand the direction of improvement and to avoid escalation of conflict with the dissatisfied

customers by responding promptly to the bad review.

4.1 Equipment Technical Characteristics

Before going into more detail about the models and techniques I used in my research, I would like to briefly describe the settings I had to work with. It is worth noting that this is not professional equipment or a supercomputer, but just a normal laptop for home use, which in some cases unfortunately became a major limitation in terms of computing time or running the model on the whole dataset.

My laptop runs the Microsoft Windows 11 Pro operating system. It is powered by an Intel Core i7-1065G7 processor with a clock speed of 1.30GHz and four cores providing eight logical processors. The laptop is a LENOVO model 20SM with x64 architecture. The BIOS version (Basic Input/Output System) is LENOVO DJCN25WW and the system operates in UEFI (Unified Extensible Firmware Interface) mode.

The laptop has 16.0 GB of installed RAM, with 15.7 GB of total physical memory, of which I had 8.81 GB available before starting work. The total virtual memory is 63.7 GB, with 56.3 GB available.

For storage, the laptop includes a 475GB SSD, providing plenty of room for the operating system, applications and user data.

According to the purpose of the study, the fact that I had these parameters actually plays a positive role, as many tourist facility owners run small or medium sized businesses and cannot afford to hire analysts to help to identify and eliminate bottlenecks, and simply have to deal with the issues themselves, usually using already existing home computers. This study proves that the models used are feasible for any average user.

4.2 Sentiment Detection Models

4.2.1 NLTK

NLTK[9] is a platform for building and processing natural language in Python. It provides more than 50 corpora from different areas such as inaugural speeches of US presidents, news, social media chats, books, etc. Moreover, this package offers many useful built-in functions for NLP, as well as text processing libraries for a variety of data manipulation tasks: tokenisation, lemmatisation, parsing, chunking, stemming, text classification, etc [54].

I used this toolkit primarily to clean up the review text. It was important to remove the stop words (such as articles, pronouns) that add no meaning to the review and just make it bigger, and to convert all the words to lower case to avoid confusion with the word repetitions. Normally many data processing tools distinguish between upper and lower case words, so without this step the same words could be counted twice in both variants.

[9] https://www.nltk.org/

In addition, transforming terms into their dictionary forms (lemmas) can help to clean up different tenses of verbs, plurals of nouns, coordination of pronouns with verbs (such as *do not*, or *does not*). All these types of words mentioned above, although written in a non-identical way, still carry the same meaning, so all the processing reduces the arsenal of terms I have to deal with.

After cleaning up the reviews, I decided to do some text exploration to get an idea of whether the comments were mostly positive or negative, what aspects were most satisfying to the customers, and vice versa. I used the **FreqDist** function, which allows you to identify the most frequent words in the text, and found the 10 most frequent words.

As you can see in Figure 9, there are a few adjectives that might indicate some positive sentiments, such as *good*, *great*, *friendly* and *clean*, but mostly the results contain nouns that are good in the role of aspects and can become a basis for aspect-based sentiment analysis, but are rather useless on their own.

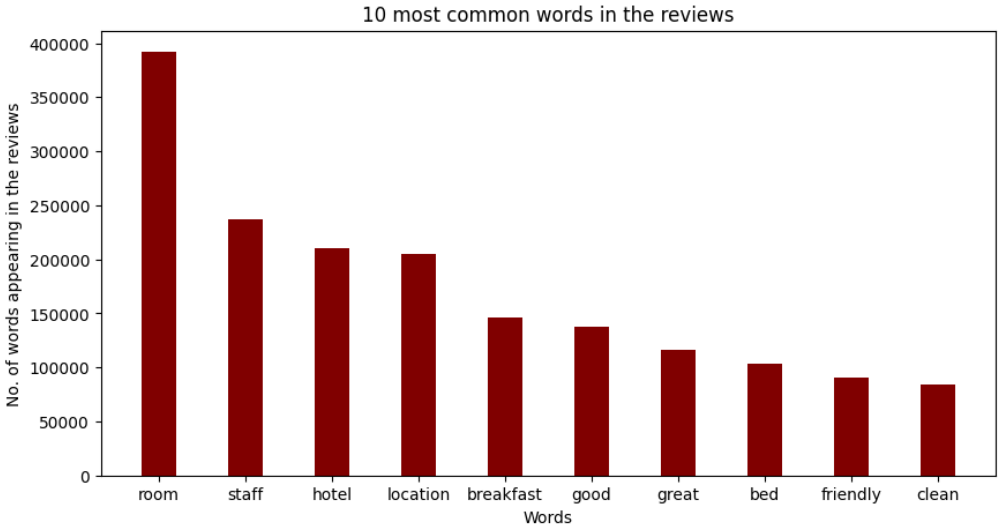


Figure 9: Most frequently used words in the reviews

In order to get more information from text exploration, I tried to increase the pool to 50 of the most common words, but got even more confusing results containing words like *would*, *really*, *one*, *get*, *need*, *nothing*, etc. In this case, these words cannot be helpful either as aspects or as sentiments.

The next thing I did was to identify the most common bigrams. Looking at Figure 10 it is clear that almost all the bigrams follow the same pattern: adjectives + nouns, and that they are used in positive contexts. The only negative bigram was *room small*.

Similar to the bigram method, collocations – phrases made up of several words that often occur together [53] – are also detected. To my surprise, this technique did not really give any meaningful results, as most of the phrases that appeared in the output are just common phrases used in everyday life: *air condition*, *value money*, *front desk*, *tea coffee*, *train* or *metro station*, etc. There were some useful collocations, such as *comfy bed* or

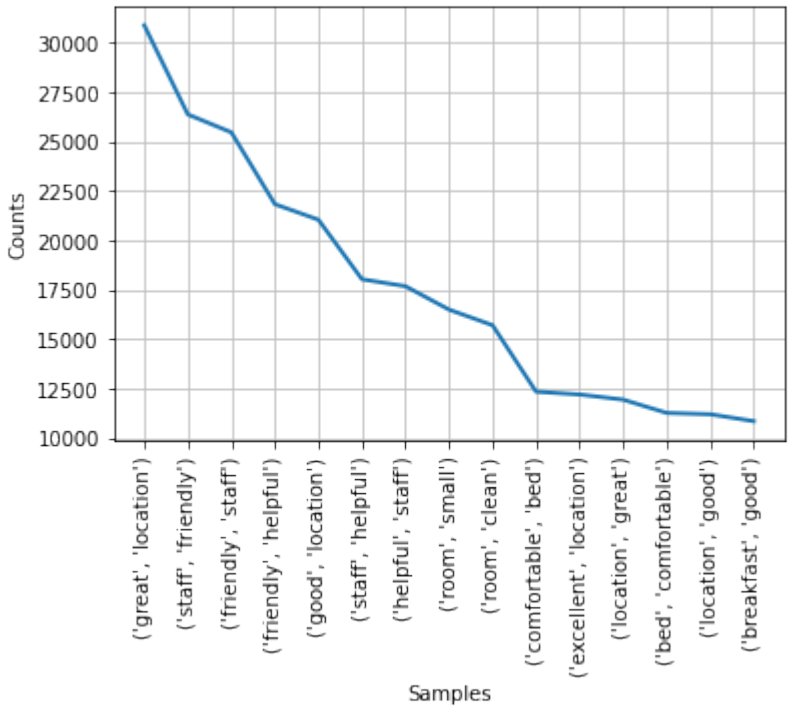


Figure 10: Most frequently used bigrams in the reviews

great location, but they are rather limited versions of the results of the previous method.

I also wanted to use POS-tagging to detect common words, as the combination of adjectives + nouns seemed to be the most meaningful so far. I identified 9 of the most common adjectives and used them as patterns for the tabulation, with the most frequently used nouns as conditions. In Figure 11, where you can see the nouns and adjectives as column names on the left, you can see that the most commonly used phrases together, such as *comfortable bed* or *friendly staff*, actually have the highest number of occurrences among all the other possible word combinations. Some odd phrases such as *friendly bathroom* or *clean staff* do appear together, but one of the possible explanations for this could be that the adjective was linked to the preceding word rather than the following word, as I considered in Figure 11. I deliberately chose the combination adjective + noun because it is more common than the other way round.

	good	friendly	great	small	clean	nice	big	comfortable	perfect
room	8660	2498	7021	16507	15717	7229	3061	7285	1573
staff	6633	26381	8146	621	3147	5906	249	1731	807
hotel	4855	2062	6403	682	4154	3026	388	1058	1920
location	11213	5610	11952	834	3054	3572	300	1788	7252
breakfast	10863	1400	4448	496	871	2760	251	741	571
bed	2964	791	2489	1498	1429	1714	701	11279	292
stay	807	296	1043	290	316	394	79	479	413
service	1622	549	1275	115	389	413	58	197	204
bathroom	1084	228	836	1758	1540	867	346	319	124

Figure 11: Tabulation of the most common adjectives towards the most common nouns

The exploratory research on the concatenated reviews could show me the possible directions to go with general and aspect-based sentiment analysis, and once again proved that I am dealing with an imbalanced dataset where almost all the reviews are positive, and it will be the main challenge to infer characteristics of the negative comments.

4.2.2 CountVectorizer

CountVectorizer is one of the techniques that deals with unstructured textual data and transforms it into numbers by encoding each word in the sparse matrix [59]. Basically, in this method the word occurrence of each text considered is written in the form of 0 (word is not in the text) or 1 (word is in the text) and then counted, so the principle of CountVectorizer is to find the most frequent words or n-grams. The technique has an option to display 'vocabulary', which in fact gives a dictionary of the most frequent words as output.

In Figure 12 you can see that I have limited the vocabulary to the 20 most common bigrams and trigrams. The numbers can be mistaken for frequency, but they represent the position of the words in the matrix [41].

```
{'booking com': 2,
 'small room': 16,
 'tea coffee': 18,
 'breakfast included': 4,
 'little bit': 9,
 'hotel room': 8,
 'room service': 14,
 'room room': 13,
 'room little': 12,
 'star hotel': 17,
 'mini bar': 10,
 'room small': 15,
 'breakfast expensive': 3,
 'air conditioning': 0,
 'double room': 7,
 'double bed': 6,
 'bit small': 1,
 'room bit': 11,
 'wi fi': 19,
 'breakfast room': 5}
```

```
{'great location': 11,
 'location good': 13,
 'good location': 10,
 'friendly staff': 8,
 'good breakfast': 9,
 'walking distance': 19,
 'comfy bed': 5,
 'comfortable room': 4,
 'staff friendly': 16,
 'friendly helpful': 7,
 'staff friendly helpful': 17,
 'helpful staff': 12,
 'clean room': 2,
 'staff helpful': 18,
 'bed comfortable': 0,
 'room clean': 15,
 'comfortable bed': 3,
 'location great': 14,
 'breakfast good': 1,
 'excellent location': 6}
```

Figure 12: The most frequent negative (left) and positive (right) word combinations in the reviews derived by CountVectorizer

First of all, I had to clean the reviews before applying the method described above, because it recognises upper and lower case words, for example, and can store them as different samples in the sparse matrix. Stop words such as articles or pronouns are also meaningless as they do not help to understand the sentiment of the reviews. In addition, lemmatisation was applied, which helped to eliminate many repetitions when the reviewer slightly modified the word according to grammatical rules (past tense, plurals, gender, etc) by converting the words to their dictionary form.

After the cleaning step, when the data was ready for processing in CountVectorizer, I specified the stopwords

and set the analyser to 'word' (tokenisation is automatically done at word level). The same procedure was repeated several times with different variations of n-grams on the concatenated reviews and on the positive and negative components separately. Unigrams were too messy and did not give at least a preliminary understanding of what customers liked and disliked most, while bigrams included both the aspect and the sentiment, usually in the form of adjective and noun. Trigrams were also used as part of the experiment, but in their 'pure' form produced only ambiguous output such as *air conditioning work*, *room small bed*, *breakfast included price* and so on. The combination of the two gave the best result (Figure 12).

To my surprise, the positive parts and the reviews as a whole gave almost the same result, therefore in order to go deeper into the aspects that made customers dissatisfied, I would only look at the positive and negative details separately for comparison.

In terms of benefits, customers were most satisfied with the location, helpful and friendly staff, clean room, comfortable bed and good breakfast. It was very difficult to get meaningful results for complaints. From Figure 12 it was only possible to identify two adequate parameters: small room and expensive breakfast. In this case, as in an exception, I got some information from trigrams: *room little small*, *room extremely small*, *room bit small*, etc – which is actually just an extended example of negative bigrams. This data will be used as a basis for aspect sentiment analysis, as a pool of aspects and possible sentiments has already been created.

4.2.3 Word2Vec

Word2Vec is an NLP technique from the **gensim**[10] library (widely used in topic modelling, indexing and performing similarity retrieval using extensive collections of data) introduced in 2013 [50]. It uses a neural network model to identify word associations from a large collection of text. After training, the model can identify similar words and make recommendations to fill gaps in sentences. Word2Vec is also a widely used implementation of word embedding (a representation that allows words of similar meaning to share a similar representation), which creates a distributed representation of words in numerical vectors. By transforming text into vectors, this technique captures the meaning and connections between words [48].

Before applying Word2Vec to the reviews, I first tokenised the text of the comments using **WordPunctTokenizer** from nltk. As a result, Word2Vec transformed these words (tokens) into 32-dimensional vectors that are difficult to work with. They could also prevent an easier visual representation of the model, simplified analysis and faster computation time, especially for a large corpus.

Therefore, I had to reduce the dimensionality to simpler 2 dimensions, which was done by **StandardScaler** [59] and **PCA** (Principal Component Analysis) [40] techniques. The former performs tasks such as standardising

[10] https://pypi.org/project/gensim/

the features by removing the mean and scaling to unit variance. This is an important prerequisite for some machine learning estimators, which can perform poorly if the individual features do not look like standard normally distributed data. PCA, in turn, identifies the most significant features of the dataset and then uses them to create new variables – principal components – that capture most of the original variance.

The results of word embedding in 2-dimensional space can be seen in Figure 13, which was created using the interactive visualisation library **Bokeh**[11]. All the words used in the review texts were plotted on this graph according to the pattern that neighbours are close together. For example, dots on the left represent past tense verbs, and so on. However, embedding at the level of individual words can be too granular and difficult to interpret.

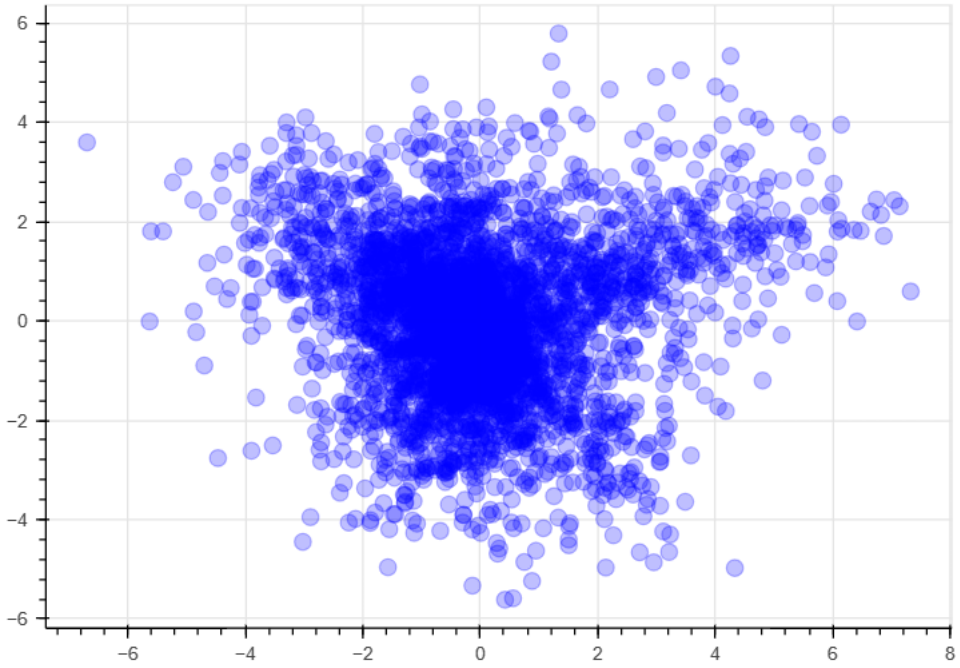


Figure 13: Word2Vec with reduced dimensionality (single word-level)

So the next step was to group the words using another library, **UMAP**, which stands for 'Uniform Manifold Approximation and Projection for Dimension Reduction' [49]. It is similar to PCA, but has some additional advantages, such as preserving the global structure of the data. It works by creating a low-dimensional representation of the data, while trying to preserve the relationships between data points in the original high-dimensional space. In addition, UMAP is computationally efficient and can handle large datasets, which is a big advantage for my data.

As can be seen in Figure 14, the graph is now denser, as it combines the neighbours under a single point. A good example could be an outlier on the left side of Figure 14, which represents all auxiliary verbs with negation grouped together (due to the lack of punctuation *'t* negation structures are recognised as separate tokens).

[11] http://bokeh.org/

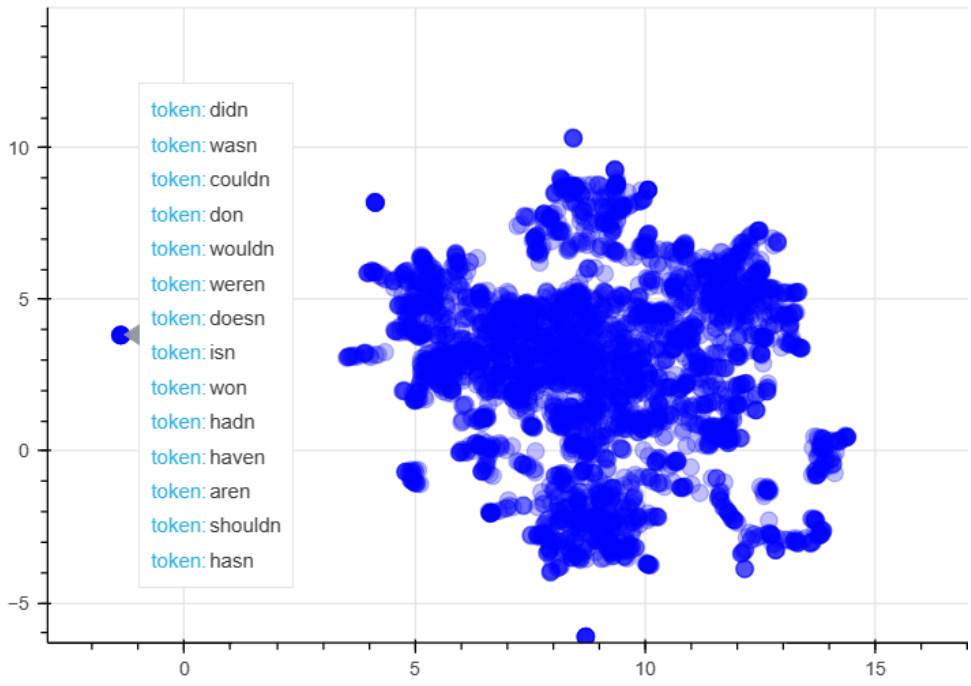


Figure 14: Similar Word2Vec vectors grouped

Another way of representing the text would be phrase-level embedding. This can be achieved in the same way as applying Word2Vec to each word in the comment, while the results of the whole text would be stored together as a **numpy array**[12] sequence (a grid of values, indexed by a tuple of nonnegative integers) of separate word vectors. Depending on the length of the comment, the results obviously differ, so as in the previous case, I used UMAP dimensionality reduction and only considered the first 50 words of the reviews. In Figure 15 you can see that all sentences related to compliments or free upgrades were grouped together. There is also an easily noticeable error made by the algorithm: the phrase *room not ready on arrival* does not belong to this bundle of positive sentences and has a negative meaning. Nevertheless, Word2Vec is a good way to obtain an embedding of words and even whole texts, and could serve as a basis for unsupervised learning techniques such as clustering, thanks to the detection of similar semantics.

4.2.4 Doc2Vec

Doc2Vec is the **gensim** library technique for representing the text document in the form of a vector and is also a generalisation and more advanced version of the Word2Vec tool (obtaining the word embedding from the whole corpus) [83]. In this method, the word embedding is derived from the paragraphs of the corpus, so that each paragraph is mapped to a unique vector. In fact, Doc2Vec learns the vector representation of text while vectorisation is applied to the small pieces of text documents (phrases, sentences, documents, etc). The same

[12]https://numpy.org/doc/stable/reference/generated/numpy.array.html

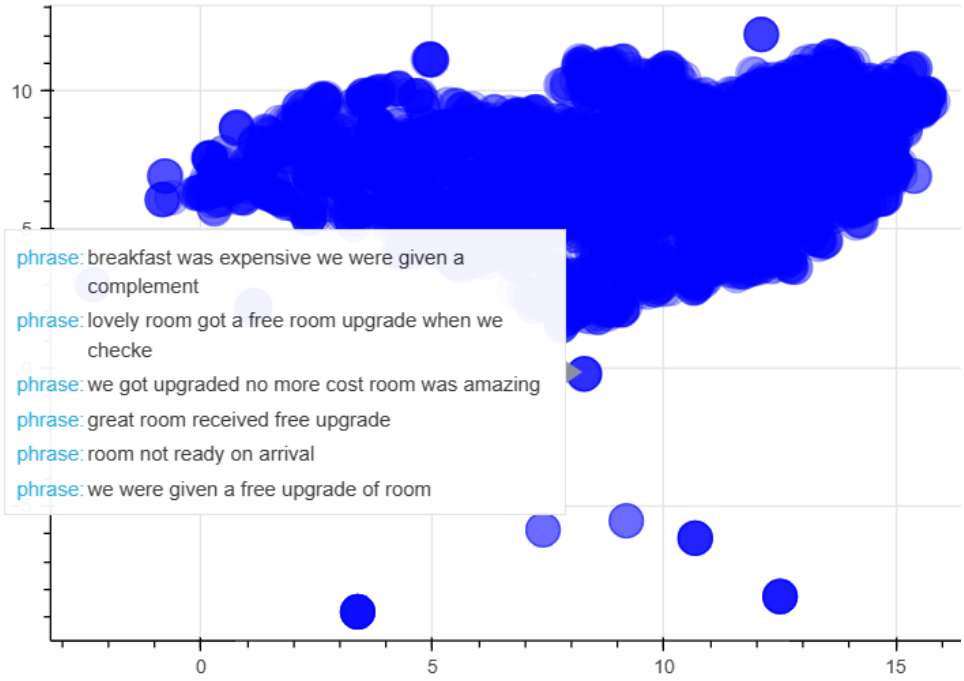


Figure 15: Word2Vec representation (sentence-level)

texts will have similar representations. This is often used to predict words in paragraphs. One of the main features of Doc2Vec is that the next word can be predicted using the paragraph vector, where it also has some context, information about the semantic relationship and can improve the model results.

I used Doc2Vec on my unstructured text data to convert it into the numerical features that can be used in the prediction models that do not recognise raw text. Before that, I did some cleaning up of the reviews, converting all words to lower case, removing all numbers and non-alphabetical characters (the dataset was already pre-cleaned and did not contain any punctuation), tagging the words using the POS-tagger from the nltk corpus, which was the basis for a **wordnetLemmatizer**, so that the words were also converted to their dictionary forms. I set the parameters of Doc2Vec to vector size=5, window=2 (maximum distance between the current and predicted word within a sentence[13]), minimum number of words=1, workers (threads to train the model) =5 and epochs=30. As output, each review received 5 Doc2Vec vectors, which are the results of shallow neural networks. These newly generated features will be used in prediction models.

4.2.5 TfidfVectorizer

Tfidf (Term Frequency Inverse Document Frequency) is an algorithm calculated by multiplying the word (term) frequency by the inverse document frequency [16]. To simplify the definition, it is used to measure the originality of words by comparing the number of texts containing these words with the frequency of the word in the given

[13]https://radimrehurek.com/gensim/models/doc2vec.html

text.

Actually it is built on the same principle as the CountVectorizer mentioned above in Subsection 4.2.2 to represent the words or text in numerical format, however it is rather an extension of it because it also depends on the 'weight' or 'importance' of the word and not just on how often the term occurs in the text.

The positive side of this approach is that the most commonly used words sometimes do not bring any useful information to the analysis, while words that occur less frequently may be much more meaningful, and TfidfVectorizer helps to derive them and count their relative importance [16].

This method also offers the option of stopwords, but I applied it to the cleaned reviews, so there was no need to resort to additional parameters. I also set the minimum frequency of the term to 100,000 (so the word should appear in about one fifth of all reviews). I made this number high on purpose. Given that there are already many useful columns in the dataset, and that I have manually created additional ones as a result of sentiment detection or review analysis, creating even more than 100 or even 1,000 extra features could lead to model overfitting. More words might also make sense.

As described in Subsection 4.2.1 devoted to nltk package, where I discovered the most common words from all the reviews, the 8 most popular words with a frequency of over 100,000 were *room*, *staff*, *hotel*, *location*, *breakfast*, *good*, *great* and *bed*. After comparing them with the Tfidf output, which also had the same frequency parameter, I could see that the words *great* and *bed* were not identified as 'important', even though they fit the model with the frequency.

As an output of TfidfVectorizer I got the weights of the words mentioned above for each review (in the reviews where the term was not present, this number is 0). Furthermore, this new data will be used in the prediction model, as it is a good representation of the text from the 'importance' side of the frequent words.

4.2.6 Vader

Vader (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool that is part of nltk (detailed overview of this module see in Subsection 4.2.1) [47]. It provides scores of the text by evaluating the individual words. This tool is widely used for semantic orientation labelling as it generates the dictionary with 4 keys: positive (pos), negative (neg), neutral (neu) and compound – which score the text in each of these three dimensions, with the last one being an overall score. It is known for recognising emoticons, capital letters, punctuation, etc, which makes it popular for social media analysis.

Vader does not require any pre-training like machine learning techniques, making it easy to use with unlabelled data. This sentiment analysis tool can be used in combination with and as a basis for more advanced methods, most of which require some training data for further learning [73].

I used Vader's **SentimentIntensityAnalyzer** on my uncleaned concatenated reviews, as it is able to detect even small changes in emotionality (such as capital letters). Processed reviews that do not contain stopwords, numbers, lower case and lemmatised characters lose the part of the information that is responsible for the correct labelling, so they are used, for example, as a basis for aspect-based sentiment detection, where the amount of text is artificially reduced to a minimum just to get to the 'important' words (like aspects) and reduce the workload.

Just for the sake of experimentation, I also tried applying polarity scores to the cleaned reviews, but the results were different. In the case of the processed text, negative reviews were often misinterpreted and qualified as positive (compound >0). The average compound score of the original reviews was 0.4, while the average of the cleaned reviews was 0.52, meaning that more texts were classified as positive when they had been processed. Therefore, for my further models, I only used the original data scores, with the compound score being useful not only as one of the most informative features, but also as a label for model training.

4.2.7 TextBlob

TextBlob is a library in Python that is widely used in NLP. Apart from sentiment analysis, it provides many other text processing options such as POS-tagging, tokenisation, spell checking, parsing, word and phrase frequency, etc. It is a lexicon-based approach, so sentiment is defined by the semantic orientation and intensity of each word in the sentence [69].

Basically, each word gets its own individual score. The final result is then calculated by taking the average of all the words. This method returns two metrics related to the text under consideration: polarity and subjectivity. The first gives a sentiment output in the range from -1 to 1, where -1 is strongly negative and 1 is strongly positive. TextBlob stores some semantic labels on which the whole analysis is based. The subjectivity parameter indicates how much personal opinion and factual information the text contains. The range is still the same, with 1 being exclusively opinion-based factual information. One of the helpful features for determining subjectivity is the so-called 'intensity', which measures the influence of the word on the next (for example, modifiers such as adverbs).

However, the problem with both polarity and subjectivity is that if there are positive and negative aspects in the text, the weighted average approach may end up with a 0, which would misleadingly imply that the text is neutral, when in fact it would only be describing two sides. Moreover, some researchers like M.Kumar Barai in his article [4] where he compared the accuracy and confusion matrices of Vader and TextBlob doubt the efficiency of the latter and claim that Vader proved to be more advanced and precise for sentiment detection. I decided to try TextBlob with the tourism domain and compare the results with other sentiment analysis tools.

After applying TextBlob to my dataset and displaying the statistics of the new features, I can already say,

based on the frequency of positive scores, that this method classifies more reviews as positive compared to Vader, which could be confusing for prediction models that have difficulty working with imbalanced data. In this particular case, TextBlob could widen the gap in the distribution of positive and negative reviews and consequently worsen the accuracy of the predictions.

4.2.8 AFINN

AFINN is a wordlist based approach to sentiment analysis developed by Finn Arup Nielsen in 2009 [56]. This tool is specifically designed for short informal text found in internet forums and microblogs, where new/slang words or abbreviations are widely used. The author's paper describes the tool as having a high potential for sentiment analysis of Internet text, outperforming **ANEW** (Affective Norms for English Words) [8] which store the pre-defined sentiments of the words regardless of the context and **OpinionFinder** [87] which automatically detects the subjective opinion of the author from the text. This method, although not part of any library, was presented in some of the researches on NLP considered in the literature review chapter. I also found it to be very easy to use compared to all the techniques described above. AFINN supports many languages, as well as punctuation and emoticons, which makes it universal for all online textual data.

After applying it to the original reviews, I got a numerical score ranging from -49 to 87, with more negative reviews being more negative and more positive reviews being more positive. The average score was 5 with the quantiles of 25%, 50% and 75% being 1, 4 and 8 respectively, showing that the majority of reviews are more or less neutral with a tendency to be positive.

The only drawback I noticed is that the tool's output is limited to the overall sentiment of the text, which can be compared to the combined (overall) score of Vader and TextBlob polarity. However, other methods include a variety of additional metrics such as subjectivity or positive/neutral/negative scores. In general, the more meaningful features (not just any features, if they are not backed by relevant information they can simply lead to overfitting) that are included in the prediction model, the better the accuracy of the model. In the case of AFINN, there is only a subjective overall score, which may limit the results of the model, but can still serve as a good basis for providing labels for the dataset.

4.2.9 SpaCy

SpaCy[14] is an open source library written in the Python and Cython programming languages, specifically designed for production use in NLP [1]. It contains many pre-trained pipelines for a wide variety of purposes, from tokenisation, POS-tagging, dependency parsing, etc to word vectors, named entity recognition (NER) in

[14] https://spacy.io/

more than 20 languages, word vectors, sentiment detection, machine learning models, and so on. SpaCy can also be used to visualise syntax and NER, which is very useful for the first steps of data exploration.

I used this package in a universal[15] way, namely as one of the tools to classify the review and to better understand the data. The first and basic step done with the help of this package was to calculate the number of sentences in the review. Considering that I am working with the Kaggle dataset, which is already pre-cleaned and contains no punctuation marks, this is not an easy task. Therefore, I used SpaCy and its ready-to-use pipeline **en_core_web_sm**, which is actually used in almost all other data processing steps with the help of SpaCy. The accuracy of this sentence identification process is obviously poor due to the lack of punctuation, but it helps to get an understanding of the length of reviews and could be very useful for prediction, e.g. determining the tendency for larger reviews (with more sentences) more likely to be negative, as people may be more willing to describe their negative experience, or vice versa.

Another important spaCy feature used in my work is **displacy** visualiser[16] in style **dep**, which shows the syntactic dependencies and POS tags of the words. As you can see in Figure 16, the parser correctly identified things like *decoration* as a nominal subject, coordination in relation to the conjunction *and*, the conjunct between *stylish* and *warm* as adjectives connected by a conjunction, etc.

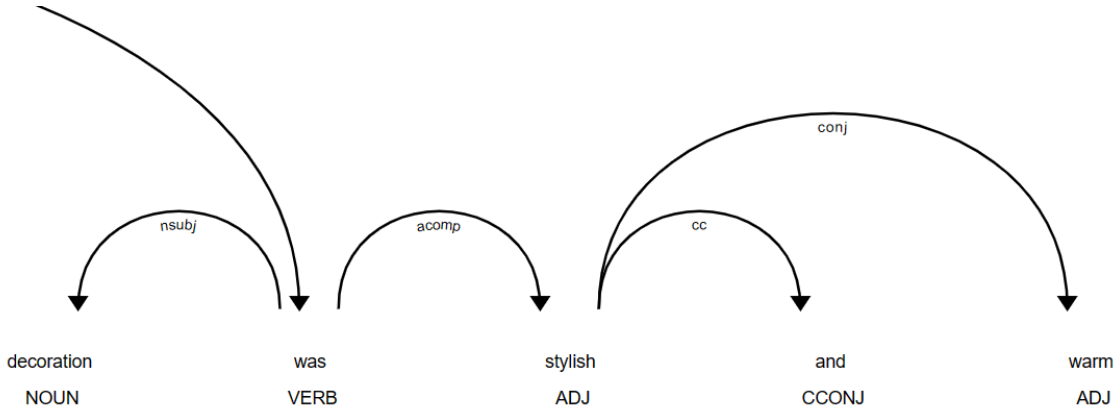


Figure 16: Visualisation of syntactic dependencies of the review

Displacy makes visualisation adaptable to purpose and style, and is not restricted to dependencies alone, as it can also be used with spans and entities. NER itself is a crucial task when dealing with large data sets. The extraction of review entities helps to sort unstructured data and detect important information related to places, brands, time, so I decided to examine my data and identify the most common built-in spaCy named entities of the reviews to see if there are any interesting patterns. An example of a named entity (GPE – geographical or political entity) appearing in one of the comments is shown in Figure 17.

[15] https://spacy.io/universe
[16] https://spacy.io/universe/project/displacy

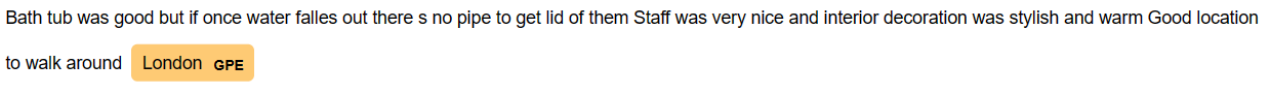


Figure 17: Named entities of the review

After running the analysis on the most popular named entities in the cleaned reviews (no numbers, all lowercase), I found that cardinal and ordinal entities were used more than 10,000 times. This result may be confusing, as both types of entities require numbers and I performed the analysis on the cleaned data, but still there are many numerical aspects written with letters in the reviews, such as *one*, *two*, *first*, *second*, which actually have no meaning for my research.

Apart from these, there were also many date and time entities in the top 50, such as *night*, *next day*, *Sunday*, *summer*, *week*, etc. After filtering out all the above labels together with quantity, there were only a few named entities left: GPE, Person, Product, Money and Percentage. For the latter, in most cases there was something similar to *one hundred percent* or *one thousand percent*, which is more likely to indicate customer satisfaction. When it came to locations, there were a lot of mentions of London and the UK, although I also came across India, California, Houston, the Middle East and Europe. There were also some totally inappropriate words such as *ndndndndndnndndndndndndbdbdbdbdndndndhdhdhdhdhdhhdhdndndndndndnbdndndndndndnd* or *xllnt* which were mistakenly classified as GPE even though they are obvious spelling errors.

In addition, this NER tool had other errors that were easy for humans to spot. *Half moon* got an entity percentage, *mm* – person, *receptionist* – NORP (nationalities or religious or political groups), which is obviously not the case in the tourism domain. Overall, despite the errors, the identified entities are quite important as they can indicate the length of stay, specific time, location, when and where the guest visited the hotel and can be used in predictive models or aspect based sentiment analysis.

The next useful built-in spaCy feature is sentiment detection. This algorithm is based on machine learning and extracts the patterns directly from the data, so it can handle a wide range of text types and contexts. In addition, spaCy uses a **convolutional neural network (CNN)** [34] architecture that is trained on a large corpus of annotated text data.

In spite of this, in my work I did not use the spaCy algorithm per se, but instead used a Python library called **ASent**[17], which was created using spaCy. Its principle is similar to the Vader library, especially in terms of the output structure, which includes positivity, negativity and neutrality scores along with the compound value of the text. ASent relies on pre-defined rules and lexicons as it actually uses words and their sentiment scores to determine the overall sentiment [26]. This algorithm works by breaking the text down into individual tokens

[17]https://spacy.io/universe/project/asent

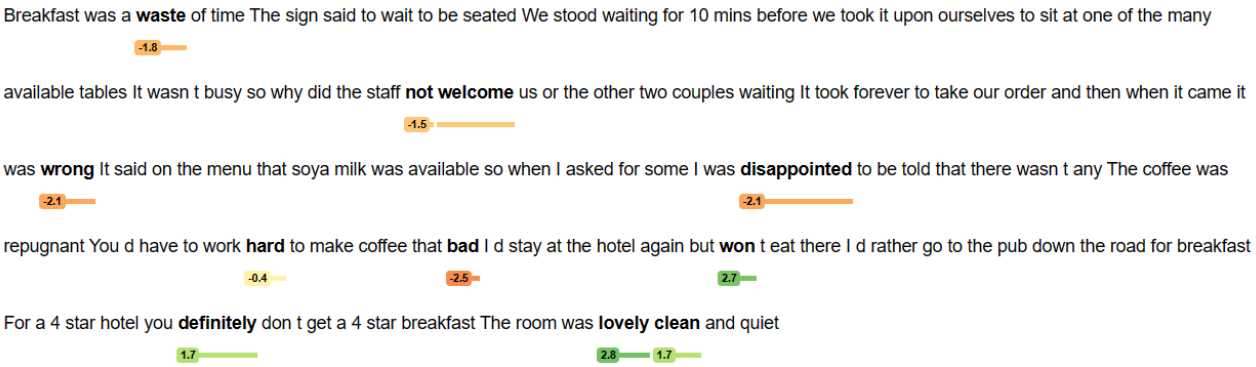


Figure 18: Sentiment analysis of ASent – 'prediction' style visualisation

(words) and looking up their corresponding sentiment scores in the lexicon. The overall score of the whole text is then calculated by averaging the sentiment scores of all the words in the text.

ASent also offers some additional useful features, such as the ability to handle negations, intensifiers or diminishers, which tend to be widely used in the tourism domain. My preference for this particular library over spaCy is dictated by the fact that ASent is better suited to specific domains where a pre-defined sentiment lexicon is available and relevant. In the case of the large dataset of reviews, it was also a question of computation time, which needed to be optimised. CNN algorithms would be much slower at this task. In addition, ASent includes visualisers for a clearer demonstration of the model predictions, making the model easy to interpret. The library offers a few options of visualisation styles such as **prediction** or **analysis**. The latter has the same visualisation as displacy in the 'dep' style in Figure 16, while the 'prediction' style, as seen in Figure 18, specifically highlights words with positive or negative sentiments along with their negations, intensifiers, etc (such as *not welcome* in Figure 18).

I applied the same sentiment detection algorithm to all reviews and calculated sentiment scores ranging from -1 to 1 for each of them. Negative compound scores are used to classify the negative text, while positive compound scores are used to classify the good comments.

4.3 Sentiment Prediction Models

4.3.1 Baseline Model

Before attempting to implement the prediction models, it is important to first build a baseline to have something to compare the results with. Usually this is a very simple model without any additional features, which will help to understand if newly created features, data cleaning, model selection or parameter adjustment, any fine-tuning, will improve accuracy and other prediction metrics compared to the model without any enhancements. I used **scikit-learn**[18] Python library, which is widely used for building machine learning techniques such as classification, regression, clustering, etc. All the other models I used in my research, except the neural network, are also part of the sklearn library.

A useful feature provided with this package is the **dummy classifier** function. It basically makes predictions without taking any input features into account. The only parameter that had to be specified was the strategy. I used **'stratified'** because the dataset is very imbalanced. This strategy makes sure that in a split for test and training data, the proportion of values in the sample is the same as the proportion of values given.

Nevertheless, apart from the dummy model, I also created some simple features and tried predictions with **Gaussian Naive Bayes Classifier**, **Decision Tree**, **Random Forest** and **Support Vector Machine** – all the models I used – and fed them with the original features: number of words from the positive and negative sides, and created two new ones: total number of words from the whole review and total number of characters. Unlike other machine learning techniques, for neural networks, namely **Bidirectional LSTM**, I used only the cleaned review and applied the embedding, the details of which are described Subsection 4.3.6 dedicated to this model.

Nevertheless, for all methods the labelling was based on the score of the reviewer. If the score was less than 5 on a 10-point scale, the comment was labelled 'negative', otherwise it was labelled 'positive'. The same proportion was used in the test and training split in the whole model to deal with the imbalance by using stratification.

Using the **train_test_split** function from the sklearn library, I split all the data into 80% for model training and 20% for testing the predictive ability of the models. I also set the random state to 42, which means that in each run the split is done randomly, but always with the same training and test set for each model. The way the sets are chosen affects the overall performance score, so to get a fair comparison of all the models I use the same splits all over again.

The prediction results of the baseline can be found in Table 1. As for the metrics used to evaluate the models, in addition to the classic accuracy score, I also chose precision, recall and f1-score. These parameters

[18] https://scikit-learn.org/stable/

are useful for measuring true positives, false positives and negatives. There may be cases where the accuracy is quite high, but if the dataset is unbalanced, it is only because the majority class is correctly predicted. But the real interest is in identifying the minority class. It is the same situation with reviews, negative comments are more interesting for the business owners as they can help to improve the service and potentially get more profit. To avoid the cases of majority class prediction, the metrics described above are used. They help to better evaluate the model and also look at the prediction of the minority class.

Model	Accuracy	Recall	Precision	F1-score
DummyClassifier	0.9176	0.0363	0.0371	0.0367
Gaussian NB	0.935	0.1604	0.1944	0.1758
Decision Tree	0.9452	0.1117	0.227	0.1497
Random Forest	0.9493	0.1057	0.2749	0.1526
Support Vector Machine	**0.9569**	0.0154	**0.5798**	0.0301
Bidirectional LSTM	**0.9624**	**0.2309**	**0.6981**	**0.347**

Table 1: Performance evaluation of the baseline models

So far, **Bidirectional LSTM** showed the best performance over all other baseline models, although it had the longest computation time. The SVM also demonstrated good results, with the second highest accuracy and precision scores. However, this model has the lowest recall and f1-scores, for which Gaussian NB, Random Forest and Decision Tree perform better.

What I have also done, and decided to consider as a more 'advanced' level of the baseline, are models with the reviewer's score label and only one sentiment analysis tool feature. For example, after processing the comment using Vader, I fed the model only the numbers it produced, namely positive, negative, neutral and compound scores. To further improve performance, I will also consider all possible features and combinations of sentiments generated by other tools.

The metrics can be found in Table 2. As you can see, I did not include the Bidirectional LSTM model there, but this technique is different from the others because it works with the embedding of the text and its label, while omitting other numerical and categorical features. Therefore, for this model, I will only experiment with different labelling methods, but an embedding base will always remain the same (i.e. a cleaned-up review text).

The conclusion that can be drawn from Table 2 is that **Random Forest** showed the best accuracy and precision results regardless of the sentiment tool used, while **Gaussian NB** had the highest recall and corresponding f1-score. As I am dealing with unbalanced data, the better f1-score is more important for performance evaluation than pure accuracy. However, it is still very low in this particular case, so further data, model parameters and labelling improvements will be made to get the best results with both high accuracy and other metrics.

Model	Accuracy	Recall	Precision	F1-score
Vader				
Gaussian NB	0.9012	**0.3965**	0.1906	**0.2575**
Decision Tree	0.957	0.0949	0.5152	0.1603
Random Forest	**0.9586**	0.0834	**0.6702**	0.1484
Support Vector Machine	0.9537	0.0709	0.3322	0.1168
TextBlob				
Gaussian NB	0.9081	**0.328**	0.1841	**0.2359**
Decision Tree	0.9293	0.2351	0.2125	0.2233
Random Forest	**0.9582**	0.0693	**0.6602**	0.1255
Support Vector Machine	0.9567	0.0024	0.6111	0.0049
AFINN				
Gaussian NB	0.9194	**0.3094**	0.2085	**0.2491**
Decision Tree	0.9572	0.0745	0.5346	0.1307
Random Forest	**0.9583**	0.0819	**0.6369**	0.1451
Support Vector Machine	0.9505	0.1548	0.3402	0.2128
SpaCy				
Gaussian NB	0.8798	**0.6518**	0.2079	**0.3152**
Decision Tree	0.9309	0.2603	0.2266	0.2266
Random Forest	**0.9593**	0.1102	**0.6153**	0.187
Support Vector Machine	0.9580	0.0025	0.5882	0.0051

Table 2: Comparison of the model performance depending on dataset labelling

4.3.2 Gaussian Naive Bayes Classifier

Gaussian Naive Bayes Classifier is a supervised learning method of the sklearn package that extends the probabilistic approach of Naïve Bayes and assumes that the features are likely to follow a Gaussian (or normal) distribution. Bayes' theorem assumes conditional independence between the features, taking into account the given class [59].

In my case, I used the same training and test sets of the data for all models that are used in the baseline, including the stratification strategy and the specification of the random parameter, but in each of the iterations I applied different metrics for labelling based on the sentiment analysis tools mentioned above. The performance results can be found in Table 3.

Labelling Method	Accuracy	Recall	Precision	F1-score
Reviewer's score	0.8919	0.6	0.2165	0.3183
Vader	0.7914	0.6341	0.5356	0.5807
TextBlob	0.853	0.7311	0.4704	0.5725
AFINN	0.9047	0.8245	0.574	0.6768
SpaCy	**0.9191**	**0.8845**	**0.6702**	**0.7626**

Table 3: Comparison of different labelling methods with Gaussian Naive Bayes Classifier Model

SpaCy showed the best result with the highest accuracy of 91% and f1-score of 76.2%. The least accurate

NB label was **Vader** with only 79% accuracy. Unsurprisingly, labelling based on the reviewer's score alone had the lowest recall and precision.

The confusion matrix of the reviewer score based labelling can be seen in Figure 19. Although the classifier was able to correctly identify more than 90,000 positive reviews, about 6,000 negative comments were misclassified as positive.

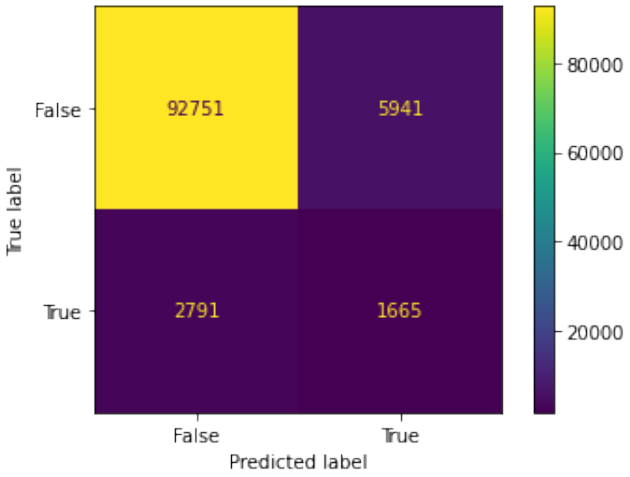


Figure 19: Visualisation of confusion matrix based on reviewer's score labelling

It also confirms that the user's numerical estimation (rating) tends to be different from the descriptive experience (review) and that they cannot be used interchangeably (even taking into account the variation in results and errors that may be made by sentiment analysis tools). Overall, Gaussian Naive Bayes was one of the fastest and easiest models to use, which also produced an acceptable result.

4.3.3 Decision Tree

Decision Tree is a supervised learning algorithm widely used for classification or regression. It consists of three main components: leaves, edges and nodes [7]. This method provides a good visual representation of how the model works, which is why it is considered a white box technique. The algorithm starts with the main or root node (certain value of the attribute), which asks a question like yes/no, whether the value is higher or lower, etc. The answer to this is in the edge, which eventually leads top-down to the exit of the tree – leaf (prediction result).

I tried a few options for Decision Trees. Without specifying the depth, the model can go down almost infinitely with extremely nested nodes and a chaotic structure. This is basically what I got when I did not specify any parameters. Furthermore, after restricting the model, namely reducing the maximum depth of the tree to 3 and specifying the criterion of entropy (measure of disorder), the performance of the model improved

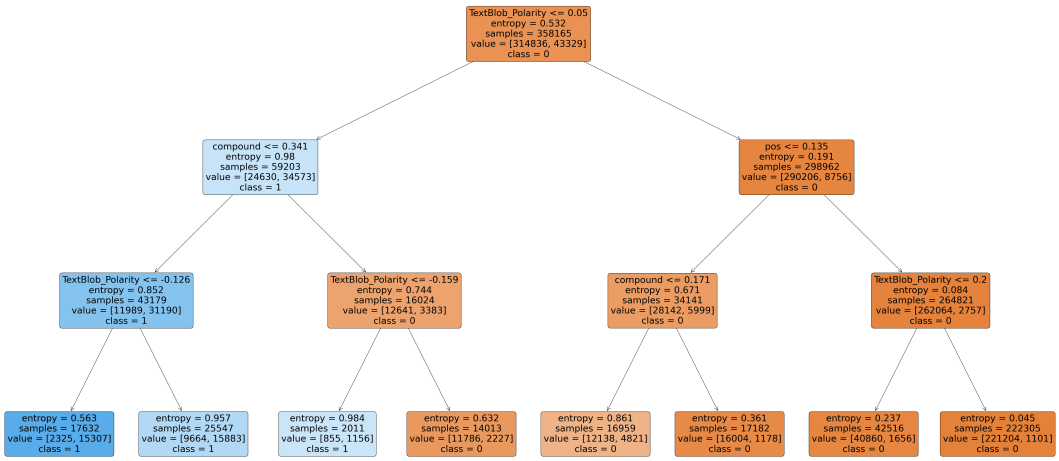


Figure 20: General overview of Decision Tree based on AFINN labelling

on average by 1-2%. The result of the AFINN decision tree can be seen in Figure 20, where class 1 means that the review is negative and 0 – positive (closer view in Figure 21).

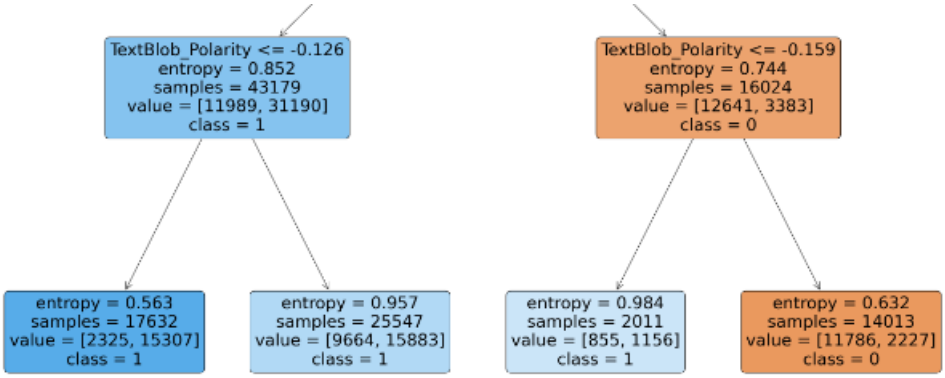


Figure 21: Detailed overview of Decision Tree nodes based on AFINN labelling

The only case where the pruned Decision Tree performed worse than the original one was with spaCy. Moreover, this type of labelling had one of the best performances compared to other sentiment analysis tools with a high f1-score. Surprisingly, the reviewer score classification showed a similarly high accuracy, but performed poorly on all other metrics, suggesting that it was a coincidence rather than a strong execution. Other models showed average results with f1-scores around 70%. Decision Tree was one of the algorithms that was easy to interpret and had an efficient running time.

4.3.4 Random Forest

Random Forest is an extension of the Decision Tree. In other words, it is a classification algorithm that combines the output of several Decision Trees and averages them to achieve higher prediction accuracy and

Labelling Method	Accuracy	Recall	Precision	F1-score
Reviewer's score	0.9585	0.1323	0.5272	0.2115
Vader	0.8389	0.681	0.6369	0.6582
TextBlob	0.8993	0.3931	0.7353	0.5123
AFINN	0.933	0.7442	0.714	0.7288
SpaCy	**0.9598**	**0.8654**	**0.8614**	**0.8634**

Table 4: Comparison of different labelling methods with Decision Tree

avoid overfitting [59].

For Random Forest in my dataset, I set the number of estimators (trees) to 100 and the random state responsible for bootstrapping (inferring the results of the small group to the whole population), randomness and sampling of features for splitting. The results of the SpaCy tool labelling were the best (Table 5), while the reviewer's score accuracy was only 1% lower. In this model, I achieved the highest f1-score I have seen so far of 91%, along with other metrics showing satisfactory results of over 90%.

Labelling Method	Accuracy	Recall	Precision	F1-score
Reviewer's score	0.9606	0.1282	0.661	0.2148
Vader	0.8731	0.6279	0.7724	0.6927
TextBlob	0.9151	0.5955	0.725	0.6539
AFINN	0.9542	0.7728	0.8367	0.8035
SpaCy	**0.9743**	**0.9082**	**0.9161**	**0.9122**

Table 5: Comparison of different labelling methods with Random Forest

In addition to the standard metrics, I also looked at the importance of the features in each of the labelling cases. For example, for the TextBlob and Vader labels, afinn_score was the most significant feature with about 20% importance. For AFINN itself and the reviewer's score, textBlob_polarity was dominant with 20% and 10% importance respectively. Other important features that always appeared in the top-20 were all doc2vec values, number of words, compound score, all words derived from TfidfVectorizer and days elapsed since the review was left.

I also plotted a **ROC curve**, which stands for 'receiver operating characteristic curve' [75], to evaluate the performance of the classification model. Similar to the confusion matrix, which looks at the number of correctly and incorrectly predicted samples of both classes, this graph is based on true positive and false positive rates and visualises the decision threshold. As can be seen from the ROC curve for random forest with Vader labelling in Figure 22, the higher the curve is above the diagonal baseline, the better the prediction.

Although the ROC curve is usually a fairly good and clear summary of the quality of the model, in the case of a highly imbalanced dataset, where the number of positive comments strongly outweighs the number

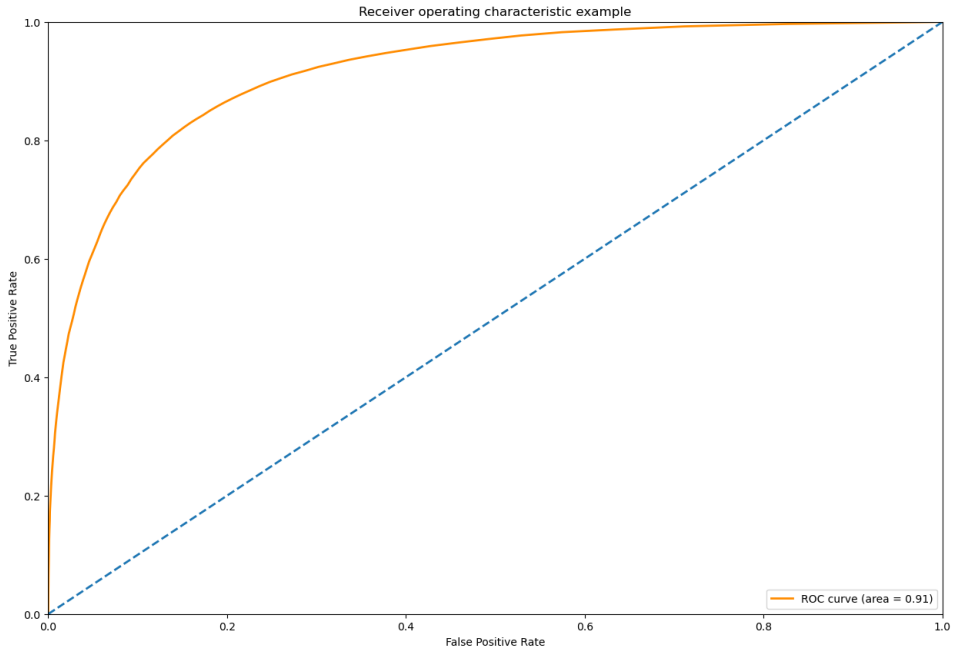


Figure 22: ROC curve of Random Forest (Vader)

of negative ones, the results can be distorted. The prediction class of good reviews is 0 because of the label 'Negative', so only reviews belonging to this class were marked as 1. Machine learning techniques, on the other hand, identify class 0 as negative, so I have to keep in mind that I have to treat correctly identified bad reviews as true positives and incorrect ones as false positives. The false positive rate is calculated as the ratio of false positives to negatives. Considering that all the negatives account for more than three quarters of the dataset, which is about 350,000 reviews, the ratio will be extremely low, even if the number of mislabelled samples is still high. This would also artificially increase the AUC – 'area under the ROC curve' [75].

In general, Random Forest was one of the most accurate, easy to use and understand techniques, with a relatively efficient computation time given the large dataset I was dealing with.

4.3.5 Support Vector Machine

Support Vector Machine (SVM) is a machine learning algorithm often used for classification purposes. Its goal is to find a hyperplane that maximises the margin – the distance between points in different classes. This dividing line is a decision boundary, since data points on opposite sides of the hyperplane belong to opposite classes [59]. The dimensionality of the model depends on the number of input features. I applied **Linear Support Vector Classification** from the sklearn library to my data. The SVM with linear kernel is more flexible in the choice of penalties and loss functions and scales better to large numbers of samples.

I also increased the maximum number of iterations to 2,000 from the default of 1,000. As can be seen in Table 6, the model showed the best accuracy and recall in conjunction with the reviewer's score labelling

method. This is an understandable result, as the SVM works best with a clear margin of separation, as in this case. It still has a very low precision score and therefore, despite the high recall, the f1-score only reached around 8%. The model showed good precision in the interaction with spaCy with around 93% and the highest f1-score of 69%.

Labelling Method	Accuracy	Recall	Precision	F1-score
Reviewer's score	**0.959**	**0.7187**	0.0407	0.0771
Vader	0.6582	0.5458	0.6579	0.5966
TextBlob	0.8825	0.1461	0.886	0.2508
AFINN	0.8052	0.4424	0.2959	0.3547
SpaCy	0.9275	0.5482	**0.929**	**0.6896**

Table 6: Comparison of different labelling methods with Support Vector Machine

Although the overall results were good, there was still no single labelling technique that performed adequately on all metrics. In all cases, either recall or precision was extremely low (less than 50%), which cannot be considered a good option.

Another way to check if the performance of the SVM is indeed low is **precision-recall curve** [11]. It measures the prediction success when the classes are very imbalanced and is more suitable for my dataset than a ROC curve. An area under the curve shows both high recall and high precision. Precision is associated with a false positive rate, while recall is associated with a false negative rate. High values for both precision and recall indicate that the classifier is providing accurate results as well as a majority of all positive results. The better the model, the higher the metrics.

In Figure 23 SVM performance is summarised with AP (average precision) by taking the area under the curve. The higher this aggregated parameter, the better the performance of the model. In my case, the SVM combined with TextBlob has an AP of 0.63, which is still not a satisfactory result. Overall, the model performed poorly compared to others and could not improve much despite promising baseline results.

4.3.6 Bidirectional LSTM

In general, there are three types of NN: feedforward neural networks, convolutional neural networks, and recurrent neural networks (which is what I am building for my data) [97]. Ordinary feedforward NNs are mainly used with independent data points that have their own weights, while convolutional NNs use a mathematical operation of the same name, which is suitable for processing pixel data in visual image recognition.

Bidirectional LSTM (long-short term memory) is a type of recurrent neural network (NN) specifically designed to work with time series or sequential data [97]. In recurrent NN, input vector components have the same weights. The LSTM architecture is introduced to solve the vanishing and exploding gradient problems.

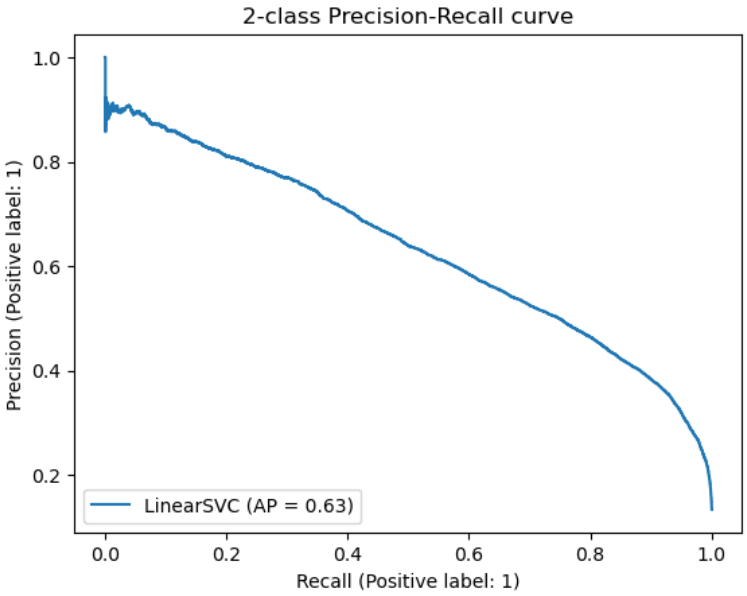


Figure 23: Precision-Recall curve of SVM (TextBlob)

It is also more suitable for maintaining long-range connections, as it recognises the relationship between the values at both the beginning and the end of the sequence. The bidirectional type of LSTM has a unique structure and is widely used for NLP-related purposes.

Unlike regular LSTM, the input flows in both directions and can use information from both sides. It is also an effective tool for modelling sequential dependencies between words and phrases in both directions of a sequence. In addition, BiLSTM adds another LSTM layer that reverses the direction of information flow. In short, the input sequence flows backwards through additional LSTM layers. The outputs of the two LSTM layers are then combined in various ways, such as averaging, summation, multiplication or concatenation [97].

I applied BiLSTM to my dataset and, as with the other models, tried four different labelling methods. The only difference in this approach was that I only used the text of the cleaned review and its class to feed the model. All other additional features such as Doc2Vec or TfidfVectorizer, etc were not included in the prediction.

First, I tokenised the reviews again, then converted them to sequences and padded (normalised) them to the same length, which in my case was the maximum number of words in the longest comment. I also built a neural network model, the parameters of which can be found in Figure 24.

The embedded sequences were further transformed into **numpy arrays**, as were the labels, which were additionally transformed into a categorical feature, with two classes '0' ('Positive') and '1' ('Negative'). Other parameters I defined in the model were the number of epochs (only 1 epoch in my case, as the computation time for such a large dataset with a multi-layer model is already long), the batch size of 64, and callbacks to ModelCheckpoint, which would save only the best results. Accuracy prediction and other model evaluation

```
embedding_vector_features=20
model1=Sequential()
model1.add(Embedding(num_words,embedding_vector_features,input_length=sent_length))
model1.add(Bidirectional(LSTM(50, return_sequences=True)))
model1.add(Dropout(0.3))
model1.add(Bidirectional(LSTM(50)))
model1.add(Dense(2,activation='softmax'))
model1.compile(loss='CategoricalCrossentropy',optimizer='RMSprop',metrics=['accuracy'])
```

Figure 24: Piece of code for building BiLSTM model

metrics were computed in the same way as for other models, and the results can be observed in Table 7.

Labelling Method	Accuracy	Recall	Precision	F1-score
Reviewer's score	0.9624	0.2309	0.6981	0.347
Vader	0.7633	0.2299	0.7633	0.3068
TextBlob	0.9457	0.8158	0.788	0.8017
AFINN	**0.9674**	**0.9405**	0.8175	**0.8747**
SpaCy	0.929	0.6519	**0.8556**	0.8556

Table 7: Comparison of different labelling methods with Bidirectional LSTM

Surprisingly, in contrast to the algorithms previously described (apart from SVM with the reviewer's score as the leader in accuracy and recall), the best score was achieved in combination with AFINN labels, which is the second highest overall result so far. SpaCy labelling achieved high results, almost reaching the same accuracy and f1-score, and even surpassing AFINN's precision score. Together with Vader, the reviewer scores, despite high accuracy (76% and 96% respectively), showed relatively low recall and consequently low f1-score.

In general, BiLSTM, as expected due to its more advanced architecture and NLP orientation, is the most accurate model with the highest average performance parameters so far (despite the fact that one of the Random Forest results outperformed this algorithm), but as one of the main drawbacks, it is important to note that it is also the slowest algorithm, taking up to 20 hours to process the results on my hardware.

4.3.7 GridSearchCV & RandomisedSearchCV

Although I have already achieved quite high accuracy with bidirectional LSTM, I decided to take the second best technique, Random Forest, from the sklearn library to try to improve its predictive ability using hyperparametric optimisation methods. Finding the settings for the model is a challenging task, and of course it could be done by rule of thumb, but this solution is too ineffective and very time consuming.

Actually, there are a couple of useful tools in the same sklearn package that can do hyperparameter tuning automatically, namely **grid** and **randomised searches** [10]. The former defines a search space as a grid of hyperparameter values and evaluates each grid position. It is preferably used for spot-checking

combinations that generally give good results. The second defines a search space as a bounded domain of hyperparameters and randomly selects points from it. Randomised search is therefore useful for discovering and obtaining hyperparameter combinations that may not be intuitively obvious, but the algorithm often requires more computation time.

The model I am trying to improve is Random Forest with AFINN-based classification, which has outperformed other machine learning techniques in the sklearn library except spaCy, so I want to achieve similarly high results with AFINN as well. The original results are shown in Table 8 for easier visual comparison of the results.

	Accuracy	Recall	Precision	F1-score
AFINN	0.9542	0.7728	0.8367	0.8035
Grid Search	0.9515	0.7511	0.8318	0.7894
Randomized Search Search	**0.9551**	**0.778**	**0.8393**	**0.8075**

Table 8: Comparison of grid and randomised searches on Random Forest

I ran this experiment on the reduced dataset of 10,000 samples. In the case of grid search, I created the dictionary with all possible model parameters listed (bootstrap, max_depth, max_features, n_estimators, etc) and also set the cross-validation to 5. I ran the algorithm a few times before I could see that I was getting similar results regardless of the random input data. The output hyperparameters that worked best were: n_estimators: 200, min_samples_split: 5, min_samples_leaf: 2, max_features: 'auto', max_depth: 10, bootstrap: False. However, despite the search provided, my original model still outperforms the advanced one. I did not specify any particular parameters in the original Random Forest, so surprisingly, defining the model more precisely did not improve the results, and in fact worsened them by about 3% on average.

In the randomised search case, I did the same pre-processing and set up a parameter dictionary. The algorithm took much longer to produce an output and got completely different hyperparameters compared to the grid search: n_estimators: 400, min_samples_split: 2, min_samples_leaf: 1, max_features: 'sqrt', max_depth: None, bootstrap: False. As can be seen in Table 8, this experiment was more successful and helped to improve all metric results by an average of 0.3%, which is still a poor result. In general, the results of both grid and randomised search were very close, with a 1-5% difference in favour of the latter.

4.3.8 Unsupervised Learning Methods: K-means

Unsupervised learning is a type of machine learning that involves training a model on a dataset that has no pre-defined labels or categories [39]. Instead, the main idea is to discover some patterns and structure within the data on its own, without any external support or human intervention, in contrast to supervised machine learning,

which involves training a model on labelled data with the aim of predicting labels for new and previously unseen data.

Although my dataset contains reviewer scores, which can be used as a basis for labelling, it does not initially contain any categories and is a good environment for testing patterns. One of the most common unsupervised learning techniques is **clustering**, where data points are grouped based on their similarity. There are many different types of clustering, such as **hierarchical**, **density-based** or **K-means** clustering, etc. The latter is one of the most popular algorithms, which divides the data into k clusters based on the mean distance between the data points.

K-means works by first randomly selecting k points from the dataset as the initial centroids for each cluster. Then, each data point is assigned to the cluster whose centroid is closest to it (with the shortest distance, which can be calculated in different ways: Euclidean, Hamming, Manhattan, etc) [71]. Once all the data points have been assigned to clusters, the centroids are recalculated as the mean of all the data points in each cluster. This iteration is repeated until the centroids converge and the clusters no longer change. The resulting clusters can be further analysed to uncover patterns or insights in the data. It is important to note that the effectiveness of K-means clustering depends on the quality of the initial centroids and the choice of the number of clusters, which is usually subjective and requires some additional experimentation.

Therefore, my primary task before starting any clustering was to identify the optimised number of clusters, which I did using the **Elbow** and **Silhouette** methods. The first technique involves calculating the **Within-Cluster-Sum of Squared Errors (WSS)** [46] for different values of k and choosing the number of clusters where the WSS starts to decrease dramatically. The visualisation (left graph in Figure 25) often resembles an elbow, hence the name. For this purpose, I have tried to work with two data formats: exclusively review texts (their embeddings in Word2Vec) or a set of features characterising the comment (including scores of sentiment detection techniques, vectorisers, text length, etc). Surprisingly, the results of the two experiments are different. In the case of pure text, the optimal number of k is 2, while the combination of features resulted in 3 clusters as the most efficient (left graph in Figure 25).

Silhouette score [46] is another metric used either to evaluate the performance of the clustering procedure or to define the initial optimal number of clusters. This parameter measures the relationship between cohesion (intra-cluster distance) and separation (inter-cluster distance) and ranges from -1 to 1, with positive values indicating better 'density' of data points within the cluster and 'greater' distance between clusters.

I got the same output for both data types as in the previous example, confirming the correctness of the results. The highest silhouette score (0.44 right graph in Figure 25) is achieved for 3 clusters with the specific features and 2 clusters in case of simple text embedding. A possible explanation for this difference, and in general for

this exact optimal number of k, could be the positive and negative (and sometimes neutral) categories of the reviews. When working with the text of the comment, the algorithm mostly recognises only sentences with a good or bad connotation, in the case of feature aggregation it is more common to get more neutral scores, which are usually obtained when the reviewer mentions both positive and negative aspects of the stay, so that for most sentiment detection techniques they cancel each other out and the final value is closer to 0.

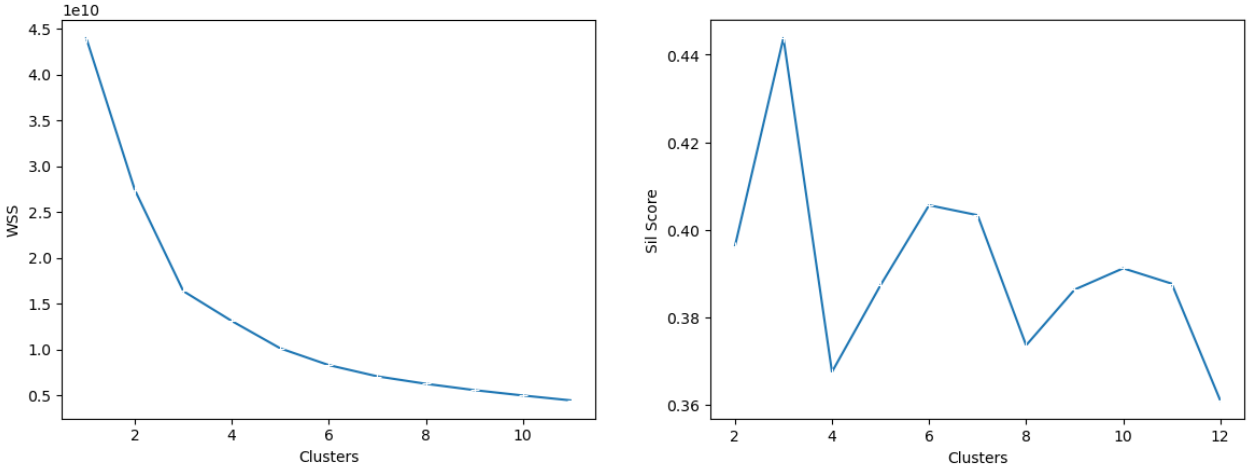


Figure 25: Elbow and Silhouette methods results for identifying an optimal number of k

In Figure 26 you can see the K-means clustering results for 2 and 3 clusters based on the word vectors. I could manually change the number of k, so I chose the results described above. On the left, blue points refer to negative reviews and red to positive ones. Reviews with similar content were grouped according to the same principle as the Word2Vec embedding of sentences in Figure 15. On the right, the green colour represents good reviews, the blue colour neutral reviews and the red colour bad reviews. The reviews that end up in the blue (neutral) area usually contain positive and negative aspects together, such as '*but*' constructions – *the room was amazing but I didn t like breakfast*.

The performance of K-means clustering can be considered satisfactory overall, since a silhouette score of more than 0.5 usually indicates a high quality cluster. I got 0.44, which is very close, but still indicates some overlapping of clusters, as can be clearly seen on both graphs in Figure 26.

4.3.9 BERTopic

BERTopic is an unsupervised modelling algorithm based on the BERT language model (Bidirectional Encoder Representations from Transformers), which is a pre-trained deep learning technique developed by Google and designed with the purpose of generating high-quality language representations [23]. The model works by first encoding each document (in my case, review) in a corpus using BERT, which provides its high-dimensional vector representation. BERTopic then performs a dimensionality reduction, such as UMAP (described in more

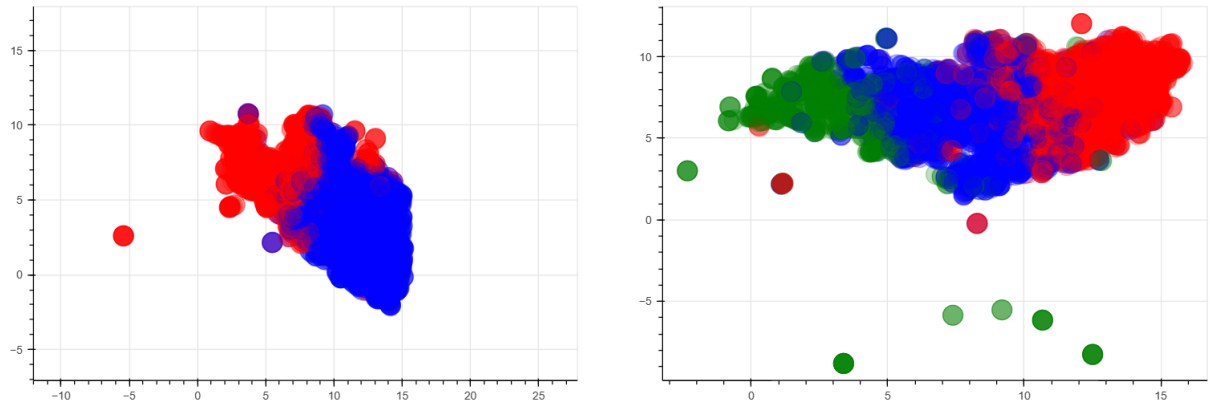


Figure 26: K-means clustering performed with 2 (left) and 3 (right) clusters

detail in Subsection 4.2.3), on these vectors to produce a lower dimensional representation that would preserve the original similarity relationships between the documents. Once the encoding is done and the dimensions are reduced, BERTopic applies clustering to group similar documents into topics. The clustering technique can be any of the standard clustering algorithms, but in particular BERTopic uses **HDBSCAN** – a density-based clustering with automatic determination of the number of clusters [33].

One of the specific features of BERTopic is the **Intertopic Distance Map** [27], which visualises the distance between topics. This map can be used to identify topics that are closely related or to detect patterns in the overall structure of the corpus. The size of topics is always defined by the number of documents that share the same topic, with larger topics consisting of more documents. The topics themselves are determined by the most representative documents within the cluster, calculated by a combination of word frequency and their coherence within the topic. This ensures that the topics are meaningful and accurately capture the topics of the corpus.

An example of the inter-topic distance map can be seen in Figure 27. This is the result I got after applying the BERTopic algorithm in combination with the spaCy embedding model. It is also important to note that this is not a visualisation of the whole dataset, but only of 10,000 randomly selected samples of original reviews to save some computation time. However, this is still enough to give a reasonable representation of the themes, apart from possibly some outliers that do not appear very often in the comments. As a result, I got 8 topics (0-7 in Figure 27):

1. Topic 0 – the largest topic with size 9, 825 – topic with most popular auxiliary words like *a*, *the*, *were*, *to*, etc
2. Topic 1 – size 41 – generic comments (*nothing*, *everything*, *all*, etc)
3. Topic 2 (shown on Figure 27) – size 34 – location-related topic (*location*, *central*, *architecture*, etc)
4. Topic 3 – size 34 – theme about food (*breakfast*, *delicious*, *excellent*, etc)

5. Topic 4 – size 28 – again location (*location*, *neighborhood*, *great*)

6. Topic 5 – size 14 – grammatically or orthographically incorrect reviews (*pergect6*, *noybing*)

7. Topic 6 – size 13 – difficult to classify, just random word combinations (*och*, *att*, *ra*, etc)

8. Topic 7 – size 11 – personnel-related topic (*staff*, *reception*, etc)

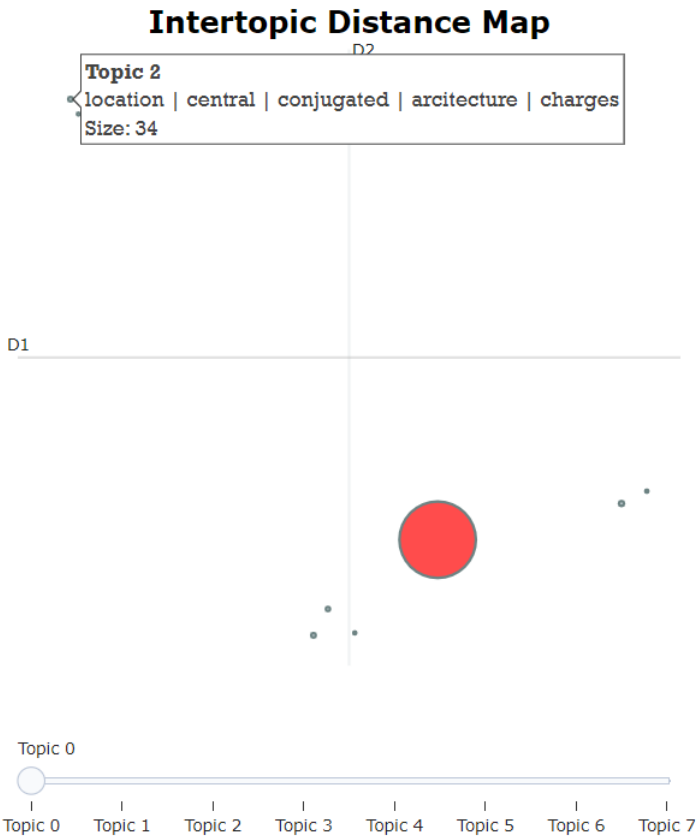


Figure 27: Intertopic Distance Map – BERTopic

Obviously, these identified topics are biased by commonly used articles, negations and auxiliary words, which is why the largest topic is devoted to reviews that contain many of them. This means that the real topics or aspects of the comments were not recognised by the algorithm. Another unexpected problem was the identification of two location-related topics with no apparent difference between them. One of the main problems was also that the technique is not able to recognise that a review may contain a few topics, it assigns exactly one topic to a document, which unfortunately is not suitable for the tourism domain. This explains why the size of the clusters related to location and staff is so small – these were only the reviews written exclusively about these aspects.

Overall, for my particular case, BERTopic gave rather poor, not really meaningful results and did not contribute much to the work, mainly due to the peculiarities of the domain mentioned above.

4.3.10 BERT Model

BERT (Bidirectional Encoder Representations from Transformers) has already been briefly described above in BERTopic, which is based on this model, but I would like to apply the pure algorithm to my data as well, since it is known to be effective in detecting the aspect and its associated sentiment in various domains. Other reasons why I chose the BERT model include its ability to understand the context and meaning of words in a sentence rather than just their dictionary definition, which is important for comments where reviewers may use tricky linguistic constructions or not explicitly describe the aspect, etc.

In general, BERT is a deep neural network that uses Transformer-based language models (a type of architecture that solves sequential tasks together with long-range dependency handling) to detect the contextual relationships between words in a text [43]. BERT is usually trained on a large corpus of text data to get a general understanding of the language. It is then fine-tuned for specific tasks such as question answering, sentiment analysis (aspect-based in my case), etc. As the name suggests, the model is bidirectional, so it considers the whole context of a sentence when making predictions. The processing steps include tokenisation, input embedding, self-attention and output generation. In the first step, the input text is broken down into words (or basically smaller units called tokens), which are further converted into and represented by numerical vectors using word embeddings. A self-attention mechanism is then used to weigh the importance of the words, taking into account the context of the sentence as a whole. Finally, the last part of the output generation itself makes a prediction based on the learned contextual relationships between words [23, 43].

The code I used for my data was found on Kaggle and is basically already a pre-trained BERT model[19] fine-tuned to a specific natural language processing task (tokenisation, word/sentence vectorisation, etc). It is worth noting that I used CPU (Central Processing Unit) to compute the model, so my capacity was limited to the capabilities of my laptop. Unfortunately, due to lack of local memory, I could not apply BERT to the entire dataset and had to randomly select 10,000 samples, taking care to preserve the original proportion of positively and negatively labelled reviews.

The first step was to load the pre-trained BERT model with a specific task of sentiment analysis on the IMDB dataset using the **Hugging Face** library[20]. This algorithm has 12 layers and 110 million parameters, as well as 2 epochs with a batch size of 32, meaning that the entire dataset was used twice for training.

I also pre-processed my comments by tokenising the sentences and converting them into input features that would be suitable for input to the BERT model. The data was then split into training and validation sets. The model architecture was defined by a classification layer added on top of the pre-loaded algorithm, while the

[19] https://www.kaggle.com/code/harshjain123/bert-for-everyone-tutorial-implementation
[20] https://huggingface.co/

pre-trained layers had to be frozen to prevent the model weights from being updated during training.

For the classification layer, I needed to specify the class weights to balance the effect of imbalanced classes during training. In many classification tasks, the number of instances of one class can significantly outnumber those of another, which can lead to a bias towards the majority class during training (i.e. the model could be overfitted and just automatically assign the most popular class label). Class weights are assigned to each class based on the inverse of its frequency in the training set, so that less frequent classes are assigned higher weights. This allows the minority classes to be given more weight and helps to improve the overall accuracy of the model. I calculated the weights using the **compute_class_weight** function from the sklearn library. As a result, negative reviews received 8.5574 as a minority label, while positive reviews received only 0.531.

The training step also involves masking the random input tokens, i.e. replacing them with a special token called [MASK], and training the model to correctly predict the masked tokens based on the surrounding context (other non-masked tokens in the sentence) [38]. This process results in the model learning rich contextual representations for the input tokens, as well as learning how to deal with missing or incomplete information.

As I am dealing with a different domain and data from the pre-trained model, I made my own model settings, namely increasing the number of epochs to 10 for training. I trained the algorithm on the training dataset, including the loss and gradient calculation, and then, based on the results (unlike the pre-trained model weights, which are usually frozen), updated the weights of the classification layer using the back-propagation technique and an optimisation algorithm.

In this case I used **Adam** (Adaptive Moment Estimation) – an optimisation algorithm widely used for training neural networks [12]. Being a stochastic gradient descent algorithm, Adam adapts the learning rate of a parameter by taking into account the history of previous gradients for that parameter. In my code this optimiser is applied with a learning rate of 2e-5 and an epsilon value of 1e-8. After performing all the necessary training and optimisations, I saved the model for future use to avoid having to run the model from scratch again.

The results of the model described above can be found in Table 9. BERT was able to achieve quite high accuracy, but other metrics have even lower than average scores compared to more simplified deep learning techniques such as Random Forest or Decision Tree. In addition, to improve the performance, BERT can be fine-tuned to specific aspects that often appear in each review (such as *room*, *bathroom*, *breakfast*, *service*, *location*, etc), which could potentially improve the model performance and its computation time.

Accuracy	Recall	Precision	F1-score
0.897	0.599	0.579	0.588

Table 9: Evaluation of the BERT model performance

To demonstrate how the BERT model works from the inside, I decided to visualise one sentence from the randomly selected review: '*the hotel was in a bad condition not really clean and everything was worn out*'. According to the classification of attention visualisations in the transformer models and the code provided to build them[21], there are three different model views: an attention head, a model and a neuron view. The first visualises the attention patterns generated by one or more attention heads in a given layer [84]. This perspective (any subgraph in Figure 28) represents self-attention as connections between 'attending' and 'attended' tokens, with different colours representing the corresponding attention heads and line weight with the attention score. This type of visualisation is also quite interactive: users can choose the layer and the attention heads (simply by clicking on certain words).

Figure 28 demonstrates attentional heads in my model that capture both within-sentence and between-sentence patterns for sentence pairs. Each head learns a unique attention mechanism. In my case, in the upper left subgraph of Figure 28, attention is distributed approximately equally across the previous words in the sentence.

In addition, a sentence-level attention filter can be specified for BERT's sentence-pair model. For example, in Figure 29 there are 4 filters to select the 'direction' of attention. In the top right-hand corner, 'Sentence A $\rightarrow$ Sentence B' only shows attention from tokens in Sentence A to tokens in Sentence B.

I also created a model view for my model, which is a bird's eye view of attention across the twelve model layers (from 0 to 11), represented in the form of attentional subgraphs. This allowed me to see all the unique and general patterns of the model in one visualisation and to track the evolution of the algorithm changes.

In order to tailor the model to my data and goals, I defined the list of most commonly used aspects in the reviews. These are ['room', 'bathroom', 'bedroom', 'bed', 'tv', 'balcony', 'ac', 'air conditioning', 'coffee', 'service', 'staff', 'reception', 'receptionist', 'food', 'restaurant', 'breakfast', 'location'] together with their plural equivalents. The maximum length of the input sequence was set to 512. As before, I loaded a pre-trained BERT model along with the AdamW optimiser, but tuned it with my pre-defined aspects. All other parameters such as learning rate, batch size, number of epochs, etc were not changed. In addition, I added binary columns to my dataframe for each of the predefined aspects, indicating whether the review contains that aspect (1) or not (0).

Then I needed to create data loaders for the training and test subsets, which I did using a helper function called **create_data_loader**. It creates a PyTorch DataLoader from a Pandas DataFrame. The review and aspects columns are concatenated into a string with a '[SEP]'separator, which is a special token used in BERT and some other transformer models to separate different sequences that are common to one. The sequences were further tokenised and the resulting tokens were converted into PyTorch tensors, from which I could create a

[21] Vig et al., 2019, https://github.com/jessevig/bertviz

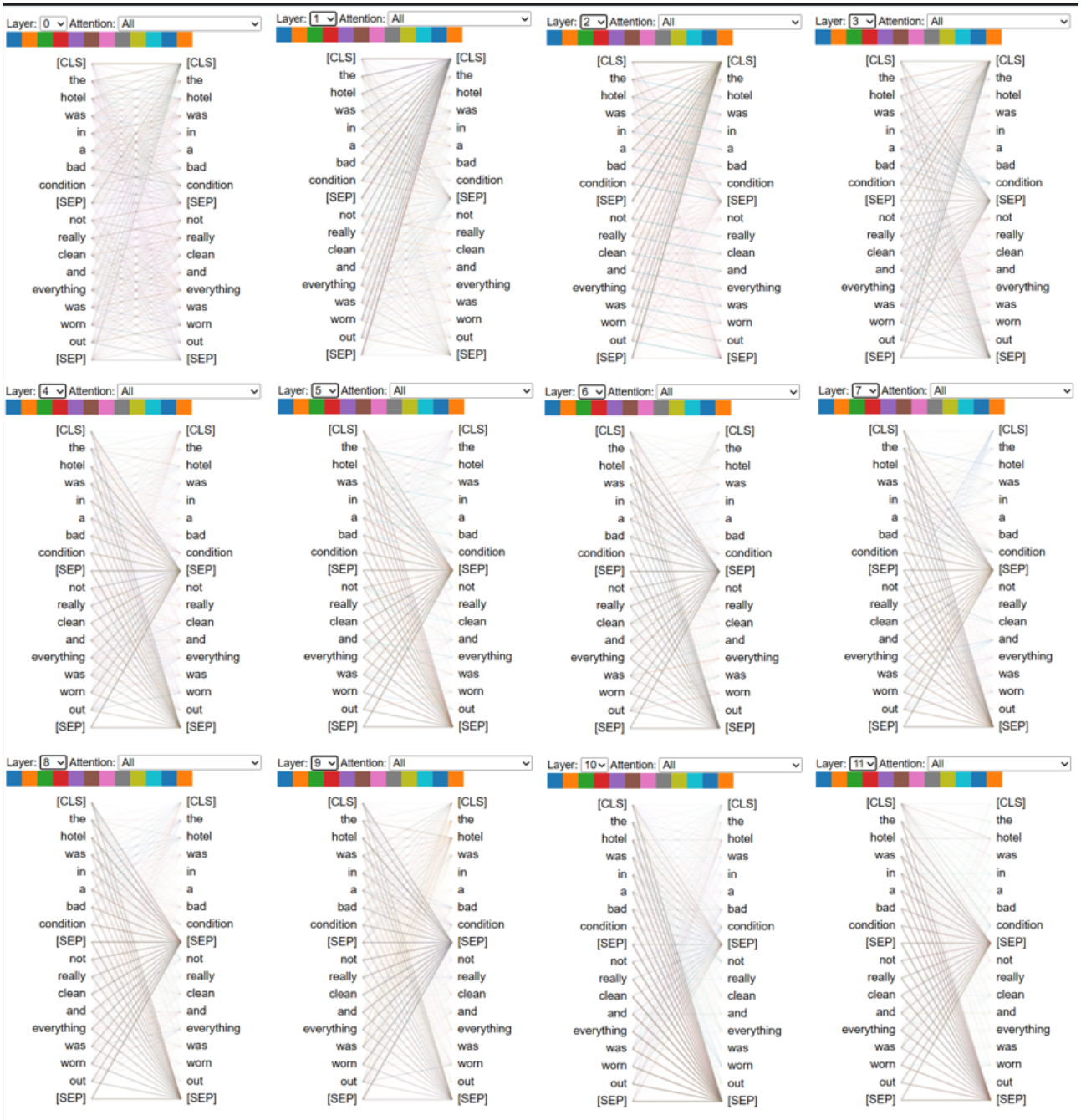


Figure 28: Model view for BERT – different layers overview

TensorDataset and as output a DataLoader that could be used for training.

The results of the performance of the updated BERT models are shown in Table 10. As can be seen, the second model outperformed the original by almost 40% in terms of metrics and 4.5% in terms of accuracy, while also showing highly satisfactory overall results (the highest recall and f1-score overall). This fine-tuned algorithm produced no false negatives (no positive reviews were classified as negative) and was able to achieve a perfect recall score of 1.

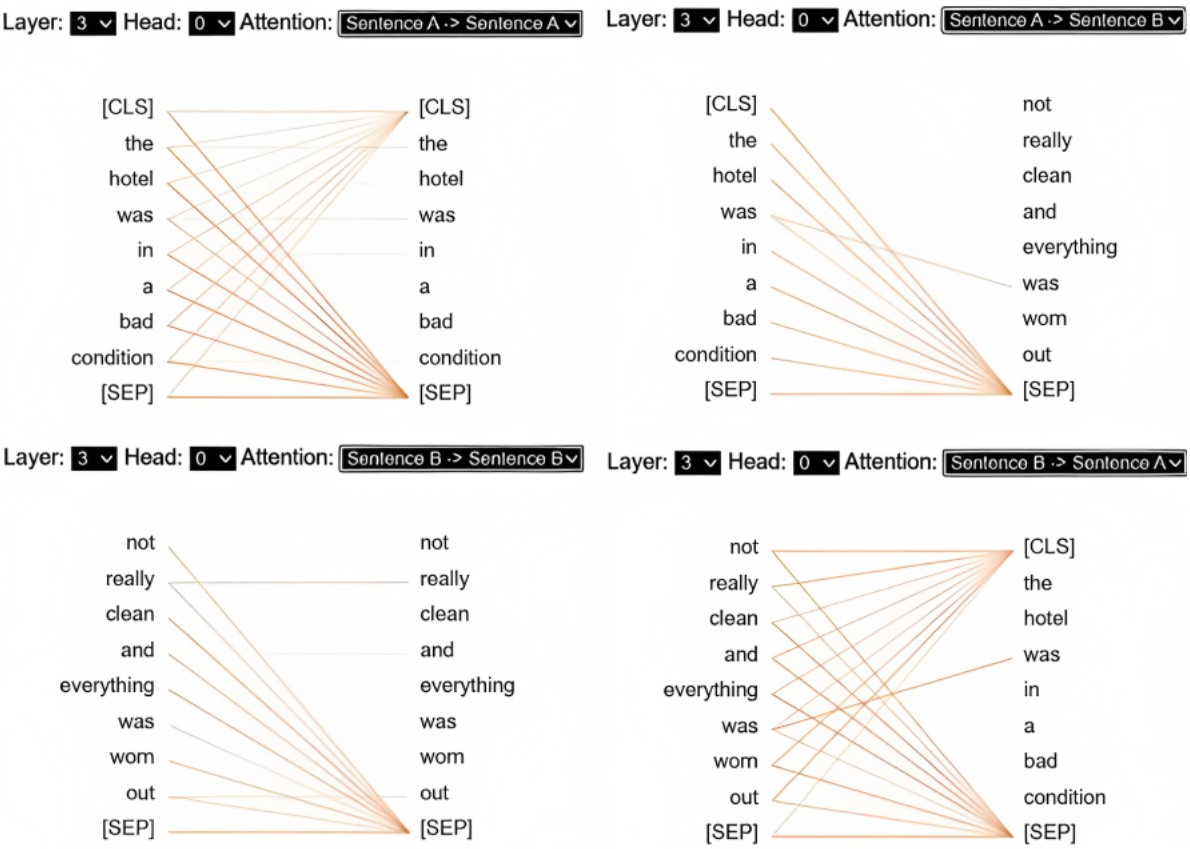


Figure 29: Attention-head view for BERT – different attentions overview

	Accuracy	Recall	Precision	F1-score
Original Model	0.897	0.599	0.579	0.588
Fine-tuned Model	0.943	1	0.943	0.97

Table 10: Comparison of the original and fine-tuned BERT model performances

The working principle of the new fine-tuned model is shown in Figure 30. It takes a review text and a list of aspects as input, runs them through the updated BERT model and generates aspect-based sentiments together with an overall sentiment of the review as output. As you can see in Figure 30, the sentiment scale ranges from 1 (1-2 being slightly and strongly negative) to 4 (3-4 being slightly and strongly positive). The statement '*Beds were very comfy bathroom was fantastic and location was excellent*', despite the aspects *bathroom* and *beds* being defined as only slightly positive, received an overall rating of 4 (very positive).

```
text = "Beds were very comfy bathroom was fantastic and location was excellent"
aspects =  ['room', 'bathroom', 'bedroom', 'bed', 'tv', 'balcony', 'ac', 'air conditioning', 'tee coffee', 'service', 'staff', 'r
result = analyze_sentiment(text, aspects)
print(result)
```

```
{'aspects': [{'aspect': 'bathroom', 'sentiment': 3}, {'aspect': 'location', 'sentiment': 4}, {'aspect': 'beds', 'sentiment':
3}], 'overall_sentiment': 4}
```

Figure 30: Example of fine-tuned BERT Model defining the aspects, their sentiments and overall sentiment of the text

Overall, the BERT model is not only a technologically advanced algorithm, but can also be useful from a business perspective. It can provide valuable insights to hotel management on how and in what aspects to improve customer satisfaction. However, the large size and processing requirements of BERT can make it difficult to use in certain applications, especially for small or medium-sized businesses.

4.3.11 RoBERTa

RoBERTa (Robustly Optimised BERT Pre-training Approach) is a transformer-based neural network commonly used for NLP purposes [32]. It is an advanced version of the previously described BERT. Developed by Facebook AI researchers in 2019 [45], RoBERTa is built on the architecture of BERT, but includes some changes in the pre-training methodology, as RoBERTa uses much larger amounts of training data (10 times larger than BERT) and trains for a longer time.

Another significant difference between these two algorithms is that RoBERTa also modifies the masking pattern in the pre-training phase by removing the next sentence prediction, which is a distinctive feature of BERT. Moreover, these minor modifications result in better performance on a variety of different tasks such as sentiment detection, NER (named entity recognition), question answering, language modelling, machine translation, etc [81].

I applied RoBERTa for sequence classification to my data using the Hugging Face transformers library and initialising the pre-trained model and tokeniser. This algorithm is a variant of RoBERTa specifically tuned for sequence classification tasks, which is particularly useful in the context of reviews. Next, as with BERT, I used the AdamW optimiser and the CrossEntropyLoss function from the Torch library for loss computation. The former updates the model parameters during training, while the latter calculates the difference between predicted and true labels.

Among others, RoBERTa has the following set parameters: padding=True, truncation=True, max_length=512, which means that the tokeniser has to pad or truncate the reviews so that they all have the same length of 512 tokens. The tokenised reviews are then used to create TensorDataset objects, the format of which can be interpreted by the RoBERTa model.

The training is performed for two epochs using randomly selected 10,000 rows (stratified) of my dataset, containing only columns with the text of the unprocessed review and its label. I have deliberately reduced my data in order to reduce the computational time of the model, as its complex architecture requires a significant amount of resources (GPU memory, computing power, etc), because my technical equipment has limited capacity, especially for the whole dataset.

The data is split into training, validation and test sets, with the proportion of test and validation data being

20%. After pre-processing, the dataset contains the tokenised input reviews, attention masks (indicating which tokens the review contains and which are padding), and the corresponding labels (1 for negative reviews and 0 for positive reviews). In each epoch, the function iterates through the training data batches, performing forward and backward propagation to update the model parameters.

The performance of the model is evaluated on both the training and validation sets using metrics such as accuracy, recall, precision and f1-score (Table 11). RoBERTa has shown significantly better results compared to BERT, achieving an average prediction accuracy of around 95% (and over 90% for other metrics). Its performance without aspect fine-tuning proved to be at the same level as BERT, which was specifically fine-tuned for text classification tasks.

	Loss	Tr Accur.	Tr Recall	Tr Precis.	Tr F1-score	Accur.	Recall	Precis.	F1-score
Epoch 1	0.08	0.96	0.96	0.96	0.95	0.95	0.95	0.94	0.94
Epoch 2	0.08	0.94	0.94	0.89	0.91	0.94	0.94	0.89	0.91
Weighted Av.	0.08	0.95	0.95	0.925	0.93	0.945	0.945	0.915	0.925

Table 11: Training and validation sets performance of RoBERTa model

Another positive aspect of this model is its robust pre-training. It uses a large corpus for pre-training and a variety of training techniques to generate a robust language model, resulting in a good ability to generalise to new tasks and domains. Being part of the popular Hugging Face Transformers library, RoBERTa also has many available resources and models, as well as a large community of users who contribute and share their code/results.

However, as mentioned above, this algorithm requires a large amount of computational resources, which makes it difficult to utilise. Another limitation, stemming from the previous one, could be the small amount of data and the training of the model for only 2 epochs. The complexity of the model together with the large number of parameters is also difficult to interpret. Not relevant to my data, but the algorithm has little or no support for non-English languages, so it is not suitable for other languages or multilingual applications.

4.3.12 Key Phrase and Entity Extractor

Another technique I used to predict review sentiment does not involve deep learning techniques, but instead uses rule-based algorithms and lexicon-based approaches to perform key phrase, entity and aspect extraction, as well as sentiment detection. Using code from Kaggle[22] as the basis for building a pipeline for my research, as shown in Figure 31, we use the **RAKE** (Rapid Automatic Keyword Extraction) algorithm [64] for key phrase extraction, the spaCy library for entity extraction, and Vader for sentiment analysis.

[22]https://www.kaggle.com/code/ritvik1909/keyphase-and-entity-extraction-sentiment-analysis

I started with RAKE key phrase extraction. It is an unsupervised, language-neutral technique that automatically identifies keywords and key phrases in the text and then ranks them according to their frequency in the text (but not in every sentence) and their co-occurrence with other words. RAKE was first introduced in a 2010 research paper by Rose et al. and has since become a popular tool for keyword extraction in NLP .

RAKE performs the following steps:

1. Tokenisation;
2. Stopword removal;
3. Phrase extraction based on word co-occurrence and frequency criteria;
4. Phrase scoring;
5. Keyword selection: phrases with the highest score are selected as key phrases.

RAKE is a fairly simple technique, but it is effective at extracting keywords and key phrases. It has also performed well on a variety of textual datasets. However, it is important to note that RAKE is still an unsupervised method and therefore may not capture all relevant keywords, especially in cases where there is some specialised vocabulary [64].

The next step in the pipeline is entity extraction using the spaCy library. This involves the identification and classification of named entities such as people, organisations, locations, etc. I then continue with the Vader library to perform sentiment analysis using logistic regression as a classifier, trained on the extracted features to predict the sentiment of the text as positive, negative or neutral. However, there is one limitation to logistic regression that is worth mentioning. It is a linear classifier and therefore may not be able to capture sophisticated non-linear relationships between the input features and the target.

The whole process of key phrase and entity extraction together with sentiment analysis is combined into a single pipeline (Figure 31) using the sklearn library **Pipeline class**, which allows the different transformation steps and a final estimator to be executed sequentially and efficiently.

The final part implements K-fold cross-validation to evaluate the performance of the pipeline. The input data is divided into k equal-sized folds, and the pipeline is trained and evaluated on each fold.

In order to illustrate the model, I have created a feature importance folds min-max scaled graph (Figure 32), which helps to get an idea of the relative importance of different features and how the importance of these features changes across multiple epochs or folds, and to visually assess the performance of each. It may also be useful for identifying any patterns that characterise particular folds or epochs.

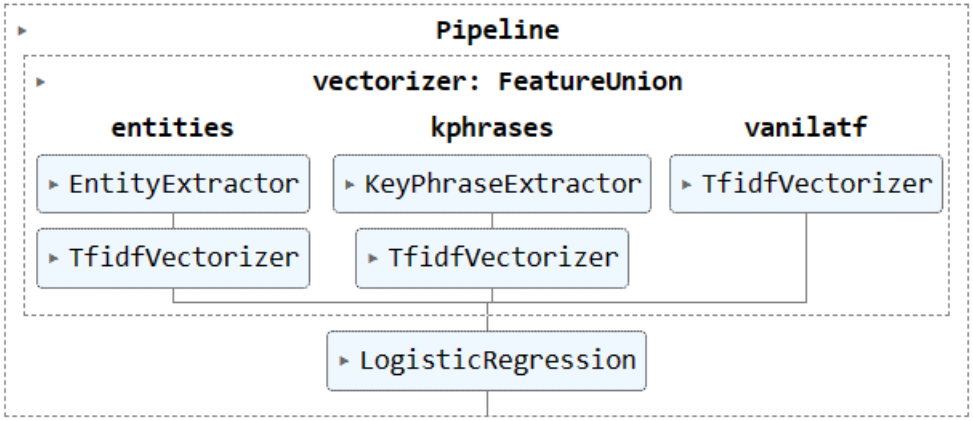


Figure 31: Pipeline of the key phrase and entity extractor model

Each of the 5 plots in Figure 32 represents the feature importance scores for a specific epoch, with the earliest fold at the top and the latest at the bottom. The x-axis is responsible for the individual features in the model. As I used a key phrase extractor, these are specific words or phrases. In addition, given the size of the data, there are many expressions that are frequent enough to be considered key, which obviously cannot be shown on the graphs. Nevertheless, I can still gain some useful insights from comparing the performance of the folds.

The y-axis shows the feature importance scores, which have been pre-normalised from -1 to 1 using the **minmax_scale** function from sklearn's pre-processing module. This scaling step is done to ensure that the feature importance coefficients are on a consistent scale and can therefore be easily compared across many other features. The height of each bar plays an important role in the interpretation of the graph. The taller the bar, the more important the feature was in that particular fold, and vice versa. You can also see red vertical lines on the graph. These were created with the aim of visually distinguishing between the different types of features, which are defined as a list of three integers, including CountVectorizer, TfidfVectorizer and custom feature transformers.

As can be seen in Figure 32, all the folds have more or less similar performance. In the first part (before the first red line) there are no visually detected leaders / outsiders. This is also the case for the second part, with the exception of fold 3, which has flatter parameters close to 0. The third set of custom feature transformers turned out to be the most important one for this particular model. It contains both the best and the worst features of the algorithm. In terms of successful folds, some slight improvements in the negatively scored features can be seen starting from the third set. In terms of the most important features, the same as before, fold 3 is a little behind. Despite these conclusions, it should be remembered that each fold represents a different subset of data, so it is also possible that certain features are simply less relevant for the third subset.

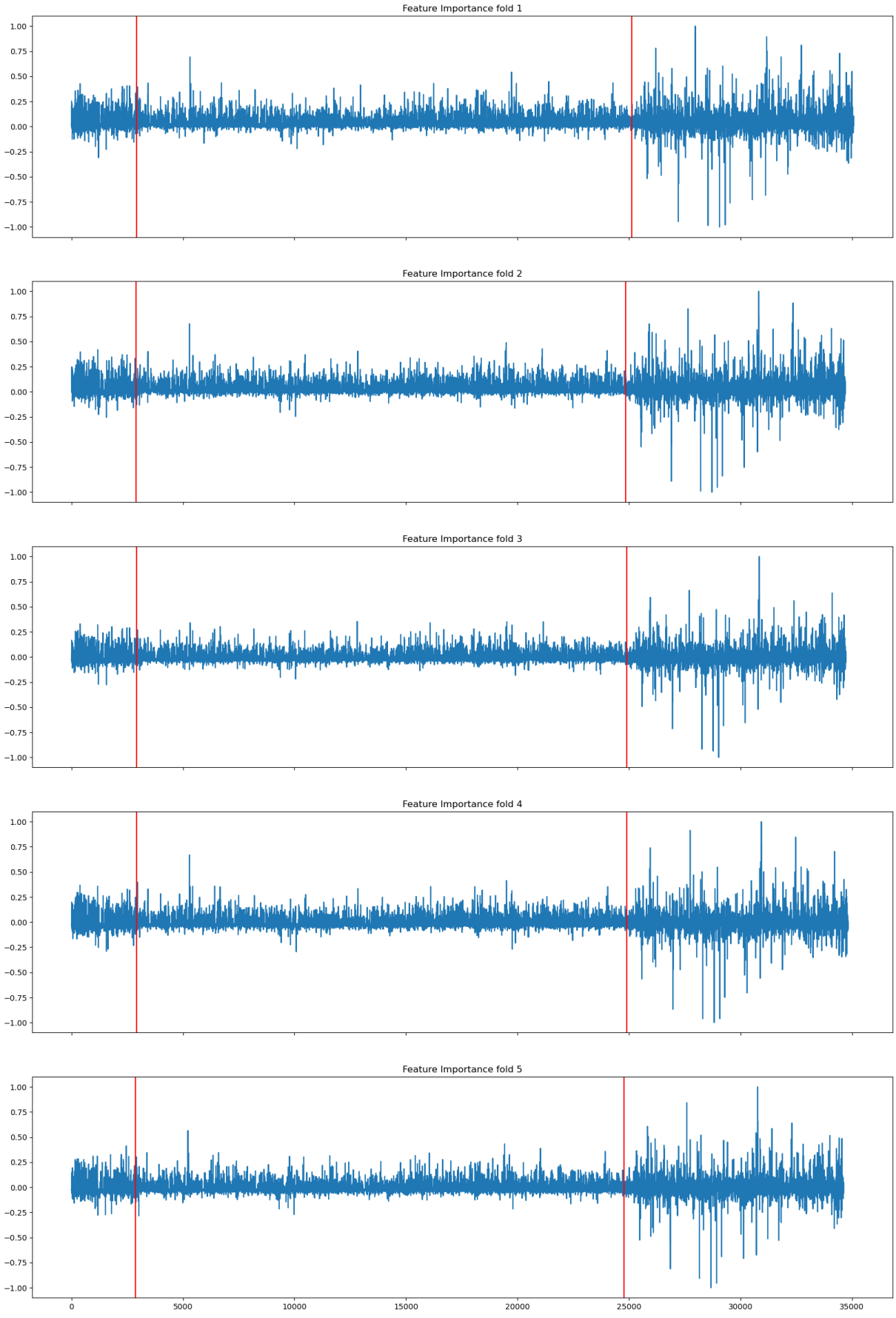


Figure 32: Feature importance folds min-max scaled graph

The overall results of the model are averaged across the folds to provide a more robust estimate of pipeline performance. Figure 33 shows the output of the pipeline classification report. The algorithm was able to achieve a high overall accuracy of 91%, despite the low parameter values of class 1 (negative reviews). The macro average for precision, recall and f1-score is quite low compared to the weighted average, with results above 90% for each metric.

	precision	recall	f1-score
0	0.97	0.94	0.95
1	0.35	0.56	0.43
accuracy			0.91
macro avg	0.66	0.75	0.69
weighted avg	0.94	0.91	0.92

Figure 33: Classification report of key phrase and entity extractor model

In the context of model evaluation for classification tasks, macro and weighted average are two commonly used techniques for calculating performance metrics [44]. The difference between macro and weighted average is how they deal with imbalances in the data. Macro-average, for example, first calculates the metric for each class separately and then takes the average. This method assumes that each class has the same weight, regardless of its frequency in the data. Macro-average may be useful when it is necessary to evaluate the performance of the model on each class independently, which is not the goal for my case.

The weighted average also first calculates the metrics for each class separately, but the final value is calculated as a weighted average across the classes, with the weights proportional to the number of samples in each class. The performance of each class is essentially weighted by its frequency. The weighted average is useful for my data because it helps to evaluate the overall performance of the model, taking into account the imbalanced nature of the dataset. Therefore, my weighted average results can be considered more representative, and being one of the highest of all the algorithms described above, indicate that the model works well [44].

4.3.13 PyABSA

PyABSA is an open source framework for ABSA (Aspect Based Sentiment Analysis) developed and first presented in the paper by Yang, Zhang and Li [90] – researchers at the University of Exeter and the Beijing Institute of Technology. The algorithm works as follows: it aims to identify aspects or entities present in the text and the sentiment associated with them. It is designed to work with different NLP tasks such as sentiment analysis, aspect extraction, aspect sentiment classification and opinion target extraction.

Although numerous models have been proposed for ABSA, they often differ in architecture and optimisation

techniques, making it difficult to reproduce results even with existing code. Specifically to address this issue and promote fair comparisons, the PyABSA algorithm has been introduced – a modular framework based on PyTorch for reproducible ABSA.

This model is extremely useful for novices or data scientists with limited programming skills, as it simplifies model training, evaluation and inference for various ABSA subtasks, and is easy to code. In addition, the developers have provided several pre-trained checkpoints, accessible through the Transformers Model Hub, to ensure reproducibility.

PyABSA follows a modularised organisation with five modules: template class, configuration manager, dataset manager, metric visualiser and checkpoint manager. This design, with minor adjustments, facilitates easy extension for different models, datasets or related tasks [90].

As mentioned above, PyABSA was developed in response to common ABSA problems, namely the increasing demand for more accurate and efficient systems. Therefore, the researchers tested the model on various datasets, including SemEval 2014 and SemEval 2016[23], and achieved state-of-the-art results.

I used the code openly provided by the authors of the paper on Github and used the PyABSA library to perform Aspect Based Sentiment Analysis (ABSA) on 10,000 samples of my data (reasons for data reduction are listed in the previous Subsections 4.3.10 and 4.3.11 and additionally for fair comparison with BERT and RoBERTa models).The training data is a restaurant review dataset called Restaurant16[24], which serves as a base for the extraction of aspect terms. It is a publicly available benchmark dataset from SemEval 2016 consisting of 1,600 customer reviews of restaurants covering four different aspects: food, service, ambience and price, which closely overlaps with my domain and is a perfect fit for my data. Each review in Restaurant16 is labelled with the aspects and their corresponding sentiments, which are positive, negative or neutral. Furthermore, the dataset was manually annotated by humans who were asked to manually identify aspects and their polarity. Therefore, it can be considered as high quality training data.

After importing the PyABSA library and the **AspectTermExtraction class**, which is used to extract aspect terms from text, I also loaded a dictionary of available checkpoints (pre-trained models). I set it to 'multilingual' and defined the 'auto_device' parameter to True, which means that the model will be loaded on a GPU if available, otherwise the model would use the CPU. The 'cal_perplexity' parameter was also set to True, so that during inference (the process of applying a pre-trained model) the model's perplexity score (how well the predictions are made on previously unseen data) is calculated. It then runs the batch prediction on my reviews and returns the dictionary containing the aspect terms and their predicted sentiment. As an output, I got the aspect terms

[23]Semantic Evaluation Workshops – series of annual workshops focusing on the evaluation of computational semantic analysis systems and providing a forum for researchers to share their work on a variety of tasks related to NLP, https://semeval.github.io/

[24]https://alt.qcri.org/semeval2016/task5/

extracted from my review texts along with their predicted sentiment.

```
[(('staff', 'Positive'), 2888),
 (('location', 'Positive'), 2500),
 (('room', 'Negative'), 1373),
 (('room', 'Positive'), 1101),
 (('breakfast', 'Positive'), 767),
 (('bed', 'Positive'), 628),
 (('breakfast', 'Neutral'), 431),
 (('staff', 'Negative'), 416),
 (('bathroom', 'Negative'), 416),
 (('hotel', 'Positive'), 389),
 (('rooms', 'Negative'), 383),
 (('rooms', 'Positive'), 378),
 (('location', 'Negative'), 365),
 (('room', 'Neutral'), 349),
 (('breakfast', 'Negative'), 322),
 (('bed', 'Negative'), 298),
 (('bathroom', 'Positive'), 252),
 (('location', 'Neutral'), 243),
 (('view', 'Positive'), 240),
 (('service', 'Positive'), 234),
 (('staff', 'Neutral'), 196),
 (('food', 'Positive'), 194),
 (('shower', 'Negative'), 166),
 (('beds', 'Positive'), 166),
 (('bar', 'Neutral'), 164),
 (('stay', 'Positive'), 158),
 (('facilities', 'Positive'), 140),
 (('building', 'Positive'), 131),
 (('shower', 'Positive'), 127),
 (('bar', 'Positive'), 125),
```

Figure 34: 20 most common aspects with sentiments and frequency

In Figure 34 you can see the 20 most popular aspects together with their sentiments that characterise the dataset. I did not specify a list of specific aspects to be considered in this analysis, but still got all the same ones in the output, such as *staff* and *location* being the most popular, and *room* with the negative comments prevailing over the positive experience. The same pattern as for *room* is followed by *bathroom*. All other aspects or facilities (*breakfast*, *food*, *view*, *service*) have significantly more positive than negative comments. Moreover, no 'strange' aspects (not related to the hotel reviews) were extracted, which is a sign of a strong PyABSA performance. The comparison of the most common aspects sentiments in numbers is also shown in Figure 35 for a more convenient overview.

In order to build a so-called 'prediction model', I added the output in the form of lists as two additional columns in the pandas dataframe, along with the review itself and its real label (based on the numerical rating of the hotel provided by the reviewer herself). My model, by processing the PyABSA aspects sentiments and following some simple patterns, performs the assignment of the overall review label. This manually tailored algorithm is inspired by the idea that the parts of the whole identify the whole itself. Similarly, aspects and their sentiments could be used to measure customer satisfaction.

	Positive	Neutral	Negative
room	1101	349	1373
location	2500	243	365
staff	2888	196	416
breakfast	767	431	322
restaurant	18	2	10
price	120	96	90
food	194	49	65
bed	628	112	298
stay	158	19	14
service	234	36	80
bathroom	252	71	416
facilities	140	24	23

Figure 35: Sentiment comparison of the most popular aspects

In the original model, I created a function that takes a list of aspect sentiments and assigns a label based on the following conditions:

1. If the list of sentiments is empty, the label is assigned as 'Positive' (meaning that the review does not contain any text, which in most cases would logically indicate the absence of the negative experience that people usually tend to express when there is one);
2. If all aspect sentiments are 'Positive', the label is assigned as 'Positive';
3. If all aspect sentiments are 'Negative', the label is assigned as 'Negative';
4. For mixed sentiments (containing mostly 'Positive' and 'Negative' aspects), the label is assigned as 'Positive' if the ratio of positive sentiments is greater than or equal to 2/3;
5. If the ratio of negative sentiments is greater than 3/4, the label is assigned as 'Negative';
6. Default label if the review does not fall into any of the above conditions – 'Positive'.

The function described above was originally applied only to the list of sentiments, the aspects themselves were not considered. The 'predicted' labels were also stored in the data frame. In order to evaluate the performance of my customised algorithm, I built another function that takes the true and predicted labels as input and compares them. It also calculates the accuracy, precision, recall and f1-score metrics based on these labels. The resulting values can be found in Table 12. Overall, it could achieve quite high accuracy and showed excellent precision. However, I tried a few more assumptions to see if they could potentially improve the performance.

In Fine-tuned Model #1, on top of the assumptions of the original model I added another one:

> Positivity and negativity dominate over neutrality (if there are labels 'Positive' and 'Neutral', the overall sentiment is 'Positive', the same with the 'Negative' label due to the subjugation of neutrality).

Adding this simple assumption helped me to improve the original model, namely accuracy increased by 2%, recall by 6% (i.e. the increase in the proportion of correctly predicted negative labels out of all truly negative cases), which consequently improved the f1-score despite the slight drop in precision. In this algorithm, I focus more on the f1-score as it is a harmonic mean of precision and recall and provides a balanced measure of the model's performance. An increase in this metric indicates a better trade-off between precision and recall.

In Fine-tuned Model #2, I additionally decided to test another assumption which seemed to be riskier:

> Customers tend to mention things they did not like first if they had an overall bad experience, and then briefly mention the benefits. Conversely, positive things tend to be mentioned first if the customer was happy.

To test this assumption, I had to derive the sentiment index from the sentiment list. It is worth noting that aspect labels appear in the list according to their mention in the review, which was useful in this experiment. A lower number would mean that this sentiment is described earlier in the text.

Unfortunately, this setup did not show any positive results. It could only improve the accuracy by 1% compared to the previous model, so it was quite good at avoiding false positive predictions (false negative labels), but still could not outperform the accuracy of the original model.

In Fine-tuned model #3, I did not add any additional assumptions, I just applied the original ones, taking into account only the important aspects. By this I mean the list of common aspects that I defined to exclude the influence of unimportant things (like the absence of tea bags or stains on the tablecloth). These aspects are: 'room', 'bathroom', 'bedroom', 'bed', 'tv', 'balcony', 'air conditioning', 'service', 'staff', 'reception', 'receptionist', 'food', 'restaurant', 'breakfast', 'location', their plurals and equivalents. In all previous models, all aspects were considered equally in the prediction. As can be seen in Table 12 I achieved the highest accuracy compared to other PyABSA models and better recall and f1-score than in the original model. Precision was 1% better than in the neutrality subjugation algorithm, but still lower than in the first technique. Since the f1-score is the most important factor for the model evaluation, the second model (with positivity and negativity dominance over neutrality) showed the best results, even though the accuracy was lower than 90%, as in most of the previously described subsections.

Overall, one of the positive aspects of the PyABSA algorithm is its ability to perform multiple ABSA tasks simultaneously, making it much more efficient than other traditional models. The algorithm is also highly customisable and flexible, allowing users to fine-tune it for their specific needs or easily extend it to different

	Accuracy	Recall	Precision	F1-score
Original Model	0.8537	0.879	0.9623	0.9187
Fine-tuned Model #1 (neutrality subjugation)	0.8715	0.9469	0.9467	0.9468
Fine-tuned Model #2 (indexing)	0.5734	0.5732	0.956	0.7167
Fine-tuned Model #3 (important aspects domination)	0.868	0.9025	0.9548	0.9279

Table 12: Comparison of the original and fine-tuned manually tailored PyABSA model performances

models or datasets due to its flexible organisation and modular framework. In addition, PyABSA includes data annotation features to address data scarcity (lack of labelled training data) in ABSA.

However, a potential drawback of the algorithm could be its reliance on a large amount of labelled data for training, which may be a barrier for users who wish to develop their own model pipeline rather than using pre-trained models. In addition, despite its flexibility, the algorithm may not perform as well on datasets other than those on which it was trained, so it may not be suitable for all domains.

4.3.14 LDA

Latent Dirichlet Allocation (LDA) is a generative probabilistic model widely used for topic modelling, first applied to machine learning in 2003 by David Blei, Andrew Ng and Michael Jordan [5].

This algorithm is designed to discover latent (hidden) topics within documents/text corpus. Its main assumption is that each document consists of a mixture of different topics and that each topic is characterised by a distribution of words. LDA is designed to identify these latent topics and their associated word distributions.

The mechanism of LDA starts by randomly assigning each word in each text to one of the themes. It then goes through the iterative process and loops over two steps until convergence, which are:

1. Topic assignment: Each word in a document is assigned to a topic based on how similar the word is to other words already in the available topics (topic-word distribution) and how prominent the topic is in the document (document-topic distribution: document has a high proportion of this topic);

2. Topic update: Once the topic assignments for words are updated, LDA recalculates the topic-word distribution, making the refined topics more accurate and representative of the content.

The next step focuses on the analysis of the results. After the LDA has converged (words and themes are connected), it provides the probabilities of themes for each text, as well as the words that characterise the theme. This result helps to understand the main themes of the texts, but what is even more interesting for my research is that it allows me to derive latent aspects from the reviews.

Usually, algorithms are only able to detect direct aspects – the words already used in the sentence: *the room **[aspect]** was big and bright*. Latent aspects, on the other hand, are invisible to most tools. In the sentence *hotel*

is far from the centre it is very likely that *hotel* will be misinterpreted as an aspect, even though the reviewer is talking about the location. This is a huge disadvantage from a business point of view, as the owner/decision maker will see the aspect *hotel* with a negative sentiment as a result of the analysis, instead of getting a real image and source of negativity in the comment.

To apply LDA to my dataset, I used the **gensim** library. I deliberately chose the cleaned and tokenised review text to avoid meaningless auxiliary or stop words that would draw additional 'attention' to themselves (all unedited reviews obviously contain these words and are assigned to the topic they belong to).

I then created a dictionary from my comments. The **corpora.Dictionary** class from gensim is used to generate a dictionary by mapping words to their unique IDs based on the pre-processed documents. Another important step would be to create a document-term matrix. Each review text is represented as a bag of words, where each word is identified not only by its ID, but also by the frequency of its occurrence, which is combined into a list of tuples.

Finally, I applied the LDA algorithm itself using the **gensim.models.LdaModel** class, where I specified all the parameters described above, such as the input document-term matrix, the number of topics, id2word as the dictionary mapping, and started the iteration process over the corpus to train the model.

In a first setup I defined the number of topics to be 10. As a result from the whole dataset LDA identified following topics:

1. Topic 0 (pricing and booking experience): 0.070*'hotel' + 0.033*'pay' + 0.032*'room' + 0.030*'book' + 0.029*'price';

2. Topic 1 (room and location): 0.124*'room' + 0.052*'location' + 0.045*'small' + 0.042*'hotel' + 0.036*'nice';

3. Topic 2 (overall stay experience): 0.056*'hotel' + 0.048*'stay' + 0.045*'everything' + 0.031*'staff' + 0.029*'nothing';

4. Topic 3 (bar and restaurant area): 0.116*'bar' + 0.064*'area' + 0.037*'restaurant' + 0.026*'drink' + 0.020*'top';

5. Topic 4 (proximity and location): 0.043*'hotel' + 0.038*'station' + 0.036*'walk' + 0.035*'close' + 0.027*'location';

6. Topic 5 (staff and helpful service): 0.137*'staff' + 0.081*'location' + 0.071*'friendly' + 0.066*'room' + 0.059*'helpful';

7. Topic 6 (room comfort and facilities): 0.072*‘room’ + 0.055*‘bed’ + 0.025*‘shower’ + 0.025*‘bathroom’ + 0.015*‘water’;
8. Topic 7 (breakfast and food): 0.134*‘breakfast’ + 0.058*‘good’ + 0.026*‘food’ + 0.022*‘pool’ + 0.022*‘wifi’;
9. Topic 8 (check-in process and staff): 0.030*‘room’ + 0.024*‘staff’ + 0.023*‘check’ + 0.023*‘u’ + 0.017*‘hotel’;
10. Topic 9 (room conditions and noise): 0.071*‘room’ + 0.033*‘work’ + 0.031*‘air’ + 0.031*‘night’ + 0.029*‘window’.

The distribution of the above topics is obviously not perfect. There are at least three topics where the location is mentioned, the same with the staff (like helpful or check-in) and the room, which appears as a separate topic and as part of the other unrelated topic. This confusion may be due to the wrong number of topics. For example, I ‘ordered’ the algorithm to find 10 topics, so it was forced to do so and only identified some mixed ones. However, the best measure of the technique’s performance is its practical application to real sentences.

As you can see in Figure 36, I chose three sentences: *hotel is far from the centre* (latent aspect *location*), *the girl at the front desk was really nice* (latent aspect *staff*) and the last sentence with direct aspects for comparison *menu is extensive and a bar with a live music* (aspects *menu* and *bar*), the last two sentences being taken from Mohammad Forouhesh and his research on latent aspect detection [28].

```
# Apply the model to new documents
new_document = "Hotel is far away from centre"
new_preprocessed_doc = word_tokenize(new_document.lower())
new_preprocessed_doc = [word for word in new_preprocessed_doc if word not in stop_words]
new_doc_bow = dictionary.doc2bow(new_preprocessed_doc)
new_doc_topics = lda_model.get_document_topics(new_doc_bow)
print(f"Topics for new document: {new_doc_topics}")
```

```
Topics for new document: [(0, 0.020004403), (1, 0.020002127), (2, 0.020002365), (3, 0.020000188), (4, 0.8199882), (5, 0.0200009
6), (6, 0.02000025), (7, 0.020000188), (8, 0.020001112), (9, 0.020000253)]
```

```
lda("the girl at the front desk was really nice")
```

```
Topics for new document: [(0, 0.016711181), (1, 0.24264096), (2, 0.016711181), (3, 0.016711181), (4, 0.016711181), (5, 0.18329
5), (6, 0.016716333), (7, 0.016711181), (8, 0.4570806), (9, 0.016711181)]
```

```
lda("menu is extensive and a bar with live music")
```

```
Topics for new document: [(0, 0.016691146), (1, 0.016696705), (2, 0.45187145), (3, 0.016695065), (4, 0.016692903), (5, 0.016692
309), (6, 0.016692834), (7, 0.23904333), (8, 0.0166946), (9, 0.19222967)]
```

Figure 36: Output of the first LDA setup implementation demonstrated on three sentences

The output in Figure 36 is a list of tuples where the first element is a topic number and the second is the probability that the text belongs to that topic. The first sentence was assigned to group 4 with 82% probability,

topic with proximity and location. Surprisingly, all other groups had the same low probability, even those where location was in the top 5 words of the topic. For the second phrase, there was not a single strong favourite, but still the highest probability was for topic 8 'check-in and staff' (45%) and a little lower for topics 1 and 5 (24% and 18%). The allocation of the fifth theme is quite obvious as it is also related to service, but the first one was not explainable. The third sentence was completely wrong. The highest probability was given to theme 2, which is devoted to the overall experience of the stay, while theme 3, which would logically be more in line with this sentence, received a probability of just 2%.

Overall, the results showed correct connections but were somewhat mixed. In the case of the direct aspects, the outcome was unsatisfactory.

Due to the unclear topic distribution I decided to reduce the number of topics to 5 in the second setup to see if this would improve the model performance. The new groups look like this:

1. Topic 0 (breakfast and overall experience): 0.059*'breakfast' + 0.037*'good' + 0.021*'price' + 0.020*'bar' + 0.019*'room';
2. Topic 1 (staff and check-in): 0.025*'room' + 0.016*'staff' + 0.016*'time' + 0.015*'check' + 0.014*'get';
3. Topic 2 (room characteristics, particularly size): 0.100*'room' + 0.035*'bed' + 0.026*'small' + 0.023*'bathroom' + 0.021*'location';
4. Topic 3 (staff friendliness): 0.102*'staff' + 0.055*'location' + 0.046*'room' + 0.046*'friendly' + 0.045*'great';
5. Topic 4 (proximity and location): 0.035*'close' + 0.034*'station' + 0.033*'walk' + 0.026*'location' + 0.024*'metro'.

Again, despite some clearer topic detection, some words (such as *location*) appeared in three topics. Identifying fewer groups might miss some important aspects of the reviews if they are not mentioned so often, so I will not reduce the number of topic models further.

For the evaluation I compared the same three sentences. *The hotel is far from the centre* again was correctly assigned to proximity and location. *The girl at the front desk was really nice* has a leader this time – topic 1 (staff and check-in). The third review also got a correct label in this setup – topic 0, which is related to the food and bar area of the hotel. These results are not perfect, but significantly better than in the previous setup.

Overall, LDA is a powerful algorithm for uncovering latent themes. However, it's worth noting some advantages and disadvantages. On the positive side, it allows automatic discovery of topics in an unsupervised manner and can handle high-dimensional text data. Secondly, it provides well-interpretable results, as each

```
lda_new("Hotel is far away from centre")
Topics for new document: [(0, 0.050033458), (1, 0.050083887), (2, 0.050026573), (3, 0.050000604), (4, 0.7998555)]

lda_new("the girl at the front desk was really nice")
Topics for new document: [(0, 0.033750862), (1, 0.5948618), (2, 0.03390067), (3, 0.30342975), (4, 0.034056954)]

lda_new("menu is extensive and a bar with live music")
Topics for new document: [(0, 0.8665099), (1, 0.033338338), (2, 0.033414774), (3, 0.033398647), (4, 0.03333834)]
```

Figure 37: Output of the second LDA setup implementation demonstrated on three sentences

topic is represented by a distribution of words. At the same time, however, LDA is sensitive to the choice of hyperparameters and, in particular, requires the number of topics to be specified in advance, which, as in my case, can be an obstacle to obtaining meaningful results. It also assumes that each document contains a mixture of all topics, which may not be true if reviewers do not mention some aspects.

4.3.15 Chat GPT

Another model that I would like to consider in the frame of my research is an absolutely no-code option that to my surprise still showed very good results in the field of sentiment and aspect-based sentiment analysis – **Chat GPT** (Generative Pre-trained Transformer)[25]. It has recently become an extremely popular tool that can help to solve many tasks, so I allowed for the possibility that it might also be helpful in NLP-related tasks.

According to the Chat GPT itself [15], it is an AI language model developed by OpenAI. It processes the user's input (query) and generates a corresponding response in a human way, mimicking a conversation. It has been trained on a variety of data sources in different domains and can help with a wide range of topics: answering questions, providing explanations or recommendations, writing code, rewriting texts, etc.

I randomly selected three reviews from my dataset that had different sentiments: positive, neutral and negative, in order to compare the tool's capabilities. I asked Chat GPT to provide an overall sentiment for the text, including the aspects it consists of and the mood/attitude towards them.

These three review texts are:

1. Negative review: *My room was dirty and I was afraid to walk barefoot on the floor which looked as if it was not cleaned in weeks White furniture which looked nice in pictures was dirty too and the door looked like it was attacked by an angry dog My shower drain was clogged and the staff did not respond to my request to clean it On a day with heavy rainfall a pretty common occurrence in Amsterdam the roof in my room was leaking luckily not on the bed you could also see signs of earlier water damage I also saw*

[25] https://openai.com/blog/chatgpt

insects running on the floor Overall the second floor of the property looked dirty and badly kept On top of all of this a repairman who came to fix something in a room next door at midnight was very noisy as were many of the guests I understand the challenges of running a hotel in an old building but this negligence is inconsistent with prices demanded by the hotel On the last night after I complained about water damage the night shift manager offered to move me to a different room but that offer came pretty late around midnight when I was already in bed and ready to sleep Great location in nice surroundings the bar and restaurant are nice and have a lovely outdoor area The building also has quite some character;

2. Neutral review: *Rooms are nice but for elderly a bit difficult as most rooms are two story with narrow steps So ask for single level Inside the rooms are very very basic just tea coffee and boiler and no bar empty fridge Location was good and staff were ok It is cute hotel the breakfast range is nice*;

3. Positive review: *All great Rooms were stunningly decorated and really spacious in the top of the building Pictures are of room 300 The true beauty of the building has been kept but modernised brilliantly Also the bath was lovely and big and inviting Great more for couples Restaurant menu was a bit pricey but there were loads of little eatery places nearby within walking distance and the tram stop into the centre was about a 6 minute walk away and only about 3 or 4 stops from the centre of Amsterdam Would recommend this hotel to anyone it s unbelievably well priced too*.

The output of the query described above can be seen in Figure 38. Overall, the results are very satisfactory, especially in the case of negative and positive reviews. Chat GPT was able to correctly classify the general sentiment and provided a large list of aspects from the texts. However, the latter underwent a massive generalisation. For example, in the case of the expression '*my room was dirty*' in a negative review, most algorithms would define *room* as an aspect, but Chat GPT correctly identified *cleanliness* as an aspect in a more human way. This would give the owner more useful information, since the room itself (its size) cannot be changed, unlike hygiene, which can easily be improved.

Unfortunately, this kind of generalisation can also have negative consequences. Detecting the *maintenance* aspect in a negative review would be too general in this example. Chat GPT left out the important details of what exactly was a negative experience, even though the reviewer mentioned it: *the roof was leaking*, *shower drain was clogged*. The same goes for *room amenities* (especially *basic tea coffee*, *empty bar and fridge*) in the neutral review. These are the things that can and should be changed to improve the guest's stay, but in the case of heavy reliance on this tool would go unnoticed.

Apart from this, as mentioned in the paper by Wang, Xie, Ding, Feng and Xia [86], which was exclusively dedicated to the investigation of Chat GPT capabilities in sentiment detection, this tool proved not to work

Figure 38: Chat GPT sentiment detection from originally negative (left), neutral (centre) and positive (right) reviews

best with neutral review in my research and tourism domain and struggled to identify it, although this method showed overall good performance.

Actually, despite the fact that I am focusing on machine learning techniques, I understand that they require some basic programming skills, appropriate hardware and data format. However, Chat GPT could become a great, easy to understand and free alternative for hotel owners. However, it should be noted that this tool is not able to process reviews in bulk and does not offer the option of uploading a file with the dataset if the user does not want to simply copy every comment in the chat. Also, there may be cases where Chat GPT generates incorrect or generic responses. Therefore, this method would require some extra effort to manually verify the information or analysis provided, at least with the help of critical thinking, as the user may not be a technical expert.

4.4 Customer Prioritisation Model

The final phase of my research is to present the results in a user-friendly format. Since the main objective is to help hotel owners gain competitive advantage by improving overall customer satisfaction, there is a great need for a tool that is extremely easy to use, especially for users without any programming background, but at the same time is efficient for customer ranking and customisable for the specific purposes of the business. Taking all this into consideration, I decided to use **Streamlit**[26] to create the prioritisation model.

Streamlit is an open source and free framework that was created as a result of collaboration between engineers from Uber, Twitter, Stitch Fix, Dropbox and other major companies, with the idea of enabling data scientists and back-end developers to build web applications and interactive dashboards without requiring front-end web development experience [79]. Being a Python-based library for experts with knowledge of this programming language, Streamlit consists of a simple set of commands that allow the user interface to be interactive and customisable. For example, this tool offers the possibility of adding sliders, boxes, date ranges, etc, which in my case were used for filtering and customer ranking. It is also convenient for working with Pandas dataframes – the main format of my data. I could easily upload my dataset to GoogleDrive with public access and access it via file url.

One of the fundamental principles of Streamlit is to embrace Python scripting. Streamlit applications are scripts that run in sequential order with no hidden state. Another principle is to treat widgets as variables. Streamlit does not rely on callbacks for interactivity, but instead re-runs the entire code from top to bottom whenever a user changes the input through interaction in the application. It can also handle large amounts of data and reuse it efficiently to reduce computation time. Streamlit has a caching mechanism specifically for this purpose. This works in such a way that Streamlit applications can download data only once and then simply reuse it in subsequent runs, resulting in simpler and faster applications [80].

I have designed three apps in Streamlit, available for public use, which serve different purposes and offer a variety of choices to analyse the customer reviews:

1. Basic filtering – https://sofiiapiven-repo-streamlit-exp2-5ypu2r.streamlit.app/;
2. Filtering based on my own developed prioritisation model – https://sofiiapiven-repo-prio-model-w0z4ak.streamlit.app/;
3. Adjustable re-ranking model where user can set up her own criteria for prioritising the reviews – https://sofiiapiven-repo-interactive-ranking-kiss21.streamlit.app/.

[26] https://streamlit.io/

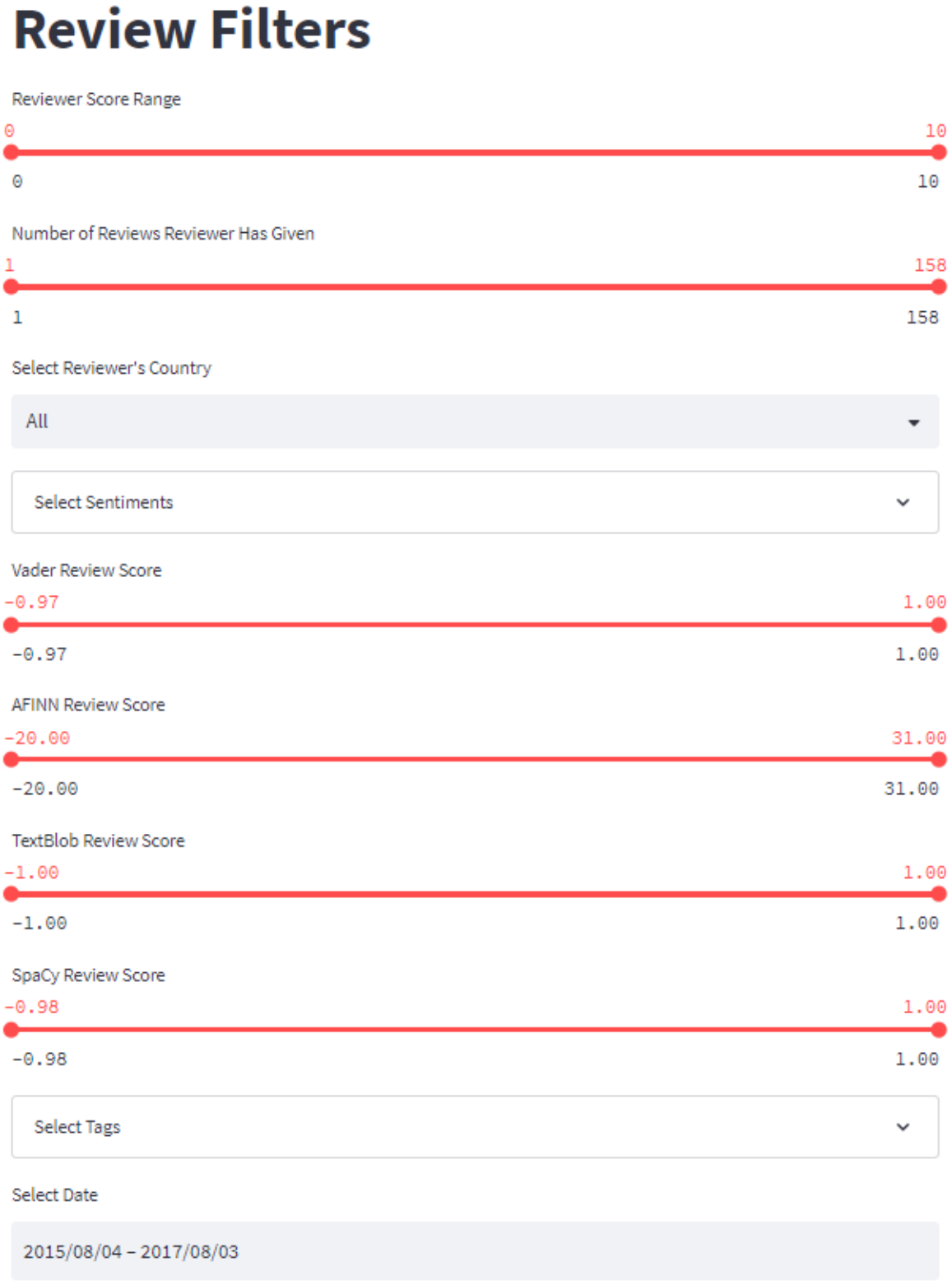


Figure 39: Interface of the filtering model in Streamlit

4.4.1 Basic Filtering

Figure 39 shows the interface of the first application dedicated to the basic filtering model. In this application, the user can customise the ranges of the following Pandas dataframe columns: the score given to the hotel by the reviewer, the number of reviews the reviewer has given in the past, and the overall sentiment of the review (scores from Vader, AFINN, TextBlob and SpaCy). It is also possible to select the country of the reviewer, the sentiments of all the aspects derived from the text of the comment and tags such as leisure or business trip, stayed 1 or 2 nights, etc (Figure 40), because I wanted not only to show the characteristics of the review itself, but also to keep some characteristics related only to the personality of the reviewer. It is also possible to adjust the dates within which the review was left.

All the filters were taken from the original dataset, with the exception of overall and aspect sentiment. The latter is divided into three subcategories, positive, neutral and negative, and allows the hotel owner to select the aspects of interest from the list of all aspects derived using the PyABSA algorithm described above. Even though some of the aspects appear in only a few reviews and may not contain much information, I thought it was important to keep them all in the filter and let the users decide what is of greater interest to them.

For this web app, I used the same reduced dataset of 10,000 samples that was used in all the sophisticated prediction models because of the inefficient computation time. The output of the filters is the review texts that correspond to the given input parameters.

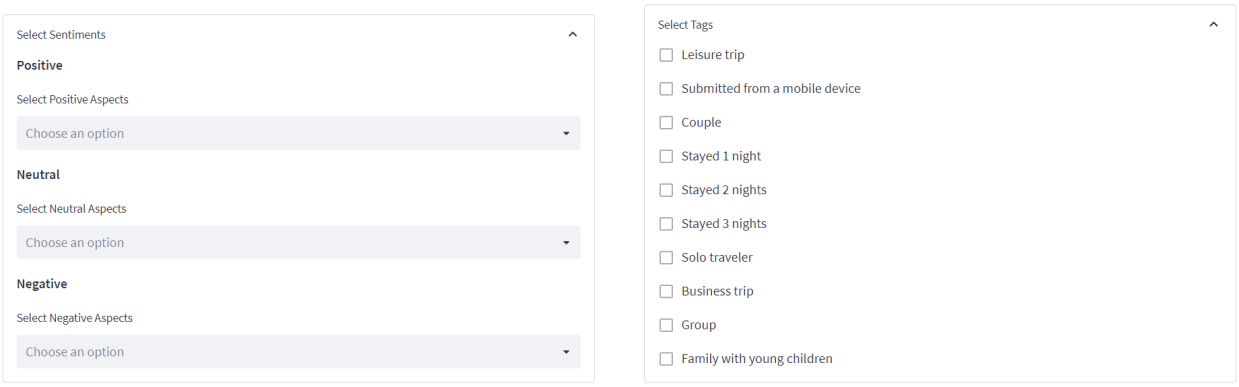


Figure 40: Closer look at filters in Streamlit app: Select Sentiments (left) and Select Tags (right)

4.4.2 Filtering Based on a Prioritisation Model

As the basis for this web application, I took a basic filtering model, but extended it with my own ranking model, which would prioritise the review text according to three newly introduced parameters: the negativity of the comment, the customer's previous experience and the possibility to influence the aspect.

The first has the lowest weight of 0.2 in the customer rank calculation. This means that negative reviews

(with a value of 1 in the negativity column) are given priority over positive ones and are displayed at the top of the output. The reason for this is that unhappy customers usually need more urgent feedback and attention to avoid radical action on the part of the customer. Of course, positive experiences are also appreciated, but they do not lead to undesirable consequences such as sharing bad reviews on social media in case of a later response.

The next parameter used to calculate an overall rank is customer expertise with a weight of 0.3. It is computed by dividing the number of previous reviews given by a reviewer by the maximum number of reviews given by any reviewer in the dataset. It allows the more 'mature' travellers to be ranked higher. The implication behind this is that reviewers who have given more reviews in the past have more travel experience and theoretically should have more expertise about the positive and negative sides of the hotel and might be more objective. I therefore added another slider to the app with the reviewer's experience based on my assumptions.

The last but most important feature for my ranking system is the changeable aspect score. What I mean by this is that, despite bad reviews about the size of the room or the fact that the hotel is not centrally located, the hotel owners are unfortunately unable to improve this. These are constant aspects that do not provide any meaningful information on how to improve. Therefore, I decided to focus on the reviews that give some feedback on aspects that can be changed, and introduced a special parameter for this purpose.

To identify the most valuable aspects from the huge list of all the things mentioned in the reviews, I turned to stochastic gradient boosting [29] – a learning method that combines traditional gradient boosting, where the entire training set is used to update the model after each iteration, with stochastic sampling. This technique introduces random training subsets (mini-batches) to reduce overfitting and improve generalisation [13].

Stochastic gradient boosting also introduces randomness into feature selection. For each weak learner (an algorithm that performs a little better than random guessing, usually referring to Decision Trees in this context), this method considers a random subset of features for splitting the data. This ensures that the weak learners do not rely too heavily on any single feature [29].

To apply stochastic gradient boosting to my data, I used the **XGboost** (distributed gradient-boosted decision tree (GBDT)) machine learning Python library [51]. In order to prepare my dataset in an appropriate way, I selected only those columns that refer only to the aspects derived by PyABSA. Each feature originally had three values, namely 0 for a neutral mention in the review, 1 for positive and -1 for negative. I then created a **DMatrix** [17] – a class with a specific internal data structure needed for XGBoost, which is optimised for both memory efficiency and training speed.

The next step was to set the parameters. I used binary classification for the objective in combination with the use of logistic regression. The evaluation metric was set to 'logloss', which is widely used in binary classification tasks, while the 'random seed' was specified at 42 for better reproducibility.

The model was then trained using the parameters and the dataset. As a result of the training, I identified the weights for all the features, which in this context means the number of times a feature is used to split across all the trees in the model. After normalisation, all columns were sorted by descending weights. The output of stochastic gradient boosting can be seen in Figure 41 for the 25 most important features that had the greatest influence on the review label.

```
location: 0.12844036697247707
staff: 0.0963302752293578
room: 0.09174311926605505
rooms: 0.06422018348623854
breakfast: 0.05045871559633028
bar: 0.04128440366972477
bed: 0.03669724770642202
bathroom: 0.03669724770642202
water: 0.03669724770642202
price: 0.03211009174311927
bathrooms: 0.03211009174311927
area: 0.027522935779816515
fridge: 0.022935779816513763
walls: 0.022935779816513763
hotel: 0.01834862385321101
window: 0.01834862385321101
wine: 0.01834862385321101
service: 0.013761467889908258
shower: 0.013761467889908258
food: 0.013761467889908258
reception: 0.013761467889908258
reception_staff: 0.013761467889908258
floor: 0.009174311926605505
beds: 0.009174311926605505
bedrooms: 0.009174311926605505
```

Figure 41: Results of stochastic gradient boosting applied to aspects

Considering the result of the model above, I selected only those aspects that were important for label prediction and at the same time could be changed by the hotel owner. The final list used for the modifiable aspect score includes [‘bathroom’, ‘bedroom’, ‘bed’, ‘tv’, ‘balcony’, ‘ac’, ‘air_conditioning’, ‘bar’, ‘water’, ‘price’, ‘fridge’, ‘walls’, ‘tee coffee’, ‘service’, ‘staff’, ‘reception’, ‘receptionist’, ‘food’, ‘restaurant’, ‘window’, ‘wine’, ‘breakfast’, ‘bathrooms’, ‘shower’, ‘reception_staff’, ‘floor’, ‘beds’, ‘bedrooms’].

This score is calculated by summing the values in the columns described above and then dividing by the total number of aspects in the list. The sign of the score is then inverted by multiplying it by -1 because, as I explained earlier, I am more interested in negative ratings in the prioritisation model. Negatively rated aspects have a value of -1, so by inverting them I can increase the total score. I also added a slider to regulate its range.

The overall customer ranking score is calculated as a combination of the factors detailed above. It is the weighted sum of the reviewer’s experience score, the sentiment score for modifiable aspects and the negative aspect score. All reviews have been sorted and stored by rank, so when the user changes the input in the filters

in the app, the output will always be sorted by rank (Figure 42).

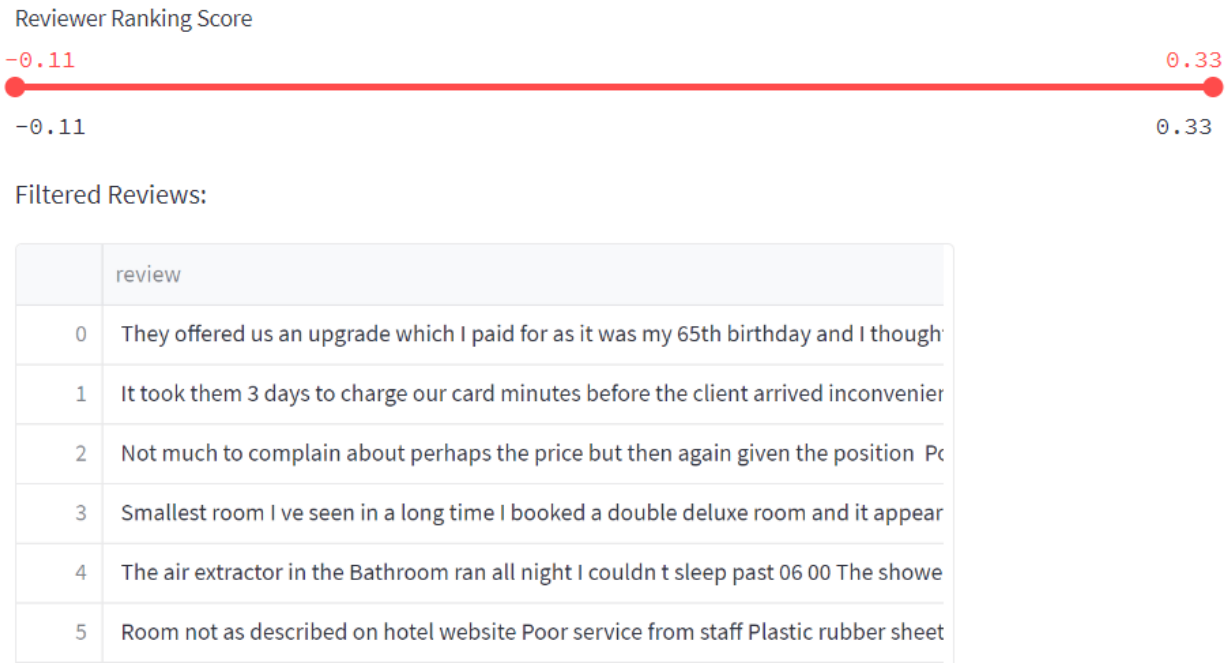


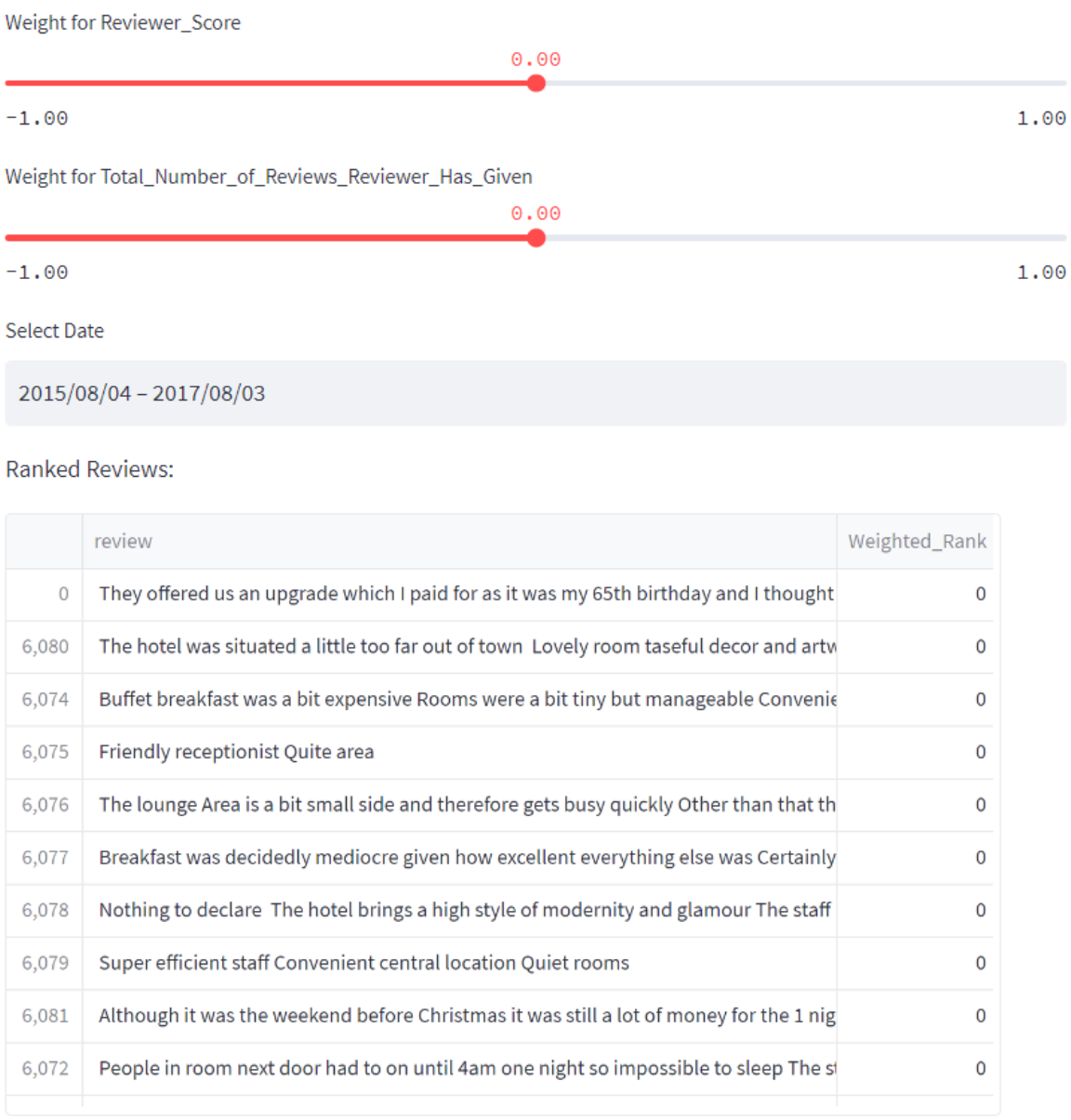
Figure 42: Customer rank filter and output of the manually tailored prioritisation model

4.4.3 Adaptable Re-ranking Model

This approach differs from the previous two in that it allows the user to choose the parameters that should influence the prioritisation and their weights. Furthermore, this model does not include any filtering, it simply re-ranks the rows and displays them in a sorted way (but all the rows are still visible). The application focuses on the aspects and their sentiments, namely these: 'room', 'bathroom', 'bedroom', 'bed', 'tv', 'balcony', 'ac', 'air_conditioning', 'tea_coffee', 'service', 'staff', 'reception', 'receptionist', 'food', 'restaurant', 'breakfast', 'location', 'noise', 'maintenance' (Figure 43) and also has sliders for the reviewer score and the number of reviews the customer has left in the past ((Figure 44). The only filter I decided to keep is the time period the review was written, for convenience if the user wants to track changes in noise or service over the last week, for example.

There are two options for users of the application: negative and positive weights. As written in the introduction in Figure 43, negative weights prioritise negative sentiments of aspects, lower reviewer scores and less active travellers, while positive weights, conversely, prioritise positive aspects, higher reviewer scores and number of previous comments.

Figure 43: Interface of the adjustable re-ranking model (part 1)



	review	Weighted_Rank
0	They offered us an upgrade which I paid for as it was my 65th birthday and I thought	0
6,080	The hotel was situated a little too far out of town Lovely room taseful decor and artw	0
6,074	Buffet breakfast was a bit expensive Rooms were a bit tiny but manageable Convenie	0
6,075	Friendly receptionist Quite area	0
6,076	The lounge Area is a bit small side and therefore gets busy quickly Other than that th	0
6,077	Breakfast was decidedly mediocre given how excellent everything else was Certainly	0
6,078	Nothing to declare The hotel brings a high style of modernity and glamour The staff	0
6,079	Super efficient staff Convenient central location Quiet rooms	0
6,081	Although it was the weekend before Christmas it was still a lot of money for the 1 nig	0
6,072	People in room next door had to on until 4am one night so impossible to sleep The st	0

Figure 44: Interface of the adjustable re-ranking model (part 2)

Overall, Streamlit is a convenient software for data scientists without previous design and front-end expertise. It is also extremely easy to use. One of the options of the web application is local access, but in my case, where I want to make the models publicly available, this would only be on my laptop. The second option, which I chose, is to upload the Python file or simply write the code on Github and add the link directly to the Streamlit website. The advantage of this approach is that anyone can access my code on Github and adapt it to their own needs. It is also possible, for example, to add a link to another public dataset of hotel reviews.

5 Results

As I have considered a number of sentiment detection and prediction models in the methodology, I have also included the results of the existing experiments. However, in this chapter I would like to compare the algorithms and their performance to determine the most efficient ones that worked best with my domain and review dataset.

I want to focus on the prediction models because they provide a basis for an adequate comparison, namely evaluation metrics like accuracy, recall, precision and f1-score. In Table 13 you can see the best performances of the algorithms I experimented with in my work.

It is important to note that most of these models underwent considerable fine-tuning during the course of the study. For instance, first five models from the Table 13 Gaussian NB, Decision Tree, Random Forest, SVM and Bidirectional LSTM were used in combination with different labelling techniques. My dataset was not explicitly classified, so I had to label it based on the numerical score given to the hotel by the reviewer. This approach, although highly standardised, was quite subjective as many users, even if they described a bad experience in the review text, did not reflect this in their rating. So I used sentiment detection methods such as SpaCy, AFINN, TextBlob and Vader to make sure I was working with the sentiment of the text.

Model	Accuracy	Recall	Precision	F1-score
Gaussian NB	0.9191	0.8845	0.6702	0.7626
Decision Tree	0.9598	0.8654	0.8614	0.8634
Random Forest	0.9743	0.9082	0.9161	0.9122
Support Vector Machine	0.9275	0.5482	0.929	0.6896
Bidirectional LSTM	0.9674	0.9405	0.8175	0.8747
BERT	0.943	1	0.943	0.97
RoBERTa	0.945	0.945	0.915	0.925
Key Phrase and Entity Extractor	0.91	0.91	0.94	0.92
PyABSA	0.8715	0.9469	0.9467	0.9468

Table 13: Best performances of all the models considered in the research

In order to avoid overloading the final results, in Table 13 I have only included the best performances of the models: Gaussian NB, Decision Tree, Random Forest and SVM with SpaCy, and bidirectional LSTM with AFINN. As for the other algorithms (mostly neural networks), they worked directly on the text, its vectors, aspect sentiments, etc, so no prior overall sentiment labelling of the text was necessary. So I experimented with different settings and input parameters and limited the range of aspects to the list of the most popular ones.

PyABSA, last model from Table 13, even though being an algorithm for detecting the aspects from the text, served as good base for a simple manually designed prediction model which did not require training data but rather used pre-defined assumption for the final label anticipation.

Random Forest has the highest accuracy of all the models with 97%, followed by bidirectional LSTM with

also almost 97%. However, due to the imbalance in my dataset, I am more interested in a higher f1-score than in accuracy. Here the best result was achieved by BERT with 97%. Overall, more complicated models had a higher f1-score and proved to work better with text analysis. Random Forest was the only one of the simpler machine learning algorithms to perform above 90% f1-score, while others were in the range of 70-80%, despite high accuracy. The manually tuned PyABSA algorithm also gave reasonably satisfactory results, with the second highest f1-score but the lowest accuracy. The worst performing method was SVM. It had the lowest recall of less than 60% and an f1-score of 69%.

6 Conclusion

In this thesis, I investigated different sentiment detection tools and machine learning prediction algorithms by testing them on more than 500,000 reviews scraped from Booking.com. I also developed three options of open-source and user-friendly web applications that include different filters for reviews and reviewers, a self-developed prioritisation model with a focus on unsatisfied customers, and an adjustable re-ranking system that allows users to control the parameter weights themselves.

As a result, I came to the following conclusions. First, of all the sentiment extraction methods, SpaCy proved to be most compatible with supervised learning techniques, as most of the models were able to reach their peak performance with the review labelling provided by SpaCy. Secondly, neural networks resulted in higher recall, precision and corresponding f1-score than classical machine learning algorithms such as SVM, Gaussian NB or Decision Tree, proving their effectiveness in NLP-related tasks. However, Random Forest, which is not part of the transformers, outperformed other models in terms of accuracy and showed excellent results in other metrics, being in the same league as BERT and RoBERTa.

Apart from the managerial implications of these results for the community of data scientists and NLP experts, my prioritisation models are also useful from a business perspective. For example, since they are publicly available, hotel owners could use them to improve overall customer satisfaction, develop loyalty among existing customers and attract new ones, expand their market share, or simply use the algorithm to get regular reports on the quality of the service they provide. In addition to individual purposes, my app could also serve the needs of the entire industry. Organisations such as The Austrian Hotelier Association, which represents the interests of the entire hotel-related sector in Austria, could offer my development to their members in order to prepare them for current and future challenges and to promote the overall success of the industry. In addition, an adaptable model could be used by various types of Austrian hotels, such as spa or ski resorts, which are very popular in the region, taking into account their characteristics.

Despite the positive aspects of my thesis, there are still some limitations and possible improvements. For example, due to the limited capacity of the equipment, I was not able to consider some more sophisticated prediction models that might lead to more accurate and comprehensive results. Secondly, the dataset only includes English reviews, whereas travellers come from all over the world. Translation into English may not be as effective as looking at the original text using the multilingual models. It may also be interesting to investigate the personal data of the reviewer, such as age, gender, occupation, income, etc to see their influence on the prediction of the review class and to extend the prioritisation model with customer-related information, which would make it possible to determine the target customer's preferences and needs.

References

[1] spaCy – linguistic features. URL https://spacy.io/usage/linguistic-features. Accessed: April 15, 2023.

[2] Jaffar Abbas, Riaqa Mubeen, Paul Terhemba Iorember, Saqlain Raza, and Gulnara Mamirkulova. Exploring the impact of COVID-19 on tourism: Transformational potential and implications for a sustainable recovery of the travel and leisure industry. *Current Research in Behavioral Sciences*, 2:100033, 2021. URL https://www.sciencedirect.com/science/article/pii/S2666518221000206.

[3] Müjde Aksoy and Özer Yilmaz. Consumer complaints and complaint management in the tourism sector. In *Managing Risk and Decision Making in Times of Economic Distress, Part A*. Emerald Publishing Limited, 2022. doi: 10.1108/S1569-37592022000108A034.

[4] Mohit Kumar Barai. Sentiment analysis with TextBlob and Vader. *Analytics Vidhya*, October 2021. URL https://www.analyticsvidhya.com/blog/2021/10/sentiment-analysis-with-textblob-and-vader/. Published on October 20, 2021 and Last Modified on October 26th, 2021.

[5] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3(Jan):993–1022, 2003. URL https://www.jmlr.org/papers/volume3/blei03a/blei03a.pdf.

[6] Ali Boluki, Javad Pourmostafa Roshan Sharami, and Dimitar Shterionov. Evaluating the effectiveness of pre-trained language models in predicting the helpfulness of online product reviews, 2023. arXiv:2302.10199.

[7] Marius Borcan. Decision Tree Classifiers Explained. *Medium*, March 2020. URL https://medium.com/@borcandumitrumarius/decision-tree-classifiers-explained-e47a5b68477a.

[8] Margaret M Bradley and Peter J Lang. Affective Norms for English Words (ANEW): Instruction manual and affective ratings. Technical report, Technical Report C-1, the Center for Research in Psychophysiology, University of Florida, 1999.

[9] Siddhartha Brahma. Improved sentence modeling using suffix Bidirectional LSTM, 2018. arXiv:1805.07340.

[10] Jason Brownlee. Hyperparameter Optimization with Random Search and Grid Search. *Machine Learning Mastery*, September 2020. URL https://machinelearningmastery.com/

hyperparameter-optimization-with-random-search-and-grid-search/. Published in Python Machine Learning, Last Updated on September 19, 2020.

[11] Jason Brownlee. ROC curves and precision-recall curves for imbalanced classification. *Machine Learning Mastery*, January 2020. URL https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/. Published in Imbalanced Classification, Last Updated on September 16, 2020.

[12] Vitaly Bushaev. Adam — latest trends in deep learning optimization. *Towards Data Science*, October 2018. URL https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c.

[13] Bowen Cai. Customer selection model with grouping and hierarchical ranking analysis, 2017. arXiv:1711.05598.

[14] Mingshan Chang, Min Yang, Qingshan Jiang, and Ruifeng Xu. Reducing spurious correlations for aspect-based sentiment analysis with variational information bottleneck and contrastive learning, 2023. arXiv:2303.02846.

[15] Chat GPT. Prompt: Can you write about yourself shortly. how do you work and what are you capable of doing? URL https://chat.openai.com/. Accessed: June 01, 2023.

[16] Mukesh Chaudhary. Tf-idf vectorizer scikit-learn. *Medium*, April 2020. URL https://medium.com/@cmukesh8688/tf-idf-vectorizer-scikit-learn-dbc0244a911a.

[17] Tianqi Chen, Tong He, Michael Benesty, and Vadim Khotilovich. Package 'XGBoost'. *R version*, 90: 1–66, 2019. URL https://dl.acm.org/doi/10.1145/2939672.2939785.

[18] Zhenxiao Cheng, Jie Zhou, Wen Wu, Qin Chen, and Liang He. Tell model where to attend: Improving interpretability of aspect-based sentiment classification via small explanation annotations, 2023. arXiv:2302.10479.

[19] Yew Ken Chia, Hui Chen, Wei Han, Guizhen Chen, Sharifah Mahani Aljunied, Soujanya Poria, and Lidong Bing. Domain-expanded ASTE: Rethinking generalization in Aspect Sentiment Triplet Extraction, 2023. arXiv:2305.14434.

[20] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale, 2020. arXiv:1911.02116.

[21] Subhasis Dasgupta and Jaydip Sen. A framework of customer review analysis using the aspect-based opinion mining approach, 2022. arXiv:2212.10051.

[22] Yue Deng, Wenxuan Zhang, Sinno Jialin Pan, and Lidong Bing. Bidirectional generative framework for cross-domain aspect-based sentiment analysis, 2023. arXiv:2305.09509.

[23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding, 2019. arXiv:1810.04805.

[24] David P. Doane. Aesthetic frequency classification. *American Statistician*, 30:181–183, 1976. URL http://www.jstor.org/stable/2683757. Retrieved December 13, 2017.

[25] Dianne Dredge, Giang Thi Linh Phi, Renuka Mahadevan, Eóin Meehan, and Elena Popescu. Digitalisation in tourism: In-depth analysis of challenges and opportunities. 2019. URL https://ec.europa.eu/docsroom/documents/33163/attachments/1/translations/en/renditions/native.

[26] Kenneth Enevoldsen. Introduction to aSent: A Sentiment Analysis Tool. URL https://kennethenevoldsen.github.io/asent/introduction.html. Accessed: April 15, 2023.

[27] Sydney F. Getting to the point with topic modeling | Part 3 – interpreting the visualization. *Alteryx Community*, August 2020. URL https://community.alteryx.com/t5/Data-Science/Getting-to-the-Point-with-Topic-Modeling-Part-3-Interpreting-the/ba-p/614992.

[28] Mohammad Forouhesh, Arash Mansouri, and Hossein Fani. Latent aspect detection from online unsolicited customer reviews, 2022. arXiv:2204.06964.

[29] Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics Data Analysis*, 38(4): 367–378, 2002. ISSN 0167–9473. doi: https://doi.org/10.1016/S0167-9473(01)00065-2. URL https://www.sciencedirect.com/science/article/pii/S0167947301000652.

[30] Bharat Gaind, Varun Syal, and Sneha Padgalwar. Emotion detection and analysis on social media, 2019. arXiv:1901.08458.

[31] Chufan Gao, Mononito Goswami, Jieshi Chen, and Artur Dubrawski. Classifying unstructured clinical notes via automatic weak supervision, 2022. arXiv:2206.12088.

[32] GeeksforGeeks. Overview of RoBERTa Model. *GeeksforGeeks*. URL https://www.geeksforgeeks.org/overview-of-roberta-model/. Accessed: May 2, 2023.

[33] Maarten Grootendorst. BERTopic: Neural topic modeling with a class-based Tf-idf procedure, 2022. arXiv:2203.05794.

[34] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern recognition*, 77:354–377, 2018. URL https://arxiv.org/abs/1512.07108.

[35] John A Hartigan and Manchek A Wong. Algorithm AS 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[36] John R Hauser and Don Clausing. The house of quality. *Harvard Business Review Vol. 66*, 1988. URL https://hbr.org/1988/05/the-house-of-quality.

[37] Yu-Chih Huang, Lan Lan Chang, Chia-Pin Yu, and Joseph Chen. Examining an extended technology acceptance model with experience construct on hotel consumers' adoption of mobile applications. *Journal of Hospitality Marketing & Management*, 28(8):957–980, 2019.

[38] Hugging Face. Chapter 7: Fine-tuning a pre-trained model. URL https://huggingface.co/learn/nlp-course/chapter7/3?fw=tf. Accessed: April 29, 2023.

[39] IBM. Unsupervised learning. URL https://www.ibm.com/topics/unsupervised-learning. Accessed: April 25, 2023.

[40] Zakaria Jaadi. A step-by-step explanation of Principal Component Analysis (PCA). *Built In*, March 2023. URL https://builtin.com/data-science/step-step-explanation-principal-component-analysis. Updated by Brennan Whitfield on March 29, 2023. Reviewed by Sadrach Pierre.

[41] Pratyaksh Jain. Basics of CountVectorizer. *Towards Data Science*, May 2021. URL https://towardsdatascience.com/basics-of-countvectorizer-e26677900f9c. Published online on May 24, 2021.

[42] Anoop Kadan, Deepak P., Manjary P. Gangan, Savitha Sam Abraham, and Lajish V. L. REDAffectiveLM: Leveraging affect enriched embedding and transformer-based neural language model for readers' emotion detection, 2023. arXiv:2301.08995.

[43] Ria Kulshrestha. Transformers. *Towards Data Science*, June 2020. URL https://towardsdatascience.com/transformers-89034557de14.

[44] Kenneth Leung. Micro, macro weighted averages of f1-score, clearly explained. *Towards Data Science*, January 2022. URL https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f. Published in Towards Data Science.

[45] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach, 2019. arXiv:1907.11692.

[46] Khyati Mahendru. How to determine the optimal K for k-means? *Analytics Vidhya*, June 2019. URL https://medium.com/analytics-vidhya/how-to-determine-the-optimal-k-for-k-means-708505d204eb. Published in Analytics Vidhya.

[47] Juveriya Mahreen. Sentiment analysis using Vader. *Analytics Vidhya*, October 2022. URL https://www.analyticsvidhya.com/blog/2022/10/sentiment-analysis-using-vader/. Published on October 2, 2022 and Last Modified on October 12th, 2022.

[48] MathWorks. Word2Vec – mathworks. URL https://www.mathworks.com/discovery/word2vec.html. Accessed: April 1, 2023.

[49] Leland McInnes, John Healy, and James Melville. UMAP: Uniform Manifold Approximation and Projection for dimension reduction, 2018. arXiv:1802.03426.

[50] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013. arXiv:1301.3781.

[51] Rory Mitchell and Eibe Frank. Accelerating the XGBoost algorithm using GPU computing. *PeerJ Computer Science*, 3:e127, 2017. URL https://peerj.com/articles/cs-127.pdf.

[52] Chanin Nantasenamat, Avratanu Biswas, JM Nápoles-Duarte, Mitchell I Parker, and Roland L Dunbrack Jr. Building bioinformatics web applications with Streamlit. In *Cheminformatics, QSAR and Machine Learning Applications for Novel Drug Development*, pages 679–699. Elsevier, 2023.

[53] Natural Language Toolkit. Collocations – NLTK 3.6.6 documentation. URL https://www.nltk.org/howto/collocations.html. Accessed: March 12, 2023.

[54] Natural Language Toolkit. NLTK 3.6.6 documentation, 2009. URL https://www.nltk.org/. Accessed: March 12, 2023.

[55] Mir Tafseer Nayeem and Davood Rafiei. On the role of reviewer expertise in temporal review helpfulness prediction, 2023. arXiv:2303.00923.

[56] Finn Årup Nielsen. A new ANEW: Evaluation of a word list for sentiment analysis in microblogs, 2011. arXiv:1103.2903.

[57] Olumide Ebenezer Ojo, Hoang Thang Ta, Alexander Gelbukh, Hiram Calvo, Olaronke Oluwayemisi Adebanji, and Grigori Sidorov. Transformer-based approaches to sentiment detection, 2023. arXiv:2303.07292.

[58] Shilpa Patil and V Lokesha. Live Twitter sentiment analysis using Streamlit framework. 2022. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4119949.

[59] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Andreas Müller, Joel Nothman, Gilles Louppe, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. arXiv:1201.0490.

[60] Letizia Lo Presti, Giulio Maggiore, and Vittoria Marino. Digital communication tools and tourism engagement: Instant CRM strategies through instant messaging apps. In *Handbook on Tourism and Social Media*, pages 242–255. Edward Elgar Publishing, 2022.

[61] Md Mahbubur Rahman and Shaila Shova. Emotion detection from social media posts, 2023. arXiv:2302.05610.

[62] V V Ramalingam, A Pandian, Abhijeet Jaiswal, and Nikhar Bhatia. Emotion detection from text. *Journal of Physics: Conference Series*, 1000(1):012027, apr 2018. doi: 10.1088/1742-6596/1000/1/012027. URL https://dx.doi.org/10.1088/1742-6596/1000/1/012027.

[63] Yassine Rodani. Movie sentiment analysis: A multinomial Naıve Bayes-based approach for assessing user and critic opinions. URL https://yassine-rodani.info/assets/pdf/Movie_Sentiment_Analysis_report.pdf.

[64] Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. Automatic keyword extraction from individual documents. *Text Mining: Applications and Theory*, pages 1–20, 2010.

[65] Sherina Sally. Sentiment analysis on Youtube smart phone unboxing video reviews in Sri Lanka. *International Journal of Research – Granthaalayah*, 10(11), nov 2022. doi: 10.29121/granthaalayah.v10.i11.2022.4884. URL https://doi.org/10.29121%2Fgranthaalayah.v10.i11.2022.4884.

[66] Lonashree Sanasam, Bibhutibhusan Pradhan, and Sasmita Mohanty. Empirical evidence from hotel industry on the dimension of customer relationship management & their influence on organisation performance. *Geo Journal of Tourism and Geosites*, 41(2):408–414, 2022. URL https://gtg.webhost.uoradea.ro/PDF/GTG-2-2022/gtg.41210-844.pdf.

[67] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter, 2020. arXiv:1910.01108.

[68] Kevin Scaria, Himanshu Gupta, Siddharth Goyal, Saurabh Arjun Sawant, Swaroop Mishra, and Chitta Baral. InstructABSA: Instruction Learning for Aspect Based Sentiment Analysis, 2023. arXiv:2302.08624.

[69] Parthvi Shah. Sentiment analysis using TextBlob and its working! *Towards Data Science*, June 2020. URL https://towardsdatascience.com/my-absolute-go-to-for-sentiment-analysis-textblob-3ac3a11d524.

[70] Thanveer Shaik, Xiaohui Tao, Christopher Dann, Haoran Xie, Yan Li, and Linda Galligan. Sentiment analysis and opinion mining on educational data: A survey. *Natural Language Processing Journal*, 2: 100003, mar 2023. doi: 10.1016/j.nlp.2022.100003. URL https://doi.org/10.1016%2Fj.nlp.2022.100003.

[71] Pulkit Sharma. The ultimate guide to k-means clustering: Definition, methods and applications. *Analytics Vidhya*, August 2019. URL https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/. Published on August 19, 2019 and Last Modified on May 19th, 2023.

[72] Jingli Shi, Weihua Li, Quan Bai, Yi Yang, and Jianhua Jiang. Soft prompt guided joint learning for cross-domain sentiment analysis, 2023. arXiv:2303.00815.

[73] Falak Singh. Sentiment analysis made easy using Vader. *Analytics India Magazine*, December 2020. URL https://analyticsindiamag.com/sentiment-analysis-made-easy-using-vader/. Published on December 8, 2020 in Mystery Vault.

[74] Uglješa Stankov, Viachaslau Filimonau, and Iva Slivar. Calm ICT design in hotels: A critical review of applications and implications. *International Journal of Hospitality Management*, 82:298–307, 2019.

[75] Doug Steen. Understanding the ROC Curve and AUC. *Towards Data Science*, September 2020. URL https://towardsdatascience.com/understanding-the-roc-curve-and-auc-dd4f9a192ecb.

[76] Ryan A Stevenson, Joseph A Mikels, and Thomas W James. Characterization of the affective norms for English words by discrete emotional categories. *Behavior research methods*, 39(4):1020–1024, 2007. URL https://link.springer.com/article/10.3758/BF03192999.

[77] Davide Stirparo, Beatrice Penna, Mohammad Kazemi, and Ariona Shashaj. Mining tourism experience on Twitter: A case study, 2022. arXiv:2207.00816.

[78] HP Suresha and Krishna Kumar Tiwari. Topic modeling and sentiment analysis of electric vehicles of Twitter data. *Asian Journal of Research in Computer Science*, 12(2):13–29, 2021.

[79] My Data Talk. Streamlit hands-on: From zero to your first awesome web app. *Towards Data Science*, November 2021. URL https://towardsdatascience.com/streamlit-hands-on-from-zero-to-your-first-awesome-web-app-2c28f9f4e214. Published in Towards Data Science.

[80] Adrien Treuille. Turn Python scripts into beautiful ML tools. *Towards Data Science*, October 2019. URL https://towardsdatascience.com/coding-ml-tools-like-you-code-ml-models-ddba3357eace. Published in Towards Data Science.

[81] Sik-Ho Tsang. Review — RoBERTa: A Robustly Optimized BERT Pretraining Approach. *Medium*, February 2022. URL https://sh-tsang.medium.com/review-roberta-a-robustly-optimized-bert-pretraining-approach-d1be4014e5ce.

[82] Omkarprasad S Vaidya and Sushil Kumar. Analytic hierarchy process: An overview of applications. *European Journal of operational research*, 169(1):1–29, 2006.

[83] Yugesh Verma. How to implement a Doc2Vec model using Gensim. *Analytics India Magazine*, February 2022. URL https://analyticsindiamag.com/how-to-implement-a-doc2vec-model-using-gensim/. Published on February 20, 2022 in Mystery Vault.

[84] Jesse Vig. A multiscale visualization of attention in the transformer model. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 37–42, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-3007. URL https://www.aclweb.org/anthology/P19-3007.

[85] H. Wang, M. Xie, and T. N. Goh. A comparative study of the prioritization matrix method and the analytic hierarchy process technique in quality function deployment. *Total Quality Management*, 9(6):421–430, 1998. doi: 10.1080/0954412988361. URL https://doi.org/10.1080/0954412988361.

[86] Zengzhi Wang, Qiming Xie, Zixiang Ding, Yi Feng, and Rui Xia. Is ChatGPT a good sentiment analyzer? A preliminary study, 2023. arXiv:2304.04339.

[87] Theresa Wilson, Paul Hoffmann, Swapna Somasundaran, Jason Kessler, Janyce Wiebe, Yejin Choi, Claire Cardie, Ellen Riloff, and Siddharth Patwardhan. OpinionFinder: A system for subjectivity analysis. In *Proceedings of HLT/EMNLP 2005 Interactive Demonstrations*, pages 34–35, 2005. URL https://aclanthology.org/H05-2018.pdf.

[88] World Tourism Organization (UNWTO). Tourism and COVID-19: Unprecedented economic impacts. URL https://www.unwto.org/tourism-and-covid-19-unprecedented-economic-impacts. Accessed: May 28, 2023.

[89] Ting Xu, Huiyun Yang, Zhen Wu, Jiaze Chen, Fei Zhao, and Xinyu Dai. Measuring your ASTE models in the wild: A diversified multi-domain dataset for Aspect Sentiment Triplet Extraction, 2023. arXiv:2305.17448.

[90] Heng Yang and Ke Li. PyABSA: Open framework for Aspect-based Sentiment Analysis, 2022. arXiv:2208.01368.

[91] Wei Yang, Luchen Tan, Chunwei Lu, Anqi Cui, Han Li, Xi Chen, Kun Xiong, Muzi Wang, Ming Li, Jian Pei, and Jimmy Lin. Detecting customer complaint escalation with recurrent neural networks and manually-engineered features. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Industry Papers)*, pages 56–63, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-2008. URL https://aclanthology.org/N19-2008.

[92] Yilin Yang, Tomas Fieg, and Marina Sokolova. Sentiment analysis of Covid-related Reddits, 2022. arXiv:2205.06863.

[93] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–

1489, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1174. URL https://aclanthology.org/N16-1174.

[94] Wenjie Yin, Rabab Alkhalifa, and Arkaitz Zubiaga. The emojification of sentiment on social media: Collection and analysis of a longitudinal Twitter sentiment dataset, 2023. arXiv:2108.13898.

[95] Xing Yin, Changhui Liu, and Xiaodong Fang. Sentiment analysis based on BiGRU information enhancement. In *Journal of Physics: Conference Series*, volume 1748, page 032054. IOP Publishing, 2021. URL https://iopscience.iop.org/article/10.1088/1742-6596/1748/3/032054/pdf.

[96] Chengze Yu, Taiqiang Wu, Jiayi Li, Xingyu Bai, and Yujiu Yang. SynGen: A Syntactic Plug-and-play module for Generative Aspect-based Sentiment Analysis, 2023. arXiv:2302.13032.

[97] Enes Zvornicanin. Differences between Bidirectional and Unidirectional LSTM. *Baeldung*, November 2022. URL https://www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm. Last updated on November 6, 2022.