

Conversational Wine Recommender System

Thursday 18th June, 2020 - 15:12

Flavio De Jesus Matias
University of Luxembourg
Email: *flavio.dejesus.001@student.uni.lu*

Sviatlana Höhn
University of Luxembourg
Email: *sviatlana.hoehn@uni.lu*

Abstract—This document presents the bachelor semester project of Flavio De Jesus Matias under the tutoring of Sviatlana Höhn. The project consists of a conversational wine recommender system, which will summarize data about wine reviews using text summarization techniques and provide the user with a chatbot to which he may ask questions about an arbitrary wine. The chatbot will then use the preprocessed data to output information about the desired wine sort.

1. Introduction

Chatbots are a very important tool for most online businesses today. Businesses rely on their virtual assistants to handle basic user requests or questions. The application of chatbots in such businesses can free up employees and allow them to focus on more complex tasks that are not achievable by artificial intelligence. The goal of chatbots is to perform the repetitive work that can appear, for example, guide a customer to products they are searching for or to give them advice about which product to buy.

According to multiple studies, most users have also already been in contact with a chatbot and liked the experience. People that have not done it yet are also willing to try it and would not feel uncomfortable doing so [9].

With a rapidly growing amount of textual information available nowadays, the necessity for an effective and automatic summarization system has increased. Writing summaries by hand is an extremely time-consuming and exhaustive process and therefore it is important to have well-performing algorithms to analyse and process lots of textual data into small readable summaries. It has become of great importance to be able to provide a good mechanism to process text effectively and provide reliable summaries.

In general, chatbots only present a small amount of textual data to the user to avoid overloading them. Providing customers with short messages is a necessity to keep them engaged with the chatbot. This is why summarization techniques are so important in the chatbot industry, as people lose interest in interacting with a virtual assistant when they are given too much text to read. Most people want quick, short and precise answers to their questions which requires advance text summarizers to keep up with the customer's needs.

Summaries present the most important information from the original text in a short form. They are made up of one or multiple text sources and their main purpose is to reduce reading time [2].

2. Project description

This section introduces the domain and objectives of the project.

2.1. Domain

The domain of this project is natural language processing (NLP). Here we focus on 2 separate fields which are natural language generation (NLG) and natural language understanding (NLU). The 2 deliverables of the project focus each on one of these fields.

The first part is about natural language generation. This is covered by the algorithm which focusses on the use of summarization techniques to summarize wine reviews. These wine reviews come from the initial data which is made of several thousand lines of information about wine sorts. The algorithm will preprocess this data for later usage of a simple chatbot.

The second part of the project is about natural language understanding. This part is covered by the conversational wine recommender system, which is a chatbot. This chatbot was created using an open-source conversational AI framework which provides the chatbot with machine learning algorithms in order to improve the quality of the product. Here the output data of the algorithm will be the starting data, which is a database consisting of wine names and summaries. This way, the chatbot is able to retrieve the summaries out of the database and display them to the user on demand.

2.2. Objectives

The main objective of this project is the understanding and use of summarization techniques. We need to create meaningful extractive multi-document summaries. This task will be accomplished by an algorithm which has to preprocess the initial dataset and generate good quality summaries of all the available reviews. This deliverable will be

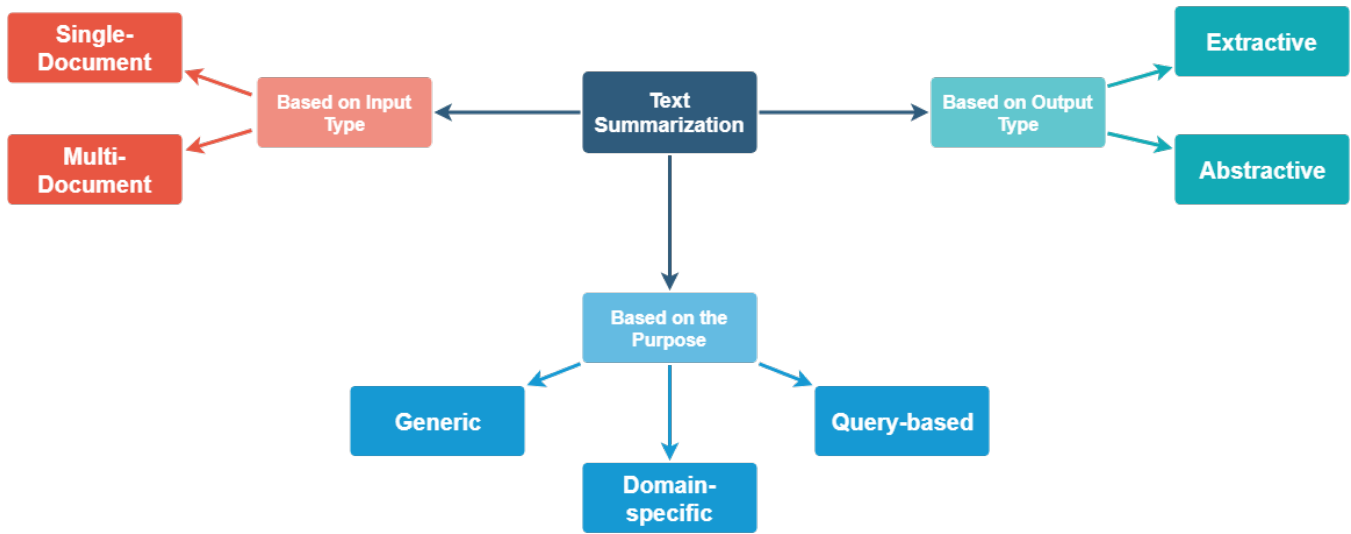


Figure 1: Text summarization methods [4]

described in section 4. The generated summaries will be stored in a separate dataset and later used for the creation of a simple chatbot which can react to basic, uncomplicated questions about arbitrary wine sorts. This deliverable will be described in section 5.

The first part is to create this algorithm and only use it once. Creating all the summaries is a very time-consuming task and it wouldn't be optimal having to run the summarization process for a specific wine sort on each demand. To prevent the algorithm from having to generate each summary at the time of demand, it has to preprocess the data and generate and store all the summaries in beforehand. Having the summaries ready for retrieval whenever the user asks a question contributes to better performance since it allows the chatbot to react faster to the questions.

The second part is the creation of the chatbot that uses the generated summaries by the algorithm. The chatbot should be able to be extended for utilization among larger wine datasets. In other words, the trained NLU model must be able to identify a large number of wine sorts including the ones not present in the training set. When the identified wine sort is found in the database provided by the algorithm, it must retrieve the corresponding summary and output it on the screen for the user to read.

3. Background

This project consists of two separate parts those being the algorithm to create the wine summaries dataset as well as the conversational wine recommender system. Both parts of the project use Python as programming language, so it was mandatory having knowledge in this language. Although the student already had some knowledge in Python, a lot still had to be researched. Likewise, knowledge in different summarization techniques was necessary, which the student did not have at the beginning of the project. This was, however, part of the learning goal of this project and therefore very

basic research was conducted on this topic at the beginning and the rest was learned while doing the project.

3.1. Scientific background

The scientific part of this project was the understanding and use of summarization techniques.

3.1.1. Choice of Summarization Technique. Unlike abstractive summarization, extractive summarization does not form its own sentences in order to provide a more unique and rational summary. Of course, this technique seems more interesting and tempting to use, but in practice, it is much harder to apply. Since the extractive technique is simpler and suits better the purpose of the project, it was chosen. Since we are working with a lot of reviews, it means that the procedure is multi-document. We are therefore working with "**Extractive multi-document text summarization**" in this project. The different summarization methods can be seen in figure 1.

3.1.2. Extractive Summarization. The extractive technique is the simplest and most used approach to generate summaries. The process happens in **3** different stages:

- 1) group the textual data to be summarized
- 2) score each individual sentence using an arbitrary process
- 3) use the n best scoring sentences to create an n -long summary

The used process to score each individual sentence is the most complicated part of the process. The word frequency of each sentence is computed using the Tf-idf model [3]. This approach will be explained in more detail later. For each individual document (single or multi-document), the word weights are calculated. The score of a sentence is the sum of the Tf-idf word frequencies. The process does

not generate new sentences but instead uses already existing sentences found in the documents. The exact summarization procedure used in this project will be explained in the algorithm section.

3.2. Technical background

The technical part of this project was the use and combination of Python and a conversational AI framework to create the conversational recommender system.

3.2.1. Python. Python is a general-purpose, object-oriented programming language. Apart from web and software development, this programming language can be used for a lot of different tasks such as GUI desktop apps or even simple system scripts. Since Python requires a lot less time, effort as well as lines of code to perform the same operations as other programming languages, it has grown to be a very popular and loved language among developers. It is also a beginner-friendly and very fast language on top of that.

3.2.2. Rasa. Rasa is an open-source conversational AI framework. With its machine learning power, it helps to build AI assistants for arbitrary tasks with ease. Rasa NLU is used to create chatbots and in order to use it correctly, a Natural Language Understanding (NLU) model has to be trained. This trained data must define the intents and entities according to the chatbot's needs. Rasa NLU works with intent classification and entity extraction, which means to specify the intention of different sentences and classify them into different categories and also to specify the sentence part (word, entity) that has to be extracted from the user's questions.

4. Wine reviews summarization

This section contains the scientific deliverable of the project which is the algorithm to create a dataset of summaries from wine reviews.

4.1. Functional requirements

The functional requirements are the tasks that the algorithm should accomplish on doing, i.e. the behaviour of the product.

4.1.1. Generate a summary out of a set of reviews. In order to function correctly, the algorithm needs to be able to process an arbitrarily long input of reviews and transform it into a summary. Each summary must be generated according to the standards needed for later usage of the chatbot.

4.1.2. Create final dataset. The goal of the algorithm is to create a dataset consisting of summaries for each wine sort. This means that it has to be able to generate a dataset where each individual summary will be stored. This dataset must be kept simple and minimal to also create an easy access to the summaries for the chatbot later on.

4.2. Non-functional requirements

The non-functional requirements specify criteria that judge the operation of a system, rather than specific behaviours. They judge the software system based on the quality of various attributes, for instance, usability and reliability.

4.2.1. Finding an adequate summary length. To ameliorate the chatbot usability, the summaries must conform to certain criteria. Message length is a remarkable contributor to good user-friendly chatbots. To keep the user engaged with the chatbot, the message length must be approximately **140** characters. This is known as the "Twitter rule" [10] [11]. Besides the message length, there is also the number of sentences for each summary which will either be **3** or **5** sentences, depending on the suitability. The decision will be made upon the evaluation, which will give a clear view of what is best for the project.

4.3. Design

For a chatbot that recommends wine to a user, it is important to have all information ready for retrieval. The bot would have a set of templates that it will fill with the extracted information and review summaries. To make this service work in practice, the information has to be provided to the bot beforehand. Data cleaning and preprocessing takes a lot of time and is crucial for further processing steps in the NLP pipeline. This is why, to begin with, the initial data had to be well understood to be able to clean and manipulate the available amount of information.

Figure 6 shows the number of reviews per wine sort. This helps to understand the data to be worked with by

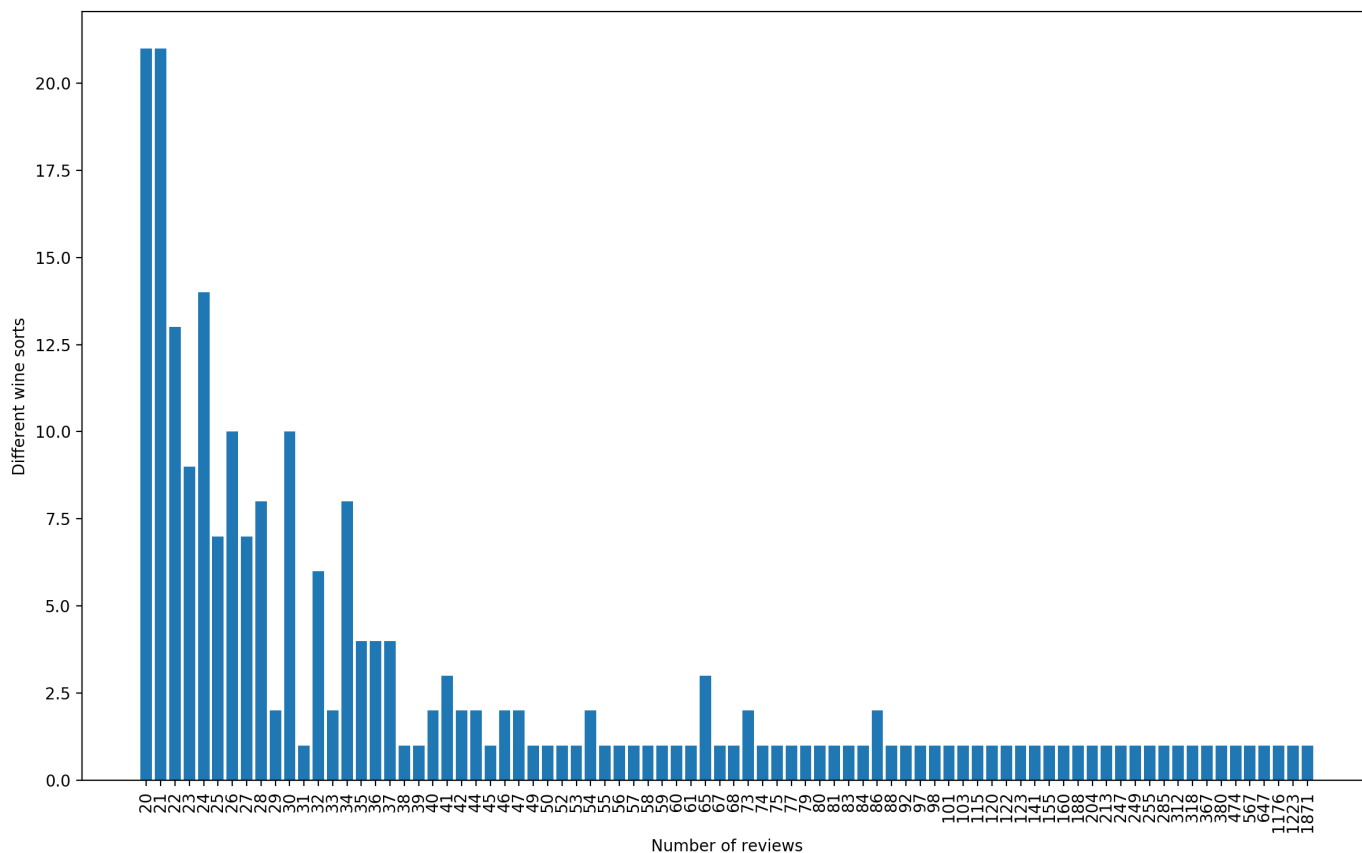


Figure 2: Number of reviews per different wine sorts

visualizing how much wine reviews each wine sort has. Using this graph one was able to see that there was an enormous amount of wine sorts that only have less than 20 reviews (this part was cut out of the figure for demonstration purposes). This led to the decision of considering only wine sorts having more or equal to 20 reviews for the purpose of this project.

The summarization part of the reviews needed to be in a precise way; multi-document because we are working with a lot of reviews and extractive since we want to reuse the sentences of the reviews. By doing some research, a similar summarization tool was found [12]. This tool was originally developed to summarize multiple news articles into one single article. Since there are a lot of similarities between the news article summarizer and this project, the code could be reused. The whole summarization process is pretty simple. Every document, which in this case are the multiple reviews, gets split into different sentences. To avoid poorly descriptive sentences, each sentence containing less than 10 words is filtered out and ignored for the final summary. Sentences with less than 10 words were considered poorly descriptive because most of the wine sorts have small review sentences indicating the year when it should be drunk. An example would be "Drink in 2020." which is an unnecessary sentence to have in a summary. As this occurs too often, even with other longer examples, the

sentence word minimum was set to 10. All the satisfactory sentences are then ranked using a more complex algorithm and inserted into a big list containing all the sentences from all the reviews including its score.

After all the reviews are processed, we end up with an organized list containing all the sentences from all the reviews. In order to create the extractive summary, we pick n sentences from the list by always taking the one with the highest score. The order of the sentences is not kept compared to the original documents, as we simply pick the highest performing sentences and as soon as the sentences are scored and stored in a list, they have no relation to the original document anymore and they are not traced backwards. This was a good decision because we no do care about the entire reviews, but only about the best sentences for all the combined reviews. By doing so, we end up with the n best sentences from all the reviews, which forms the desired summary.

4.4. Production

The entire algorithm consists of 4 files. 2 of these files are Python files and the other 2 are CSV files. Each of them will be explained in more details below.

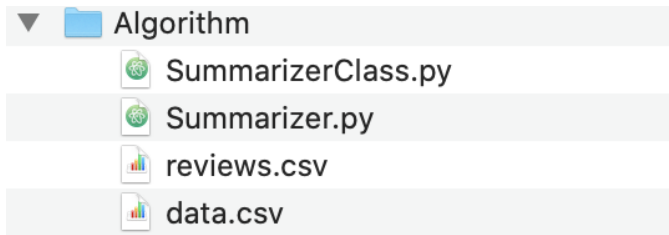


Figure 3: File structure of the algorithm to generate summaries from wine reviews

4.4.1. data.csv. The file data.csv contains the initial dataset of the algorithm. This file was provided by Kaggle¹ which is a data science community that provides free data to analyze and "achieve your data science goals". With approximately 130 thousand rows of different wine reviews, this gives the project a solid base to work with.

4.4.2. SummarizerClass.py. This file does all the summarization work. The code was intentionally designed to produce summaries about news articles, but since summarizing news articles and summarizing wine reviews is a very similar procedure, the most part of the code could be reused and adapted to the needs of the project. All the reviews that are passed to this class are split into sentences. Now we have a list containing all the sentences from all the reviews, we just need to score them and then select the n best performing sentences to create the summary. But, how do we score the sentences? There are **two** crucial aspects when it comes to scoring the sentences: the sentence length and the Tf-idf score, where the Tf-idf score counts **twice** as much as the sentence length.

• Sentence length

Since we are expecting sentences that describe well the wine sort and are informative while also being relatively small, the ideal sentence length was set to **15** words.

$$\frac{|\text{Ideal sentence length} - \text{word count in sentence } s|}{\text{Ideal sentence length}} \quad (1)$$

Using this formula, it gives a score on how close the sentence's length is to the ideal length. This means that the closer a sentence is to having 15 words, the higher its score is.

• Tf-idf

Tf-idf stands for term frequency (tf) - inverse document frequency (idf), and the Tf-idf weight is often used in information retrieval. This weight reflects how important a specific word is to a given document. [5]

- The **term frequency** calculates the number of times a word appears in a document, divided by the total number of words in that document.

$$TF(i, j) = \frac{\text{Term } i \text{ frequency in document } j}{\text{Total words in document } j} \quad (2)$$

- The **inverse document frequency** is computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

$$IDF(i) = \log_2 \left(\frac{\text{Total documents}}{\text{Documents with term } i} \right) \quad (3)$$

The Tf-idf score for term i in document j is:

$$TFIDF(i, j) = TF(i, j) * IDF(i) \quad (4)$$

Now that Tf-idf score for each word can be calculated, the Tf-idf score for each sentence is simply the sum of the scores for each word in that particular sentence. The Tf-idf score for each sentence is then multiplied by 2 since it counts twice as much as the sentence length.

To generate the final summary, all reviews must first be split into sentences. During this process, each sentence of fewer than 10 words is ignored and left out of the final summary. After splitting all the reviews into sentences, these must be scored using the above-mentioned procedure. All the sentences from all the reviews are then finally put into a list with the sum of its sentence length and Tf-idf score. The final summary now consists of the n best-scored sentences in this list.

4.4.3. Summarizer.py. This file acts as the bridge between the old dataset and the new dataset. The first task is to load all the needed libraries and open the initial dataset data.csv. The new dataset file reviews.csv is also opened in write mode. This has the advantage of also deleting all the old content which might still be in the file. When opening the initial dataset, we don't need all the provided information. Therefore, we also specify the desired columns to be loaded, this being the country, designation and the description of all the different wines. We can then drop each duplicate of these 3 columns since it must logically be the same wine sort and we don't want to create a summary for a wine sort twice.

After removing all the duplicates we are left with approximately 38 thousand different wine sorts, the majority of which only have very few reviews. Since it does not make a lot of sense summarizing something that is already small, only wine sorts with more or equal to 20 reviews were therefore considered. Since the focus of the project is on summarization techniques, it wouldn't make a lot of sense to include those wine sorts too.

Since the SummarizerClass only accepts a list of reviews for each wine sort, there had to be a way to reformat the data before passing it to this class. To do this, an empty

1. <https://www.kaggle.com>

dictionary was created. Then, we iterated over every single review and added the wine sort to the dictionary as key if it wasn't already present. As the value of this key is a list which stores the reviews. On each iteration, the current review was thus added to this list which at the end created a key-value pair of wine name - list of reviews.

After creating the dictionary with the wine sorts and reviews, the dictionary is ordered by the length of the value. Since the value is the list with the reviews, by getting the length of the list, it returns the number of reviews that a particular wine has. Using this length, the list is ordered in ascending order, which will then contribute to a more organized final dataset.

Finally, we iterate over the dictionary and initialise the summarizer class with the equivalent reviews. After initializing the class, we can then generate the summary for each wine sort and we write a line to, the at the start opened file, which contains: the wine name, the number of reviews and the generated final summary. This leaves us with the final dataset which, in this case, contains exactly **227** different wine sorts.

4.4.4. reviews.csv. The file reviews.csv contains the final dataset created by the summarizer algorithm. The structure of the file is kept very simple, it only has **3 different columns**: wine name, number of reviews and the summary itself.

4.5. Assessment

In order to evaluate the quality of the generated summaries, the ROUGE evaluation metric [13] was used. The ROUGE evaluation is a set of metrics for evaluating machine-generated summaries. The results are computed by comparing the produced summary with a set of reference documents, which is generally human-produced. In this case, it is the concatenation of all the reviews for each wine sort.

To do the evaluation, all the wine sorts were divided into categories consisting of its reviews range. Now that the wine sorts are all divided into 5 categories, the evaluation process begins. For each of the categories, 2 randomly selected wine sorts are picked. This leaves us with 10 different wine sorts to be evaluated and perform the ROUGE metric on each of the generated summaries. The results are document in the table 1 below.

Wine sort	Reviews range	Rouge-1	Rouge-2	Rouge-L
Keefer Ranch	20-200	0.140585	0.135792	0.204355
Extra Brut	20-200	0.113799	0.110019	0.198224
Barrel Sample	200-300	0.021862	0.021095	0.124548
Old Vine	200-300	0.028717	0.027975	0.078996
Dry	300-400	0.016957	0.016542	0.069836
Crianza	300-400	0.015293	0.014859	0.063811
Brut	400-1000	0.011967	0.011744	0.051851
Riserva	400-1000	0.008396	0.008173	0.060836
Reserve	1000-2000	0.003765	0.003713	0.024489
Estate	1000-2000	0.005340	0.005300	0.030703

Table 1: Evaluation results for 10 random wine sorts

ROUGE-1 refers to the overlap of **unigrams** between the machine-generated summary and reference documents, while ROUGE-2 refers to the overlap of **bigrams**. ROUGE-L measures the longest matching sequence of words using LCS (longest common subsequence) [6]. The ROUGE-L score bases on the fact that the longer the LCS of two summary sentences is, the more similar the two summaries are. As we can observe from the results by averaging them out, the more reviews a wine sort has, the less the summary is similar to the initial reviews dataset, which is obvious.

Despite it being hard to judge the quality of the generated summaries based on the presented results, we can still notice good numbers. We can observe that the ROUGE-1 score is generally higher than the ROUGE-2 score, ranging from 1,5% to 14%. The ROUGE-2 score ranges from 0,4% to 13,5%, which represents almost the same difference. Looking at the numbers, we can identify that by averaging the scores, the bigger the reviews range is, the less similar is the summary to the initial reviews. Bigger reviews ranges make it difficult to achieve good scores, but if we consider the fact that we are comparing a small summary to a relatively large reviews dataset, the scores show themselves in a good light.

Another key factor that plays along in the evaluation of the summaries is the length of the summary. It was of great importance to decide whether the summaries would be 3 or 5 sentences long, which was not an obvious decision. To facilitate the choice, two separate datasets with 3 and 5 sentences long summaries were created. Plotting both of the datasets as the length of summaries in characters gives the charts in figure 5 and figure 4.

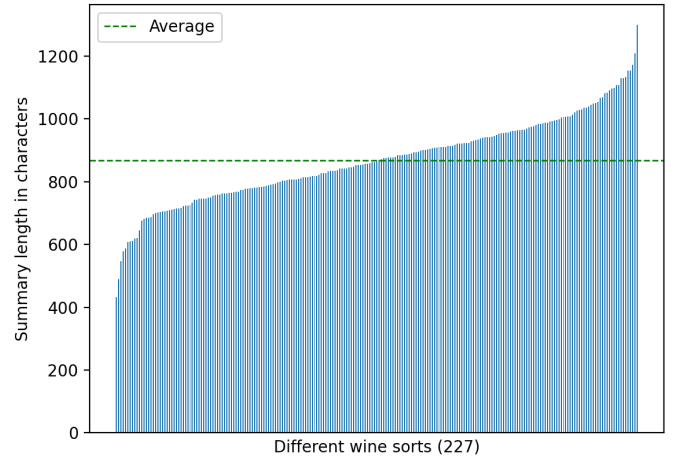


Figure 4: Summary length in characters for 5 sentences

The average length of the summaries for 5 sentences is **867,67** characters long. Knowing that 140 is the ideal/recommended message length in characters for chatbots, this number is too high. If we consider each sentence as an individual message, we end up with $867,67/5 = 173,4$ characters per message, which is a pretty good score since it is not very far away from the 140 characters guideline.

Nonetheless, it is a relatively large number of characters overall, which might overload the user.

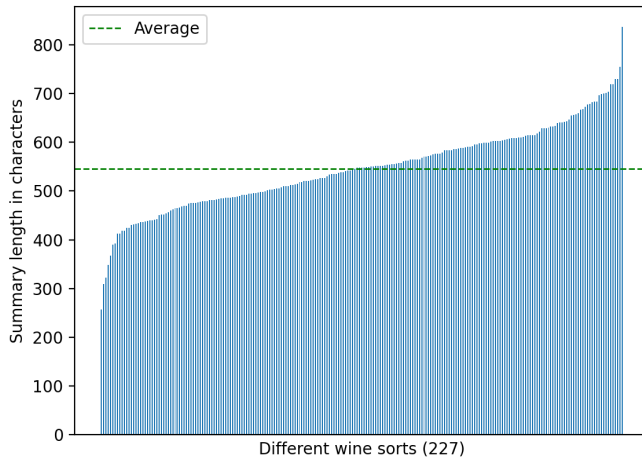


Figure 5: Summary length in characters for 3 sentences

The average length of the summaries for 3 sentences is **545,53** characters long. If we perform the same reasoning on this chart as with the other one and consider each sentence as an individual message, we end up with $545,53/3 = 181,8$. This number is slightly higher than the result that we obtain for the 5-sentence summaries, but even so, the overall character length is less.

Despite the choice, considering each sentence as an individual message lowers average characters, which is reasonably close to the 140 characters mark. Cutting down the average symbols in the summaries would lead to a quality loss, which is why the higher average is appropriate. However, to avoid overloading the user with too much content, 3 sentences/messages were the better choice.

Thus, the final dataset was successfully created containing summaries for all the wine sorts conforming to the requirements.

5. Conversational wine recommender system

This section contains the technical deliverable of the project which is the conversational wine recommender system. Here will be explained what was required to create the simple chatbot using a conversational AI framework and how it was produced to satisfy the needs of the project.

5.1. Functional requirements

The functional requirements are the tasks that the program should accomplish on doing, i.e. the behaviour of the product.

5.1.1. Natural language understanding. The NLU model has to recognize the different wine sorts and be able to extract custom entity values from the user's request. It does not matter if the requested wine sort is contained in the training set or not, because either way, the chatbot has to identify the intent of the user's request correctly.

5.1.2. Natural language retrieval. The chatbot must be able to deliver the summaries from the detected entity values to the user. Every time the user asks a question, the wine sort must be detected and the corresponding summary retrieved and outputted by the chatbot. If the wine sort is not present in the database, then the default answer from the chatbot is returned or similar wine sorts are suggested to the user.

5.2. Non-Functional requirements

The non-functional requirements specify criteria that judge the operation of a system, rather than specific behaviours. They judge the software system based on the quality of various attributes, for instance, usability and reliability.

5.2.1. Efficiency of the chatbot. The efficiency of the chatbot is improved by data preprocessing. The summary retrieval process is more efficient and performant by preprocessing every summary. This is especially important when working with a lot of data and this is the case for this project because we are working with several thousands of wine reviews.

5.3. Design

The second part is a chatbot which retrieves the generated summaries out of the dataset in response to the user's requests. For this purpose, a conversational AI framework was needed. There are lots of different open-source frameworks available to the public to create this kind of chatbots, but a decision needed to be made.

The NLU model was trained using thousands of different examples. As we can see in figure 7, multiple sentence/question templates were used and filled with uppercase and lowercase wine sorts. This big training set implicates higher accuracy when it comes to the entity extraction

part. The training of the NLU model is very important as the chatbot needs to recognize wine sorts contained in the database and beyond. It should also be trained in such a way that by expanding the wine sorts dataset, the chatbot automatically works for the new entries as well. If it does not manage to extract the new entity values and recognize them as a wine sort, then the chatbot does not work as intended.

The chatbot has very basic actions. Every time the user writes a message, the intent has to be recognized. In this project, we have 3 different outcomes: greeting, goodbye and request for a summary. Apart from greeting the user and saying goodbye, the chatbot has to recognize whenever the user is asking a question. Using entity extraction the chatbot needs to identify the wine sort and check whether it is in the database or not. Then, it takes appropriate measures based on the result. Again, we have 3 different outcomes: the wine is found in the database, similar wine sorts are found in the database or nothing is found. When the wine sort corresponds exactly to an entry in the database, then the chatbot just outputs the corresponding summary. However, when the wine sort is not in the database, then the chatbot searches for a maximum of 5, randomly selected similar wine sorts and suggests them to the user. This is done to prevent the user from not getting the expected results when for instance, there only is a small typo in the wine name.

5.4. Production

The first thing to do in order to create the chatbot is to identify which open-source framework to use. There are lots of different options to choose from, but one which stands out in multiple reviews is Rasa. It is a very powerful and well-known tool to build chatbots using Python, which is why the decision was not complicated. Another benefit of Rasa is that it can be easily extended over time to implement several new features.

To kick off the chatbot creation, one very important step had to be done upfront, which was to set up a virtual environment on the local development machine. Python virtual environments are to create an isolated environment for each project, where each project can have its own dependencies, regardless of what dependencies every other project has [7]. This is the best way to avoid conflicts between different projects.

The next thing on the list was to install all the required packages and Rasa itself in the previously created virtual environment. This will install the default Rasa chatbot which already has multiple intents and responses working. The chatbot comes with several pre-defined stories and a trained NLU-model.

After cleaning all the code from the unnecessary intents, actions and stories it was now time to start defining the necessary actions the chatbot needs to perform. Since a chatbot is a very complex project containing a lot of files, it is simpler to explain what exactly needed to be modified, respectively added, and why.

1) **domain.yml**

The domain of the chatbot defines what user inputs it should expect, what actions to predict, how to respond, and what information to store [8]. The first thing to be defined in this file are the intents of the chatbot. Here we removed all the existing intents that pre-existed except for the greeting and goodbye, which allows the chatbot to greet the user and to say goodbye. A new intent was then added with the name **ask_info**, which will be responsible to detect if a user is asking a question about a wine sort. To detect the wine sort in the user's question, we have to define a new entity. The only entity in this project is **wine**, which represents the wine name in the chatbot. Apart from greeting the user and saying goodbye, 2 more responses were also added to the domain. The purpose of the first one is to return a default answer to the user when the chatbot detects a question but is not able to identify a wine name. In this case, the chatbot will ask the user "Sure! Which one?", since it did not detect the wine name. The second one builds upon a custom action which handles the cases when the chatbot detects a question by the user and is able to retrieve the entity value, thus it detected a wine sort in the question. This custom action will be explained in the last block of this list.

2) **stories.md**

The stories.md file of the chatbot contains real conversational data in the form of training "stories". A story is a representation of a real conversation between a user and an assistant. There are lines containing the intents and possible entities, which reflect the user's input. The action names below show what the chatbot should do in response [8]. The first thing to do was to clean the file and remove all the unnecessary stories. Since the chatbot does not have a lot of different actions, intents, etc, it is enough to define a story for each possible outcome. In this case, there are only 4 different outcomes: greet the user, say goodbye and then it depends whether the chatbot recognizes a wine name or not. Therefore, besides the 2 default stories, 2 more were added. Both of them are based on the **ask_info** intent added before, but one of the stories contains an entity and one not. The desired outcome is to handle both cases in 2 different ways. The story containing an entity means that the wine name was identified and therefore we need to handle this with a custom action to retrieve the summary out of the database. The other story means that no entity was identified, thus there is no wine name to use. In this case, the chatbot refers to the default response set in the domain.yml file.

3) **nlu.md**

The NLU model turns user messages into structured data. For this, we have to provide the chatbot with sentence/question examples, which allows it to understand the user messages. After providing the NLU model with the arbitrary long number of examples, it must be trained


```

(BSP2) flavio@Flavio-MBP bot % rasa shell
2020-05-19 15:20:45 INFO root - Connecting to channel 'cmdline' which was specified by the '--connector' argument.
Any other channels will be ignored. To connect to all given channels, omit the '--connector' argument.
2020-05-19 15:20:45 INFO root - Starting Rasa server on http://localhost:5005
Bot loaded. Type a message and press enter (use '/stop' to exit):
[Your input -> Hello
Hey! How are you?
Your input -> Tell me something about pinot noir
Red berries, blood orange, a touch of Grand Marnier, impressive body and spicy highlights all combine in a darn near irr
esistible rosé with the power to be a four season wine. The striking package sets it apart from all other Rex Hill wines
, and the rich, complex flavors add to the impression that this is something truly special. There's sufficient acidity t
o take a bit of a chill-down in stride, and the finish, though bone dry, has ample fruit sweetness to round out the mout
hfeel.
Your input -> ]

```

Figure 6: Chatbot interface with example question

[8]. The more examples we provide the chatbot with, more accurate it will work. Each line that starts with `##` defines a new block of examples by precisizing the desired intent. Apart from the default greeting and goodbye examples, everything was deleted inside this model. Since we have a new intent, which is `ask_info`, we needed to add a set of examples. The best way found to train the model was to take 7 example sentences/questions and use them with every wine sort in the database. For every wine sort, every sentence/question template was written twice: with capitalized initial letters and with lowercase initial letters. This covers the whole database including capital and lowercase examples and since we have 227 different wine sorts, this equals to $7 * 2 * 227 = 3178$ training examples in the NLU model.

```

## intent:ask_info
- How is the [Solomon Hills Vineyard](wine) like?
- How is the [solomon hills vineyard](wine) like?
- Can you tell me about [Solomon Hills Vineyard](wine)?
- Can you tell me about [solomon hills vineyard](wine)?
- I would like to know something about [Solomon Hills Vineyard](wine)
- I would like to know something about [solomon hills vineyard](wine)
- Can you tell me something about [Solomon Hills Vineyard](wine)?
- Can you tell me something about [solomon hills vineyard](wine)?
- Tell me about [Solomon Hills Vineyard](wine)
- Tell me about [solomon hills vineyard](wine)
- Give me details about [Solomon Hills Vineyard](wine)
- Give me details about [solomon hills vineyard](wine)
- I would like to know about [Solomon Hills Vineyard](wine)
- I would like to know about [solomon hills vineyard](wine)
- How is the [Don David Reserve](wine) like?
- How is the [don david reserve](wine) like?
- Can you tell me about [Don David Reserve](wine)?
- Can you tell me about [don david reserve](wine)?

```

Figure 7: Some training examples from the NLU model

4) data.sqlite

The data.sqlite file is the dataset containing all the wine reviews summaries. Originally generated by the algorithm, the file was a CSV file. However, to facilitate the fetching in the dataset and use it as a real database, the CSV file was converted to an SQLite database using an online tool.

5) actions.py

The last file which played a role in the chatbot creation was the actions.py file. This file contains a custom action, which in this case is the `ask_info`-action. This action handles the cases where the wine name was identified in the message written by the user. The first thing to do is to connect to the provided SQLite database which contains all the summaries to be retrieved. Then we extract the entity value, which represents the wine name, and search for it in the database. If the result is a hit, then we just output the summary and we are done. However, if the result is a miss (when the given wine name is not in the database), we search for wine names in the database that are similar to the one given by the user. If there are any, we take a maximum of 5 randomly selected wine names and suggest them back to the user. If there are still no results, we inform the user that nothing was found according to their search.

5.5. Assessment

This section states whether the different requirements set have been successfully reached or not.

5.5.1. Natural language understanding. The NLU model of the chatbot was trained with 3178 different examples.

This gives the chatbot enough examples to be able to identify and extract custom entity values from the user's requests. While testing the chatbot, multiple requests using wine sorts not included in the database were performed and marked as successful. The accuracy of the NLU model is therefore pretty good as similar requests not explicitly available in the training examples also work. The requirement was hereby met and it can be marked as successful.

5.5.2. Natural language retrieval. Each entity value detected by the chatbot as to be a wine sort is looked up in the database. If the wine sort corresponds to an existing wine sort from the projects database, there, of course, is a corresponding summary which can be retrieved. Every time a wine sort is matched in the database, its summary is fetched. If the recognized wine sort is not present in the database, the chatbot tries to identify similar wine sorts to suggest to the user. If there really is no match, then the user gets notified about the empty results. Thus, the chatbot meets the requirement of retrieving the summaries out of the database and deliver them to the user. The requirement can be stamped as successful.

5.5.3. Efficiency of the chatbot. This requirement is not mandatory for this project, but to have a more performant chatbot, it is of great interest to preprocess the data before the summary retrieval. Due to the high amount of wine reviews, generating the summary for each retrieval would be a waste of time and not efficient at all. Allowing the chatbot to just retrieve the already generated summaries when asked to, makes it more efficient and performant overall. Moreover, adding new wine sorts to the chatbot only requires an entry in the database since the NLU model was well enough trained to recognize wine sorts not included in the training set. This makes the chatbot very efficient and easy to scale, therefore the requirement is therefore met with success.

6. Conclusion

This paper presents a bachelor semester project whose goal is the use of summarization techniques to generate summaries out of wine reviews. The generated summaries are stored and used by a conversational wine recommender system. Users can interact with the chatbot by asking questions about wine sorts and get a short description about the wine sort as a return.

Overall the project can be closed off as a success because the initial tasks were all accomplished and work as intended. However, in the end it is debatable if the summaries generated by the algorithm are good enough to describe a given wine sort. In theory, the summaries seem to be well-performing in terms of the evaluation, but it is hard to evaluate the value of the information contained in each of the individual summaries when working in a real-life scenario.

There are multiple things that could still be done to improve the project in the future such as adding more information about the wine sorts. Currently, the chatbot only supports one type of question which is asking about the

wine sort. Adding new information about each wine sort would enlarge the database and the chatbot could provide the user's with more details about their favourite wine sort, for instance, the country of origin or the producer.

Additionally, the chatbot could, for example, be linked to a social media account of a wine selling company. By connecting the chatbot with the Facebook page of the company, it could answer the user's questions. Although connecting the chatbot with Facebook Messenger would be an interesting thing to do, it wasn't necessary to complete the project.

References

- [2] Samrat Babar: Text Summarization - An Overview
https://www.researchgate.net/publication/257947528_Text_SummarizationAn_Overview
- [3] Matthew Mayo: Approaches to Text Summarization - An Overview
<https://www.kdnuggets.com/2019/01/approaches-text-summarization-overview.html>
- [4] Types of Text Summarization approaches [picture]
<https://medium.com/jatana/unsupervised-text-summarization-using-sentence-embeddings-adb15ce83db1>
- [5] Tf-idf
<http://www.tfidf.com/>
- [6] Kavita Ganesan: An intro to ROUGE, and how to use it to evaluate summaries
<https://www.freecodecamp.org/news/what-is-rouge-and-how-it-works-for-evaluation-of-summaries-e059fb8ac840/>
- [7] Python Virtual Environments: A Primer
<https://realpython.com/python-virtual-environments-a-primer/>
- [8] Tutorial: Rasa Basics
<https://rasa.com/docs/rasa/user-guide/rasa-tutorial/>
- [9] Consumer chatbot perceptions
<https://www.userlike.com/en/blog/consumer-chatbot-perceptions>
- [10] Ish Jindal: Building chatbots: Why message length matters
<https://www.invisionapp.com/inside-design/chatbots-message-length/>
- [11] Ish Jindal: Chatbot Building Best Practices — Why Message Length Matters
<https://uxdesign.cc/chatbot-building-best-practices-why-message-length-matters-e951bed1b550>
- [12] Luke Reichold: Multi-document news article summarizer
<https://github.com/lukereichold/News-Summarizer>
- [13] What is ROUGE and how it works
http://www.ccs.neu.edu/home/vip/teach/DMcourse/5_topicmodel_summ/notes_slides/What-is-ROUGE.pdf

7. Appendix

Resources used during this project:

- 1) Rasa:
<https://rasa.com/>
- 2) Python implementation of ROUGE metric:
<https://github.com/pltrdy/rouge>
- 3) Wine reviews dataset:
<https://www.kaggle.com/zynicide/wine-reviews>