MSc in Business Analytics

Machine Learning and Content Analytics

Professor: X. Papageorgiou

Assistants: G. Perakis, D.Pappas, S.Kotitsas

Retrieval Augmented Generation in product reviews

August 2023, Athens

Angelidis Athanasios (f2822202)
Charizanou Evanthia (f2822220)
Gkouma Konstantina (f2822208)
Zikou Panagiota (f2822210)

Table of Contents:

Introduction..... 3
Problem Statement..... 3
Proposed Solution..... 4
The vision..... 5

Data Collection..... 5
Dataset Overview..... 6
Data Processing..... 11
Methodology..... 12

Model Results..... 19
Summarization Evaluation using ROUGE..... 20
Tools..... 23
Discussion, Comments/Notes and Future Work..... 23

Members- Roles..... 24
Time Plan..... 24
References..... 25

Introduction

At X Company, a leading force in the electronics sector of e-commerce, our dedication to enhancing the online shopping experience knows no limits. We deeply understand the pivotal role that product reviews play in shaping the decisions of consumers. Our aim is to ensure that individuals have all the practical and important information that assists them in making informed purchasing choices. However, we're aware that the current style of writing reviews can be confusing at times. Some reviews don't offer sufficient information or address particular concerns, creating uncertainty for potential buyers.

As a team of four members at X Company, driven by our strong dedication to generating new ideas and ensuring customer satisfaction, we're prepared to overcome these challenges. We're excited to bring in a solution that could really change things. It has the possibility to change how people look at product reviews and think about shopping online. Our idea is Retrieval Augmented Generation, which combines the power of information retrieval and natural language generation.

Also, we understand the bigger impact our work can have. We know that bad feedback can hurt how people see companies. So, as we work to make the shopping experience better, we're also making sure our reputation stays strong. We're doing this by dealing with negative feedback and working together with the makers of the products to fix any problems. This way, we want to make sure people see us as a trustworthy place to shop.

In conclusion, our dedication to coming up with new ideas, making customers happy, and managing our reputation keeps us moving forward. We imagine a future where customers not only feel sure about reading reviews, but also make their purchasing decisions mainly by interacting with those reviews. Our plan shows how we're always committed to helping customers and creating a good and trusted shopping experience.

Problem Statement

Through a comprehensive analysis, we have identified a significant challenge associated with the existing state of product reviews. The issue is that when customers aren't happy

with a product, they often share their disappointment by leaving negative comments on our website.This situation has the potential to damage our reputation, as these unfavorable reviews can overshadow the positive ones.

Considering the high level of competition in this sector, it's crucial for us to take proactive steps to address this challenge. Not only do we want to keep our reputation of being a reliable online shopping platform, but we also want to make a space where customers can feel confident about their choices when buying new products. Plus, given the intense competition in the market, it's even more crucial for us to stand out, maintain customer engagement and ensure their coming back.

Proposed Solution

In response to this issue, we decided to implement a Retrieval Augmented Generation (RAG) system. This innovative approach involves two main components: retrieval and generation.

1. For the **Advanced Information Retrieval** component, our proposal involves utilizing advanced methods to retrieve information that goes beyond just matching keywords. By understanding the meaning and context of the reviews' content, our system will accurately find the most suitable reviews for each user's question. This precision ensures that users receive reviews that directly address their specific concerns, making the information they find more relevant and reliable.

2. In the context of **Natural Language Generation Excellence**, we're using advanced models to generate text. This helps us improve the quality of reviews that are already there. We're making detailed summaries and adding more information from what we've found. The goal is to give users a full picture of the product – what it has to offer, its benefits, and any downsides it might have. This makes regular product reviews really useful for making decisions, helping customers feel more sure about their choices.

The vision

We're aiming for more than just small improvements. We see a big change happening that will make it easy for our customers to find and choose products they love. By giving customers reviews that truly matter and are trustworthy, we're making their decision-making process a lot simpler. This gives them the confidence they need to choose products that match their needs and preferences.

Because of this, we expect good things to happen. Customers will be happier, more people will buy things and our bond with our customers at X Company will become even stronger. This plan perfectly aligns with our goal of using technology to make our customers' lives better. It also shows how we're leaders in our industry, always finding new and better ways to serve our customers.

Expanding on our vision, we have some additional goals in mind. We're looking to increase sales on our site and boost our site's reputation. Thanks to our algorithm and the way we'll be sorting the reviews– from the best-reviewed to the least – we're not only helping customers make better choices but also pushing makers to improve their performance. This means the overall quality of products will go up, creating a win-win situation for everyone involved.

Data Collection

In order to perform RAG, we needed an available dataset to serve as the foundation for generating contextually relevant and informative responses. For this purpose, and after thorough research, it was decided to utilize the Amazon Customer Reviews dataset that was publicly available through kaggle. The dataset is sourced primarily from Amazon's e-commerce platform, where customers voluntarily contribute their opinions and experiences regarding products, and more specifically electronics, available on the website. The choice of this dataset was not arbitrary, it was made due to its format that closely mirrors the data we extract in our company. The dataset used can be found here.

Dataset Overview

We directed our focus towards the domain of electronics among the diverse product categories available in the dataset by leveraging the amazon_reviews_us_Mobile_Electronics_v1_00.tsv data file. It consists of 104,854 rows and 15 columns out of which we worked with the 6 most significant columns. More specifically, the columns considered more appropriate and that we worked with, are the following:

- product_id: a unique Product ID the review pertains to
- star_rating: a 1-5 star rating of the review
- helpful_votes: the number of helpful votes
- total_votes: the number of total votes the review received
- review_headline: the title of the review
- review_body:the text of the review

This dataset is utilized as a test dataset given as input to the NLP models that are created.

Dataset Descriptive Statistics:

The first step after choosing the dataset was to explore and understand it by analyzing its descriptive statistics.

Summary statistics for numerical columns:

	star_rating	helpful_votes	total_votes
Count	104,852	104,852	104,852
Mean	3.76	1.24	1.62
Standard Dev	1.52	7.07	7.91
Minimum	1	0	0
25th Percent	3	0	0
Median (50%)	4	0	0

75th Percent	5	1	1
Maximum	5	769	791

Interpretation:

- The ‘star_rating’ variable has a mean of approximately 3.76, indicating that, on average, reviews are slightly positive. The ratings range from a minimum of 1 to a maximum of 5, with the majority of reviews concentrated between 3 and 5 stars.
- The ‘helpful_votes’ variable has a mean of approximately 1.24 but has a relatively high standard deviation of 7.07, indicating significant variability. The range of helpful votes spans from 0 to 769, with 75% of reviews receiving 1 or fewer helpful votes.
- The ‘total_votes’ variable has a mean of approximately 1.62 and a standard deviation of 7.91. Similar to ‘helpful_votes’, this variable has a wide range of values, with the highest total votes being 791. Most reviews receive 1 or fewer total votes.

Summary statistics for text data:

	review_headline	review_body
count	104850	104851
unique	68373	102054
top	Five Stars	Good
freq	7318	120

Interpretation:

Concerning the ‘review_headline’ column:

- There are 104,850 reviews with a headline.
- The ‘review_headline’ column contains 68,373 unique headlines.
- The most frequently occurring headline is “Five Stars” which appears 7,318 times.

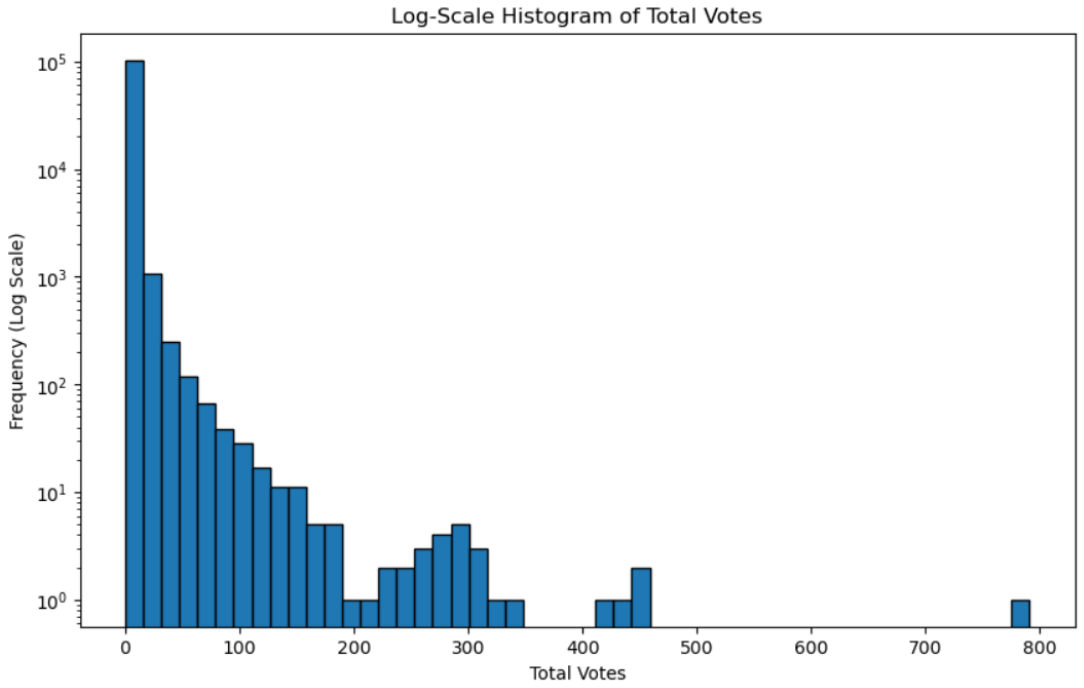
Concerning the ‘review_body’ column:

- There are 104,851 reviews with body text.
- The ‘review_body’ column contains 102,054 unique reviews.
- The most frequently occurring review text is “Good” which appears 120 times.

Log-Scale Histogram of Total Votes

The histogram below helps us see how popular or influential reviews are on a log-scale, which is useful when some reviews get a lot of votes and others get very few.

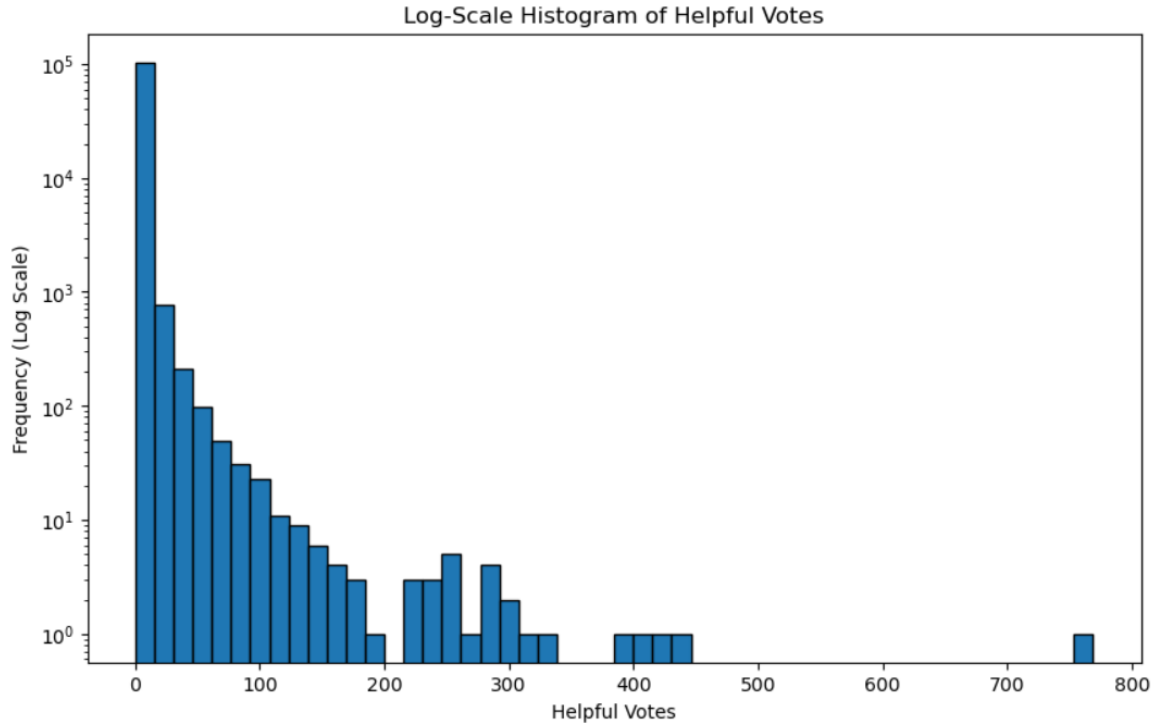
- The first bin, representing reviews with very high vote counts (greater than or equal to 100,000 votes), contains a significant number of reviews. This means that a small portion of reviews receives a very large number of votes.
- As we move to higher vote count ranges, we see a continuous decrease in the frequency of reviews. This means that fewer and fewer reviews receive higher numbers of votes.
- There is an absence of bars in the histogram within the range of approximately 500 to 700 total votes. This gap indicates that there are very few, if any, reviews that fall within this specific vote count range. In other words, reviews receiving votes in this range are relatively rare.
- When we reach the bin that represents reviews with approximately 790 total votes, we observe that the frequency drops to a very low level. This means that only a small number of reviews receive such a high number of votes.



Log-Scale Histogram of Helpful Votes

We notice that both the log-scale histograms of ‘Total Votes’ and ‘Helpful Votes’ exhibit similar distribution trends. More specifically:

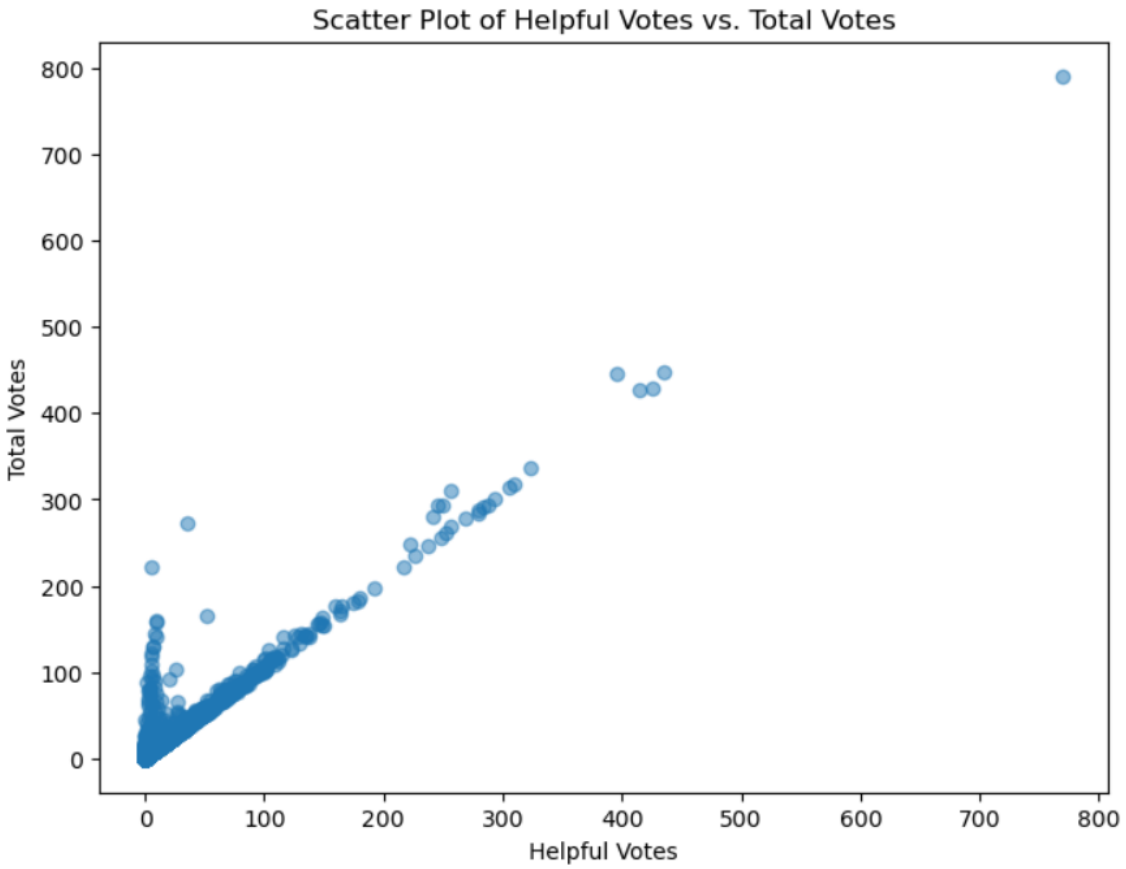
- In both histograms, we observe a significant concentration of reviews on the left side, indicating that the majority of reviews receive relatively few votes (either total or helpful). This observation aligns with the fact that the median (50%) values for both ‘Total Votes’ and ‘Helpful Votes’ are 0.
- Both histograms extend into the right tails, revealing that there are reviews with higher counts of votes. This tailing pattern corresponds to the high standard deviations in both ‘Total Votes’ (7.91) and ‘Helpful Votes’ (7.07), signifying a wide range of variability.
- Similar gaps in the middle range of vote counts exist in both histograms. This common feature suggests that reviews with moderate vote counts are relatively less frequent.



Scatter Plot of Helpful VS Total Votes

- As expected, the majority of data points fall along a diagonal line, indicating that reviews with higher total votes also tend to have more helpful votes. This observation makes sense, as popular reviews are often considered helpful by other users.

- In contrast, there are outliers where reviews have very low helpful votes (close to zero) but relatively high total vote counts, from 100 to more than 200. This suggests that there are reviews that, despite receiving many total votes, may not be seen as helpful by other users.
- There is a distinct outlier with a high count of both helpful votes and total votes, approximately reaching 800 for each variable. This data point represents a review that is not only popular but also widely considered helpful by other users.



Additional/complementary datasets:

I. "amazon_reviews_us_Automotive_v1_00.tsv"

In order to train and fine tune the Summarization-Product-Reviews model, which is analyzed in detail in the "METHODOLOGY" section, we thought that additional data were needed in order to have a variety and a less biased model. That's why we decided to choose at random an additional dataset from a different product category and more specifically the

“amazon_reviews_us_Automotive_v1_00.tsv” file containing reviews for the automotive category. The same preprocessing steps analyzed in the next section for the electronics dataset were also followed for the automotive dataset.

II. “summaries_api.txt”

To fine-tune our summarization model, we need to create target summaries, that is, those we desire to have as output. Since we focus on producing summaries of product reviews, we are only interested in the comments (either positive or negative) regarding the characteristics. To do so, we leveraged the OpenAI GPT-3.5 language model.

Before using the OpenAI API, we defined a prompt instructing the model to create concise summaries of reviews. The API uses the GPT-3 "text-davinci-003" model with a 90-token limit per summary and a 0.9 temperature for creativity. Each review is processed individually, and the resulting summaries are saved in .txt format alongside their respective reviews."

Data Processing

It was observed that not all the reviews are of the same importance. Some reviews had gained 0 helpful votes, while others 0 votes in general. The aim is to consult customer reviews, so we need only the valuable ones. We must ensure that the reviewer had no intention to advertise, defame or provide false information regarding a product or seller with his review, and thus not take it into consideration.

That’s why we chose to center our approach on prioritizing reviews that provide the most substantial information. To achieve this, we filtered reviews based on “helpful_votes” and “total_votes” columns and retained only those reviews having more helpful votes than the half of the total votes for a review, ensuring that it has received more positive votes than negative ones. Additionally, we also involved reviews that had received at least 1 helpful vote, i.e. gathered at least some positive feedback from users.

After the data processing, the total number of valuable reviews reached 15,613, with no missing values present in the dataframe.

Methodology

In order to apply RAG to product reviews, Pinecone was utilized. Pinecone is a cloud-based vector database, which provides optimized and efficient querying and storage capabilities for embeddings.

These embeddings represent vectorized versions of the content within a high-dimensional space. The key idea behind embeddings is that similar words of the review are mapped to similar points in this high-dimensional space, while dissimilar ones are mapped farther apart. These embeddings capture the semantic relationships between texts facilitating the retrieval of relevant reviews (in our case) based on their similarity to a query- question given by the user.

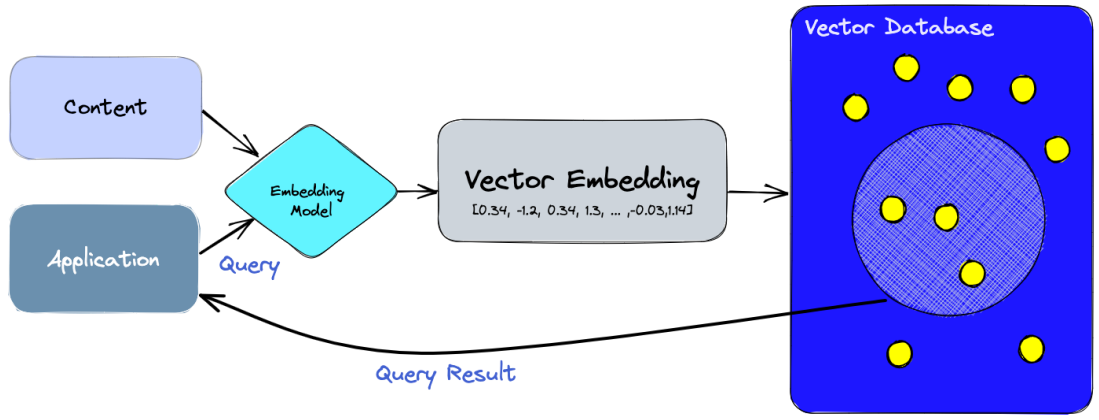


Image 1: The role of a vector database, Source: Pinecone

In our case, we specified the name of the index to be “generative-review”. This is where our reviews will be stored. On top of that, the similarity metric was set to “cosine” and the embedding dimension to “768”, since most retrievers use such parameters.

Haystack nodes:

I. **Node:** “Retriever” - **Model:** sentence-transformers/all-mpnet-base-v2

As a first step, a retriever is initialized to perform the document retrieval process, returning a set of candidate reviews that are most relevant to the query.

Specifically, we made use of the “EmbeddingRetriever” node, which will transform an input query into a vector and compare it with the rest of the vectors in the DocumentStore (Pinecone). In the end, the retriever will return the most similar reviews in regard with the input query.

To create the embeddings, we utilized the “all-mpnet-base-v2” fine tuned model from the Sentence Transformers library in Hugging Face Model Hub and we configured it to retrieve the top 20 most similar reviews.

II. **Node:** “Summarizer”- **Model:** ThanosAng/Product_Review_Summary(Hugging Face)

After obtaining the reviews through the retriever, our next step is to create summaries that highlight the characteristics, whether positive or negative, that are mentioned in these reviews. By summarizing the reviews, our goal is to extract and maximize the relevant information.

To create these summarizations, we decided to fine tune a pre-trained model from HuggingFace. We chose to use the “facebook/bart-large-cnn” as a base model, since it provides great summaries for its number of parameters.

As mentioned previously, during the training and fine-tuning stage of the model, we used a dataset consisting of reviews regarding products from both the mobile electronics and automotive categories.

After converting the review body column to a list type, a new dataframe was created consisting of two columns; one including the reviews and one including their corresponding target summaries, generated by GPT’s API and saved in the aforementioned “summaries_api.txt” dataset. The, overall, resulted training dataset was structured as follows:

- rows 0 to 499 correspond to the reviews and target summaries for mobile electronics data
- rows 500 to 899 correspond to the reviews and target summaries for automotive data

> 1st stage of fine-tune: Initializations & Processing

From that dataframe, named “training”, the first 500 rows were converted into the “dataset” dataset, using the Hugging Face “Datasets” library. This dataset is going to be used to finetune the pre-trained BART model. The training- test split was set to be 90-10.

By using the Hugging Face Transformers library, we load a pre-trained tokenizer for the “facebook/bart-large-cnn” model. The 'AutoTokenizer' is then employed to create a model-specific tokenizer. This tokenizer's primary role is to get text input ready for the BART model by breaking it into tokens, transforming it into numerical input, and then reversing the process to make model outputs understandable in human-readable form.

> Peprocess_function:

To prepare reviews and target summaries for training our sequence-to-sequence model, we developed a ‘preprocess_function’. This function extracts columns containing reviews and target summaries from the input dataset, which are then tokenized using our previously created ‘tokenizer’ . We set a maximum token limit of 512 for reviews and 90 for summaries. If these limits are exceeded, reviews are truncated accordingly. The ‘inputs’ and ‘labels’ are combined and returned when the function is called.

The “preprocess_function” is then applied to the entire dataset containing 500 rows of products from the electronics category, tokenizes and processes accordingly all the reviews and target summaries in batches. Finally, the results are stored in a new dataset, “tokenized_billsum”, which is going to be used for the training of the pre-trained model.

> Compute_metrics function:

To assess our model's performance, we created a ‘compute_metrics’ function that utilizes the ROUGE (Recall-Oriented Understudy for Gisting Evaluation) metric for evaluating auto-generated summaries.

The process involves saving model predictions and labels as ‘eval_pred’, where predictions are the generated review summaries, and labels are the reference review summaries from the dataset. Both predictions and labels are tokenized, special

tokens are removed during tokenization, and ‘-100’ (which is a placeholder for ignored tokens) in labels is replaced with the tokenizer's pad token ID.

After that, the ROUGE metric is computed by comparing tokenized model predictions to tokenized ground truth summary labels. Additionally, we compute the mean number of tokens per batch of generated predicted summaries and round the value metrics to four decimal places before returning them.

> DataCollatorForSeq2Seq & AutoModelForSeq2SeqLM

Furthermore, a data collator object is initialized using DataCollatorForSeq2Seq. The data collator will form a batch by leveraging the training dataset and also dynamically pad the received inputs and labels. We define as arguments our tokenizer and the pre-trained BART model and after that, the model that is to be fine-tuned is loaded using AutoModelForSeq2SeqLM class .

> Training_args & trainer:

The last step is the actual training execution of the model. The training process was executed using the Hugging Face's training API combined with PyTorch. In more detail, the training arguments are defined using the Seq2SeqTrainingArguments as follows:

- output_dir: corresponds to the directory where model checkpoints and training logs will be saved
- evaluation_strategy= “epoch”: the evaluation strategy is set to be done at the end of each epoch
- learning_rate=2.3e-5: corresponds to the learning rate used during training
- per_device_train_batch_size=7: defines the size per GPU/TPU core/CPU for training
- per_device_eval_batch_size=7:defines the size per GPU/TPU core/CPU for evaluation
- weight_decay=0.008: defines the weight decay to apply to prevent overfitting
- save_total_limit=3: defines that the maximum number of checkpoints to keep will be 3
- num_train_epochs=3: defines that the number of training epochs is 3
- predict_with_generate=True: indicates predictions generation during evaluation is enabled

- fp16=True: indicates that 16-bit precision training is used to accelerate training
- push_to_hub=True: indicates that the trained model and tokenizer is pushed to the Hugging Face Model Hub after training
- optim="adamw_torch": indicates that adamw_torch optimiser is used

For the initialization of the trainer, Seq2SeqTrainer was used, which is an extension of the standard Trainer class using the following parameters, as well as incorporating the computation of Rouge metrics inside the evaluation loop with:

- model=model: defines the the pre-trained model
- args=training_args: defines the training configuration
- train_dataset=tokenized_billsum[“train”]: defines the training dataset in a tokenized format
- eval_dataset=tokenized_billsum[“test”]: defines the test dataset again in a tokenized format
- tokenizer=tokenizer: defines the tokenizer
- data_collator=data_collator: defines the data collator object responsible for batching and padding previously set with DataCollatorForSeq2Seq
- compute_metrics=compute_metrics: defines the function for the evaluation metrics computation

> Fine-tuning of BART

Finally, by running trainer.train() command, we execute the BART model finetune on our mobile electronics data. This step combines and utilizes all the previously established and configured objects. The resulting model is named “final_model2” and was stored locally in our resources.

> 2nd stage of fine-tune

As mentioned earlier, we adopted a two-stage fine-tuning approach for our final model. In the 2nd stage, we followed the same steps as the 1st stage, but instead of using the pretrained ‘facebook/bart-large-cnn’ model, we employed our ‘final_model2’. We trained this stage on rows 500 to 899 of the initial ‘training’ dataframe, which contains automotive data. These rows were converted into the ‘dataset’ dataset. For checkpoint purposes, we switched from ‘final_model2’ to ‘Summarization-Product-Reviews’.

> Fine-tuning of final_model2

Once again, by running trainer.train() command, we execute the “final_model2” model finetune on our automotive data. This step combines and utilizes all the previously established and configured objects. The resulting model is named “Summarization-Product-Reviews” and represents our final model that will be employed and evaluated in the next section.

III. **Node:** “Prompt” - **Model:** gpt-3.5-turbo

Additionally, we utilize the GPT-3.5 Turbo language model from OpenAI to generate responses based on user-provided query-questions and the previously mentioned top 20 most similar reviews. This involves processing the reviews and obtaining answers to the user's queries from the model.

To execute this, the first step was the setup of the GPT-3.5 language model through the OpenAI API. The model is then instructed, through the PromptTemplate named “rag_prompt”, to generate a comprehensive answer by synthesizing key information from the most relevant reviews and the user's question.

To achieve this, we give as input variables the “documents” and most specifically “{join(documents)}” which combines the relevant reviews into a coherent text returning only one prompt instead of 20 and the “{querry}” where the answer is posed to the model. After that, the output_parser used, converts the output of the model to the Haystack Answer object using the AnswerParser() and, thus, extracts the generated answer from the model's response.

After completing the setup of the model, we proceed to initialize the PromptNode responsible for creating prompts for reviews summarization. We specify that the model of interest is “gpt-3.5-turbo”, set the “api_key” parameter and define the prompt template as the previously created “rag_prompt”. The generated responses are saved in the “prompt_node” variable.

IV. **Node:**"Translator" **- Model:** papluca/xlm-roberta-base-language-detection **&** Helsinki-NLP/opus-mt-fr-en **&** Helsinki-NLP/opus-mt-en-fr

Finally, we decided to add an extra feature to enable users to interact with our RAG system, also in other languages apart from English. To start with, we added French language. In order to make this feasible, we utilized two pre-trained transformer models from hugging face, the Helsinki-NLP/opus-mt-en-fr to achieve the French-to-English translation and the Helsinki-NLP/opus-mt-en-fr to achieve the English-to-French. More specifically, a "Translator_Input" node was created where the user will give a question query. In order for the model to understand which is the language that the query is written in, we used the "papluca/xlm-roberta-base-language-detection" model from Hugging Face, for text classification which serves as a language detector. According to the classification outcome, and if the detected language is not English, from the instantiated dictionaries the appropriate language mapping model is selected. These dictionaries contain mappings between foreign languages and English, so as for the model to be able to converge between the different languages. After choosing the model that fits the detected language, the query is translated into English and all the pipeline's aforementioned components will run as if the query was in English. Finally, the model's response to the query will be translated from English back into the original foreign language and, then, returned to the user. It should be noted that if the query is classified as being in English, the pipeline runs without using any language mapping model, it just goes through the pipeline and gives a response to the query. Currently, the "papluca/xlm-roberta-base-language-detection" model supports 20 languages: arabic, bulgarian, german, greek, english, spanish, french, hindi, italian, japanese, dutch, polish, portuguese, russian, swahili, thai, turkish, urdu, vietnamese and chinese. In our case, we chose to use French but all of them can be incorporated with their dictionaries and used by the users.

The final step in setting up the RAG system for product reviews is to form the pipeline by combining the previously defined Haystack nodes. More specifically, we include to the pipeline the "Translator_Input" node that translates the input query from the foreign language to English, if this is the case, the "Retriever" node which is responsible for information retrieval based on the query given as input and returning the top 20 most relevant results, the "Summarizer" node which is responsible for summarizing the reviews to scale down the

number of tokens and the “Prompt” node which is responsible for generating answers based on the reviews retrieved by our retriever. Once the pipeline is set up with the aforementioned nodes, it can be executed by giving a sample query as input, so as to generate an answer based on the information retrieved from the product reviews.

Model Results

In this section, we present the outcomes of deploying our summarization model to handle product reviews effectively. To illustrate this, an example is provided where we selected a random product and a corresponding user query:

- Product: **“**Sony Reader**”**
- Query: **“**Is the product value for money**”**

The model provided the following response:

- Response: **“**The product offers a variety of features and functionalities that are highly praised by users, such as easy reading, multiple font sizes, and a convenient touch screen. It is also commended for its durability and long battery life. However, there are some drawbacks mentioned in the reviews, such as poor software and limited content in the Sony bookstore. Some reviewers also mentioned issues with glare and reflection on the screen. Despite these drawbacks, the product is still considered a good investment by many users due to its ease of use, intuitive interface, and compatibility with various file formats. Additionally, the device is praised for its form factor and value for money compared to traditional books. Overall, while there are some minor issues, the product offers a good balance of features and performance for its price.**”**

As far as the french language is concerned, another example was employed, by giving a query in french this time, for the same product:

- Query: **“**Y a-t-il des reflets?**”**
- Response: **“**Oui, il y a des réflexions mentionnées dans les commentaires. Certains évaluateurs ont mentionné que l'écran est réfléchissant et peut causer des reflets et des reflets, ce qui rend difficile à lire. Cependant, d'autres évaluateurs ont mentionné que les problèmes d'éblouissement et de réflexion peuvent être réduits en ajustant

l'inclinaison de l'écran ou en utilisant un protecteur d'écran. Il semble que la quantité de réflexion expérimentée peut varier en fonction de l'angle et des conditions d'éclairage.”

In order to check that the model will give the same response if the same question was made in english, we run again the example with the query in english, and the results were exactly the same:

- ➔ Query: “Does the product reflect”
- ➔ Response: “Based on the reviews, the product does reflect light and has issues with glare and reflection. Many reviewers mentioned that the screen is reflective and makes it difficult to avoid glare from light sources. Some reviewers found the glare to be a major issue, while others found it to be a minor inconvenience that can be mitigated by adjusting the screen tilt or using a screen protector. It is recommended to use a cover with a built-in light for night reading. Overall, while the product has positive features, the glare and reflection issues are a drawback for some users.”

Generally, by examining the model’s responses, it can be observed that its answer included a relatively thorough examination of the product. Additionally, it took into account previous feedback, while using clear and simple language for the user to understand. Finally, it concludes using all the provided information with a direct answer to the addressed question.

Summarization Evaluation using ROUGE

In order to evaluate the performance of our summarization mode, ROUGE metrics were utilized. More specifically, ROUGE, or Recall-Oriented Understudy for Gisting Evaluation, is a set of metrics and a software package used to assess the performance of natural language processing applications, and in our case automatic summarization. The metrics compare the automatically produced review summaries against the reference- ground truth summaries that we had generated with the help of GPT API, by counting the number of overlapping units such as n-gram, word sequences and word pairs.

Our Product_Review_Summary model, achieved the following results on the evaluation set:

➔ Validation loss = 1.4651:

Validation loss assesses the performance of our deep learning model on the unseen reviews. It is calculated from the sum of the errors for each review in the testing dataset. In our case, validation loss equals 1.4651 which indicates the average error or mismatch between the predicted summaries generated by the model and the actual target summaries in the validation dataset. The error from the expected summaries is relatively low, meaning that generally the summarization is of good quality during validation.

➔ Gen Len = 77.275:

Gen Len represents the average length of the generated summaries produced by the model. More specifically, it indicates that, on average, the summaries generated by the model have a length of approximately 77 tokens. In our case, it can be concluded that the generated summaries are neither too short nor too long, but they have a rather logical length.

➔ Rouge1 = 0.5072:

The Rouge1 metric measures the degree of overlap between the words present in the summary output produced by the model and the words in the reference summary output. It, actually, quantifies how similar the two summaries are, indicating the model's ability to accurately capture and reproduce the content from the reference.In our case, Rouge1 equals 0.5072, meaning that, on average, approximately half of the words in the model's output match those in the reference output.

➔ Rouge2 = 0.2453:

The Rouge2 metric measures the degree of overlap between the words present in the summary output produced by the model and the words in the reference summary output, focusing on two-word sequences (bigrams).In our case, Rouge2 equals 0.2453, meaning that, on average, approximately 24.53% of the bigrams in the model's output match those in the reference output.

➔ Rougel = 0.3808:

The Rougel metric measures the degree of overlap between the words present in the summary output produced by the model and the words in the reference summary output, based on the longest common subsequence of words between them at a

sentence level. This metric is order sensitive, meaning that it is affected by the generated order of the words. In our case, the Rougel metric equals 0.3808, meaning that, on average, approximately 39% of the terms of the longest common subsequence between the model's output and the reference output appear in the exact same order.

➔ Rougelsum = 0.3822:

In contrast to Rougel, RougeLsum refers to tit being computed over a whole summary, and not as the average over individual sentences.In our case, the Rougelsum metric equals 0.3822, meaning that, on average, approximately 38% of the terms in the longest common subsequences between the model's output and the reference output appear in the exact same order.

It should be noted that, most of the time, two summaries are rather unlikely to be generated exactly the same and this is why Rouge metrics usually peak around 0.50, indicating an output of good quality. ARegarding our model's metrics, they show that our model generates relatively good-quality summaries, as most of the metrics closely approach the 0.5 threshold, with approximately 50% word overlap. This demonstrates its effectiveness in word extraction and summarization.

In Table 1 below, the validation metrics for our Product_Review_Summary model are provided. The table contains information on the Epochs, Validation Loss, all the aforementioned Rouge metrics and the Gen Len metric.

Training results

Training Loss	Epoch	Step	Validation Loss	Rouge1	Rouge2	Rougel	Rougelsum	Gen Len
No log	1.0	45	1.4795	0.5164	0.246	0.3857	0.385	78.875
No log	2.0	90	1.4651	0.5072	0.2453	0.3808	0.3822	77.275

Table 1: Validation metrics for Product_Review_Summary model

It can be observed that, firstly, the Validation Loss decreases from 1.4795 in Epoch 1 to 1.4651 in Epoch 2 indicating that the model's generalization to unseen data improves in Epoch 2. Secondly, Rouge metrics for both epochs are very close, with slightly lower scores in Epoch 2. However, since the decreases are really small, the model's performance is maintained between the two epochs. Finally, the average generated length decreases from 78.875 tokens to 77.275 tokens from Epoch 1 to Epoch 2, respectively. This decrease signifies that the model is becoming more concise in its summaries.

Tools

In our project, the following tools were employed:

- Summary generation with OpenAI API
  OpenAI API was used for the creation of the review-summaries by employing its summarization NLP models
- Python Programming language
  The entire project is implemented using Python as the programming language
- Haystack framework for building question answering system
  For building the question answering system, we utilized Haystack framework, since it enables the development of simple question answering pipelines
- Pinecone vector database
  Pinecone vector database was utilized for storing and querying embeddings of our text data
- Libraries: Pandas, NumPy, Hugging Face Transformers, Hugging Face Pytorch, Hugging Face Datasets, Hugging Face Tokenizers

Discussion, Comments/Notes and Future Work

The idea and the implementation analyzed in this report, was made as an assignment for our master's degree. Thus, there is definitely room for improvement. The first limitation of our model is the size of the training dataset used. The training dataset should be further expanded in size and also to contain a wider variety of product categories, so that it can be able to understand reviews across diverse domains. Furthermore, there are also scale

limitations that should be addressed. The created model is not scalable, thus further work should be done in order for the model to be optimized and able to perform well in larger amounts of reviews and products.

Members- Roles

Our team consists of four members who bring unique backgrounds and skills to the table; Angelidis Athanasios, Gkouma Konstantina, Charizanou Evanthia and Zikou Panagiota. We maintained close collaboration throughout every stage of the assignment, by scheduling meetings and online calls to ensure we were on the same page. However, given our individual interests and strengths, Athanasios concentrated more on the technical part of the assignment, while Konstantina, Eva and Panagiota focused more on the initial dataset exploration, writing of the business report and preparing an effective presentation, so as to convey our work. This division of roles, enabled us to work effectively and to make the most of our combined skills for a successful result.

Time Plan

After receiving approval from our professor for our selected subject (Business Plan), we structured our project timeline as follows:

- Subject Selection (Business Plan) - Day 0: We initiated the project by identifying a subject suitable for introduction as a business plan. After approval, we proceeded to the next stages.
- Code Development - Week 1-2: During the first two weeks, our team focused on writing the necessary code to support the chosen subject. This phase included developing algorithms to integrate the RAG approach into the company's operations.
- Report Generation - Week 2-3: With the code development completed, we dedicated Weeks 2 and 3 to create a detailed report summarizing our findings, methodologies and commentary on our discoveries.

- Presentation (.ppt) - Week 3-4: In Weeks 3 to 4, we prepared a presentation to effectively communicate our project's key highlights, strategies and results.

References

Documentation

Auto Classes

bart-large-cnn

Datasets

Haystack (1)

Haystack (2)

Helsinki-NLP/opus-mt-en-fr

Helsinki-NLP/opus-mt-fr-en

Metric: rouge

PineconeDocumentStore

PromptNode

PromptNode API

Retriever

Trainer

XLM-RoBERTa

Bibliography

Lin, C. Y. (2004, July). Rouge: A package for automatic evaluation of summaries. In Text summarization branches out (pp. 74-81)