Master of Science (MS) in Data Science

Module: ITC6010A1 – Natural Language Processing

Instructor: Lazaros Polymenakos

Students:

Alkiviadis Kariotis 241735

Celsa Almeida 267628

Georgios Aliferis 272183

Konstantinos Margaritis 271868

Term: Spring Term 2023

Type: Group Term Project

Submission Date: Monday, 24 July

Table of Contents

Introduction

In this project we aimed to perform in-depth analysis and natural language processing (NLP) of customer reviews listed within the Women's Clothing E-Commerce Reviews Dataset (Kaggle) (Brooks 2016) which is focused on 23, 485 reviews written by customers of a women's clothing brand. It has nine supportive features including the review text. The provider of the dataset, Nick Brooks has clarified that the dataset is real commercial data, web-scraped around 2016 and then anonymized. References to the company in the review text and body have been replaced with "retailer".

Our project includes EDA, cluster exploration, topic extraction, summarization, sentiment analysis and both review and dataset querying with the aid of simple chatbots/generative AI . Trending analysis was not viable since our dataset does not contain any feature w.r.t time tracking.

Our python code snippet(s) covers the entire data science pipeline, starting from loading and cleaning our dataset, extensive EDA (exploratory data analysis) to identify important characteristics, topic extraction of review text, summarization, sentiment analysis, applying a question answering model and more specifically experimenting with the FLAN model from google and finally creating chat bot for data frame interactions. Through the following extensive report, we will unravel and try to elaborate on all the processes encapsulated in our code snippets, trying to illuminate the underpinning reasoning behind each method.

Main NLP Tasks

Data Loading & Preprocessing

As per standard workflow, we first acquired the data and then imported the required standard Python and NLP libraries such as Pandas, NumPy, Seaborn, Nltk, Sklearn etc. Additional libraries were imported later as analysis progressed. For loading, cleaning and handling missing values in the csv dataset, we used the Pandas library.

We proceeded with removing the rows of the dataset that contained missing review text, so we could be certain about the completeness of the data. Special attention was given to the column named ‘Review Text’, as it plays a vital role for our later analysis steps that we performed. So, as aforementioned, by using the dropna function of pandas, we completely cleared the dataset based on the particular column, and in that way, we ensured the availability of textual data that we were going to be needed for the effective sentiment analysis and topic extraction.

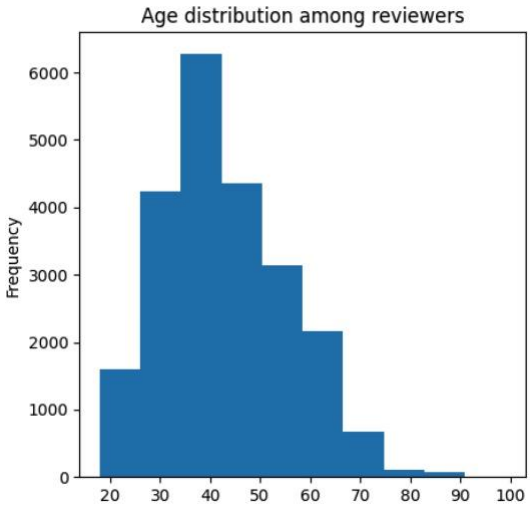
The data was then pre-processed for NLP analysis. We used NLTK's RegexpTokenizer to achieve text tokenization, meaning to split the text into tokens, i.e. individual words or punctuations. Then in order to ensure uniformity, we converted all our data into lower case. We proceeded with removing punctuations. Furthermore, we removed all the common ‘stop words’ that while having high frequency do not have notable significance (such as “the”, “and”, “a”, etc.). Finally, we used lemmatization, which is a technique towards reducing the words down to their base root form (like for example “playing” becomes “play”). This aids us not only at grouping similar words together but also by reducing the dimensionality of the data.

Exploratory Data Analysis

Exploratory data analysis process or EDA refers to investigation of a given dataset in order to understand its trends, its characteristics and patterns (Amaratunga et al., 2009). EDA also helps identify any anomalies in the dataset, reveals hidden insights and sets the foundation for model building and subsequent analysis, helping us to refine our general approach. After identifying columns with corresponding descriptions, we proceeded with plotting various features with Matplotlib and Seaborn python libraries.

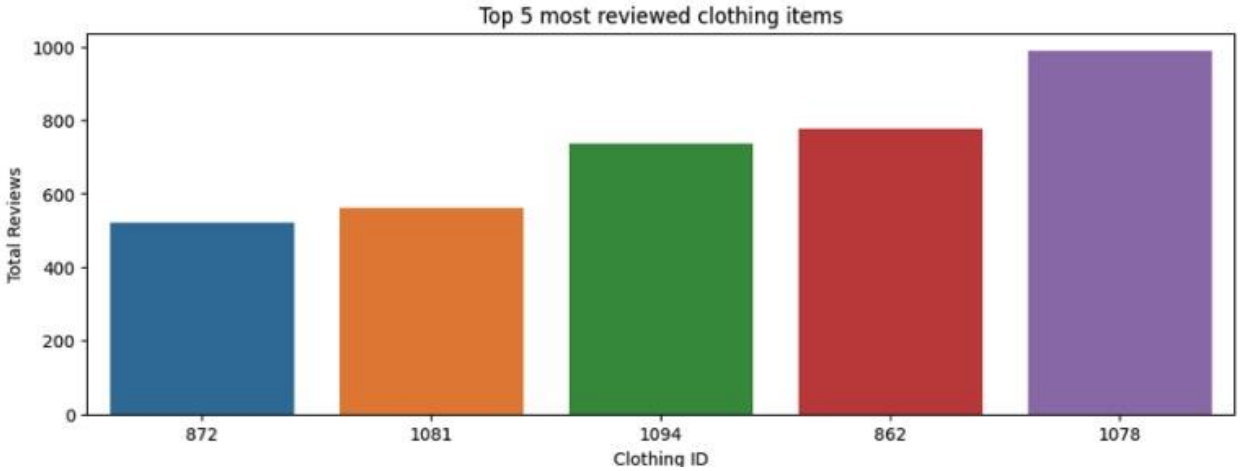
Age Distribution of Reviewers

We began our EDA by investigating the age distribution of the reviewers. The histogram function of Matplotlib was most suitable, since it provided immediate insights into the most actively participating age group(s). For a business, gaining this kind of understanding could be vital, since demographics and the overall understanding of how different age groups interact with the e-commerce platform constitute the basic criteria used to formulate marketing strategies and product offerings.



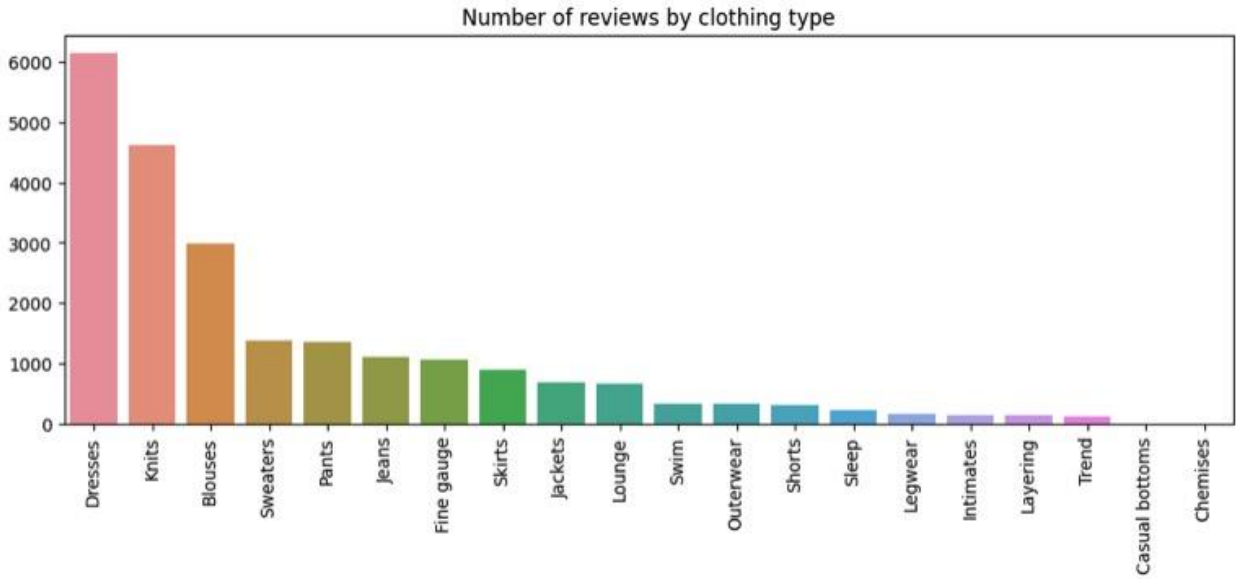
Most Reviewed Clothing Items

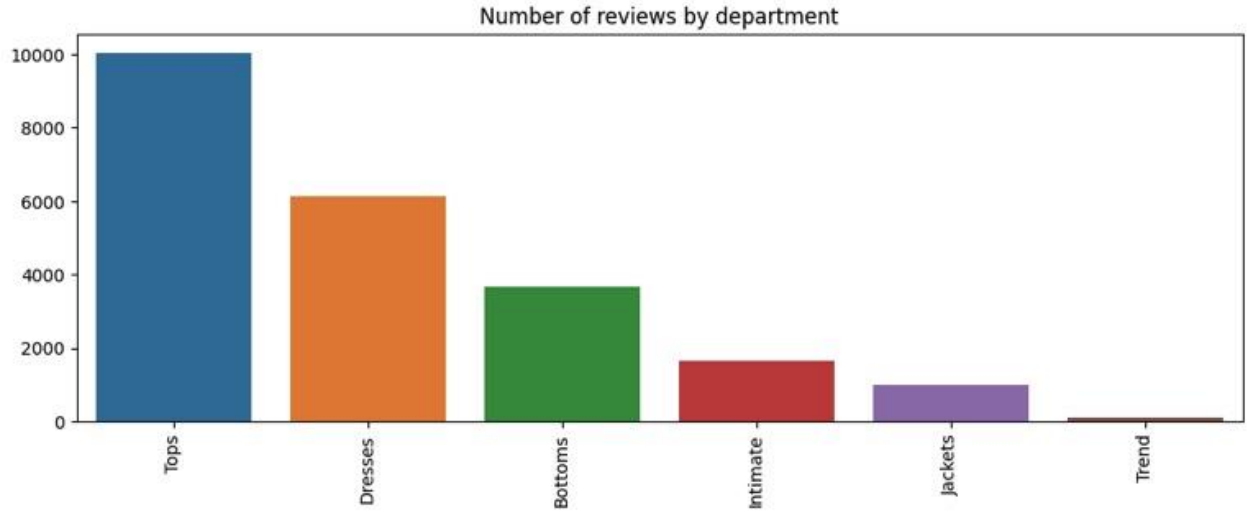
We then proceeded to determine the 5 most reviewed items, to identify the most popular products in our dataset that the customers seem to discuss or review the most. The popularity of those items could be due to factors which may not be included in our dataset, but again item popularity is quite important in the business context for marketing as well as strategic efforts toward inventory planning.



Reviews by Clothing Type and Department

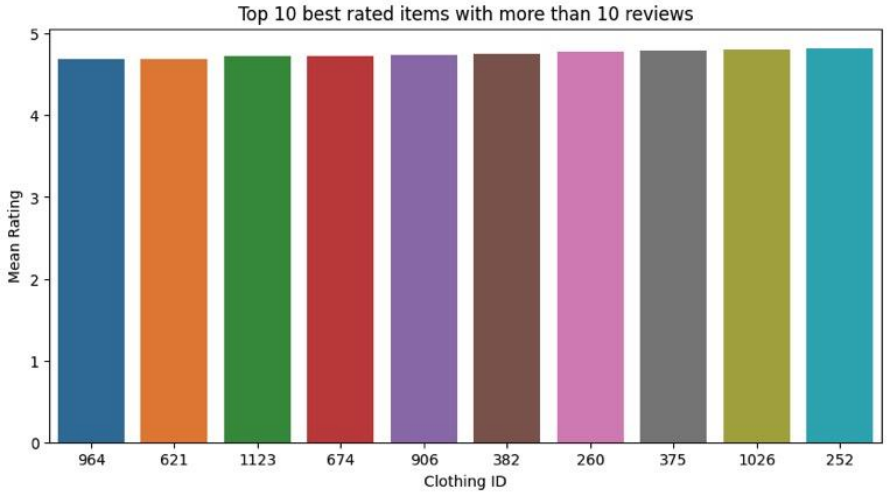
Next, we utilized visualizations of the distribution of reviews w.r.t. clothing type and department. This analysis aims to understand and inform us in a more detailed manner in regard of customer engagement behavior. Additionally, we see 6 departments identified but closer individual scrutiny of reviews corresponding to the 'Trend' department reveals that this category refers to clothing item types already included within the other 5 departments. Thus, there are only 5 relevant departments. The Trend category may possibly refer to reviews that trended in terms of views or other features not included in the dataset as these reviews also include those with low no. of positive feedbacks.





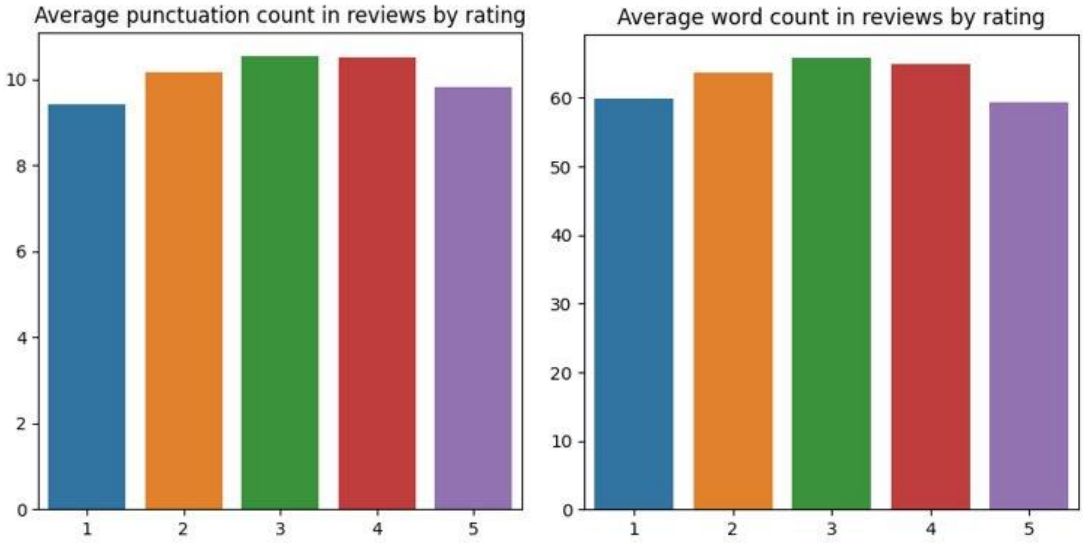
Best-rated Products

In our python code snippet, we also examine the products that are best rated but at the same time that have more than ten reviews. Through this step we aim to understand which products are more successful based on customer rating.



Word and Punctuation Count in Reviews

Last within the EDA section, we investigated the average word and punctuation count in the reviews, sorted descending by their rating. This may provide insights on whether the reviewers who are submitting a high or low rating, tend to write longer and more complex reviews. Through this visualization, we aim toward understanding whether a longer and more detailed review may also be more negative and complaint oriented. In our dataset however, we find that word count and punctuation are both slightly higher for neutral and 4-star ratings.



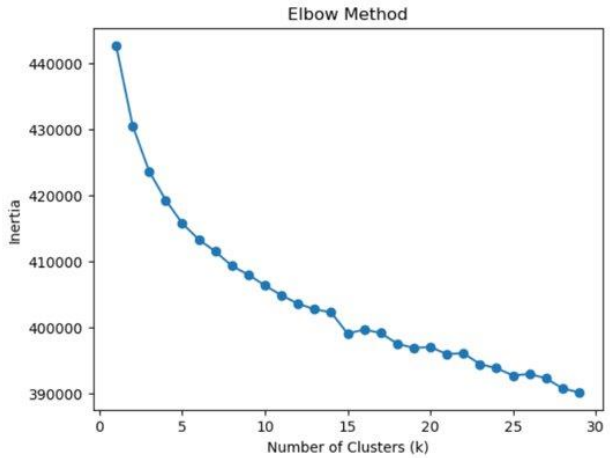
This is unexpected as there is a tendency to assume longer and more expressive reviews may be negative rather than positive.

Exploring the Data Structure in Our Dataset (Clustering)

K-means Clustering is a commonly used unsupervised machine learning algorithm we have used for clustering or partitioning our data into distinct groups or clusters, based on feature similarities. The no. of clusters (K) may be predetermined, and the algorithm accordingly randomly initializes K points in feature space referred to as centroids, which represent the initial cluster centers. Data points are assigned to these clusters using common metrics ex. Euclidean distance to calculate a point's distance to each of the K centroids and to assign it to the nearest centroid's cluster.

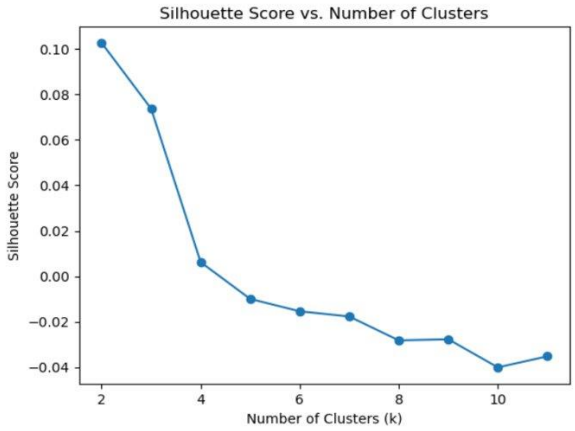
The Elbow Method: is well-established graphical technique is utilized to determine the optimal number of clusters in a clustering algorithm, such as K-means and also to identify the point at which increasing clusters may not significantly improve the model's performance.
Steps in plotting the same include.

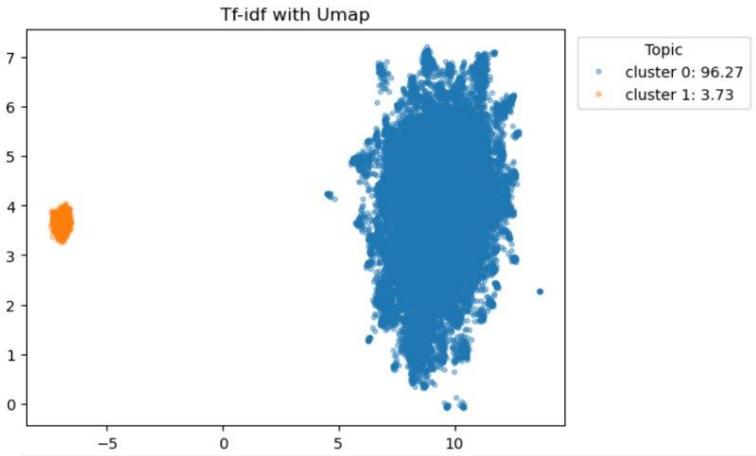
- Running the K-means clustering algorithm
- Calculating the inertia (measure of the data points spacing within a cluster from the cluster's centroid i.e., the sum of squared distances (SSE (Sum of Squared Errors)) of each data point to its assigned cluster center Plotting the inertia values vs. the number of clusters.



Identifying the "elbow" point where the plot starts to level off indicating the optimal no. of clusters. Our plot does not have a clear elbow, which is a common occurrence.

Silhouette Score: is the next tool we utilize to determine the optimal number of clusters, where typically the highest silhouette score or a clear peak in the silhouette score plot indicates the optimal number of clusters. The silhouette score is the difference between the average distance of a data point to all data points in the nearest neighboring cluster and the average distance of a data point to all other data points within the same cluster, normalized to the range [-1. 1], The score is plotted vs the no. of clusters. Here, the negative scores indicate wrong clustering and the highest score being 0.10 (2 clusters) and close to zero also indicates poor clustering. We attempted to extract these poor clusters to gain a clearer understanding as to whether they signify any relevant aspect of feature in the data.





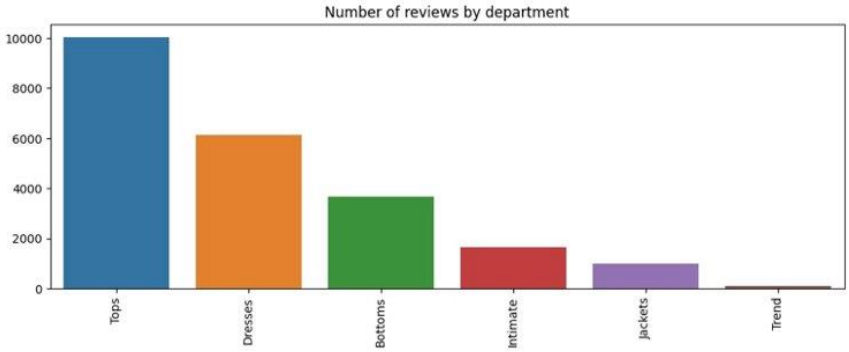
Two clear clusters are observed when utilizing TF-IDF embeddings, which are reduced in dimension by coupling with the UMAP (Uniform Manifold Approximation and Projection) algorithm and then clustering with the K-Means algorithm. Principle Component Analysis and t-SNE (t-Distributed

```
Cluster #1:
['nan', 'zuma', 'flight', 'flexors', 'flexibly', 'flexible', 'flexibility', 'flex', 'fleshy', 'flesh', 'fleetwoods', 'fleetwoo
d', 'fleecy', 'fleece', 'flecks', 'flecking', 'flecked', 'flea', 'flax', 'flawy', 'flaws', 'flawlessly', 'flawless', 'flies',
'flights', 'flops', 'flimsy', 'flop', 'floored', 'floor', 'floofs', 'flood', 'flocked', 'floaty', 'floats', 'floating', 'floa
t', 'flo', 'flirty', 'flirtiness', 'flirtier', 'flirt', 'flips', 'flipped', 'flipp', 'flip', 'flawed', 'flaw', 'flavor', 'flapp
ing']

Cluster #2:
['dress', 'love', 'size', 'top', 'great', 'fit', 'like', 'wear', 'just', 'small', 'color', 'fabric', 'perfect', 'really', 'litt
le', 'flattering', 'cute', 'comfortable', 'ordered', 'look', 'soft', 'will', 'beautiful', 'fits', 'well', 'can', 'one', 'nice',
'shirt', 'bought', 'back', 'looks', 'material', 'large', 'sweater', 'jeans', 'bit', 'much', 'colors', 'got', 'length', 'qualit
y', 'long', 'pretty', 'also', 'medium', 'petite', 'work', 'waist', 'xs']
```

Stochastic Neighbor Embedding approaches did not yield any separate clusters (Please see notebook for Data Structure) Word clusters obtained by applying kmeans =2 do not provide any meaningful insights

Topic Extraction

Topic extraction is an essential text analysis technique that is used to group or cluster text data based on their themes or “topics”. For identifying the main themes of the reviews and extracting hidden “abstract” topics from text data, we proceeded with applying topic modeling techniques. In our script, we used two different methods or modeling algorithms, the Latent Dirichlet Allocation (LDA) and the Non-Negative Matrix Factorization (NMF) (Karl et al., 2015). From both models used, we extracted the top 10 topics to help us identify common themes that the customers may be discussing in their reviews. This process provides a general understanding of the primary themes that customers were tending to discuss in their reviews. Topic extraction may allow a business to not address common issues, capitalize on widespread praises or perhaps even understand most popular products or departments.



Since clustering methods did not yield any clear indication for no. of topics, we resort to using a domain-knowledge approach. We recall the department divisions from EDA performed earlier. ‘Manual’ scrutiny of reviews corresponding to the Trend’ department or division indicate that this is not a legitimately separate department but that items included in this division are from all the other departments. It is possible that ‘Trend’ may be more of a category representing items that got the most views or utilizing features not included in our dataset.

We proceed to consider the remaining departments 5 in all, to be a possible guide toward number of topics that may be present in the customer reviews.

Topic Extraction with Latent Dirichlet Allocation (LDA)-5 topics

The LDA model assumes that in each review in our dataset, may be a mixture of a of topics, each word in that specific review is attributable to one of the general topics and each topic is characterized by a distribution of words. (Jelodar et al., 2018). The model assumes that there are underlying, latent topics that influence the usage of words within a review. LDA infers these latent topics and their associated word distributions from the dataset corpus. LDA is a probabilistic method, which means that the output may differ throughout different executions of the code.

```
Topic #0: usual blouse sheer thing bra washed fabric trying suit wash

Topic #1: arrived money product hole cheap retailer disappointed hopes package taking

Topic #2: sweater sleeves warm pockets long soft weight wear coat like

Topic #3: love great color wear fit jeans comfortable perfect soft bought

Topic #4: dress size like fit small just fabric ordered love wear
```

These 5 topics can be clearly matched to all 5 Tops, Intimates, Jackets, Bottoms and Dress departments.

Topic Extraction with NMF (Non-Negative Matrix Factorization)-5 topics

NMF model performs dimensionality reduction and works better when the matrix to factorize (tf-idf matrix, in our specific case) has no negative elements. NMF has an advantage over probability-based techniques like LDA since it can be used together

with the 'tf-idf' model, which often results in a better performance. Finally, another difference of NMF is it is deterministic, meaning that it will provide the same output for the same input independently of how many times we executed our code snippet.

For dimensionality reduction, the general idea is to factorize the given matrix into two non-negative matrices, capturing patterns in the original one. To create a review-term matrix, we use the TF-IDF vectorizer from the library of Scikit-learn (Nugumanova et al., 2022). That way we converted the text data into a matrix of TF-IDF features, assigning importance to words that are frequent in the review column.

```
Extracting Topics with NMF...

Topic #0: size small large medium ordered runs usually fit wear true

Topic #1: dress beautiful flattering love perfect slip wear comfortable dresses summer

Topic #2: love great jeans comfortable perfect soft wear color pants sweater

Topic #3: like just really fabric look nice good didn color fit

Topic #4: shirt cute super white little material boxy really underneath soft
```

Here the topic matching is less clear with Dress, Bottoms and Tops departments clearly identified as topics.

Text Summarization

Text Summarization is an NLP technique that can shorten long pieces of information and at the same time maintain its original context, meaning and information. So, for example if a review is lengthy, reading through all of them might be impractical and not so time- and cost-effective for a business, so that is where a summarization tool comes in handy, by condensing these reviews and offering a brief gist of the customer's opinion, enabling business to quickly understand the main points of each one.

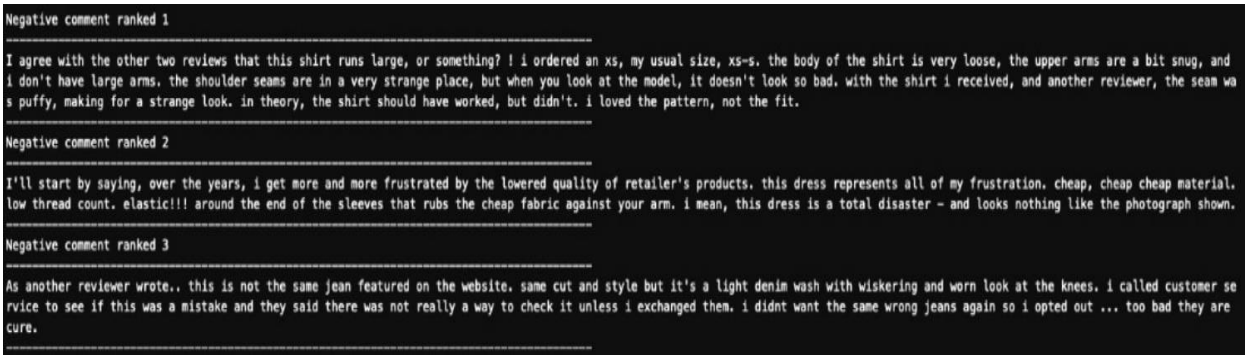
```
Summarizing text with BART...

Original: I had such high hopes for this dress and really wanted it to work for me.
i initially ordered the petite small (my usual size) but i found this to be outrage
ously small. so small in fact that i could not zip it up! i reordered it in petite
medium, which was just ok. overall, the top half was comfortable and fit nicely, bu
t the bottom half had a very tight under layer and several somewhat cheap (net) ove
r layers. imo, a major design flaw was the net over layer sewn directly into the zi
pper - it c

Summary: </s><s>The top half was comfortable and fit nicely, but the bottom half ha
d a very tight under layer and several somewhat cheap (net) over layers. imo, a maj
or design flaw was the net over layer sewn directly into the zipper.</s>
```

In order to generate a summarized version of the review texts, we used the BART transformer-based model (Bidirectional and Auto-Regressive Transformers), obtained from the Hugging Face's transformers library which provides numerous pre-trained models used for NLP tasks and it is developed by Facebook (Chen & Song, 2021). This model is a denoising autoencoder, meaning that it is trained to reconstruct original text after "noise" integration. It is also fine-tuned for downstream and generation tasks, such like text summarization, where you can prompt it a sample text and then the summarized version is displayed.

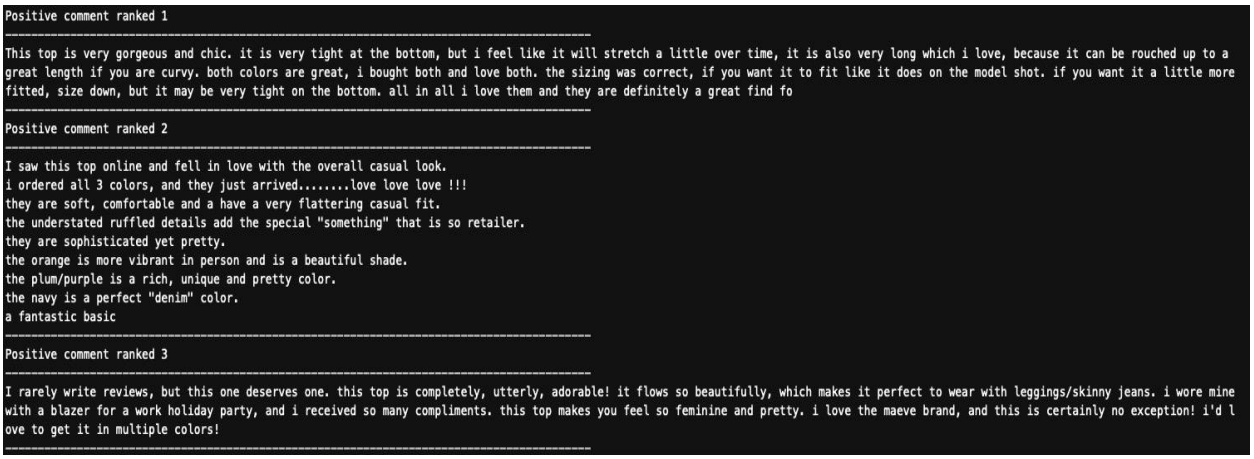
Sentiment Analysis

Businesses and other entities utilize sentiment of feedback and comments to understand their current public image/reputation/ perceived reliability etc. and to adjust their sales strategies accordingly. In our project, this analysis is a route to quantitatively measure and compare customer sentiment.

Exploring Sentiment within the Dataset



We utilized VADER (**V**alence **A**ware **D**ictionary and s**E**ntiment **R**easoner) which is a lexicon and rule-based sentiment analysis modelling tool that is considered to be resource-efficient, to assign sentiment /polarity scores to the reviews (Hutto & Gilbert, 2014).The Vader sentiment intensity analyzer, which is part of the NLTK library, is a popular tool known to perform effectively though like other tools and models may not detect complex sarcasm due it containing opposing sentiment. The analyzer assigns a 'compound "score between -1 and +1, where the - indicates the negative sentiment, close to 0 indicates neutral sentiment, and + indicates positive sentiment (Medhat et al., 2014).



So, it categorizes them into the 3 aforementioned categories, assigning them a new rating based on their sentiment score. We proceeded to identify reviews with the highest positive and negative scores.

Sentiment Prediction

Besides understanding the sentiment reflected in reviews already documented, we may use these to further predict the sentiment of a fresh review, comment on social media, email or tweet. An automated system may be created to categorize new or unseen reviews, based on their sentiment.

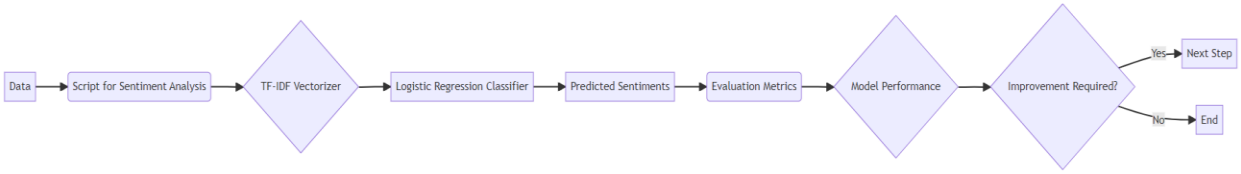
In this part of our analysis, we evaluate appropriate machine learning models to predict the sentiment of a review.

An initial precursory multi-class analysis with a Naive-Bayes algorithm performed poorly. so we proceeded to explore a more streamlined approach with reduced 3-class modeling and simultaneously attempting an alternative for 5-class modelling approach.

Logical Regression

Logical regression is a resource-efficient algorithm that can be used for multi-class modelling and text feature extraction. We used a combination of the Logistic Regression classifier with TF-IDF (Term Frequency-Inverse Document Frequency) vectorizer.

To reduce the 5-classes to 3, we used the already processed data frame reclassified by VADER. From VADER: compound score > 0.05 is classified as 'pos'/positive, compound score < -0.05 is assigned as 'neg'/negative class. The remaining range is classified as 'neutral'/neutral lass.



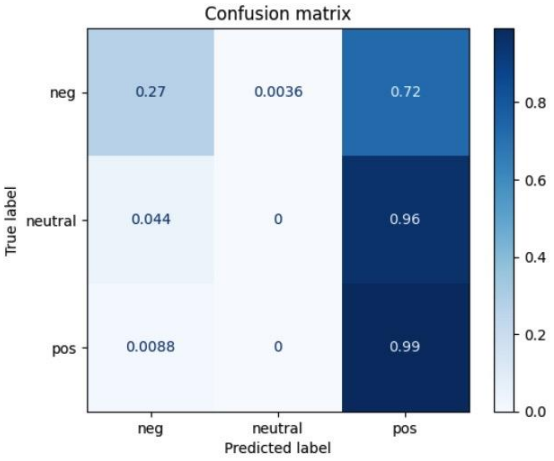
The TF-IDF vectorizer converts the text into a matrix of TF-IDF features, and then assigns ‘importance’ to words that are most frequent in a specific text piece but not across the entire data. Also, for hyper parameter tuning of the TfidfVectorizer we used the GridSearchCV technique, as to identify the best values. Through this process we determine appropriate weightages, which may often provide a better performance in tasks like text classification (Ramadhan et al., 2017).

The Logistic Regression model is ideal in our machine learning pipeline not only for its simplicity and effectiveness but also has the capability to adequately handle sparse data like that derived from TF-IDF. (As mentioned earlier scores from VADER are utilized.)

We proceeded to evaluate key metrics with generation of the confusion matrix, classification report, accuracy score, precision, recall and F1 score. This provided a clear read of identified True Positives, False Positives, True Negatives and the False negatives for all the three sentiment categories (negative, neutral, and positive). Since the metrics for the neutral class were poor, we proceeded to evaluate further.

Classification Report:

	precision	recall	f1-score	support
neg	0.66	0.27	0.38	276
neutral	0.00	0.00	0.00	45
pos	0.94	0.99	0.97	4208
accuracy			0.94	4529
macro avg	0.53	0.42	0.45	4529
weighted avg	0.92	0.94	0.92	4529



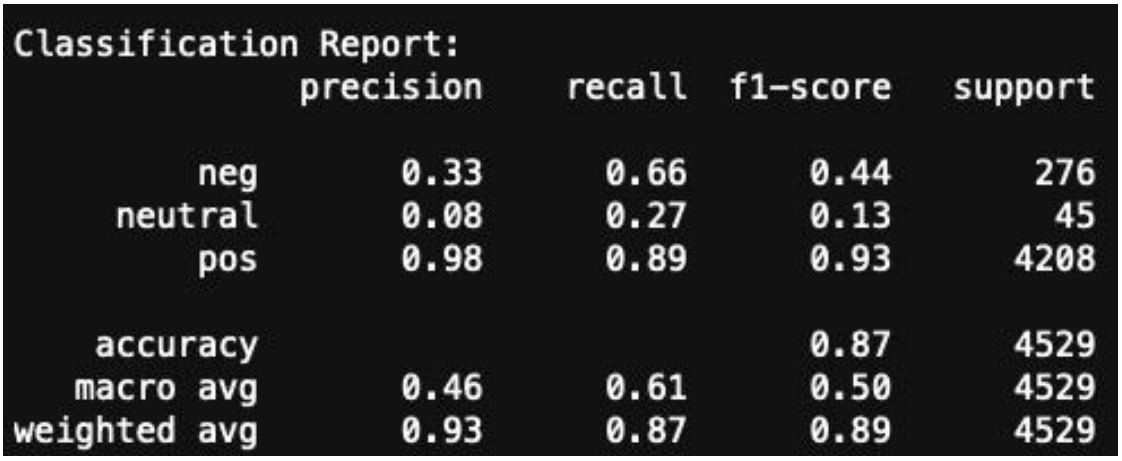
IMBALANCE FITTING for Logistical Regression

So, as we observed from the Linear Regression results above, there were significantly more instances of positive reviews as compared to the negative ones, in our dataset. So, our model tended to be biased. To remedy this bias, we utilize random oversampling in combination with text pre-processing TF-IDF and logistic regression for multi-classification modeling . More specifically, we proceeded with up-sampling the minority class. This technique adds copies of instances of the minority class in the training dataset, increasing the instances of that specific class (Barr et al., 2022). We once again generated the confusion matrix,

Coding steps :

- We started by importing all the necessary libraries for this part.
  - TfidfVectorizer, obtained from sklearn.feature_extraction.text. This as aforementioned is a numerical statistical tool used in reflect the importance that a word has to a document in a collection.
  - RandomOverSampler, from imblearn.over_sampling. This technique was used in order to handle the imbalanced in our ecommerce dataset. We worked with up sampling method in order to enhance the strength of our minority class.
  - Counter, from collections. A dictionary subclass used for counting hash able objects. In other words, it is a collection where elements are stored as the keys of the dictionary and their counts are stored as the values of that dictionary.
- Then we proceeded with initializing the TfidfVectorizer. We initialized it with a maximum of 5000 features (or unique words).

- Then we fitted and transformed the TfidfVectorizer on our training ecommerce dataset. The specific function used learns the global term weights of the idf vector through the fit step, and then it proceeds with applying that into the transform step. In the last step it also applies scaling of sublinear tf type, to be more specific it goes forward and replace tf with 1 + log(tf).
- Then using the same vectorizer we transformed our test data into TF-IDF features. But here, we did not use the fit_transform function because we choosed to use the idf that was learned from the training data set.
- Here is where the RandomOverSampler makes its appearance. With upscaling or up sampling our negative reviews minority class, we tried to eliminate the bias of our model as much as possible and that way not to harm the accuracy of the model's prediction about our minority class. This sampler achieves that by making the representation of the minority class higher in our dataset.
- Then we proceeded with fitting again a Logistic Regression Model on the sampled data. We chose as solver parameter the liblinear one, because when it comes to binary classification and especially for small datasets, it is a considered a good one. Then for the class_weight parameter we used the balanced one, which was because we wanted to automatically adjust our weights, but in inverse of the proportion of our class frequencies.
- Then for the prediction and the evaluation part we used the now fitted logistic regression model. By using accuracy_score() function we printed the respected accuracy of the predictions of our model.

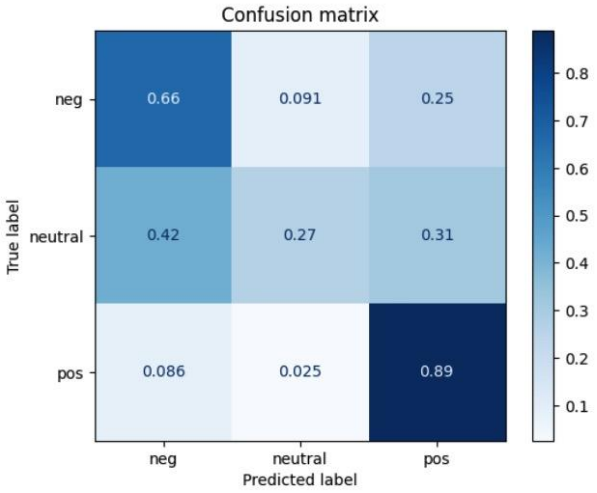


Classification Report:

	precision	recall	f1-score	support
neg	0.33	0.66	0.44	276
neutral	0.08	0.27	0.13	45
pos	0.98	0.89	0.93	4208
accuracy			0.87	4529
macro avg	0.46	0.61	0.50	4529
weighted avg	0.93	0.87	0.89	4529

- Further on we print the classification report in which we displayed the following:
  - Recall score, which is the ability of the classifier to locate all the samples that are positive.
  - The precision, which is the ability of our classifier not to label a negative sample as positive.
  - The F1 score for our model, which is the harmonic mean of recall but also precision.
- Finally, we print out the Confusion Matrix, which is a table used for performance description reasons of a classification models on a specific dataset, in which we know it's true values. We plotted the table through a function that we have named

as plot_confusion_matrix and it revealed in which aspect was our classification model confused.




```
Enter text: Horrible fir, will return
Predicted sentiment: neg

Enter text: mediocre but like the color
Predicted sentiment: pos

Enter text: Not sure, bought 2 items, not very impressive
Predicted sentiment: neg

Enter text: Love it. So perfect.
Predicted sentiment: pos
```

```
Enter text: As another reviewer wrote.. this is not the same jean featured on the website. same cut and style but it's a light
denim wash with wiskering and worn look at the knees. i called customer service to see if this was a mistake and they said ther
e was not really a way to check it unless i exchanged them. i didnt want the same wrong jeans again so i opted out ... too bad
they are cure.
Predicted sentiment: neg
```

Alternative Approach: BERT (Bidirectional Encoder Representations from Transformers) + Feed-Forward Multilayer Perceptron (MLP) Network

The above approach with logistic regression performs well for both positive and negative reviews but even with oversampling the neutral reviews are not evaluated as well. In our alternative approach, we attempt to have not only better evaluation for the neutral class but also overall good evaluation of all 5 original classes in the dataset.

Some key differences in processing besides the modelling itself:

- The Title text wherever added by customers is appended with the Review Text, with the assumption that customers with stronger sentiment may spend extra time adding a separate title.
- The Bert Tokenizer is utilized in processing the text.

Processing

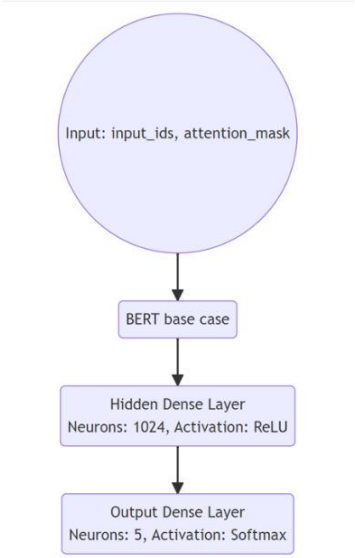
Sequence length may be preset at 512 or finetuned as per longest token sequence or maximum length of reviews which is 126 tokens as determined approximately using RegexpTokenizer. Additional libraries, including TensorFlow and the BERT tokenizer and model from the Transformers library are required.

This alternative approach uses a slightly altered dataframe with the Clothing ID, Rating and Concat (Review Text +Title text if available). Also the tokenizer is instantiated from

the bert-base-case model(12-layer, 768-hidden, 12-heads, 110M parameters.) made accessible by Hugging Face.

Variables are then initialized for tokenized text and attention masks. The algorithm Iterates over each review in the dataframe and encodes it with the tokenizer, while truncating or padding it to the specified sequence length. One-hot encoded labels are generated for the ratings for all 5 classes /star -ratings. A TensorFlow dataset object is created from the input data and labels. This is followed by define a mapping function to format the tensorflow dataset object which is also shuffled and batched, to avoid biases and then slit into training and validation sets.

Model Architecture



The BERT base model is instantiated to load the pre-trained BERT model with a cased vocabulary, which is suitable when there is a requirement for case sensitivity.

Inputs corresponding to the tokenized text and attention mask are passed through the BERT base model using a function to compute embeddings, returning a tuple of the generated embeddings and the pooled output. We opt to utilize the pooled output as a feed into a dense layer with 'hidden' neurons. As per approaches further described, the number of neurons may be fine-tuned to improve metrics and as per number of samples (initial number was set to 1024). A ReLU (rectified linear activation function) activation which is a default activation function for most neural networks is added, which passes the input directly if positive, otherwise, passing zero.

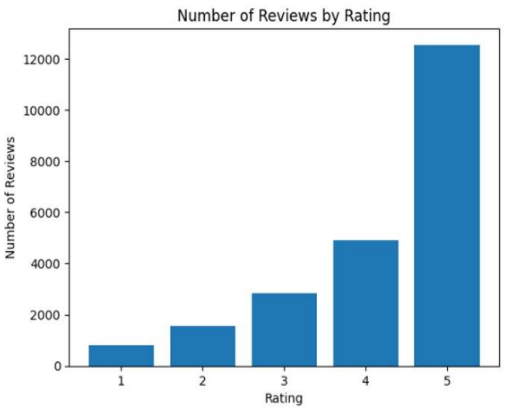
A dense output layer with 5 units/neurons and a softmax activation function maps the intermediate representations to the multi-class sentiment labels. The number of units in the output layer (5) corresponds to the number of sentiment classes in the dataset.

Classification Report:

	precision	recall	f1-score	support
0	0.04	0.06	0.04	84
1	0.05	0.04	0.05	157
2	0.13	0.15	0.14	295
3	0.25	0.20	0.22	493
4	0.55	0.56	0.56	1243
accuracy			0.38	2272
macro avg	0.20	0.20	0.20	2272
weighted avg	0.38	0.38	0.38	2272

The model is complied with an optimizer, loss function, and accuracy metric. A checkpoint callback is added to save the model during training (whole model is saved to *.h5). The algorithm is adjusted to handle keyboard interrupts to halt training, save the model and reload the partially trained model for further training as per convenience. Due to limited configuration the algorithm uses only CPU (16 GB RAM)

In the *first approach* where excessive neurons :1024 were used, without bias handling we observed poor precision and recall, with worse metrics for negative and neutral ratings. Please note the classes 0,1,2,3,4 correspond to 1,2,3,4,5-star ratings respectively. (Number of samples:1273)



Rating-wise Distribution of Reviews

Accordingly, we investigate in detail the distribution of the star-ratings and confirm the strong positive bias in our dataset. Since the web-scraping details are not clearly documented by the dataset compiler, we are not able to determine the reason for this bias.

In the *second approach,* we use ***under-sampling*** to handle the positive bias. The neurons remain at 1024. The overall metrics improve though still poor vs the 3-class linear regression approach.

(Number of samples:230).

Classification Report:

	precision	recall	f1-score	support
0	0.25	0.23	0.24	87
1	0.19	0.21	0.20	75
2	0.27	0.24	0.25	92
3	0.20	0.24	0.22	79
4	0.23	0.20	0.22	83
accuracy			0.23	416
macro avg	0.23	0.23	0.23	416
weighted avg	0.23	0.23	0.23	416

```
Enter text: no
1/1 [==============================] - 5s 5s/step
Predicted class: 1

Enter text: Horrible fir, will return
1/1 [==============================] - 0s 440ms/step
Predicted class: 1

Enter text: Love it. So perfect
1/1 [==============================] - 0s 426ms/step
Predicted class: 5

Enter text: Not sure, bought 2 items, not very impressive
1/1 [==============================] - 0s 410ms/step
Predicted class: 2

Enter text: mediocre but like the color
1/1 [==============================] - 0s 428ms/step
Predicted class: 3

Enter text: As another reviewer wrote.. this is not the same jean featured on the website. same cut and style but it's a light
denim wash with wiskering and worn look at the knees. i called customer service to see if this was a mistake and they said ther
e was not really a way to check it unless i exchanged them. i didnt want the same wrong jeans again so i opted out ... too bad
they are cure.
1/1 [==============================] - 0s 445ms/step
Predicted class: 2

Enter text:
```

It is notable that in spite of poor performance metrics , the prediction works well with short user inputs and entering reviews from the dataset as well.

We adjust the second approach (2A), reducing the neurons to 256 and reducing batch size to compute more efficiently. Recall slightly improves for 5 star ratings but overall metrics are poor. (Number of samples:461)

Classification Report:

	precision	recall	f1-score	support
0	0.20	0.17	0.18	88
1	0.26	0.21	0.23	91
2	0.10	0.15	0.12	60
3	0.26	0.23	0.24	92
4	0.25	0.31	0.28	85
accuracy			0.22	416
macro avg	0.22	0.21	0.21	416
weighted avg	0.22	0.22	0.22	416

We further adjust the second approach (2B), with very few i.e. 16 neurons a slightly increased batch size : 12 Recall and precision for 3 star-rating increases at the cost of peromance for 5-star ratings. Poor metrics. (307 training samples.)

Classification Report:

	precision	recall	f1-score	support
0	0.17	0.20	0.18	74
1	0.27	0.26	0.26	92
2	0.30	0.28	0.29	89
3	0.19	0.16	0.18	85
4	0.19	0.20	0.19	80
accuracy			0.22	420
macro avg	0.22	0.22	0.22	420
weighted avg	0.23	0.22	0.22	420

A final more balanced tuning for second approach with undersampling (2C) with 64 neurons , Batch size : 12 (307 training samples) , yields poor metrics again with improved recall for more negative ratings (1-star) and moderately positve (4-star) rated review.

Classification Report:

	precision	recall	f1-score	support
0	0.23	0.25	0.24	85
1	0.16	0.15	0.15	79
2	0.22	0.22	0.22	78
3	0.24	0.24	0.24	90
4	0.18	0.17	0.18	88
accuracy			0.21	420
macro avg	0.21	0.21	0.21	420
weighted avg	0.21	0.21	0.21	420

The *third approach* utilizes **over-sampling** 64 neurons, Batch size : 12 (4702 training samples) requires excessive computing time per epoch viz. ~ 50 hours on the low config. setup. So it is tuned for a smaller batch (number of samples-1175), which yields poor performance metrics again.

Classification Report:

	precision	recall	f1-score	support
0	0.26	0.28	0.27	403
1	0.19	0.18	0.19	293
2	0.19	0.19	0.19	286
3	0.20	0.20	0.20	307
4	0.19	0.18	0.18	283
accuracy			0.21	1572
macro avg	0.20	0.20	0.20	1572
weighted avg	0.21	0.21	0.21	1572

The alternative approach while yielding somewhat improved metrics for middling/close-to-neutral sentiment than our 3-class approach, performs poorly overall. This is in agreement with similar attempts at 5-class prediction such as Rao.P. (2019). Clearly successful attempts with similar architecture are observed with 3-class (reduced) approaches such as M.P. Geetha, D. Karthika Renuka (2021). Our approach modelled sentiment entirely from the dataset and using a pretrained model to generate contextual embeddings. An improved approach requiring better computing resources would be to use a pretrained sentiment model such as nlptown/bert-base-multilingual-uncased-sentiment also distributed via HuggingFace to fine-tune on our dataset. However, this approach may require a powerful discrete GPU. A preliminary attempt on our low config. system resulted in out-of-memory state.

Querying with Generative AI

FLAN MODEL

Then in our notebook we included a question-answering model, that uses the transformer model named FLAN T5 base. The model either takes a code snippet and a question as input or you can directly accept questions and then moves on into generating an answer (Varshney et al., 2023). Not only for the purpose of research, but also for practical implementation, this could be very handy in various applications where is required to understand and answer questions based on text data. After experimenting and trying the small version also, we ended up going with the base model and not the medium one, because of the large storage that the model required and the heavy computational power that it asked for. In that specific code block, we implemented the aforementioned model in order to answer questions based rather on a given text or without using an input from us, and then through the transformer's models in the Hugging Face library come up with

an answer. According to Varshney et al. (2023) here is a more detailed step by step breakdown in what we exactly did:

- Imported all the libraries that we needed for the FLAN model to run. That libraries were the AutoModelForSeq2SeqLM and the AutoTokenizer from the transformer's library.
- Then we instantiated the model with the following command: AutoModelForSeq2SeqLM.from_pretrained("google/flan-t5-base")
- Respectively the tokenizer with the following command: AutoTokenizer.from_pretrained("google/flan-t5-base")
- So, the "google/flan-t5-base" is loaded from the Hugging Face's model repository, and we proceeded with the choice of the base one due to storage and computational issues. The bigger models that the repository had required a lot of disk space and more computational power that we could afford in order to see results in "real-time".
- Then we moved forward defining a function with the name answer_question, which is responsible for taking reading a text snippet that we provide according to our liking and also a question as input. So, based on the text we provided the model with the goal is to answer the question. The flow of the whole process that is taking place is the following:
    - The whole input of the model is firstly created, through the combination of our text input and the question that we have set for the model.
    - Then through instantiated tokenizer, our whole input package is tokenized. Then this package is truncated to the specific length of 512 tokens, which is the limit that the FLAT base model gave us, for smaller models the tokens are fewer and for larger respectively bigger.
    - Then all that results that are now tokenized are fed into the model. Then for the generation the model uses beam search, and more specifically with a width of 5 for decoding. This is a heuristic searching algorithm that is used for exploration of the most promising nodes of the input package. At each step it tries to reduce the risk of not getting the optimal path, by keeping track of the best most promising ones.
    - Furthermore, by the use of the tokenizer the model decodes back into text its output, which is a sequence of tokens ids.
    - Finally, the answer that has been decoded in printed in a conversational manner/format.
- To sum up, the last step that we did is to use the answer_question function to generate/print the answer to the question that we have fed the model with.

This is an example on our input text and asking the model to answer on that:

```
code_snippet = """
def calculate_area(length, width):
    return length * width
"""
question = "What does the calculate_area function do?"
answer_question(code_snippet, question)
```

```
Based on the text, it appears that return length * width
```

This is an example of just questioning without our input, and just asking of the FLAN to answer:

```
# Use the function to answer a question
question = "What is the name of the sea around Greece?"
answer_question2(question)
```

```
Answer: aegean sea
```

```
# Use the function to answer a question
question = "How are you doing today?"
answer_question2(question)
```

```
Answer: I'm fine.
```

```
# Use the function to answer a question
question = "Who was president of U.S that got assassinated?"
answer_question2(question)
```

```
Answer: john f kennedy
```

DataFrameChatBot

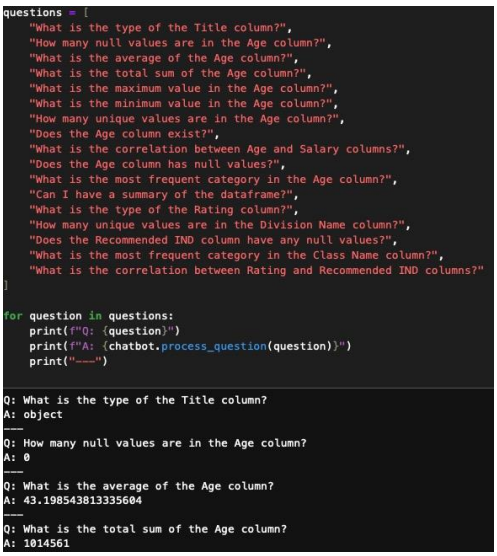
In the last part of our code script, we defined a Python class, named **DataFrameChatBot**, that acts as a chatbot interface in order to achieve some interaction with our panda's data frame, Women's Clothing E-commerce. This could be proved very useful for data analysis purposes where a user can get information about the data and its structure in a conversational manner and the same time offering the ease that in the case that the imported dataset changes then minor changes are needed for the code to be executed correctly on the new data. So, here is a summarization on how we separated that part of our code snippet into 3 main parts:

1. **Initialization (init):** It first reads and takes a pandas data frame as input, and then saves it as a class attribute. Then it loads the **en_core_web_sm** model from SpaCy, and then proceeds with assigning it to the class attribute named **nlp**. This is the model that it is going to be used for natural language processing tasks.
2. **Information Retrieval Methods:** In this class we defined several methods, in order to retrieve information from the our imported data frame, like the shape of the data frame, the columns that it consists of, the type of specific columns, the number of null values contained in a column, the mean or the sum or the max or the min calculated on specific columns, the number of the unique values that a column has, and many more. We also implemented methods in order to check

whether a column does exist in our dataset, if a column has null values in it, and finally to obtain the most frequent category of a column.

3. **Question Processing (process_question):** This is our core or main function if you like of our manual chatbot. It receives as an input a question and then uses the SpaCy model, in order to parse the question. Then depending on the words and phrases that the user concludes in the question, it calls the relevant method of combination of multiple one's, in order to retrieve the required information from the data frame.

Finally, using the process_question function that we have defined, we process the question, retrieve the answer from the chatbot and finally print it. This is kind of unique and extraordinary way for a user to get and understand the information that is needed in a unique way though interacting or speaking if we can say, directly with the data frame.



```
questions = [
    "What is the type of the Title column?",
    "How many null values are in the Age column?",
    "What is the average of the Age column?",
    "What is the total sum of the Age column?",
    "What is the maximum value in the Age column?",
    "What is the minimum value in the Age column?",
    "How many unique values are in the Age column?",
    "Does the Age column exist?",
    "What is the correlation between Age and Salary columns?",
    "Does the Age column has null values?",
    "What is the most frequent category in the Age column?",
    "Can I have a summary of the dataframe?",
    "What is the type of the Rating column?",
    "How many unique values are in the Division Name column?",
    "Does the Recommended IND column have any null values?",
    "What is the most frequent category in the Class Name column?",
    "What is the correlation between Rating and Recommended IND columns?"
]

for question in questions:
    print(f"Q: {question}")
    print(f"A: {chatbot.process_question(question)}")
    print("~~~")
```

```
Q: What is the type of the Title column?
A: object
~~~
Q: How many null values are in the Age column?
A: 0
~~~
Q: What is the average of the Age column?
A: 43.198543813335604
~~~
Q: What is the total sum of the Age column?
A: 1014561
```

Conclusion and Future Work

We formulated a complete NLP pipeline, as to understand specifically the customer reviews in the dataset that we are investigating. From the little baby steps of data cleaning and EDA, we systematically proceeded to utilize machine learning models and other deep learning tools, toward achieving the sentiment analysis, text summarization and topic extraction. Finally with the use of the FLAN model we queried the reviews to understand how Generative AI handles such data.

We successfully extracted 5 topics corresponding to the 5 departments of the company. The most efficient approach to model/predict sentiment was to logically reduce the 1 to 5 star rating to 3 classes positive, neutral, negative. Logical regression was successfully deployed and resource efficient. We understood the shortfalls and possible

requirements/solutions to pursue prediction with the original 5 classes. Text summarization was successfully achieved by deploying BART. Similarly, the FLAN model adequately handled the dataset. Our setup both allowed a data scientist to query metrics within the dataset as well as to query sentiment of summaries/highlights of review text.

Our pipeline gives a starting point of how a business may take advantage of customer reviews to extract meaning and actionable insights. They can better understand their customers, satisfy their needs, and drive growth, by interpreting key sentiments and summarizing key topics. The insights generated by this report and code snippets, may be used as a basic template for more informed strategic decision making.

With improved computational resources it will be possible to predict sentiment with increased reliability. Additional analysis may be planned such as utlilizing Large Language Models in an appropriate interface with a customer on social media platforms for example, and auto-categorize customer feedback, fresh reviews or comments while also adding to the current review dataset through appropriate database coupling.

Since our data has an added binary class of sentiment Recommend/Not Recommend, this may be harnessed to downstream specific marketing strategies to undecided/neutral customers. For example, if a customer provides comments that are then auto-rated 3 or neutral by customized small models or LLMs , the pipeline may be extended to further classify the entered text with a simple binary class classifier, utilizing this extra sentiment data to predict if the customer may recommend or not recommend and target them with discounts, advertisements to ‘flip’ their sentiment if required.

References

Brooks N., 2016 https://www.kaggle.com/datasets/nicapotato/womens-ecommerce-clothing-reviews

Amaratunga, D., Cabrera, J., Fernholz, L. T., & Morgenthaler, S. (2009). Exploratory data analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, *1*(1), 33–44. https://doi.org/10.1002/wics.2

Barr, J. T., Sobel, M., & Thatcher, T. (2022). *Upsampling, a comparative study with new ideas*. https://doi.org/10.1109/icsc52841.2022.00059

Chen, Y., & Song, Q. (2021). *News Text Summarization Method based on BART-TextRank Model*. https://doi.org/10.1109/iaeac50856.2021.9390683

Hutto, C. J., & Gilbert, E. (2014). Vader: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. *Proceedings of the International AAAI Conference on Web and Social Media*, *8*(1), 216–225. https://doi.org/10.1609/icwsm.v8i1.14550

Jelodar, H., Wang, Y., Yuan, C., Feng, X., Jiang, X., Li, Y., & Zhao, L. (2018). Latent Dirichlet allocation (LDA) and topic modeling: models, applications, a survey. *Multimedia Tools and Applications*, *78*(11), 15169–15211. https://doi.org/10.1007/s11042-018-6894-4

Karl, A. T., Wisnowski, J. W., & Rushing, W. H. (2015). A practical guide to text mining with topic extraction. *Wiley Interdisciplinary Reviews: Computational Statistics*, *7*(5), 326–340. https://doi.org/10.1002/wics.1361

Medhat, W., Hassan, A. E., & Korashy, H. (2014). Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal*, *5*(4), 1093–1113. https://doi.org/10.1016/j.asej.2014.04.011

Nugumanova, A., Akhmed-Zaki, D., Mansurova, M., Baiburin, Y., & Maulit, A. (2022). NMF-based approach to automatic term extraction. *Expert Systems With Applications*, *199*, 117179. https://doi.org/10.1016/j.eswa.2022.117179

Ramadhan, W. P., Novianty, S. T. M. T. A., & Setianingsih, S. T. M. T. C. (2017). *Sentiment analysis using multinomial logistic regression*. https://doi.org/10.1109/iccerec.2017.8226700

Varshney, N., Parmar, M., Patel, N., Handa, D., Sarkar, S., Luo, M., & Baral, C. (2023). Can NLP Models Correctly Reason Over Contexts that Break the Common Assumptions? *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2305.12096

Rao P. (2019) https://towardsdatascience.com/fine-grained-sentiment-analysis-part-3-fine-tuning-transformers-1ae6574f25a6

M.P. Geetha, D. Karthika Renuka (2021) Improving the performance of aspect based sentiment analysis using fine-tuned Bert Base Uncased model, International Journal of Intelligent Networks,Volume 2. https://doi.org/10.1016/j.ijin.2021.06.005.