Group

**29th February**

20



08

**Fall**

F21AA

Table of Contents

[Table of Contents 2](#_Toc33879819)

[Introduction 2](#_Toc33879820)

[Q1 Data Exploration and Visualization 2](#_Toc33879821)

[Q2 Text Processing and Normalization 3](#_Toc33879822)

[Q3 Vector space Model and feature representation 3](#_Toc33879823)

[Q4 Model training, selection and hyperparameter tuning and evaluation 3](#_Toc33879824)

[Q5 Topic Modelling of high and low ratings 3](#_Toc33879825)

[Conclusions 3](#_Toc33879826)

[Appendix 3](#_Toc33879827)

# Introduction

# Q1 Data Exploration and Visualization

For this task we began by exploring our training data set. It’s composed of 10 attributes and 426,340 examples.

The **first** **task** was getting aquatinted with our data types:

Data columns (total 10 columns):

Id 426340 non-null int64

ProductId 426340 non-null object

UserId 426340 non-null object

ProfileName 426326 non-null object

HelpfulnessNumerator 426340 non-null int64

HelpfulnessDenominator 426340 non-null int64

Score 426340 non-null int64

Time 426340 non-null int64

Summary 426320 non-null object

Text 426340 non-null object

As seen from above, we have 6 numerical attributes and 5 categorical or non-numeric attributes in total. Our class attribute is **Score.**

The **second task** was **statistical analysis of our data set** [appendix-stats table] An interesting insight that can be observed is that the 50% and 75% for score had the value of **5.0** which is the maximum score. We began to explore this further. [appendix-hist per column] As seen from the above plot. **Most of the data is biased towards the score 5.0** that means our class values are not **stratified** or evenly distributed. our next step was to explore this further; we began to analyse the data considering the score distributions. As seen from above [appendix-count group by score], **272492** examples are on the on the class value of 5.0. this over **50% of our data.** Our next step was split into 2 stages: numerical correlation and categorical correlation.

We first began by pair plotting our numerical attributes with our score to see if we can find any interesting relationships.

A screenshot of a cell phone

Description automatically generated

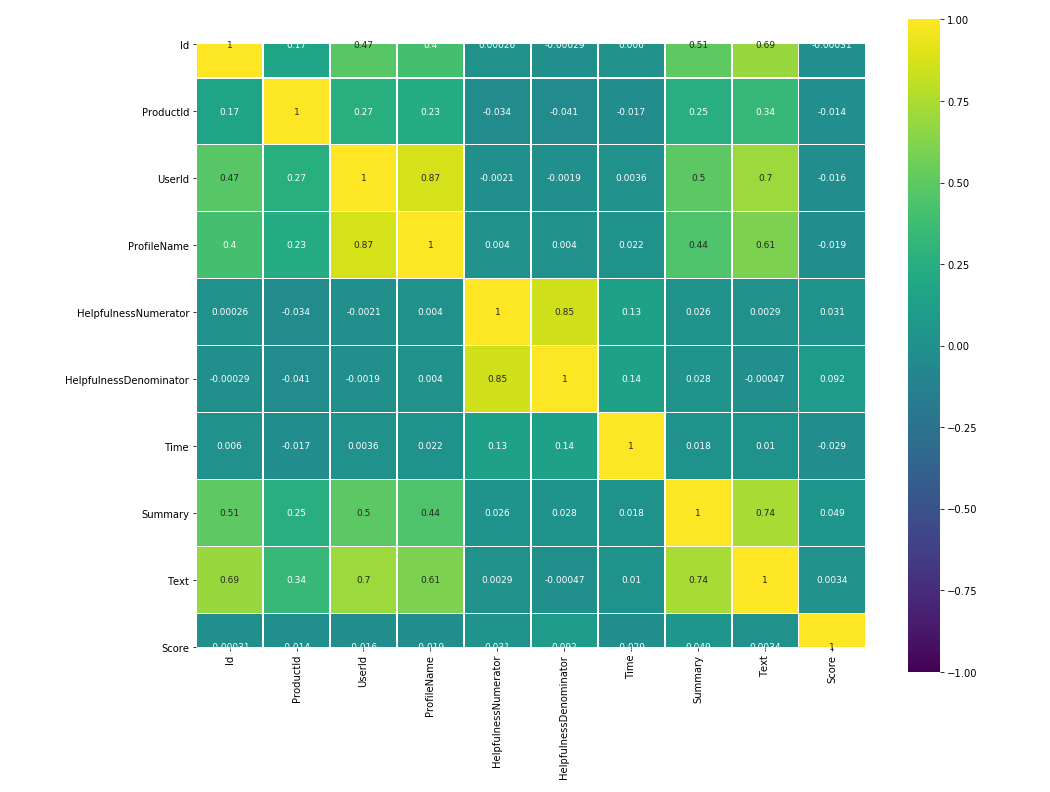
the above plots gave no indicative relationship, so we explored further. We began by calculating the correlation values of each of the numerical attributes with our class attribute [appendix-correlation scores-score] As seen from above, the correlation values confirm the plots that there is no distinct correlation with the class. We moved on to explore the non-numeric attributes in our data set. We factorized the categorical attributes to be able to calculate a correlation score. Furthermore, we first explored **inter-attribute** correlation.

A close up of text on a white background

Description automatically generated

The correlation matrix provided some interesting results **text is highly positively correlated with product name and summary while summary is highly positively correlated with text as well.**

From this we concluded that among the categorical attributes **text** and **summary** seem to be strong attributes. We further began to plot a full correlation matrix to verify our assumptions.



From the correlation matrix, we discovered that none of the attributes had a strong correlation to the class value. This could be for a variety of reasons so instead we diverted our focus to strong inter-correlations, and we found that the best candidates are **Text** and **Summary** attributes. We merged these attributes together to lay foundations for our text processing. In the next stage of our analysis we began by exploring the **values** in our data set. We first began inspection for null values. [appendix-null values] As seen from above we’ve found that there are null values in the **summary and profile name** attributes. Furthermore, we investigated if there are any duplicate values. [appendix-duplicate values]

As can be seen from above. We found duplicate reviews in the data set; those were removed to avoid noise in our data. We then shifted our focus to pre-processing of our text. After further analysis we found the following properties in our text attribute:

1. Html tags
2. Accents
3. Punctuation

These 3 were cleaned from the data before we can begin our pre-processing. Finally, the next stage in our pipeline was tokenization, lemmatization / normalization so that we may begin representing our textual data in a way for modelling.

# Q2 Text Processing and Normalization

We divided this task into 2 tasks,

The first one is **Text processing** ,we found *HTML tags* and *punctuations* that we removed from all the Corpus, we do believe that's HTML tags and punctuation's will not add any value to the model accuracy , on the contrary they will have a negative impact on the model accuracy and execution speed, we did that through a function we created called **tokenizer** that uses remove the HTML tags and the punctuation from the corpus [appendix-stemming/lemming figure].

The second phase was **Data** **Normalization** and we test using 2 techniques **stemming** and **lemmatization**.

Stemming in the Lemmatization are two common techniques used for text normalization where stemming cut the word to get a shorter description or representation of the word without looking into the lexical meaning of the word , lemmatization though try to do the same thing by shortening the word by getting the lexical meaning of the word.

In our assignment, we used **Port Stemmer** and **Worldnet Lemmatizer** from *NLTK* library.

Below you can find examples of corpus document after applying our processing and normalization functions

**#1 Original Corpus Document**

'BEST CHIPS and GLUTEN FREE! These chips are so good they are addictive! Extremely fresh and crispy. Even potato chips can contain gluten, so when I noticed Gluten Free marked on the bag, I had to give them a try. Now these are the only potato chips I will purchase--Thanks for making a GF product that rocks!!'

**#2 After applying tokenizer (HtmlTags and punctuation removal)**

'BEST CHIPS and GLUTEN FREE These chips are so good they are addictive Extremely fresh and crispy Even potato chips can contain gluten so when I noticed Gluten Free marked on the bag I had to give them a try Now these are the only potato chips I will purchase Thanks for making a GF product that rocks'

**#3 Applying Stemming**

'best chip and gluten free these chip are so good they are addict extrem fresh and crispi even potato chip can contain gluten so when I notic gluten free mark on the bag I had to give them a tri now these are the onli potato chip I will purchasethank for make a GF product that rock'

**#4 Applying Lemmatization**

'best chip and gluten free these chip be so good they be addictive extremely fresh and crispy even potato chip can contain gluten so when i notice gluten free mark on the bag i have to give them a try now these be the only potato chip i will purchase thanks for make a gf product that rock'

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Original | Tokenizer | Stemming | Lemmatization | Comments |
| addictive! | addictive | addict | addictive | Lemmatization didn’t do any changes while stemming was was shorter |
| noticed | noticed | notic | notice | Stemming gave a shorter token while lemmatization gave the correct lexical output |

We did notice a lot of pros and cons for each technique, stemming in many cases give a short representation while not preserving the word meaning, lemmatization in several cases gave a correct word however it was not acting on all words,

To prove which technique is better for our corpus in hand, We did a comparison using a Pipeline of CountVectorizer, TFIDF Transformer and then **LogisticRegression** , this was done without the use of Grid Search to fix all variables, we did however tokenize by removing *html tags* and *punctuation* with the same function on all 3 tests (we did restart the python kernel after each experiment to prevent model-retraining )

In our Experiment, the use of lemmatization and Stemming actually decreased our overall model score, slightly

The model Scored **0.761** without any Data Normalization

Scored **0.756** with Stemming

And scored **0.756** with Lemmatization

[appendix lemma and stemming scores]

# Q3 Vector space Model and feature representation

For this segment of the course work we aimed to experiment with **count-based** representations. For the count-based representations, we utilized **term frequency** and **inverse-document term frequency** to evaluate the relative importance of each word or feature.

We represented each review as a matrix of **token counts** through the process called Count Vectorization.

<https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html>

term frequency is simply accounting for the number of times a term (a word) occurs in each corpus (collection of documents) it does not differentiate between the context of the term or the document it is located in. A matrix of term frequencies is returned by the algorithm.

tf(*t*,*d*) = *ft*,*d*

The relative weight of each term using turn frequency is calculated by dividing the term frequency by the cumulative frequency to gain the relative frequency of each term.

In contrast inverse document frequency inverts the frequency of the term in the documents Under the assumption that the most common words would be the least indicative of the documents overall meaning.

Idff=log10(N/dff)

in combination turn frequency inverse document frequency indicates the relative weight or importance of a term regarding the entire corpus of documents. This is crucial in feature extraction as it identifies the most important terms in the corpus that can provide insight into the documents overall meaning.

[appendix count vector representation]

Appendix X shows a sample of the output of the term frequency matrix representation of the data. Each row represents a document in the corpus in our case a review with each column representing a term and its frequency.

[appendix tfidf representation]

Appendix X shows a sample of the output of the TF-IDF matrix representation of the data. In review we conclude that in the context of food reviews TF-IDF is a more informative vector space representation then turn frequency as it indicates the relative value of each word regarding our corpus of reviews.

Furthermore, we experimented with n-grams. By default, unigrams were used to represent features then we attempted to try bigrams and trigrams and quad grams.

Term Frequency – Bigrams

TF-IDF – bigrams

Term Frequency – Trigrams

TF-IDF – Trigrams

Term Frequency – Tera-grams

TF-IDF – Tera-grams

As seen from the above examples, n-grams affected …..

# Q4 Model training, selection and hyperparameter tuning and evaluation

in this section we were trying to figure out the best model with the best parameters possible, to do that we have combined **Pipeline** with **the Grid Search** where the Pipeline would be using CountVectorizer and TFIDF transformer and one of 3 estimators

LogisticRegression

MultinomialNB

SGDClassifier

**Pipeline** is a process that enabled us to create all the steps needed for our text analytics model to work starting from text vectorization, TF-IDF and then applying our statistical model. We have also used **GridSearch** and that enabled us to test several parameters with several settings allowing us to test several parameters combinations and compare results in one run , we even tested functions we wrote in previous steps and saw how they impact the efficiency of the model. We did test multiple combination of models, estimators and parameters, we used dictionaries to load estimators and the parameters, using dictionaries enabled us to add remove estimators and parameters as quickly as needed.

The drawback of that process is it was very compute intensive process and complex to scale we did go through several iteration of processor and memory optimization to make the code work on larger servers ( the use of **n\_jobs** on several cores lead to known symptom of *memory explosion* , we had to optimize using **pre\_dispatch** to control the number of jobs that get dispatched during parallel execution ) .

What we have also learned by experiment that many of the parameters did not contribute positively or impact the results, in fact in many scenarios using the default parameters were the best option

Below is a sample of the model output using *LogisticRegression* with GridSearch n**-gram** of (1,1), (1,2), (1,3)

[appendix Q4 results – logistic regression results]

# Q5 Topic Modelling of high and low ratings

**test**

# Conclusions

**test**

# Appendix

**APPENDX a**

**Doc flow:**

1. **introduction**
   1. **scope**
   2. **objective**
2. **data exploration**
   1. **outliers**
   2. **data distribution skewness**
   3. **duplicates**
   4. **null values**
   5. **correlation**
3. **data preprocessing**
   1. **stop words**
   2. **accents**
   3. **lemmitization**
   4. **stemming**
   5. **word conjugates**
   6. **punctuation**
   7. **numbers removal**
4. **vector space model representations**
   1. **count based**
   2. **tf-idf weight based**
   3. **n-grams**
   4. **influence of ngrams**
5. **model EVALUATION, TRAINING, hyperparameters**
   1. **models used**
   2. **hyperparameter selection**
   3. **evaluation**
6. **topic modeling**
   1. **lda**
   2. **nmf**
   3. **svd**
   4. **observations**
7. **conclusion**
8. **appendix**