**29th February**

20



08

**Fall**

F21AA Applied Text Analytics - Coursework 1

Tauro, Bruce H00228269 – Salah, Tamer H00343334 – Itani, Tarek H00292565 - Serry, Mohamed H00313456

Table of Contents

[Table of Contents 2](#_Toc33993044)

[0. Introduction 2](#_Toc33993045)

[1. Data Exploration and Visualization 3](#_Toc33993046)

[2. Text Processing and Normalization 5](#_Toc33993047)

[3. Vector space Model and feature representation 6](#_Toc33993048)

[4. Model training, selection and hyperparameter tuning and evaluation 8](#_Toc33993049)

[5. Topic Modelling of high and low ratings 9](#_Toc33993050)

[6. Experiments Discussion 12](#_Toc33993051)

[7. Conclusion 12](#_Toc33993052)

[8. References 13](#_Toc33993053)

[9. Appendix 14](#_Toc33993054)

# 0. Introduction

**0.1 Purpose**

In this report we will get introduced to essential text processing techniques, representation and analysis, and compare different machine learning techniques on text reviews classification.

**0.2 Approach**

Our dataset constitutes of 400K+ reviews of products which are classified into 5 different classes representing scores. Model was implemented using pipeline and grid search, trying all hyper parameters to get the best result. All this was done after analyzing data and implement the best processing and cleaning techniques to get the best results.

**0.3 Method**

To achieve this, we used python programming with the help of necessary libraries for Text Processing including but not limited to NLTK. In our experiment, we followed different steps to clean, process and run the model on our dataset.

Below Steps are done to achieve our GOAL:

1. Data exploration and visualization
2. Text processing and normalization
3. Vector space model and feature representation
4. Model training, selection and hyper parameters tuning and evaluation
5. Topic modeling for high and low ratings

# 1. Data Exploration and Visualization

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

**Step 1**: we had an overview about columns, their count and type:

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 found 6 numerical attributes and 5 categorical or non-numeric attributes in total with Score as our class attribute.

**Step 2:** we did a **statistical analysis of our data set.** [Figure 1.1] and [Figure 1.2] showed us that **most of the data is biased towards the score 5.0** that means our examples are not **stratified** or evenly distributed, we began to analyze the data considering the score distributions. As seen from [Figure 1.3], **272492** examples have a class value of 5.0 this over **50% of our data.**

A close up of a device

Description automatically generated

Figure 1.1 – Columns Histogram

A screenshot of a cell phone

Description automatically generated

Figure 1.2 – Statistical Table

A screen shot of a social media post

Description automatically generated

Figure 1.3 – Count group by Score

**Step 3**: We did investigation about correlation, although this step was not really necessary since we are predicting the score using text written by customer, but it gave us a better understanding about our data and features. Our analysis was split in to 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 appear [Figure 1.4]. The below plots gave no indicative relationship.

**A screenshot of a cell phone

Description automatically generated**

Figure 1.4 – Score/Feature Relation

For the above reason, we thought that a good idea is to get rid of unnecessary columns which we did at a later stage. On the other hand, and as we are predicting our score according to input from user, we merged users two inputs (Text and Summary) into one field called review.

**Step 4:** we did analysis on length of text field by adding new column, displaying info and plotting its data. In [Figure 1.5] and below info, we found that we have some outliers in terms of text field length that needs our attention. And by examining data in the maximum text length record, we found that it contains irrelevant data which was copied from different place and may be considered confusion to our model.

count 426340.000000

mean 435.396902

std 443.943421

min 12.000000

25% 179.000000

50% 301.000000

75% 526.000000

max 21409.000000

A close up of a logo

Description automatically generated

Figure 1.5 – Text Field Length

**Step 5**: during our examination to some large text in our data, we found that our data contains some html tags, punctuations, accents and some other irregular characters that needs to be cleaned.

# 2. Text Processing and Normalization

We divided this into 2 stages **Text processing** and **Data** **Normalization**:

**2.1 Text Processing:** we found *HTML tags*, *punctuations, special characters, and duplicates* in the data which we removed using a series of functions [Appendix A]. The reason behind this is it is likely that HTML tags, punctuation and other special characters will not add any value to the model accuracy, on the contrary they will have a negative impact on the accuracy and execution speed of our model as they do not contribute to the text. Below steps were used to clean and process the data:

Extract Only Needed Columns Remove Nulls Remove Html Tags Remove Outliers Correct Accented Letters Remove Punctuations Expand Short Words Remove Remaining Apostrophe Remove Numbers Remove Extra Spaces Transform to Lower Case Remove Stop Words Remove Duplicates

**2.2 Data Normalization:**  this step involved **stemming** and **lemmatization**, these are two common techniques used for text normalization. Stemming cuts the word into a shorter term or representation without looking into its lexical meaning of the word without considering if the new word is a correct word or not, while lemmatizing does the same thing by simplifying the word into its basic form. We used the **Port Stemmer** and **WordNet Lemmatizer** from the *NLTK* library. Lemmatize used more complex algorithm and takes more time to execute than the stemmer. And since we are not working with a model that needs to maintain the context of the word, we found that using stemmer is more convenient to us since it is fast and will fulfill our needs. [Figure 2.1] shows an example taken from our data on the effect on stemming and lemmatizing on couple of words.

|  |  |  |  |
| --- | --- | --- | --- |
| Original | Stemming | Lemmatization | Comments |
| addictive | addict | addictive | Lemmatization didn’t do any changes depending on which parameter we are using (adjective, verb or noun) while stemming gave shorter word |
| noticed | notic | notice | Stemming gave a shorter word which is not contextual, while lemmatization gave the correct basic form of the word. |

Figure 2.1 – Stemming/Lemmatizing Comparison

In the above table, we can see that lemmatizing has more practical use when we have model that needs to maintain the correct meaning of the word such as summarizing.

# 3. Vector space Model and feature representation

For this segment of the course work we aimed to experiment with **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.

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. [1]

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

The relative weight of each term is its relative frequency which is calculated by dividing the term frequency by the cumulative frequency. 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)

When used 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 to the documents overall meaning.

[Figure 3.I] shows a sample of the output of the term frequency matrix representation of the data after applying function [Appendix B] which takes parameter vectorizer “count” to give output as term frequency. Each row represents a document in the corpus in our case a review with each column representing a term and its frequency.

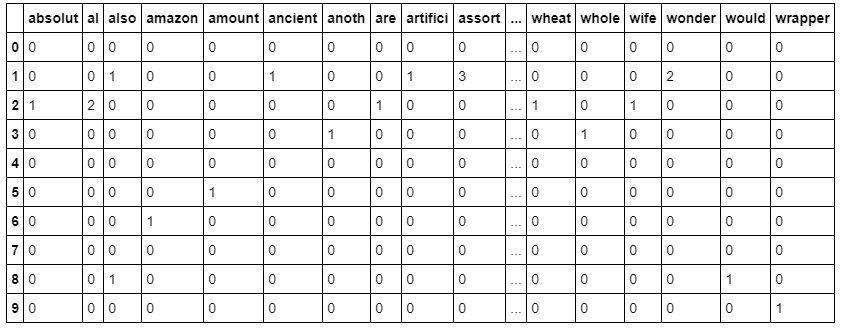


Figure 3.1 – Term Frequency Vector Model

[Figure 3.2] shows a sample of the output of the TF-IDF matrix representation of the data after applying function [Appendix B] which takes parameter vectorizer “tfidf” to give output as term frequency. 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.

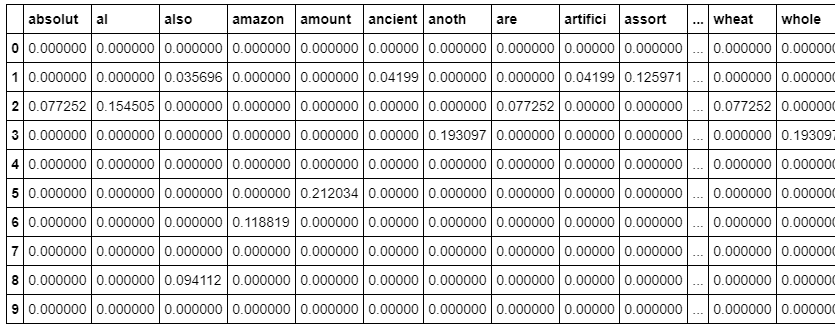


Figure 3.2 – TFIDF Vector Model

Analyzing and reviewing some words after vectorizing examples (“taste”) revealed that single word vectorizing is not enough to get real insight of user opinion. So furthermore, we experimented with n-grams [Figure 3.3]. By default, unigrams were used to represent features then we attempted to try bigrams and trigrams. Using bigram helped us to show the real opinion of the user by adding another word to (“taste”) to be (“good taste”) or (“bad taste”). The effect of n-gram will be discussed later in the modeling and running real scenarios with and without n-grams.

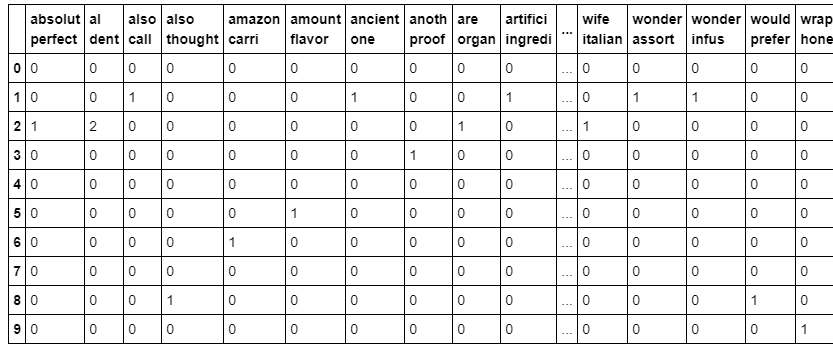


Figure 3.3 – Term Frequency Vector Model with Bi Gram

# 4. Model training, selection and hyperparameter tuning and evaluation

In this section we were trying to identify the best model with the optimal parameters. to achieve this, we have combined **Pipeline** and **GridSearch** for hyperparameter tuning.



Figure 4.1 – Model Pipeline Used

**Pipeline** is a process that enabled us to automate the required steps needed for our model to work starting from data loading, processing, reduction and finally training a model on the prepared textual data.

Our **Pipeline [Figure 4.1]** begins with **CountVectorizer** to map out a token matrix of our existing data. Followed by a **TF-IDF transformer** to give us the relative weight of each feature/word in our vocabulary, initially we experimented with **data reduction** using **TruncatedSVD** but we later decided it did not impact our results thoroughly. Finally, our pipeline included one of our chosen estimators. We have also used **GridSearch** which enabled us to test several hyperparameters for the entire pipeline, we also tested multiple combination of models using dictionaries to store/load the parameters, this enabled us to add or remove estimators and parameters as needed.

The drawback of that process is it was a computationally intensive process and complex to scale up if we attempted to use **GridSearch** for numerous parameters such as the **min\_df** for the vectorizers. We went through several iterations of processor and memory optimization to make it feasible 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). Our experiments revealed that many of the parameters did not contribute positively or impact the results, in fact in many cases using the default parameters were the best option.

As a result of our pipeline process, **LogisticRegression** gave the best results among our chosen estimators **with little to no difference** in results between training and evaluation. This means our model has **low variance. [Figure 4.2]**



Figure 4.2 – Results on Test Data

[Appendix E] shows several samples of the model cross validation output

# 5. Topic Modelling of high and low ratings

In this section we use topic modelling to find common threads between reviews of score 5 and reviews of score 1, with the aim of discerning any patterns in the terms that contribute to the topic.

We explored several techniques for topic modelling namely NMF, SVD and LDA:

Non-negative matrix factorization (NMF)

The NMF model decomposes its input matrix into two smaller approximate product matrices that only contain nonnegative values these are iteratively adjusted until they more closely result into the input matrix due to this process the features are clustered as the error value is reduce during each iteration. NMF can take input matrices that have been processed by both term frequency and TF-IDF

Singular value decomposition (SVC)

The SVD model decomposes the input matrix into its constituent parts in the form of 3 matrices. SVD acts as a feature reducer removing terms that are not important to the overall corpus.

Latent Dirichlet Allocation

LDA assumes that all topics follow a Dirichlet distribution across the documents in the corpus this leads to the probabilities of context between words being preserved. LDA groups together terms that occur together often into a topic which at times may not lead to topical grouping.

**Experiment**

We extracted 1 and 5 Scores into two different sub data sets [Appendix C], then we ran the experiment using NMF, SVD and LDA on both sets. After that we displayed our Topic Models using function in [Appendix C] and we compared our results.

**Observations**

NMF TF

When using NMF with a TF input there is a clear trend in the topics it is very easy to find an overarching commonality between the terms and possible “topic header” for example [Figure 5.1], topic 9 of the score 5 group and topic 11 of the score 1 group are both about hair products. There is also a trend in the use of positive words in the Score 5 group and negative words in the Score 1 group, while the words No and Not appear in the Score 5 Group more telling words like bad, disappointment, weak and burn are seen in the Score 1 Group in comparison Score 5 group has no such words instead words like loves, like, quality and favorite are seen but some positive terms are seen in Score 1’s topics.

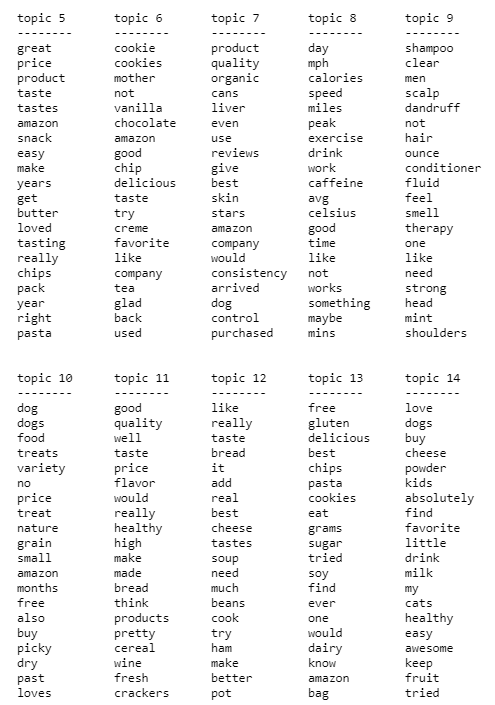


Figure 5.1 – NMF result with TF

NMF TF-IDF

The results of NMF IDF are similar to those seen in NMF TF with terms being grouped in similar topic groups. Score 5 group Topic 2 of NMF models are almost the same.

SVD

SVD’s topics tends to have several improperly assigned terms in the Score 5 group topic 16 which seems to cover to topic of beverages has an out of place term “dog” this improperly assigned terms occurs in all topics with some topics having no clear topic.

LDA

LDA has mixed results there is no clear positive or negative arrangement of terms in the Score groupings. Some topics like Score 5 group topic 0 is clearly about Oil while theory topics like topic 0 of Score 1 Group has many unrelated terms in them making topic assignment unclear.

Overall Observation

The NMF model has the most coherent results all topics have a clear theme and the Score groups do not have any contamination of positive and negative terms. LDA picks up word groups that occur commonly with one another leading to mis-grouped terms while SVC approximates the groups very loosely.

# 6. Experiments Discussion

In this course work the team collectively got together trying to solve the IMDB problem using statistical Natural Language Processing techniques , we have to admit that the questions and their chronological sequence helped shape our thinking to come out with a good working framework to solve this challenge , to the best of our ability.

As we started to understand the data that we were given, visualizing it gave us valuable insights on the action plan that we took during the coursework, we became very clear on the data bias toward a specific score, the relevance of some features, and the diversity and magnitude of normalization that we need to do (HTML Tags, null values, duplicates. Etc.), which we did in question 2 and we have documented several functions in our code to deal with those issues.

After normalizing the data, we did a research on the best text vectorization technique we can use for this data, we experimented several techniques and several libraries to reach an optimal vectorization technique, and we did document our findings and also tested them with our model in later questions.

On then, we tried to came up with a method to apply some of the techniques we have learned during the course to come out with an appropriate model , from estimators selections, Pipelines generation…etc., We tested that on the validation data set as a true measure of our model effectiveness.

We finished this by applying Topic Modeling to the data and generating further insights, you will find our all our research summary detailed in section 6.

Note from the team: There were a great deal of learning and collaboration during the coursework, from applying principals we have learned in the class and researching things helped us conclude what is presented in this document. It was a great learning and experience for all team members.

# 7. Conclusion

In conclusion, we found that the hyper parameters of the statistical models were challenging to optimize.

Furthermore, feature engineering plays an important role for the success and validity of the models. Preprocessing is required and it is computationally expensive to run, even on relatively high spec server (20 Core/80GB) , it took several hours to process the entire date set with GridSearchCV.

We observed that utilizing Logistic Regression with bi-grams usually yielded on average 2-3 pts. higher accuracy and well-balanced tradeoff between recall and precision.

Lemmatization and stemming did not yield any observable improvement to our models, moreover, Stop words actually degraded the model accuracy .

# 8. References

[1] https://scikit-learn.org/stable/modules/generated/sklearn.feature\_extraction.text.CountVectorizer.html

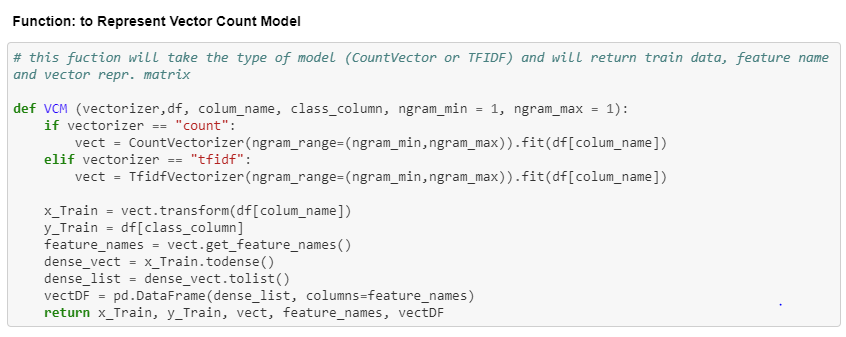
# 9. Appendix

Appendix A – Text Processing

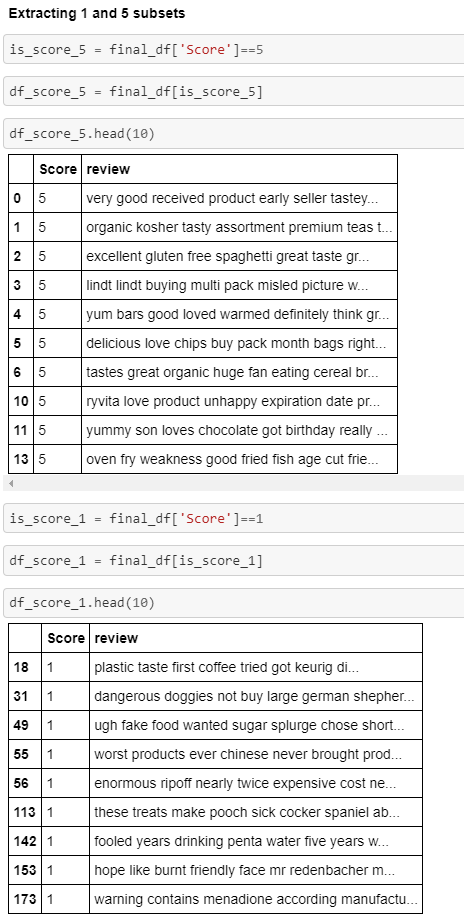




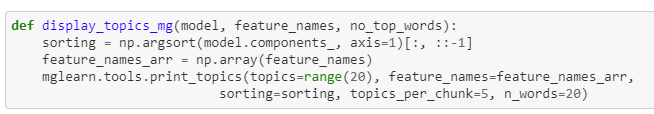
Appendix B – Vector Count / TFIDF Function



Appendix C – Extracting 1 and 5 Scores



Appendix D – Display Topic Models Function



Appendix E – Model Cross Validation results

Testing Default Parameters

LogisticRegression scored 0.7424452417652164

Best parameter (CV score=0.739):

{}

precision recall f1-score support

1 0.66 0.71 0.68 7066

2 0.40 0.20 0.26 4096

3 0.45 0.32 0.37 5845

4 0.49 0.26 0.33 10919

5 0.81 0.95 0.88 49277

accuracy 0.74 77203

macro avg 0.56 0.49 0.51 77203

weighted avg 0.70 0.74 0.71 77203

Cross validation with unseen test data

Model score on unseen data 0.6766680270768538

precision recall f1-score support

1 0.77 0.34 0.47 13075

2 0.45 0.06 0.10 7416

3 0.56 0.07 0.12 10647

4 0.65 0.01 0.02 20346

5 0.68 1.00 0.81 90630

accuracy 0.68 142114

macro avg 0.62 0.29 0.30 142114

weighted avg 0.66 0.68 0.57 142114

Testing with default parameters with weighted labels and C= 1 (Values obtained from GridSearch)

LogisticRegression scored 0.7410333795318835

Best parameter (CV score=0.739):

{'classifier\_\_C': 1}

precision recall f1-score support

1 0.65 0.71 0.68 7105

2 0.41 0.19 0.26 4090

3 0.44 0.31 0.36 5838

4 0.49 0.25 0.33 10963

5 0.81 0.95 0.87 49207

accuracy 0.74 77203

macro avg 0.56 0.48 0.50 77203

weighted avg 0.70 0.74 0.71 77203

Cross validation with unseen test data

Model score on unseen data 0.6784201415764809

precision recall f1-score support

1 0.77 0.35 0.48 13075

2 0.41 0.07 0.11 7416

3 0.52 0.07 0.13 10647

4 0.73 0.01 0.02 20346

5 0.68 1.00 0.81 90630

accuracy 0.68 142114

macro avg 0.63 0.30 0.31 142114

weighted avg 0.67 0.68 0.58 142114

Testing with Default parameters with no Stop Words removal (GridSearch for ngrams 1,2,3)

LogisticRegression scored 0.7571978086024013

Best parameter (CV score=0.757):

{'cv\_\_ngram\_range': (1, 2)}

precision recall f1-score support

1 0.68 0.74 0.71 7017

2 0.47 0.16 0.23 4106

3 0.49 0.37 0.42 5873

4 0.55 0.26 0.35 11078

5 0.81 0.97 0.88 49137

accuracy 0.76 77211

macro avg 0.60 0.50 0.52 77211

weighted avg 0.72 0.76 0.72 77211

Cross validation with unseen test data

Model score on unseen data 0.762901614197053

precision recall f1-score support

1 0.72 0.73 0.72 13075

2 0.58 0.17 0.27 7416

3 0.55 0.37 0.44 10647

4 0.58 0.25 0.35 20346

5 0.80 0.98 0.88 90630

accuracy 0.76 142114

macro avg 0.65 0.50 0.53 142114

weighted avg 0.73 0.76 0.72 142114

Appendix F – Source Code

<https://github.com/mhserry/F21AA>