29th February 20

F21AA Applied Text Analytics - Coursework 1

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

TABLE OF CONTENTS

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.**

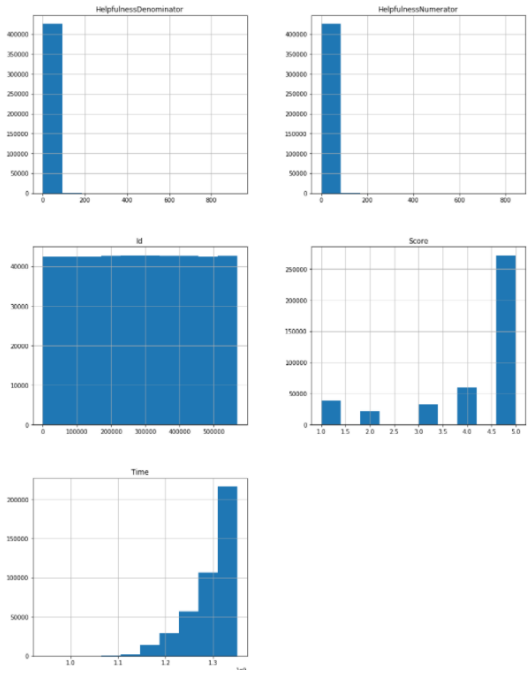

Figure 1.1 – Columns Histogram

	Id	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
count	426340.000000	426340.000000	426340.000000	426340.000000	4.263400e+05
mean	284300.799618	1.733159	2.220244	4.183455	1.296222e+09
std	164012.600602	7.328184	8.014894	1.310577	4.808061e+07
min	1.000000	0.000000	0.000000	1.000000	9.408096e+08
25%	142363.750000	0.000000	0.000000	4.000000	1.271203e+09
50%	284308.500000	0.000000	1.000000	5.000000	1.311034e+09
75%	426183.250000	2.000000	2.000000	5.000000	1.332720e+09
max	568454.000000	844.000000	923.000000	5.000000	1.351210e+09

Figure 1.2 – Statistical Table

	Id							
	count	mean	std	min	25%	50%	75%	max
Score								
1	39193.0	282904.535606	162865.104452	2.0	142706.0	283604.0	421880.0	568434.0
2	22353.0	280643.847850	164445.543153	68.0	137929.0	278032.0	422521.0	568451.0
3	31993.0	279476.092333	165426.046308	46.0	136097.0	274969.0	424951.0	568423.0
4	60309.0	281941.848729	164780.725409	3.0	138600.0	280352.0	424999.0	568421.0
5	272492.0	285890.169021	163781.498091	1.0	144535.5	287247.5	427344.5	568454.0

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.

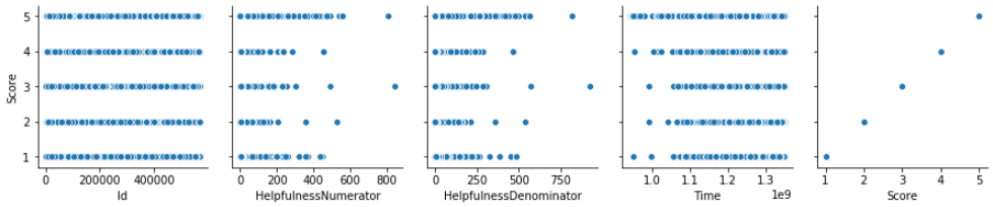

Figure 1.4 – Score/Feature Relation

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
```

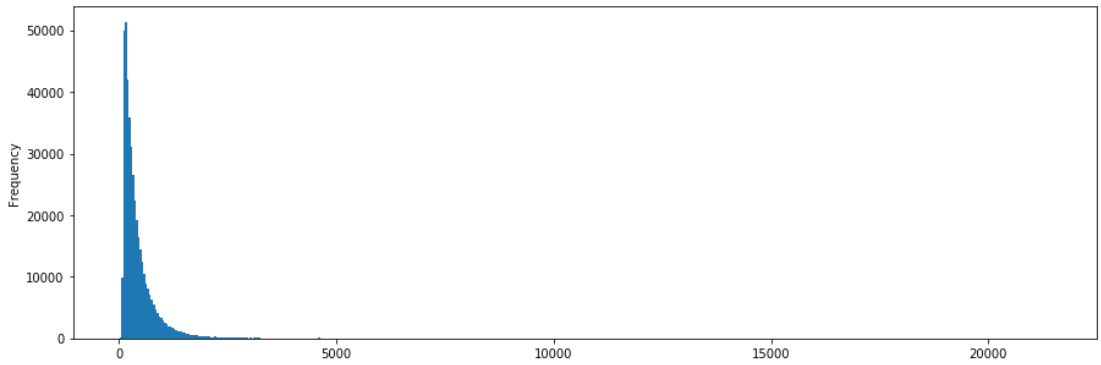

Figure 1.5 – Text Field Length

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]

$$\mathrm{tf}(t,d) = f_{t,d}$$

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.

$$\mathrm{Idf}_f = \log_{10}(N/\mathrm{df}_f)$$

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.

	absolut	al	also	amazon	amount	ancient	anoth	are	artifici	assort	...	wheat	whole	wife	wonder	would	wrapper
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
1	0	0	1	0	0	1	0	0	1	3	...	0	0	0	2	0	0
2	1	2	0	0	0	0	0	1	0	0	...	1	0	1	0	0	0
3	0	0	0	0	0	0	1	0	0	0	...	0	1	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
5	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0	0
6	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0
8	0	0	1	0	0	0	0	0	0	0	...	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	1

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.

	absolut	al	also	amazon	amount	ancient	anoth	are	artifici	assort	...	wheat	whole
0	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.00000	0.000000	...	0.000000	0.00000(
1	0.000000	0.000000	0.035696	0.000000	0.000000	0.04199	0.000000	0.000000	0.04199	0.125971	...	0.000000	0.00000(
2	0.077252	0.154505	0.000000	0.000000	0.000000	0.00000	0.000000	0.077252	0.00000	0.000000	...	0.077252	0.00000(
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.193097	0.000000	0.00000	0.000000	...	0.000000	0.19309(
4	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.00000	0.000000	...	0.000000	0.00000(
5	0.000000	0.000000	0.000000	0.000000	0.212034	0.00000	0.000000	0.000000	0.00000	0.000000	...	0.000000	0.00000(
6	0.000000	0.000000	0.000000	0.118819	0.000000	0.00000	0.000000	0.000000	0.00000	0.000000	...	0.000000	0.00000(
7	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.00000	0.000000	...	0.000000	0.00000(
8	0.000000	0.000000	0.094112	0.000000	0.000000	0.00000	0.000000	0.000000	0.00000	0.000000	...	0.000000	0.00000(
9	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.00000	0.000000	...	0.000000	0.00000(

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.

	absolut perfect	al dent	also call	also thought	amazon carri	amount flavor	ancient one	anoth proof	are organ	artifici ingredi	...	wife italian	wonder assort	wonder infus	would prefer	wrap hone
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
1	0	0	1	0	0	0	1	0	0	1	...	0	1	1	0	0
2	1	2	0	0	0	0	0	0	1	0	...	1	0	0	0	0
3	0	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0
6	0	0	0	0	1	0	0	0	0	0	...	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0
8	0	0	0	1	0	0	0	0	0	0	...	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	1

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.

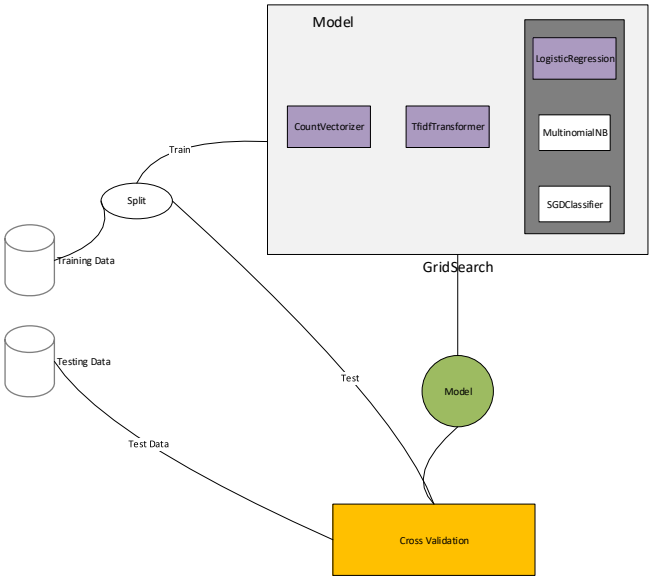

Figure 4.1 – Model Pipeline Used

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]**

```
Model score on unseen data 0.7733087521285729
              precision    recall  f1-score   support

           1       0.70      0.76      0.73     13075
           2       0.56      0.25      0.34      7416
           3       0.54      0.40      0.46     10647
           4       0.59      0.31      0.41     20346
           5       0.82      0.97      0.89     90630

    accuracy                           0.77    142114
   macro avg       0.65      0.54      0.57    142114
weighted avg       0.75      0.77      0.75    142114
```

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.

topic 5	topic 6	topic 7	topic 8	topic 9
great	cookie	product	day	shampoo
price	cookies	quality	mph	clear
product	mother	organic	calories	men
taste	not	cans	speed	scalp
tastes	vanilla	liver	miles	dandruff
amazon	chocolate	even	peak	not
snack	amazon	use	exercise	hair
easy	good	reviews	drink	ounce
make	chip	give	work	conditioner
years	delicious	best	caffeine	fluid
get	taste	skin	avg	feel
butter	try	stars	celsius	smell
loved	creme	amazon	good	therapy
tasting	favorite	company	time	one
really	like	would	like	like
chips	company	consistency	not	need
pack	tea	arrived	works	strong
year	glad	dog	something	head
right	back	control	maybe	mint
pasta	used	purchased	mins	shoulders

topic 10	topic 11	topic 12	topic 13	topic 14
dog	good	like	free	love
dogs	quality	really	gluten	dogs
food	well	taste	delicious	buy
treats	taste	bread	best	cheese
variety	price	it	chips	powder
no	flavor	add	pasta	kids
price	would	real	cookies	absolutely
treat	really	best	eat	find
nature	healthy	cheese	grams	favorite
grain	high	tastes	sugar	little
small	make	soup	tried	drink
amazon	made	need	soy	milk
months	bread	much	find	my
free	think	beans	ever	cats
also	products	cook	one	healthy
buy	pretty	try	would	easy
picky	cereal	ham	dairy	awesome
dry	wine	make	know	keep
past	fresh	better	amazon	fruit
loves	crackers	pot	bag	tried

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

Function: to drop unused columns

```
#This fuction will return modified dataframe with selected columns only
def neededColumnsOnly(df, columns_needed):
    #return df.filter(columns_needed)
    #Or
    to_drop_columns = list(x for x in df.columns.tolist() if x not in columns_needed)
    df.drop(to_drop_columns,axis=1,inplace=True)
    return df
```

Function: to find columns containing nulls

```
#This fuction will check which columns having null values
def columns_with_nulls(df, index_column):
    null_columns = []
    for column in df.columns:
        if column != index_column:
            check_null = df.isnull()[[column,index_column]].groupby(column).agg('count')
            try:
                if check_null.loc[True][0] > 0:
                    null_columns.append(column)
            except:
                pass
    return null_columns
```

Function: to remove all null values from columns

```
#This fuction will remove all nulls in columns found earlier
def remove_nulls(df, columns):
    for column in columns:
        df[column].fillna('', inplace=True)
    return df
```

Function: to find columns containing html tags

```
#This fuction will check which columns having html tags
def columns_with_html(df):
    null_columns = []
    for column in df.columns:
        try:
            text_html = df[column].str.find('<')
            text_html = text_html[text_html != -1]
            check_null = len(text_html)
            if check_null > 0:
                null_columns.append(column)
        except:
            pass
    return null_columns
```

Function: to remove html tags from data

```
#This fuction will remove all html tags in text
def remove_html_tags(text):
    sp = BeautifulSoup(text, "html.parser")
    returned_text = sp.get_text(separator=" ")
    return returned_text
```

Function: to merge Summary and Text in one column and get rid of productId column

```
#This fuction will result in two columns only score and review
def merge_summary_text(df):
    df['review']=df['Summary']+' '+df['Text']
    df.drop(['Summary','Text','ProductId'],axis=1,inplace=True)
    return df
```

Function: to correct accent in letters

```
#This function will replace accents in letters with regular letters example nescafé will be nescafe
def correct_accent(text):
    returned_text = unidecode.unidecode(text)
    return returned_text
```

Function: to visualized score count

```
def visualize_score_count(df):
    # visualize total review by score count
    final_df_grouped = df[['Score', 'review']].groupby('Score').agg('count')
    final_df_grouped= final_df_grouped.reset_index()
    f, ax = plt.subplots(figsize=(8, 6))
    fig = sb.barplot(x='Score', y="review", data=final_df_grouped)
```

Function: to remove punctuations

```
#This function will remove punctiatons
def remove_punctuations(text):
    for punc in string.punctuation.replace("'",""):
        if punc in text:
            text = text.replace(punc," ")
    return text
```

Function: to remove extra spaces

```
#This function will remove extra spaces
def remove_extra_space(text):
    removed_space = " ".join(text.split())
    return removed_space
```

Appendix B – Vector Count / TFIDF Function

Function: to Represent Vector Count Model

```
# this fuction will take the type of model (CountVector or TFIDF) and will return train data, feature name
and vector repr. matrix

def VCM (vectorizer,df, colum_name, class_column, ngram_min = 1, ngram_max = 1):
    if vectorizer == "count":
        vect = CountVectorizer(ngram_range=(ngram_min,ngram_max)).fit(df[colum_name])
    elif vectorizer == "tfidf":
        vect = TfidfVectorizer(ngram_range=(ngram_min,ngram_max)).fit(df[colum_name])

    x_Train = vect.transform(df[colum_name])
    y_Train = df[class_column]
    feature_names = vect.get_feature_names()
    dense_vect = x_Train.todense()
    dense_list = dense_vect.tolist()
    vectDF = pd.DataFrame(dense_list, columns=feature_names)
    return x_Train, y_Train, vect, feature_names, vectDF
```

Appendix C – Extracting 1 and 5 Scores

Extracting 1 and 5 subsets

```
is_score_5 = final_df['Score']==5
```

```
df_score_5 = final_df[is_score_5]
```

```
df_score_5.head(10)
```

	Score	review
0	5	very good received product early seller tastey...
1	5	organic kosher tasty assortment premium teas t...
2	5	excellent gluten free spaghetti great taste gr...
3	5	lindt lindt buying multi pack misled picture w...
4	5	yum bars good loved warmed definitely think gr...
5	5	delicious love chips buy pack month bags right...
6	5	tastes great organic huge fan eating cereal br...
10	5	ryvita love product unhappy expiration date pr...
11	5	yummy son loves chocolate got birthday really ...
13	5	oven fry weakness good fried fish age cut frie...

```
is_score_1 = final_df['Score']==1
```

```
df_score_1 = final_df[is_score_1]
```

```
df_score_1.head(10)
```

	Score	review
18	1	plastic taste first coffee tried got keurig di...
31	1	dangerous doggies not buy large german shepher...
49	1	ugh fake food wanted sugar splurge chose short...
55	1	worst products ever chinese never brought prod...
56	1	enormous ripoff nearly twice expensive cost ne...
113	1	these treats make pooch sick cocker spaniel ab...
142	1	fooled years drinking penta water five years w...
153	1	hope like burnt friendly face mr redenbacher m...
173	1	warning contains menadione according manufactu...

Appendix D – Display Topic Models Function

```
def display_topics_mg(model, feature_names, no_top_words):
    sorting = np.argsort(model.components_, axis=1)[:, ::-1]
    feature_names_arr = np.array(feature_names)
    mglearn.tools.print_topics(topics=range(20), feature_names=feature_names_arr,
                               sorting=sorting, topics_per_chunk=5, n_words=20)
```

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