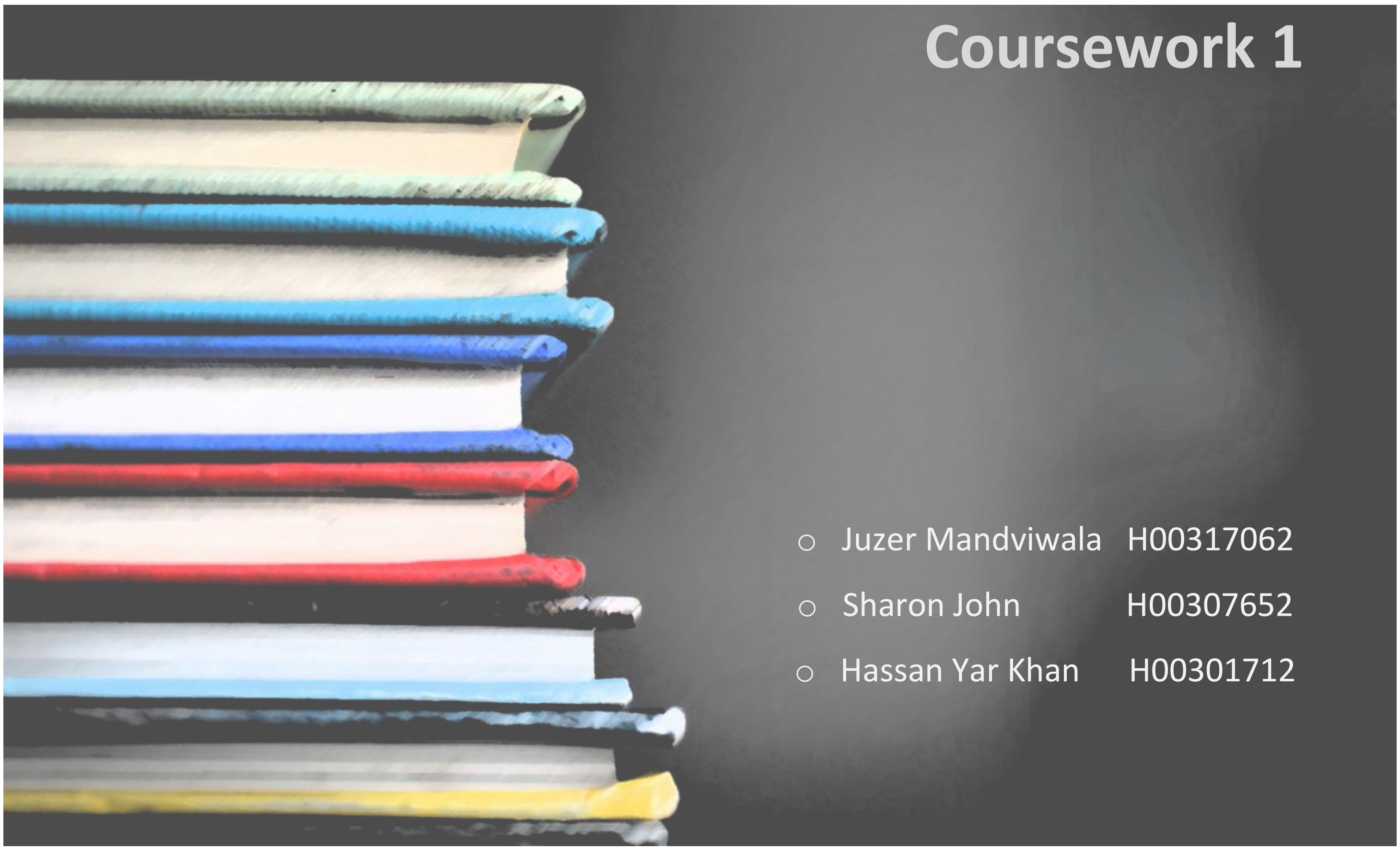
F21AA Applied Text Analytics



Program: F2D1-DSC Master of Science in Data Science

March 2020

Repository Link : https://heriotwatt-my.sharepoint.com/:f:/g/personal/jsm7_hw_ac_uk/EqE0x_aH_6dKi9ptvPh8JGsBea_0AM9GAMFV_E7XI3_VSg?e=IHvcPU

1. Data Exploration and Visualization:

The exploration and visualization are to be done on a dataset consisting of reviews of fine foods from amazon. We were given a subset consisting of *4,26,340* reviews from approximately 2008-2012.
Below are the attributes in the dataset :

Attribute	Data Type	Description
ID	int64	Row ID
Product ID	object	Unique Identifier of the Product being reviewed
User ID	object	Unique Identifier of the User
Profile Name	object	Profile Name of the user who is reviewing the product
Helpfulness Numerator	int64	Number of users who found the review to be helpful
Helpfulness Denominator	int64	Number of users who indicated whether they found the review helpful or not
Score	int64	Rating between 1 and 5
Time	int64	Timestamp of the review
Summary	object	Brief summary of the review
Text	object	Text of the review

Table 1

We started our data exploration by check on missing data in the given dataset. We found out, there are 20 *nulls* in Summary and 14 *nulls* in Profile Name. We handled the missing data, by updating the Summary field with blank and 'NA' in Profile Name.
As per the initial notebook available on vision, we also concatenated the Summary and Text Field , and we will be doing the pre-processing and normalization on the concatenated field called 'Summary_text'
Our visualization task , started by looking at the distribution of the Score in the data set, we used the seaborn library, to achieve the same.

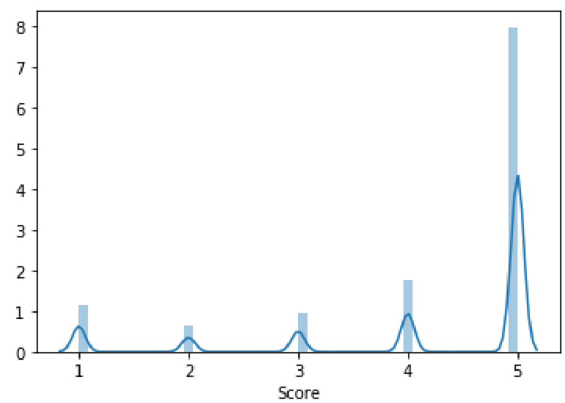
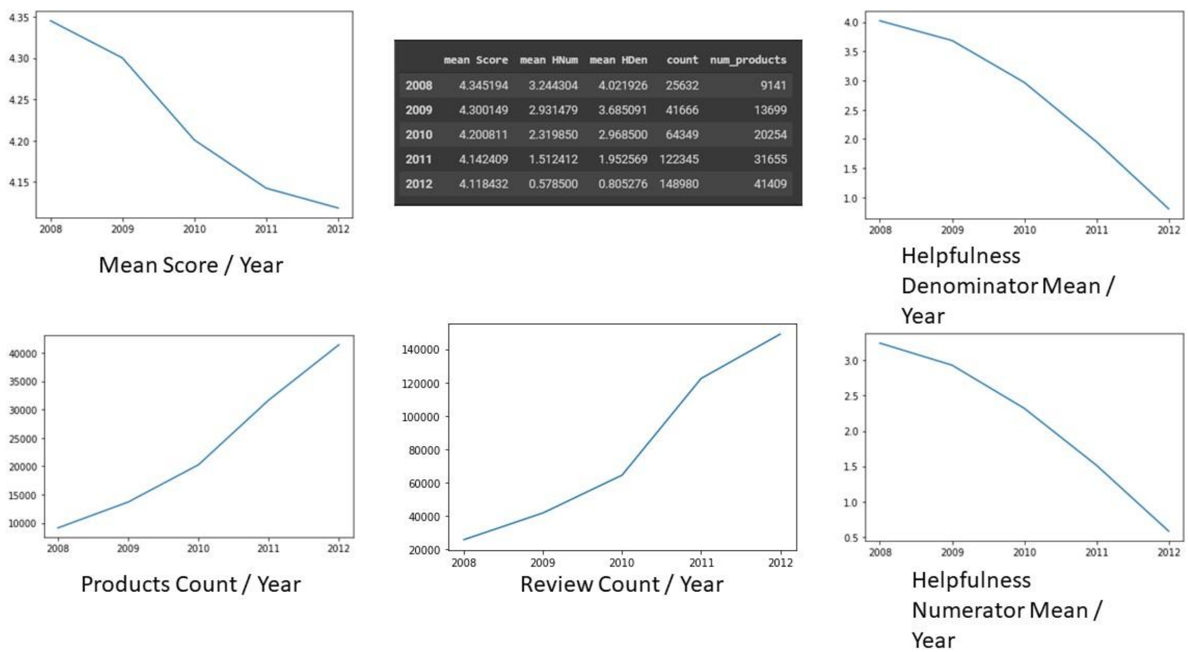


Figure 1

Observation: We can clearly see here the data has high proportion of 5 Star Review compared to the rest.

We then took reference from a Kaggle Project of Amazon Fine Food Reviews, to gather insightful exploration and visualization ideas (Anon., 2020)
Time Range Visualization of various attributes, to achieve the said visualization we started by converting the given *'time'* attribute to datetime data type and then creating new data-frame and setting the datetime column as the index. Post that we split the dataset into different dataframes for each year and mean values of Score / HelpfulnessNumerator / HelpfulnessDenominator, total number of reviews and no. of products. Post that we summarize the above statistical data in one final data-frame which has the year

as its index.



	mean Score	mean HNum	mean HDen	count	num_products
2008	4.345194	3.244304	4.021926	25632	9141
2009	4.300149	2.931479	3.685091	41666	13699
2010	4.200811	2.319850	2.968500	64349	20254
2011	4.142409	1.512412	1.952569	122345	31655
2012	4.118432	0.578500	0.805276	148980	41409

Figure 2

Observations: Mean Score over the period has gradually dropped. While the number of products and number of reviews over the years have steadily risen. Mean Helpfulness Numerator, is seen to be declining , which indicates that over the years, less people have fond reviews helpful. Helpfulness Denominator, which covers both votes of both helpful as well as unhelpful vote of a review, the means shows a decline over the year, indicating the helpful / unhelpful votes are not being added by a user.

Visualization of Helpful Vote among user Scores

We now investigated the helpful metrics in more details, we have taken reference from the Kaggle Project (DLao, 2019). We calculated the helpful % by dividing the Helpfulness Denominator with the Helpfulness Numerator, where the value is zero, we update it with -1. With the helpful percentage, we set up a UpVote % bins, by making 7 Bins, staring from ['Empty', '0-20%', '20-40%', '40-60%', '60-80%', '80-100%']. We then create a data-frame containing aggregate from review ID , group by Score and UpVote%. the below heatmap of how many reviews are helpful.

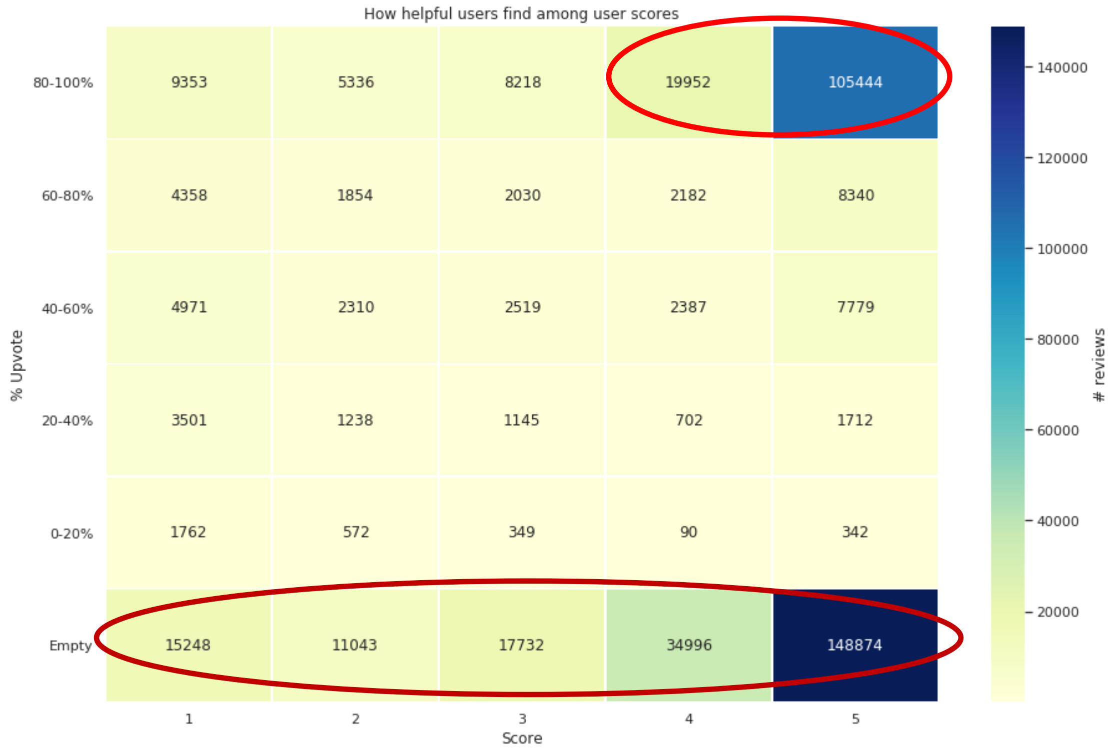


Figure 3

Observations:

- Majority of the reviews do not have any vote , as Empty Upvote% has more than 50% of reviews in it.
- Reviews with score 5, show a high helpful vote, indicating many people agreeing with the rating. The over result is skewed towards the positive

Exploration and Visualization of the "Summary+Review" Text , we are focused on the task ahead which is the review text itself on which we must process and build our model on. We started with recording the length of the summary review. We also recorded the word count of the summary review. We visualized the length of the review against each score.

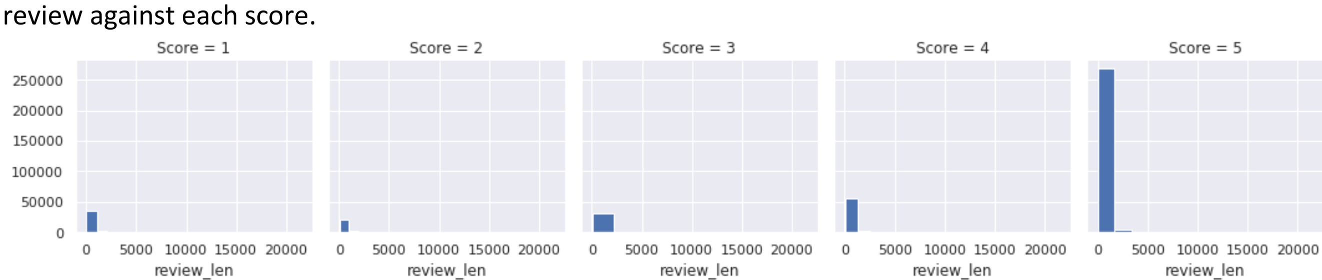


Figure 4

Observation : 5 Score reviews have long review compared to the rest.

We visualized a box plot to better understand the distribution

Observation : We can see how the overall mean of the review length is the same among the scores, but its outliners are rising towards the higher score. it is seen that Score 3; a neutral score has the max length review.

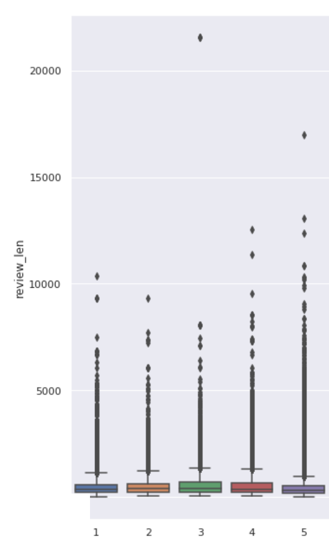


Figure 5

2. Text Processing and Normalization:

We started our text processing task by first removing the HTML tags present in the summary review text. This was done by creating a function which removes all different kinds of HTML tags. Function name : "`preprocess_htmltags`".

Outputs Screenshot:

Summary text before removing HTML tags :

```
[87] train.loc[400,'summary_text']

.ng from someone w/no green thumb, lol. highly reccomend<a href="http://www.amazon.com/gp/product/B003U2MDGO">3 Baby Staghorn Fern Plan
```

Figure 6

Summary text after removing HMTL tags:

```
[91] train.loc[400,'summary_text_pp']

'WOW!!!!!!!!!!! these plant are easy to grow coming from someone w/no green thumb, lol. highly reccomend'
```

Figure 7

We then passed this text to convert all text to lower case using the below command :

```
train['summary_text_pp_lc'] = train['summary_text_pp'].apply(lambda x: " ".join(x.lower() for x in x.split()))
```

```
[95] train.loc[400,'summary_text_pp_lc']

'wow!!!!!!!!!!! these plant are easy to grow coming from someone w/no green thumb, lol. highly reccomend'
```

Figure 8

We then wanted to check on the numerals present in the review, this could be unwanted information , as numerals would not play a major factor in review predictions.

Observations: There is not much numerals in the text, as the means is very less, the maximum number of numerals in a text is 32.

	summary_text_pp	numerics
426335	i like it Like a lot of the gums by Lotte, the...	0
426336	The Anti-Fatigue This is a fantastic product. ...	5
426337	Always the right formula I trust this brand--t...	0
426338	Smoked Black Pepper This pepper is great! I wa...	0
426339	Canidae Dog Food made my dogs extremely ill I ...	2

```
[ ] train['numerics'].describe()

count    426340.00000
mean          0.44047
std           1.06398
min           0.00000
25%           0.00000
50%           0.00000
75%           0.00000
max          32.00000
Name: numerics, dtype: float64
```

Figure 9

We then continued to be reviewing the stop words present in the text, we are using the *nltk wordnet* to filter the stops words present in our text.

```
[ ] train['count_of_stopwords'].describe()

count    426340.000000
mean         32.051121
std          31.398492
min           0.000000
25%          13.000000
50%          23.000000
75%          40.000000
max        1275.000000
Name: count_of_stopwords, dtype: float64
```

Figure 10

We then moved on to normalizing the text, we experimented with various nltk packages, below are the packages we tested the text on :

- `nltk.WordPunctTokenizer:` This package tokenizes a text into sequence of alphabetic and non-alphabetic characters using regex. (nltk, n.d.)
- `nltk.TreebankWordTokenizer:` The Treebank tokenizer uses regular expressions to tokenize text as in Penn Treebank. This method splits standard contractions, e.g. don't -> do n't, treats most punctuation characters as separate tokens, splits off commas and single quotes, when followed by whitespace and separate periods that appear at the end of line (nltk, n.d.)
- `WordNetLemmatizer:` Lemmatization is the process of converting a word to its base form. Wordnet is a large, freely and publicly available lexical database for the English language aiming to establish structured semantic relationships between words. It offers lemmatization capabilities as well and is one of the earliest and most used lemmatizers. (Prabhakaran, 2018)
  - `pos_tag:` to find out the POS tag for every word for large text, mapping it to the right input character that the `WordNetLemmatizer` accepts and passes it as a second argument to lemmatize. the `nltk.pos_tag()` method accepts a list of word and returns a tuple with the POS tag.
- `PorterStemmer:` A common algorithm used for stemming English. Its available in NLTK library. Porter Stemmer like other stemming algorithms reduces words that have the same root word . Stemming can also result in errors such as over-stemming and under-stemming .
  Over-stemming : When two words are stemmed to same root but are from different stems.
  Under-stemming :When two words are stemmed to same root that are not of different stems.
- `SnowballStemmer:` The algorithms have been developed by Martin Porter. These stemmers are called Snowball, because Porter created a programming language with this name for creating new stemming algorithms. The reason we wanted to try a different stemming option is to check if better stemming can be achieved from Snowball compared to Porter.

Using the above packages , we created 5 functions which cover various normalization techniques, such as removal of numbers, special characters \ whitespaces as well as stop words removal. below is an example on how the methods work on randomly selected review text :

	Summary Text -- Loc 1686
Original Text	10%??!?!?!?! how wrong is that???? ok, I must have missed out on some kind of product review or something.... becuase I would never have agreed to this change in rockstars juiced drinks. I had one today for the first time and all I can say is DISGUSTING!! Thanks for killing the one engery drink I actaully liked!!! I hope rockstar thinks again really soon, becuase as of this moment I am off the soft drink band wagon.
1st Prepocess	10%??!?!?!?! how wrong is that???? ok, i must have missed out on some kind of product review or something.... becuase i would never have agreed to this change in rockstars juiced drinks. i had one today for the first time and all i can say is disgusting!!thanks for killing the one engery drink i actaully liked!!!i hope rockstar thinks again really soon, becuase as of this moment i am off the soft drink band wagon.
`Tokenize_WordPunct`	wrong ok must missed kind product review something becuase would never agreed change rockstars juiced drinks one today first time say disgusting thanks killing one engery drink actaully liked hope rockstar thinks really soon becuase moment soft drink band wagon
`Tokenize_TreeBank`	wrong ok must missed kind product review something becuase would never agreed change rockstars juiced drinks one today first time say disgusting thanks killing one engery drink actaully liked hope rockstar thinks really soon becuase moment soft drink band wagon
`Tokenize_Lemmatize`	wrong ok must miss kind product review something becuase would never agree change rockstars juiced drink one today first time say disgust thanks kill one engery drink actaully like hope rockstar think really soon becuase moment soft drink band wagon
`Tokenize_Stem`	wrong ok I must miss kind product review someth becuas I would never agre chang rockstar juic drink I one today first time I say disgust thank kill one engeri drink I actaulli like I hope rockstar think realli soon becuas moment I soft drink band wagon
`Tokenize_StemSnowball`	wrong ok i must miss kind product review someth becuas i would never agre chang rockstar juic drink i one today first time i say disgust thank kill one engeri drink i actaulli like i hope rockstar think realli soon becuas moment i soft drink band wagon

Table 2

Observations:

- HTML tags / numericals / punctuations are removed
- Lemmatization effect : 'disgusting' → 'disgust' 'missed' → 'miss'
- Stemming effect : 'something' → 'someth' 'actaully' → 'actaulli'
- not much difference between the two Tokenizer methods
- not much difference between the two Stemmers

```python
def normalize_document_Tokenize_Lemmatize(review_text):
     #remove numbers, special characters\whitespaces
    review_text = str(review_text)
    review_text = re.sub(r'[^a-zA-Z\s]',' ',review_text, re.I|re.A)
    review_text = re.sub(r'[0-9]+',' ',review_text)
    review_text = review_text.strip()
    # tokenize document
    wptb = nltk.TreebankWordTokenizer()
    tokens = wptb.tokenize(review_text)
    # filter stopwords out of document
    filtered_tokens = [token for token in tokens if token not in stop_words]
    #lemmatize docment
    lemmatizer = WordNetLemmatizer()
    # Lemmatize list of words and join
    review_text = ' '.join([lemmatizer.lemmatize(w,get_wordnet_pos(w)) for w in filtered_tokens])
    return review_text
```

Figure 11

Looking at the result from the different normalization techniques, we decided to go ahead with the result set from the Tokenize_Lemmatize function.

3. Vector space Model and feature representation:

Now to convert our normalized text into bag of words and then convert it into a Vector Space Model , we used two packages from scikitlearn.
`CountVetorizer` , which converts a collection of text documents to a matrix of token counts
`TfidfVectorizer`, which converts a collection of raw documents to a matrix of TF-IDF features.

Tf-Idf method, which is known as Term frequency – Inverse document frequency, this method gives high weight to any term that appears often in a particular document, but not in very many documents, which means that the feature is going to be very descriptive of the content of that document.
As stated on the scikit learn documentation , `TfidfVectorizer` is equivalent to a `CountVectorizer` followed by a `TfidfTransformer`, our below findings have all been recorded from running `TfidfVectorizer`

min_df	max_df	ngram	Features
1(D)	1(D)	1,1(D)	99,451
5	0.5	1,1(D)	28,941
5	0.5	1,2	548,118
5	0.5	1,3	1,119,569

Table 3

Observations: With the default setting on `TfidfVectorizer`, we recorded a feature set of 99,451 features. We then played with the parameters like `min_df` which puts a condition on each feature set that it should be existing in certain minimum number of documents to be considered, default is 1, we changed it to minimum 5 documents. `max_df` parameter puts a condition on each feature to be existent only in certain maximum number of documents, default is 1, we changed to 50%. We noticed that changing the `max_df` did not result in any difference in feature set reduction. We then investigated the `n-gram range` parameter , which enables us to store more than one word (bi & tri) in the feature set, the number of features increased as we stored all the three length of features.

Understanding the features with their corresponding tfidf value :

The figure shows features are from the (1,3) n-gram setting feature set, the low tfidf are those that are either are very commonly used across documents or are only used sparingly, and only in very large documents. ‘cakesters’ seems to be one

Figure 12

```
Features with lowest tfidf:
['cells' 'consider dedication eat' 'calorie grandson consume'
 'calorie grandson' 'calorie grand scheme' 'calorie grand'
 'calorie fact nabisco' 'calorie enjoy eat' 'calorie definitely give'
 'calorie appearance nabisco' 'calorie appearance' 'call name adult'
 'cakesters would small' 'cakesters would' 'cakesters translates mini'
 'cakesters thumb believe' 'cakesters thumb' 'cakesters sign approval'
 'cakesters sign' 'cakesters present calorie']
features with hihgest tfidf:
['chimp' 'lard' 'crabmeat' 'conch' 'word word' 'trix' 'pickapeppa' 'mahi'
 'problem consistency' 'pocky' 'rice second' 'booty' 'ramune'
 'good excellent' 'ia' 'ranch' 'aaa' 'carmel' 'good' 'review']
```

such case. Looking at the highest tfidf, features like 'review' , 'good' as these are to appear more review text. We can also observe here, a feature , like 'ia' seem to be unwanted feature, which may appear more in reviews but don't produce any meaning. Highest and lowest

We can also find the words that have low inverse document frequency, i.e., those that appear too often and are this deemed less important, they are stored in `idf_` attribute. below is a screenshot of the top 100 low idf words:

```
Features with lowest idf:
['not' 'like' 'taste' 'good' 'great' 'love' 'flavor' 'one' 'get' 'product'
 'make' 'try' 'well' 'use' 'would' 'go' 'no' 'best' 'time' 'really' 'much'
 'eat' 'food' 'price' 'coffee' 'amazon' 'also' 'order' 'buy' 'give'
 'little' 'find' 'even' 'say' 'store' 'bag' 'come' 'recommend' 'tea' 'day'
 'cup' 'first' 'look' 'want' 'add' 'year' 'dog' 'delicious' 'think' 'take'
 'found' 'way' 'favorite' 'know' 'bought' 'work' 'brand' 'box' 'sweet'
 'need' 'thing' 'treat' 'purchase' 'two' 'bit' 'since' 'drink' 'could'
 'still' 'nice' 'sugar' 'free' 'lot' 'enjoy' 'small' 'water' 'bad' 'snack'
 'keep' 'many' 'ever' 'stuff' 'never' 'easy' 'seem' 'every' 'something'
 'mix' 'chocolate' 'without' 'see' 'review' 'healthy' 'perfect' 'package'
 'right' 'quality' 'always' 'high' 'old']
```

Figure 13

The above words though not in the stop word from the nltk library (exception of not and no), seem here more food-review domain specific, 'food', 'eat', 'amazon', 'taste' and 'flavor', they are deemed less important according to tf-idf measure, we can expect them to be of importance in our prediction problem.

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

The 3 Models , we decided to train and evaluate on are the following :

Logistic Regression	Multinomial NB	SGD Classifier

Modus Operand : We studied and read on the above 3 model classifiers and decided the below steps on tuning the optimal hyperparameter as well as n-gram range for each model.

- **Step1**: Using `make_pipeline` to run the `TfidfVectorizer` and one of the models selected on different conditions.
- **Step2**: Using the `param_grid` and `GridSearchCV` model selection ,keeping `cv=5` we would enter a set of different values for the selected hyperparameter as well as different n-gram range
- **Step3**: Then using `grid.fit` we would train the model with the normalized train data against the label data which will be our Score column.
- **Step4**: Output of the pipeline would give us the best cv score and the best parameters used to achieve the score. We will be also looking into the feature importance, by looking into the coefficient value against for each feature, available in the `coef_` attribute of each model classifier.
- **Step5**: Using the optimal hyperparameter setting and the n-gram range, we would the proceed to create another pipeline, this time using the `Pipeline` method to predict the train model against the test data provided and study the classification report of the predictions against the actual result.

Multinomial NB: Naïve Bayes classifier for multinomial models. It is one of the two classics naïve bayes variants used in text classification. the multinomial disturbution requires integer feature counts, but in practice, fractional integer count may also work.
We have chosen **alpha** as the hyperparamter, which by default is set to 1 known as Laplace smoothing, while anything less than 1 is known as Lidstone smoothing . (sklearn, n.d.)
Below is the result set from the 1st pipeline :
Best cv score:0.81 Best parameters: {'multinomialnb__alpha': 0.01, 'tfidfvectorizer__ngram_range': (1, 3)}

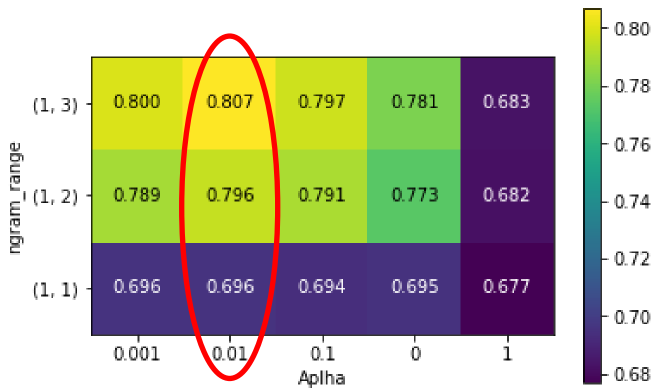


Figure 14

Below is the classification report from the above best parameters:

	precision	recall	f1-score	support
1	0.72	0.81	0.77	11611
2	0.44	0.77	0.56	4243
3	0.48	0.75	0.59	6800
4	0.46	0.73	0.56	12799
5	0.98	0.83	0.90	106661
accuracy			0.81	142114
macro avg	0.62	0.78	0.67	142114
weighted avg	0.87	0.81	0.83	142114

SGD Classifier: Liner classifiers (SVM, logistic regression) with SGD training. The estimator implements regularized linear models with Stochastic Gradient Descent (SGD) learning; the gradient of the loss is estimated each sample at a time and the model is updated along the way with a learning rate. This implementation works with data represented as dense or sparse arrays of floating-point values for the features. The model it fits can be controlled with the **loss parameter**; by default, it fits a linear support vector machine (SVM).
We have chosen **loss** as the hyperparamter, default is set to 'hinge' and the possible options are 'log' (gives logistic regression), 'modified_huber' (smooth loss that brings tolerance to outliners), 'squared_hinge' (same like hinge but is quadratically penalized), 'perceptron' (linear loss used by the perceptron algorithm)
Below is the result set from the 1st pipeline :
Best cv score:0.81 Best parameters: {'sgdclassifier__loss': 'perceptron', 'tfidfvectorizer__ngram_range': (1, 3)}

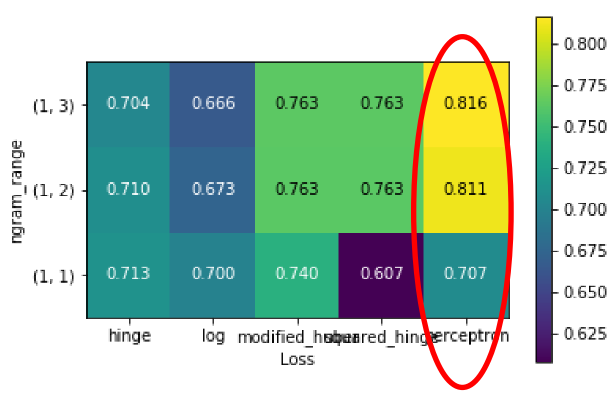


Figure 15

Below is the classification report from the above best parameters:

	precision	recall	f1-score	support
1	0.79	0.78	0.79	13214
2	0.50	0.64	0.57	5821
3	0.57	0.65	0.61	9379
4	0.57	0.65	0.61	18011
5	0.94	0.89	0.92	95689
accuracy			0.82	142114
macro avg	0.68	0.72	0.70	142114
weighted avg	0.84	0.82	0.83	142114

Logistic Regression: Logistic regression is a popular machine learning algorithm for classification problems. it is like linear regression, except it predicts whether something is true or false rather than predicting something continuous. It is available under scikit learn class as a linear model classifier. The default solver algorithm is set to 'lbfgs' which is used for multiclass problems and handles multinomial loss.
We have chosen C-ordered arrays , parameter known as **C** as the hyperparameter, by default it is set to 1, as per the sklearn documentation, smaller value specifies stronger regularization.

** While running logistic regression on the below setting, we were not able to get the output, as the execution of the pipeline was taking a lot of time, we even tried it on colab. Still the same result.

```
param_grid = {'logisticregression__C':[0.001,0.01,0.1,1,10],
              'tfidfvectorizer__ngram_range':[(1,1)(1,2)(1,3)]}
```

Thus, we took an alternate approach and ran the pipeline with smaller sets on a loop. below are the findings:

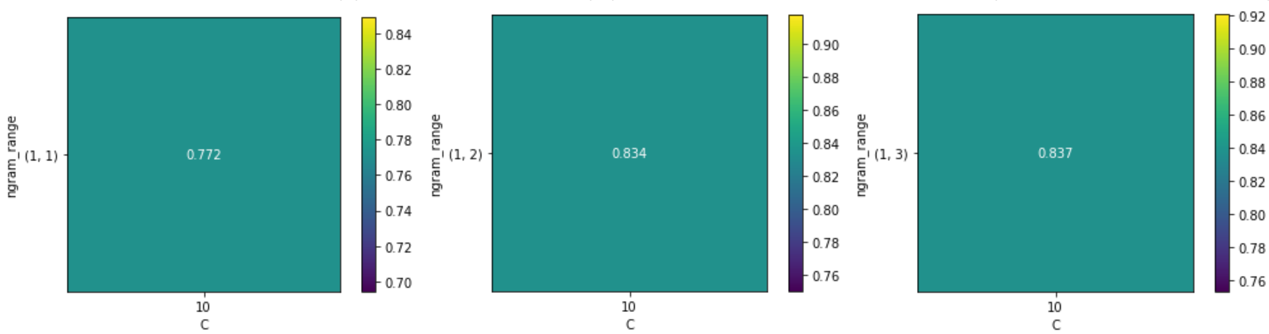


Figure 16

Below is the Test Classification report: based on n-gram rand of (1,3) and C = 10

	precision	recall	f1-score	support
1	0.82	0.80	0.81	13438
2	0.53	0.68	0.59	5754
3	0.60	0.70	0.64	9075
4	0.57	0.72	0.63	16183
5	0.96	0.89	0.92	97664
accuracy			0.84	142114
macro avg	0.70	0.76	0.72	142114
weighted avg	0.86	0.84	0.85	142114

Observations :

- The highest accuracy was achieved from Linear Regression Model at 84%
- Most processing time was taken by Linear Regression
- All models seem to predict Score 1 and 5 with an above average Precision rate, but the rest of the scores predictions are below average, this could be because of the higher number of 5 score reviews
- While training the model on different hyperparameters values , all model gave better performance with values which were not their defaults.

- All models gave optimal performance with range of (1,3) , thus leaning towards better classification with more than one word in a feature set.

5. Topic Modelling of high and low ratings*:

- As per instructions given for this task, we first separated the 1-star and the 5-star reviews separately.
- Count 1-star reviews were just 39,193 compared to the 5-star reviews count of 2,72,492 reviews. we thus decided to reduce the 5-star reviews count to 1,00,000 reviews, reduction was achieved using the `np.random.seed()method` inside the drop command for panda's data-frame to randomly drop rows from the 5 star review dataframe.
- For Topic Modelling on our two-feature set we have used a popular decomposition method called Latent Dirichlet Allocation (LDA). LDA model tries to find group of words (the topics) that appear together frequently (Müller & Guido, 2016)
- As we have an unsupervised text document model, using `CountVectorizer` feature extraction we'll remove words that appear in at atleast 15% of the documents to prevent very common words from dominating the analysis, also limit the bag-of-words model to 10000 words that are the most common after removing the top15 percent.
- We have decided to review set of 20 topics for both 1- and 5-star reviews. As per our readings about LDA , the topics don't have an inherent ordering and changing the number of topics will change all of the topics. On the parameters in LDA, we have decided to use the "batch" `learning_method`, which provides better results but takes times, compared to the default ("online") and `max_iter` to 25 which can also lead to better models.
- We used the `print_topics` function, which provides a nice formatting for these features.

All the 20 topics formed by LDA for both 1-star and 5-star reviews are seen in the Images section at the end of the report. Below are snippets of the topics we found interesting insights on :

5-Star		1-Star
topic 0		topic 9
--------		--------
chip		chip
cooky		cooky
free		can
gluten	vs	stale
salt		rancid
bag		bag
eat		case
snack		dent
cookie		cookie
potato		food

- Topic 0 for 5-star and Topic 9 for 1-Star covers same product of the like 'cooky chip' , but certain features like 'stale' 'dent' have been considered which can distinguish and make it a low-star review compared to a high-star review.
- Looking into the 5-star reviews, we can easily distinguish topics , related to certain food types, like Topic 6 is related to dog foods, topic 9 is related to tea products
- There are topics, which do not indicate any food types and just seem as random words clustered together , like topic 18
- Looking at the 1 Star reviews, here again we can easily distinguish topics related to food types and even services from amazon, like topic 3, which talks about bad product packaging while topic 4 is related to coffee product and topic 14 related to baby cereal.

6. Conclusion

We have completed all the above requirements and tried our best to gather insightful data from the various working with trials and errors. With the huge amount of text data , it was a challenge in cleaning the data and feeding relevant feature set for our model. Our biggest challenge came on logistic regression modelling , where our systems often crashed when working on hyperparameter tuning, reduction technique may have helped our case. Overall accuracy of over 80% when running the test data on all the 3 models, shows our pre-processing, normalization and tuning efforts were in the right direction.

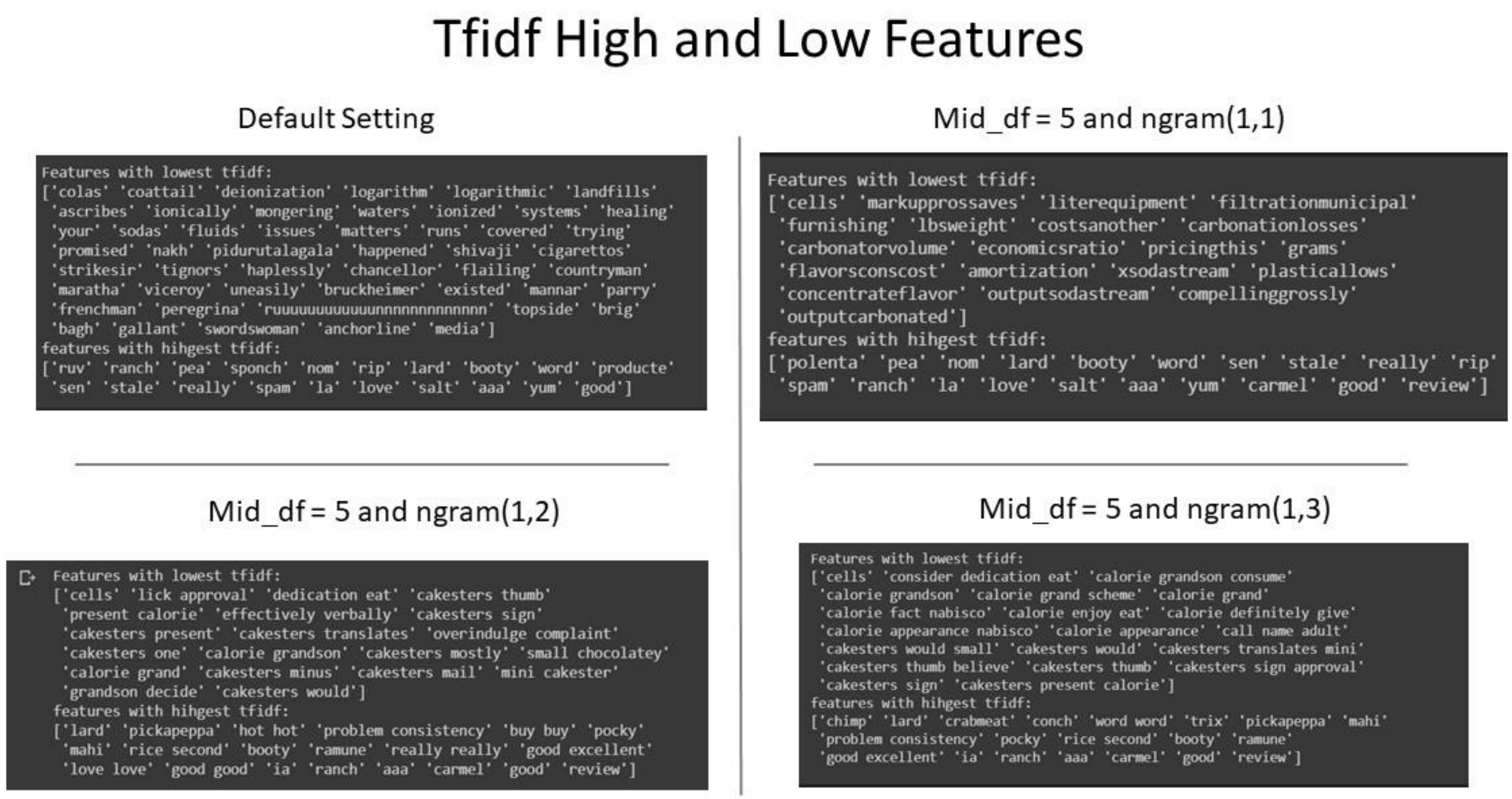
Team Members Contribution:

Juzer Mandviwala
Took the responsibility of completing Task 1 and Task 3
Shared the responsibility in completing Task 2 ,4 and 5
Created the Report (Task 6) and Data Arrangement on the share drive

Hassan Yar Khan
Shared the responsibility of completing Task2 and 4, contributed in Task 1

Sharon John
Shared the responsibility of completing Task 5 and contributed in Task 1

Images :



Tfidf High and Low Features
Default Setting
Features with lowest tfidf:
['colas' 'coattail' 'deionization' 'logarithm' 'logarithmic' 'landfills'
'ascribes' 'ionically' 'mongering' 'waters' 'ionized' 'systems' 'healing'
'your' 'sodas' 'fluids' 'issues' 'matters' 'runs' 'covered' 'trying'
'promised' 'nakh' 'pidurutalagala' 'happened' 'shivaji' 'cigarettos'
'strikesir' 'tignors' 'haplessly' 'chancellor' 'flailing' 'countryman'
'maratha' 'viceroy' 'uneasily' 'bruckheimer' 'existed' 'mannar' 'parry'
'frenchman' 'peregrina' 'ruuuuuuuuuuuunnnnnnnnnnnn' 'topside' 'brig'
'bagh' 'gallant' 'swordswoman' 'anchorline' 'media']
features with hihgest tfidf:
['ruv' 'ranch' 'pea' 'sponch' 'nom' 'rip' 'lard' 'booty' 'word' 'producte'
'sen' 'stale' 'really' 'spam' 'la' 'love' 'salt' 'aaa' 'yum' 'good']
Mid_df = 5 and ngram(1,1)
Features with lowest tfidf:
['cells' 'markupprossaves' 'literequipment' 'filtrationmunicipal'
'furnishing' 'lbsweight' 'costsanother' 'carbonationlosses'
'carbonatorvolume' 'economicsratio' 'pricingthis' 'grams'
'flavorsconscost' 'amortization' 'xsodastream' 'plasticallows'
'concentrateflavor' 'outputsodastream' 'compellinggrossly'
'outputcarbonated']
features with hihgest tfidf:
['polenta' 'pea' 'nom' 'lard' 'booty' 'word' 'sen' 'stale' 'really' 'rip'
'spam' 'ranch' 'la' 'love' 'salt' 'aaa' 'yum' 'carmel' 'good' 'review']
Mid_df = 5 and ngram(1,2)
Features with lowest tfidf:
['cells' 'lick approval' 'dedication eat' 'cakesters thumb'
'present calorie' 'effectively verbally' 'cakesters sign'
'cakesters present' 'cakesters translates' 'overindulge complaint'
'cakesters one' 'calorie grandson' 'cakesters mostly' 'small chocolatey'
'calorie grand' 'cakesters minus' 'cakesters mail' 'mini cakester'
'grandson decide' 'cakesters would']
features with hihgest tfidf:
['lard' 'pickapeppa' 'hot hot' 'problem consistency' 'buy buy' 'pocky'
'mahi' 'rice second' 'booty' 'ramune' 'really really' 'good excellent'
'love love' 'good good' 'ia' 'ranch' 'aaa' 'carmel' 'good' 'review']
Mid_df = 5 and ngram(1,3)
Features with lowest tfidf:
['cells' 'consider dedication eat' 'calorie grandson consume'
'calorie grandson' 'calorie grand scheme' 'calorie grand'
'calorie fact nabisco' 'calorie enjoy eat' 'calorie definitely give'
'calorie appearance nabisco' 'calorie appearance' 'call name adult'
'cakesters would small' 'cakesters would' 'cakesters translates mini'
'cakesters thumb believe' 'cakesters thumb' 'cakesters sign approval'
'cakesters sign' 'cakesters present calorie']
features with hihgest tfidf:
['chimp' 'lard' 'crabmeat' 'conch' 'word word' 'trix' 'pickapeppa' 'mahi'
'problem consistency' 'pocky' 'rice second' 'booty' 'ramune'
'good excellent' 'ia' 'ranch' 'aaa' 'carmel' 'good' 'review']

5-Star Review Topics

topic 0	topic 1	topic 2	topic 3	topic 4	topic 5	topic 6	topic 7	topic 8	topic 9
chip	sauce	snack	low	ingredient	mix	treat	bread	coffee	tea
cooky	add	bar	calorie	organic	milk	dog	year	cup	green
free	hot	eat	fat	natural	add	chew	syrup	popcorn	bag
gluten	soup	cereal	protein	no	vanilla	small	ever	pod	drink
salt	rice	healthy	sugar	health	powder	give	family	machine	cup
bag	chicken	fruit	diet	help	cream	teeth	back	keurig	black
eat	salt	delicious	pasta	high	cake	size	find	espresso	chai
snack	spice	nut	weight	food	recipe	training	found	brew	favorite
cookie	food	tasty	high	formula	sugar	bone	ago	starbucks	iced
potato	spicy	oatmeal	bean	take	ice	toy	home	maker	hot

topic 10	topic 11	topic 12	topic 13	topic 14	topic 15	topic 16	topic 17	topic 18	topic 19
chocolate	price	coffee	order	food	butter	drink	amazon	time	oil
candy	quality	cup	bag	cat	peanut	water	store	day	coconut
sweet	brand	roast	box	dog	cracker	sugar	find	give	gum
delicious	save	strong	package	eat	pumpkin	bottle	local	take	jerky
hot	amazon	blend	arrive	feed	ginger	energy	order	eat	smell
dark	buy	bold	receive	dry	pie	no	price	say	hair
cocoa	organic	favorite	come	year	jelly	juice	buy	would	skin
bar	excellent	dark	time	no	pb	work	grocery	first	dry
ever	value	smooth	would	old	jar	day	found	know	also
treat	cost	bitter	gift	brand	bear	really	carry	thing	olive

1-Star Review Topics

topic 0	topic 1	topic 2	topic 3	topic 4	topic 5	topic 6	topic 7	topic 8	topic 9
chocolate	work	sugar	box	coffee	price	food	food	eat	chip
candy	hair	ingredient	bag	cup	store	eat	change	bar	cooky
popcorn	cake	free	open	roast	pack	day	new	review	can
look	bread	syrup	package	pod	oz	cat	old	give	stale
gift	plant	artificial	packaging	weak	cost	thing	year	really	rancid
disappointed	red	natural	amazon	bean	amazon	feed	formula	love	bag
nut	oil	list	broken	keurig	local	always	month	awful	case
seed	mix	label	plastic	blend	grocery	three	back	peanut	dent
melt	color	gluten	arrive	bitter	much	whatever	switch	texture	cookie
small	time	sweetener	seal	ground	box	really	dog	cheese	food

topic 10	topic 11	topic 12	topic 13	topic 14	topic 15	topic 16	topic 17	topic 18	topic 19
salt	dog	fruit	money	baby	milk	water	cat	tea	amazon
jerky	treat	soup	waste	cereal	sauce	drink	food	green	receive
meat	china	sweet	coconut	organic	hot	bottle	ingredient	bag	item
eat	give	juice	date	rice	powder	smell	eat	chai	company
beef	chew	dry	water	best	add	take	corn	leaf	return
noodle	eat	organic	old	trap	mix	first	oil	black	say
salty	bone	apple	stuff	food	sugar	say	chicken	cinnamon	customer
food	food	tomato	smell	earth	water	think	diet	ginger	purchase
dog	pet	bean	stale	formula	brand	work	grain	cup	never
chicken	sick	potato	horrible	child	sweet	could	fat	licorice	back

References

Anon., 2020. *Amazon Fine Food Reviews.* [Online]
Available at: https://www.kaggle.com/snap/amazon-fine-food-reviews
DLao, 2019. *Amazon fine food review - Sentiment analysis.* [Online]
Available at: https://www.kaggle.com/laowingkin/amazon-fine-food-review-sentiment-analysis
[Accessed 20 Feburary 2020].
Müller, A. C. & Guido, S., 2016. *Introduction to Machine Learning with Python: a guide for data scientists.* Sebastopol, California: O'Reilly Media .
nltk, n.d. *nltk.tokenize package.* [Online]
Available at: https://www.nltk.org/api/nltk.tokenize.html
Prabhakaran, S., 2018. *Lemmatization Approaches with Examples in Python.* [Online]
Available at: https://www.machinelearningplus.com/nlp/lemmatization-examples-python/
[Accessed 10 Feburary 2020].
sklearn, n.d. *Naive Bayes.* [Online]
Available at: https://scikit-learn.org/stable/modules/naive_bayes.html#multinomial-naive-bayes
[Accessed 20 Feburary 2020].