NLP - Sentiment Analysis Project
Beauty Product Reviews

Engin Turkmen

April 15, 2019

Contents

1. INTRODUCTION

1.1. General:

Sentiment analysis ,which is a subtopics of Natural Language Processing (NLP), has been gradually becoming more and more popular. It is a contextual mining of text which identifies and extracts subjective information in source material and helping a business to understand the social sentiment of their brand, product or service while monitoring online conversations.

Sentiment Analysis has many applications ranging from ecommerce, marketing, to politics and any other research to tackle with text or unstructured text data. Companies, especially in e-commerce, also do sentiment analysis to collect and analyze customer feedback about their products. Besides that, potential customers prefer to review the opinions of existing customers before they purchase a product or use a service of a company. As seen here, there are two parts in e-commerce; one is the online retailer, which wants to maximize e-commerce sales or services, and the other is the consumers, who want to have the best product or service over alternatives.

1.2 Problem:

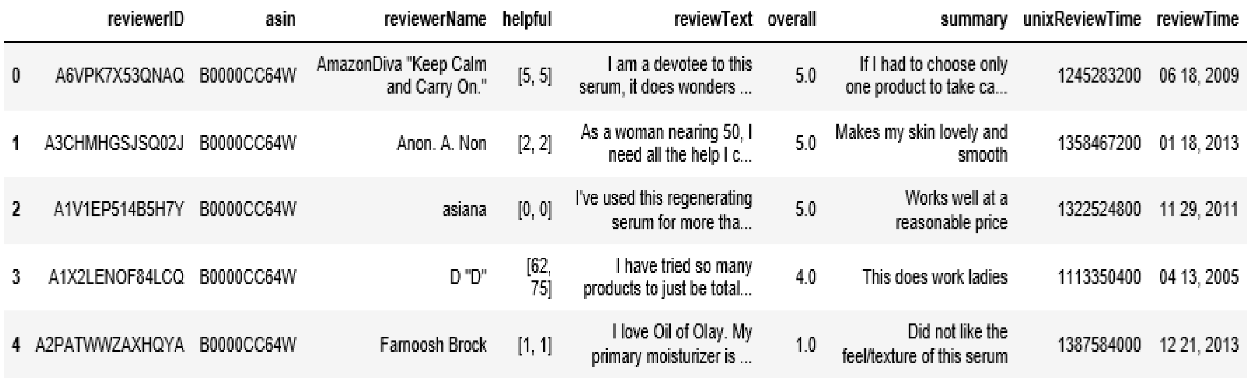
In this project, Amazon is our client. The company wants to develop a software tool that will identify the positive and negative words which customers use when they write reviews for the beauty products as their purchase inclination. For that, they gave their 9 years beauty products' reviews between 2005-2014 and asked us to develop a model which will identify positive and negative words used in the reviews as a component of customer's sentiment towards to the company's beauty products.

According to the customer request, we will build a sentiment analysis model as part of natural language processing, based on their reviews on the beauty product online purchases. Our dataset consists mainly of customers' reviews and ratings.

1.3 Data Set Description:

Beauty dataset revolving around the reviews written by customers. This is a real commercial data.

This data includes 28798 rows and 9 feature variables. Memory usage is 2.2+ MB.



	reviewerID	asin	reviewerName	helpful	reviewText	overall	summary	unixReviewTime	reviewTime
0	A6VPK7X53QNAQ	B0000CC64W	AmazonDiva "Keep Calm and Carry On."	[5, 5]	I am a devotee to this serum, it does wonders ...	5.0	If I had to choose only one product to take ca...	1245283200	06 18, 2009
1	A3CHMHGSJSQ02J	B0000CC64W	Anon. A. Non	[2, 2]	As a woman nearing 50, I need all the help I c...	5.0	Makes my skin lovely and smooth	1358467200	01 18, 2013
2	A1V1EP514B5H7Y	B0000CC64W	asiana	[0, 0]	I've used this regenerating serum for more tha...	5.0	Works well at a reasonable price	1322524800	11 29, 2011
3	A1X2LENOF84LCQ	B0000CC64W	D "D"	[62, 75]	I have tried so many products to just be total...	4.0	This does work ladies	1113350400	04 13, 2005
4	A2PATWWZAXHQYA	B0000CC64W	Farnoosh Brock	[1, 1]	I love Oil of Olay. My primary moisturizer is ...	1.0	Did not like the feel/texture of this serum	1387584000	12 21, 2013

Each row corresponds to a customer review, and includes the variables:

reviewerID : ID of the reviewer, e.g. A2SUAM1J3GNN3B - type: object

asin : ID of the product , e.g. 0000013714 – type: object

reviewerName : name of the reviewer – type: object

helpful : helpfulness of the review, e.g. 2/3 – type: object

reviewText : text of the review – type: object

overall : Rating (1,2,3,4,5)– type: float64

summary : summary of the review – type: object

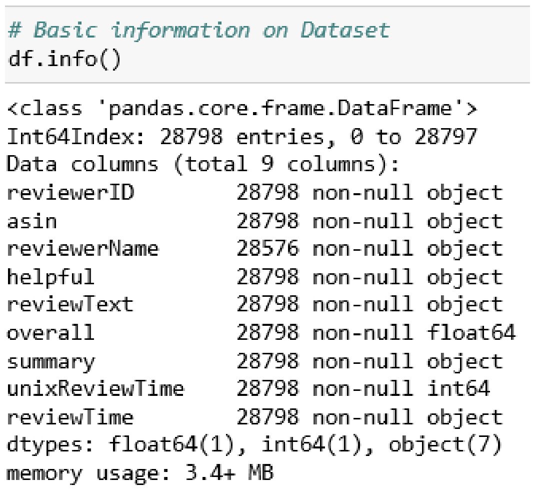
unixReviewTime : time of the review (unix time) – type: int64

reviewTime : time of the review (raw) – type: object

The data was in Standford Analysis Project webpage. The original data was in a JSON format there. In order to analyze the data, I should change the data format. For that, I import JSON and decode JSON file with using query in order to convert JSON file to csv file format.
Data Source:
http://snap.stanford.edu/data/amazon/productGraph/categoryFiles/reviews_Beauty_10.json.gz

2. DATA WRANGLING

2.1 Inspecting the Data Set:



```
# Basic information on Dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 28798 entries, 0 to 28797
Data columns (total 9 columns):
reviewerID        28798 non-null object
asin              28798 non-null object
reviewerName      28576 non-null object
helpful           28798 non-null object
reviewText        28798 non-null object
overall           28798 non-null float64
summary           28798 non-null object
unixReviewTime    28798 non-null int64
reviewTime        28798 non-null object
dtypes: float64(1), int64(1), object(7)
memory usage: 3.4+ MB
```

Amazon beauty products data includes 28798 rows(observations) and 9 columns(feature variables) and its memory usage is 3.4+ MB. In the dataset, we have 7 object, 1 float64 and 1 int64 data types.

222 **'reviewerName'** information is missing in the dataset. Since customers don't give their identity, it may not be reliable to make an analysis on their reviews and ratings. I would prefer to drop the missing values from dataset since we have enough observations to conclude a prediction for sentiment analysis.

We concatenated **'reviewText'** and **'summary'** since both gave the approximately same type of information about product in text format, and later dropped both **'reviewText'** and **'summary'** columns.

'helpful' feature was dropped since we didn't need that column for our model.

We classified the **'overall'** (ratings) as good and bad in order to make sentiment analysis. For that, we dropped observations whose 'overall' columns' values are equal to 3 since that rating group doesn't give an exact opinion about product whether it is good or bad. We created a new column named as 'rate_class' from 'overall' column and converted its' values as 'good' and 'bad'. Later, we dropped 'overall' column.

In the dataset, **'reviewerID'** and **'reviwerName'** were used both for identification of customers. We dropped one of them from the dataset. Preferably, I dropped **'reviewerName'** since customer names were not standardized and there were lots of different style to represent them in it.

'unixReviewTime' was dropped since it has already been represented in **'reviewTime'** feature in a more understandable format. Also, **'reviewTime'** was converted to datetime data type and a new **'year'** column was created to make analysis between other variables in the future work. After that, 'reviewTime' column was also dropped.

We renamed the columns in order to improve practicality/readability of coding:

reviewerID : **"customer"**

asin : **"product"**

reviewText: This will be concatenated with "summary" and renamed as **"review_text"**

overall: **"rating_class"**

reviewTime: **"year"**

2.2 Descriptive Statistics:

In our dataset, we have 2084 reviews which have bad ratings whereas 22425 reviews which have good ratings.

We have 1340 unique customers and 733 products in this dataset. Each customer averagely gives 18 reviews for products and on the other hand, there is averagely 34 reviews for each product in the website.

2.3 Preprocessing the Text:

Since, text is the most unstructured form of all the available data, various types of noise are present in it and the data is not readily analyzable without any pre-processing. The entire process of cleaning and standardization of text, making it noise-free and ready for analysis is known as text preprocessing. In this section, I apply the following text preprocessing respectively.

Removing HTML tags

We wrote a function to remove the HTML tags which typically does not add much value towards understanding and analyzing text.

Removing accented characters

We wrote a function to convert and standardize accented characters/letters into ASCII characters.

Expanding Contractions

We wrote a function to convert each contraction to its expanded, original form in order to help with text standardization.

Removing Special Characters

We used simple regular expressions(regexes) to remove special characters and symbols which are usually non-alphanumeric characters or even occasional numeric characters.

Lemmatization

We removed word affixes to get to the base form of a word, known as root word.

Removing stopwords

We wrote a function to remove stopwords which have little or no significance in the text.

Building a Text Normalizer

Based on the functions which we have written above and with additional text correction techniques (such as lowercase the text, and remove the extra newlines, white spaces, apostrophes), we built a text normalizer in order to help us to preprocess the new_text document.

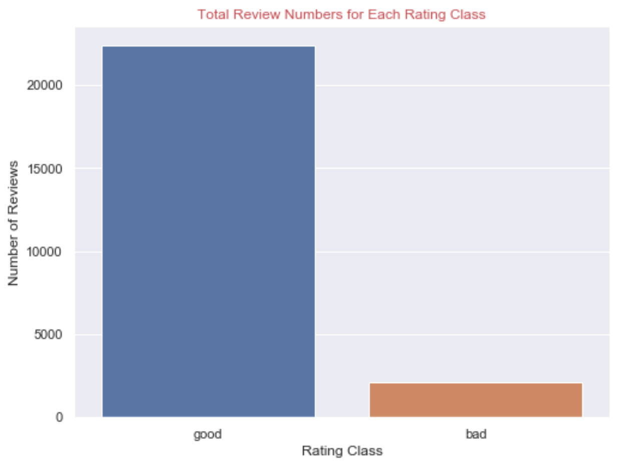
After applying text normalizer to 'the review_text' document, we applied tokenizer to create tokens for the clean text. As a result of that, we had 1706537 words in total with a vocabulary size of 25023. Max review length is 1090 whereas min review length is 1 as a word based.

Eventually, after completing all data wrangling and preprocessing phases, we save the dataframe to csv file as a 'cleaned_dataset'.

A clean dataset will allow a model to learn meaningful features and not overfit on irrelevant noise. After following these steps and checking for additional errors, we can start using the clean, labelled data to train models in modeling section.

3. DATA STORYTELLING

3.1 Target Variable ("rating_class")

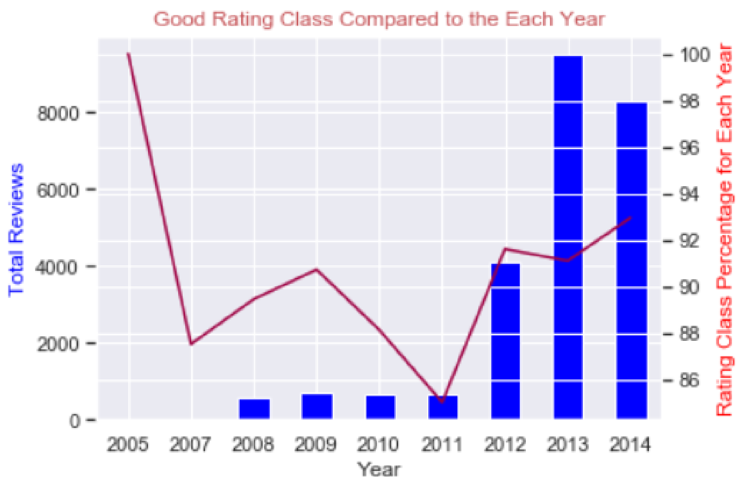


Customers wrote reviews and gave ratings, which ranged between 1 to 5, for each beauty product they bought in the Amazon online market between 2005 and 2014. In overall, customers were seemed to be averagely satisfied with the products they purchased.

We diminished those 5 rating categories into two categories such as 'good' and 'bad' in order to develop a sentiment analysis model based on their reviews. According to those reviews, 91.5% of them (22425) are classified as good, whereas 8.5% of them (2084) are bad.

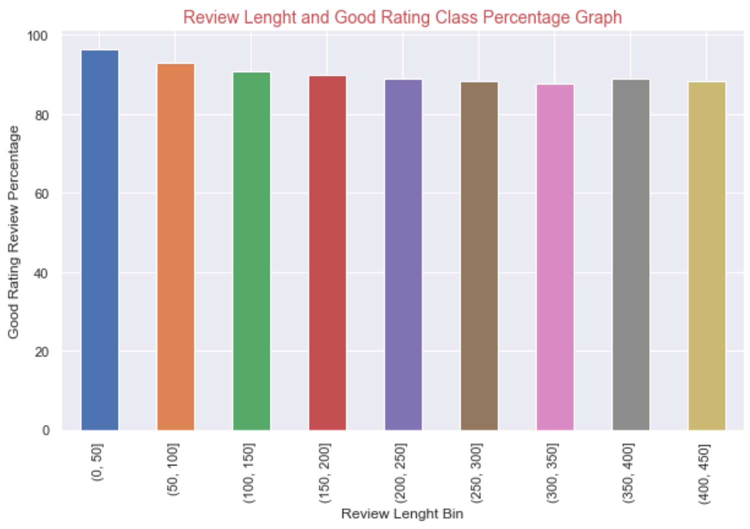
3.2 Features

3.2.1 "Year" Feature



After 2012, good ratings' percentage is progressing over 90%. Before 2012, only 2009 also shows a slightly rapid increase in good ratings from 87.5% to 90.7%. Besides those, 2011 has the lowest good ratings with 85% overall. As it might be seen in the graph, the overall good rating is progressing between 85% and 93% in beauty products.

3.2.2 "Review Length" Feature



As it might be seen the graph, the highest percentage of good rating reviews lies between 0-50 bin with 96.4% whereas lowest percentage of good rating reviews lies between 400-450 bin with 88.3%. As the review length extends, the good rating tends to decrease slightly. The difference between good rating review percentage with shortest and longest of review length bin is 8.1%. Insightfully, the customers who have complains about the products are more willingly to write longer reviews than other customers who are satisfied with company's products.

3.2.3 “Text Review” Feature

Good Rating Words:

Words	Avg.	Words	Avg.	Words	Avg.	Words	Avg.
rosehip	1.0000	hyaluronic	0.9838	combo	0.9788	additional	0.9753
shany	1.0000	fantastic	0.9836	nail	0.9787	importantly	0.9750
adovia	1.0000	tree	0.9834	dropper	0.9786	amazing	0.9742
instanatural	0.9945	wonderfully	0.9832	win	0.9784	complimentary	0.9741
palette	0.9932	awesome	0.9829	gym	0.9782	reasonably	0.9739
express	0.9918	shipping	0.9828	compliment	0.9780	scar	0.9739
mud	0.9908	gift	0.9813	highly	0.9777	men	0.9737
mencare	0.9903	sea	0.9811	vera	0.9776	handy	0.9735
cucumber	0.9903	bonus	0.9810	youthful	0.9772		
pleasantly	0.9879	heel	0.9809	exchange	0.9764		
relief	0.9862	economical	0.9807	winner	0.9760		
excellent	0.9849	buildup	0.9803	polish	0.9757		
eczema	0.9842	conjuction	0.9801	bath	0.9754		
incredible	0.9840	suprisingly	0.9801	pleased	0.9754		

Fixing the rating count value is above 100, the most common 50 words which belong to good rating class are shown in the table above. Each of these words define which products what kind of good impression have on the customers. For example, ‘mencare’ and ‘men’ words tell male beauty products are more appreciated in the reviews. On the other hand, ‘eczema’, and ‘scar’ tell some beauty products are praised for covering them. ‘economical’ and ‘shipping’ words might give the insight that products are accepted as reasonably priced and conveniently shipped to the customers.

Bad Rating Words:

Words	Avg.	Words	Avg.	Words	Avg.	Words	Avg.
sorry	0.4818	wax	0.7168	perhaps	0.7617	understand	0.7860
disappointed	0.5154	john	0.7169	skip	0.7627	ok	0.7863
unfortunately	0.5365	throw	0.7277	marketing	0.7642	intend	0.7865
fail	0.5865	glove	0.7298	direct	0.7658	shake	0.7867
attempt	0.6261	barrel	0.7401	dirty	0.7703	african	0.7867
awful	0.6328	wand	0.7401	gross	0.7714	possibly	0.7870
terrible	0.6328	weird	0.7412	initially	0.7737	odd	0.7877
horrible	0.6643	consumer	0.7445	waste	0.7757	idea	0.7878
toss	0.6730	frieda	0.7452	nothing	0.7785		
hop	0.6827	direction	0.7478	doubt	0.7800		
awkward	0.6959	miss	0.7500	okay	0.7812		
american	0.7045	description	0.7553	hope	0.7822		
whatsoever	0.7075	maybe	0.7583	instruction	0.7847		
impossible	0.7129	battery	0.7611	strange	0.7857		

Same standards as above, the most common 50 words which belong to bad rating class are shown in this table. Likewise, in good ratings, each of these words define which products what kind of bad impression have on the customers. For example, ‘wax’ products have mostly used to complain. ‘description’ word gives insight that the product’s usage is not clearly depicted in the description or beauty product has side effects which the description fails to explain.

Controversial Cases

The controversial case such as "I was expecting better - negative meaning" or "it was better than my expectation - positive meaning " will be handled in the modelling section via using deep learning technique (Keras with Word2Vec).

4. NATURAL LANGUAGE PROCESSING

4.1 Feature Engineering and Selection

Machine Learning models take numerical values as input. Our dataset is a list of sentences, so in order for our algorithm to extract patterns from the data, we first need to find a way to represent it in a way that our algorithm can understand, i.e. as a list of numbers.

We implemented CounterVectorizer, TF-IDF, Hashing Vectorizer, Word2Vec, adding most common words into the stopwords list, SMOTE, PCA, and Truncated SVD techniques into our classification models in the following sections as a part of feature engineering and selection.

4.2 Data Preprocessing

Separate response variable and features:

Firsts, we separated features(clean text) and response variable(rating class) and as X and y respectively.

Splitting the dataset into the Training set and Test set:

Then, we divided the dataset into the training and test sets. We have used 25% of dataset as testing set and 75% of the dataset as the training set. We also set the random state of the split to ensure consistent results.

4.3 Selecting the Right Evaluation Metric

Since we have a data imbalance in our case, the evaluation of the classifier performance must be carried out using adequate metrics in order to consider the class distribution and to pay more attention to the minority class. Based on this thought we used f1 score which is harmonic average of precision and recall as my evaluation metric.

Understanding the types of errors our model makes, and least desirable are important . A good way to visualize that information is using a Confusion Matrix, which compares the predictions our model makes with the true label. With that in mind, we used confusion matrix besides our evaluation metric (f1 score).

4.4 Modeling

This is a supervised binary classification problem. We are trying to predict the sentiment based on the reviews left by females who bought beauty products in Amazon e-commerce online platform. We used Python's Scikit Learn libraries to solve the problem. In this context, we implemented Logistic Regression, Random Forest, Naive Bayes, XGBOOST, CatBoost algorithms and Simple Neural Network as well.

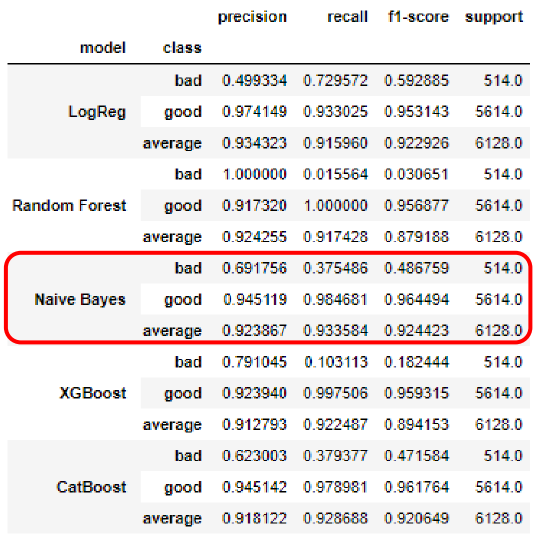
Since the ratings of the reviews were not distributed normally, we decided to decrease rating classes from 5 to 2 by merging Rating 1-2 as 'Bad' and Rating 4-5 as 'Good' while dropping Rating 3 from the dataset.

For feature selection, we applied threshold for word occurrence with using min_df/max_df, PCA and Singular Value Decomposition. For feature engineering, we applied CountVectorizer, TF-IDF, Hashing Vectorizer and Word2Vec to the text data in order to turn a collection of text documents into numerical feature vectors.

Before start modeling, we looked the dummy classifier f1 score. This classifier is useful as a simple baseline to compare with other real classifiers. It is equal to 0.84. It means that randomly selection will yield this score.

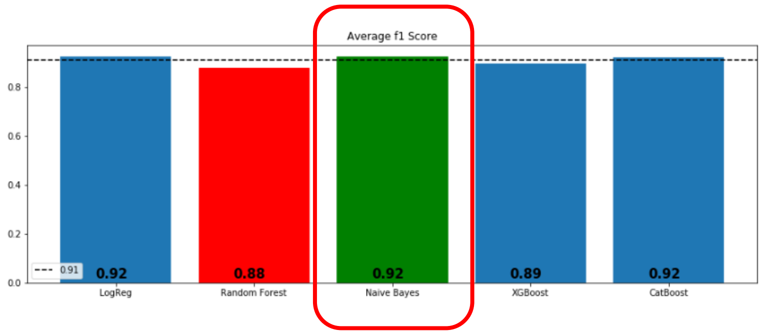
4.4.1 Count Vectorizer

We call **vectorization** the general process of turning a collection of text documents into numerical feature vectors. This specific strategy (tokenization, counting and normalization) is called the **Bag of Words** or “Bag of n-grams” representation. Documents are described by word occurrences while completely ignoring the relative position information of the words in the document. **"CountVectorizer"** implements both tokenization and occurrence counting in a single class.



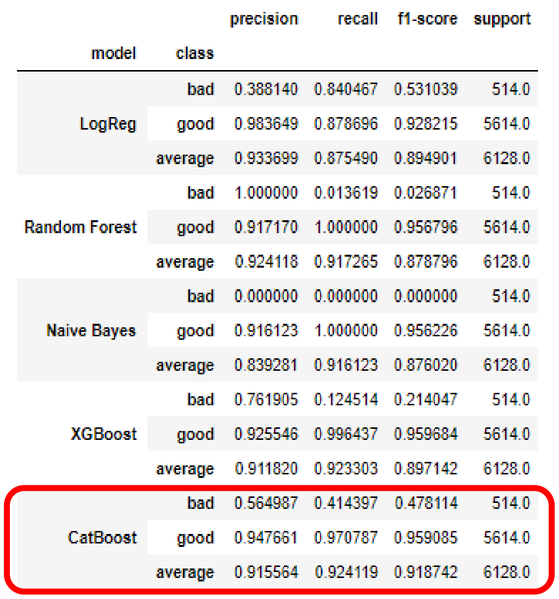
		precision	recall	f1-score	support
model	class				
LogReg	bad	0.499334	0.729572	0.592885	514.0
	good	0.974149	0.933025	0.953143	5614.0
	average	0.934323	0.915960	0.922926	6128.0
Random Forest	bad	1.000000	0.015564	0.030651	514.0
	good	0.917320	1.000000	0.956877	5614.0
	average	0.924255	0.917428	0.879188	6128.0
Naive Bayes	bad	0.691756	0.375486	0.486759	514.0
	good	0.945119	0.984681	0.964494	5614.0
	average	0.923867	0.933584	0.924423	6128.0
XGBoost	bad	0.791045	0.103113	0.182444	514.0
	good	0.923940	0.997506	0.959315	5614.0
	average	0.912793	0.922487	0.894153	6128.0
CatBoost	bad	0.623003	0.379377	0.471584	514.0
	good	0.945142	0.978981	0.961764	5614.0
	average	0.918122	0.928688	0.920649	6128.0

Logistic regression is the winner with 0.922926.



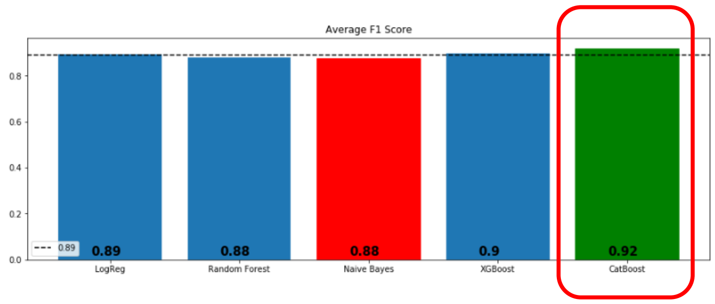
4.4.2 TF-IDF Vectorizer

In order to help our model focus more on meaningful words, we can use a TF-IDF score (Term Frequency, Inverse Document Frequency) on top of our Bag of Words model. TF-IDF weighs words by how rare they are in our dataset, discounting words that are too frequent and just add to the noise. TF-IDF works by penalizing these common words by assigning them lower weights while giving importance to words which appear in a subset of a particular document.



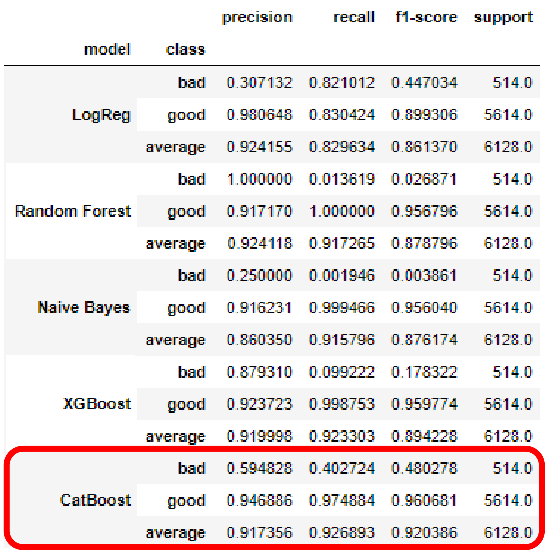
		precision	recall	f1-score	support
model	class				
LogReg	bad	0.388140	0.840467	0.531039	514.0
	good	0.983649	0.878696	0.928215	5614.0
	average	0.933699	0.875490	0.894901	6128.0
Random Forest	bad	1.000000	0.013619	0.026871	514.0
	good	0.917170	1.000000	0.956796	5614.0
	average	0.924118	0.917265	0.878796	6128.0
Naive Bayes	bad	0.000000	0.000000	0.000000	514.0
	good	0.916123	1.000000	0.956226	5614.0
	average	0.839281	0.916123	0.876020	6128.0
XGBoost	bad	0.761905	0.124514	0.214047	514.0
	good	0.925546	0.996437	0.959684	5614.0
	average	0.911820	0.923303	0.897142	6128.0
CatBoost	bad	0.564987	0.414397	0.478114	514.0
	good	0.947661	0.970787	0.959085	5614.0
	average	0.915564	0.924119	0.918742	6128.0

Catboosting is the winner with 0.918742 score.



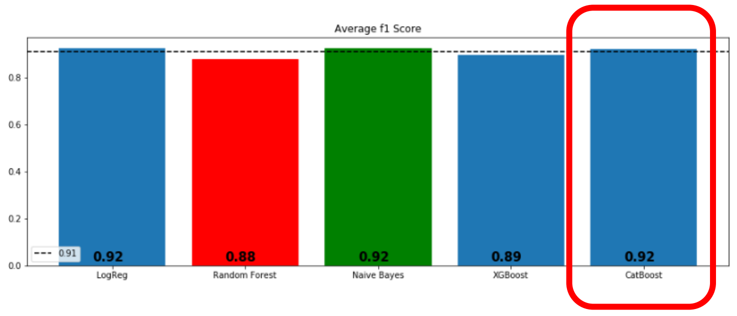
4.4.3 Hash Vectorizer

Hash Vectorizer is designed to be as memory efficient as possible. Instead of storing the tokens as strings, the vectorizer applies the hashing trick to encode them as numerical indexes. The downside of this method is that once vectorized, the features' names can no longer be retrieved.



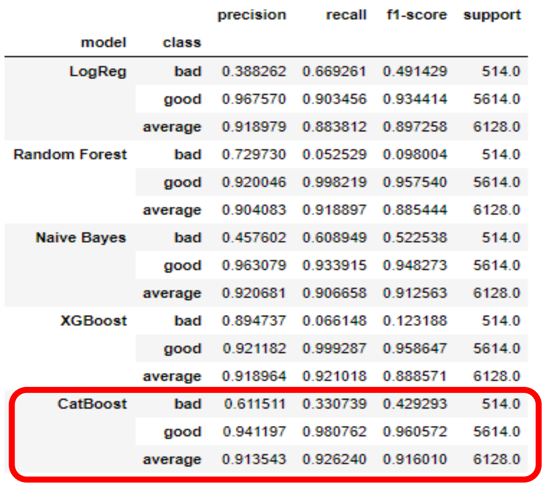
model	class	precision	recall	f1-score	support
LogReg	bad	0.307132	0.821012	0.447034	514.0
	good	0.980648	0.830424	0.899306	5614.0
	average	0.924155	0.829634	0.861370	6128.0
Random Forest	bad	1.000000	0.013619	0.026871	514.0
	good	0.917170	1.000000	0.956796	5614.0
	average	0.924118	0.917265	0.878796	6128.0
Naive Bayes	bad	0.250000	0.001946	0.003861	514.0
	good	0.916231	0.999466	0.956040	5614.0
	average	0.860350	0.915796	0.876174	6128.0
XGBoost	bad	0.879310	0.099222	0.178322	514.0
	good	0.923723	0.998753	0.959774	5614.0
	average	0.919998	0.923303	0.894228	6128.0
CatBoost	bad	0.594828	0.402724	0.480278	514.0
	good	0.946886	0.974884	0.960681	5614.0
	average	0.917356	0.926893	0.920386	6128.0

Catboosting is the winner with 0.920386 score.



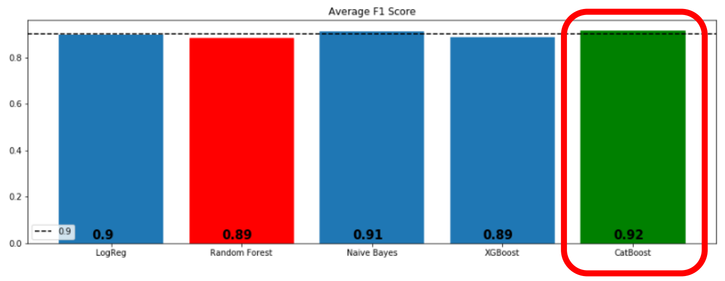
4.4.4 Adding Most Common and Lest Common Words to Stopwords List (Count Vectorizer)

Since there were not too many distinguisher words in different classes, the most and least common 70 words added to the stopwords list and models were applied in order to see any changes in evaluation metrics.



model	class	precision	recall	f1-score	support
LogReg	bad	0.388262	0.669261	0.491429	514.0
	good	0.967570	0.903456	0.934414	5614.0
	average	0.918979	0.883812	0.897258	6128.0
Random Forest	bad	0.729730	0.052529	0.098004	514.0
	good	0.920046	0.998219	0.957540	5614.0
	average	0.904083	0.918897	0.885444	6128.0
Naive Bayes	bad	0.457602	0.608949	0.522538	514.0
	good	0.963079	0.933915	0.948273	5614.0
	average	0.920681	0.906658	0.912563	6128.0
XGBoost	bad	0.894737	0.066148	0.123188	514.0
	good	0.921182	0.999287	0.958647	5614.0
	average	0.918964	0.921018	0.888571	6128.0
CatBoost	bad	0.611511	0.330739	0.429293	514.0
	good	0.941197	0.980762	0.960572	5614.0
	average	0.913543	0.926240	0.916010	6128.0

Catboosting is the winner with 0.916010 score.

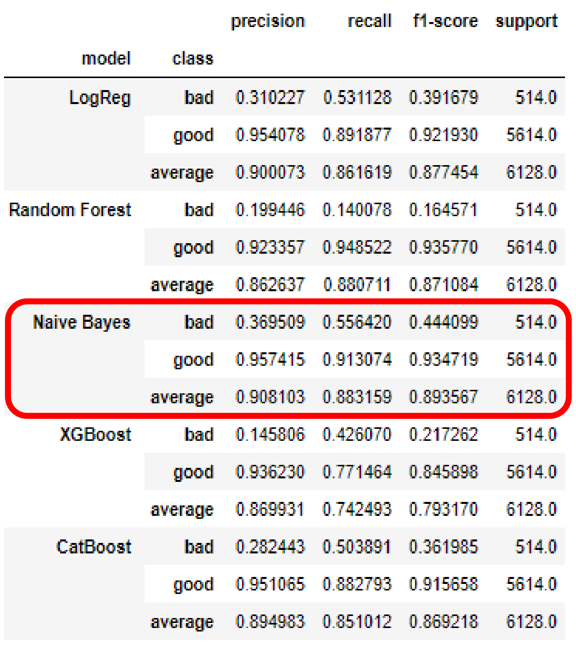


Adding most and least common words to the stopword list didn't have impact on models' performance.

4.4.5 Synthetic Minority Oversampling Technique (SMOTE)

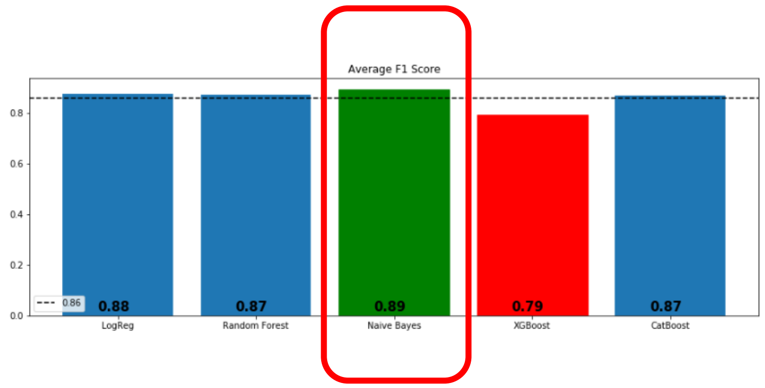
Since we have already noted the imbalance in the values within the target variable, let us implement the SMOTE method in the dealing with this skewed value in order to see whether we may improve our accuracy score.

After SMOTE mechanism to improve target class imbalance and identifying best hyper-parameters, f1 score of our model did not show an improvement and decreased to 0.8276. Besides that, we should keep in mind that oversampling will generate artificial observations which may be tricky to evaluate the accuracy of the model.



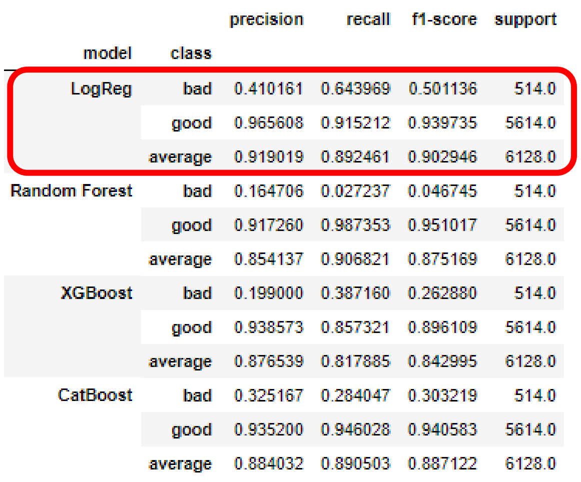
model	class	precision	recall	f1-score	support
LogReg	bad	0.310227	0.531128	0.391679	514.0
	good	0.954078	0.891877	0.921930	5614.0
	average	0.900073	0.861619	0.877454	6128.0
Random Forest	bad	0.199446	0.140078	0.164571	514.0
	good	0.923357	0.948522	0.935770	5614.0
	average	0.862637	0.880711	0.871084	6128.0
Naive Bayes	bad	0.369509	0.556420	0.444099	514.0
	good	0.957415	0.913074	0.934719	5614.0
	average	0.908103	0.883159	0.893567	6128.0
XGBoost	bad	0.145806	0.426070	0.217262	514.0
	good	0.936230	0.771464	0.845898	5614.0
	average	0.869931	0.742493	0.793170	6128.0
CatBoost	bad	0.282443	0.503891	0.361985	514.0
	good	0.951065	0.882793	0.915658	5614.0
	average	0.894983	0.851012	0.869218	6128.0

Naïve Bayes is the winner with 0.893567.



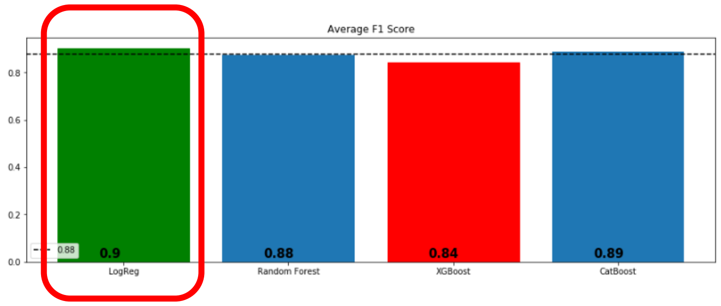
4.4.6 Applying PCA to Decrease the Linear Dimensionality + SMOTE

We have used Principal Component Analysis (PCA) which is a dimension-reduction tool and reduces a large set of variables to a small set that still contains most of the information in the dataset.



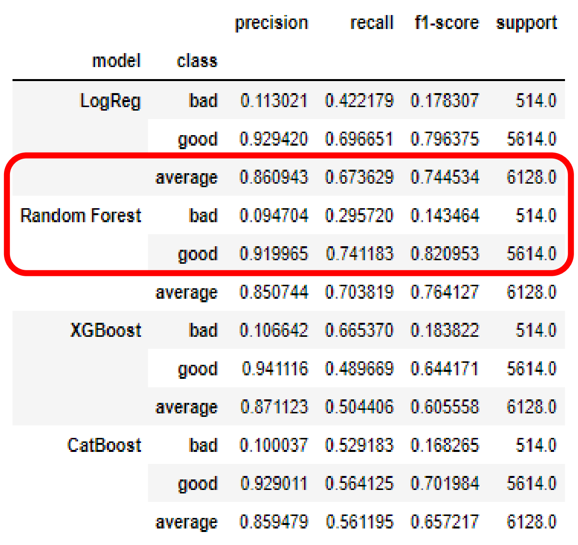
model	class	precision	recall	f1-score	support
LogReg	bad	0.410161	0.643969	0.501136	514.0
	good	0.965608	0.915212	0.939735	5614.0
	average	0.919019	0.892461	0.902946	6128.0
Random Forest	bad	0.164706	0.027237	0.046745	514.0
	good	0.917260	0.987353	0.951017	5614.0
	average	0.854137	0.906821	0.875169	6128.0
XGBoost	bad	0.199000	0.387160	0.262880	514.0
	good	0.938573	0.857321	0.896109	5614.0
	average	0.876539	0.817885	0.842995	6128.0
CatBoost	bad	0.325167	0.284047	0.303219	514.0
	good	0.935200	0.946028	0.940583	5614.0
	average	0.884032	0.890503	0.887122	6128.0

Logistic Regression is the winner with 0.9202946.



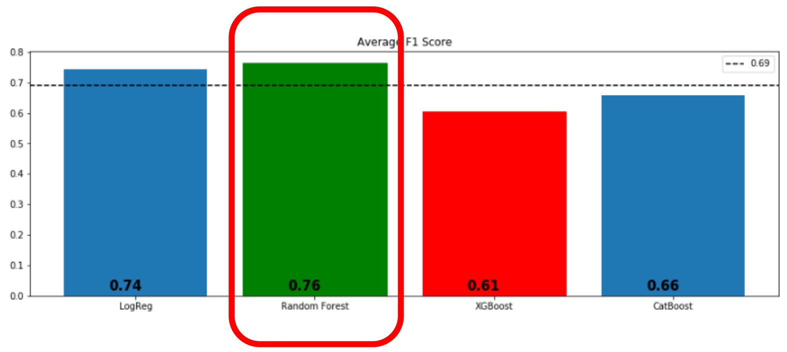
4.4.7 Truncated SVD + SMOTE

Since the dimension is reduced and some information is lost, the f1 score for truncated SVD models are worst of all.



model	class	precision	recall	f1-score	support
LogReg	bad	0.113021	0.422179	0.178307	514.0
	good	0.929420	0.696651	0.796375	5614.0
	average	0.860943	0.673629	0.744534	6128.0
Random Forest	bad	0.094704	0.295720	0.143464	514.0
	good	0.919965	0.741183	0.820953	5614.0
	average	0.850744	0.703819	0.764127	6128.0
XGBoost	bad	0.106642	0.665370	0.183822	514.0
	good	0.941116	0.489669	0.644171	5614.0
	average	0.871123	0.504406	0.605558	6128.0
CatBoost	bad	0.100037	0.529183	0.168265	514.0
	good	0.929011	0.564125	0.701984	5614.0
	average	0.859479	0.561195	0.657217	6128.0

Random Forest is the winner with 0.820953 score.



4.4.8 Applying Word2Vec and Simple Neural Network

We created word vectors using Word2Vec and the model has 25026 unique words where each word has a vector length of 100. Then we used these dense vectors - word embeddings - in a simple neural network to predict. In training and validation accuracy graph, the model starts to overfit after 3th epoch. The accuracy for this simple neural network is 0.9235.

Visualization of the words of interest and their similar words using their embedding vectors after reducing their dimensions to a 2-D space with t-SNE is presented in the graph. Similar words based on gensim's model can be viewed as well.

5. CONCLUSION

In this project, we tried to predict the rating scores based on the reviews left by the female customers. Before going through the model result, it is explicitly shown that data set preparation and feature engineering are as much as important as the model creation. We applied;

- Count Vector, TF-IDF, Hashing Vector, Word2Vec

- Classification Models and Simple Neural Network

- Adding most and least common words to CountVect,

- SMOTE,

- PCA + SMOTE

- Truncated SVD + SMOTE

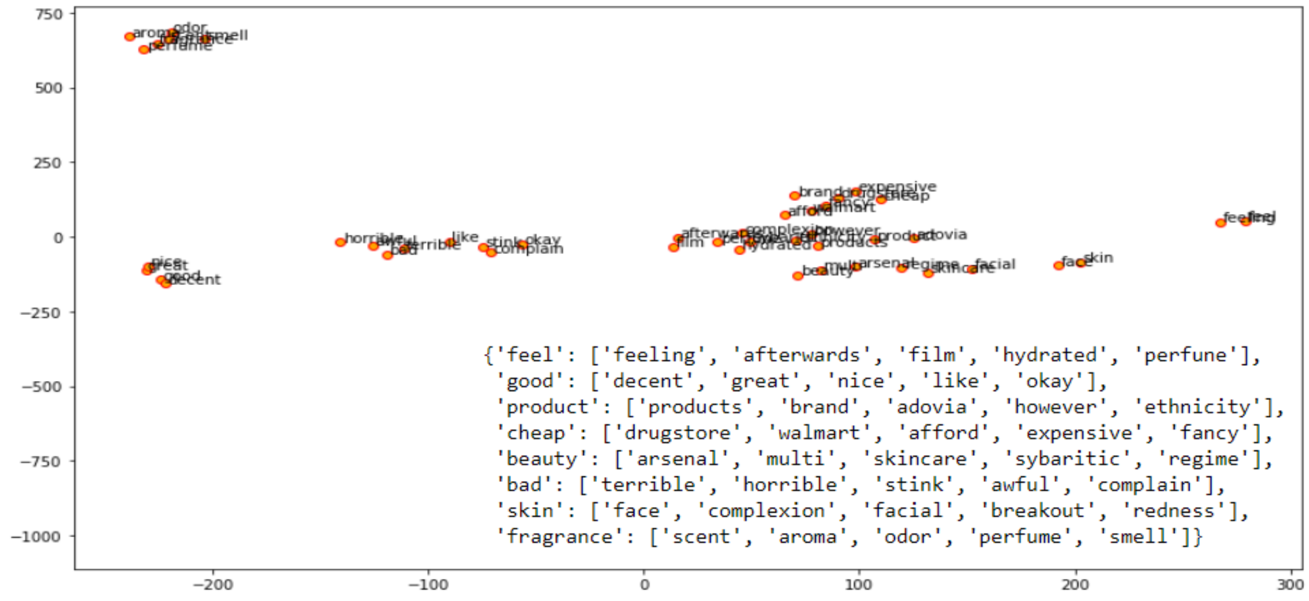
I will go with Naïve Bayes with Count Vectorizing in deploying section (f1 score is 0.924423). Adding most and least common words to the stopword list didn't have impact on models' performance. Resampling technique and linear dimensionality reduction did not make a positive improvement of the model accuracy but decreased the model performance.

5.1 Recommendations

We recommend the client to use the model as it is and give us some time to develop neural network algorithms to get better scores. By the way, prediction time and the size of the data set are also important and need to be considered. Especially neural network algorithms may not be outperforming with the low size data sets.

Provide a system that user can write their reviews/feedbacks without mistakes such as not accepting the review if it does not satisfy the word/grammar correctness.

Encourage users to reflect their experience with products via giving incentives.



```
{'feel': ['feeling', 'afterwards', 'film', 'hydrated', 'perfune'],
 'good': ['decent', 'great', 'nice', 'like', 'okay'],
 'product': ['products', 'brand', 'adovia', 'however', 'ethnicity'],
 'cheap': ['drugstore', 'walmart', 'afford', 'expensive', 'fancy'],
 'beauty': ['arsenal', 'multi', 'skincare', 'sybaritic', 'regime'],
 'bad': ['terrible', 'horrible', 'stink', 'awful', 'complain'],
 'skin': ['face', 'complexion', 'facial', 'breakout', 'redness'],
 'fragrance': ['scent', 'aroma', 'odor', 'perfume', 'smell']}
```

Use the reviews as a feedback mechanism, take actions towards them, and let the customers know about the conclusion.

5.2 Future Study

As a future study, we may focus on the following topics below.

* Implementation of Dask library for parallel processing to decrease run time.

* After decreasing run time, focusing on hyperparameter tuning more.

* Implementation of Deep Learning with different neural network types and different layer combinations to get better results.