CAPSTONE PROJECT 2- MILESTONE REPORT 2

NLP Sentiment Analysis
Amazon Home and Kitchen Product Reviews

4. NATURAL LANGUAGE PROCESSING

Due to computational considerations, I reduced the number of observations. I dropped the good_rating_class_reviews longer than 150 words, and I dropped all observations earlier than year 2010. After reducing, there are 8933 observations (7731 "good" reviews and 1202 "bad" reviews).

4.1 Feature Engineering and Selection

Machine Learning models take numerical values as input. The reviews are made of sentences, so in order to extract patterns from the data; we need to find a way to represent it in a way that machine learning 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 classification models in the following sections as a part of feature engineering and selection.

4.2 Data Preprocessing

Separate response variable and features:

First, 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 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

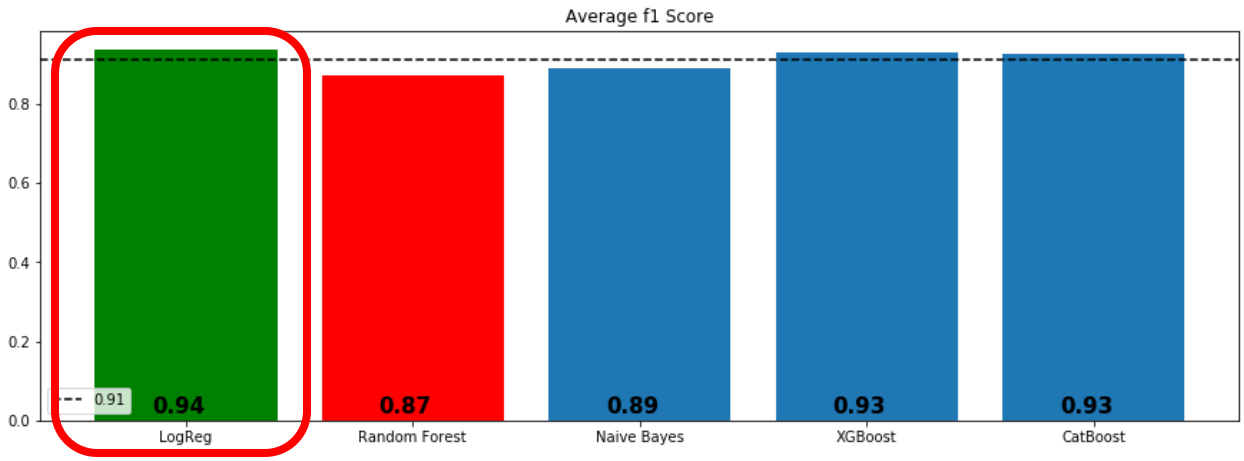
This is a supervised binary classification problem. We are trying to predict the sentiment based on the reviews left by customers who bought home and kitchen products in Amazon e-commerce online platform. We used Python's Scikit Learn libraries to solve the problem. In this context, we implemented Logistic 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 3-4-5 as 'Good' . 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 0.77. 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. **Logistic Regression is the winner with 0.936437.**

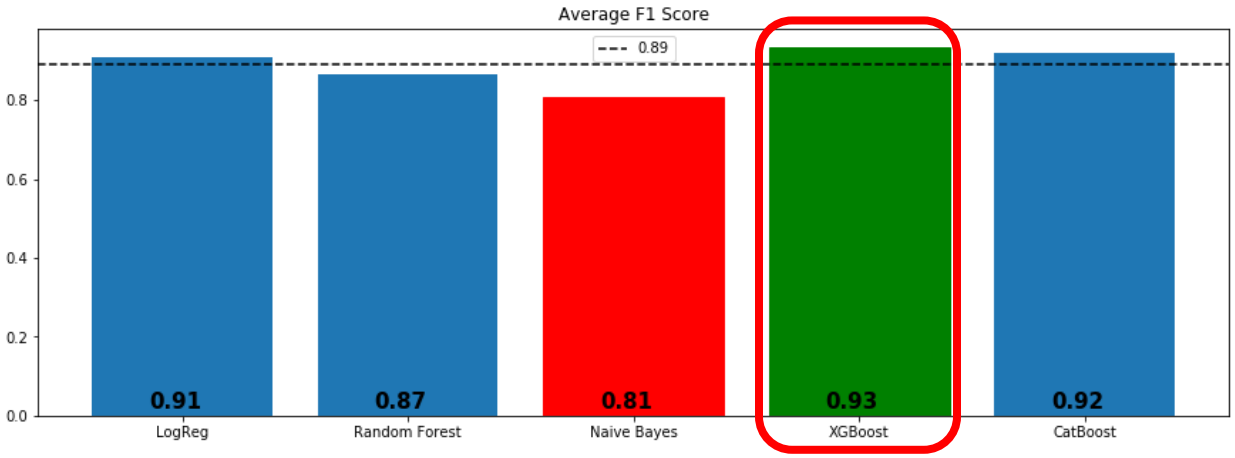
model	accuracy	class	precision	recall	f1-score	support
LogReg	0.936437	bad	0.745161	0.785714	0.764901	294.0
		good	0.967256	0.959278	0.963251	1940.0
		average	0.938028	0.936437	0.937147	2234.0
Random Forest	0.900627	bad	1.000000	0.244898	0.393443	294.0
		good	0.897317	1.000000	0.945880	1940.0
		average	0.910831	0.900627	0.873178	2234.0
Naive Bayes	0.886750	bad	0.559420	0.656463	0.604069	294.0
		good	0.946533	0.921649	0.933925	1940.0
		average	0.895588	0.886750	0.890515	2234.0
XGBoost	0.935542	bad	0.921348	0.557823	0.694915	294.0
		good	0.936770	0.992784	0.963964	1940.0
		average	0.934741	0.935542	0.928556	2234.0
CatBoost	0.931513	bad	0.840580	0.591837	0.694611	294.0
		good	0.940799	0.982990	0.961432	1940.0
		average	0.927610	0.931513	0.926317	2234.0



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. **XGboosting is the winner with 0.934794 score.**

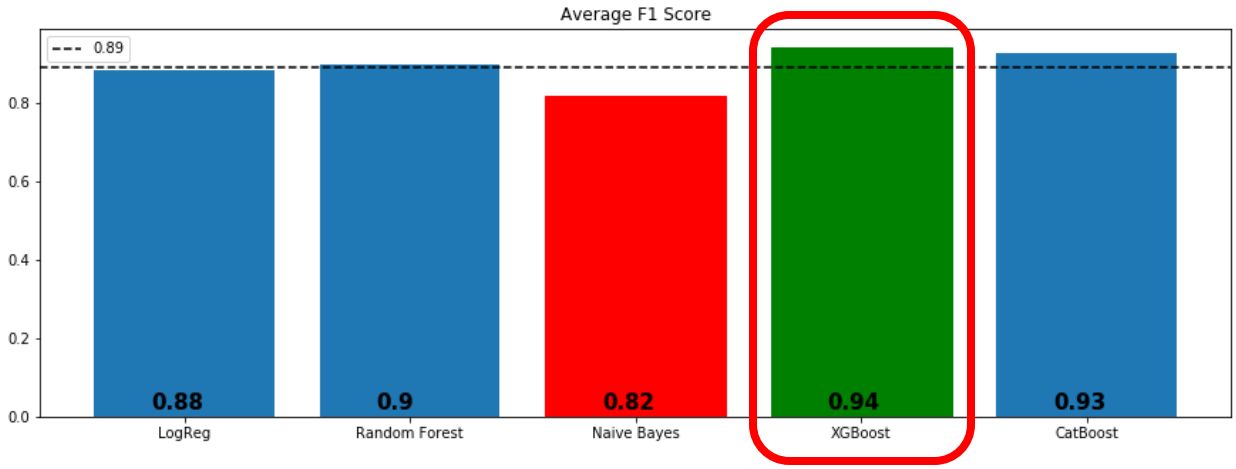
model	accuracy	class	precision	recall	f1-score	support
LogReg	0.904208	bad	0.595694	0.846939	0.699438	294.0
		good	0.975220	0.912887	0.943024	1940.0
		average	0.925274	0.904208	0.910968	2234.0
Random Forest	0.897493	bad	1.000000	0.221088	0.362117	294.0
		good	0.894421	1.000000	0.944269	1940.0
		average	0.908316	0.897493	0.867656	2234.0
Naive Bayes	0.868397	bad	0.000000	0.000000	0.000000	294.0
		good	0.868397	1.000000	0.929564	1940.0
		average	0.754114	0.868397	0.807231	2234.0
XGBoost	0.941361	bad	0.965714	0.574830	0.720682	294.0
		good	0.939291	0.996907	0.967242	1940.0
		average	0.942768	0.941361	0.934794	2234.0
CatBoost	0.926141	bad	0.824121	0.557823	0.665314	294.0
		good	0.936118	0.981959	0.958491	1940.0
		average	0.921379	0.926141	0.919908	2234.0



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. **XGboosting is the winner with 0.941092 score.**

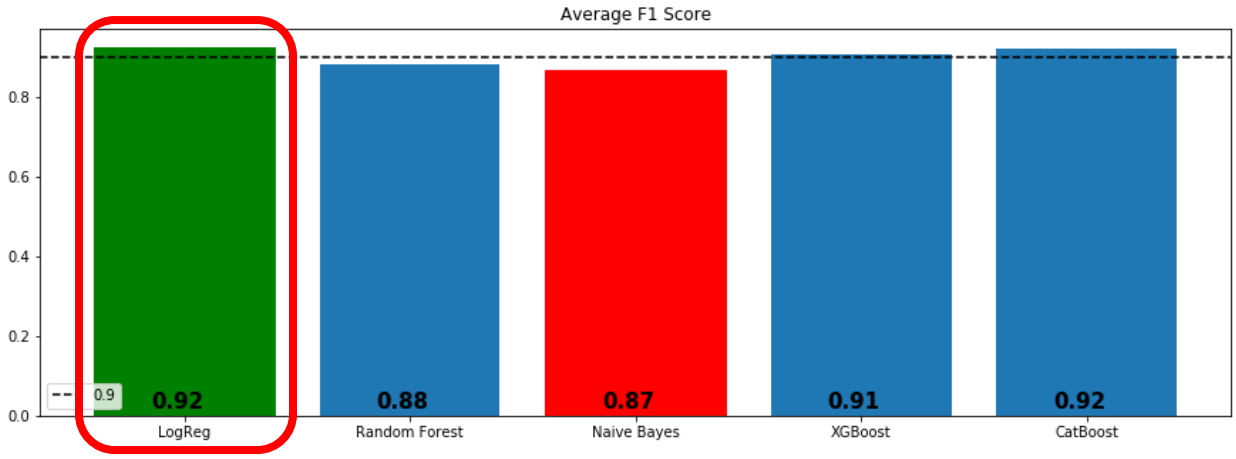
model	accuracy	class	precision	recall	f1-score	support
LogReg	0.870636	bad	0.505133	0.836735	0.629962	294.0
		good	0.972524	0.875773	0.921616	1940.0
		average	0.911015	0.870636	0.883234	2234.0
Random Forest	0.914951	bad	1.000000	0.353741	0.522613	294.0
		good	0.910798	1.000000	0.953317	1940.0
		average	0.922537	0.914951	0.896635	2234.0
Naive Bayes	0.871979	bad	1.000000	0.027211	0.052980	294.0
		good	0.871518	1.000000	0.931349	1940.0
		average	0.888427	0.871979	0.815753	2234.0
XGBoost	0.946285	bad	0.962766	0.615646	0.751037	294.0
		good	0.944770	0.996392	0.969895	1940.0
		average	0.947139	0.946285	0.941092	2234.0
CatBoost	0.931513	bad	0.847291	0.585034	0.692153	294.0
		good	0.939931	0.984021	0.961471	1940.0
		average	0.927739	0.931513	0.926028	2234.0



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. **Logistic Regression is the winner with 0.924259 score.** Adding most and least common words to the stopword list didn't have impact on models' performance.

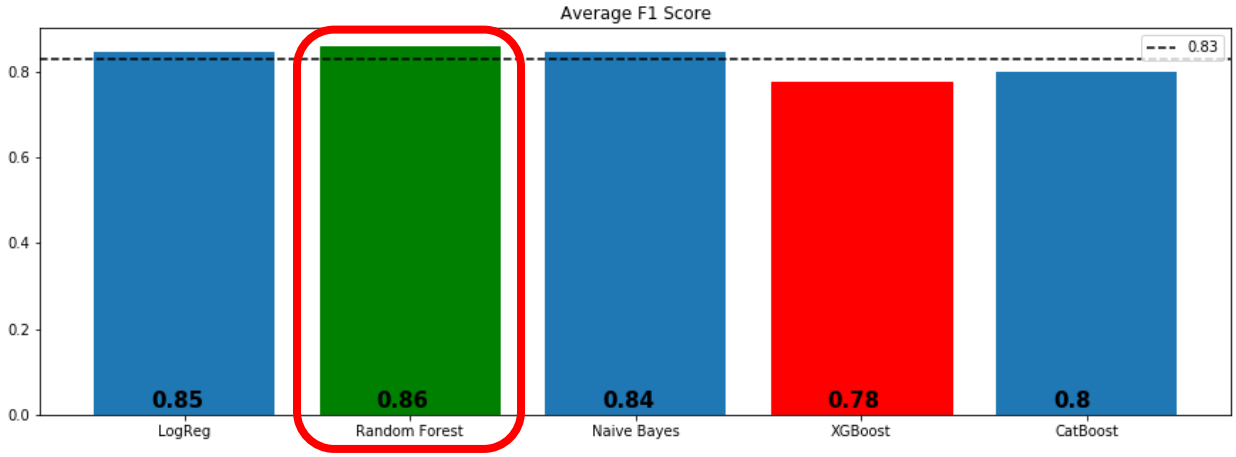
model	accuracy	class	precision	recall	f1-score	support
LogReg	0.923456	bad	0.699029	0.734694	0.716418	294.0
		good	0.959481	0.952062	0.955757	1940.0
		average	0.925204	0.923456	0.924259	2234.0
Random Forest	0.905998	bad	0.988372	0.289116	0.447368	294.0
		good	0.902700	0.999485	0.948630	1940.0
		average	0.913975	0.905998	0.882663	2234.0
Naive Bayes	0.854073	bad	0.463964	0.700680	0.558266	294.0
		good	0.950838	0.877320	0.912601	1940.0
		average	0.886764	0.854073	0.865969	2234.0
XGBoost	0.921218	bad	0.953846	0.421769	0.584906	294.0
		good	0.919202	0.996907	0.956479	1940.0
		average	0.923761	0.921218	0.907579	2234.0
CatBoost	0.928380	bad	0.860215	0.544218	0.666667	294.0
		good	0.934570	0.986598	0.959880	1940.0
		average	0.924785	0.928380	0.921292	2234.0



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 0.86. We should keep in mind that oversampling will generate artificial observations which may be tricky to evaluate the accuracy of the model. **Random Forest is the winner with 0.858826.**

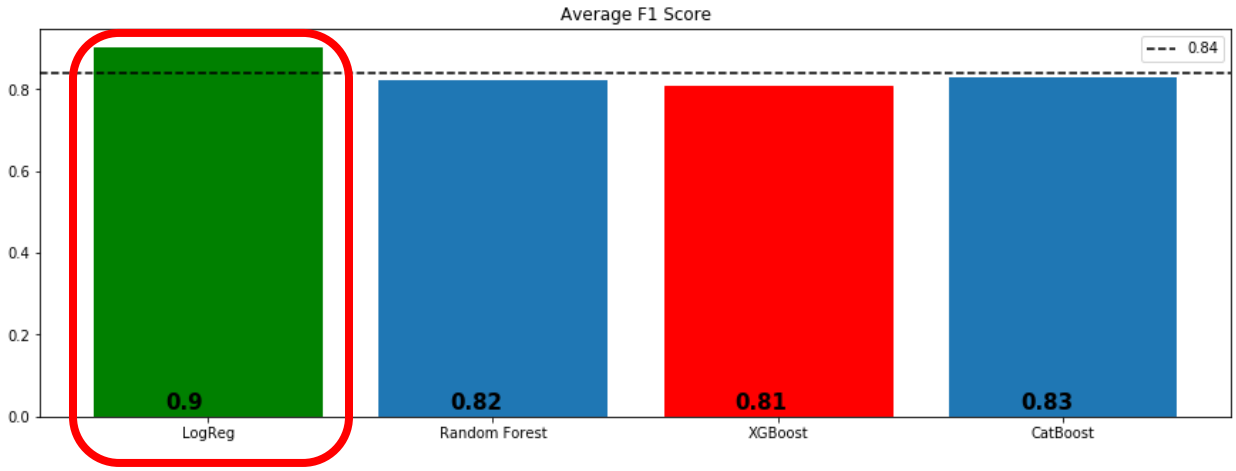
model	accuracy	class	precision	recall	f1-score	support
LogReg	0.834825	bad	0.411765	0.595238	0.486787	294.0
		good	0.934218	0.871134	0.901574	1940.0
		average	0.865462	0.834825	0.846987	2234.0
Random Forest	0.873321	bad	0.531792	0.312925	0.394004	294.0
		good	0.901989	0.958247	0.929268	1940.0
		average	0.853270	0.873321	0.858826	2234.0
Naive Bayes	0.829902	bad	0.405702	0.629252	0.493333	294.0
		good	0.938695	0.860309	0.897795	1940.0
		average	0.868552	0.829902	0.844566	2234.0
XGBoost	0.740824	bad	0.285068	0.642857	0.394984	294.0
		good	0.933164	0.755670	0.835090	1940.0
		average	0.847873	0.740824	0.777171	2234.0
CatBoost	0.771710	bad	0.310526	0.602041	0.409722	294.0
		good	0.929688	0.797423	0.858491	1940.0
		average	0.848204	0.771710	0.799432	2234.0



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. **Logistic Regression is the winner with 0.903021.**

model	accuracy	class	precision	recall	f1-score	support
LogReg	0.900179	bad	0.605341	0.693878	0.646593	294.0
		good	0.952557	0.931443	0.941882	1940.0
		average	0.906862	0.900179	0.903021	2234.0
Random Forest	0.870636	bad	0.592593	0.054422	0.099688	294.0
		good	0.874037	0.994330	0.930311	1940.0
		average	0.836998	0.870636	0.820999	2234.0
XGBoost	0.863921	bad	0.187500	0.010204	0.019355	294.0
		good	0.868801	0.993299	0.926888	1940.0
		average	0.779140	0.863921	0.807454	2234.0
CatBoost	0.864369	bad	0.447059	0.129252	0.200528	294.0
		good	0.880875	0.975773	0.925899	1940.0
		average	0.823784	0.864369	0.830438	2234.0

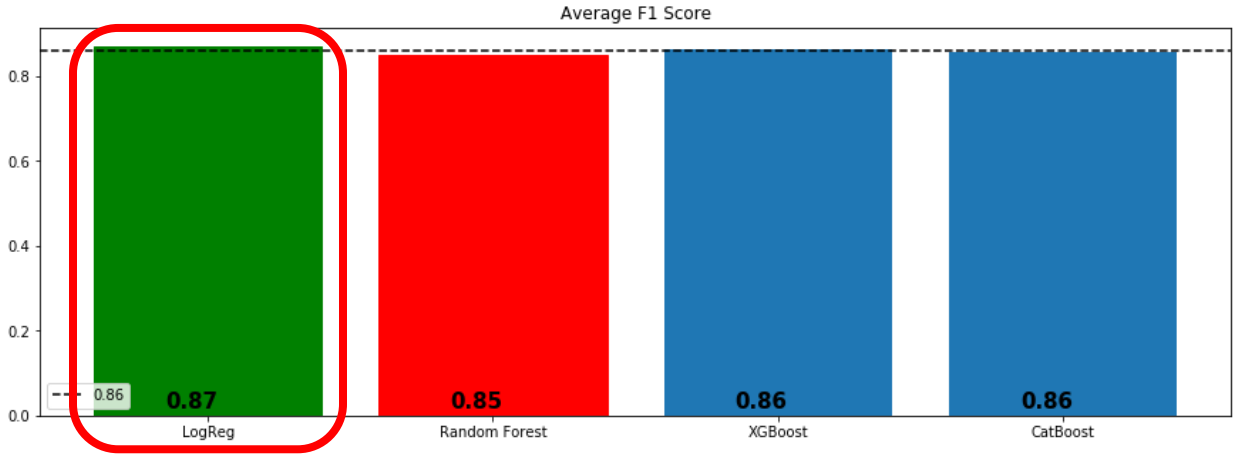


4.4.7 Truncated SVD + SMOTE

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

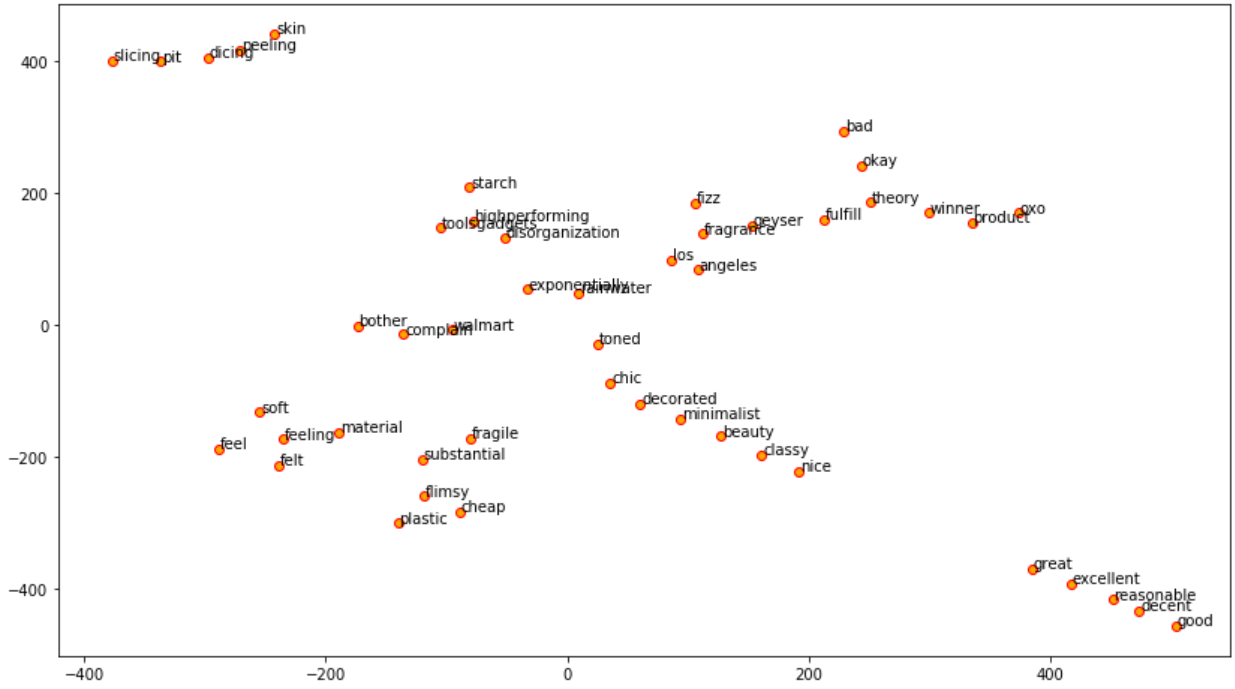
Logistic Regression is the winner with 0.870563 score.

model	accuracy	class	precision	recall	f1-score	support
LogReg	0.860340	bad	0.478774	0.690476	0.565460	294.0
		good	0.949724	0.886082	0.916800	1940.0
		average	0.887746	0.860340	0.870563	2234.0
Random Forest	0.835273	bad	0.420259	0.663265	0.514512	294.0
		good	0.944068	0.861340	0.900809	1940.0
		average	0.875133	0.835273	0.849971	2234.0
XGBoost	0.851835	bad	0.458797	0.700680	0.554509	294.0
		good	0.950700	0.874742	0.911141	1940.0
		average	0.885965	0.851835	0.864207	2234.0
CatBoost	0.844673	bad	0.441501	0.680272	0.535475	294.0
		good	0.947221	0.869588	0.906745	1940.0
		average	0.880667	0.844673	0.857885	2234.0



4.4.8 Applying Word2Vec and Simple Neural Network

We created word vectors using Word2Vec and the model has 20149 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 0.9161.



{**'feel'**: ['feeling', 'felt', 'soft', 'substantial', 'material'],
'good': ['decent', 'great', 'nice', 'excellent', 'reasonable'],
'product': ['winner', 'toolsgadgets','disorganization','oxo','highperforming'],
'cheap': ['flimsy', 'fragile', 'plastic', 'walmart', 'exponentially'],
'beauty': ['chic', 'toned', 'minimalist', 'decorated', 'classy'],
'bad': ['complain', 'okay', 'theory', 'fulfill', 'bother'],
'skin': ['peeling', 'starch', 'slicing', 'dicing', 'pit'],
'fragrance': ['fizz', 'los', 'angeles', 'geyser', 'rainwater']}

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 above. 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 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

We can go with XGboosting with Hash Vectorizing (f1 score is 0.941092) or Logistic Regression with Count Vectorizing (f1 score is 0.936437) in deploying section. 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.

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

5.2 Future Study

The future studies on the similar data may focus on;

• Using different methods in order to minimize the effect of the matching words

• Implementation of deep learning with different neural network types and different layer combinations.

• Using different AutoML tools.

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