Predicting Recommendation using Reviews

Problem Definition

For my project, I used women's clothing e-commerce website dataset. The aim of this project is to build and evaluate different models and compare them to look for best model for predicting recommendation from reviews text. I found this problem interesting because there are many websites like Target and Amazon which asks for a lot of information from users while writing a review like ratings, review description and if they would recommend the product. Although recommending is a small step and does not take a lot of time for the users, but it would be more convenient if they did not have to provide that information at all. My project helps to remove this extra step of user and this small change would improve the interaction experience for user.

Data Understanding and Preprocessing

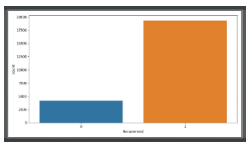
I have retrieved my dataset from Kaggle and it consists of 23486 rows and 10 feature variables. Here, each row indicates a user review. Out of 10 variables I am focusing on "Review text and recommendation Index which is my target variable. There are other features related to demographics, department and ratings but, for my project those are not needed.

Problems faced and What I learned at each step of the project:

My first step was to clean the unstructured review texts, so for that I did preprocessing on the review text to convert it into standardized form. I did that in two steps: Noise removal and Lexicon Normalization. In Noise removal, first I converted texts into lowercase and then did stopwords removal. I learned that python provides a list of stopwords library "SKlearn" which can be used to remove stopwords. After doing this Noise Removal, I did lexicon Normalization to remove different variations of same word. For that, first I used stemming method but with that I didn't get the correct solution because stemming just strips the suffixes of word so, I was still having word duplication in the dataset. Then I looked for other method which can give me root form of the word. I learned about lemmatization and used this method to get root form of word. So, this method helped me to get more accurate results.

In the next step, I converted review words into Tf-Idf weighted matrix, where I learned that Scikit-learn in python provides two methods to get weighted matrix). One is a two-part process of using the CountVectorizer class to count how many times each term shows up in each document, followed by the TfidfTransformer class generating the weight matrix. The other does both steps in a single TfidfVectorizer class. I did two-part process as I wanted to know the n-gram count of words with weights and the number of words which non-zero values. I also learned about **'bag of words' representation** which helped me to create a separate column for each term that contains the count within each document. Then, I also looked at the **sparsity** of this data. I came to know that the sparser the data is the more challenging it can be to build a model. But, in my case it was not too sparse otherwise it might have led to a different problem. So, this step went smoothly during the project and I got end result in the form of matrix.

In the third step, I did some data exploration before doing data modelling where I looked for highest frequency words and highest weighted words. I also analyzed target variable. And, I found that there are five times recommendation then there is non-recommendation.



At this point I was not aware that this class imbalance can create problem during data modelling and evaluation.

Then, in the data modelling and evaluation stage, I applied 4 models Gaussian Naïve Bayes, Random forest, Logistic Regression and Linear SVC and did evaluation over weighted text matrix. I gauged accuracy by doing 5-fold cross validation to get smoothed out representation. I found that the Class 0 precision and recall values were very low. And, this also affected overall accuracy of the result which can lead to incorrect prediction. So, I searched the reason for this and I found that this issue is due to the class imbalance of recommendation and non-recommendation. As I mentioned earlier, there are around five times fewer 0 classifications compared to 1 classification, which can sometimes be problematic within Machine Learning. I learned that modelling generally tends to work better when there is an almost equal numbers of samples from each class in the target variable. With this dataset, I was way off that.

So, to overcome this issue, I learned about two methods DownSampling the majority Class and OverSamping the minority class. Out of this I used Oversampling. Python has SMOTE library which I have used to do oversampling the class. After this I re-ran all the 4 models and to re-assess the performance of models. In the 2nd round, I got drastic improvement on Precision & Recall across all algorithms, even better overall accuracy. So, from these, Random forest Classifier has the highest accuracy therefore, I can say that it is the best model compared to other model for predicting recommendation.

Then, I discovered that that Random forests offers a good feature selection indicator. It automatically computes the relevance score of each feature in the training phase. Then it scales the relevance down so that the sum of all scores is 1. So, I further looked for top predictive features from the Random Forest Classifier. It shows the relative importance or contribution of each feature in the prediction. Here, I found that the word “love” is the strongest predictor for recommendation.

I also learned related to polarity of the review that can be used to measure how positive or negative a statement is in the range of -1 (very negative) to +1 (very positive).

Overall, I learned a lot out of problems I faced and solving it.