Text Classification of E-commerce Clothing Line Reviews

Dona Ray

May, 2018

1 Introduction

User reviews are major sales drivers, and most users will read reviews prior to making an informed decision about a purchase. Consumer reviews are significantly more trusted than product description from the manufacturer, according to a survey of US internet users by online review video site EXPO. According to Reevo Stats, 63% of customers are more likely to make a purchase from a site that has user reviews and that a product that has 50 or more reviews can result in a 4.6% conversion rate. In fact, although counter-intuitive, according to Reevo Stats, a mix of good and bad reviews are valuable, with bad reviews actually improving conversion rates by 67%. This is not surprising since, people will believe reviews to be fake if there are no bad reviews for a product. A few percentage of bad reviews is good then, when a product has a large number of positive reviews.

Clearly then, reviews play an increasingly important role in the decision making process of an online product. User generated reviews are greatly influential in persuading an individual to make a purchase. There are primarily two ways reviews are generated online:

a) Internal word of mouth (WOM): Reviews located on the retailer's website
b) External WOM: Reviews located on a third party website.

Amazon is an example of the first category, while Yelp is an example of the second category. In our paper, we analyze review data from a Women's Clothing E-Commerce dataset containing reviews of their customers on their products. This is an example of Internal word of mouth, like Amazon. Our goal is to predict the label on the review using supervised binary classification. Why is this useful? Predicting ratings for review has many applications. For example, a company might want to analyze social media text data, to understand the public sentiment about the company. Or may also want to identify negative reviews in order to reach out to people posting such reviews. Predicted ratings from text reviews are also important where ratings are not available.

We also apply dimension reduction with topic modeling and compare the results of binary classification with and without dimension reduction. Word count or frequency feature matrix of text data can be very large, with the number of features or columns being the number of unique words in the corpus. One problem with a large number of features, is that the algorithm may be prone to overfitting. An overfitted model can mistakenly interpret small fluctuations as important variance in the data leading to classification errors. Increase in dimension also increases computational cost, usually exponentially. We try to address both these problems by trying to find projections of the data onto a smaller number of variables or features which preserves the information as much as possible. We do this by using topic modeling techniques like Latent Dirichlet Allocation (LDA) and Latent Semantic Indexing[1] (LSI).

2 Data

The Women's Clothing E-Commerce dataset has 23,486 rows and 10 feature variables in a zipped csv format. The feature variables include, reviewer's age, review text, as well as rating, which is an integer from 1 (worst) to 5 (best). It also includes variables like, Division Name, whether its general, petite or intimates. The feature variable Department has 6 categories or classes, and namely information about the type of clothing, dresses, tops, bottoms, jackets, intimates and trend. After dropping 845 rows with missing values in the review column, the dataset is reduced to 22,641 rows.

We would like to use text classification to analyze a Women's Clothing E-Commerce dataset, and predict the labels based on the review. To be able to do a binary classification, we group the labels into two classes as follows: the target variable y = 0 if rating is 1, 2 or 3 and y = 1 if the rating is 4 or 5 stars. Prior to training the data using various supervised learning algorithms, we need to perform some text data analysis and cleanup. To understand the data and the variables associated with it, we perform the following summary statistics below:

2.1 Distribution of the Ratings

The frequency distribution of the ratings as well as the bar chart showing the total count of for each rating is given below. 5-star ratings is the maximum at 12,540 while 1-star rating is the minimum at total count 821. The total next highest rating is less than half at 4,908, followed by 2,823 for 3-star ratings and 1,549 for 2-star ratings respectively. This is clearly good news for the clothing line, the number of 5-star ratings is more than 12 times the number of 1-star ratings. In addition, 5-star ratings account for a little over 50% of ratings. And, 4-star and 5-star ratings together account for almost 77.06% of all ratings.

[1] LSI is an application of a mathematical technique, called Singular Value Decomposition or SVD.

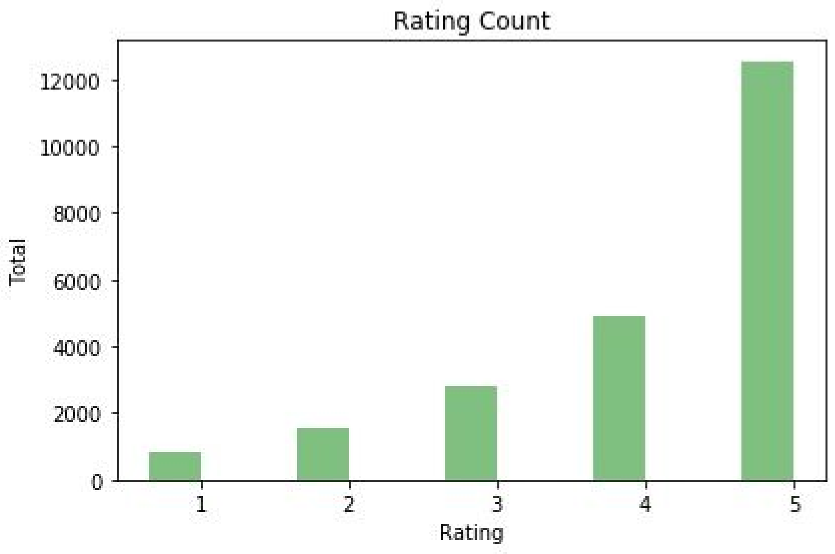

Figure 1

2.2 Total Positive Feedback Count by Rating

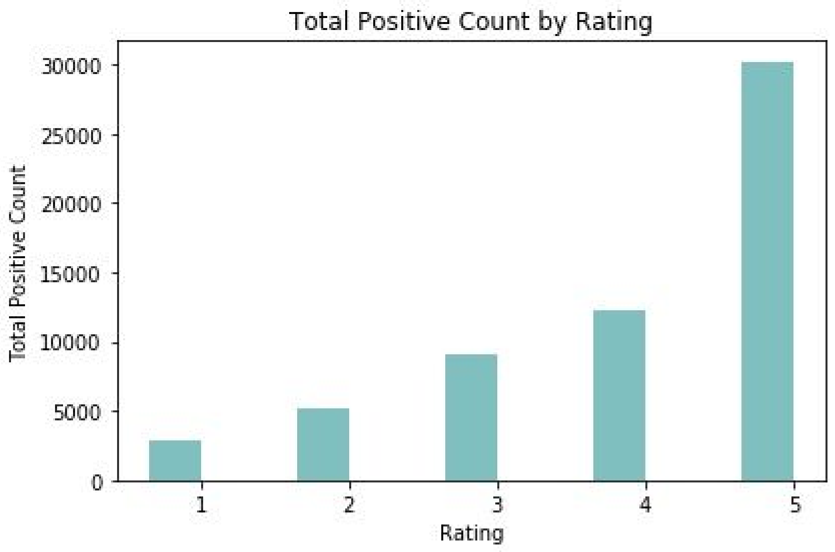

Figure 2

The feature variable “Positive Feedback Count” gives the number of upvotes for a particular review. While this a helpful variable, we do not have information on the date of review. So naturally, older reviews will have more positive counts than more recent ones. Nevertheless, we can still gain some insights by understanding the distribution of the total and average positive count by rating.

The frequency distribution of the total positive feedback count is shown as a bar chart in Figure 2. The x-axis shows the 5 ratings, while the y-axis shows the total number of positive feedback count. The total count is highest for 5-star ratings at 30,198, followed by 4-star ratings at less than half at 12,213. 1-star ratings have the lowest count at 2913. One reason the counts are highly skewed to the right is because, the rating distribution itself is skewed to the right. Note that, from Figure 1, 5-star ratings accounts for more than 50% of overall ratings.

2.3 Positive Feedback Count by Rating

Since total positive feedback count is skewed to the right, because the ratings themselves are skewed, to understand the distribution of the feedback counts better, we need to look at the overall distribution positive feedback count.

The box-plot of the ‘Positive Feedback Count’ by rating is depicted in figure 3[2]. The horizontal or x-axis shows the rating, and the positive feedback count is shown in the vertical or y-axis. The 1-star ratings have the highest average positive feedback count, at 3.55 while the 5-star rating have the least value at 2.41. It is interesting to note that the lower ratings[3] have a average positive feedback count between 3.2 and 3.5, while higher ratings[4] have a average positive feedback count of 2.4. This seems to indicate that potential customers view negative reviews as being more useful than positive ones. One reason this is so because, explaining certain flaws in a product, helps the buyer make a more informed decision about a purchase.

[2] The box-plot for the Positive Feedback Count does not show outliers, because the data is highly skewed with a very high standard deviation.
[3] 1-star, 2-star and 3-star are considered low ratings.
[4] 4-star and 5-star can be considered high ratings.

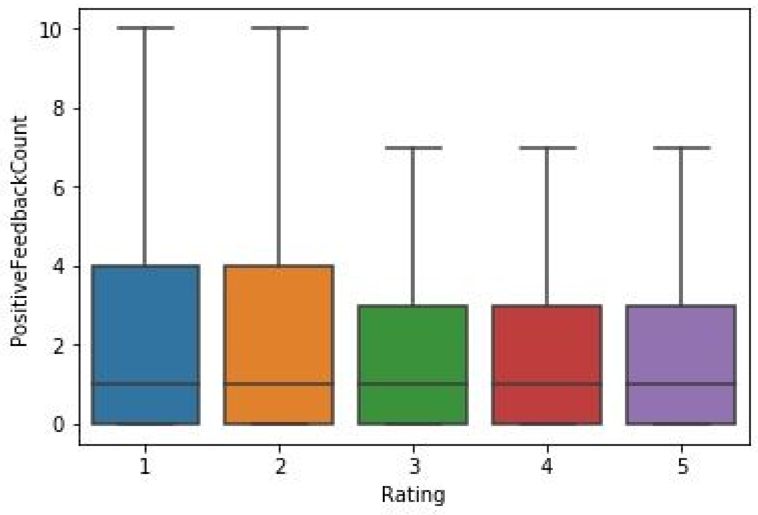

Figure 3

2.4 Cross Frequency of Rating by Division

We next try to understand the distribution of ratings by Division. Division has three categories, General, Petite, Intimates. We would like to see if a particular Division category has better ratings than another category. We compute the cross frequency of the feature ‘Division Class’ by rating.

Rating	**Division Name**		
	General	**General Petite**	**Intimates**
1	479	291	51
2	950	524	75
3	1740	929	154
4	2910	1702	296
5	7286	4391	850

Table 1

Table 1 gives the cross frequencies of each Division category by rating. It is difficult to see from the total counts, if the ratings favor a particular Division Class. The General class and rating 5 will naturally have the highest count at 7,286, while

‘Intimates’ with rating 1 has the lowest count at 51. Next we compute the percentage of total counts by rating in each Division category. This is shown in Figure 4. For example, the bottom left cell in red shows that 55% of all ratings in the General category has a rating value of 5. The cell above in light green, shows that 22% of all ratings in the General category has a rating of 4. It can be easily seen that the distribution of ratings is uniform across General, Petite and Intimates. This is expected and there is no discernible difference in products in each of the three categories.

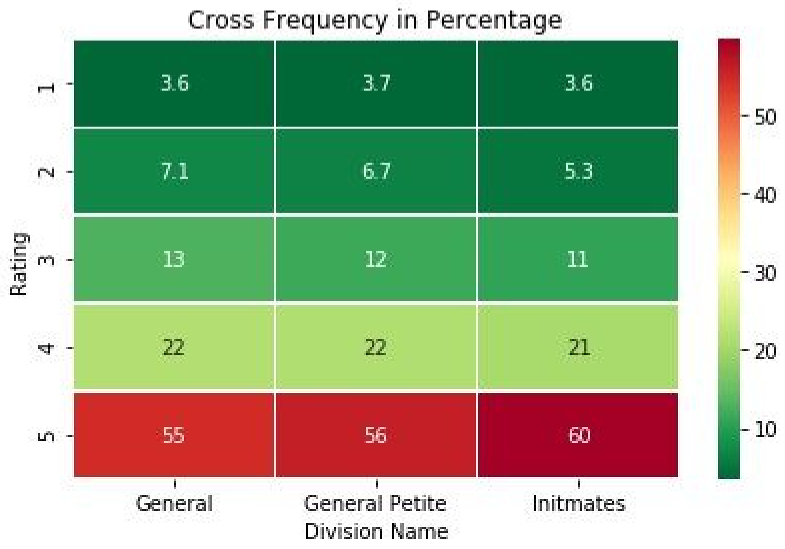

Figure 4

2.5 Cross Frequency of Rating by Department

Table 2 shows the cross frequency of rating by Department. This is similar to the above figure, except we now have 6 categories for Department instead of three. Tops with a with a 5-star rating have the highest count at 5,444, while Trend with a 1-star rating has the lowest count at 10. Figure 5 shows the percentage counts by each Department category. Again, we can clearly see that the distribution of percentage of rating labels by each Department category is uniform. That is the percentage is monotonically increasing from 1-star to 5-star rating over all categories. In addition, the percentage is similar across all 6 categories.

Rating	Department					
	Bottoms	Dresses	Intimate	Jackets	Tops	Trend
1	114	222	60	48	367	10
2	203	459	87	60	729	11
3	407	830	177	90	1300	19
4	762	1367	350	195	2208	26
5	2176	3267	979	609	5444	52

Table 2

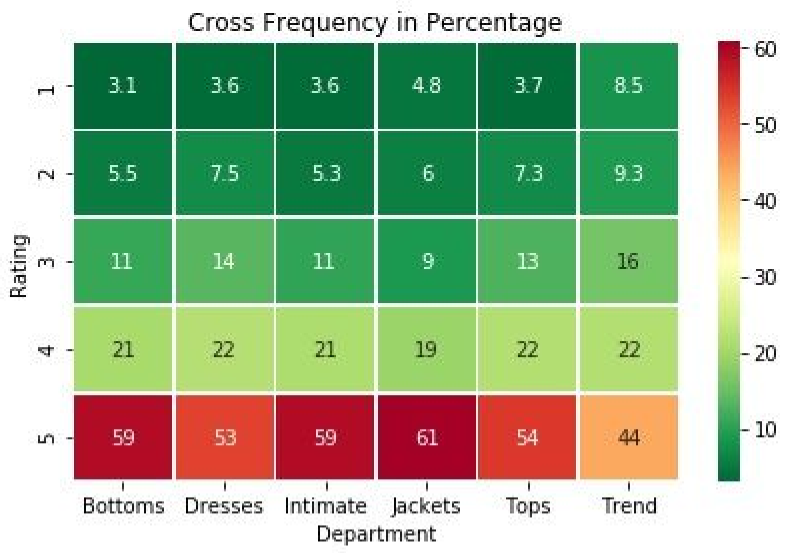

Figure 5

Table 2 shows the cross frequency of rating by Department. This is similar to the above figure, except we now have 6 categories for Department instead of three. Tops with a with a 5-star rating have the highest count at 5,444, while Trend with a 1-star rating has the lowest count at 10. Figure 5 shows the percentage counts by each Department category. Again, we can clearly see that the distribution of percentage of rating labels by each Department category is uniform. That is the percentage is monotonically increasing from 1-star to 5-star rating over all categories. In addition, the percentage is similar across all 6 categories.

2.6 Positive Feedback Count by Division

The feature, Positive Feedback Count, gives the number of positive counts by a user for a given review. It would be interesting to see if this distribution changes by Division category. The total number of counts will be misleading, since the General category has more reviews and consequently more positive counts. Figure 6 below, shows the box-plot of the positive feedback count by division. The x-axis shows the three Division categories, General, Petite and Intimates, while the y-axis shows the positive feedback count. The average count for General and Petite category is similar at 2.7, while Intimates is lower at 1.9.

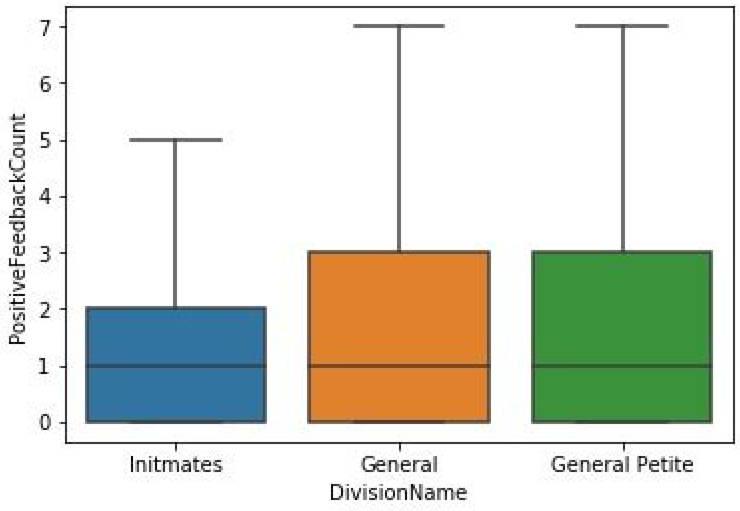

Figure 6

2.7 Positive Feedback Count by Department

We next look at the distribution of the feedback count by Department in figure 7. Department is categorized into 6 classes: Bottoms, Dresses, Intimate, Jackets, Tops and Trend. Again, the x-axis depicts the class, and the positive feedback count is plotted on the y-axis. The lowest average positive feedback count is for class ‘Intimate’, while the highest is for class ‘Trend’ followed by ‘Dresses’. The median, first and third quartile for trend is higher compared to the other 5 categories. But note that the trend also has fewer data points. The distribution of the rating data for ‘dresses’, ‘bottoms’, ‘tops’ and ‘jackets’ is similar. The third quartile for ‘intimates’ is lower at around 3.

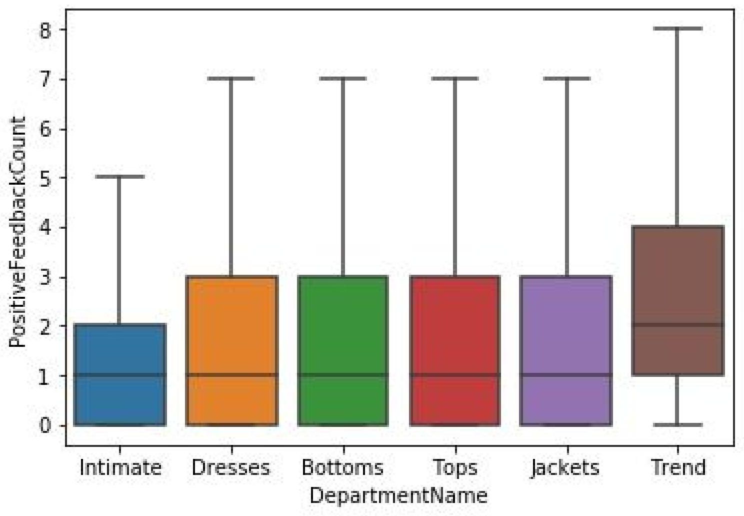

Figure 7

2.8 Word Count by Rating

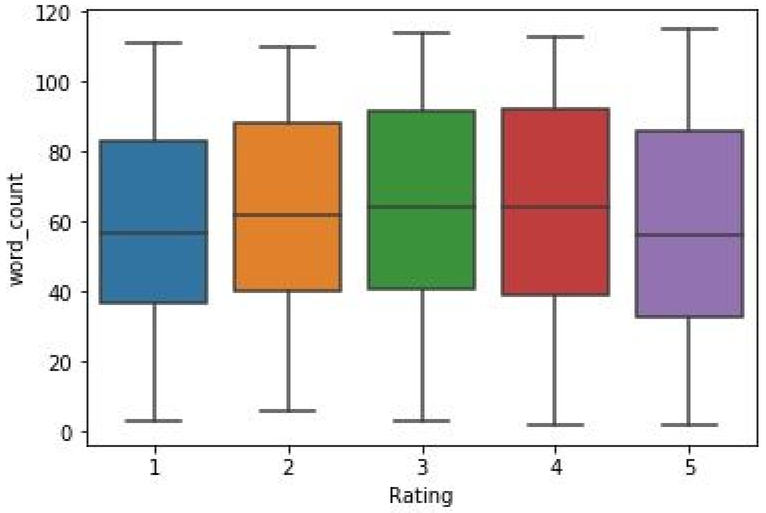

Figure 8

Each document has an average of 60.21 words per review, with total of 1,363,325 words. The minimum review has 2 words while the longest review is at 115 words. Rating 3 has the highest average word count at 64.23 and rating 5 has the lowest average word count at 57.97. Rating 1 is similar to rating 5 with a average word count of

58. This is probably because, people who give a rating of 3 have conflicting opinions, that requires relatively more description. People giving the highest or the lowest rating can describe their feelings about the product succinctly in a few words. Figure 8 above, shows the box-plot of word counts by rating.

2.9 Stopword Count by Rating

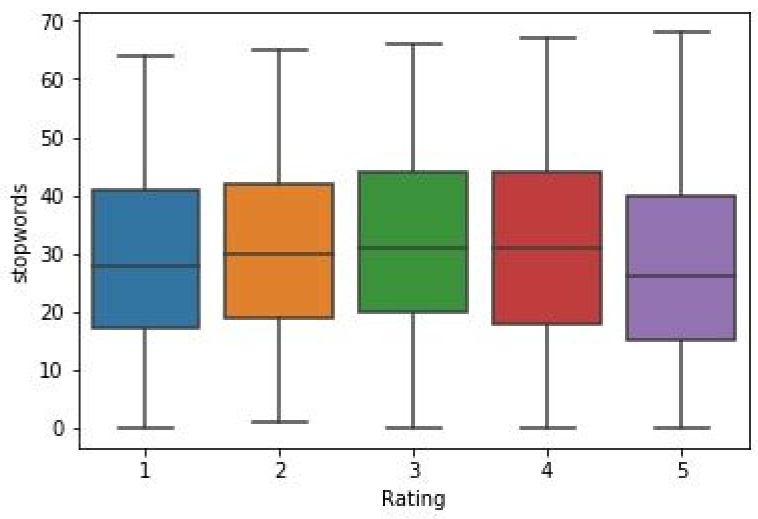

Figure 9

Figure 9 shows the box-plot of stopwords by rating. As expected, the distribution is similar to the distribution of average word counts. Rating 3 has the highest average at 31.58 and ratings 1 and 5 have the least average number of stopwords at 28.49 and 27.58 respectively. The minimum is zero, maximum is at 68 and the mean is at 28.99. Comparing this with the corresponding values for word counts, we can see that stopwords account for half of the total number of words. For both the word count and the stopwords, there are no outliers.

2.10 Top 10 most common words

The table below show the 10 most common words in the column ‘ReviewText’. Not surprisingly dress is the most common word with a count frequency of 11,319 followed by fit, size, love and top making the top 5 most common words.

Top 10 Common words	Count
dress	11,319
fit	10,091
size	9,349
love	8,968
top	8,256
like	7,018
color	6,903
look	6,873
wear	6,512
great	6,076

Table 3

2.11 Final word and unique word count

Prior to converting the text data to a sparse matrix, we need to do some pre-processing. We do some basic text cleanup operations like removing punctuation, stopwords, numerics and stemming or lemmatization of words.

	Total Words	Unique Words
Overall Total	1,363,325	40,071
Convert to Lower Case	1,362,913	37,720
Remove Punctuation	1,362,913	19,386
Remove Stopwords	668,290	19,244
Remove Numerics	652,217	18,840
Stem words	652,217	17,483
Remove words that appear once	642,694	7,960

Table 4

Table 4 shows the total number of words as well as the total number of unique words after each operation. The initial word count is 1,363,325 and the total number of unique words is 40,071. Converting to lowercase, reduces the unique word count to 37,720. Note that removing punctuations reduces the total number of unique by half. Almost half of the total count is stopwords, and removing them lowers the total word count to 668,290 from 1,362,913. Removing numerics and stemming words reduces the unique word count to 18,840 and 17,483 respectively. Finally, we drop words that occur only once as rare words will most likely not be very predictive in classification. Secondly, this will also drop incorrectly spelt words, as these will also most likely occur just once.

3 Modeling

Our goal in this paper is to use text classification to analyze a Women's Clothing E-Commerce dataset containing reviews of their customers. We first do a binary classification using this target variable, by converting it into a 2-class target variable. We define y=1 if its a 4 or a 5-star review (good review) and y=0 if the review is a 1, 2 or 3-star rating. The feature space is the numpy matrix created by the CountVectorizer or the TFIDF vectorizer.

3.1 Converting text data to numerical vectors

Text data cannot be directly used as a feature for a machine learning algorithm. It needs to be first converted into numerical feature vectors. This can be accomplished in the following two ways:

3.1.1 Bag of Words

The bag of words representation assigns a fixed id to each word occuring in any document of the Review text data. We next count the number of occurrences of each word in the corpus in a for each review or document. These counts are then stored in the feature space matrix X. The number of columns m, of the feature space is the number of unique words[5] in the vocabulary or text corpus. The number of rows n, is the total number of reviews, or 22,641. The size of our feature matrix that will be using for training in Supervised Learning is then a 22,641 x 7,960 matrix of non-negative numbers[6]. This matrix X can be created by Scikit-learn's CountVectorizer method.

[5] Note from Table 4, after pre-processing of text data, the number of unique words is 7,960.

[6] In the case of CountVectorizer, the i,jth element is the number of times the jth word occurs in the ith review. For the TFIDF, the i,jth element is the term frequency of the jth word in the ith document or review.

3.1.2 Frequencies

One problem with counts is that longer documents will have larger counts, but may be similar to smaller documents since they may both talk of similar topics. To avoid these discrepancies, we can divide the number of counts by the total number of words in that document to get TF or term frequencies. We can also downscale weights of words that occur in many documents as popular words that frequently appear are less informative or not as predictive as words that occur in a smaller proportion of the documents. This can be done by using Scikit-learn TF-IDF vectorizer.

3.2 Cross-Validation

For a simple model with very few features, the model may fail to capture important patterns in the data. When this happens, the model is said to be under-fitting or biased. On the other hand, as a model gets more complex, it may pick up patterns in the data that do not generalize to the overall population. In this scenario, we say that the model is overfitting the data, because it is finding interesting but chance occurrences in the data that do not generalize. This is also referred to as a model with high variance. While we clearly do not want our model to be biased[7], models with high variance[8] are also undesirable. One reason why overfitting may occur or go undetected is if training and testing is done on the same dataset. To avoid such scenarios, we split our data into a training and testing or hold out data set. Testing the model on a hold out data that the model has not been trained on, ensures that the model does not get learned on occasional erroneous labels.

We randomly split the data into a 80% training set and a 20% testing set. We then use a k-fold cross-validation technique[9] on the training set. The k-fold cross-validation then splits the training data into k folds, and uses (k-1) folds in each iteration for training the model. At the end of the cross-validation process, each data point is used only once for testing but k-1 for training. We can then evaluate the performance of the classifier by taking an average of the accuracy score of all the models of all k iterations.

[7] Models that do not include relevant or predictive features are biased.
[8] Models that include irrelevant or features that are not good predictors of general patterns in data have high variance.
[9] In our model we use k=5

3.3 Multinomial Naive Bayes

Multinomial Naive Bayes is a special case of Naive Bayes that is designed for text classification. In spite of its simplicity, and its assumption of conditional independence[10], this model often performs well in supervised learning problems. Similar to Naive Bayes, the multinomial naive bayes model assumes that the probability of each word in the document is independent of the word's context and position in the document. Thus each document is drawn from a multinomial distribution of words with as many independent trials as the length of the document or review.

Multinomial Bayes	**Classification Report**			
Target	**Precision**	**Recall**	**F1-score**	**Support**
y=0	0.76	0.67	0.71	1024
y=1	0.91	0.94	0.92	3505
Avg/Total	0.87	0.88	0.87	4529

Table 5

3.4 Random Forest Classifier

Random Forests are an ensemble learning method for classification and regression. They are a way of averaging a multitude of deep decision trees, and in the process reducing variance. The idea is to compute k number of trees[11] using a random sample of the data for each tree. Furthermore, for each split in each tree, only a random sample of the feature space is available. This is because, we want each tree to be as uncorrelated as possible. So in addition to the randomness of the data, feature variables are also randomly selected at each split. The final result is then computed by averaging over all the trees.

We build a pipeline with CountVectorizer and a Random Forest Classifier. To tune the hyperparameters, minimum and maximum number of documents, uni-grams or bigrams, and number of trees for the Random Forest, we perform a k-fold cross-validation with GridSearch and accuracy score as score function. Prior to training we balance the data, using Scikit-learn's classweight functionality. From table 6, the classifier identifies 82%

[10] Naive Bayes assumes conditional independence in the feature variables. Nevertheless, this is a weaker assumption than the full independence assumption implicit in Logistic or Linear Regression.
[11] Number of trees is a hyperparameter and needs to be predetermined.

of the ‘good’ reviews correctly and 83% of the ‘bad’ reviews. So the model is correctly identifying the same proportion of both reviews.

Random Forest	**Classification Report**			
Target	**Precision**	**Recall**	**F1-score**	**Support**
y=0	0.57	0.83	0.61	1024
y=1	0.94	0.82	0.92	3505
Avg/Total	0.86	0.82	0.83	4529

Table 6

3.5 Logistic Regression

Logistic Regression is a supervised learning method for binary classification. The output of the logistic regression model is interpreted as the log-odds ratio of belonging to a particular class. In other words, it estimates the probability of class membership over a categorical class.Similar to naive bayes, logistic regression can overfit the data, leading to high variance. To detect this, we perform k-fold cross validation using GridSearch and accuracy as our score to test the performance on the test data. To control overfitting, we need to add a regularization parameter in the classifier. We do an exhaustive search over a range of values for the inverse of the regularization parameter C from 0.001 to 100[12]. Again, as in random forest, we balance the classes prior to training.

Logistic Regression	**Classification Report**			
Target	**Precision**	**Recall**	**F1-score**	**Support**
y=0	0.74	0.71	0.73	1024
y=1	0.92	0.93	0.92	3505
Avg/Total	0.88	0.88	0.88	4529

Table 7

[12] Note that C in this case is the inverse of the regularization, so high value of C indicate weak regularization and vice-versa.

From table 7, the logistic regression is capturing 71% and 93% of the ‘bad’ and the ‘good’ reviews.

3.6 Support Vector Machines (SVM)

SVMs construct linear separating hyperplanes in high-dimensional vector spaces. Optimal classification occurs when such hyperplanes provide maximal distance to the nearest training data points. Concretely, instead of separating with a line, idea is to first fit the fattest bar between the classes, and once this is found, the linear discriminant will be the center line through the bar. Thus, for the scenario where the classes are linearly separable, the objective function of the SVM is to maximize the margin between the two classes.

As before, we balance the classes and build a pipeline with the CountVectorizer method to convert the documents into a word-count matrix and the SVM as the binary classifier to train the model. We perform a k-fold cross-validation with GridSearch to tune our hyperparameters: minimum and maximum values of the number documents, uni-grams or bi-grams and the hyperparameter alpha for the support vector machine. We do an exhaustive search over a range of values for the hyperparameters using accuracy score as our score function.

From table 8, the algorithm identifies almost 89% of the ‘good’ reviews correctly, and 80% of the ‘bad’ reviews. The precision is above 69% and 94% for both y=0 and y=1 repectively.

SVM	**Classification Report**			
Target	**Precision**	**Recall**	**F1-score**	**Support**
y=0	0.69	0.80	0.68	1024
y=1	0.94	0.89	0.92	3505
Avg/Total	0.88	0.87	0.88	4529

Table 8

3.7 TF-IDF

TF-IDF stands for term frequency inverse-inverse document frequency. Term frequency measures how often a word appears in a document. Since this will naturally be influenced by the length of the document, it is usually divided by the length of the document. Inverse document frequency on the other hand measures how important a word is. We would like to give less weight to words that occur frequently while scaling up the more rare words. This is achieved by computing the IDF has follows: taking logarithm of the ratio of total number of documents to the number of documents with a particular word[13]. The TF-IDF is then calculated by taking the product of TF and IDF. This can be done using Scikit-learn's TfidfVectorizer method.

We build a pipeline with TfidfVectorizer to convert the word document in to a term frequency matrix and the classifier Logistic Regression. We perform a GridSearch over hyperparameters for both the TfidfVectorizer and the logistic model. For the TfidfVectorizer, we do an exhaustive search over different values for the minimum and maximum number of documents, uni-grams and bi-grams, and to scale or not to scale the term frequency by the inverse document frequency. For the logistic regression, we do a search over different values of the regularization parameter.

From the table 9 below, we see that, the model does well for the class y=1. For the class y=0, it correctly identifies about 55%. The precision for y=0 is pretty good at 82%.

TF-IDF	**Classification Report**			
Target	**Precision**	**Recall**	**F1-score**	**Support**
y=0	0.82	0.55	0.66	1024
y=1	0.88	0.97	0.92	3505
Avg/Total	0.87	0.87	0.86	4529

Table 9

3.8 Topic Modeling

In Natural Language Processing, topic modeling is a kind of unsupervised learning model for discovering what "topics" occur in a collection of documents. Topics produced by topic modeling can be interpreted as clusters of similar words. It can be described as a method of finding a group of words, i.e. topic, in a collection of documents that best describes the content of the document. There are various algorithms to find topics in documents, and we look into two of them, Latent Dirichlet Allocation (LDA) and Latent Semantic Analysis[14] (LSA).

[13] IDF(t) = log(Total number of documents / Number of documents with term t in it).

[14] This is also sometimes referred to as Latent Semantic Indexing or LSI

3.8.1 Latent Dirichlet Allocation (LDA)

In LDA, each document can be viewed as a mixture of topics, where each topic is considered to have a set of words or phrases allocated by the LDA. In other words, LDA is a statistical machine learning model, where words are clustered into “topics”, and documents are clustered into “mixture of topics”. Similar to various clustering algorithms like KMeans, LDA does not give us the optimal number of topics. In this paper, our goal is to reduce the dimension of the feature matrix X using LDA, which will then be used in a supervised learning classification model.

We convert the documents into a matrix of word counts with Scikit-learn’s CountVectorizer. We set the parameters minimum number of documents = 5, that is words that occur in 5 or less documents is not included in the corpus. We also set the parameter maximum number or fraction of documents to 0,5, that is words that occur frequently or in more than 50% of documents is deleted from the corpus. To find the optimal number of topics, we use Scikit-learn’s pipeline functionality to set up our model parameters. We build the pipeline with the following:

a) LDA to cluster the words into topics, and documents into “mixture of topics”
b) Logistic Regression to classify the documents, using the output of the LDA model, that is, the reduced dimension topic distribution as the feature matrix and target variable y.

We perform a K-fold cross-validation with GridSearch and accuracy as a score to test the performance on the hold-out test data. Again, prior to training, we balance the classes. We do an exhaustive search over a range of hyperparameters: a range of values between 100 and 1000 for number of topics, and for the inverse of the regularization parameter C from 0.001 to 100[15].

LDA and LR	**Classification Report**			
Target	**Precision**	**Recall**	**F1-score**	**Support**
y=0	0.49	0.77	0.60	1024
y=1	0.92	0.77	0.84	3505
Avg/Total	0.82	0.77	0.78	4529

Table 10

[15] Note that C in this case is the inverse of the regularization, so high value of C indicate weak regularization and vice-versa.

From Table 10, precision for y=1 is 0.92 while recall is 0.77 for both y=0 or y=1. So the model is correctly identifying around 77% of the 'bad' and 'good' categories'. Compared to the previous models, it is only correct 49% of the times the classifier labels a review 'bad'.

3.8.2 Latent Semantic Analysis (LSA)

Latent Semantic Analysis (LSA), as the name suggests is the analysis of latent, that is, hidden semantics in a corpora of text. As seen previously, a collection of documents can be represented numerically in word-count or term-frequency matrix, and two documents can be inferred to be similar or dissimilar by using cosine similarity. LSA transforms the original matrix into a lower dimensional matrix by Singular Value Decomposition or SVD of the original matrix. This is similar to PCA analysis, where we find linear combinations of the feature variables such that the components are uncorrelated with each other, and then use the top n components with the highest explained variance given by the eigenvalues. In our problem, we apply LSA to reduce the dimension of the term-document matrix and then use the reduced feature matrix to perform classification. We reduce the dimension to 1000 components as this gives a explained variance ratio of 90.93%. We do a GridSearch over a range of values for the regularization parameter C, after balancing the classes.

SVD and LR	**Classification Report**			
Target	**Precision**	**Recall**	**F1-score**	**Support**
y=0	0.63	0.83	0.72	1024
y=1	0.95	0.86	0.90	3505
Avg/Total	0.88	0.85	0.86	4529

Table 11

From table 11, the recall is 0.83 and 0.86 for class 'bad' and class 'good' respectively. That is the model is identifying 83% of y=0 and 86% of y=1. For precision, the model is correct 95% of the times its classifies a document as 'good', and 63% when it classes a document as 'bad. This is improves on the LDA model where precision for y=0 is 0.49.

3.9 Accuracy and ROC

Table 12 gives the accuracy score of training and testing data, as well as the ROC score. For supervised learning, all the classifiers are doing good with a test accuracy of

0.86 and above. Logistic regression and the random forest have a similar ROC with 0.82, with multinomial naive bayes coming in second with a marginally lower ROC score at 0.80. The training and test accuracy score for the SVM is 0.97 and 0.87, indicating there is some over-fitting of the data. But the SVM has the highest ROC score of 0.85. The CountVectorizer does better than the TF-IDF with the Multinomial Naive as the classifier[16]. This is probably because the documents are not too long, and term frequencies with inversely weighted word document frequency are usually useful for long documents where simply using word counts may give misleading results. For unsupervised learning, reducing the dimension with LSA gives a train and test accuracy score of 0.87 and 0.85. Again this indicates there is no over-fitting of the data. In addition, the ROC is higher at 0.84. Dimension reduction with LDA does not do as well, with a test accuracy of 0.77 and a ROC at 0.77.

Model	**Accuracy (Training)**	**Accuracy (Testing)**	**ROC**
Multinomial NB	0.99	0.88	0.80
Random Forest	0.82	0.82	0.82
Logistic Regression	1.00	0.88	0.82
SVM	0.97	0.87	0.85
TF-IDF (Multinomial)	0.93	0.87	0.76
LDA (Logistic)	0.78	0.77	0.77
LSA (Logistic)	0.87	0.85	0.84

Table 12

4. Conclusion and Future Research

This paper analyzes text review data from a Women's ecommerce clothing line and classifies the review as 'good' or 'bad' using various supervised classification algorithms. In addition, we apply dimensional reduction to the feature matrix by using topic modeling like LDA and LSA, and then train the reduced feature matrix using supervised learning. We achieve an accuracy of 0.88 on the test data and ROC of 0.82 using supervised learning. WIth topic modeling and classification, we actually improve

[16] Unless explicitly mentioned, we have used CountVectorizer to convert the text data to a matrix of word counts.

the ROC to 0.84 and test accuracy of 0.85. With a training accuracy of 0.87, this clearly shows that reducing dimensionality has helped reduce the problem of over-fitting.

In addition to labeling a review as ‘good’ or ‘bad’, it would be useful to the retailer to assign some degree of usefulness to the review[17]. The variable ‘Positive Feedback Count’ gives some idea of the usefulness of the review. But older reviews will naturally have more counts than fewer reviews. Additional information about date of the review, as well as ‘negative feedback count’ will help in building a model that addresses the usefulness of a review.

One could also apply topic modeling using LDA, and find the optimal number of topics using some measure like perplexity. The results of the topic modeling can then be used to make recommendation to customers. Reviews can also be sorted based of the results of the topic modeling and the preferences of the customer[18]. This will help drive sales, as both products and reviews can be appropriately targeted.

[17] According to UX expert Jared Spool, Amazon’s question “was this review helpful?” was responsible for $2.7 billion revenue each year.
[18] For example, reviews can be categorized by topics using topic modeling, and customers more inclined to shop jackets will be recommended ‘useful’ reviews in that particular category.