Amazon Food Review Analysis

Submission by Aditya Athota

Abstract

This document is about classifying the reviews from the given dataset into positive review or negative review based on ratings given by the users. The dataset consists of reviews of fine foods from Amazon. Here, we will develop a model to classify the given ratings either to positive or negative review based on the ratings given by the customer. This model will be helpful in recommendation systems such that the new customer will find it helpful to purchase a product.

1 Introduction

Online platforms and social media are now being used by businesses to sell their products and services. The growing usage of social media and the internet has drastically altered how people shop for goods. Customers can also get improved access to product information through social media. Customers trust online consumer evaluations more than the information offered by vendors, according to studies (Salehan et al., 2016). Furthermore, online customer reviews are more user-centric and describe products from the perspective of the user.

Consumers can assess product quality by reading and analyzing various online reviews available on public sites such as Amazon.com. However, if these reviews are not properly organized or displayed, clients may be left in an uncertain situation. Because of the inconsistent quality or untruthfulness (spam) of the evaluations, the overwhelming amount of reviews can influence consumers' unbiased decision making (Salehan et al., 2016; Liu et al., 2008). To address this issue, Amazon.com includes a 'helpful' field beside the review text that allows users to rate how useful the information supplied in the review was. Consumers can use this as a guide to help them make an informed purchase. Amazon evaluations are frequently the most visible consumer product reviews. As a frequent Amazon customer, I was intrigued by the idea of visualizing the structure of a vast collection of Amazon reviews in order to become a more informed consumer and reviewer.

The 'Helpfulness Voting' mechanism is anticipated to bring in 2.7 billion dollars in additional income for Amazon.com (Cao et al., 2011). It is critical for any firm to understand which elements influence the usefulness of internet reviews. This can assist internet business owners in increasing revenue by building and deploying an automated system for categorizing a large amount of data from online consumer reviews.

Although this voting procedure is a step forward in the development of an automatic classification system, there are a few drawbacks that must be addressed before moving forward. First, recent reviews may have a small number of votes or none at all, making determining the usefulness of these evaluations difficult. Furthermore, the most popular reviews are shown first, followed by newly published reviews that have yet to be voted on (Liu et al., 2008). In their study, Kim et al. (2006) discovered that 38 percent of 20919 Amazon evaluations for all MP3 player items received three or fewer helpfulness ratings. It is also well recognized that human interpretation of textual material is usually skewed, as users want to believe what aligns with their beliefs (Liu et al., 2012).According to studies, the ratings that reviews receive do not transmit significant information because they are either exceedingly high or incredibly poor (Ghose et al., 2007).

Given the foregoing, it is critical to develop an automatic classification system that forecasts the usefulness of online consumer evaluations regardless of when they are posted. If the system built on machine learning classification algorithms instantly classifies newly uploaded reviews as 'Helpful' or

’Not helpful,’ it gives buyers a logical choice to make their purchasing decisions rather than relying entirely on older evaluations. In the past, text mining and Natural Language Processing (NLP) were utilized to create a text classification system to identify spam reviews (Crawford et al., 2015; Shojaee et al., 2013; Lilleberg et al., 2015). The same method can be used to determine whether or not the reviews are useful.

2 Dataset

This dataset contains Amazon reviews of exquisite meals. The data spans more than a decade, with all 500,000 reviews up to October 2012 included. Product and user information, ratings, and a plain-text review are all included in reviews. We also have feedback from all of Amazon’s other categories.

The dataset is available in two forms (1) .csv file and (2) SQLite Database In order to load the data, We have used the SQLITE dataset as it easier to query the data and visualize the data efficiently.

3 Processing the Dataset

Before processing the dataset, we need to clean the dataset as many duplicate entries were discovered. Duplicate entries need to be removed as they cause unbiased results. After removing the duplicates, it is required to preprocess the data before making the prediction model. The following steps are followed to preprocess the data:

HTML Tags

If the reviews or texts were scraped from the internet, they are likely to contain HTML tags. These tags aren’t relevant for our NLP activities, thus it’s best to get rid of them.

Removal of punctuation marks

While extracting the semantics from the text, the punctuation marks are removed from the text because they contribute no more information.

Converting uppercase tokens into lowercase

Because the semantics of a word or phrase are unaffected by the case in which it is expressed, uppercase words are changed to lowercase to minimize potential word repetition. The phrases ’DOG’ and ’dog,’ for example, have the same meaning. This conversion aids in the reduction of the feature set’s dimensionality.

Numbers

The next step is to eliminate the numbers. We may toggle on or off the steps by setting parameters to True or False values, as we’ll see later. For sentiment analysis, removing numbers may make sense because numbers offer no information about sentiments. However, we will not want to delete numbers if our NLP objective is to extract the number of tickets ordered in a message to our chatbot.

Stop Words

Stopwords, as previously said, are relatively common words. Words like ”we” and ”are” are unlikely to aid sentiment analysis or text classification in NLP applications. As a result, stopwords can be removed to save computer time and effort while processing vast amounts of text.

We used spaCy’s built-in stopwords in our scenario, but we need be cautious and adjust the stopwords list accordingly. For example, the word ”not” is crucial in the meaning of a sentence like ”not good” for sentiment analysis.

Expand Contractions

Don’t and can’t are two examples of contractions. Text can be standardized by expanding such terms to ”do not” and ”cannot.” To expand the contractions, we use the contractions module. This step is optional depending on your NLP goal because spaCy’s tokenization and lemmatization routines will expand contractions like can’t and don’t in the same way. The only difference is that spaCy expands ”we’re” to ”we be,” whereas contractions result in ”we are.”

Snowball Stemming

”Different versions of the same word are mapped to a common ”stem” in stemming; for instance, the English stemmer maps connection, connections, connective, connected, and connecting to connect. As a result, a search for connected would also turn up documents that just contain the other types.”

Snowball is a simple string processing language that can be used to create stemming algorithms for use in information retrieval, as well as a set of stemming algorithms that use it.”

Generally, the word will be the stem form but in ”text search systems”, this is not the case that occurs frequently. We also try to combine terms which has similar meanings instead of conflating all words with the same linguistic origin (so awe and horrible don’t have the same stem). It is better to avoid in difficult scenarios as over-stemming can create more trouble than under-stemming. Snowball’s stemming algorithms aren’t the best choice if you want to always reduce words to a root form and/or get a root form that is also a word.

Tokenization
Tokenization is a process that divides lengthy strings of text into smaller tokens. We can tokenize paragraphs into sentences and senteces into words. We will proceed further only after the word or sentence has been tokenized properly. Text segmentation or lexical analysis are other terms for tokenization. Segmentation can be referred as breaking of a huge text into portions larger than words (e.g. paragraphs or sentences), whereas tokenization refers to the breakdown process that produces just words.

It is not a simple procedure even though it appears to be simple. *"How do sentences within bigger amounts of text become identified? You probably utter "sentence-ending punctuation" off the top of your head, and you might even believe that such a remark is clear for a moment."*

4 Transforming the Text Data

The data that is present in the dataset are text features. But, in machine learning, we require numerical vectors to develop a model.

If we convert our text features to d-dimensional numerical vectors, we can draw the normal to plane to separate positive and bad reviews. In the image below, the blue crosses represent positive reviews and the red crosses represent negative reviews, which are separated by plane W.

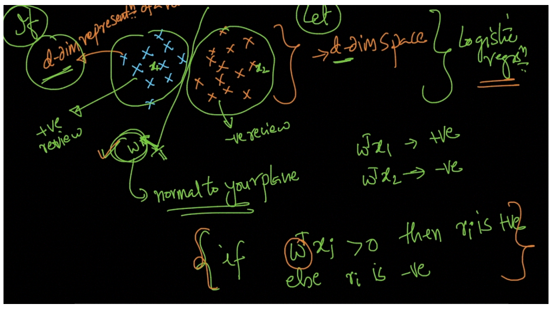

Figure 1: This is an image that explains how text features are being converted into numerical vectors. (Image is obtained from Medium Article)

If someone submits a new review for categorization, we convert it to a numerical vector and multiply it with weights and if the result is more than zero, we consider it a positive review else if the result is less than zero, we consider it a negative review as all points in the same direction of the normal to the plane are considered positive in linear algebra, and all points in the opposite direction of the normal to the plane are considered negative. Therefore, in order to convert text features to numerical vectors, we use some of the techniques from the NLP.

"Word Embeddings, also known as Word Vectorization, is an NLP technique for mapping words or phrases from a lexicon to a matching vector of real numbers, which may then be used to derive word predictions and semantics. Vectorization is the process of translating words into numbers."

The following techniques are being used to find the vector representation of given text:

Bag of Words (BoW)
A bag-of-words model, or BoW for short, is a method that can extract attributes so that the attributes can be used in modelling, such as machine learning techniques. The method is straightforward and adaptable, and it may be used to extract information from texts in a variety of ways. A bag-of-words is a text representation that specifies the frequency with which words appear in a document. It entails two steps: A lexicon of well-known terms, A metric for determining the existence of well-known terms. Because all information about the sequence or structure of words in the text is deleted, it is referred to as a "bag" of words. The model simply cares about whether or not recognized terms appear in the document, not where they appear. The assumption is that documents with comparable content are similar. Furthermore, we may deduce something about the document's significance only from its content. You may make the bag-of-words as basic or as complex as you like. The difficulty arises from determining how to create a vocabulary of known words (or tokens) as well as how to rate the existence of known terms.

The next stage is to score each document's words. The goal is to convert each free text document into a vector that may be used as input or output for a machine learning mode. We can use a fixed-length document representation of 10 with one place in the vector to score each word because we know the vocabulary comprises 10 words. The simplest scoring approach is to assign a boolean value to the existence of words, with 0 indicating absence and 1 indicating presence. We may walk through the first document ("It was the best of times") and convert it to a binary vector using the arbitrary ordering of terms specified above in our vocabulary. All word orders are theoretically ignored, and we now have a consistent method for extracting characteristics from each document in our corpus, ready for modeling.

The limitations of BoW is that the meaning of the term in the paper is ignored by the basic BOW technique. It pays no attention to the context in which it is employed. Depending on the context or neighboring terms, the same word might be used in different places and the vector size for a long text might be enormous, requiring a lot of calculation and effort. Words that aren't relevant to your use case may need to be ignored.

Term Frequency - Inverse Document Frequency
"Term Frequency — Inverse Document Frequency" is abbreviated as TF-IDF. This is mainly used for quantifying a word in the given text. Based on the word value in the document, we assign the weight to the word. This approach is widely used in the disciplines of information retrieval and text mining. TF-IDF is basically a multiplication of two terms – Term Frequency and Inverse Document Frequency.

"Term Frequency (tf) is the number of times a term appears in each document in the corpus. It's the proportion of the number of times a word appears in a document to the total number of words in the document. It rises in proportion to the amount of times that term appears in the manuscript. There is a separate tf for each document."

$\text{tf}_{i,j} = n_{i,j} / \sum_k n_{i,j}$
(Source from Medium Article)

The term Frequency tells us how frequently the word 'w' appears in document d. Tf will grow if the word 'w' is used more frequently.

"Inverse Data Frequency (idf) is similar to tf in that it gauges the relevance of a document in a corpus as a whole. The sole distinction is that DF is the count of occurrences of word t in the document set N, whereas tf is a frequency counter for a term t in document d."

idf(w) = log(N / df$_t$)

The weight of unusual terms across all documents in the corpus is calculated using idf. The idf score of terms that appear seldom in the corpus is high.

We compute idf even though the tf value provides a vectorized form because, even if we calculated the tf value, there are still a few issues. For example, words like "is, are" will have very high values, giving such words a very high priority. However, when these terms are used to calculate relevance, the results are poor. Although we will eliminate stop words later in the preprocessing stage, calculating the word's relevance across all texts and normalizing using that value depicts the documents much better.

Finally, we get the tf-idf score by multiplying the values of tf and idf. There are many other versions of tf-idf, but for now, let us focus on the most basic one.

$\text{w}_{i,j} = tf_{i,j} \times log(N/df_i)$

$\text{tf}_{i,j} =$ $number\ of\ occurences\ of\ i\ in\ j$
$df_i =$ $number\ of\ documents\ containing\ i$
$N =$ $total\ number\ of\ documents$
$(SourcefromMediumArticle)$

The limitations of TF-IDF is, like BoW it also disregards the word's semantic meaning. It pays no attention to the context in which it is employed. Depending on the context or neighboring terms, the same word might be used in different places.

Word2Vec
Mikolov et al. (2013) at Google were the first to come up with this notion. This approach represents words in a vector space with hundreds of dimensions using 2-layered shallow neural networks. CBOW (Continuous Bag of Words) and Skipgram are two versions of this technique.

The CBOW model predicts a word based on the context words around it, whereas the Skip-gram model predicts context words based on the present word. The one-hot encoder vectorizes the words before they are entered into the model. The first word in the vocabulary corpus V, for example, will contain a vector of components X1, X2,......, Xv, where X1 is 1 and all subsequent elements are zero. The back- 13 propagation method is utilized to train word vectors, and the Softmax function is applied at the output layer to calculate the likelihood of the result (Rong X., 2014). The weights between the projection layer and the output layer reflect the constituents of word vectors for the words that are expected to be predicted once the model has been trained. When it comes to capturing the semantic link between the words, Skip-gram is rated superior in terms of performance (Mikolov et al., 2013). A document or text must be represented in the form of the words it contains in a text categorization challenge. Calculating the element-wise arithmetic mean of the word vectors, which provides a single vector for the page, is one option. Each of the document vector's elements can therefore function as a separate feature.

Average Word2Vec
Initially, we train our model using Word2Vec before using the sentence vectors. Then, the average of the Word2Vec vectors is calculated. We can simply take the vectors of all the words in a phrase and average them. This average vector will represent your sentence vector.

We need a huge text corpus that generates a vector for each word. It tries to automatically learn the connection between vectors from raw text. The more dimensions it possesses, the more information-rich the vector is.

TF-IDF weighted Word2Vec
Lilleberg et al. (2015) employed this method in their text categorization research. A basic Word2vec method just returns the vector or location of a word in an n-dimensional vector space, but it doesn't tell you how important the word is or how often it appears in a text. The goal behind this method is to take word significance into account as well as its vector representation. The word vectors of a specific document may be multiplied with the appropriate TF-IDF values in a text classification issue, and then the average can be taken to produce a single document vector. The following is a mathematical representation of this:

$\mathrm{V}_D = 1/w \times \sum_{w\epsilon D} V_w TI_{w,D}$

$\mathrm{V}_D = \ Document\ vector$
$V_w = \ Word\ vector$
$TI_{w,D} = TF - IDF\ value\ of\ the\ word\ for\ that\ document$
$(Source from Medium Article)$

5 Training the Dataset

After, transforming the text features to numerical vector, I am developing models based on KNN. The K-nearest neighbor method (KNN) is a supervised learning technique that has been utilized in a variety of applications including data mining, statistical pattern recognition, and more. When an input is supplied, K-nearest neighbour utilizes the training set to categorize it. When a k-nearest neighbor method is applied to an input with d characteristics, based on the majority vote of the k (k is user-specified constant), the input is categorized, training closest records across all d attributes. The term "closest" refers to the distance between an attribute and the same attribute in the training set, as measured by a similarity metric. K-closest neighbour (kNN) is a method of calculating the nearest neighbor based on the number of k, which determines how many nearest neighbors should be considered when defining the class of a sample data point. Based on the distance available between training points and sample data points, the weights are applied to the training points. However, the primary concern will always be computational complexity and memory requirements. In the general model of a KNN query, the user provides a variety of query forms, such as a point query in multidimensional space and a distance metric for measuring distances between points in this space. The system tries to find the K closest or closest responses in the database based on the query (i.e. query point). Some of the examples of the distance measurements are Euclidean distance, Manhattan distance. The most commonly used distance measurement is Euclidean distance.

The following is the algorithm for computing the K-nearest neighbors:
1. Determine the parameter K, which equals the number of closest neighbors, ahead of time.
2. Find the distance between the query-instance and each of the training samples. Any distance algorithm can be used.
3. Sort the distances for all of the training samples and use the K-th minimal distance to find the closest neighbors.
4. Because this is supervised learning, gather all the Categories from your training data that fall under K for the sorted value.
5. As the query instance's prediction value, use the majority of nearest neighbors.
6. Nearest Neighbour (approximate) (ANN)

Obtaining a "good estimate" of the nearest neighbor may be sufficient in some cases. In some cases, we can use a method that doesn't always return the true nearest neighbor in exchange for faster processing or less memory usage. In most cases, such an algorithm will ignore the nearest neighbor, but this is highly dependent on the dataset in question.

Here, I am approaching in two ways to build KNN model. They are: (1) K-d tree and (2) Brute Force.

K-d tree
A k-d tree, also known as a k-dimensional tree, is a data structure for arranging a large number of points in a k-dimensional space. It's a binary search tree with some additional limitations. For range and nearest neighbor searches, K-d trees are quite beneficial. The whole simulation volume is represented

by the root-cell of this tree. The remaining cells are rectangular sub-volumes containing the mass, center-of-mass, and quadrupole moment of their enclosing areas. It was one of the first structures to be utilized for multiple-dimensional indexing. The partitioning is done along one dimension of the node at the top level of the tree, along another dimension in nodes at the next level, and so on, iterating through the dimensions. The splitting is carried out in such a way that about half of the points contained in the subtree fall on one side and half on the other at each node. When a node has fewer than a specified maximum number of points, partitioning ends.

Their goal is to divide space into a small number of hierarchical cells, with no single cell containing too many input items. This enables you to access any input object quickly based on its location. We progress through the hierarchy until we find the object's cell. Typical techniques split point sets recursively along different dimensions to create k-d trees. A plane across one of the dimensions defines each node in the tree, dividing the set of points into left/right (or up/down) sets, each containing half of the parent node's points. These children are divided into equal halves once more, but this time with planes in a different dimension. Each point is placed in its own leaf cell after log n layers of partitioning.

K-d Tree Algorithm:*"The k-d tree is a binary tree with each node representing a k-dimensional point. Every non-leaf node may be thought of as creating an implicit splitting hyper-plane that divides the space into two half-spaces. The left subtree of that node is represented by points to the left of this hyper-plane, while the right subtree is represented by points to the right of the hyper-plane. The hyper-plane orientation is determined as follows: each node in the tree is assigned to one of the k dimensions, with the hyper-plane perpendicular to the axis of that dimension. If the "x" axis is chosen for a specific split, all points in the subtree with a smaller "x" value than the node will show in the left subtree, while all points with a bigger "x" value will appear in the right subtree. In this situation, the hyper-plane is determined by the point's x-value, and its normal is the unit x-axis."*

The nearest neighbor search (NN) technique attempts to locate the tree node that is closest to a given input point. The tree attributes may be used to swiftly remove huge areas of the search space, making this search more efficient. In a k-d tree, searching for a nearest neighbor goes like this:
1. Beginning at the root node, the method recursively proceeds down the tree, as if the search point were being added (i.e., it goes left or right depending on whether the point is less than or larger than the current node in the split dimension).
2. When the algorithm reaches a leaf node, it records the "current best" node location.
3. The algorithm unwinds the tree's recursion at each node by: (a) The current node becomes the current best if it is closer than the existing best. (b) The algorithm looks for locations on the other side of the splitting plane that are closer to the search point than the current best. In theory, this is accomplished by crossing the splitting hyper-plane with a hyper-sphere having a radius equal to the current closest distance around the search point. Because the hyperplanes are all axis-aligned, this is implemented as a simple comparison to determine if the distance (overall coordinates) between the search point and the current best is smaller than the difference between the splitting coordinate of the search point and the current node.

If the hyper-sphere crosses the plane, there may be closer points on the other side, thus the algorithm must go down the other branch of the tree from the current node, using the same recursive procedure as the whole search, seeking for closer points. The algorithm continues traveling up the tree if the hyper-sphere does not meet the splitting plane, and the whole branch on the other side of that node is deleted.

The search is finished when the algorithm completes this step for the root node.

Brute Force
Brute-force search, also known as exhaustive search, is a problem-solving approach that entails systematically enumerating all possible solution candidates and determining if each one fulfills the problem's statement.

1. Determine all distances between the query and reference sites.
2. Sort the distances that have been calculated.
3. Choose the k reference sites with the shortest distances from each other.
4. Vote on classification by the k closest items.
5. Complete steps 1–4 for each query point.

Vectorizer	Model	Test Accuracy Score
BoW	Brute	77.83
BoW	KD-Tree	79.6
TF-IDF	Brute	78.33
TF-IDF	KD-Tree	76.07
Word2Vec	Brute	83
Word2Vec	KD-Tree	83.7
Avg. Word2Vec	Brute	83.91
Avg. Word2Vec	KD-Tree	84.0
tf idf-Word2vec	Brute	51.7
tf idf-Word2vec	KD-Tree	80.95

Table 1: Results.

6 Results and Conclusion

The following table is the results that are obtained for both the approaches of KNN with all the text featurization techniques.

After classifying both models with all the text featurization techniques, the model K-d tree with average word2vec technique has the highest accuracy of 84 percent when compared with the rest. And K-d tree proves to be better model than brute force in almost every aspect.

7 References

https://www.kaggle.com/prasheel1047/review-positive-or-negativeK-NN-using-KD-Tree-with-Simple-CV
https://medium.com/analytics-vidhya/amazon-fine-food-reviews-featurization-with-natural-language-processing-a386b0317f56
https://en.wikipedia.org/wiki/K-nearest$_neighbors_algorithm$
$https://snowballstem.org/$