***Sentimental Recommendation System.***

In this article, an implementation of Unsupervised Machine Learning concept will be discussed, wherein the emotions, thoughts and opinions extracted as a mathematical notion from the text are the determining factors of a recommendation system.

The application was performed as a project for the course DS-5230, at Khoury College of Computer Sciences under the guidance of Professor Tina Eliassi Rad, along with another Data Science aspirant - Aanand Dhandapani.

It all started with a straightforward perception of generic recommendations which then proceeded down to a complex yet simple kind of mathematical calculations leading to amazing results unthought of.

The principal focus here is to retrieve user's search query (Product & Category), based on which the user will be recommended top-n products from that category alone. The Underlying mechanism in simplest terms is to figure out the sentiments of the reviews either as positive or negative, followed by clustering unique items to decide top-k products based on higher average of connotation scores.

The full-length implementation consists of the following strides:

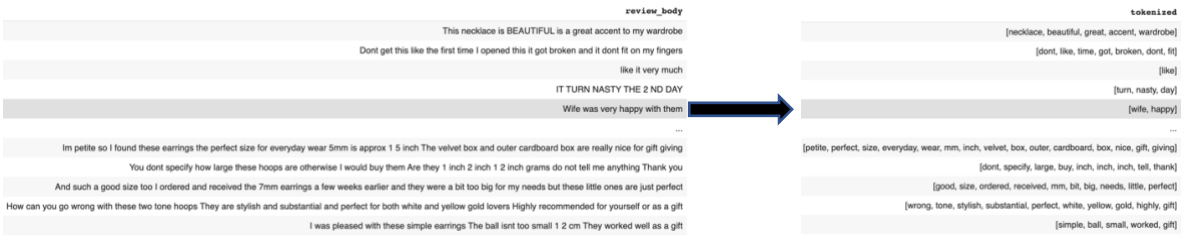
1- Preprocessing (or Tokenizing)
2- Word Embeddings, Clustering and Connotation Calculation
3- Hyperparameter Tuning and Evaluation

Datasets used: Amazon reviews datasets, segregated based on Categories. For instance, reviews for different musical instruments such as Guitar, electronic-keyboard, saxophone, violin, drums, etc. are massed together as one dataset. Likewise, each category has a dedicated dataset.

*So, here's the process:*

## Preprocessing the data

The datasets comes with 15 features such as 'marketplace', 'customer_id', 'review_id' and many more. The relevant ones are picked, and the rest are dropped along with observations with missing values.
These datasets have a 'review_body' column, which contains string sentences (text) as feedback for corresponding products.
The texts aka reviews are converted into standard form and turned into lower case. Stop words are removed using 'remove_stopwords()' function from 'genism.parsing.preprocessing' package. Words that have no semantic meaning are eliminated based on corpus from 'words()' function that is derived from 'nltk.corpus.words' package. Texts are then passed to 'simple_preprocess()' function from 'genism.utils' package that deaccents and tokenizes them into Unicode strings.



| review_body | tokenized |
|---|---|
| This necklace is BEAUTIFUL is a great accent to my wardrobe | [necklace, beautiful, great, accent, wardrobe] |
| Dont get this like the first time I opened this it got broken and it dont fit on my fingers | [dont, like, time, got, broken, dont, fit] |
| like it very much | [like] |
| IT TURN NASTY THE 2 ND DAY | [turn, nasty, day] |
| Wife was very happy with them | [wife, happy] |
| ... | ... |
| Im petite so I found these earrings the perfect size for everyday wear 5mm is approx 1 5 inch The velvet box and outer cardboard box are really nice for gift giving | [petite, perfect, size, everyday, wear, mm, inch, velvet, box, outer, cardboard, box, nice, gift, giving] |
| You dont specify how large these hoops are otherwise I would buy them Are they 1 inch 2 inch 1 2 inch grams do not tell me anything Thank you | [dont, specify, large, buy, inch, inch, inch, tell, thank] |
| And such a good size too I ordered and received the 7mm earrings a few weeks earlier and they were a bit too big for my needs but these little ones are just perfect | [good, size, ordered, received, mm, bit, big, needs, little, perfect] |
| How can you go wrong with these two tone hoops They are stylish and substantial and perfect for both white and yellow gold lovers Highly recommended for yourself or as a gift | [wrong, tone, stylish, substantial, perfect, white, yellow, gold, highly, gift] |
| I was pleased with these simple earrings The ball isnt too small 1 2 cm They worked well as a gift | [simple, ball, small, worked, gift] |

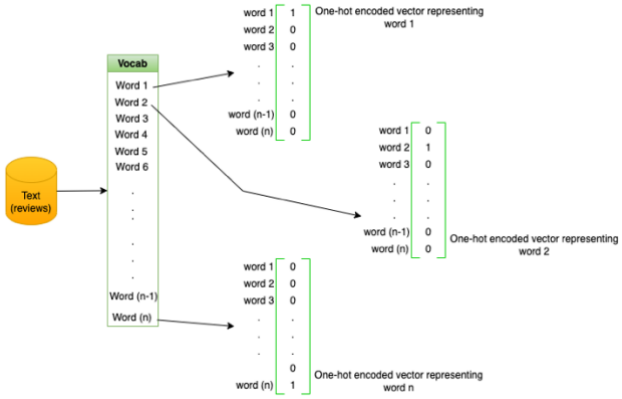
# Word Embeddings, Clustering and Connotation Scores:

To understand **word embeddings**, lets first build up on word2Vec technique and the two architectures on which word2Vec model works to generate word embeddings.

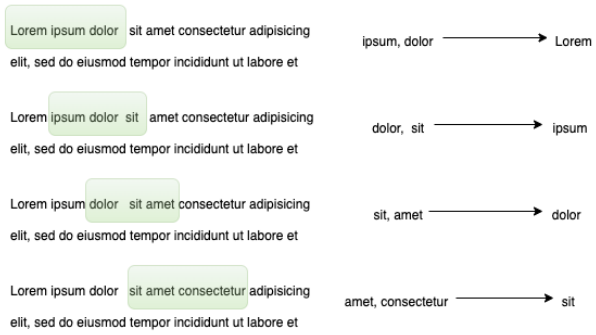
Words as input cannot be interpreted for calculations by any algorithm, hence the need to represent them as mathematical notion which is achieved by word2Vec model that converts words into vectors that near accurately represents the meaning of the word mathematically.

## Word2Vec:

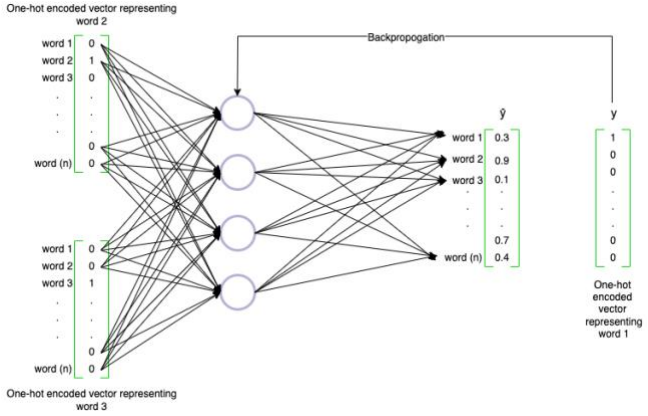
Given the bulk of text (all reviews collectively), a vocabulary consisting of unique words is created. A one-hot encoded vector is created for each word, where the number of components (rows) corresponding to each word from vocabulary is equivalent to the length of vocabulary. The word that the one hot-encoded vector represents is filled with a '1', '0' otherwise.



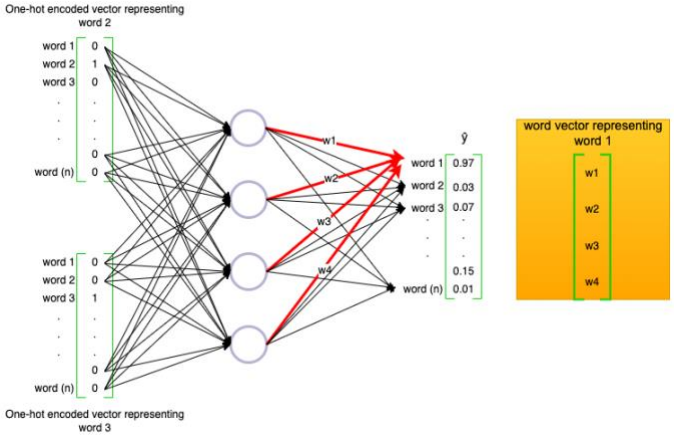

A window of fixed size slides over the given corpus of text, wherein each window is a training sample with the first word as the target and the remaining words as context that is used as input. For example, in the image below, the window size is 3, where the first word is to be predicted (target) based on the next two words (context).



The input one-hot encoded vectors (context) are given to the neural networks resulting in an output vector that is just random (not one-hot encoded) for the first epoch. The loss is calculated between predicted and actual output, propagated backwards through the network adjusting the weights to minimize the loss. This backpropagation occurs for multiple epochs until the max iterations are performed or the loss is minimal.




Once, the error is minimized after certain number of epochs, that is the predicted value is almost near to actual value, the weights pointing to the word being predicted are components representing that word. The vector of those weights is the word-vector representing the predicted word.




The approach explained above is the foundation of CBOW architecture, that stands for Continuous-Bag-Of-Words. In CBOW, the context words are given as input to the neural network based on which target is predicted. The predicted mathematical notion of the word is based on its surrounding.

The second architecture is known as skip-gram (SG), which has similar working principles as CBOW architecture with a subtle difference of input/output. In Skip-Gram, the target word is given as input based on which its surrounding words are predicted.

*"CBOW: Given the surrounding context (words), predict the word.*
*SG: Given the word, predict its surrounding context (words)"*

Now that we have understood what Word2Vec is and how does it create mathematical notion of words, let's dive into the implementation.

Post tokenization of the reviews, an object of ***Word2Vec*** is created, which comes from the package 'gensim.models'. There are some significant parameters that needs to be specified in the arguments while

creating an object of word2vec such as minimum frequency of words, the window size, size of the hidden layer in neural network, etc. Then a function named 'build_vocab()' is invoked to build vocabulary of the tokenized reviews upon which the model is trained by calling the 'train()' function.

As soon as the training of the model completes, the model object contains the high dimensional mathematical representation of each word also known as the word-embeddings. The words are represented by 300 component/feature vector (since the default size of hidden layer is 300).

Now, K-means clustering is performed on the word embeddings to cluster them into two groups and determine their centroids, using the ***KMeans*** model from the 'sklearn.cluster' package. The centroid word-vectors are retrieved using the parameter 'cluster_center_' accessible by KMeans object. The first centroid word-vector represents the cluster containing positive words while the second represents that of negative. The words in the vocabulary of Word2Vec model object is labeled as either positive (1) or negative (-1), depending upon the centroid that the word-embeddings are closest to.

Once, the words are determined as positive or negative, the connotation scores of each review are calculated to determine the sentiment of the reviews. **Connotation score** is nothing but the addition of 1's for each positive word and -1's for each negative word in a particular review.

For example, if the review is *"Unsupervised Machine Learning is a mandatory course for data science graduates"* and if according to word embeddings and its distance to centroids, for instance, following are the labels for specific words:
{Unsupervised: +1, Machine: +1, Learning: -1, mandatory: +1, course: -1, data: +1, science: +1, graduates: -1}

So, adding the labels, (+1) + (+1) + (-1) + (+1) + (-1) + (+1) + (+1) + (-1) = 5 - 3 = 2 > 0, hence a positive review (labeled 1), negative otherwise (labeled 0). This is the predicted sentiment of the review.

There are no actual values to contrast the results in order to evaluate the performance, so a notion of 'high raters, well-wishers' was considered. It implies that the users that rated a product highly would also provide positive feedback. It would be a rare case to observe an opposite pattern where the product is rated high while also being criticized. So, the true value was feature engineered to binary class, labeled as 1 if the product was rated 4 or above and labeled as 0 if the product was rated less then 4. Based on this True value and its contrast with the Predicted Value determined by the connotation score, the hyperparameter tuning was performed and evaluation metrics were calculated.

**Hyperparameter Tuning and Evaluation**

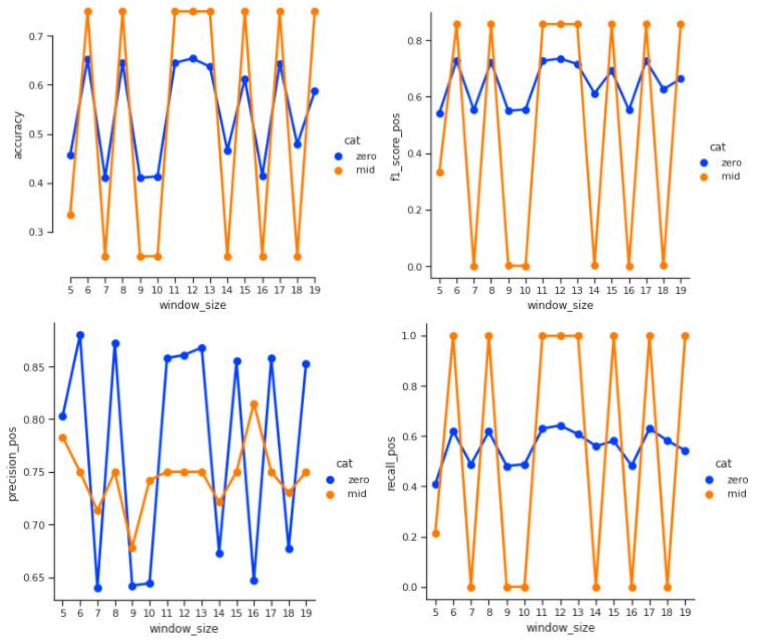
In this section of the article, the significant hyperparameters such as window size and size of the hidden layer will be emphasized. But, since the change in size of the hidden layer didn't show much of an impact on the performance (which is weird or we were unable to make out the cause), the default size was chosen (i.e 300).

Note: *The complete project was applied on two reviews datasets from Amazon (Jewelry and Watches), where the preprocessing and model training were separately executed for both the datasets. It was due to the reason that the contextual meaning of a word in one dataset (say Home Essentials) differs from the contextual meaning of a word in another dataset (say Games), and since Word2Vec determines the view of a word based on its surrounding, it made more sense to not merge the dataset because effect of the word in one surrounding would be biased or neutralized by the effect of same word in another surrounding. But the implementation is generic and can be applied to any of the review datasets.*

The determining factors were the window size, zero splits and mid-splits on the scale.
The zero-splits refer to the fact that if a connotation score was greater than or equal to zero, the review's sentiment was tagged as positive, negative otherwise.

The mid-split refers to the point of split being the average value of the largest and smallest connotation score, wherein if the connotation score of a review is greater than the split average value, than that review's sentiment is tagged as positive, negative otherwise.

The below plots show the effect of window size on Accuracy, Precision, Recall and F1-Score metrics individually for the Jewelry reviews dataset.
X-axis represents window size.
Y-axis represents performance measure.
The blue line indicates performance on zero-split.
The Orange line indicates performance on mid-split.

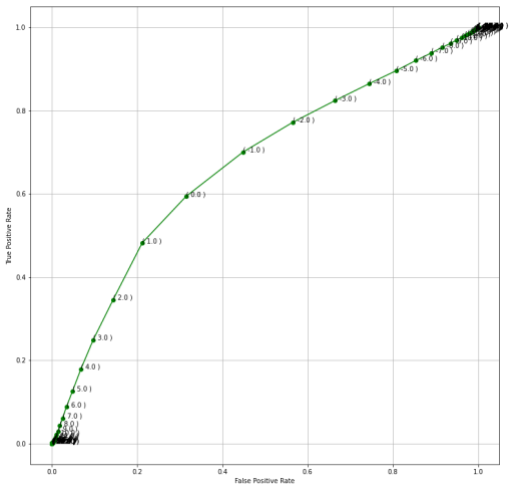



It can be inferred from the above plots that mid-split gives the best results in most of performance metrics, though in ***precision*** zero splits performed better but mid splits results were well enough almost 75% for the window sizes 12, 13 and 14. For the same window sizes, the ***f1-score*** was observed to be nearly 85% and ***recall*** was an amazing 100%. *Note that the model training and hyperparameter tuning was done with more emphasis on positive class, hence plots only evince the evaluation of positive class as we're inclined to recommending a product that had positive feedback rather than something that the general public is riled up about.*

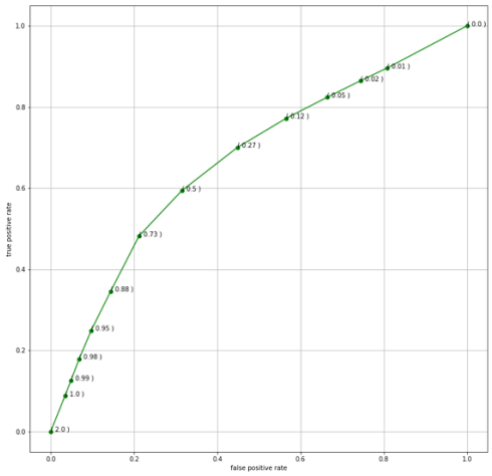
The numerical values of the evaluation metrics for the window size 11,12 and 13 for the mid split is as follows:

| window_size | cat | accuracy | f1_score_pos | precision_pos | recall_pos |
|---|---|---|---|---|---|
| 11 | mid | 0.75 | 0.857 | 0.75 | 0.999 |
| 12 | mid | 0.75 | 0.857 | 0.75 | 0.999 |
| 13 | mid | 0.75 | 0.857 | 0.75 | 0.999 |

Also, the calculated connotation scores are on the scale of integers that can range to anything from negative to positive numbers. The mid-point of this range was used to determine the sentiment of reviews. The ROC-AUC Curve for when the connotation scores are integers (below).




When the connotation scores were compressed using Sigmoid function to rational numbers in the range of 0 to 1, determining the sentiment with similar notion by setting 0.5 as the threshold, wherein the compressed values greater than or equal to 0.5 were reviews with positive sentiment, negative otherwise- showcased the same results in the evaluation metrics as prior to compression, i.e when the scores were on integer scale. The ROC-AUC Curve for when the connotation scores are compressed fractions (below).




To end with, when a user tries to search a product in a particular category, based on the determined sentiments- the average of the connotation scores of positive reviews is calculated for each unique product and top-n products with highest average scores are recommended. [Note*: average of the connotation scores is taken into account and not just the count of total positive reviews because we require the degree of extent to which the product was scrutinized by the public and not just the count of users that gave it a review*]