Comparing Word2vec, Doc2Vec model driven Sentiment Analysis using SVM, LR vs Keras CNN and Bidirectional LSTM with and without pre-trained Word and Document Embeddings

Sofia Dutta

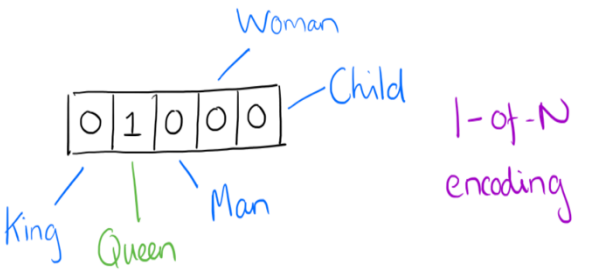
Data 602

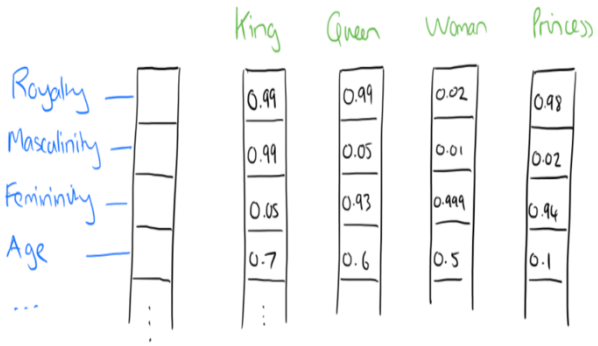
Spring 2019 – Term project

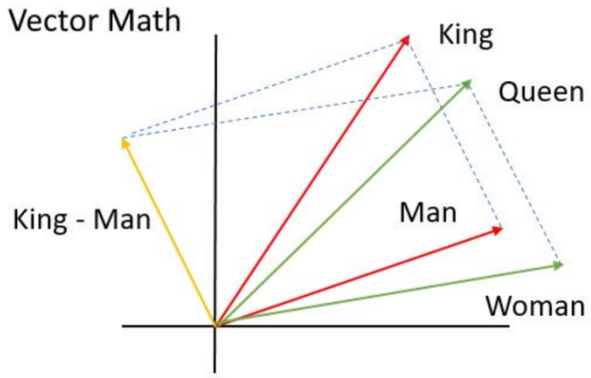
Introduction: Since the advent of the world wide web, organizations around the world have strived to create an online presence. A key motivation behind creating online presence through forums, websites or social media, is to gather customer opinion or feedback. However, manually sifting through millions of customer inputs is simply not possible, so there needs to be an automated mechanism for companies to know what all the feedback means. The solution to the problem is opinion mining or sentiment analysis.

Sentiment analysis is a natural language processing technique that aims to determine the affective context of free text. Sentiment analysis has a variety of practical applications and therefore it has been an area of great interest for the past couple of decades. It can be used in forming opinions about products, services, brands, politics, or any topic that people can express opinions about. Applications like marketing analysis, public relations, product reviews and customer service are just few of the areas where sentiment analysis could be very helpful.

Let's assume a vocabulary has five words: King, Queen, Man, Woman, Child. A one-hot encoded vector of a word in this language will have '1' in a single position to represent a specific word. All other elements will be '0'. Such an encoding can only allow comparisons in form of equality testing. Meaningful comparisons cannot be performed as each word is independent of each other. Word2Vec on the other hand represents words using a distributed representation. Each word represented by a vector is defined by the combination of its various aspects. Aspects are represented in the elements of the vector. As a result, we can have an aspect of royalty, gender, age etc. in our "little language". Once such a representation is created, we can perform algebraic operations with language. We can remove the masculinity of a King by performing vector("King") – vector("Man"). Then we can add femininity to the result vector by adding vector("Woman") to it and obtain a vector that will be closest to the vector representation of the word Queen. See image below.







One of the most popular algorithms that is used to model text data for sentiment analysis is Word2Vec. Developed by Mikolov (Tomas Mikolov, 2013), Word2Vec trains *word embeddings* using a shallow two-layer neural network. The network has one input layer, one hidden layer and one output layer. Word embedding is a language modeling technique that uses vectors with several dimensions to represent words from large amounts of unstructured text data. Word embeddings can be generated using various methods like neural networks, co-occurrence matrix, probabilistic models, etc. There are two models for generating word embeddings, i.e. CBOW and Skip-gram. Continuous Bag of Words (CBOW) model predicts the current word given a context of words. The input layer contains context words and the output layer contains current predicted word. The hidden layer contains the number of dimensions in which to represent current word at output layer. The CBOW architecture is shown in Figure 1. Skip-gram model flips CBOW's network architecture and aims to predict context given a word. Given current word, Skip gram predicts the surrounding context words. Input layer for it contains the current word and output layer provides the context words. The hidden layer consists of number of dimensions in which we want to represent current input word. The skip-gram architecture is shown in Figure 2 below.

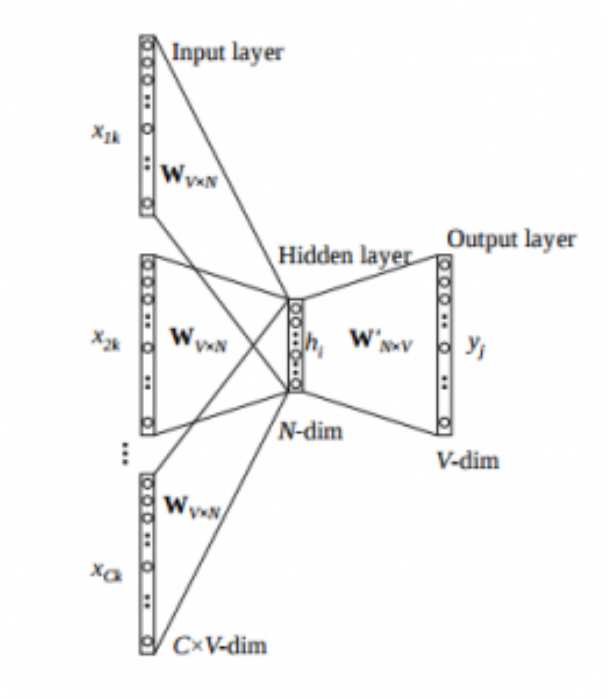


Figure 2 - CBOW Architecture

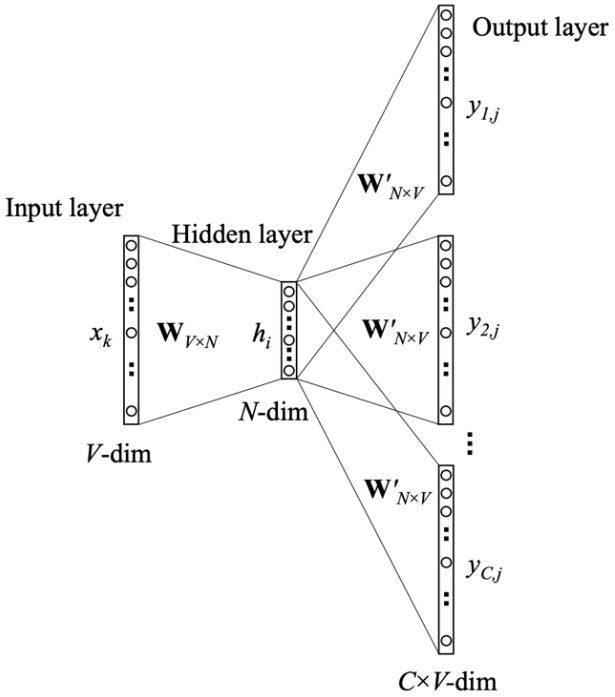


Figure 1 - Skip-gram Architecture

The Doc2Vec algorithm (Quoc V. Le, 2014), is a generalization of the Word2Vec algorithm and applies at the document level. For Doc2Vec algorithm a paragraph vector is trained using an unsupervised framework that learns continuous distributed vector representations for pieces of texts. Texts can range from a phrase or sentence to a large document. The paragraph vector can be considered as another word in the document, but it acts as memory of the model and remembers missing context like topic of the paragraph. Often called paragraph vector distributed memory (PV-DM or DM) is a model obtained by training a neural network on the task of inferring a center word based on context words and a context paragraph. See architecture in Figure 3. This model is analogous to CBOW in Word2Vec. Doc2Vec can also be modeled using paragraph vector distributed bag of words (PV-DBOW or DBOW model) which

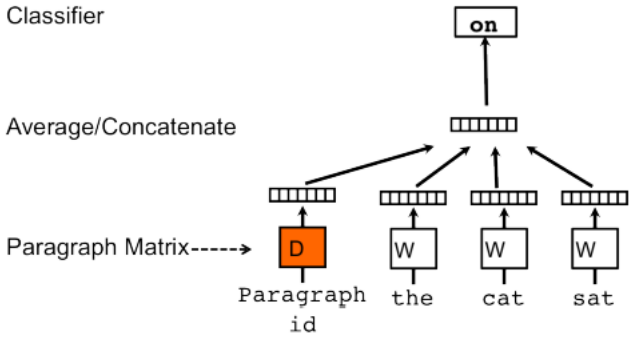


Figure 3 - DM Architecture

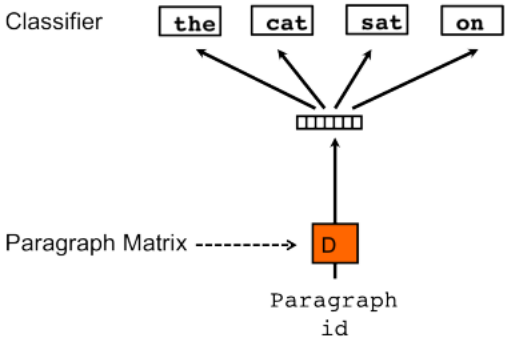


Figure 4 - DBOW Architecture

is a model analogous to Skip-gram in Word2Vec. The document vectors are obtained by training a neural network on the task of predicting a probability distribution of words in a paragraph given a randomly-sampled word from the document. See architecture in Fig 4.

Once a Word2Vec or Doc2Vec model is created one can use any conventional classification algorithms like logistic regression, support vector machine, Keras CNN, Bidirectional LSTM etc. In this paper, I have described the datasets used, methods followed and presented my results. The paper concludes with a comparison of the performance of the various classification algorithms and modeling techniques used for the sentiment classification task.

Dataset overview: In this project, I worked on applying the above-mentioned word embedding and document embedding models to carry out sentiment classification of user reviews. I performed this classification on two different datasets. The first dataset contained laptop product reviews (Hongning Wang, 2011) from Amazon's website and the second dataset consisted of movie reviews (Maas, 2011) from IMDb's website.

The first dataset of laptop product reviews from Amazon's website contained 40,762 reviews. The dataset contained over 30K positive reviews and close to 10K negative reviews. The longest positive review length contained over 20,000 characters and longest negative review contained close to 3,000 characters. The box plot of review length shows exponential properties. Looking at the graph shown in Figure 8, it can be estimated that the mass of the distribution can probably be covered with a clipped length of 900 to 1,000 words. The review dataset contains just two columns, one with the rating and the other with the review text.

The IMDb movie review dataset contains 50,000 reviews. The dataset contains equal number of reviews annotated as positive and negative. The longest positive review length contains about 14,000 characters and the longest negative review contains close to 6,000 characters. Similar to the first dataset, the box plot of review length shows exponential properties. Looking at the graph shown in Figure 7 it can be estimated that the mass of the distribution can probably be covered with a clipped length of 1,400 to 1,500 words. The review dataset contains just two columns, one with the rating and the other with the review text.

Methodology: For the sentiment classification task, I implemented Word2Vec (word2vec, 2019) and Doc2Vec (doc2vec, 2019) models using the Gensim Python library. When creating Word2Vec feature vectors I used CBOW model which is faster and produces better results for high frequency words. When creating document embeddings using DM model, I used two variants, first by averaging (Distributed Memory Mean or DMM) and then by concatenating (Distributed Memory Concatenated or DMC) the word with the context paragraph vectors. Next, I generated vectors for DBOW model. Finally, I created vectors that combined DBOW + DMC and DBOW + DMM. On the word embedding vectors, I ran Logistic Regression, SVC with linear kernel, XGBClassifier. I also ran Keras CNN with Keras's own *Tokenizer*. After that I used pre-trained word embeddings with three different classifiers, i.e. Keras CNN, Keras Bidirectional LSTM and a combination of Keras CNN and Bidirectional LSTM. Finally, I ran Keras neural network on all the document embedding vectors. Later in this document, I have presented a comparison of results for all techniques mentioned above. Next, I will dive into the details of techniques used.

Project implementation: First, I imported all the important libraries into the notebook. For each of the datasets I used python code to download, extract and load data from several files into a single JSON dictionary. I extracted the content of reviews from the dictionary and inserted into a dataframe of shape (X, 2). Here, X represents the number of reviews in the dataset. Next, I performed some data clean-up to remove any missing values and dropped rows with duplicate data. After the cleanup for Amazon dataset, I had 40,744 reviews. For the IMDb movie review dataset there were no loss of rows. I also counted number of null data points per column and found there were no null data columns. For the Amazon dataset ratings ranged from 1 to 5. If the rating was less than or equal to 2.0, I considered it as a negative review and if it was more than 2.0, I considered it as a positive review. For IMDb movie review dataset, the data was already annotated as positive or negative.

Next, I performed some data visualization and exploratory data analysis on the reviews to understand the dataset better. I created a new column in the dataframe called "*review_length*" with length of text in the review column. For my data exploration, I used *FacetGrid* (See Figures 5 and 6) from the *Seaborn* library (Waskom, 2018) to create a grid of two histograms of *review_length*, one each for positive and negative reviews for both datasets.

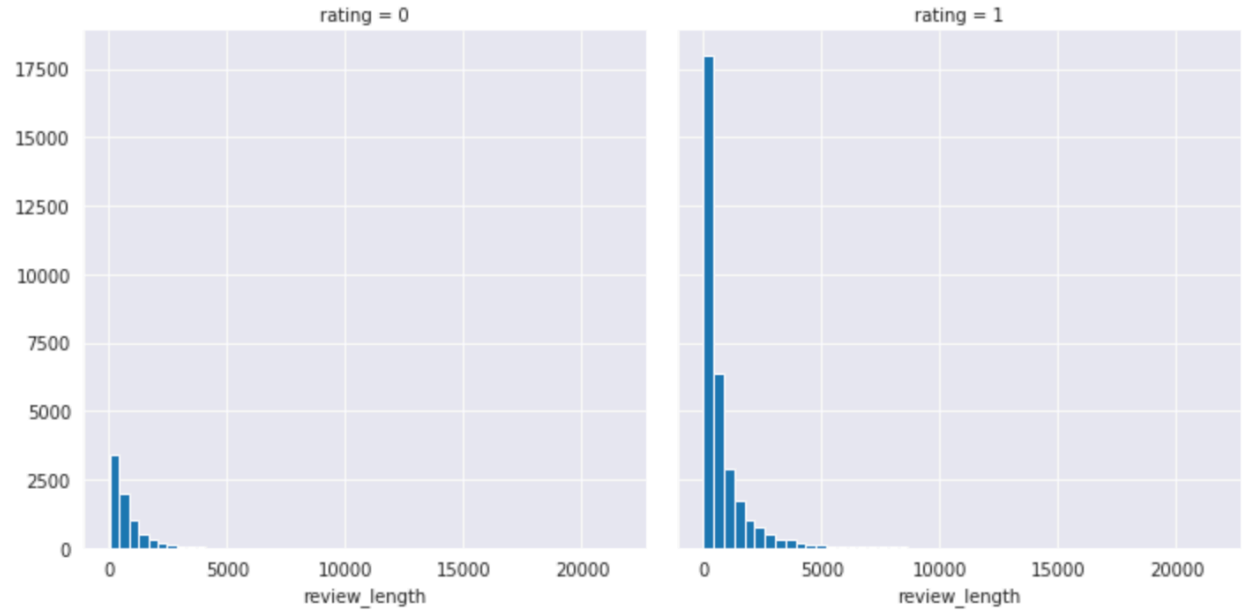


Figure 5 - Review lengths for laptop reviews

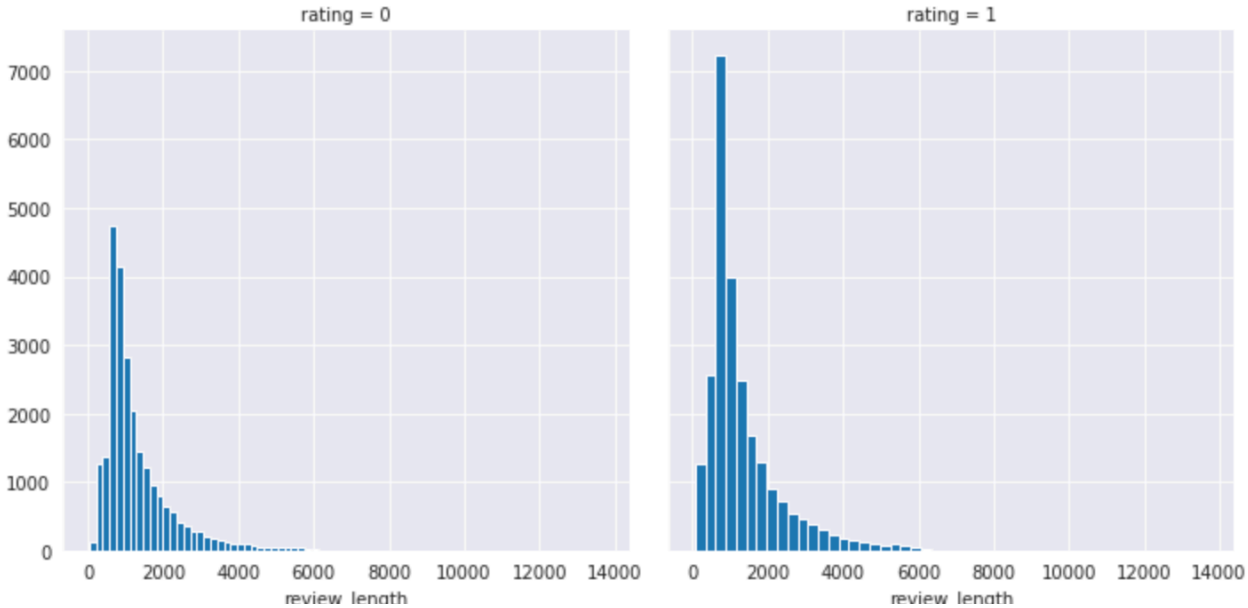


Figure 6 - Review lengths for movie reviews

Looking at the box and whisker plot for the *review_length*, I could see that it was an exponential distribution. A clipped length of 900 to 1000 words for the Amazon datasets and 1400 to 1500 for the IMDb movie review dataset could

cover the mass of the distribution. The significance of the clipped lengths will be clarified when I describe my *Keras* implementation.

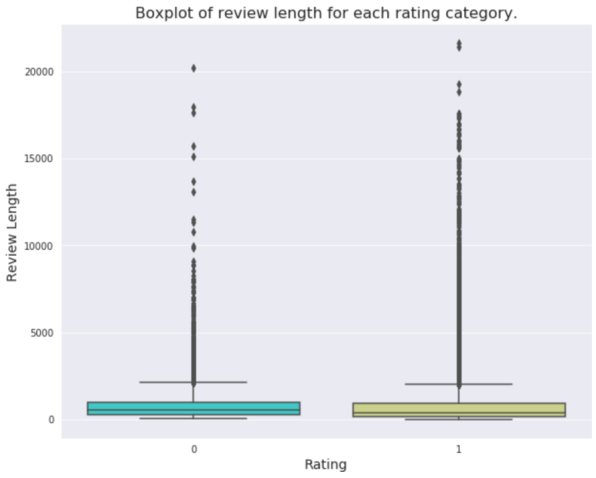


Figure 7 – Amazon Laptop Review Dataset

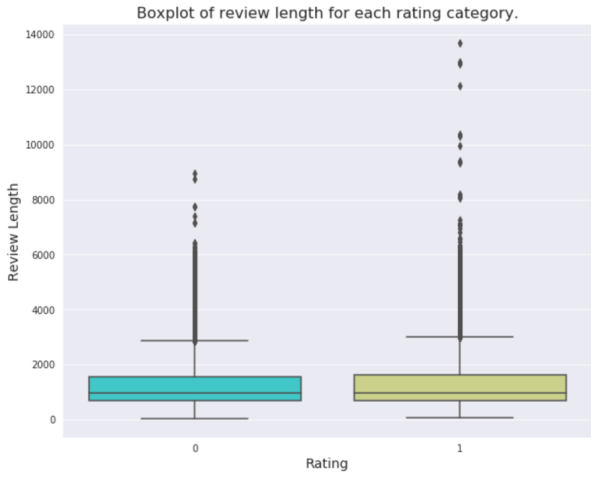


Figure 8 – IMDB Movie Review Dataset

The count plot shows the Amazon dataset has about 30K positive and 10k negative reviews. The IMDb movie dataset on the other hand has a balanced number of positive and negative data samples.

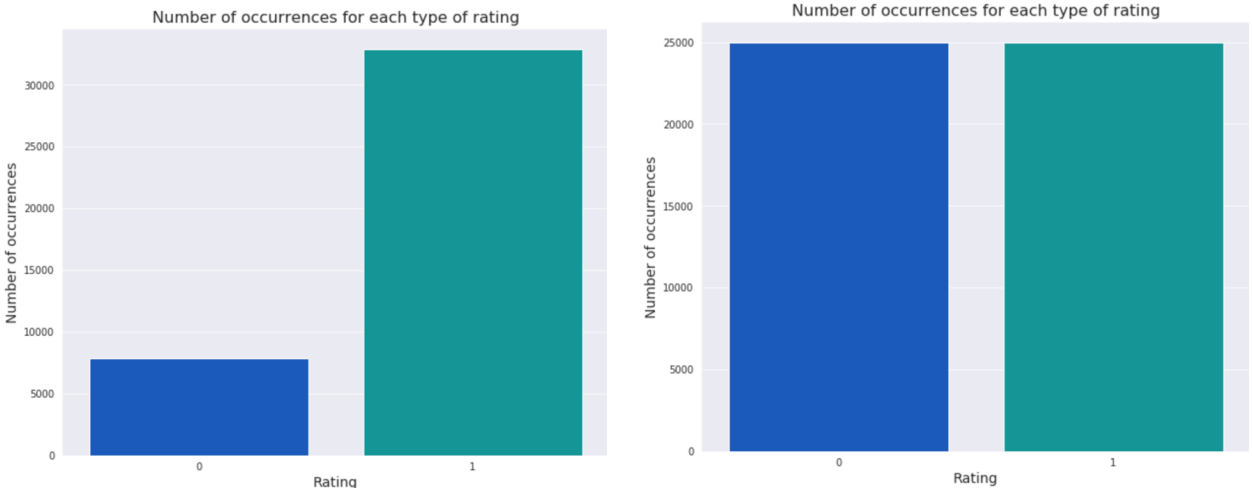


Figure 9 - Review counts for Amazon Laptop and IMDB Movie datasets

After that, I worked on pre-processing the review text. For each of the review, I took as input, I removed any HTML tags or URLs (Stackoverflow, 2013). I then converted characters in the review to lowercase and split them into word tokens. I removed any tokens that contained punctuations, empty strings or stop-words. Finally, I returned the review after concatenating the list of tokens as a "pre-processed" sentence. The pre-processing step was applied to each review in both the dataframe. I further studied the data by creating a list of top words using Natural Language Toolkit's (NLTK) *FreqDist* (Steven Bird, 2009) function. Following which I created Wordcloud (Kadam, 2019) visualizations that helped me understand commonly used words in each of the datasets. See Figure 10.

Figure 3 - Wordclouds for topics of Amazon Laptop and IMDB Movie review datasets

Following the pre-processing step, I split the data into train, test and validation sets and using the Gensim methods, created Word2Vec model with vectors of 150 dimensions. The parameters I used to create the model included a *window* size of 10. The window size represents the maximum distance between current and predicted word within a sentence. I used *min_count* value as 2 which ignores words that have a total absolute frequency lower than 2. I used 10 workers threads and *sg* as 0 to indicate that I wanted to use CBOW model. In order to avoid repeating the Word2Vec model training step unless absolutely need in the future, I saved the wor2vec model to a file. During this model training phase I used *simple_preprocess* method from Gensim to which I passed a list of lists of tokens from sentences in the dataset. This list of list allows Word2Vec to internally create a vocabulary and behind the scenes train a simple neural network with a single hidden layer.

Once trained, I tested the model by looking up top 10, 8, 5 or 3 words that were similar to my input word. I tried words like 'terrible', 'excellent', 'mac', 'issue', 'movie' etc. In order to create the word-clouds I used the *most_similar()* function with input word for the parameter "*positive*." By default, this returns the top 10 similar words. In case one wants to see top 8, 5 or 3 words it can be specified that using the "*topn*" input parameter. I also tested the model *similarity()* function that compares the similarity between two words. I tried this with two different words, with the same word for both input word 1 and input word 2, between two related words. The output values for different words came out to be low, the same word twice came to 1.0 and similar words came out to be closed to 1.0. I also tried to find the "odd one out" word and the results were good. Figure 11 shows a wordcloud visualization of positive and negative words I found in the reviews.

Figure 11 - Wordcloud visualizations for positive and negative words

I also tested the Word2Vec model by plotting the word vectors on a 2-D plot using dimensionality reduction algorithms, i.e. PCA and t-SNE. This function takes as input a word, a list of words and the word2vec model. It then outputs vector representations for the input word, eight words that are most similar to the input word and the list of words. The function creates a dataframe containing the word, its reduced 2-D vector representation and corresponding color. The dataframe is then used to create the plots shown below. In the visualizations we will look at query word (in blue), and most similar words (in green), and list of words passed in the function (in red). With an input word I plotted graphs that had eight most similar words versus ninth to sixteenth most similar words (see Figure 12) and eight most similar words versus eight random words (see Fig 13).

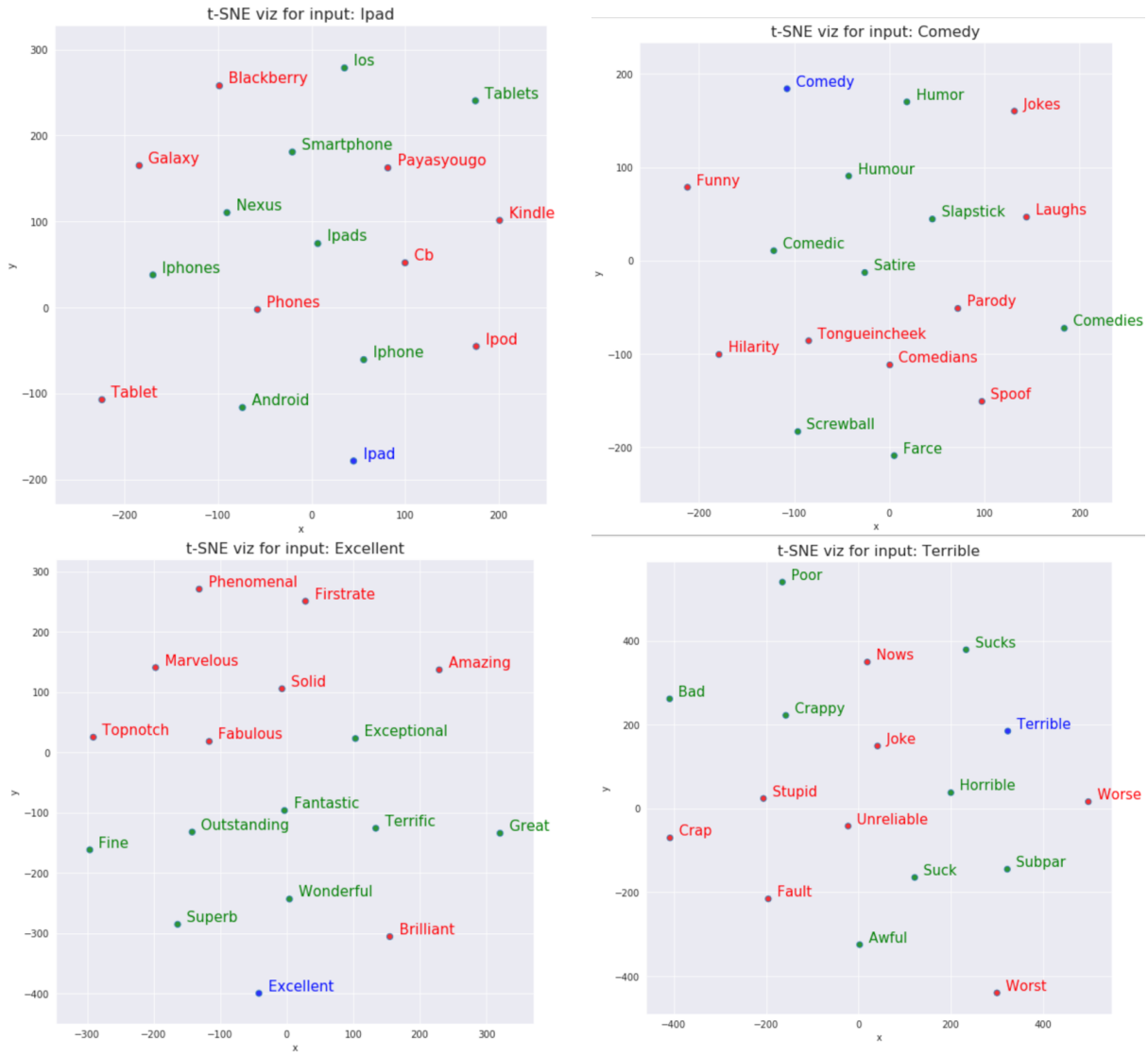


Figure 12 – Annotated plots for eight most similar words vs ninth to sixteenth most similar words

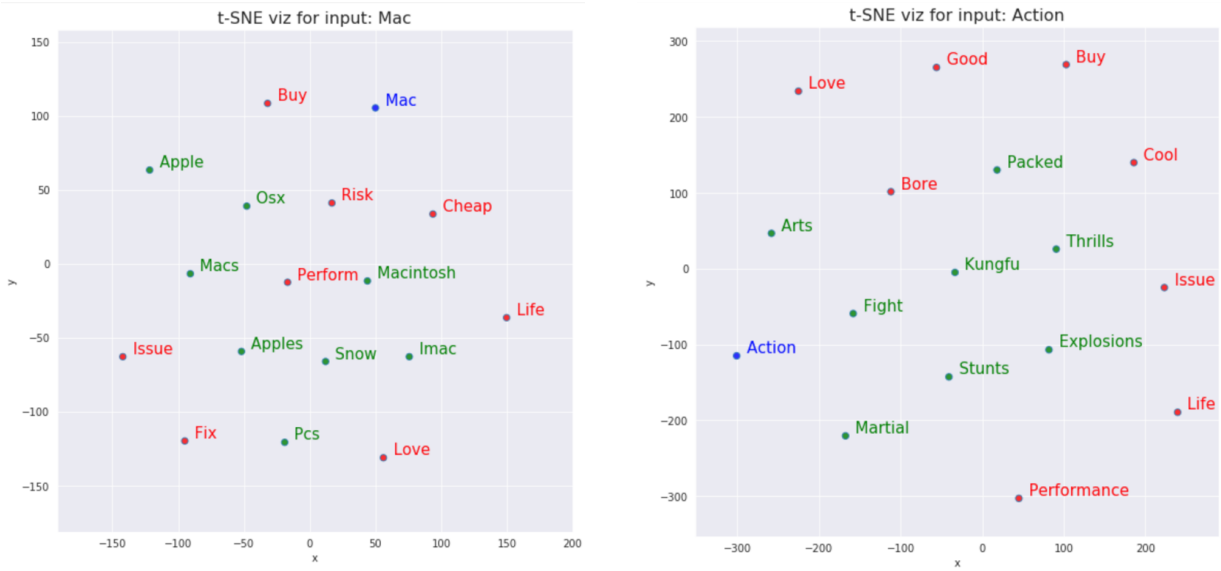


Figure 13 – Annotated plots for eight most similar words vs random words

I moved on to the main sentiment classification task after that. For this purpose, I took my training, testing and validation datasets and provided them as input to the *generate_features* function I created along-with the Word2Vec model created above. The output of the function was training, testing and validation vectors. In order to create the vectors, I simply averaged the vectors found in the model for the words in the input over the total number of words in each input set. Once the vectors were created, I started my classification task using algorithms like Logistic Regression, Support Vector Machine, XGBoost classifier. For each classifier, I generated classification report, confusion matrix and accuracy scores on the validation and test sets. See results in table 1.

In this project, I wanted to compare the classification accuracies between two different sets of classifiers, Neural Networks and traditional classifiers mentioned above. Therefore, the next step was to start with my implementation for Keras CNN. Keras embedding layer needs integer inputs and each integer maps to a token. Tokens have real-valued vector representation in the embedding. The Keras API contains a class called *Tokenizer*. This class contains methods that allows creation of a vocabulary of tokens from the reviews. Once the vocabulary is created, *texts_to_sequences()* method can be used to encode the reviews. For efficiency reasons, Keras requires that each document is of the same length. I have used the Keras function *pad_sequences()* to pad the sequences to the maximum length by adding 0 values on the end to achieve this goal. The maximum length used here is derived from clipped lengths that covers the mass of the distribution in the box-plots for the two datasets. This is the significance of the clipped review length, that we referred to in the *Dataset overview* section, we use it to determine the max length for our Keras CNN model.

Next, I worked on creating the Keras CNN Sequential model (See Figure 14). For the model I have used Conv1D layer with 128 filters, kernel size as 5, Rectified Linear Unit (ReLU) as activation function for hidden layers and a sigmoid function for the output layer binary classification. I used GlobalMaxPooling1D layer to reduce the size of incoming feature vectors. The learning process needs to be configured before training the model. For that purpose I used the *compile()* method, where I used the *adam* optimizer for good performance and *binary_crossentropy* as loss function. I also used the Keras *summary()* function, which is a useful method that lets you look into the model and parameters for training. For prediction I used *predict_classes()* method. The *evaluate()* method was used to study the accuracy and loss for train, validation and test sets.

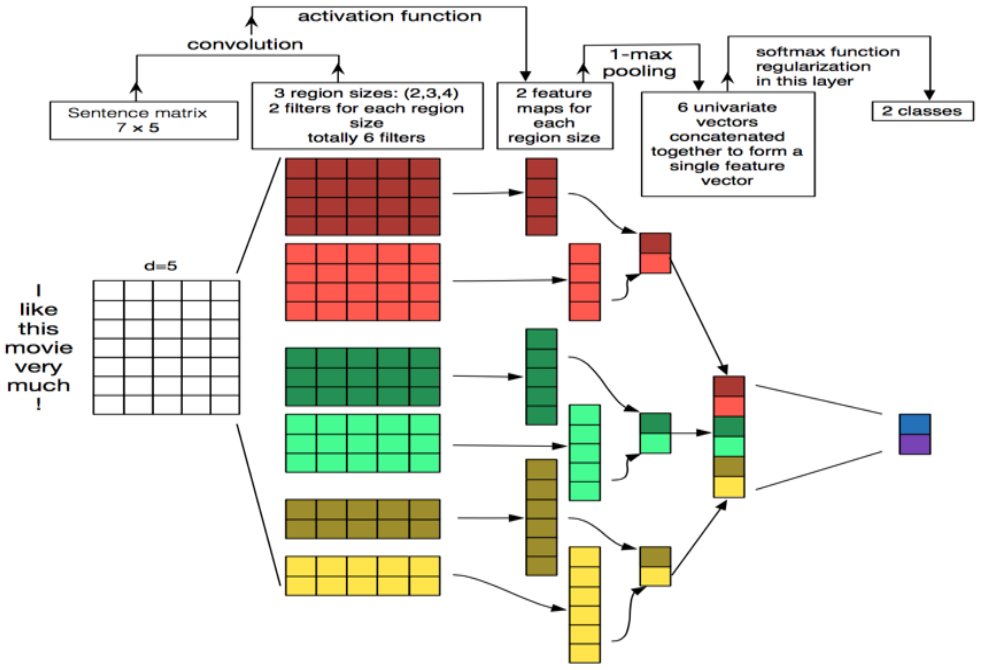


Figure 4 - Keras CNN

For Keras CNN I first used its own model training methods. However, since I already had Word2Vec model pre-trained (Chollet, 2016) I wanted to try Keras CNN on that model. The first step for that was to load the word embedding as a dictionary of words to vectors. The word embedding was saved in so-called '*word2vec*' format. The function named *load_word_embedding()* allowed me to load the embedding and return a dictionary of words mapped to the vectors in NumPy array format. Then I created a function *get embedding weight matrix()* that took the loaded word embedding and the *tokenizer.word_index* vocabulary as input and returned an embedded weight matrix with the word vectors. The prepared weight matrix *embedding weight vectors* is then passed to the new embedding layer as an input. I used the *trainable* parameter as *False* to prevent any updates to the weights already trained. I then added this layer to my model with Conv1D (with 128 filters, kernel size as 5) and MaxPooling1D. Finally, I used sigmoid activation function for the output layer. Just like earlier, I compiled the network, fit the training data and evaluated performance for train, validation and test sets.

For the next set of classifiers, I used Keras Bidirectional LSTM (See Figure 15). LSTM or Long short-term memory (LSTM) is a recurrent neural network which is powerful but computationally expensive. Similar to Keras CNN using pre-trained Word2Vec, I created embedding layer with weight matrix as *embedding_weight_vectors*. In this part, I added an extra Conv1D layer on top of the Bidirectional LSTM layer to reduce the training time. Keras API simplifies the creation of Bidirectional LSTM by letting us wrap LSTM with Bidirectional. I added Dropout Layer to prevent overfitting. Following this step, just like earlier, I compiled the network, fit the training data and evaluated performance for train, validation and test sets.

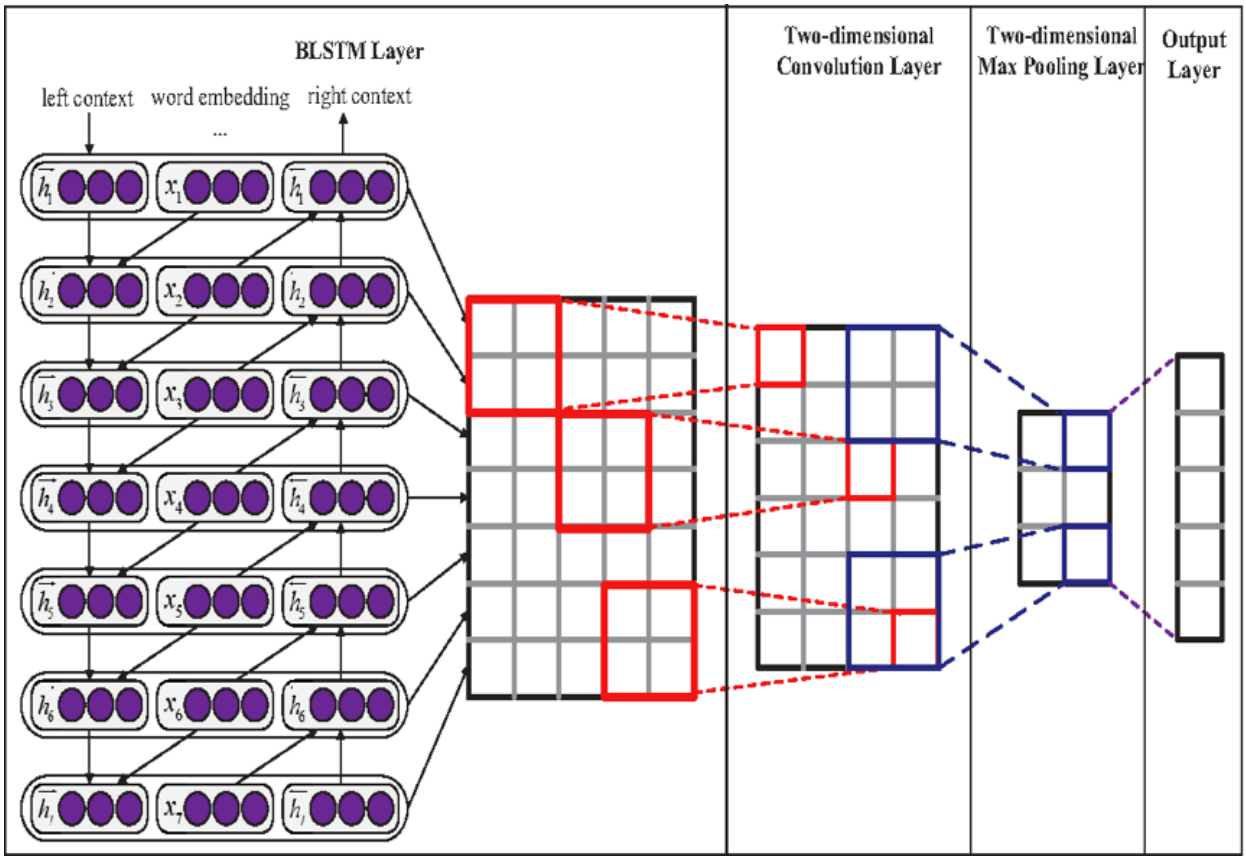


Figure 5 - Keras Bidirectional LSTM

Next, I carried out classification using pre-trained Word2Vec word embeddings and Keras Bidirectional LSTM without the Conv1D layer. For this, the process was similar to the one right above just without the Conv1D layer addition. For each of the above Keras classification technique I also generated the classification report, confusion matrix and accuracy scores on the validation and test sets. See results in table 1.

Sentiment Analysis Using Doc2vec: To use doc2vec documents need to be tagged with unique IDs. This task is performed through Gensim's *LabeledSentence* function. Doc2Vec uses two different categories of models, i.e. Distributed Bag of Words (DBOW) and Distributed Memory. Distributed memory has two variants Distributed Memory Concatenated (DMC) model and Distributed Memory Mean (DMM) model. Gensim's Doc2Vec class can be used to create these models. If input parameters *dm* is set to 0 it creates a DBOW model and *dm* as 1 creates a DM model. The *size* parameter defines the dimensionality of feature vectors. If *dm_mean* is set to 1 the model uses the mean of the context vector. If *dm concat* is set to 1 the model concatenates the context vectors. Finally, the *alpha* parameter defines the learning rate. For each of these models, I built the vocabulary and trained the model with multiple iterations, alpha-reduction and shuffling. Then I generated train, test and validation vectors using *generate vectors()* function which gives the document vectors of the model represented as *docvecs* objects and a tag for the document. Based on document vectors of DBOW, DMC and DMM model I have done sentiment analysis using LogisticRegression, SVC with linear kernel, XGBClassifier. I generated the classification report, confusion matrix and accuracy scores on the validation and test sets. See results in table 1.

Then I used document vectors from all three different models, and concatenated them to see how it affects the performance. I defined a function, *generate_concat_vectors()* to concatenate document vectors from different models. I generated train, test and validation vectors for the combined models using this function and performed sentiment analysis using LogisticRegression, SVC with linear kernel and XGBClassifier. I generated the classification report, confusion matrix and accuracy scores on the validation and test sets. See results in table 1.

Next, I performed sentiment analysis using combination of Doc2Vec DBOW and DMC document embeddings and Keras Neural Network. Followed by, the same using Doc2Vec DBOW and DMM document embeddings. For the last two approaches I created Keras model with dense layers, compile, fit the network and evaluated the performance for train, validation and test set. I generated the classification report, confusion matrix and accuracy scores on the validation and test sets. See results in table 1.

Results: Following are the train, validation and test set accuracy scores for the models I have tried.

Model		**IMDB Movie Reviews Accuracy**			**Amazon Laptop Reviews Accuracy**		
		Train	**Validation**	**Test**	**Train**	**Validation**	**Test**
Word2Vec	LogisticRegression	.8753	.8692	.8706	.9046	.9060	.9009
	SVC with linear kernel	.8755	.8714	.8690	.9050	.9089	.9001
	XGBClassifier	.8695	.8540	.8506	.9058	.9057	.8967
Keras Convolutional Neural Networks (CNN)		.9994	.8770	.8788	.9992	.9239	.9146
Using Pre-trained Word2Vec Word Embedding to Keras CNN		.9593	.8444	.8268	.9690	.8969	.8891
Using Pre-trained Word2Vec Word Embedding to Keras CNN And Bidirectional LSTM		.9356	.8854	.8902	.9567	.9212	.9144
Using Pre-trained Word2Vec Word Embedding to Keras Bidirectional LSTM		.9011	.8800	.8786	.9418	.9205	.9229
Doc2Vec DBOW	LogisticRegression	.8732	.8738	.8778	.8982	.9094	.8955
	SVC with linear kernel	.8738	.8742	.8766	.8985	.9072	.8969
	XGBClassifier	.8682	.8500	.8556	.9008	.8964	.8849
Doc2Vec DMC	LogisticRegression	.5939	.5862	.5986	.8088	.8137	.7961
	SVC with linear kernel	.5933	.5864	.5936	.8086	.8130	.7956
	XGBClassifier	.6214	.5952	.5958	.8137	.8193	.7980
Doc2Vec DMM	LogisticRegression	.8187	.8196	.8174	.8472	.8571	.8307
	SVC with linear kernel	.8193	.8184	.8190	.8428	.8554	.8280
	XGBClassifier	.8115	.7858	.7920	.8494	.8525	.8282
DBOW + DMC	LogisticRegression	.8741	.8738	.8778	.8982	.9099	.8940
	SVC with linear kernel	.8751	.8744	.8790	.8990	.9072	.8977
	XGBClassifier	.8692	.8510	.8574	.9000	.8969	.8849

DBOW + DMM	LogisticRegression	.8813	.8742	.8804	.9024	.9092	.8994
	SVC with linear kernel	.8809	.8756	.8786	.9033	.9092	.9001
	XGBClassifier	.8736	.8548	.8590	.9016	.8986	.8854
Combination of Doc2Vec DBOW And DMC Document Embedding and Keras Neural Network		.9088	.8742	.8746	.9312	.9040	.9033
Combination of Doc2Vec DBOW And DMM Document Embedding and Keras Neural Network		.9295	.8722	.8720	.9475	.9156	.8984

Table 1 – Results for various classification algorithms and models

Below are the confusion matrices from the classification algorithm runs for test data set.

Model		**IMDB Movie Reviews Confusion Matrix with Accuracy**					**Amazon Laptop Reviews Confusion Matrix with Accuracy**				
		TN	FP	FN	TP	Accuracy	TN	FP	FN	TP	Accuracy
Word2Vec	LR	2185	344	303	2168	0.87	554	279	125	3117	0.90
	SVC	2182	347	308	2163	0.87	557	276	131	3111	0.90
	XGB	2127	402	345	2126	0.85	535	298	123	3119	0.90
Keras CNN		2302	227	379	2092	0.88	591	242	106	3136	0.91
Pre-trained Word2Vec	CNN	2110	419	447	2024	0.83	571	262	190	3052	0.89
	CNN+ BLSTM	2208	321	228	2243	0.89	615	218	131	3111	0.91
	BLSTM	2176	353	254	2217	0.88	652	181	133	3109	0.92
DBOW	LR	2205	324	287	2184	0.88	530	303	123	3119	0.90
	SVC	2204	325	292	2179	0.88	533	300	120	3122	0.90
	XGB	2152	377	345	2126	0.86	467	366	103	3139	0.88
DMC	LR	1586	943	1064	1407	0.60	2	831	0	3242	0.80
	SVC	1654	875	1157	1314	0.59	0	833	0	3242	0.80
	XGB	1452	1077	944	1527	0.60	28	805	18	3224	0.80
DMM	LR	2064	465	448	2023	0.82	246	587	103	3139	0.83
	SVC	2062	467	438	2033	0.82	201	632	69	3173	0.83
	XGB	1981	548	492	1979	0.79	193	640	60	3182	0.83
DBOW + DMC	LR	2204	325	286	2185	0.88	524	309	123	3119	0.89
	SVC	2210	319	286	2185	0.88	529	304	113	3129	0.90
	XGB	2164	365	348	2123	0.86	472	361	108	3134	0.88
DBOW + DMM	LR	2206	323	275	2196	0.88	538	295	115	3127	0.90
	SVC	2209	320	287	2184	0.88	538	295	112	3130	0.90
	XGB	2166	363	342	2129	0.86	465	368	99	3143	0.89
DBOW + DMC and Keras Neural Network		2187	342	285	2186	0.87	564	269	125	3117	0.90
DBOW + DMM and Keras Neural Network		2142	387	253	2218	0.87	572	261	153	3089	0.90

Table 2 – Confusion matrices for various classification algorithms and models

Conclusion: In this project, I have presented a comparison of results for a variety of word and document embedding models. I have also compared traditional machine learning algorithms with Keras CNN and Keras Bidirectional LSTM. I tried out various combinations of models and classification algorithms. I tried all of these variants on the sentiment analysis task to classify reviews into positive and negative sentiment. I used datasets containing reviews for laptops from Amazon's website and IMDb movie reviews.

I observed that when I used Word2Vec word embedding model with LogisticRegression, SVC and XBGClassifier classifiers, LogisticRegression gave better performance than SVC, which in turn gave better performance than XGBClassifier. Keras Convolutional Neural Networks (CNN) performed better than all of the traditional classification algorithms whereas, Keras CNN with pre-trained word2vec word embeddings gave worse performance than Keras's own embedding and CNN classifier.

In case of IMDB Movie Review dataset, the best performance was given by Keras CNN combined with Bidirectional LSTM using pre-trained word2vec word embeddings. While for Amazon Laptop Review dataset, the best performance was given by Keras Bidirectional LSTM using pre-trained word2vec word embeddings.

Next, I looked at Doc2Vec model-based classification. In case of traditional classifiers using doc2vec models, DBOW model gave better performance than DMM, which in turn gave better performance than DMC. Combining DBOW with DMC or DMM gave better performance than DMC and DMM models on their own. For the combo models, DBOW + DMM gave better performance with traditional classification algorithms when compared to DBOW + DMC document embeddings. Both the combo models performed almost equally well with Keras neural network.

One reason I believe Bidirectional LSTM did well is that it is a Recurrent Neural Network (RNN). RNNs have the advantage of being able to persist information over several time steps whereas CNN recognizes patterns over space. RNN considers current inputs as well as, previously received inputs. Hence, it works really well with sequence data like text, time series, videos, DNA sequences, etc.

Bibliography

Chollet, F. (2016, July). *Using pre-trained word embeddings in a Keras model*. Retrieved from Keras.io: https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html

doc2vec, g. (2019, April). *models.doc2vec – Doc2vec paragraph embeddings*. Retrieved from models.doc2vec – Doc2vec paragraph embeddings: https://radimrehurek.com/gensim/models/doc2vec.html

Hongning Wang, Y. L. (2011). *Latent Aspect Rating Analysis without Aspect Keyword Supervision.* Retrieved from sifaka.cs.uiuc.edu: http://sifaka.cs.uiuc.edu/~wang296/Data/LARA/Amazon/AmazonReviews.zip

Kadam, S. (2019, May). *Generating Word Cloud in Python*. Retrieved from geeksforgeeks.org: https://www.geeksforgeeks.org/generating-word-cloud-python/

Maas, A. L. (2011). *Learning Word Vectors for Sentiment Analysis.* Retrieved from ai.stanford.edu: https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz

Quoc V. Le, T. M. (2014, May). Distributed Representations of Sentences and Documents. *Distributed Representations of Sentences and Documents*. arXiv.

Stackoverflow. (2013, July). *Python code to remove HTML tags from a string*. Retrieved from stackoverflow.com: https://stackoverflow.com/questions/9662346/python-code-to-remove-html-tags-from-a-string/9662410

Steven Bird, E. K. (2009). *Natural Language Processing with Python.* O'Reilly Media Inc. Retrieved from http://www.nltk.org/api/nltk.html?highlight=freqdist

Tomas Mikolov, K. C. (2013). Efficient Estimation of Word Representations in Vector Space. *Efficient Estimation of Word Representations in Vector Space*. arXiv.

Waskom, M. (2018). *FacetGrid*. Retrieved from seaborn.pydata.org: https://seaborn.pydata.org/generated/seaborn.FacetGrid.html

word2vec, g. (2019, April). *models.word2vec – Word2vec embeddings*. Retrieved from models.word2vec – Word2vec embeddings: https://radimrehurek.com/gensim/models/word2vec.html