Sentiment Classification of Amazon Reviews Using Machine Learning and Deep Learning Models.

Springboard Capstone Project 1

By: Harish Prabhala

Mentor: Ryan Rosario

Introduction:

Client: Petco Animal Supplies Inc.

In the new age of e-commerce and online shopping, there has been an upsurge of websites that enable the users to write reviews for products and services available both online and offline. The success of websites like Yelp, IMDB, TripAdvisor etc can be largely attributed to millions of user reviews available for the peruse of the public on their websites. Product sellers on the largest e-commerce platforms such as Amazon and ebay predominantly rely on the quality of the reviews of their products, in order to succeed in their business. Along with the text of the review, these sites allow the users to assign a star rating for each of the product. This star rating system helps us to classify if a review is good or bad. Given the text of the review and the star rating, a machine learning model can be trained to classify the sentiment of a review. Such a model can have a wide range of applications, especially to classify reviews, comments or any sort of text where there is no star rating system corresponding to the text. This capstone project aims to apply different machine learning and deep learning models for sentiment classification of Amazon pet supplies reviews for our client Petco.

Exploratory Data Analysis:

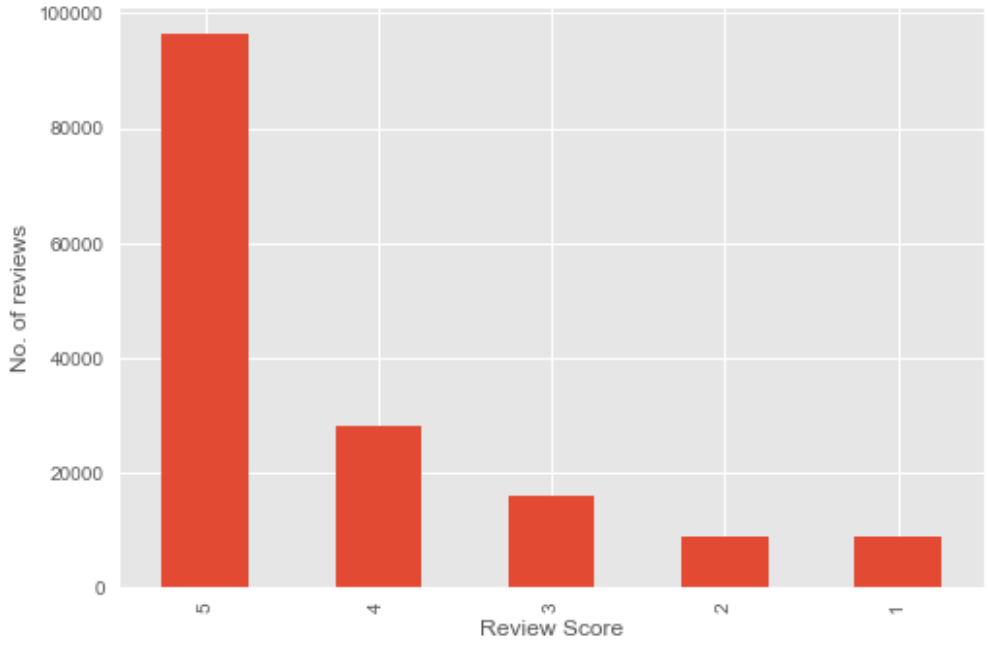
The dataset used for this project is Amazon pet products review dataset containing about 160,000 reviews. This dataset was obtained from the UCSD Amazon product database by Dr. Julian McAuley*. The Amazon Pet Product Reviews dataset is ~110 MB dataset, which consists of around 160k reviews about Amazon pet supplies and products. Each review has the following features:

- ID of the reviewer, e.g. A14CK12J7C7JRK
- ID of the product, e.g. 1223000893
- Name of the reviewer
- A helpfulness rating of the review, e.g. 2/2
- Text of the whole review
- Rating of the product
- Summary of the review
- Time of the review (unix time)
- Time of the review (raw)

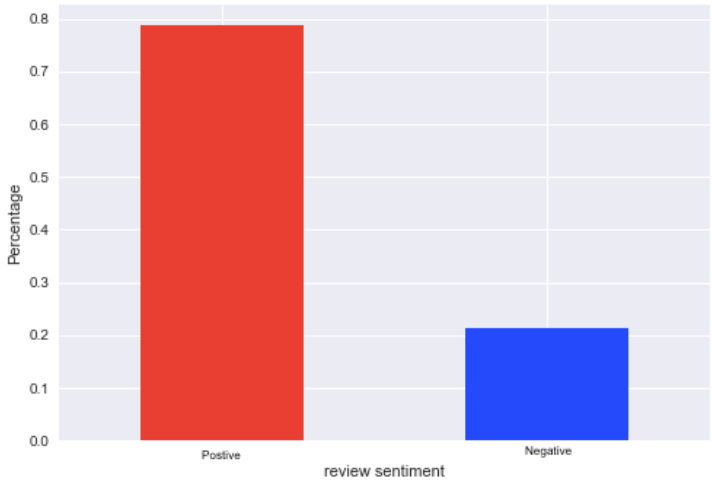
* http://jmcauley.ucsd.edu/data/amazon/

	asin	helpful	overall	reviewText	reviewTime	reviewerID	reviewerName	summary	unixReviewTime
0	1223000893	[0, 0]	3	I purchased the Trilogy with hoping my two cat...	01 12, 2011	A14CK12J7C7JRK	Consumer in NorCal	Nice Distraction for my cats for about 15 minutes	1294790400
1	1223000893	[0, 0]	5	There are usually one or more of my cats watch...	09 14, 2013	A39QHP5WLON5HV	Melodee Placial	Entertaining for my cats	1379116800
2	1223000893	[0, 0]	4	I bought the triliogy and have tested out all ...	12 19, 2012	A2CR37UY3VR7BN	Michelle Ashbery	Entertaining	1355875200
3	1223000893	[2, 2]	4	My female kitty could care less about these vi...	05 12, 2011	A2A4COGL9VW2HY	Michelle P	Happy to have them	1305158400
4	1223000893	[6, 7]	3	If I had gotten just volume two, I would have ...	03 5, 2012	A2UBQA85NIGLHA	Tim Isenhour "Timbo"	You really only need vol 2	1330905600

Of the nine columns for each review in the dataset, we will be using the "overall" and "summary" as our target and feature columns in our project. The star ratings of the reviews are on an ordinal scale of 1-5 and have the below distributions of each scores.



In this project, instead of using five categorical variables for the review classification, we encode the review scores into 2 categories 1 and 0. Scores equal to or below 3 will be considered as negative reviews and encoded as 0 and scores above 3 will be considered positive and thus encoded as 1. This dichotomous encoding has resulted in the following distribution of the positive and negative reviews.



As we can see above, the data is heavily skewed towards the positive reviews; hence to train our machine learning models, we used a stratified sampling approach. In a stratified sampling method, we divide the data into two different strata, positive and negative in this case, and then the random sampling in done from each of the strata. This helps us to minimize the bias from the skewed data.

Data Preprocessing:

As our input data for training the models will be text, we will use text-preprocessing techniques used in natural language processing in order to clean our input features. We use the NLTK dependency/library in order to perform the preprocessing. The following steps are used to clean and preprocess our data;

- Upper Case to Lower Case: convert all upper case letters to lower case letters

- Stopword removal: Stopwords are the extremely common words in the English vocabulary, which do not hold significant predictive power or value of the contents or semantics of a document. Examples include *'a', 'an','the', 'all','any'* etc. Removal of stop words helps us in reducing the size of the featureset.

- Lemmatization: is the process of grouping together the derived forms of a word so they can be analyzed as a single item. For example, *'running','ran','runner'* are the derived words of the root word 'run'. Lemmatizing helps us is reducing the redundancies of multiple words of the same meaning.

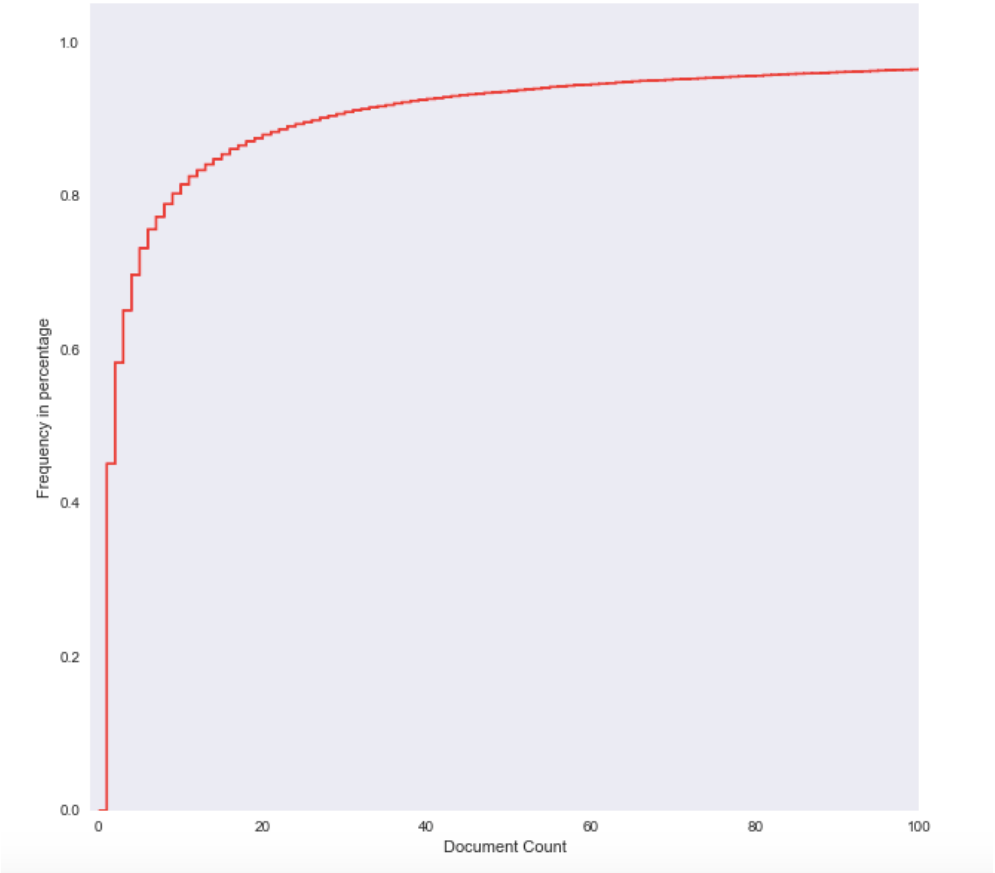
- Punctuation Removal: We remove the punctuation to reduce the number of features.

- Numeric values removal: Numerical values in the text are not essential in classifying the sentiment. Hence we remove all the numeric values in order to reduce the size of our features.

Once the preprocessing step is completed for each document in the text, we then have a clean corpus.

Feature Selection:

There are several ways to do feature selection. In this project, I use the Term Frequency-Inverse Document Frequency (TF-IDF) method to select my features. Initially, we draw a cumulative frequency graph in order to select the minimum document frequency. Examining the graph, the minimum document frequency was selected at 5 and the total size of our features has been reduced from initial ~18k words to ~5k words which have a higher predictive value.



Training the machine learning models:

Once we have our features and the labels, we can now split our entire dataset into test and train samples. In this project, I used the stratified KFold cross validation method to split my dataset into 5 folds. The next step is to train the different machine learning algorithms on our training data and predicts the accurate classification (1 or 0) on our test data.

In this project, I used four machine learning models to train my classifiers. I have tuned my hyperparameters in order to select the optimal values for the highest performance. Below are the four classifiers and their respective hyperparameters:

1. **Multinomial Naïve Bayes**

   Hyperparameters: Alpha is one of the regularization hyperparameters for multinomial naïve Bayes model. We have chosen α as 0.01 through a grid search over a set of parameter values, using cross validation to evaluate the performance of the model on the test and train data at each parameter value.

2. **Logistic Regression**

   Hyperparameters: Similar to the alpha in multinomial naïve bayes, C is a regularization hyperparameter for logistic regression. We used grid search over different values of C to select the best optimal parameter C value as 10.

3. **Decision Tree Classifier**

   Hyperparameters: The maximum depth (max_depth) of the tree in a decision tree classifier is the number of nodes that the tree can contain. We have chosen the maximum depth as 200 by incrementally trying different node sizes and finding the optimal node size that gives us the highest performance.

4. **Random Forest Classifier**

   Hyperparameters: The number of estimators (n_estimators) of a random forest classifier is the number of trees in the forest. Similar to the decision tree classifier, we chose the estimators size to be 50 by incrementally trying different tree sizes and finding the optimal size

size that gives us the highest performance.

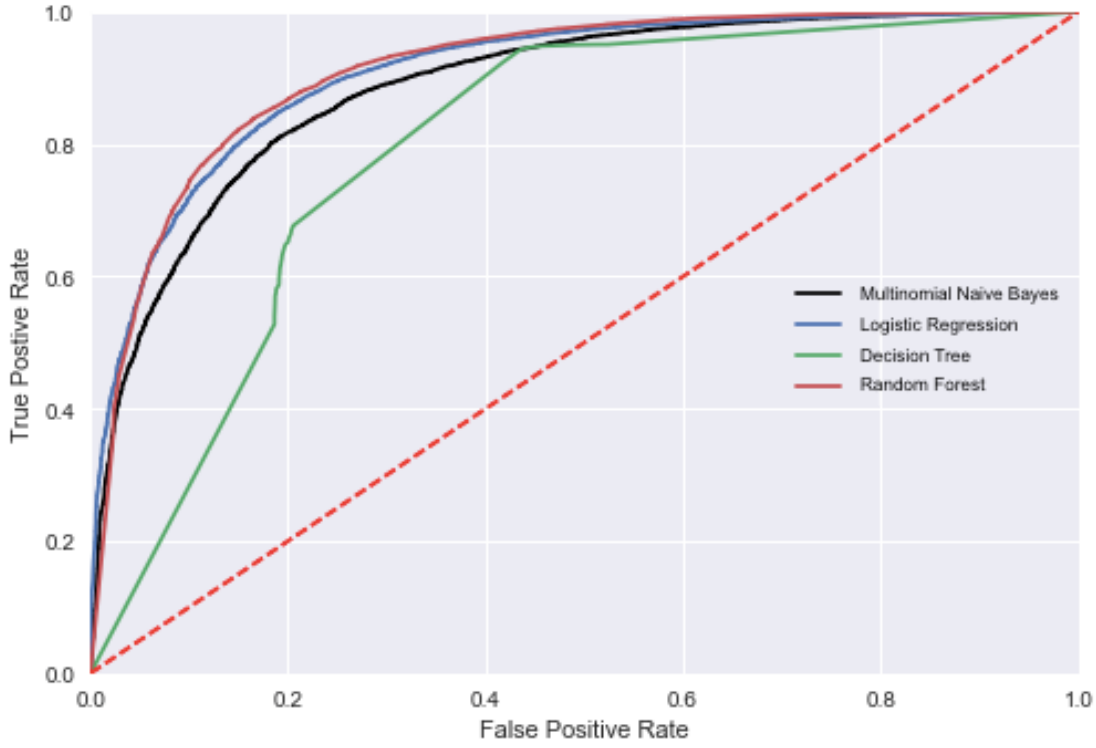
Results:

Below are the results of the accuracies of the different models. The output of each of the classifier gives us test and train accuracies, precision, recall and F1 scores. The F1 score is the harmonic mean of the precision and recall. The choice of the accuracy metric is subjective and depends on the nature of the project and desired output. In this project, we mainly look at the F1 scores and the area under curve (AUC) as our accuracy metrics. Higher the F1 the better performing is the model. For AUC, ideally a perfect classifier should have a value of 1. Our models have shown AUCs between 0.7 and 0.8 and can be considered as fairly good classifiers.

Classifier	Precision	Recall	F1
Multinomial Naïve Bayes	0.90	0.86	0.87
Logistic Regression	0.89	0.88	0.88
Decision Tree Classifier	0.88	0.86	0.87
Random Forest Classifier	0.90	0.88	0.89

Classifier	Accuracy
Multinomial Naïve Bayes	Test - 86%
	Train - 86.7%
Logistic Regression	Test - 87.9%
	Train - 89.2%
Decision Tree Classifier	Test - 86.2%
	Train - 94.2%
Random Forest Classifier	Test - 88.5%
	Train - 98%

Classifier	Area Under Curve (AUC)
Multinomial Naïve Bayes	0.72
Logistic Regression	0.78
Decision Tree Classifier	0.75
Random Forest Classifier	0.79



Among the four machine learning models, logistic regression and random forest classifier have the highest F1 scores and AUCs than multinomial naïve bayes and decision trees.

Artificial Neural Networks:

Artificial Neural Networks (ANN) is information processing systems that are inspired by the way biological nervous systems work. A typical ANN consists of thousands of interconnected artificial neurons, stacked sequentially in rows that are known as layers, forming millions of connections. The ANNs are fed large amounts of data to be trained on and is configured for a specific application, such as pattern recognition or data classification, through an iterative learning process.

Deep learning models:

In this project, I have employed two deep learning models to conduct the sentiment analysis. The two models I used were:

a. Multilayer Perceptron (MLP)
b. Long Short Term Memory (LSTM) Recurrent Neural Network (RNN)

Data Preprocessing for Neural Networks:

The preprocessing steps used to clean the data for neural networks are mostly similar to what has been done previously but with one change. Here, instead of vectorizing the sentences, we are directly feeding the sentences into our neural network. In order for the input to be of the same dimension, we padded the sentences to the length of the longest sentence. This allows us to input a constant dimension matrix into the network.

We then randomize the features and labels and then split our data into test and train sets. As our data is ready, our next step is to build our neural network. We use Google TensorFlow library to build our network. Below is a pictorial representation of a multilayer perceptron neural network.

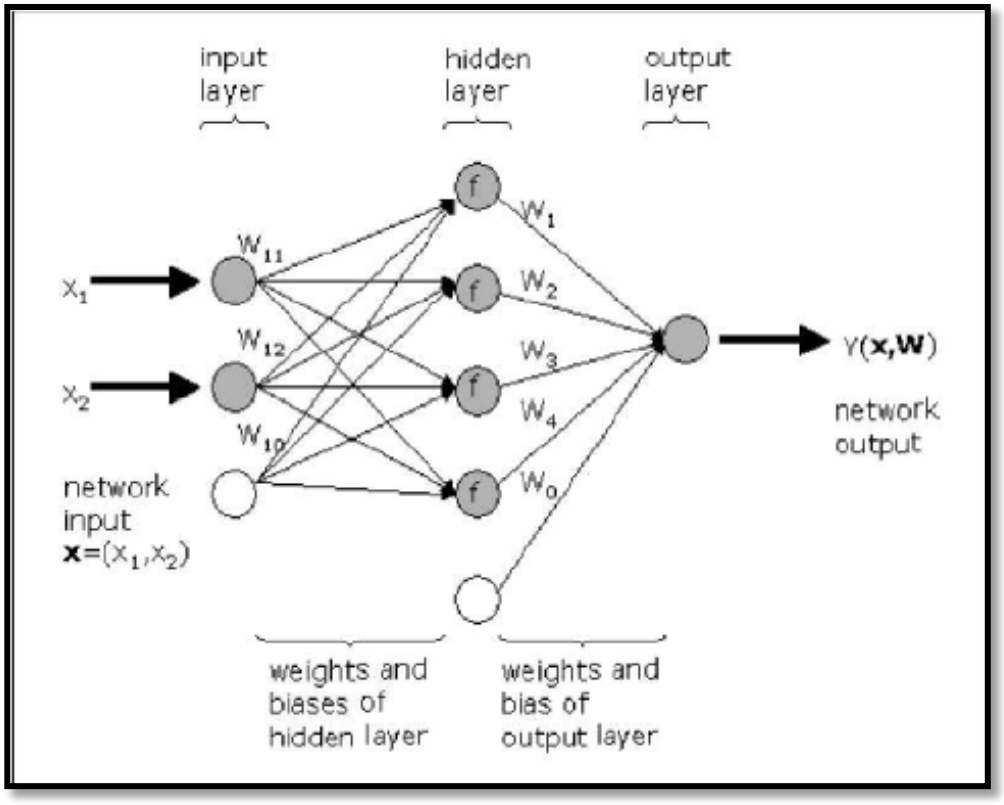


Figure a: Source- *Optimization of an Adaptive Restraint System Using LS-OPT and Visual Exploration of the Design Space Using D-SPEX.*

A. Multi Layer Perceptron

A simple neural network has one hidden layer and a deep neural network has multiple hidden layers. In this project, we used a deep neural network with four hidden layers. The neural network is built as follows:

- **Selecting the number of hidden layers –** Here we can choose the number of hidden layers in our network.
- **Node count –** We can choose the number of nodes containing in each hidden layer. Our network has 1000 nodes in each of the four layers
- **Classes -** The number of classes in the output. In our case, its 2 (positive and negative sentiment).
- **Batch size –** Batch size is the number of samples we want to pass through the network as batches for each training cycle.
- **Number of Epochs –** An Epoch is one cycle of feed forward and back propagation in a neural network. In this project, we used 10 epochs.
- **Initiate the hidden and output layers –** We initiate the hidden and output layers by randomly assigning weights and biases to each of the layer.
- **Network Activation –** We activate the hidden layers using one of the several activation functions. In this project, we used the ReLu (Rectified Linear Unit) activation.
- **Training the network** – Two important steps in training the neural network are defining the cost function and optimizer. Our cost function is a softmax cross entropy function. The optimizer used to reduce this cost is an AdamOptimizer.
- **Initializing the session** – The steps until now have created a graph of the functions in the tensorflow. Using an initializer function, we initialize the neural network.
- **Feeding the network in batches** – We then feed the network our training data according the chosen batch size.
- **Run the session** – Next, we run the session with all the given inputs and we calculate the mean of the correct predictions on our test data in the end.

```
train_neural_network(x)
```

```
WARNING:tensorflow:From <ipython-input-56-a9d38ba7d3df>:7:
bles) is deprecated and will be removed after 2017-03-02.
Instructions for updating:
Use `tf.global_variables_initializer` instead.
Epoch 1 completed out of 10 loss: 43986638935.5
Epoch 2 completed out of 10 loss: 22530532362.2
Epoch 3 completed out of 10 loss: 14898702297.9
Epoch 4 completed out of 10 loss: 11097606738.5
Epoch 5 completed out of 10 loss: 7782826075.75
Epoch 6 completed out of 10 loss: 5833431690.56
Epoch 7 completed out of 10 loss: 4564821568.38
Epoch 8 completed out of 10 loss: 3408047976.12
Epoch 9 completed out of 10 loss: 2487923995.53
Epoch 10 completed out of 10 loss: 2005104086.91
Accuracy: 0.760377
```

Results: The above results show the cost for each of the 10 epochs. As the number of epochs increase the cost decreases. The final accuracy of our neural network is 76%. This is lower than the machine learning models that we have modeled earlier. But, we have to notice that no feature engineering or selection has been done on our input data for the deep learning model. Also, the accuracy of the model increases has we increase the number of hidden layers and nodes in each layer, which is computationally expensive. Also, feature engineering and parameter tuning can be additionally done to improve the model in the future.

B. LSTM Recurrent Neural Network

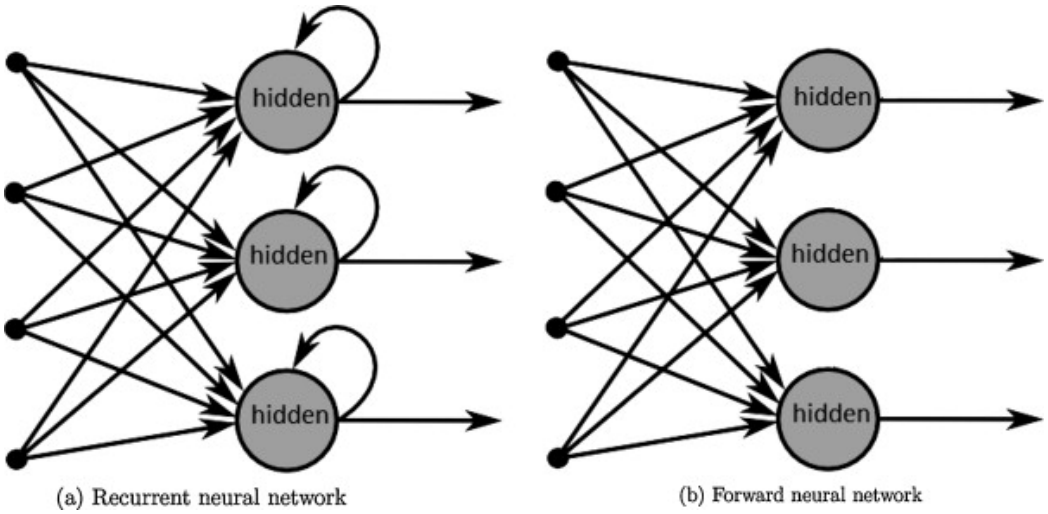


Figure B: Source: *A Survey on the Application of Recurrent Neural Networks to Statistical Language Modeling*.

The second deep learning model we used is the Long Short Term Memory (LSTM) Recurrent Neural Network (RNN). Unlike traditional neural networks, RNNs make use of sequential information. Because of this property, they have gained popularity for performing NLP tasks. In a traditional neural network we assume that all inputs (and outputs) are independent of each other. For tasks that have value in the spatiotemporal data, normal neural networks are a not particularly efficient. RNNs are called *recurrent* because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far.

To model our RNN, we use the Keras library. To build the model, we will map each review into a real vector domain, a technique known as word embedding. Keras provides a function to convert the integer representations of words into a word embedding by an embedding layer. We already have our vocabulary and the sequence length values that can be used to develop an LSTM model to classify the sentiment of the reviews. We use the Keras sequential model to compile our RNN.

- The first layer is the embedded layer, where we use 32 length vectors to represent each word, the vocabulary size and the sequence length to create the layer.
- The next layer is the LSTM layer with 100 hidden neurons.
- We use a Dense output layer as the final layer with a single neuron

and a sigmoid activation function to make 0 or 1 predictions for the two classes (positive and negative) in the problem.

- We finally define our cost and optimizer functions in our compiler.

Once we have compiled our model, we fit the model on the training set and validate on our test set. We ran 3 epochs for this model and test the accuracies after each epoch.

```
Train on 105750 samples, validate on 52086 samples
Epoch 1/3
105750/105750 [==============================] - 253s - loss: 0.3381 - acc: 0.8632 - val_loss: 0.2916 - val_acc: 0.88
25
Epoch 2/3
105750/105750 [==============================] - 248s - loss: 0.2738 - acc: 0.8912 - val_loss: 0.2906 - val_acc: 0.88
25
Epoch 3/3
105750/105750 [==============================] - 251s - loss: 0.2486 - acc: 0.9010 - val_loss: 0.3038 - val_acc: 0.87
43
```

Results: The final accuracy of the LSTM RNN model is 87.43%. Due to computationally expensive nature of this neural network and time constraints, further explorations of the model with parameter and layer tunings were not done. Hence, we believe that on a much faster processor we would be able to explore and fine-tune the model further to improve the accuracy.

Conclusion: In conclusion, among the various machine learning models, we have found that simple logistic regression and random forest performed the fastest and best as compared to the other models on this dataset. Among the deep learning models we have tested, the RNN LSTM model we believe is better suited for text classification since it is good at dealing with sequential information. So, barring any computational and time constraints, further exploration can be done on the LSTM RNN model to improve the accuracy of classification.

Recommendation: Our recommendation to Petco is to use our trained logistic regression machine learning model and the RNN LSTM deep learning model to conduct the sentiment classification of their pet supply reviews or comments.