# Capstone Project

Machine Learning Nanodegree

Pratik Joshi
21/10/2017

## Definition

### Project Overview

Natural Language Processing is a crucial domain in machine learning. It involves the extensive analysis of text bodies in order to understand the way language is used to express and communicate. It can be used to detect underlying sentiment and meaning from a large mass of text, which would be extremely difficult to do manually. It can be used to extract the style of writing, and has large applications in research as well as industry. Industrial applications involves the analysis of customer reviews of products, sentiment analysis of movie reviews, as well as document summarization. Research in natural language processing is used largely in the field of linguistics, behavioral study, and neuroscience(particularly in the use of language to communicate).

In this project, I have created a model which classifies movie reviews from Rotten Tomatoes according to their sentiment. I obtained the dataset from a Kaggle Competition(Link:https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews/data). I have used a combination of a Word2Vec model with a Keras neural network.

This project was particularly interesting for me because I wanted to explore the capabilities of the Word2Vec model and its effectiveness when combined with another classifier(like a neural net).

### Problem Statement

In this project, I will be attempting to classify a series of movie reviews by their sentiment. This project is also one undertaken by companies like Netflix and Amazon Prime in order to assess the public mood/rating of movies and TV shows in order to decide what to stream on their hosting service.
I will be using natural language processing (NLP) techniques to effectively classify the sentiment of reviews.

This problem has 2 steps:

1. **Language analysis**: The process of understanding and analysing which words infer positive,negative, and neutral sentiment, and extracting the final sentiment of the combination of the words.
2. **Classification**: The process of using the combined inferred sentiment of the words in the phrase to classify the phrase/sentence into its respective sentiment.

## Metrics

The main metric I will be using for this project will be the **categorical accuracy**. It is the mean accuracy across all classes in the problem. The formula for this is:

$$Categorical\ Accuracy\ =\ \sum_{i=1}^{n}\ (accuracy_i)\ /\ n$$

where n is the total number of classes, and $accuracy_i$ is the accuracy in class i.
The main reason I have used categorical accuracy is because simple accuracy is not enough to determine how well a model performs. The model needs to be accurate across all classes. Categorical accuracy is a more solid metric particularly in multi-class classification compared to other metrics. Scores like the F1 score are also a harmonic average over the net precision and net recall. Categorical accuracy calculates accuracy across each class and averages it.

# Analysis

## Data Exploration

The dataset I have used is from a Kaggle competition. It has 4 attributes:

- PhraseId: The unique ID of the phrase from the review.
- SentenceId: The unique ID of the review.
- Phrase: The phrase from a certain review.
- Sentiment: The sentiment associated with the phrase. The sentiment is as following:
    - 0-Negative
    - 1-Somewhat Negative
    - 2-Neutral
    - 3-Somewhat Positive
    - 4-Positive

Here is an example:

| | PhraseId | SentenceId | Phrase | Sentiment |
|---|---|---|---|---|
| **99513** | 99514 | 5218 | a dysfunctionally privileged lifestyle | 2 |
| **100672** | 100673 | 5286 | cries | 2 |
| **84878** | 84879 | 4389 | someone 's | 2 |
| **73566** | 73567 | 3761 | Heavy with flabby rolls | 2 |
| **108927** | 108928 | 5766 | be swept up in Invincible and overlook its dra... | 3 |

The dataset has 156060 entries, with 8544 reviews in total, and therein lies a problem. The dataset has been structured very strangely in terms of the splitting of phrases within a review. As you can observe from the example itself, a sentence has been split in all possible combinations, and a phrase itself may not carry much attached sentiment. For example, the 4th entry has the phrase ‘A’, which does not carry sentiment at all since it is not surrounded by context. This strange split makes the dataset difficult to understand and work with.

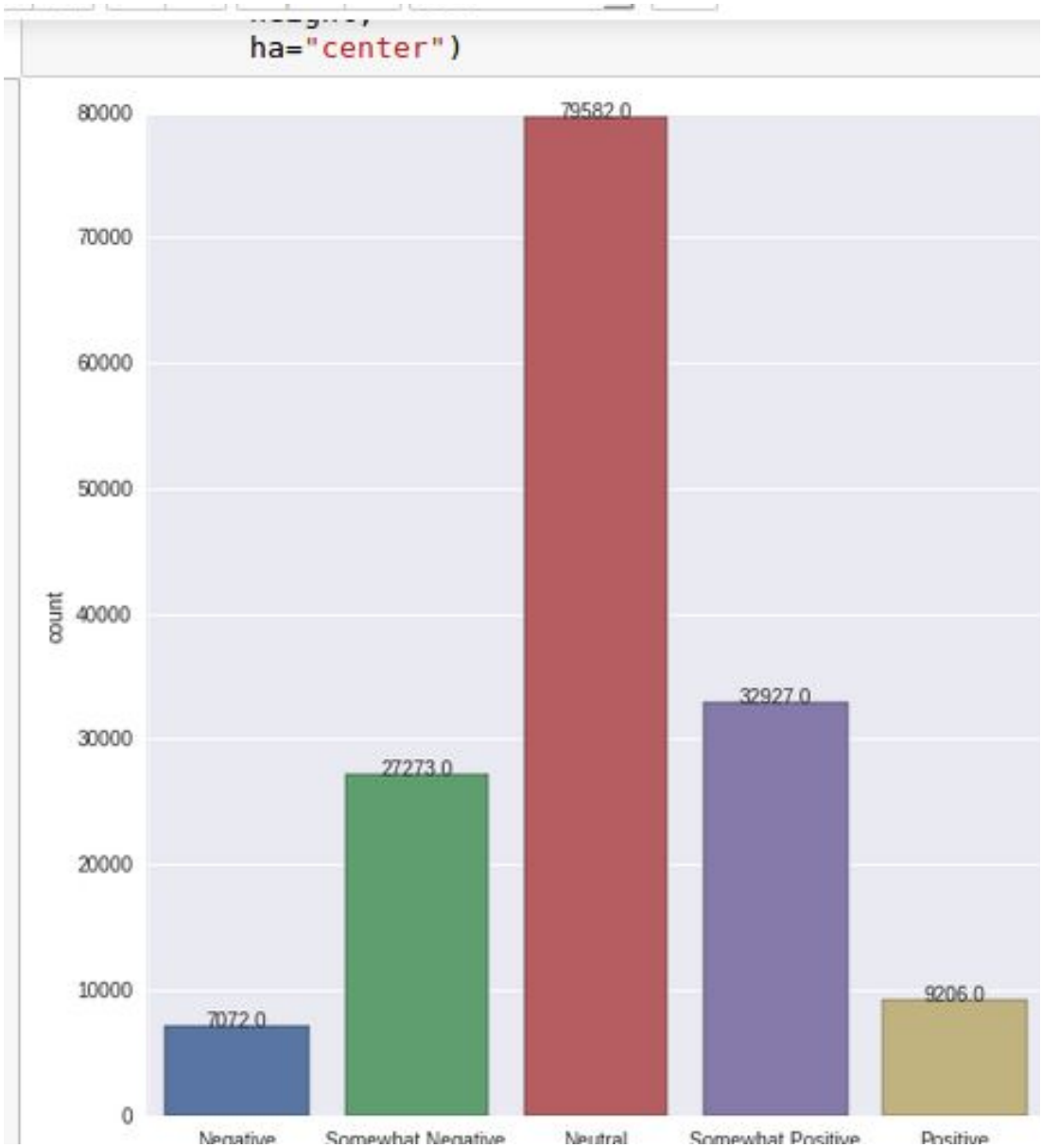
The data has 15,179 unique words/symbols, and the most frequently used words in the reviews are, barring common words like ‘the’,’a’ (called stop words), are :

- ‘Film’ - 7688 times
- ‘Movie’ - 7250 times
- ‘Character’ - 2850 times
- ‘Story’ - 2831 times

Interestingly, the word ‘comedy’ is another very frequently used word(1939 times). This could suggest that a many reviewed movies are comedy movies. However, it depends on the context in which it was used. For example, it could be used sarcastically or in a different way(e.g. ‘...a comedy of errors…’).

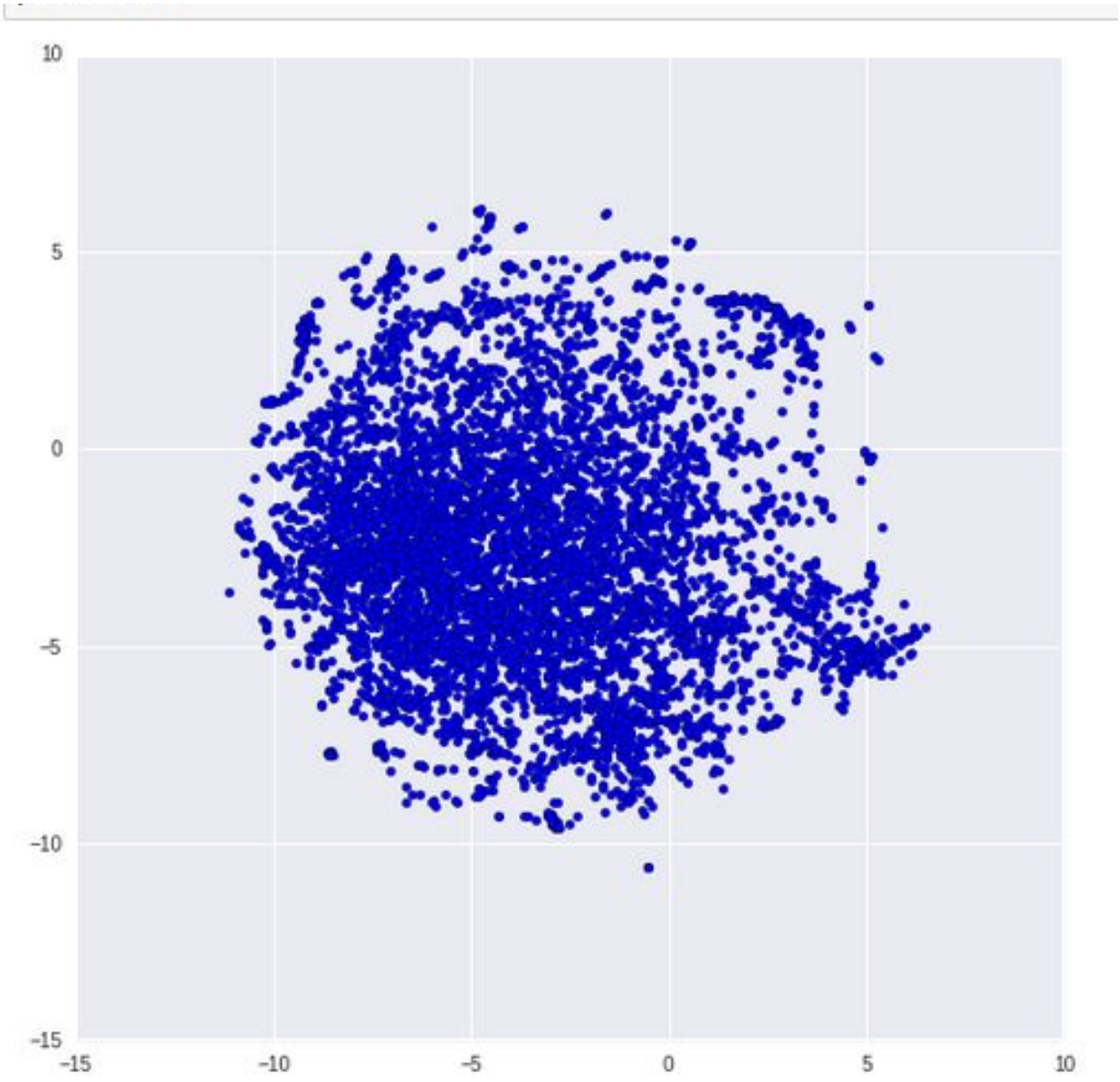
## Exploratory Visualization

1. This visualization shows the count of sentiments across all reviews. The primary sentiment, as can be observed is neutral (79582 entries), with second most being somewhat positive(32927 entries). We can see that the dataset as a whole is on a general more positive than negative.




2. The second visualization is a t-SNE graph, or a t-distributed stochastic neighbor embeddings graph. This graph helps us visualize the relationship of words in a 2D space, based on the embedding model. To obtain this graph, we feed a body

of text into an embedding model like Word2Vec, which creates an n-dimensional vector(called embedding) for each word in the text, based on the context and use of the word. What the t-SNE graph does is that it maps the n-dimensional vectors of all the words into two dimensions. An interesting property of a t-SNE graph is that the distance(Euclidean) between two points holds a great deal of meaning. Embeddings whose associated words have similar meaning/sentiment are clustered closer together in the t-SNE graph(a.k.a their distance is smaller), and those whose associated words have different meanings/sentiments are farther apart.



## Algorithms and Techniques

**Language Model Creation**

To create a language model, I used Mikolov's Word2Vec model. I used the Gensim library implementation of Word2Vec. The Word2Vec model converts words in a corpus(text body) into n-dimensional vectors, where n is the degrees of freedom(level of complexity) of the Word2Vec model.

A few parameters which can be tuned to optimize the language model:

1. Degrees of freedom(n): The higher this value, the more complex the language model will be.
2. Minimum word count(min_count): The amount of times a word has to occur in the corpus to be taken into account for language modeling.

After this, to generate embeddings for each phrase, I used an weighted-averaging method, with the weights being the TF-IDF(Term Frequency-Inverse Document Frequency). The intuition behind the TF-IDF term is:

- The more a word appears in a phrase(Term Frequency), the more important it is to its' associated sentiment.
- The more a word appears across all phrases(Document Frequency), the less important it is to a particular phrase, due to its common use.

The embeddings are multiplied to a TF-IDF term, and then an average is taken of all the embeddings whose associated words occur in the phrase. This churns out a phrase vector. This method is much better than a simple sum or a simple average since it reduces the importance of common generic terms like 'film' which do not affect the sentiment.

**Multi-Class Classification**

For this, I created a fully-connected neural network using Keras. A neural network is a interconnected group of nodes which are used for various classifying and machine learning processes. Each node carries a certain weight, and carries out a certain computational function. The data is passed through the input layer, where number of nodes are equal to the dimensionality of the input. This input is passed through the nodes through connections. For training, the inputs are passed through nodes, with randomly initialized weights, and after a predicted output is generated, the loss is calculated with respect to the actual labels, and this loss is used to adjust the weights of the neural network based on the amount of contribution it had towards the loss. A fully-connected neural network is one in which all nodes in a layer are connected to all layers in the next layer. I used a neural network over other classifiers because it has

proved very effective in multi-class classification for a variety of problems. On top of that, a neural network is very compatible with the embeddings generated by Word2Vec. Some parameters which can be tuned to optimize the neural network:

- Number of epochs: The number of times the whole data is used to train.
- Batch Size: Number of images used in every training step.
- Loss function: The function used to the deviation of predictions from original values.
- Optimizer: The function used to optimize the weights of the nodes.
- Learning rate: The speed of learning

The neural network has the following features:

- Number of layers
- Types of layers:
    - Dense
    - Dropout
- Parameters associated with each layer(Number of nodes, dropout factor)

## Benchmark

The benchmark I will be using will be the winner of the Kaggle Competition using this dataset. The winner had an accuracy of 76%.

# Methodology

## Data Preprocessing

I carried out the following pre-processing steps:

**Language Model Creation**

1. Carried out  the tokenization of each phrase(Tokenization is the splitting of a sentence into words) from the dataset, which was a tsv file. I used NLTK's(Natural Language ToolKit) word-tokenize capability for this.
2. Removed punctuation from the tokens, as they didn't carry any semantic meaning.
3. Lemmatized the tokens. Lemmatization is the process of simplifying a token(word) into its simplest form(e.g. 'Tyres' is lemmatized to 'tyre', 'deadliest' is lemmatized to 'deadly'). For this, I used NLTK's WordNetLemmatizer. This would ensure that the vocabulary generated by Word2Vec isn't unnecessarily large('tyres' and 'tyre' would be generated into 2 separate embeddings if not

lemmatized, even though they essentially mean the same thing and contribute equally to a sentiment).
4. Using Gensim's LabeledSentence capability, the tokens were grouped (by phrase) and tagged according to whether they belong to a training or testing set, and their respective phrase id. This makes it easier to account for the data.

**Multi-Class Classification**

1. The data was split into training and testing set using sklearn's train_test_split.
2. Using sklearn's TFIDFVectorizer, TF-IDF weights were generated for each word vector, and then a weighted average was taken for each phrase vector.
3. The phrase vectors were scaled to make the data homogeneous and easy to use for the neural network. This was done using sklearn's scale function
4. Finally, the sentiment labels for each entry were one-hot encoded for use in the Keras neural network.

## Implementation

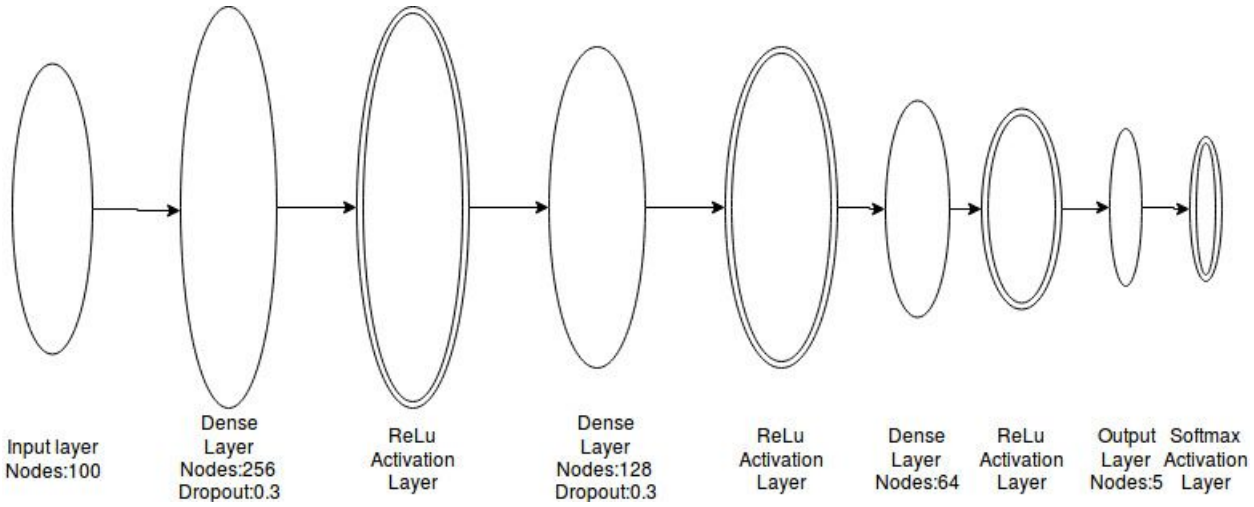
The following steps ensued as part of the implementation:

**Language Model Creation**

1. Preprocessed the data as mentioned in previous section.
2. Instantiated a Word2Vec model, with degrees of freedom(n) = 100, and min_count = 15(Words only taken into account if they occur more than 15 times in the corpus)
3. Built vocabulary of the data using Word2Vec, and trained the model with the preprocessed data.

**Multi-class Classification**

1. Created a fully-connected Keras neural network with the following design:




    a. Dropout was applied on the first two dense layers to prevent overfitting of the model. The dropout factor defines the probability that a node will not participate in the training in that particular epoch.
    b. Activation layers used were ReLu(Linear Activation function: $y = \{x, x > 0\}\ else\ \{0\}$ ), and Softmax at the output.
2. Compiled the model with the following hyperparameters:
    a. Set optimizer as rmsprop, which has learning rate 0.001
    b. Set loss function as categorical cross-entropy.
    c. Set the evaluation metric as categorical accuracy.
3. Created a Model Checkpointer which saves the best weights for generating best validation accuracy.
4. Fit the model with the preprocessed phrase vectors using the following hyperparameters:
    a. Set number of epochs to 20.
    b. Set batch size to 32

A major complication I faced during coding was knowing exactly how to preprocess the data in order to make the data usable for a neural network. It took me some time to find out optimal ways to pre-process the raw data. Here is my final model summary:

## Refinement

My initial solution involved the removal of stop-words during preprocessing in order to make classification easier for the neural net. However, I found that this step yielded very bad results for the word2vec model, since it lost a lot of context during training. The tradeoff was too high, and this resulted in an overall fall in accuracy, therefore I decided to remove this pre-processing step altogether.
My initial solution involved the use of stemming, which involves the derivation of base words from the tokens. Stemming can be similar to lemmatizing, except stemming can lead to a derivation of a non-existent word(e.g. Stemming of deadly yields deadli). Hence, I changed to lemmatization.
The primary refinement process was the repeated tuning of the size(degrees of freedom) of the word2vec model and the size of the neural net. The higher the degrees of freedom, the deeper the neural network had to be(more nodes, more layers) to generate better accuracy. This tuning was dependent on the dataset size, which was misleading because although the dataset size was large, it was because of the split and combination of only 8544 distinct reviews. Finally, after numerous tests, I settled on n=100 and the respective neural network.
Other factors that I tuned were the number of epochs and the dropout factor for the dropout layers.
My initial untuned model had a categorical accuracy of 52.5%.
A few parameter values I tried and their respective generated accuracies were as follows(Number of epochs=20):

- n=50,Neural net dimensions: 50*1024*512*256*64*5 Accuracy: 53.2%
- n=50,Neural net dimensions: 50*128*64*32*5 Accuracy:54.7%
- n=50,Neural net dimensions: 50*64*32*5 Accuracy:54.0%
- n=100,Neural net dimensions: 100*128*64 Accuracy:55.0%
- n=100,Neural net dimensions: 100*256*128*64 Accuracy:55.9%
- I had also conducted a few trials with n=200, but they all yielded unsatisfactory accuracies.
- Final accuracy results: n=100,Neural net dimensions: 50*256*128*64*5 Accuracy:56.7%

# Results

## Model Evaluation and Validation

I used a validation set which was 20% of the total dataset(exact size:31212 samples). The final hyperparameters and neural network parameters are a result of repeated trial and error based on a certain driven intuition.
The model that I created had a final validation accuracy of 56.7%, and a training accuracy of around 56%, which was the highest validation accuracy among all the tried combinations of parameters. The training and validation accuracy is nearly the same, which indicates that there is no overfitting occurring in the model. The model is able to, with a particular certainty, classify the reviews according to their associated sentiment. The model is quite robust, and can be applied to different training data and testing data. However, as it comes with any natural language processing problem, generalization is always an issue if enough data has not been given to the model. I have provided dropout on the first two layers of the neural network to improve generalization. In the case of my model, the training data was severely insufficient to produce satisfactory generalization. Given more distinct training data, the model will be able to do a much better job.
However, one benefit of training on this dataset is that small changes/errors in the dataset will not affect this model greatly due to the way the review is split into phrases. If we observe the following:

| | PhraseId | SentenceId | Phrase | Sentiment |
|---|---|---|---|---|
| **0** | 1 | 1 | A series of escapades demonstrating the adage ... | 1 |
| **1** | 2 | 1 | A series of escapades demonstrating the adage ... | 2 |
| **2** | 3 | 1 | A series | 2 |
| **3** | 4 | 1 | A | 2 |
| **4** | 5 | 1 | series | 2 |

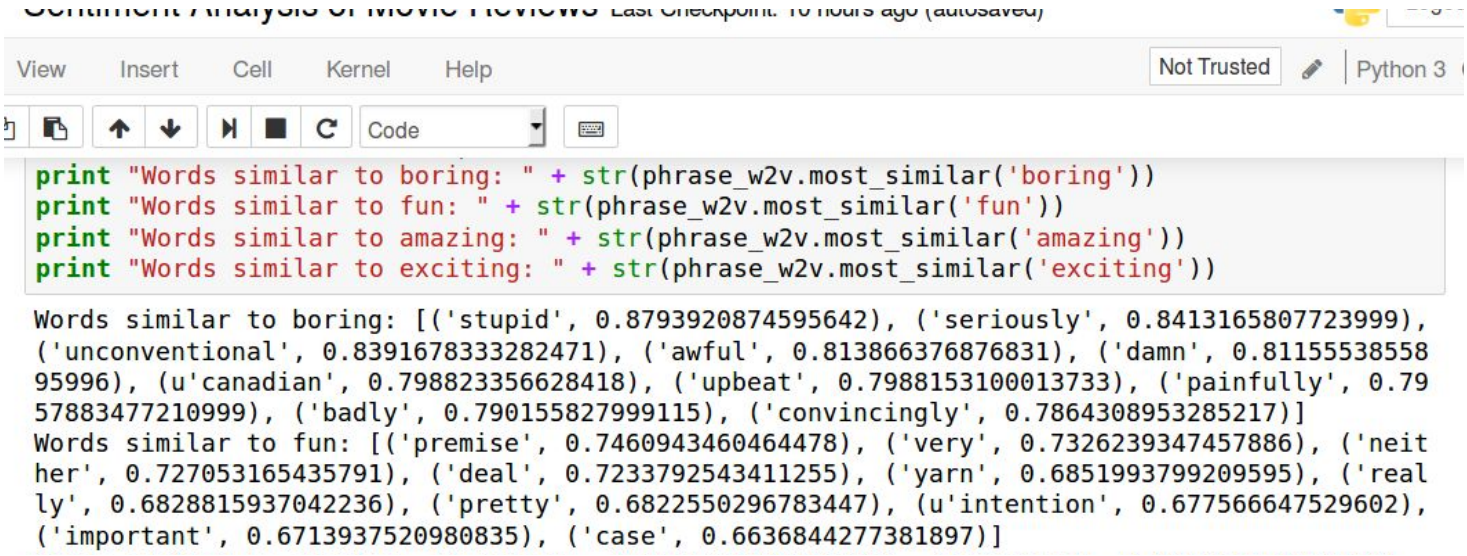
Each sentence is split into phrases, and possible splits are entries in the dataset. If one entry, say entry 2 was ‘A seraes’(some misspelling of series), this would not disturb the model’s interpretation of ‘series’ since there are other entries which provide enough context for it.

## Justification

My model is not up-to-mark with the level of accuracy and generalization that my benchmark model has achieved. There are several reasons that this could be. Although I am not able to view how the contest leader has created his model, I have realized a few mistakes in the design structure of my model, and the approach I took to this problem.
The leader of this contest achieved an accuracy of 76.5%, whereas my categorical accuracy was far less, 56.7%.

# Conclusion

## Free Form Visualization




One of the main reasons I undertook this project is to realize the effectiveness and power of Word2Vec. I had the chance to enjoy using Word2Vec capabilites. A Word2Vec model is capable of finding words similar to a given word. For example: Although there are a few words which are weirdly connected, you can notice that the model is able to realize that words like boring, stupid, awful are similar, exciting and inspirational are similar. The fact that the model is able to realize similarities between words when only given a corpus of text, and not knowing the English language, is something to be admired.

## Reflection

The entire end-to-end problem that I approached and the process that I went through in an attempt to solve it is summarized below:

1. Found a relevant problem and an associated dataset.
2. Found a benchmark solution to the problem.
3. Design an attempted approach and solution to the problem.
4. Libraries like nltk,gensim, and keras were explored and tested.
5. A step-by-step process was implemented for pre-processing.
6. A neural network structure was implemented and tested.
7. Trade-offs between parameters and implementations were discovered and tuning was carried out to optimize the solution.
8. Finally, the whole process was organized and reported after a lot of trial and error of the implementation.

This was my first project venture in natural language processing, which I had no prior experience on. I had to first study the theory behind and possible implementations of language models and sentiment analysis. I had to familiarize myself with various methods to pre-process and optimize natural language analysis.
I was largely impressed with the power of Word2Vec. Gensim made the implementation of Word2Vec very convenient and allowed me to access more of its capabilities. I feel that among all possible implementations of context analysis(Bag of words, CBOW), the Skip-gram implementation as Word2Vec was the most effective. I was also very pleased with how NLTK made pre-processing much more easier and quicker than it had to be.

## Improvement

There are many areas in which I could have improved my solution:

1. The main reason for the lack of accuracy, I believe, is the lack of use of a neural memory model like an LSTM/RNN/GRU network in the creation of phrase vectors. Although the TF-IDF weighted average method was a step-up from crude sum/average of word embeddings, a memory model would have captured the essence of the context of the words in a much more effective manner, and would have generated much better phrase vectors.
2. The use of GPU's would have allowed me much more flexibility and speed in training the Word2Vec/Keras neural network.
3. There could be additional metrics I could have used to evaluate the model.
4. I could have used a Doc2Vec implementation rather than a Word2Vec implementation, which instantly yields sentence vectors. However, I was unsure how to proceed with this implementation.