IMDb Movie Genre Classifier with LSTM

Daniela Raygadas Dominguez, Dayun Piao, Om Prakash Singh, Hari Prasad Kotapalli
Drexel University
{dr976, dp636, os338, hk672}@drexel.edu

Abstract

In this project we attempted to build a movies genre classifier using the IMDb movies extensive dataset available in Kaggle and using Long Short-Term Memory (LSTM) as our deep learning model. We experimented with different models by looking at using pre-trained word embeddings vs using an embedding layer and using all classes in the dataset vs using a smaller number of classes which had a higher number of observations in the dataset. From this project, we concluded that the best model resulted from using pre-trained word embeddings and using a lower number of classes.

1 Introduction

In this ever-evolving today's world, we find many sources of entertainment among which movies has its significance. Movie Genre may be defined as a category of artistic composition, characterized by similarities in form, style, or subject matter. While, IMDb, in full Internet Movie Database was first launched in 1990 & it is the world's most popular and authoritative source for movie & TV content, designed to help fans explore the world of movies and shows and decide what to watch. It is also the essential resource for entertainment industry professionals.

To be specific, a film genre is a motion picture category based on the narrative elements that relate to the main driving force behind the story arc. An easy way to identify the elements of genre is to piece together the narrative arc:

Story (Action) + Plot + Character + Setting = Genre

Based on wordy description of movie in IMDB genre is described as it shows how similar features are repeated in a movie and how those features include most of the literary terms that are featured in other movies within the same genre classification. Thus, determining the entertainment content.

Given the reference of identification of sentiment in textual description of movies on IMDB, it is an important field of study with social media platforms such as IMDB and Twitter, as it captivates the interest of researchers in network analysis and natural language processing (NLP) techniques. In the past few years deep learning stands out in particular, and we see a significant increase in deep learning and neural network methods such as CNN and LSTM networks which were subsequently used. Supervised SVM and Lib.linear were also very popular, with several participants combining SVM with neural network methods or SVM with dense word embedding features.

Among the multiple project ideas brought onto the table, we observed movies as a common idea & had individual contributions to anchor on this topic. It is not just to experiment with our academic understanding of CNN & LSTM concepts, but also to enrich practical expertise on deep learning techniques, adding to which movie we wanted to correlate with the facts such as, viewers most of the times are inspired by accurately classifying the content.

1.1 Project goals

Our goal for this project was to produce a genre classifier based on the deep learning architecture of LSTMs. The pipeline of this text classification model could fit in other applications such as categorizing books into different genres according to book description or abstract. Or any other general text data classification task with given sets of labels.

1.2 Neural methodology

For this project the initial plan was to use the proposed architecture in the research paper we selected which is a combination of CNNs and LSTMs as our neural methodology approach. The combination of CNNs and LSTMs provided the best results on the SemEval 2017 Task 4. However, we decided to only focus on LSTM as our deep learning model.

Long Short Term Memory networks – usually just called "LSTMs" are a very special kind of recurrent neural networks (RNN), which are capable of learning long-term dependencies, unlike traditional NNs.

RNN are networks with loops in them which prefer information to persist. These loops make recurrent neural networks seem, kind of mysterious. However, if you think a bit more, it turns out that they are not all that different than a normal neural network. a recurrent neural network can be thought of as multiple copies of the same network, each passing the message to a successor.

A chain like nature reveals that recurrent neural networks are intimately related to sequences of lists. They are the natural architecture of neural network to use for NLP data.

LSTM is a very special kind of neural network which works for many tasks, much better than the standard version & results in RNN.

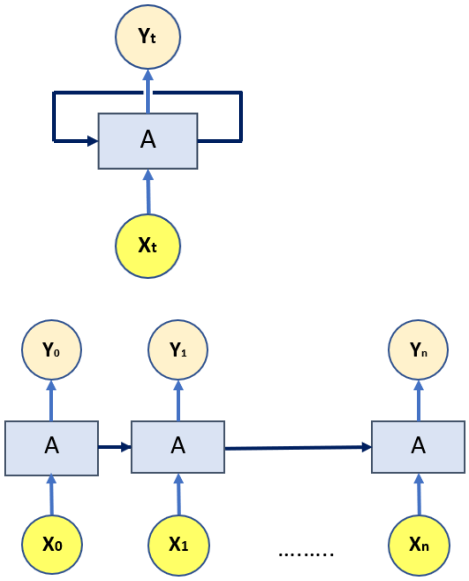

Figure1: Recurrent Neural Network
Figure 2: Unrolled Recurrent Neural Network

In this above Figure 1, represents a chunk of neural network (NN), with A, looks at some sample input Xt, and produces an output value Yt. While in Figure 2, the linked or chained neural networks represent a well-connected sequences and lists of words (inputs: X0, X1...Xn & Outputs: Y0, Y1...Yn). RNN results as a successful working model for long term dependency problems which are applicable in todays evolved world of AI assisted projects.

2 Materials

For this project, the dataset we selected was the IMDb movies extensive dataset available in Kaggle. IMDb is the most popular movie website and it is known for storing almost every movie that has been released. The dataset includes 85, 855 movies with attributes such as movie description, average rating, number of votes, genre, and many others. The deep learning framework we decided to use for this project was Keras. Keras is a powerful and open-source Python library for developing and evaluation deep learning models.

3 Methods

3.1 Data Preprocessing

The first step of this project was to preprocess our data. The IMDb movies dataset from Kaggle had a total of 85, 855 rows and 23 columns. Some of the movies in our dataset had multiple genres associated to that movie which would refer to a multi-label classification in which one sample can have multiple classes at once. Therefore, this was out of our project scope which was to build a multi-class classification in which we have one class per sample. To solve this, we created a new column called first label that would refer to the first listed genre. By doing this, we made sure we only had one class per sample. Another step we took to preprocess the data was to remove any missing values in the description feature as for this task we are only interested in two features, the description of the movie that we used to predict the genre and the newly created first label feature that contains the actual genre this movie belongs to. Finally, we lower case the descriptions, removed any special characters, and only kept the alphanumeric characters in our text descriptions.

3.2 Exploratory Data Analysis

After preprocessing our data, we looked at how many movies we had per movie genre. We could observe that our data was highly imbalanced, and we had a total of 23 labels to predict from. The distribution of samples across class label was not equal. For some of the labels we had as low as two samples for those labels. Therefore, we also decided to explore this in our project and compare both models. One model having all 23 classes and the second model having only the classes with the most instances in this case we took the classes that had over 1000 instances which were only 9 classes.

Moreover, we looked at the description length distribution and we observed that a length of 44 represented about 99% of all the descriptions. This analysis helped us in defining the sequence length that all our text description should have by truncating longer text to this size or padding shorter text with zeros up to this size.

3.3 LSTM Model

For this project we decided to use Long Short-Term Memory (LSTM) as our deep learning model to build our movie classifier. Throughout this project, we did not experiment with the parameters of LSTM, but rather kept those constant in our models.

We define the max_sequence_length to be 44 and set the embedding dimensions to 100. We define the LSTM layer with 100 memory units and drop out of 0.2, the activation function as SoftMax, and categorical_crossentropy for the loss function. We train our models over 10 epochs with a batch size of 64 and define an early stopping call to detect an increase in the validation's loss.

For all of our models we created a classification report in order to evaluate the model using the f-1 weighted average as we have a highly imbalance dataset.

4 Experiment

For this project we decided to experiment and compare the performance of model with and without pre-trained word vectors in embedding layer. For this project we decided to use Glove embeddings as it does not just rely on local statistics but incorporates global statistics in this case the word co-occurrence to obtain word vectors.

The first model we built did not include the pre-train model. We used default Keras Tokenizer for the first model and weighted average accuracy is 0.43, recall of 0.45, f1-score of 0.43 after training of 5 epochs due to early stopping recall. The model's loss and accuracy across all the epoch shows there is overfitting issue.

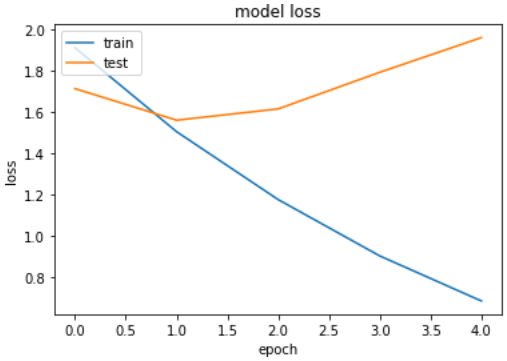

Figure 3: First Model Accuracy

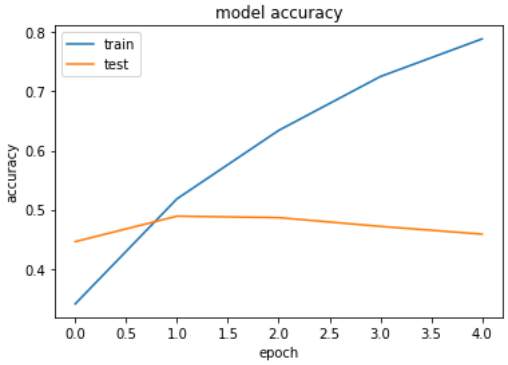

Figure 4: First Model Loss

For the second model, we tied pretrained Glove with first embedding layer. This simply not only increase the performance on test set of getting better weighted average metrics after 10 epochs, but also enhanced the model's ability to generalize the predictions on test set. Compare the second model's accuracy and loss graph with the first one, the overfitting issues seems resolved by adding pretrained word vector.

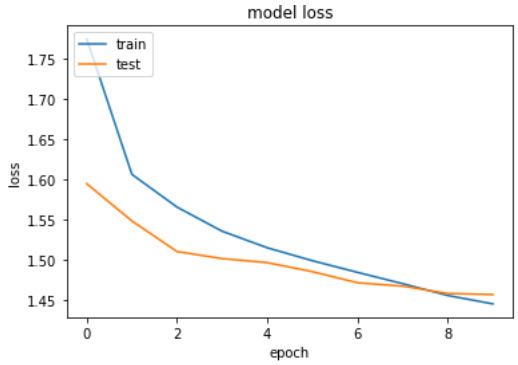

Figure 5: Second Model Accuracy

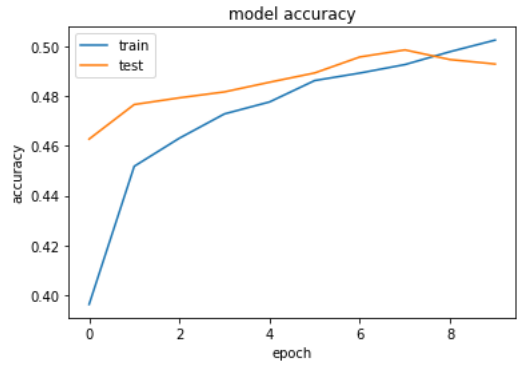

Figure 6: Second Model Loss

For the final model, the labels being predicted is limited to only 9 class which has more than 1000 data for each. As a result, the amount of data in each class does help improve the model's performance as the model keeps improving prediction on test set with after 10 epochs with weighted average accuracy of 0.51, recall of 0.52 and f1-score of 0.51. The loss and accuracy graph across each epoch is also improving after limited labels.

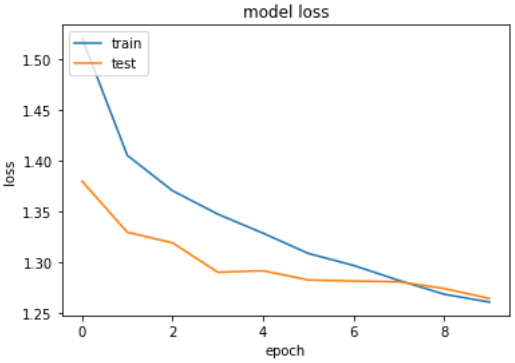

Figure 7: Third Model Accuracy

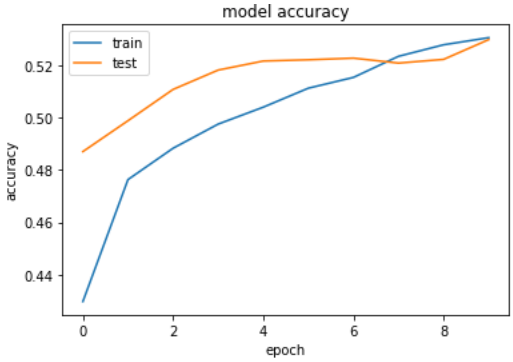

Figure 8: Third Model Loss

5 Results

Model	Weighted F1 score
Embedding Layer/full set of classes	0.43
Pre-trained word vectors/ full set of classes	0.47
Pre-trained word vectors/ reduced classes	0.51

Figure 9: LSTM Models Results

In the above figure we can observe the performance of our three models. In this table we recorded the weighted F1 score. As we can observe the last model had a better performance with a weighted F1 score of 0.51. This showed that using pre-trained word embeddings had a major impact on the model and resulted in a higher F1 score. Additionally, having a smaller size of class labels does help improve the model's performance.

6 Challenges

Some of the major challenges we had on this project was working with a highly imbalanced

dataset in which the distribution of samples across labels was not equal. Hence, we wanted to experiment on this and see if having a lower number of classes that had a higher number of samples would result in a higher performance in our model. Another challenge was having a multi-label dataset for a multi-class classification problem, to solve this we simply took the first genre on each movie in order to be able to use this dataset for our multi-class classification problem. The final challenge we had was that the movie description in the dataset was not complete. For some reason the description was cutoff after a certain number of characters and followed by an ellipsis. This issue might have come from the people in Kaggle that created this dataset and did not acquire the data correctly. This was a major challenge because we might have been missing some words that would have been very descriptive or particular to that movie genre.

7 Conclusion and Future Work

This project consisted of a simple LSTM for a multi-class classification problem in which we decided to explore how using pretrained word vectors and using a lower number of classes that had a higher number of samples would impact the performance of the model. From this project we concluded that for this dataset the use of pretrained word vectors and a more balance dataset improved the performance of the model.

This project can be taken further by exploring other deep learning models such as CNN and see if the performance is better compared to the LSTM, or even look at combining these two models together to achieve a better performance. Another thing that can be done in the future to improve this project is try different pretrained vectors such as Word2Vec instead of Glove. One could also explore different model configurations through hyperparameter tunning as we did not experiment on changing the configuration of our model and kept everything constant instead. Finally, this dataset could be used to look at a multi-label classification problem instead of a multi-class classification as we did.

References

BB_twrt at SemEval 2017 Task 4: Twitter Sentiment Analysis with CNNs and LTMs. *Mathieu Cliché, Bloomberg. mcliche@bloomberg.net*

"IMDB History": *IMDb*: https://help.imdb.com/article/imdb/general-information/why-should-i-register-on-imdb/GHB62T7USTMYMCDC?ref_=helpart_nav_2#

Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling *Hasim Sak, Andrew Senior, Franc¸oise Beaufays Google, USA {hasim,andrewsenior,fsb@google.com}*

Colah's blog. August 27, 2015 *Understanding LSTM Networks (https://colah.github.io/posts/2015-08-Understanding-LSTMs/)*