Final Year Project

Interim Report

Spoiler Alert: Deep Learning with Natural Language Processing to identify spoilers

By Yasmin Paksoy

January 2022

Abstract

Twitter being the endless sea of information that it is, it has become impossible for users to avoid TV show/movie spoilers. In the day and age of TV shows being released by season over episode, users have no way of knowing which spoilers they may run into when logging into social media platforms. An algorithm to identify and block these spoilers would allow users to scroll stress free. This is a problem faced by most Twitter users, with Twitters algorithm showing users most recent tweets about topics they are interested using the data they collect on users.

This project uses deep learning on a dataset of movie reviews to train a neural network model to identify spoilers. This paper presents the developments made so far, in particular the use of LSTM and GRU models for this problem, and includes a breakdown of the future steps, which includes the use of transformers.

Keywords

- Deep learning
- Spoiler detection
- Classification
- LSTM
- GRU

Table Of Contents

Table Of Figures

1 Introduction

Spoilers, defined as information about the plot of a motion picture that can spoil a viewer's sense of surprise or suspense [1], have the potential to ruin any entertainment experience: Jon kills Daenerys, Dumbledore dies, Bruce Willis is dead the whole time.

The problem of consumers running into spoilers has been well-documented. There are many articles online detailing the hardship users face when new movies or TV shows are released [2]: many users avoid social media all together until they've seen the new release [3], while others even avoid talking to their friends and family in case they might accidently let a spoiler slip.

This long existing issue has been amplified due to technology in recent years. Before the growth of the internet and social media, the risks were contained to the people physically around you. Now, streaming services such as Netflix release new episodes a season at a time; the days of watching a new episode at a scheduled time with the rest of the fans has ended, with viewers now watching TV shows at the time and speed they'd like. "Live Tweeting", the process of sharing tweets during an event, such as watching the finale of a TV show [4], has also heightened the risk of running into spoilers. Worst of all, even with viewers avoiding social media, they can receive spoilers via push alerts, with newspapers like The New York Times sending app users notifications containing spoilers [5].

This is the main motivation for this project; this is an issue that has long been a source of stress in many people's lives, and with the advances in machine learning, it no longer needs to be.

Deep learning and artificial intelligence has been a growing field in the past decade; with the growth of the internet and access to high powered CPU/GPUs, the field has been reinvigorated [6]. There has been a lot of investment and competition websites like Kaggle has encouraged many computer scientists to join the field. Regardless of this, there has not been concrete development in the area of avoiding spoilers; while there has been similar research to that done in this project, it has not been executed to its fullest potential. Algorithms have not yet yielded high accuracy scores or been integrated into a platform in a way that is accessible to users. This is the main objective of this project; to not only train a model for spoiler detection, but to make it possible for Twitter users to avoid spoilers using it.

2 Literature Review

As mentioned previously, there has not been tangible development in this area from a user perspective. While there have been many research projects pursuing possible solutions to this problem, none of them have been available to users. Out of the few available projects to review for this concept, I have decided to focus on one in particular. This is the paper I believe to be the most complex, up to date and similar to what I am trying to achieve for this project. I have also looked at other research, which I will briefly discuss.

Spoiler Net [7], developed by the computer science department at the University of California San Diego in 2019, was trained identify spoilers from a dataset consisting of Good Reads book reviews. The dataset identified spoilers on a sentence level; each sentence was labelled as contains spoiler/does not contain spoiler. They used Gated Recurrent Networks (GRUs) to train the model and achieved an accuracy of 89% for book spoilers and 74% for movie spoilers.

The main difference between this project and my own is the way they detect spoilers on a sentence level, and so not treat the review as a whole. With each tweet only being a maximum of 280 characters, I did not see this as necessary; a single tweet is not long enough for there to be a significant portion that does not contain spoilers, and thus should not be blocked. Due to this, I have selected a dataset that would treat each review as a whole to train and test my deep learning models.

Another research paper, "Killing Me" Is Not a Spoiler [8], used Graph Neural Networks (GNNs) to leverage the dependency relations between context words in sentences to accurately capture the semantics of the sentence. Unlike attention based models, their model was able to understand metaphors. They also used the Goodreads dataset, along with the TVTropes dataset, where achieved an accuracy higher than the one achieved by Spoiler Net.

There have been other papers published addressing this subject, using a range of different methods. For example, a paper names "Spoiler detection in TV program tweets" [9] in 2015 was also aimed at detected spoilers in tweets. This project used a support vector machine (SVM) based prediction model and was then trained on tweets collected about the TV show Dancing with the Stars season 13 and the 2014 World-Cup final. Contrary to the other papers, his was a self-training model, "to reduce the amount of labelled data in an experiment that displays a similar level of performance".

3 Methods

There are many different methods that could be used to train a model for such a binary classification problem. The main method I am using for this project is deep learning with python. So far, I have experimented with dense, Long Short-Term Memory (LSTM) and GRU layers in various deep learning models using Keras and TensorFlow on a Jupyter notebook.

3.1 Dataset

The dataset chosen for this project was found on Kaggle, names the IMDB Spoiler dataset [10]. It is a dataset developed using IMDB and contains two distinct databases, movies and reviews. The movies dataset contains information regarding the 1572 movies the reviews are based on.

The second dataset, reviews, contains over 500,000 data samples. Each sample consists of the review text, rating, summary etc., along with a boolean field that indicates whether the review contains a spoiler or not. Of these over 573,913 reviews, 150,924 of them actually contain spoilers.

I choose this dataset for a series of reasons; not only does it have a significant number of reviews, but it also has labels indicating whether each review contains a spoiler. IMDB allows users to know which reviews contain spoilers by tagging the reviews with "Warning: Spoilers". Reviews which include spoilers but have not been indicated to do so by the reviewer will be removed when reported under the "Spoiler without warning" option. This method depends on crowdsourcing, which can usually be unreliable, however, with IMDB being the biggest movie/TV review platform available, I believe it would be reliable enough to train these models.

3.2 Pre-processing the data

The reviews dataset is provided as a JSON file. In order to make it accessible for the models to train on, I tokenized and vectorised the data. Tokenizing was achieved by using the Tokenizer module from Keras. During this process, I set the maximum length to 500 and the maximum words in total to 15,000. I then vectorized the data by using the pad_sequences module from Keras, which converted the tokenized data into 2D tensors that could then be inputted into a deep learning module, as shown below.

```
[ ]
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import numpy as np

maxlen = 500 # cuts reviews off after 500 words
max_words = 15000 # considers only the top 15000 words in the dataset

# tokenizing texts
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

# vectorizing texts
data = pad_sequences(sequences, maxlen=maxlen)

# vectorizing labels
labels = np.asarray(labels).astype('float32')

print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)
```

```
Found 379856 unique tokens.
Shape of data tensor: (573913, 500)
Shape of label tensor: (573913,)
```

Figure 1: Tokenizing and vectorizing the data

From here, I split the data two times, once into training and testing, the next into partial training and validation after shuffling the data. The training set consisted of 200,000 data samples, while the testing set consisted of 100,00. The training was then split into sets containing 150,000 and 50,000 samples, for partial training and validation sets respectively.

3.3 Optimiser

The optimiser will adjust the model to get the best possible performance on the training set. If the model is optimised too strongly, it will not generalise well, meaning the testing process on the test data will produce lower than expected results. The optimiser used when training models for this project is the RMSprop optimiser, one of the most used optimisers in machine learning problems.

3.4 Models

Going into training models, I wanted to find the best model is terms of both high accuracy and low computational power. In order to compare the models, I started with the most basic method for training such a network: using a bag-of-words algorithm with dense layers and word embeddings. The second set of models consisted of LSTM and GRU models.

3.4.1 First Set of Models

For these first set of models, I split the dataset into 200,00 training samples and 100,000 testing samples. I then further split the training set into 150,000 partial testing samples and 50,000 testing samples. I then experimented with different configurations of models.

The initial model consisted of two layers used for word embeddings, 3 dense layers for classification and a last output layer. The model definition can be seen in figure 2.

```
[ ]
from tensorflow.keras import models
from tensorflow.keras import layers
from keras.layers import Embedding, Flatten

model = models.Sequential()

# Embedding layers
# network will learn 8-dimensional embeddings for each of the 15,000 words
model.add(Embedding(15000, 8, input_length=maxlen))
# flattens the 3D tensor output to a 2D tensor
model.add(Flatten())

# training single dense layer for classfication
model.add(layers.Dense(128, activation = 'relu'))
model.add(layers.Dense(64, activation = 'relu'))
model.add(layers.Dense(32, activation = 'relu'))

# output layer
model.add(layers.Dense(1, activation = 'sigmoid'))

model.compile(optimizer = 'rmsprop',
              loss = 'binary_crossentropy',
              metrics = ['accuracy'])

model.build()
model.summary()
```

Figure 2: Initial model using word embeddings and dense layers

I then tuned the learning rate and regularized this model. The regularization methods used consisted of weight regularization and adding dropout. Weight regularization forces the network to set the weights to small values, making the distribution of them more regular [6]. This is accomplished by adding a cost to the loss function that applies to the higher weights only. There are two types of cost, L1, where the cost is equivalent to the absolute value of the weight coefficients, and L2, where the cost is equivalent to the square of the weight coefficients. I experimented with both.

Dropout is a technique which sets to zero a number of output features of the model during training [6]. This method breaks apart coincidental patterns that are probably not significant, thus making the final model more generalisable.

3.4.2 Second Set of Models

The second set of models trained used LSTM and GRU layers. Before I started training these models, I altered the dataset. Doing more research showed me that having a balanced dataset would be more beneficial to my model. This is due to the inequality of the two classes in the set, where the spoiler to non-spoiler ratio is 1:3, which is highly imbalanced. This could result in the models learning which samples don't contain spoilers, rather than which do, as 75% of the training samples don't contain spoilers [6]. Figure 3 shows the code written to balance the dataset, making the ration 1:1.

```
[5]  1 import json
     2
     3 labels = []
     4 texts = []
     5 sum = 0;
     6
     7 with open('IMDB_reviews.json', 'r') as json_file:
     8   for jsonObj in json_file:
     9     data = json.loads(jsonObj)
    10     if data['is_spoiler'] == True:
    11       labels.append(1)
    12       texts.append(data['review_text'])
    13     else:
    14       if sum < 150924:
    15         sum += 1
    16         labels.append(0)
    17         texts.append(data['review_text'])
```

```
[6]  1 spoilers = 0;
     2 nonSpoilers = 0;
     3
     4 for i in labels:
     5   if i == 1:
     6     spoilers += 1
     7   else:
     8     nonSpoilers += 1
     9
    10 print("Total number of reviews: " + str(len(labels)))
    11 print("Total nummber with spoilers: " + str(spoilers))
    12 print("Total number without spoilers " + str(nonSpoilers))
```

```
Total number of reviews: 301848
Total nummber with spoilers: 150924
Total number without spoilers 150924
```

Figure 3: Creating a balanced dataset

I split the dataset after this process so that the training and testing sets consisted of 251,848 and 50,000 samples respectively, and partial training and validation consisted of 201,848 and 50,000 samples respectively.

I started by making a model with one LSTM layer using this dataset. LSTM layers are a variant of SimpleRNN layers [6], meaning it allows for information to be carried across many timesteps. Saving information for later prevents older signals from gradually getting lost. This is especially important since this is a sequence problem; the previous words in a sentence have an effect on whether the next ones will be classes as a spoiler. I then tuned the dropout for this model.

This dropout is different to the ones added to the first set of models; that type of dropout could obstruct learning as it adds a random dropout mask which would disrupt the error signal. Instead, I added recurrent dropout, which applies a constant dropout mask instead, allowing the network to propagate its learning error though time.

Next, I moved on to training GRU layers, which are less computationally expensive and streamlined when compared to LSTM layers [6], while sharing the same characteristics. I tuned the learning rate of this model, before training my final model for this set.

4 Results

4.1 First Set of Models

Figure 4 shows the results of every model run in the first set of models. This table shows the configuration of the tuned hyperparameters and regularization added, along with the result for the optimal number of epochs.

Model	Learning Rate	Weight Regularization	Dropout	Optimal Epochs	Validation Loss	Validation Accuracy
1	0.001	N/A	N/A	2	0.5113	0.7658
2	0.005	N/A	N/A	4	0.5758	0.7607
3	0.003	N/A	N/A	3	0.5304	0.7646
4	0.0005	N/A	N/A	1	0.5134	0.7612
5	0.001	L1	N/A	5	0.7946	0.7629
6	0.001	L2	N/A	3	0.5084	0.7662
7	0.001	L1 & L2	N/A	8	0.7931	0.7650
8	0.001	N/A	0.3	2	0.5103	0.7655
9	0.001	N/A	0.2	2	0.5047	0.7670
10	0.001	N/A	0.1	2	0.5079	0.7658

Figure 4: Results of first set of models

The best model from this set was model number 9, as it has the lowest validation loss. Due to this, the final was trained and tested using this configuration, shown in figure 5. This achieved a decent accuracy score of 76%.

Model	Learning Rate	Weight Regularization	Dropout	Loss	Accuracy
Final (9)	0.001	N/A	0.2	0.5277	0.7607

Figure 5: Results of final model from first set

4.2 Second Set of Models

Figure 6 shows the results of every model run in the second set of models. This table shows the configuration of each model, along with the result for the optimal number of epochs.

Model	LSTM/GRU	Learning Rate	Recurrent Dropout	Optimal Epochs	Validation Loss	Validation Accuracy
1	LSTM	0.001	N/A	10	0.4992	0.7567
2	LSTM	0.001	0.2	13	0.4941	0.7562
3	LSTM	0.001	0.4	8	0.5061	0.7577
4	LSTM	0.001	0.3	16	0.4972	0.7615
5	GRU	0.001	0.2	10	0.4847	0.7652
6	GRU	0.003	0.2	1	0.4819	0.7647
7	GRU	0.006	0.2	6	0.4849	0.7640
8	GRU	0.004	0.2	3	0.4841	0.7642
9	GRU	0.01	0.2	5	0.4849	0.7637

Figure 6: Results from second set of models

The best model from this set was model number 6, as it has the lowest validation loss. The final was trained and tested using this configuration, shown in figure 5. This achieved an accuracy of 76%.

Model	LSTM/GRU	Learning Rate	Dropout	Loss	Accuracy
Final	GRU	0.003	0.2		

Figure 7: Results of final model from second set

5 Discussion

My first set of models achieved a final accuracy score of 76% matching the average validation accuracy achieved by all the models, which was 0.76447, showing that the models were not over-tuned. This would have meant the models did very well on the validation accuracy, but fit those samples too closely, causing it to generalise poorly. Since the accuracy of the final model was just as high, the model generalises well.

However, these models were trained on an imbalanced dataset, so the results cannot be regarded too highly; a ratio of 1:3 is too high for this model to be said to have successfully understood how to classify text that contains spoilers to its fullest potential.

My second set of models achieved a final accuracy score of 73%. The average validation accuracy for this model was 0.7615, showing that the model did worse when testing on the testing set. Although this model did 3% worse when compared to the final model from the first set, I believe it has more potential; the first model consisted of 6 layers in total, 2 for word embeddings and 4 dense layers. The LSTM and GRU models however, only consisted of one layer.

This is due to the computational power required to train these models. The first set of models were by far easier to train; not only did it use less data for training, at 150,00 compared to 201,848, but took my less time and computational power. LSTM and GRU not only take longer to train, but also require more graphical processing unit (GPU) power. For example, I attempted to train an LSTM model consisting of two layers; this model trained for 8 hours before failing to complete training due to the runtime timing out. I believe training bigger models like this will achieve accuracy higher than 76% but was unable to do so in those circumstances. Since training these models, I have acquired more GPU, tensor processing unit (TPU) power and access to longer runtimes via Google Colab Pro, I believe this will now allow me to train these bigger models and achieve high accuracy scores.

When comparting the highest achieved score of 76% to other projects mentioned in my literature review , it's clear that these results are significant. Spoiler Net achieved a final result of 76% at the end of their project [7], while I have achieved this score halfway down the line. I believe with more work, I will be able to beat this accuracy level as stated in my initial project proposal.

In my initial project specification, I expressed that I wanted to reach an accuracy score above 80%. The scores achieved show my progress towards this; they are significant in that they are on 4 to 7% away from my initial objective. With more complex models, it should be possible to each 80% soon.

6 Conclusion

This report outlines the models I have trained so far and the results they have attained, showing the strengths, such as the large dataset, and weaknesses of the project, such as the lack of computational power available until now. I believe looking at these results and keeping such weaknesses in mind make it clear how close my project is to being a success and achieving one of the higher scores this subject has seen till now.

7 Progress & Future Plans

In my initial project specification, I stated that I would make use of the Natural Language Processing Toolkit but have not found this necessary while using TensorFlow and Keras. Additionally, I has believed I would use the Twitter API to help collect tweets which can then be classified as spoiler/non-spoiler, but the amount of time needed to class each Tweet would have been a significant portion of the time allocated for this project, so I decided to use the IMDB Spoiler dataset instead.

Now that the halfway point has been reached and this project is so close to reaching the minimum viable product outlined in the initial project specification, I can focus on trying more complex methods to achieve the highest accuracy possible, confident that those models will not have diminishing returns when comparing the computational power and accuracy achieved.

Moving forward, I not only want to continue training more LSTM and GRU models, but also experiment with transformers. Transformers are a state of the art method for training natural language processing machine learning problems. They use self-attention to compute representations of sequences to find correlations between different words, indicating the syntactic and contextual structure of the sentence [11]. This will allow me to train models that should reach higher accuracy scores when compared to LSTM and GRU layers.

Additionally, so far, I have been training on 500 words per review. Tweets are at max 280 characters. Due to this, the models I have trained may not be as successful at picking up on spoilers on such short pieces of text. I would like to experiment with training model with less words per review, as the difference between 280 characters and 500 words is immense as the project progresses.

References

[1] Merriam Webster. n.d. Spoiler. [online] Available at: <https://www.merriam-webster.com/dictionary/spoiler> [Accessed 31 January 2022].

[2] Alexander, J. 2021. How Trying to Avoid Spoilers in 2021 Became Futile. IGN Middle East. [online] Available at: https://me.ign.com/en/movies/186513/feature/how-trying-to-avoid-spoilers-in-2021-became-futile [Accessed 20 October 2021].

[3] Rose, S. 2021. A tangled web: have super-fans ruined the new Spider-Man movie? The Guardian. [online] Available at: https://www.theguardian.com/film/2021/feb/15/have-super-fans-ruined-new-spiderman-movie [Accessed 18 October 2021].

[4] TheFreeDictionary.com. n.d. live tweeting. [online] Available at: <https://idioms.thefreedictionary.com/live+tweeting> [Accessed 31 January 2022].

[5] McCormick, R., 2015. The New York Times spoiled the ending to HBO TV show The Jinx on Twitter. [online] The Verge. Available at: <https://www.theverge.com/2015/3/15/8221055/hbo-show-the-jinx-spoiled-on-twitter> [Accessed 31 January 2022].

[6] Chollet, F., 2018. Deep learning with Python. 1st ed. Shelter Islands: Manning.

[7] Riccio, G. 2019. SpoilerNet, the anti spoiler AI: it signals them to you and avoids them. Futuroprossimo.it. [online] Available at: https://en.futuroprossimo.it/2019/07/spoilernet-anti-spoiler [Accessed 18 October 2021].

[8] Chang, B., Lee, I., Kim, H. and Kang, J. (2021). "Killing Me" Is Not a Spoiler: Spoiler Detection Model using Graph Neural Networks with Dependency Relation-Aware Attention Mechanism.

[9] Jeon, S., Kim, S. and Yu, H., 2016. Spoiler detection in TV program tweets. Information Sciences, 329, pp.220-235.

[10] Misra, R., 2019. IMDB Spoiler Dataset. [online] Kaggle.com. Available at: https://www.kaggle.com/rmisra/imdb-spoiler-dataset/metadata [Accessed 27 December 2021].

[11] Adaloglou, N., 2020. How Transformers work in deep learning and NLP: an intuitive introduction. [online] Theaisummer.com. Available at: <https://theaisummer.com/transformer/> [Accessed 30 January 2022].

Appendix

All the code mentioned can be found at this link:

https://github.com/ysmnpksy/Final-Project