Movie Spoiler Detection

Group Name: KanyaRaasi

Group Members:

First name	**Last Name**	**Student number**
Akash	Guje	C0835384
Noah	David	C0846073
Bhanu Prakash	Mahadevuni	C0850515
Pramod Reddy	Gurrala	C0850493
Sai Varun	Kollipara	C0828403

Submission date: 22-04-2023

Table of Contents

Abstract:

Movie spoilers are a widespread problem for movie enthusiasts who want to enjoy a film without prior knowledge of the plot. In this project, we propose a solution to predict the presence of spoilers in movie reviews using natural language processing techniques and blur them on any website so that it is not visible to the end users of that website. The machine learning model was trained on a dataset of movie reviews labeled as either containing spoilers or not. Our model uses a combination of text preprocessing techniques, feature extraction, and classification algorithms to predict the presence of spoilers in each review. We evaluate the performance of our model using metrics such as precision, recall, and F1-score. Our experimental results show that our approach can accurately predict the presence of spoilers in movie reviews with an F1 score of 0.85. This research can help prevent spoilers and improve the movie-watching experience for viewers who want to avoid plot spoilers.

Introduction:

Movie reviews are essential to the entertainment industry as they help individuals decide what movies to watch. However, many websites contain spoilers that can ruin the movie-watching experience for individuals who have not seen the movie. So, there is a need to develop a solution to detect spoilers in movie reviews and blur them on the website. The business problem that we aim to address is the presence of movie spoilers in online reviews. Spoilers not only ruin the experience for movie-goers but can also lead to a decrease in box office sales. Moreover, websites that allow spoilers can lose traffic and revenue due to the negative impact on user experience.

Proposed Solution:

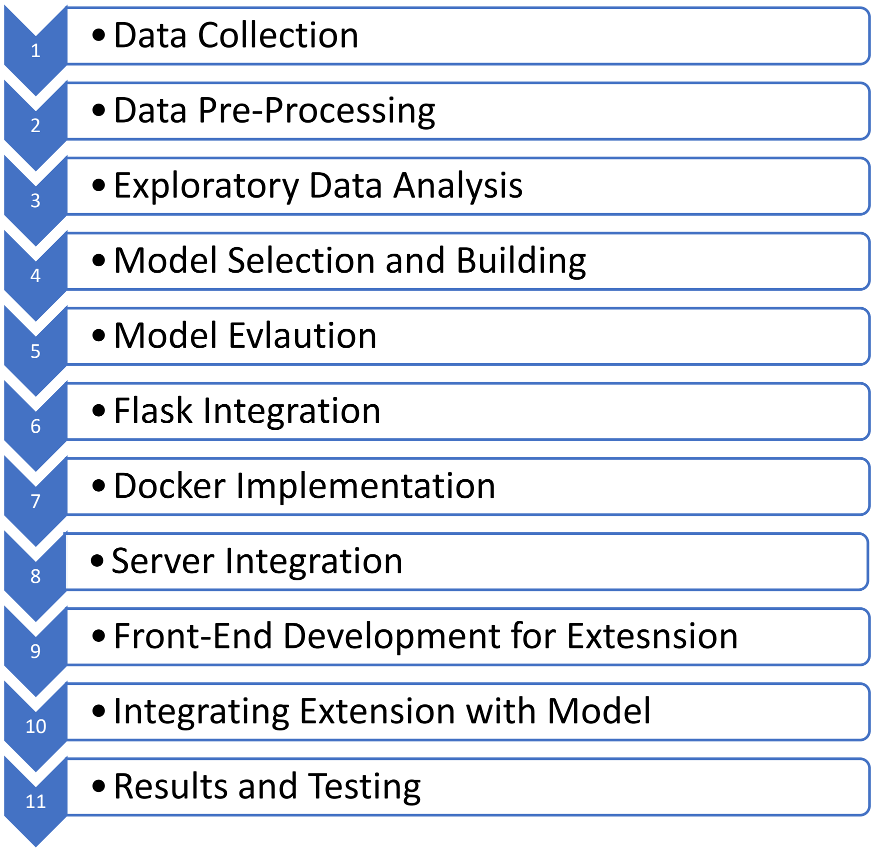
To address this problem, we propose a solution using Natural Language Processing (NLP) techniques to detect spoilers in movie reviews automatically. The proposed method involves training a deep learning model on a large dataset of movie reviews to identify keywords and phrases indicative of spoilers. Once detected, the identified spoilers will be automatically blurred out on the website, ensuring that users who have not seen the movie are not exposed to the spoilers.

Workflow:

The proposed solution method will be implemented in the following steps:

1. **Data Collection:** We will collect the dataset for movie reviews from IMDB.
2. **Data Pre-processing:** We will perform data cleaning, tokenization, and stop-word removal to prepare the dataset for model training.
3. **Model Training:** We will train and explore different machine learning and deep neural network models and choose one with superior performance to identify spoilers in movie reviews.
4. **Spoiler Detection and Blurring:** The trained model will be connected to a Chrome extension using a Flask framework and will be used to automatically detect spoilers in movie reviews and blur them on the website.
5. **Evaluation:** We will evaluate the performance of the proposed method using metrics such as precision, recall, and F1-score to measure the spoiler detection and blurring accuracy.

Here are the detailed Workflow for the steps involved:



Data Collection:

Data collection is the first step for any project. Since we aim to identify a movie review as a spoiler or not, we need to extract movie reviews and label them on whether they are spoilers. Various websites provide users an opportunity to leave a review so that it can be helpful for the audience that is yet to watch the movie. A movie review can be obtained from websites like Rotten Tomato, IMDB, and Google Reviews.

However, it is essential to reiterate that, apart from movie reviews, we must also know whether a certain review is a spoiler. This is the reason we chose IMDB as our source for reviews. IMDB is one the most popular and reliable websites if you are a movie lover. Apart from reviews, this website also contains all the necessary information regarding movies, from cast lists to plot synopsis. An important reason to choose IMDB is that the website also contains spoiler tags for movie reviews that contain spoilers. This way, we can extract and segregate the spoilers from non-spoilers and label our data which will be helpful when building the machine-learning model.

To extract the reviews from the IMDB website, we have leveraged Python's Beautiful Soup package to extract reviews from "https://www.imdb.com/chart/top/." This webpage consists of top-rated movies on IMDB. We initially send a GET request to the URL using the request package and then create a beautiful soup object. Using this object, we used a method like "find_all" to traverse each webpage by finding the appropriate class name. This way, we collected movie reviews of up to five pages for each movie.

In the end, around 6336 spoilers and 16543 non-spoilers' data were collected from 250 movies.

Data Pre-Processing:

Data must be in a specific format before the statistical analysis since the data associated with the project is text data. It is essential to clean the data and vectorize it. A function named clean () is defined, which takes in a string and leverages the regex package in Python to terminate unnecessary information. The following steps are performed to clean the data.

- Remove Html Tags<>.
- Remove Markdown URLs
- Remove Punctuations
- Remove all URLs.
- Remove @ symbol.
- Remove Brackets
- Remove Newline character \\n.
- Remove &amp.
- Remove Emojis
- Remove the sequence of special characters.
- Remove the sequence of white spaces.
- Convert to lowercase.

Vectorization is an essential step in any NLP project. It is a process of converting text into numbers. There are many vectorization techniques, and all these techniques can be implemented using scikit_learn's feature_extraction.text package. From this package, CountVectorizer and Tfidf vectorizer are used for traditional models. These vectorization techniques only consider the appearance of a word in the document but do not consider the contextual meaning of the word and the series in which the word occurs. So, to vectorize the data based on the context word2Vec Model from the Gensim package is used. The Word2Vec model takes in a set of words in a sentence and converts them into a sentence. This vectorization technique is used for Deep Learning models.

Exploratory Data Analysis:

Apart from collecting reviews, we also collected a Plot synopsis for each movie. Though the plot synopsis was not used as a part of modeling and is in our plan to inculcate in our plans, this was used to find the similarity between the plots and each review. To calculate the similarity score, we have used a couple of methods. Initially, we used Cosine similarity, which finds the cosine angle between two texts. However, the issue with cosine similarity is that it needs to take semantic similarity into context. This is why we used Doc2vec similarity, which overcomes this issue. This similarity score gave us a better understand of how each text is classified as either spoiler or not. However, most importantly, it also gave us an insight into how certain reviews, even after having many similarities, were not classified as spoilers.

Based on the Doc2vec similarity score, the median similarity score is around 0.23 and 0.32 for non-spoilers and spoilers' data. Can we consider similarity as a feature to be included in identifying a review as a spoiler? If we find the answer to this question, we can over-sample our spoilers' dataset as it is imbalanced. To verify this, we can perform Hypothesis testing.

To verify whether similarity score can be considered as a feature to identify whether a review is a spoiler or not, we have considered the null hypothesis (h0) as there is no significant difference between the text similarity scores of reviews that contain spoilers and those that do not and alternative hypothesis (Ha) as there is a significant difference between the text similarity scores of reviews that contain spoilers and those that do not for an alpha value of 0.05.

We have considered the Mann-Whitney U Test to perform this hypothesis testing, and after performing the test, the p-value was 0.0, which is less than 0.05; hence, we can reject the null hypothesis. This means that the distribution of similarity scores between the spoilers and non-spoilers' datasets is different, and this feature can be used to identify whether a review is a spoiler.

From a user and model point of view, the model needs to identify non-spoilers as spoilers than classify spoilers as non-spoilers. If a review is classified as a non-spoiler but contains a spoiler (false, negative) is a bigger problem than classifying a review as a spoiler even if it is not a spoiler (false positive). So, we will set a threshold on similarity score beyond which, if there are any reviews, we will classify them as spoilers as that is the priority of this model. Now what is the threshold value? We have decided to consider thirty as the threshold value. There are two reasons:

1. a score of 30 represents the third quartile in the non-spoiler dataset. This means 75% of data in non-spoiler has a similarity value less than 30, which makes sense as less value means less spoiler content. This constitutes 4405 data points from a total of 16543.
2. The median value of Doc similarity in the spoiler's dataset is also around thirty. Why spoiler's dataset? Because of this information, on average, if a review is a spoiler, it has a similarity score of around thirty.

Now to prove our assumption, we have conducted another hypothesis test. Here, the null hypothesis (H0) would be that there is no relationship between the similarity score of a movie review and whether it contains spoilers or not, and the alternative hypothesis (Ha) would be that there is a relationship, and specifically, that a similarity score of more than 30 indicates that the review is likely to contain spoilers, Alpha: 0.05

To perform this test, we used the chi-squared test, and after performing the test, the p-value was 0.0, which means we can reject the null hypothesis. This way, we could up-sample our spoiler dataset by converting some of the non-spoiler's data points as spoilers beyond the threshold we set. The final spoiler's data set had around 10216 records, and the non-spoiler data set had 12663 records.

Model Building:

Firstly, Logistic Regression and SGD Classifier algorithms are implemented using the Tfidf and Count Vectorizer. The evaluation metric is the Recall and False Negative Rate (FNR). Here are the outputs of the algorithms.

Training Data:

Model	Count vectorizer		Tfidf Vectorizer	
	Recall	FNR	Recall	FNR
Logistic Regression	56%	28.8%	61%	27.5%
SGD Classifier	53%	27.2%	62%	27%
Random Forest Classifier	58%	31%	62.9%	27.5%

Testing Data:

Model	Count vectorizer		Tfidf Vectorizer	
	Recall	FNR	Recall	FNR
Logistic Regression	60%	31.8%	63%	29%
SGD Classifier	57%	34%	63%	29%
Random Forest Classifier	58%	33%	65%	28%

After implementing the traditional model, we tried exploring deep learning neural networks, starting with the Artificial Neural network model with three fully connected layers and 512 neurons in each layer. After performing 20 epochs, the following are the results:

Loss	Train Accuracy	Test Loss	Test accuracy
0.5709	0.7034	0.5854	0.7063

We also implemented **LSTM:**

Loss	Train Accuracy	Test Loss	Test accuracy
0.3742	0.8373	0.6962	0.6732

The LSTM model was completely overfitting. The DNN model has given a good result and the model is also stable hence we have decided to go ahead with this model's implementation for our final output as well.

Saving the Model

Since our model is now ready to predict whether a movie review is a spoiler, we want to integrate the model with the front end by developing it into a complete web extension. The spoilers get blurred out upon loading the extension on a movie review website. To connect the model with UI, we should first save our model into a pickle file or h5 file. We can save by loading the Python pickle package. Once the model was, we developed a function to receive movie reviews from the flask and return outputs, whether it is a spoiler or not for a given review.

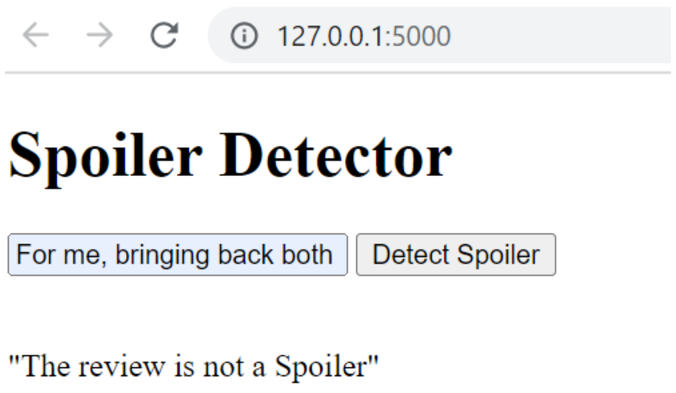
Model Integration with Flask

Python-based Flask is a popular microweb framework for creating web applications. Its versatility enables developers to select and utilize only the libraries and tools required for their projects, making it simple to customize and optimize. Flask also features a built-in support for unit testing, debugging, and routing and an intuitive and straightforward API that makes it simple to learn and use. Additionally, Flask is effective and quick because of its lightweight architecture and few dependencies, which let developers construct scalable, lightweight apps that are simple to deploy to various hosting settings.

To develop our model into a flask application, we started a flask instance and defined a function to predict the output in which the saved model is incorporated. We call the prediction function using Flask app routes which will oversee all the requests for our model.

```
@app.route('/predict',methods=['POST'])
```

As part of the application, we developed an index.html page to take the reviews of a movie as input and call the prediction function upon submission using the routes given. To oversee the input data, we extracted the reviews from the request object using the flask request module, which we imported in the beginning. Since our model takes vectorized inputs, we are converting the given review inputs into numerical vectors by passing the inputs to a clean function which we defined to convert text into numeric vectors. Once the /predict route is called with the review inputs, the model is loaded, takes the numeric vectors as input, and predicts whether the review is a spoiler or not by giving 0 and 1 as output.



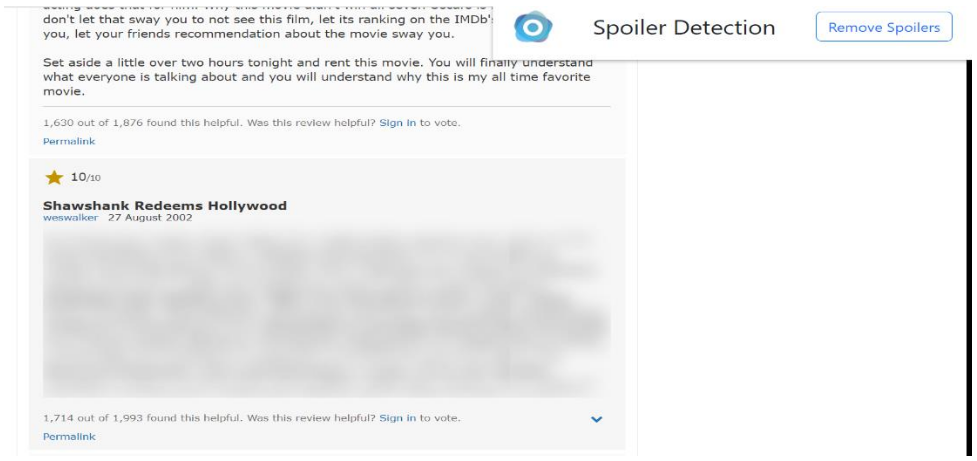
Spoiler Detector

"The review is not a Spoiler"

Predicting whether a review is spoiler or not.

Flask connection with a Web-Extension:

Developing into an extension eliminates giving review inputs manually. The critical task here is to integrate the extension with Flask Application. The extension requires a manifest.json file consisting of the extension's metadata like properties, version, and permissions. As part of the Chrome extension, we designed an index.html page with a toggle button upon enabling the extension to activate on the IMDB movie website. Once the switch enables, it scrapes all the reviews from the IMDB page through the event listener for which we have implemented code in script.js in JavaScript. We are going with only the IMDB website as the backend structure is different for different websites, and it is time-consuming to develop an extension that can manage all the structures. Once the reviews are extracted, we call to detect and blur function in which the API for the flask application is configured and pass the reviews to the application through POST methods which the flask app will manage. Once the reviews are sent to the model through Flask, and the model predicts the output as a spoiler or not, the response is returned to the function and blurs all the reviews on the page if they are returned as spoilers.



don't let that sway you to not see this film, let its ranking on the IMDb'
you, let your friends recommendation about the movie sway you.

Set aside a little over two hours tonight and rent this movie. You will finally understand what everyone is talking about and you will understand why this is my all time favorite movie.

1,630 out of 1,876 found this helpful. Was this review helpful? Sign in to vote.
Permalink

10/10

Shawshank Redeems Hollywood
weswalker 27 August 2002

1,714 out of 1,993 found this helpful. Was this review helpful? Sign in to vote.
Permalink

Blurring the Spoilers

Docker Deployment:

An open-source technology called Docker enables programmers to create, package, and distribute programs in a containerized setting. Python web developers often use Flask to develop elegant and scalable web apps. By integrating Docker with Flask, developers can deploy scalable, practical applications to several hosting environments, including cloud platforms like Microsoft Azure, Amazon Web Services, and Google Cloud. Recurrent neural network (RNN) model deployment for applications like language modeling, speech recognition, and picture captioning is one practical usage of Docker with Flask. RNNs are a particular class of neural networks that excel at handling sequential input and have been used in several speeches and natural language processing applications.

In conclusion, Docker and Flask offer a robust framework for creating and deploying web apps, including ones that integrate RNN models for tasks like speech recognition and natural language processing. By combining these tools, developers may build effective, scalable, and simple-to-deploy apps that can operate in various hosting situations.

For the deployment of our application, we considered initially dockerize the flask-based app using Docker and then hosting our application on Azure cloud. The main reason we considered Docker is to simplify the deployment process. It helps to develop, package, ship, and run using containers. It helps to package the application along with all the dependencies and libraries and reduces the likelihood of compatibility issues. It also helps in increasing the scalability and reliability of our application.

Steps Followed:

1. Created a docker file with the project contents and what requirements to be installed.
2. Created a .yml file to define the behavior and configuration of the container and what docker file should be used.
3. Once we have both files, we build the docker image using the below command.
4. After the image build, we start the container using the docker-compose-up command.

Microsoft Azure Hosting:

Virtual machines (VMs), a container registry, and web apps for developers are all available on the cloud computing platform Azure. Azure virtual machines offer a versatile platform for running cloud-based applications, while Container Registry offers a safe approach to controlling Docker images. These technologies may be used with Azure Web Apps to enable developers to build a scalable platform for hosting web apps in a containerized environment. A Docker image of the web application must be created by developers and stored in Container Registry to deploy it using Azure VMs and Container Registry. They may then deploy their application using the Docker image kept in Container Registry by building a virtual machine that runs Docker using Azure VMs. In order to make their containerized web application easily accessible to consumers, developers may now leverage Azure Web Apps to deploy it.

Once the container was up and running in Docker, we deployed the image to the Azure cloud and made the application available from the server end. To make this process work, we implemented the following steps. Create the Azure portal login and configure a virtual machine on Azure by following the required steps. Since we wanted to deploy our image on the virtual machine and make the application available on the server, we followed some pre-required steps.

Initially, we uploaded our entire Flask application to GitHub, and then we configured a connection to a virtual machine on Cloud through WSL. Once the connection is established, the image is transferred through the git clone from the Linux terminal to the Virtual machine. Once the Docker is deployed and the container is started, it installs all the required dependencies and libraries like the local environment. It hosts the application on a public Ip through which we can access our application.

Furthermore, for our final connection with the web extension, we configured the URL provided by our Azure platform in the extension through which we can apply the changes on movie review websites from the server-side model.

Conclusion and Future Work:

To conclude, we have created a google chrome web extension that can identify and blur the spoilers from movie review websites. We have built our own Neural Network models trained and tested with the review data we scrapped from multiple online resources. Furthermore, we have done Flask integration, Docker Deployment, and Azure Cloud Hosting with the model. We are limited to working on the Chrome browser and the IMDB website for this research.

Future works for this project can include expanding the scope of the extension to support additional movie-related websites and developing more sophisticated spoiler detection algorithms. Another potential direction for future work is to integrate natural language processing techniques to further enhance the accuracy of spoiler detection. Additionally, it may be beneficial to explore alternative platforms for hosting and deploying the application to provide more flexibility to users. Overall, the project has laid a solid foundation for further research and development in the field of spoiler-blocking tools.

References:

[1]. “Spoilers Detection in Online Reviews using Deep Learning" by Pedro Saleiro, Alberto Abad, and Luis Sarmento. This paper proposes a deep learning-based approach for detecting spoilers in movie reviews on social media platforms.

[2]. "Detecting Spoilers in Online Reviews using Machine Learning Techniques" by Sumit Pandey, Sarthak Bhatia, and Anubhav Maurya. This paper presents a machine learning-based method for identifying spoilers in movie reviews on e-commerce websites.

[3]. "Detecting Spoilers in Reviews using Natural Language Processing Techniques" by V. S. S. Bharadwaj, G. Swaroop, and R. V. S. R. Kumar. This paper presents a natural language processing-based approach for identifying spoilers in movie reviews on e-commerce websites.

[4]. Mirza, H., Ahmed, F., & Khan, M. A. (2020). An Empirical Evaluation of Flask, Django and Pyramid for Dockerized Python Web Application Development. International Journal of Advanced Computer Science and Applications, 11(4), 136-142. https://doi.org/10.14569/IJACSA.2020.0110417

[5]. Microsoft. (n.d.). Flask web app with Azure Cosmos DB for MongoDB API. Azure Documentation. Retrieved from https://docs.microsoft.com/en-us/azure/cosmos-db/create-mongodb-flask

[6]. Gers, F. A., & Schmidhuber, J. (2001). LSTM recurrent networks learn simple context-free and context-sensitive languages. IEEE Transactions on Neural Networks, 12(6), 1333-1340. https://doi.org/10.1109/72.963769

[7]. Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. Neural computation, 12(10), 2451-2471. https://doi.org/10.1162/089976600300015015

[8]. Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2017). LSTM: A search space odyssey. IEEE transactions on neural networks and learning systems, 28(10), 2222-2232. https://doi.org/10.1109/TNNLS.2016.2582924

[9]. Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., ... & Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144. https://arxiv.org/abs/1609.08144