Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

MASTERTHESIS

Using Neural Language Models to Detect Spoilers

vorgelegt von

Hans Ole Hatzel

MIN-Fakultät

Fachbereich Informatik

Language Technology Group

Studiengang: Master Informatik

Matrikelnummer: 641655

Abgabedatum: 14.2.2020

Erstgutachter: Prof. Dr. Chris Biemann

Zweitgutachter: Dr. Seid Muhie Yimam

Betreuer: Benjamin Milde, Steffen Remus

Abstract

Social media users enjoy discussing crucial plot developments in recently released media, often to the dismay of those users who have yet to watch or read the TV series, book, or movie in question. Users may have their enjoyment spoiled by discussions revealing major turns in the plot. An automated approach, filtering out such spoilers, would be ideal as manual labeling is impossible due to the sheer amount of content. Filtering would allow interested parties to partake in the discussion while leaving others the option to stay uninformed. In this thesis, we first approach the task of identifying spoilers as a classification task on the sentence or paragraph level. Second, we conduct the more challenging task of sequence classification, where each token is classified based on whether it constitutes (part of) a spoiler.

Our approach makes use of machine learning methods based on neural networks, relying on user-generated content as training data. More specifically, BERT, a variant of the Transformer architecture, is used, since BERT has recently shown promising results on various language processing tasks.

The results show that while our approach outperforms many previous ones, it is still not viable for real-world use. In real-world data, we observe an extreme class imbalance. As a result, all of our models yielding sufficient recall also produce a very high number of false positives. In other words, if enough of the actual spoilers are recognized as such, an unacceptable amount of non-spoilers are also incorrectly marked as spoilers.

Contents

1. Introduction **1**

1.1. Motivation . . . 2
1.2. Definition of Spoilers . . . 2
1.3. Reddit . . . 2
1.4. Research Questions . . . 4
1.4.1. Question 1: Document Classification . . . 4
1.4.2. Question 2: Sequence Labeling . . . 4
1.4.3. Question 3: Story Document . . . 5
1.4.4. Question 4: Analyzing Models . . . 5

2. Background **7**

2.1. Neural Networks . . . 7
2.1.1. Multilayer Perceptrons . . . 8
2.1.2. Activation Functions . . . 9
2.1.3. Backpropagation and Stochastic Gradient Descent . . . 9
2.1.4. Learning Rate Schedules . . . 10
2.1.5. Early Stopping . . . 10
2.1.6. Binary Cross-Entropy Loss . . . 10
2.1.7. RNN . . . 11
2.2. Transformer Architecture . . . 13
2.2.1. Attention . . . 13
2.2.2. BERT . . . 14
2.3. NCRF++ . . . 16
2.4. Multivariate Naïve Bayes . . . 17
2.5. Evaluation Metrics . . . 18
2.5.1. Accuracy, Recall, Precision, and F_1 Score . . . 18
2.5.2. WindowDiff and WinPR . . . 19
2.6. Information Gain . . . 21

3. Related Work **23**

3.1. TV Tropes Dataset . . . 23
3.2. Using Story Documents . . . 24
3.3. Feature Engineering for Spoiler Detection . . . 25
3.4. Neural Approaches . . . 26

4. Dataset **27**
4.1. Spoilers On Reddit . . . 27
4.1.1. Data Exploration . . . 28
4.1.2. Subreddit-Specific Rules . . . 30
4.1.3. Spoiler Annotations . . . 30
4.2. Building The New Dataset . . . 31
4.2.1. Subreddit Whitelisting . . . 33
4.2.2. Comment Filtering . . . 34
4.2.3. Extracting Negative Samples . . . 35
4.2.4. Dataset Format . . . 36
4.3. Summary and Final Dataset . . . 36

5. Implementation **37**
5.1. Software Toolkit . . . 37
5.2. Working with the Pushshift Dataset . . . 37
5.3. Machine Learning Models . . . 38

6. Methodology **41**
6.1. Document Classification . . . 41
6.1.1. Baseline Classification Model . . . 41
6.1.2. BERT Classification Model . . . 41
6.2. Token Model . . . 45
6.2.1. NCRF++-based Model . . . 45
6.2.2. BERT Sequence Model . . . 45
6.3. BERT Story-Document-Supported Model . . . 45
6.4. Extracting and Visualizing Attention . . . 47

7. Evaluation **49**
7.1. Evaluation Metrics . . . 49
7.1.1. Token Model . . . 49
7.2. TV Tropes Dataset . . . 50
7.2.1. Story Document Supported Model . . . 51
7.3. Reddit Dataset . . . 52
7.3.1. Classification Model . . . 52
7.3.2. Sequence Model . . . 54
7.4. Discussion . . . 55
7.4.1. Real-World Application . . . 55
7.4.2. Overperformance of our baseline model . . . 56
7.4.3. Comparative Difficulty of the Task . . . 56

8. Conclusion **59**
8.1. Future Work 60
8.1.1. Potential Model Improvements 60

Appendices **69**
A. Reddit's Inline Spoiler Annotations 69
B. Whitelisted Subreddits 69

1. Introduction

Information giving away a major turn in a plot or an important event can be considered a so-called spoiler (see Section 1.2 for a more concise definition). Many people appear to perceive spoilers as lessening their enjoyment of a TV series, movie, book, or even sports broadcast rerun. The definition of what exactly constitutes a spoiler is fairly subjective. It is unclear which piece of information is central enough to be considered a spoiler. Detecting spoilers is a problem that has previously been tackled using natural language processing methods (Guo and Ramakrishnan, 2010; Boyd-Graber et al., 2013; Jeon et al., 2013; Iwai et al., 2014; Maeda et al., 2016). Some online platforms offer the option to manually annotate spoilers so that users can be warned before viewing them.

For example, the second sentence in the Reddit[1] comment in Figure 1.1, regarding a specific movie, is marked as a spoiler.

> Deus Ex Machina, I was totally taken in by the machine like the protagonist was. **In the end, when she left him to die trapped in the house it really stuck with me, because like him I thought she wanted to be with him instead she was using him.**
>
> https://old.reddit.com/r/AskReddit/comments/7vf2ng/what_movie_was_so_disturbing_it_left_you_feeling/dts89dm/
> [Retrieved: 20-02-12]

Figure 1.1.: In this comment containing a spoiler for the movie "Ex Machina", the user-generated spoiler annotation is visualized using bold font.

As spoiler annotations are not universally supported, and necessarily require manual effort, an automatic approach would be preferable. The task of detecting spoilers is nontrivial. For example, the information of a character dying could be part of the premise of a story and, therefore, not a spoiler, but may also constitute a major turn in the plot. At the same time, it seems likely that a simple heuristic, e.g., one looking for the word 'killed,' would have some degree of success at identifying spoilers.

We will use pre-trained language models, fine-tuning them using user-generated annotations to predict which documents contain spoilers. On the same data, we will also build a model that predicts the presence of spoilers on a token level, thereby marking sections of sentences as containing spoilers.

[1] `https://reddit.com`

1.1. Motivation

For years now, Internet users have been discussing TV shows live, online, as they air (Harrington et al., 2013). With the recent prevalence of nonlinear TV consumption, as found in Video-on-demand services (Abreu et al., 2017), users may choose to watch TV shows well after their original air date. They then have every opportunity to be presented with information that spoils the plot for them. This risk also applies to other media like books and movies. The results on whether being spoiled impacts people's enjoyment are unclear (Johnson and Rosenbaum, 2018), yet the prevalence of measures like spoiler tags shows that many users care about avoiding spoilers. The implementation of automatic filtering could provide an easy way to enable users to enjoy online discussions instead of having to stay away from them until they have caught up with the latest TV episodes or most recent movie releases. Work on the task of spoiler detection has the potential to yield advancements for other language processing tasks. Previous work has, for example, suggested that objectivity is a good indication of spoilers (Jeon et al., 2013), meaning posts containing more words indicative of emotions (i.e., those that are more subjective) are less likely to contain spoilers. The feature of objectivity has also been used in other tasks such as the detection of hate speech (Gitari et al., 2015). While deep learning approaches, like the one we take, do not require feature engineering (Goodfellow et al., 2016, p. 3), it is clear that knowledge from this task could still be transferred to other tasks.

1.2. Definition of Spoilers

In their paper specific to television spoilers, Jeon et al. (2016) define a spoiler as "any kind of information that affects or spoils people's enjoyment of a TV program by revealing unknown facts to them." They provide examples of what kind of information might be considered a spoiler: "crucial events, important reversals of fortune, the final denouement of a suspense drama or movie, the end results of reality TV shows or sports broadcasts including the identities of the winner and loser of a contest or the final score of a match."

Ultimately, a precise definition of what does and does not constitute a spoiler is not required for our purposes. Central to the application of machine learning is only that *some* shared understanding of spoilers exists between annotators. We assume that a sufficient agreement on the definition of spoilers exists to enable detection using machine learning techniques. The degree to which a shared understanding of spoilers exists implies an upper bound for how well an automated system could perform.

1.3. Reddit

Reddit is a "social news aggregation, web content rating, and discussion website."[2] The platform offers users the capability to discuss a wide range of topics. It is structured into so-called *subreddits*, communities that are each focused on a specific topic.

[2] `https://en.wikipedia.org/w/index.php?title=Reddit&oldid=936603154`

A user can submit a *post* (sometimes called a *submission*) to a specific *subreddit*. They[3] can either submit a link (e.g., to another website or an image) or a text. In either case, they also provide a title for their post. Users can start discussing a post by submitting a *comment*. Comments are associated with the post to which they are submitted in response. Comments can also be posted in direct response to other comments, such that a tree of comments is formed. Ultimately, a post has a set of comments associated with it, with each comment having associated child comments. In this way, the tree of comments can, in principle, reach an arbitrary depth. Figure 1.2 shows a Reddit comment tree, including individual comments containing spoilers.

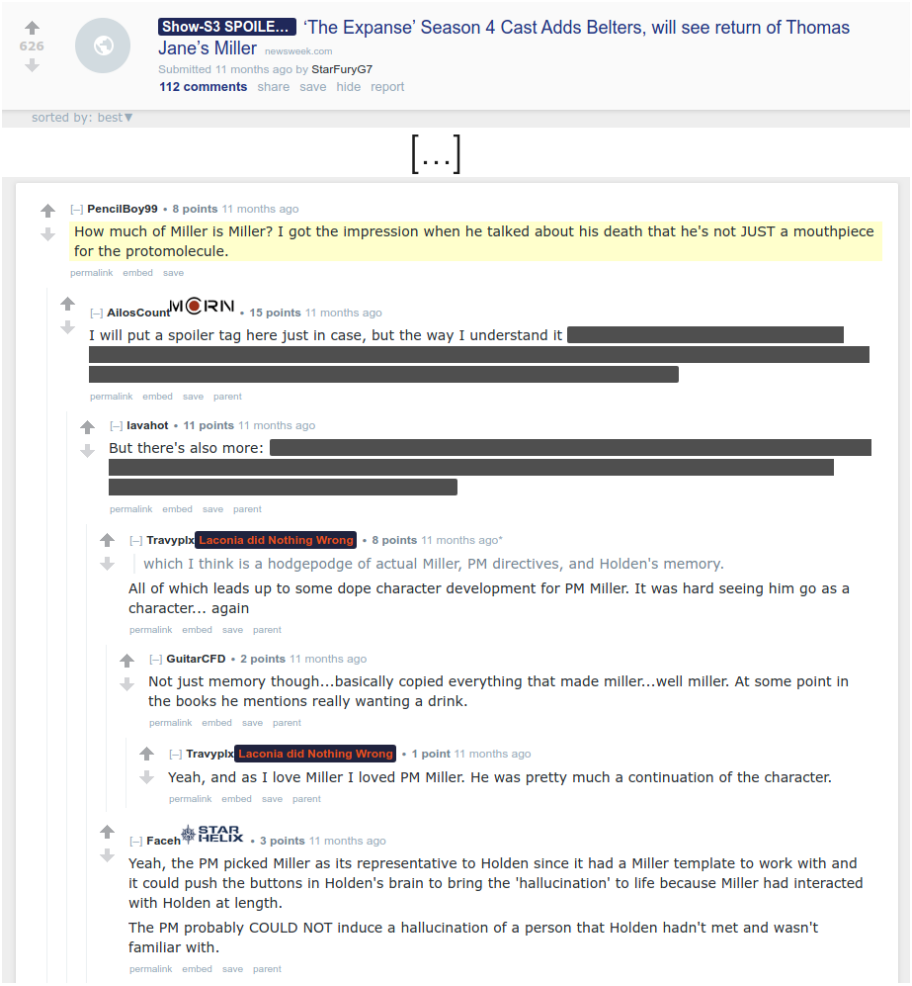


Figure 1.2.: This Reddit post[4] links to a news article. Comments, arranged in a tree structure, discuss the article while spoilers are redacted (following manual annotations by the individual users) and only revealed on hovering over them. The ellipsis signifies an omission made by us.

[3] Throughout this thesis, we will use "they" as a gender-neutral pronoun for singular entities in addition to its plural usage.

[4] `https://old.reddit.com/r/TheExpanse/comments/avd3ud/the_expanse_season_4_cast_adds_belters_will_see/ehed83t/` [Retrieved: 20-02-07]

1.4. Research Questions

We will investigate the viability of neural methods for the detection of spoilers in user-generated content. Specifically, we explore these four research questions, which are explained in more detail in the next sections:

1. How can neural methods be used to classify comments as spoilers effectively?
2. How can sequence tagging approaches be applied to spoiler classification in order to find spans of spoiler text within comments?
3. How can additional data, specifically story documents, be used to aid spoiler classification?
4. What can we learn about the models by inspecting the inner workings of the neural networks we used?

1.4.1. Question 1: Document Classification

Question 1 involves investigating whether or not neural methods are suitable to detect documents containing spoilers (i.e., classifying documents into two classes, based on whether or not they contain spoilers). A document, in our case an entire Reddit comment, is classified as containing a spoiler even if only a small part of it contains the relevant information. Exploring the question involves building a neural classification model and comparing its results with those achieved by other researchers, as well as evaluating the model's performance against the requirements for real-world usage. We will use a Transformer architecture for this task, more specifically the BERT architecture (Devlin et al., 2019). We assume that previous models can be outperformed using our approach because Transformers have improved results on many downstream language tasks.

1.4.2. Question 2: Sequence Labeling

Question 2 builds on Question 1, but instead of providing a document level classification, this task aims to make a distinction for each token. In this manner, a comment can be considered only to partially be a spoiler. In the sentence, "We went to the cinema and loved the experience, **a pity that Snape was killed.**" only the bold section might be considered a spoiler, whereas the rest of the sentence would not. Take this real example by Reddit user cara123456789: "Lets just say there were also some scenes involving or i thought might show a certain character **infantata** that really scared me.".[5] The user tries to share their sentiment towards how scary a particular episode of a TV is while redacting critical information they perceive to be a spoiler. In this example, the annotation only covers a specific character's name ("infantata"). We will use the same underlying BERT model architecture as in Question 1.

[5] `https://www.reddit.com/r/AmericanHorrorStory/comments/2o8l9h/how_scary_is_murder_house/cmkv52g/` [Retrieved: 20-02-12]

1.4.3. Question 3: Story Document

Question 3 is asking whether it is possible to beneficially employ background information on the storyline in question, for spoiler detection. Knowing the work which a potential spoiler relates to would enable the system to look for information on the work that could aid in identifying information that constitutes a spoiler. In our approach, we assume that a spoiler's background information is given in the form of a natural language story summary.

Intuitively, the idea is to cast the question of "Is this a spoiler for this specific plot?" as an information retrieval task, reducing it to checking if a piece of information is part of the plot summary. Our model then operates on two texts instead of one, evaluating not only the potential spoiler individually but also in conjunction with the summary. We expect that this approach can slightly improve upon the results of classification without additional inputs as motivated by Question 1. This expectation is caused by the previous success of incorporating structured meta-information into spoiler detection tasks (see Chapter 3) and the fact that story documents have been shown to contain information relevant to spoilers (see Section 3.2).

1.4.4. Question 4: Analyzing Models

Question 4 involves visualizing and inspecting the internal parameters of the networks built for Questions 1, and 2. Specifically, attention mechanisms in modern neural architectures have the potential to make the network interpretable. In this manner, we investigate the potential to identify specific features that are important to our classification results. We suspect that specific marker words will play a crucial role in the classification of spoilers. This expectation is based on the success of previous methods using specific words as features (see Section 3.3).

2. Background

This chapter provides background information required for understanding the techniques used in later chapters. Specifically, we will discuss the Transformer architecture (Vaswani et al., 2017) and BERT, an adaption of the architecture (Devlin et al., 2019). We will also provide background information on how the Transformer approach differs from more traditional recurrent neural network approaches like LSTMs (Hochreiter and Schmidhuber, 1997). Apart from neural methods, we will also describe the fundamentals of Naïve Bayes, which we will use in our baseline model. Traditional metrics for evaluation of machine learning systems are discussed as well as less widely adopted, more task-specific metrics.

2.1. Neural Networks

This section will discuss some specific methods used in neural-network-based machine learning models. Most concepts are only briefly touched upon, giving the necessary context for the rest of this thesis.

In principle, neural networks are used to approximate functions; they map a vector of real numbers to an output vector of real numbers (Goodfellow et al., 2016, p. 205). Neural networks are said to have layers. The input layer accepts multiple input values (i.e., a single vector), transforming them to fit the first hidden layer's size. Subsequent hidden layers transform the input further until the final transformation is performed by the output layer, resulting in the desired number of output values.

For some domains, say for predicting price developments, real-valued inputs can be a natural fit. This does not apply to the domain of text-processing, one way of transforming text into a real-valued representation is using one-hot encoding (Goodfellow et al., 2016, p. 523) on a word-by-word basis. This scheme represents any word in a given vocabulary using a vector the size of the vocabulary, with all but one value being zero and a single non-zero value (typically 1) at the position corresponding to the given word. In the case of classification tasks, output values are interpreted as classes by comparing them to decision boundaries or building pseudo probabilities of class memberships (e.g., using the softmax function, Goodfellow et al. 2016, p. 179). When considering a one-hot word encoding scheme, it is not apparent how a sentence with an arbitrary number of tokens would be encoded, as a simple neural network only accepts one fixed-size input. This problem is addressed in different ways by recurrent neural networks (see Section 2.1.7) and Transformers (see Section 2.2).

Most classical machine learning approaches require feature engineering beyond an encoding scheme like one-hot encoding. Machine learning engineers have to select suitable features

derived from the raw data to serve as the model input. Such features have to be salient in that they contain information relevant for modeling the specific function. Neural networks are used in deep learning approaches, which alleviate the need for explicit feature engineering. Neural networks can, while allowing for feature engineering approaches, also learn feature representations on their own, which not only has the potential to save engineering efforts but can also yield improved results (Goodfellow et al., 2016, p. 3).

In principle, a set of parameters could be handcrafted for a neural network to be able to approximate a specific function. However, to use neural networks as a machine learning method, the parameter choice is handled by a training process. In the training process, relying on a loss function (see Section 2.1.6) that describes the deviation from the desired result, the parameters are gradually adjusted (see Section 2.1.3). Neural networks are, in the simple case, trained in a supervised manner, where the network's parameters are adjusted to match the known, desired output for a given input. The network can then be applied to input data for which the desired output is unknown and, given the training set was representative of the unseen data, produce a correct result. For a more detailed explanation of the training process, see Section 2.1.3.

Neural networks form the basis of the more specific deep neural architectures (Transformers, specifically BERT) that we will be using in our spoiler detection tasks. For a more in-depth introduction to neural networks, see Goodfellow et al. (2016) or Rojas (1996).

2.1.1. Multilayer Perceptrons

Multilayer perceptrons (MLPs) are a basic form of neural network. Based on an input vector x, they produce an output vector y, also known as the hypothesis. Specifically, the input is transformed using bias vectors b, weight matrices w, and an activation function g. The transformation can be described as the function $f(x; w, b) = g(x \cdot w + b)$ (Goodfellow et al., 2016, p. 167). A sequence of such transformations with different weight matrices and bias vectors is called an MLP. Each application of the function represents a layer in the neural network; each layer can use different matrices with different sizes. The weight and bias matrix dimensions define the hidden, input, and output sizes of the network.

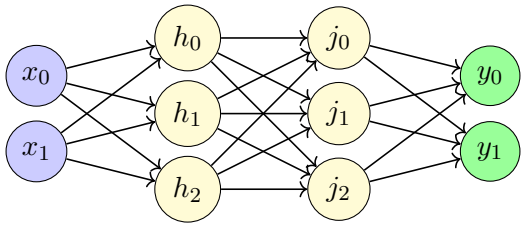


Figure 2.1.: Schematically, the MLP can be visualized by representing each vector element as one node. The input layer is followed (from left to right) by two hidden layers (h and j) and one output layer.

Figure 2.1 shows a multilayer perceptron with two hidden layers. The input and output vectors each have a length of two, with both hidden layers having a size of three. The arrows shown in the diagram are described and weighted using the matrices referred to as w above.

With the usage of non-linear activation functions, such as the sigmoid function, MLPs, given sufficient size, can approximate all (Borel) measurable functions. This capability prompts their designation as a "universal approximator"(Hornik et al., 1989).

2.1.2. Activation Functions

An activation function $g(x)$ is used to introduce non-linear properties into neural networks, as mentioned in Section 2.1.1. A variety of different activation functions have been explored, with the sigmoid function σ traditionally being a popular choice. The Rectified Linear Unit (ReLU) is perhaps the most popular in modern deep learning approaches. Ramachandran et al. (2017) performed automated searches and found their "swish" function to perform the best. They also analyzed the Gaussian Error Linear Unit (GELU) and found it to perform well. The GELU activation function has been used by Devlin et al. (2019) and Radford et al. (2018), among others.

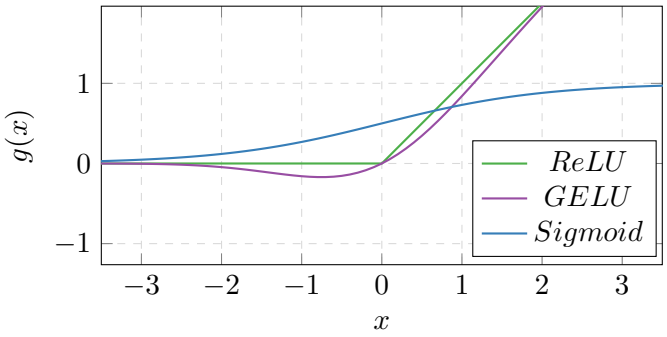


Figure 2.2.: The ReLU function is a common activation function (Goodfellow et al., 2016, p.169) whereas GELU has only recently seen usage, for example, by Devlin et al. (2019) and Radford et al. (2018)

2.1.3. Backpropagation and Stochastic Gradient Descent

Given the previously introduced elements of an MLP, an initialized network can already perform classification and regression tasks. In order to use it as a machine learning method, however, a technique to determine a suitable set of weights or matrices is required. Stochastic Gradient Descent (SGD) is a technique to do just this. It describes the iterative alteration of the weights (in this case, inside a neural network), which results in iteratively improving approximations of the desired function.

Based on a differentiable loss function (indicating the divergence from the desired result), the weights of the network are adjusted in accordance with the gradient of the loss function (i.e., such that the loss is reduced). As a result, the weights are updated to produce output that is closer to the desired result. For an example of a loss function, see Section 2.1.6.

The analytical calculation of the gradient, often across multiple network layers, is called backpropagation (Rumelhart et al., 1986). In backpropagation, the (calculus) chain rule is

used to calculate the loss function's gradient with regard to each parameter in the network (Goodfellow et al., 2016, p. 198).

SGD adjusts weights based on the gradient and a meta parameter ϵ, the learning rate. The learning rate scales the amount by which weights are updated. As a result, too small values for ϵ lead to slow convergence, whereas too large values can prevent convergence altogether (Goodfellow et al., 2016, p. 286).

2.1.4. Learning Rate Schedules

Learning rate schedules are a way of dynamically changing the learning rate during training. Some optimization algorithms, like Adam, already perform learning rate scaling of sorts by keeping momentum on a per parameter basis (Goodfellow et al., 2016, p. 301). Heuristics that adapt the learning rates have been shown to result in faster convergence (Jacobs, 1988). Learning rate schedules have found application in stabilizing the learning process with large batch sizes (Goyal et al., 2017).

2.1.5. Early Stopping

Early stopping is the practice of terminating the training of a machine learning model early, usually in an effort to prevent overfitting (Prechelt, 2012). It is a regularization method, in that overfitting is prevented by checking the error or loss on a validation set. Once overfitting on the training data begins, the validation loss is no longer improving and will start to increase.

Prechelt (2012) analyzed different criteria for when to perform early stopping. He found the criterion of stopping after a certain number of epochs with no improvement in validation loss to have the best trade-off between the final result and training time. Specifically, stopping after 15, 20, or 30 epochs was found through experimentation to yield the best performance, with only multiples of 5 epochs being considered in the experiment. These early stopping criteria were applied by keeping weights (i.e., the network state) of previous epochs. After stopping, the state of the network at the best performing epoch is saved. The performance at each epoch is evaluated using the validation loss.

2.1.6. Binary Cross-Entropy Loss

Cross-Entropy loss is a loss function often used in the training of neural networks. Loss functions guide the training of neural networks by indicating the divergence from the desired output. Weights are adjusted, during training, to minimize the loss function. Its binary variant is used to train binary classifiers that have a single output neuron.

$$BCE(\hat{y}, y) = -(y \cdot log(\hat{y}) + (1 - y) \cdot (log(1 - \hat{y}))) \tag{2.1}$$

Where y is the class, and $\hat{y}$ is the predicted value, as output by the model. As illustrated by Figure 2.3[1], the loss decreases as the correct class (in this case 1) is approached by the model's

[1] `https://pytorch.org/docs/stable/nn.html#torch.nn.BCELoss` [Retrieved 20-02-12]

output. Similarly, the loss decreases as the prediction approaches 0 when the target class is 0.

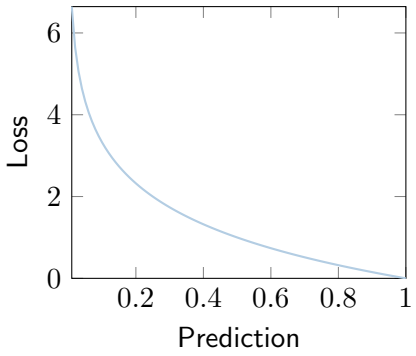


Figure 2.3.: The BCE-Loss for different network outputs, given the target class 1, decreases as the output approaches 1.

2.1.7. RNN

Recurrent neural networks (RNN) are an extension of MLPs that can operate on sequences of arbitrary length. Such networks, generally, operate on the current input and a hidden state, producing an output and the next hidden state (Goodfellow et al., 2016, p. 363). The hidden state, depending on the architecture, can be the output or an internal set of weights of the previous iteration.

Consider a temporal sequence of inputs of arbitrary length. This sequence can not easily be operated on by a traditional MLP, as the matrix dimensions would have to be adjusted. In a recurrent network, in addition to the input x^t at timestep t, the output of the previous timestep y^{t-1} is used as an additional input. Alternatively, instead of the previous output, any part of the previous hidden state can be used as an input. The different inputs are combined by concatenating their vector representations.

This way, a sequence of arbitrary length can be used as an input, with an output value being generated at each step in time. Importantly, the weights at each timestep remain constant, allowing, in principle, for generalized learning across timesteps.

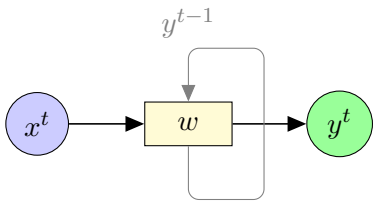


Figure 2.4.: A recurrent neural network includes the previous timestep's output in its calculations (graphic inspired by Christopher Olah[2]).

Figure 2.4 visualizes the idea of a recurrent network; the previous iterations' value in the network is involved in the next step's computation. Note that unlike Figure 2.1, we now

[2] http://colah.github.io/posts/2015-08-Understanding-LSTMs/ [Retrieved 20-01-19]

model an entire hidden layer using a box instead of multiple nodes, focusing on information transformation over network-state.

Training of recurrent networks can be performed using a variant of backpropagation (Rumelhart et al., 1986), called Backpropagation Through Time (BPTT) (Mozer, 1989). BPTT runs into the issue of vanishing gradients (Goodfellow et al., 2016, p .391). Since the same functions using the same parameters are applied iteratively, the gradient value can quickly vanish or explode. For this reason, training recurrent networks to model long term dependencies is often infeasible. Vaswani et al. (2017) set out to circumvent this problem, among others, by introducing the Transformer, which uses a fixed-sized architecture instead.

LSTM

Long Short-Term Memory (LSTM) units are an extension of RNNs, attempting to solve the vanishing gradient problem (Hochreiter and Schmidhuber, 1997). The approach taken is to hold an internal state that represents the information on previous elements in the sequence and to only change this state when required. This internal memory (or state) and its protection are modeled using multiplications with learned weights. That way, the entire network can be trained via BPTT like other RNN variants.

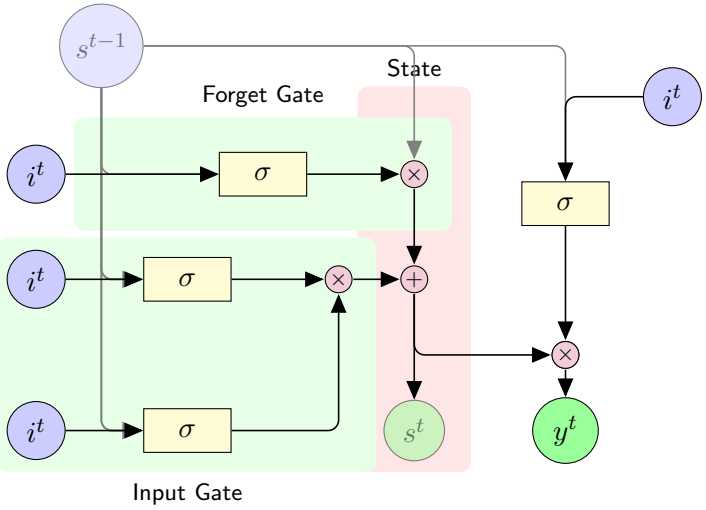


Figure 2.5.: The internal state of an LSTM is protected by the input gate (graphic inspired by Christopher Olah[3], showing the variant of LSTMs proposed by Gers and Schmidhuber 2001).

In Figure 2.5, yellow boxes indicate transformations using learned weight matrices with σ indicating a sigmoid activation function. i^t represents the input at timestep t, which is the concatenation of the input x^t and the previous hypothesis y^{t-1}. Red circles indicate transformations performed on the data, either addition ($+$) or multiplication ($\times$). Merging lines in the diagram represent concatenation operations.

The internal state s^t, can only be changed using the input and forget gates. Instead of using

[3] `http://colah.github.io/posts/2015-08-Understanding-LSTMs/` [Retrieved 20-01-19]

the previous output, as is the case in traditional RNNs, the state is used as an input to the next iteration of the network. Thereby the problem of vanishing and exploding gradients can be, at least partially, mitigated.

The particular variant of an LSTM shown here adds peephole connections, which is the inclusion of the state into the input and forget gates. Gated Recurrent Units (GRU), introduced by Cho et al. (2014), are an adaption of LSTMs that are simpler to implement and compute. They only have two gating units as opposed to four in the LSTM.

2.2. Transformer Architecture

The Transformer model architecture, introduced by Vaswani et al. (2017), is a non-recurrent approach to sequence modeling. RNNs have an inherently sequential property as their previous iteration provides part of their input. Due to the non-recurrent nature of Transformers, performance gains over recurrent models are possible through parallelization. Models with this architecture were able to achieve state-of-the-art results on machine translation tasks (Devlin et al., 2019). Transformers form the architectural basis for our spoiler classification models.

Generally, the Transformer architecture follows an encoder-decoder pattern; such an architecture involves creating a hidden state using one section of a neural network (the encoder) and creating an output from said hidden state (the decoder). This means that a sequence of layers transforms the input into a hidden state, from which another sequence of layers builds the output. The input and output comprise a fixed number of tokens, called the context size. A central component of the architecture is attention, an approach that allows the network to focus on relevant elements, instead of always considering the entire sequence.

Different adoptions of the Transformer architecture focus on the pre-training aspect, enabling fine-tuning of models with comparatively little resources (Radford et al., 2018; Devlin et al., 2019).

2.2.1. Attention

Transformers, as originally proposed by Vaswani et al. (2017), use self-attention in an attempt to model dependencies between different tokens. Attention can generally be described as the model focusing on specific aspects of the input instead of having to handle all information (Bahdanau et al., 2015).

The specific attention variant used is called scaled dot-product attention. The model uses this attention mechanism in three places: in each of the decoder layers to attend (i.e., pay attention to) to the final encoder state, in each of the encoder layers to attend to the previous layer's representation, and finally in each decoder layer in order to attend to the previous decoder layer's representation. The desire to decrease the length of the path information takes is listed as a motivation for the attention mechanism. In the case of a simple recurrent model, this path's length is bounded by $O(n)$, where n is the length of the input sequence. The self-attention mechanism reduces this to a low constant length, as each attention-head can

directly attend to any token's representation in the previous layer.

Scaled dot-product attention is an attention mechanism that relies on three inputs. The key (K) and the query (Q) are used to select important aspects of the value (V). As illustrated by Figure 2.6, key, and query are multiplied and scaled, with the optional mask being applied only in certain cases. Finally, a softmax transformation is performed. After said transformation, the attention score is multiplied with the value (V), scaling it in accordance with the attention scores. Consequently, the matrix after the softmax operation (but before the final multiplication) is a representation of which tokens are to be paid attention to. Visualizing this matrix allows some insight into which part of a sequence is considered in a given layer.

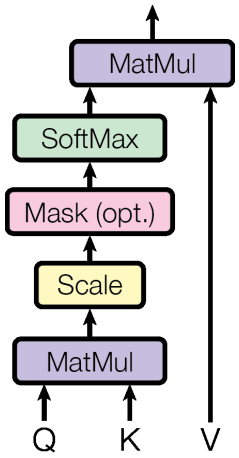


Figure 2.6.: Scaled dot-product attention is based on three inputs: (key) K, (query) Q, and (value) V (graphic is taken from Vaswani et al. 2017).

Specific knowledge of the attention mechanism is required for our model analysis in Section 6.4.

2.2.2. BERT

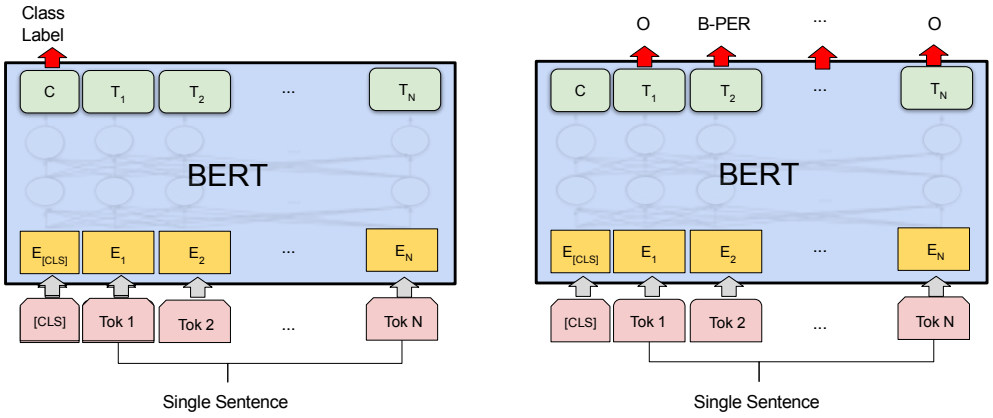
BERT (Bidirectional Encoder Representations from Transformers) is a language model pre-training approach and model architecture introduced by Devlin et al. (2019). It is based on the Transformer architecture but implements only the encoder aspect. The absence of a decoder means BERT is not capable of language generation, but classification tasks and extractive question answering can be performed by operating on the final encoder state.

Different pre-trained BERT models are available. The intended use of the pre-trained models is to fine-tune them with task and domain-specific training data, thereby adapting them for the task at hand. Such fine-tuning, when following the recommendation of the authors', is done in only 3 to 4 epochs across the training data. The BERT model, fine-tuned in this manner, was able to achieve state-of-the-art performance on multiple downstream language tasks (Devlin et al., 2019). BERT is pre-trained using two tasks:

- The first task is called masked language models, meaning the model is tasked to predict a set of words that are hidden from the input. Masked words are predicted using an output softmax layer on the masked token's internal representation.

- The second task is known as next sentence prediction. It involves presenting the network with a pair of sentences. The model then predicts if the two sentences are consecutive sentences in the training data.

Both approaches can be considered unsupervised learning in that the target label is only derived from the input data but not known ahead of time.



(a) Document classification is based on the internal representation corresponding to the `[CLS]` token.

(b) Each token is classified based on its internal representation

Figure 2.7.: BERT can be used for sequence and token classification. (graphic taken from Devlin et al. 2019)

Figure 2.7 schematically shows BERT as used for the classification of tokens or entire documents. Devlin et al. (2019), in this case, consider any text to be a sentence, meaning what is labeled as a sentence in the illustration can be an entire document. The encoder produces an internal representation of each token. A single linear layer can be used to transform this representation into a class, such that the model performs classification on a token basis. For the document classification, the final transformation step is performed on the representation `C`, which corresponds to a special token (referred to as the `[CLS]` token) that is inserted at the start of each sequence. The representation `C` is, in pre-training, only trained for next-sentence prediction. For that reason, it is not a meaningful representation of the whole document until fine-tuning is applied (Devlin et al., 2019).

Crucially, in fine-tuning, a final layer is trained to transform internal representations into a class, while the rest of the model is also updated using backpropagation. Earlier layers in the encoder are adapted to the task as a result of backpropagation affecting the whole network. `C` thereby can be trained to be a task-specific representation.

WordPiece

WordPiece, originally proposed by Schuster and Nakajima (2012), is the tokenization model used by BERT. The tokenization scheme aims to solve the problem of out-of-vocabulary words.

Instead of being unable to handle an unknown word, the model splits it up into known subsets, so-called WordPieces. In this way, "eating" might be represented as "eat" and "ing."

An algorithm iteratively builds a set of WordPieces. The approach taken is to iteratively introduce new WordPieces, starting from a set of all known characters. A new WordPiece is formed by combining those two existing pieces that are the most likely, according to a language model. No WordPieces are removed during the process. The merging of WordPieces is repeated until a pre-defined amount of WordPieces is found. The vocabulary size (i.e., after how many iterations to stop), therefore, is parametrizable. The initial set of pieces contains all characters that were encountered; out-of-vocabulary word pieces are only possible when unknown characters are encountered.

The process of tokenization into word pieces begins with simple whitespace separation. After the initial whitespace tokenization, further separation into WordPieces is required. Schuster and Nakajima (2012) use the language model they built to find the most likely segmentation. In BERT, however, a greedy solution is used instead. In said greedy approach, the longest WordPiece matching the start of the remaining input is used. Markers on the tokens are used to indicate whether they are separated by whitespace from the next token or are immediately followed/preceded by them. In BERT, an output for the input "unaffable" might look like this: `[‘‘un’’, ‘‘##aff", ‘‘##able’’]`. [4]

A similar algorithm to WordPiece is called Byte Pair Encoding or BPE. BPE, designed initially for compression, takes the more straightforward approach of combining the most frequent character bigram into a newly created character (Shibata et al., 1999). Consequently, no language model is involved in the creation of BPE based embeddings.

2.3. NCRF++

NCRF++ is a sequence modeling toolkit; it makes use of neural networks in addition to conditional random fields (Yang and Zhang, 2018). Conditional Random Fields (CRFs) are a discriminative model for labeling sequence data (Lafferty et al., 2001). In our sequence tagging approach, this toolkit will be used as our baseline model.

The underlying model architecture used by NCRF++ is shown in Figure 2.8. It consists of character-level RNNs followed by a word-level RNN. The final classification is performed using either a softmax layer or a CRF on the RNN's output. In the case of the CRF, the Viterbi algorithm (Forney, 1973) is used for finding the optimal output. While the word-level RNN relies on words being contained in the vocabulary, the character-level RNN does not. Through the usage of configuration files, it is possible to use the toolkit without writing additional code (Yang and Zhang, 2018). The RNNs can, through configuration parameters, be replaced by convolutional neural networks (CNN) (Goodfellow et al., 2016, p. 321).

[4] `https://github.com/google-research/bert/blob/88a817c3/tokenization.py#L311` [Retrieved: 20-02-12]

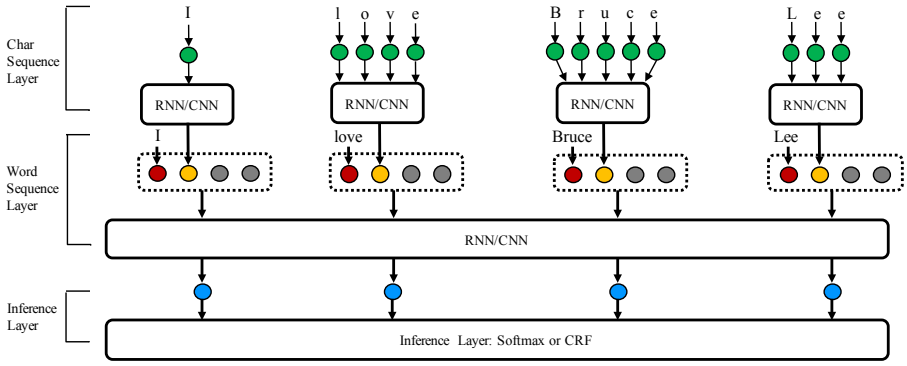


Figure 2.8.: NCRF++ makes use of a CRF in combination with RNNs (graphic taken from Yang and Zhang 2018).

2.4. Multivariate Naïve Bayes

Naïve Bayes is a standard method in machine learning and can be used for text classification. We make use of it in our spoiler classification baseline models. Its *Bernoulli model* variant considers the binary feature of a given word's occurrence in the document. The *multinomial* variant of the model takes into account the number of occurrences of a word (Sammut and Webb, 2010, p. 714).

$$P(c|d) = P(c) \prod_{1 \leq k \leq n_d} P(t_k|c) \tag{2.2}$$

Equation (2.2) shows how, using Naïve Bayes, the probability of a given document d belonging to a given class c is calculated. n_d is the number of tokens in a document and t_k the k-th token, with $P(c)$ being the probability of a specific class.

$$P(t|c) = \frac{T_{ct}}{T_c} \tag{2.3}$$

The probabilities for a token t given the class c are calculated on the training data as detailed in Equation (2.3). Here, T_{ct} and T_c are the number of occurrences of the term t in c, and the total number of tokens in c respectively (Manning et al., 2008, p. 253).

A Naïve Bayes classifier classifies a document by maximizing Equation (2.2) for a given document. Naïve Bayes, when applied to terms in a document, is a bag-of-words model, meaning that the order of words in the document is not relevant. An approach to remedy this loss of information is to operate on n-grams of terms instead of terms themselves, thereby capturing some contextual information.

The application of Naïve Bayes operates under the assumption of conditional independence of features, that is $P(A, B|C) = P(A|C) \cdot P(B|C)$, where A, B and C are random variables (Blitzstein and Hwang, 2019, p. 65). In other words and applied to the domain of text classification, the presence of a specific token does not give an indication as to the probability of the presence of a different token in the same document. In practice, however, this assumption is often violated when operating on text (Manning et al., 2008, p. 258). For example, in a

corpus related to programming, the word "hello" would presumably be very likely to occur in conjunction and thus in one document with the word "world."

Smoothing

A problem that often arises when classifying documents using Naïve Bayes is the probability of unseen words, meaning words that are not in the training data. Any unseen word has the probability $P(t|c) = \frac{0}{T_c} = 0$. The probability zero of a single word presents a problem, as the probability of any document containing this word will also be zero (due to the multiplication in Equation (2.2)). A simple approach to fix this is to use *Laplace Smoothing* (Manning et al., 2008, p. 258), which sets the minimal count of any unseen word to one, such that:

$$P(t|c) = \frac{T_{ct} + 1}{T_c + |V|} \tag{2.4}$$

In Equation (2.4), $|V|$ is the vocabulary size, i.e., the number of known terms. In effect, any word now occurs at least once in each document. Smoothing makes Naïve Bayes more robust as even documents with unknown words produce sensible class probabilities. Juan and Ney (2002) have empirically shown that the Laplace smoothing approach is suboptimal, finding other methods to be preferable. Generally, the suboptimal performance of Laplace Smoothing can be attributed to the fact that frequencies of words follow the power-law (i.e., there is a long tail of very infrequent words) (Gale and Church, 1994). In this thesis, we only use Naïve Bayes with Laplace-Smoothing as a baseline model; as a result, the potential shortcomings of the approach are not a substantial problem.

2.5. Evaluation Metrics

This section starts by discussing widely adopted metrics for classification tasks. Subsequently, less widely adopted metrics, specific to segmentation tasks, are discussed.

2.5.1. Accuracy, Recall, Precision, and F_1 Score

Traditionally, many machine learning tasks are evaluated using the measures accuracy, recall, precision, and F_1 score. These metrics can be explained using the confusion matrix. Figure 2.9 shows such a matrix, listing the names for certain categories in a binary classification task. While the rows show the actual class of a given sample, the columns show the predicted class. The number of true positives is the number of samples with the actual class 1, which were also predicted to be of class 1 by a model. In our case, this would be the number of spoilers that are recognized as such. The other elements of the confusion matrix follow the same general pattern, indicating the number of samples for all true and predicted class combinations.

Accuracy is defined as the number of correct predictions divided by the number of total predictions (Sammut and Webb, 2010, p. 9). In the case of the confusion matrix, this is $\frac{TP+TN}{TP+TN+FP+FN}$. While accuracy works well for the balanced classification case, it is not well

		Predicted Class: 1		Predicted Class: 0	
True Class	1	True Positives	(TP)	False Negatives	(FN)
	0	False Positives	(FP)	True Negative	(TN)

Figure 2.9.: A confusion matrix (Sammut and Webb, 2010, p. 209) shows the number of examples per actual and predicted class.

suited for tasks with class imbalance. To remedy the class imbalance problem, we use two other metrics: **precision** and **recall** calculated using $\frac{TP}{TP+FP}$ and $\frac{TP}{TP+FN}$, respectively (Sammut and Webb, 2010, p. 209,p. 902). A combination of the two into just one number allows for easier comparisons of models. This is called the F_1 **score** and is calculated as follows: $\frac{2 \cdot precsion \cdot recall}{precision+recall}$. The generalized form of this score is called F_β, in which the precision's impact on the general score can be increased by increasing β (with $\beta = 1$ being equivalent to the F_1 score) (Sammut and Webb, 2010, p. 397). **Specificity** is defined as $\frac{TN}{TN+FP}$ (Sammut and Webb, 2010, p. 209), and can be understood as the recall for the negative class.

Another approach to deal with class imbalance is to calculate metrics on a per-class basis and average them. This per-class approach is referred to as macro averaging (Sammut and Webb, 2010, p. 292), and increases the impact of small classes. In this case, the accuracy would then be calculated as $\frac{\frac{TP}{TP+FN}+\frac{TN}{TN+FP}}{2}$, meaning the average of recall and specificity, and is referred to as average class accuracy (Kelleher et al., 2015, p. 417).

2.5.2. WindowDiff and WinPR

WindowDiff and WinPR are evaluation metrics specifically designed for segmentation tasks (Scaiano and Inkpen, 2012). We will make use of these metrics for evaluating our spoiler sequence tagging efforts. These metrics can be used to interpret near misses in segmentation as partially correct results while allowing the leniency with regard to partial correctness to be adjusted. When segmenting natural language, placing segment boundaries wrong by just one word can often mean that the solution is close to being correct. When only interpreting a segmentation as correct, if all segment boundaries are correct, this means that the resulting score gives little insight into an approach's actual performance. While F_1 score and related metrics on a word level do take partially correct solutions into account, there is no way to adjust how far a boundary can be misplaced to be considered acceptable.

Both WindowDiff and WinPR operate on sequences of class labels. They additionally work with the notion of boundaries. A boundary occurs between any two neighboring labels wherever they are different from each other.

WindowDiff

WindowDiff, originally introduced by Pevzner and Hearst (2002), works by comparing the predicted number of segment boundaries with the number of actual segment boundaries. WindowDiff was developed as an adaption of the P_k metric (Beeferman et al., 1997) to

overcome several problems with the older metric.

In WindowDiff, a sliding window is moved across the data. The number of windows with the number of predicted boundaries being equal to the number of actual boundaries is counted. The equation $1 - \frac{\text{number of correct windows}}{\text{total number of windows}}$ gives the actual WindowDiff score. More formally, with N being the number of items to be segmented and k the window size, the score is given by Equation (2.5), where $R_{i,i+k}$ and $C_{i,i+k}$ denote the number of real and computed class boundaries, between two class labels, respectively. i represents the start index and $i + k$ the end index of the segment.

$$WindowDiff = \frac{1}{N-k} \sum_{i=0}^{N-k-1} (R_{i,i+k} \neq C_{i,i+k}) \tag{2.5}$$

Figure 2.10 illustrates WindowDiff by showing the first three windows over two given sequences. Lines indicate boundaries, and background colors indicate the element classes, meaning that in the case of text segmentation, each token might be represented by a square. The first window is considered correct; the second one is not as only the gold-data contains a boundary. The third window, however, illustrates the idea of WindowDiff. While the boundaries are not perfectly aligned, in this case, they both fall into the same window, which is considered correct.

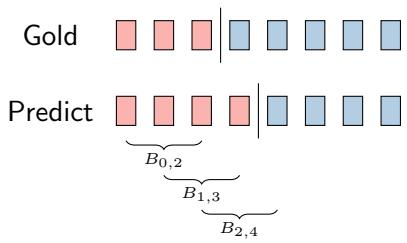


Figure 2.10.: In WindowDiff, different positions of the sliding window are evaluated based on the number of boundaries they contain (graphic inspired by Scaiano and Inkpen 2012).

This way, WindowDiff produces a score that is equal to $1 - accuracy$ for all windows (where a window is correct if the number of boundaries inside it is correct). Consequently, for WindowDiff, lower scores indicate a better result. Note that a window size of $k = 3$ means that only two potential boundaries are considered in each window.

Scaiano and Inkpen (2012) list, among others, these potential shortcomings of WindowDiff:

- The single score of WindowDiff does not allow for prioritizing some errors over others. This is, for example, useful when false positives have less cost associated with them than false negatives.

- Adjusting the tolerance for near misses is only possible by changing the window size. This, however, also changes the fraction of windows that have a boundary on them, altering the ratio of positive and negative samples. Due to WindowDiff working similarly to accuracy, it is greatly affected by class imbalance.

WinPR

In an effort to address the potential issues with WindowDiff, WinPR was developed by Scaiano and Inkpen (2012). WinPR works differently to WindowDiff in that it produces a confusion matrix of error types. Said matrix can be used to calculate variants of precision and recall (called WinP and WinR, respectively). WinPR works similarly to WindowDiff in that it also evaluates sliding windows across the entire sequence. In WinPR, however, for each window, the evaluation is not only a check of whether or not the numbers of boundaries match. Instead, extra boundaries are counted as false positives, missing ones as false negatives, extending to all four classes of the confusion matrix. In contrast to WindowDiff, WinPR considers the boundaries of each window to be included. I.e., instead of 2 boundaries being considered with a window size of $k = 3$, WinPR considers 4.

2.6. Information Gain

Information gain is a measure of how two random variables correlate. In this thesis, it is used as a means for identifying features, specifically unigrams, indicative of spoilers (see Section 7.4.3). Equation (2.6) gives a formula for calculating the information gain for two discrete such variables S and Z. Where $H(X)$ is the entropy of a random variable, and $H(X, Y)$ is the joint entropy of two random variables (Li et al., 2014).

$$IG(S; Z) = H(S) + H(Z) - H(S, Z) \tag{2.6}$$

Equation (2.7) defines the entropy of a random variable P. Entropy is a measure for the probability distribution of the i values a discrete random variable can take, with a low entropy value indicating even distributions (Downarowicz, 2007). When using the base-two logarithm, entropy is measured in bits or shannons. In the case of the natural logarithm, entropy is measured in nats (Latham and Roudi, 2009).

$$H(P) = -\sum_{i=1}^{n} p_i log_2(p_i) \tag{2.7}$$

In the case of classification in machine learning, information gain can be used as a means of measuring the correlation of a random feature variable with the random class variable (Li et al., 2014). Intuitively, if the information gain of a feature, with regard to a class, is high, the feature is salient.

3. Related Work

In this chapter, we will discuss previous work on the automatic detection of spoilers. We will discuss the different approaches to the task that have been taken and how they relate to the questions we explore in this thesis.

3.1. TV Tropes Dataset

Boyd-Graber et al. (2013) introduced a dataset for detecting spoilers. The dataset is based on TV Tropes data. Users of the TV Tropes platform collect occurrences of common motifs in TV shows and other media; these are provided in free text form by users. Sections of sentences can be tagged as spoilers. The released dataset is a collection of sentences, the title of the work (TV show or movie) they concern, and a binary feature indicating if the sentence constitutes a spoiler. Figure 3.1 shows one instance of a spoiler and one instance of a non-spoiler from the TV Tropes dataset. Note the second example ends in a period that is not an actual sentence ending; this is evidence of the fact that individual sentences in this dataset were extracted from longer documents. Sentences in the dataset have an average length of 20.03 whitespace-separated tokens. The dataset is balanced, meaning it has as many spoilers as non-spoilers. Additionally, each included work only occurs in exactly one of the dev1, dev2, test, and train splits. The exact sizes of each of the four splits are listed in Table 3.1.

Boyd-Graber et al. target a relative size of 10% for each of the development and test sets with the training set containing 70% of the total sample count. The actual distribution diverges from that target, presumably to satisfy the constraint of not including a single work in two separate sets.

> A newspaper Easter Egg in "The Reichenbach Fall" reveals that Arthur Conan Doyle exists in the Sherlock universe as a well-known writer.
>
> **Non-Spoiler**

> "The Hounds of Baskerville": The Grimpen Minefield , which is prominently introduced soon after Sherlock and Watson arrive on Dartmoor (like the original story's Grimpen Mire) , then forgotten about until the climax, when Dr.
>
> **Spoiler**

Figure 3.1.: The dataset by Boyd-Graber et al. (2013) consists of single sentences extracted from TV Tropes pages on TV shows or movies, in this case the TV show "Sherlock."

Table 3.1.: The splits of the Boyd-Graber et al. (2013) have different sizes.

Split	Number of Examples	Number of Different Works
Dev1	1066	71
Dev2	1748	72
Test	1477	62
Train	11974	679

As a classifier, Boyd-Graber et al. (2013) use an SVM (Support Vector Machine), initially classifying documents based on their tokens, token-bigrams, and stemmed tokens. The authors calculated the information gain of meta-information, specifically the genre, with regards to the sentence being a spoiler. They successfully used this metadata and other metadata, such as runtime and initial air date, to enhance their classification model. Chang et al. (2018) operate on this dataset and achieve the best results known to us to date. Their approach is discussed in detail in Section 3.4.

Table 3.2.: Classification results on the TV Tropes dataset are evaluated using the accuracy and F_1 score. Models using genre information outperform equivalent models not using it. Approaches using additional meta-information are marked using the * symbol.

Model	Accuracy	F_1 Score
Boyd-Graber et al. Baseline SVM	0.6019	0.6947
Boyd-Graber et al. SVM with Genre*	0.6777	0.6327
Chang et al. CNN	0.7082	0.7351
Chang et al. Sentence Encoder of HAN	0.7231	0.7480
Chang et al. Sentence Encoder	0.7183	0.7584
Chang et al. Sentence Encoder + Genre Encoder + Last Hidden State*	**0.7556**	**0.7847**

Table 3.2 shows a selection of existing results on the TV tropes dataset. Some approaches, in addition to the TV Tropes sentences themselves, also rely on additional meta-information.

3.2. Using Story Documents

Guo and Ramakrishnan (2010) interpret the problem of spoiler detection as a ranking problem, ranking texts by their likelihood to contain a spoiler. They build their model on a manually annotated dataset of IMDb reviews. Their basic approach is to use LDA (Latent Dirichlet Allocation) models (Blei et al., 2003) to assess how well a movie review matches the movie's synopsis.

Maeda et al. (2016) propose a method to detect spoilers based on word co-occurrence with story documents in the domain of Japanese novels. They build a set of words associated with spoilers for each document, using a combination of manual annotations and automatic extraction. They establish that many of these spoiler words tend to occur in the latter half of story documents. Based on this, they suggest checking word co-occurrences of reviews with

the latter half of story documents they relate to, in order to determine if a review constitutes a spoiler.

We will also make use of story documents in our case summaries (see Section 6.3). Relatively simple aspects like word co-occurrence and document similarity are established as meaningful features, with regard to detecting spoilers. In light of this, we suspect that our model will be able to improve its results based on story document input.

3.3. Feature Engineering for Spoiler Detection

Jeon et al. (2016) used an SVM to classify tweets about TV programs regarding whether they contain spoilers. To facilitate this, they extract different features from the raw text to then train the SVM. They used four different sets of features, comparing their effectiveness:

1. Named Entity: The frequency of named entities in a tweet. Named entities are, among others, the names of "people, organizations, and geographic locations" (Grishman and Sundheim, 1996; Yadav and Bethard, 2018).

2. Frequently used verb: The frequencies of a set of specific, frequently used verbs in a tweet.

3. Objectivity: The objectivity of a tweet, meaning the absence of words conveying emotion in the tweet. They extract this feature using existing subjectivity estimation models.

4. Tense: The predominant tense in the document.

In their experiments, objective tweets were also often those referencing another web page; for that reason, the binary feature of URLs being contained in the document was also included. The feature was not analyzed separately but only applied in conjunction with objectivity, which is why we do not explicitly list is. They found that, while tense, when used as the only features in an SVM, yielded the best F_1 score, it was outperformed by other features in terms of precision. This insight, combined with experiments on the performance of combined features, led them to the conclusion that a combination of the features was necessary to achieve the best results in spoiler detection. Generally, they found each of the features to contribute to the model's performance significantly.

Iwai et al. (2014) took a sentence-based approach to spoiler detection. They worked on a dataset that they extracted from review comments on the IMDb platform[1], which they manually annotated. In their bag-of-words approach, they used stemmed versions of contained words without any stop-word filtering as features. Additionally, they generalize over names in the input by replacing them with specialized tokens for categories of names (e.g., author, character, other). They compared five different learning algorithms (Naïve Bayes, SVM, Logistic Regression, Decision Tree, and k-Nearest Neighbor), with Naïve Bayes yielding the best performance. Finally, a user interface to enable users to use these models is proposed.

[1] `https://imdb.com`

Their proposed interface involves blacking out sections that are predicted as spoilers. A slider lets the user decide at which level of certainty to start blocking out sections.

For our purposes, the feature engineering efforts are not directly required, as we are using a deep learning approach. However, we are interested in inspecting our model (see Question 4). For these purposes, it is useful to know which features have previously been found to work well. The knowledge of which features have successfully been used allows us to assess if our models make use of similar features.

3.4. Neural Approaches

Ueno et al. (2019) rely on an LSTM model to classify spoilers in Japanese review comments. As word representation, fastText[2] vectors are used. They perform binary classification on a sentence level using an LSTM model, with the model's input being the sequence of word vectors for words in the sentence.

Chang et al. (2018) have produced the best results (that are known to us) to date on the TV Tropes dataset, using a deep learning approach. They use a GRU-based model with GloVe embeddings[3]. Specifically, a bidirectional GRU model is used, meaning recurrent units from both temporal directions are used in any given token's classification (Goodfellow et al., 2016, p.383). They make use of meta-information in the form of genres by building genre embeddings. The genre embeddings are used as input to an attention layer, thereby attributing importance to different tokens based on which genre a spoiler is associated with. Chang et al. list an example to motivate this design: "the word 'kill' is more likely to be a spoiler in the thriller genre than in the romance genre."

Both the neural approaches discussed here use recurrent architectures based on word embeddings. The word embedding approach means less training data is required as word embeddings were pre-trained using different texts (Ueno et al., 2019). Especially the work by Chang et al. (2018) is relevant to us in that they operate on the TV Tropes dataset and provide a basis for comparison with other recent approaches. In contrast to these two approaches, we will not use external word embeddings. Instead, BERT is used as an end-to-end model.

[2] https://fasttext.cc/
[3] https://nlp.stanford.edu/projects/glove/

4. Dataset

In this chapter, we discuss the Reddit dataset and the processing necessary for our purposes of using it as a machine learning dataset for spoiler detection. For an introduction into the basic terminology and concepts associated with Reddit, see Section 1.3.

We will first explore the dataset and provide an understanding of the specific information required for building a spoiler dataset. Later on, the data processing to obtain our final dataset is detailed.

The Reddit data is readily available to the research community due to the work of Jason Baumgartner[1]; we will refer to this as the Pushshift dataset. On his website[2], he provides posts and comments dating back to the inception of Reddit. The data is available in a JSON based format. Implementation details for work with this dataset are explained in Chapter 5.

Gaffney and Matias (2018) pointed out that the dataset is not complete and that a relatively large number of items are missing. While missing data has the potential to bias machine learning models, they point out that, as long as no "claims about the population" are made, the impact on the models' validity should be minimal. Since building our dataset will involve resampling for class balance, we estimate the chances of quality reduction due to biases introduced by the sampling to be minimal.

We aim to build a dataset that, while specific to Reddit, is as generally applicable as reasonably possible. While any models trained on this data are specific to Reddit, we aim to make this bias as small as possible to allow the transfer of experiences from this specific task to the general detection of spoilers. Our desire to not be overly Reddit specific means we strive to remove examples that would not generally be recognized as spoilers. An example of this is the use of spoiler tags to mark solutions to riddles, which will be discussed in Section 4.2.1. In Section 4.1.1, we explore the relevant data to aid us in designing the data cleaning process in such a way that the final dataset satisfies our requirements.

We operate on a dataset of 5.6 billion comments (all Reddit comments available in the Pusshift dataset up to April 2019). After filtering as described in this chapter, the dataset reduces to 2.6 million comments, with 50% of them containing spoilers.

4.1. Spoilers On Reddit

Generally, an essential distinction between two types of spoilers annotations on Reddit is to be made:

[1] `https://old.reddit.com/r/datasets/comments/3bxlg7/i_have_every_publicly_available_reddit_comment/` [Retrieved: 19-06-24]

[2] `https://files.pushshift.io/reddit`

1. Inline Spoiler annotations: a specific part of a post's or comment's text is marked as a spoiler. An example of this is the sentence "**Apocrypha.** Makes the College look like childish scribbles."[3], in which only the name of a place is marked as a spoiler.

2. Post spoiler annotations: An entire post is marked as containing spoilers, meaning the comments may contain spoilers, or the linked content contains spoilers. An example of this is the post titled: "Book Readers Episode Discussion - S03E06 'Immolation' - Spoilers All"[4] calling for discussions of an episode and allowing all spoilers, even those for an associated book series, in the comments.

We will mainly focus on inline spoilers, as these will provide positive spoiler examples for our dataset. Not every comment associated with a spoiler post, or even the original spoiler post's text, should be considered as a positive spoiler example. The tag might just have been added to mark the content linked in the original post as containing spoilers. Post annotations, however, are required in our approach when building the set of negative examples (see Section 4.2.3).

4.1.1. Data Exploration

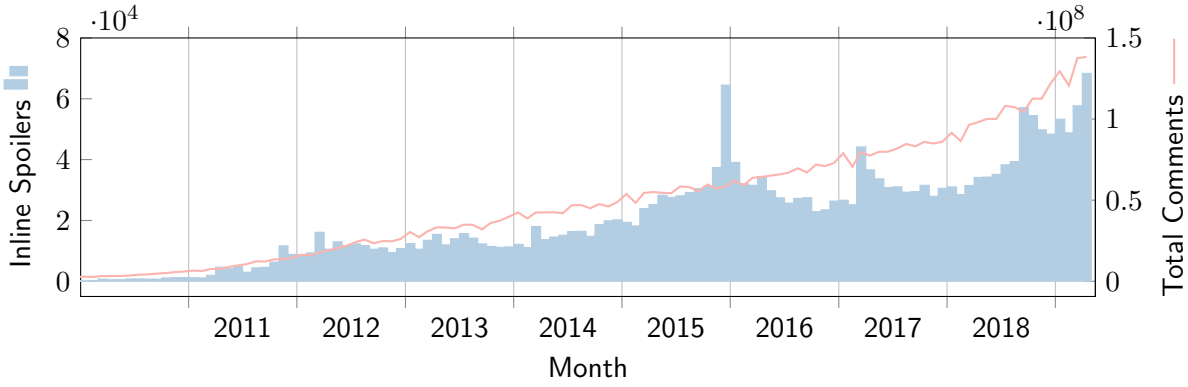


Figure 4.1.: The amount of inline spoilers per month increases with the total number of comments per month.

Figure 4.1 shows the number of comments containing inline spoilers per month in recent Reddit history. These are the comments we are interested in and those that form the basis for our dataset. The graph shows a significant spike in December of 2015. Figure 4.2 is a stacked bar chart; it was built to explain spikes in the data such as the one in late 2015. The total size of the bar represents the total number of spoilers in that month; its color shows how these comments are distributed among the different subreddits. "Others" includes all subreddits that are not individually colored in the plot. We combined two of the largest communities discussing the TV series "Game Of Thrones" to help identify the impact of the series on the total number

[3] https://old.reddit.com/r/skyrim/comments/18mf36/where_would_you_live/c8g559p/ [Retrived 20-02-13]

[4] https://old.reddit.com/r/TheExpanse/comments/8jybvs/book_readers_episode_discussion_s03e06_immolation/ [Retrieved 20-02-13]

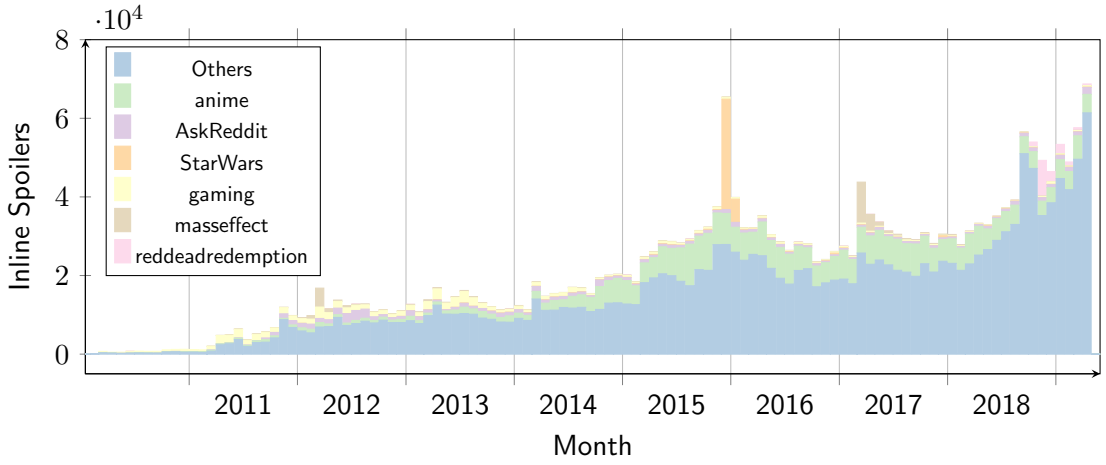


Figure 4.2.: The color indicates to which subreddit inline spoilers were posted.

of spoilers. In December 2015, a new Star Wars movie was released[5], and the spike can largely be explained by the comments that were posted to the `StarWars` subreddit in this month (see Figure 4.2). Many spikes in the plot seem to be, at least to a degree, explained by a significant uptick in comments containing spoilers in a single community (e.g., *masseffect* in 2018-03 and 2017-03). The spike in September of 2018, however, can not be explained in this way. In fact, none of the top 5 subreddits in this month exceeded *anime* in terms of the number of spoilers. Therefore, we would need to seek the effect in the long tail of subreddits with few spoiler comments. Both plots show all inline spoilers, not just those from subreddits that are included in our final dataset (see Section 4.2.1). The graphs do not show any data before 2010. The first month with a significant amount of inline spoilers is October 2009 (with 125 examples), while the very first spoiler annotation appears in December of 2008. Only roughly 0.042% of all comments contain inline spoiler annotations.

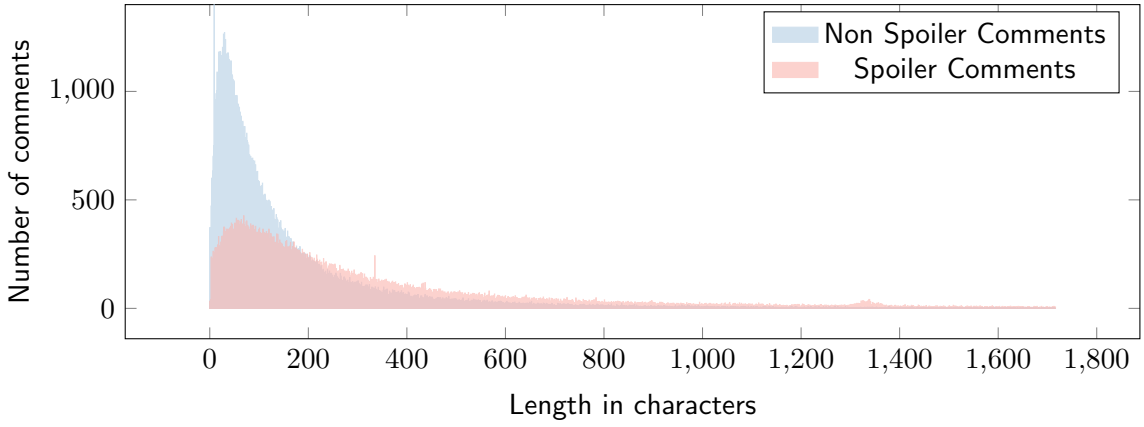


Figure 4.3.: Comparing comment lengths for the spoiler and non-spoiler comments reveals that spoiler comments are, on average, longer.

[5] `https://www.imdb.com/title/tt2488496` [Retrieved: 20-02-12]

Figure 4.3 shows a breakdown of the length distributions of comments belonging to the different two classes (either containing a spoiler or not containing one). Note that this is a visualization of the lengths before any filtering, which will be introduced later in this chapter. Specifically, two potential issues for our final dataset we observed in this data exploration step, each of these will be addressed later in this chapter.

- The very early spike in non-spoiler comments is due to comments with the text `[deleted]`. These represent comments that were deleted by their poster and can easily be filtered out.
- The spike at 335 is caused by an automated comment in the manga subreddit, explaining their spoiler policy and giving an example; this problem is addressed in Section 4.2.2.

4.1.2. Subreddit-Specific Rules

Different subreddits on Reddit set up their own rules for how to handle spoilers. While there are subreddits that allow all spoilers without tagging them (e.g., `www.reddit.com/r/freefolk`, a community discussing the TV series "Game of Thrones" and its associated book series), many others have a strong policy that explicitly forbids untagged spoilers. When looking for negative spoiler samples, this means we must implement a policy to prevent such subreddits from occurring in our data. We chose to use a manual white list of subreddits that are well-moderated and have explicit rules on spoilers (the exact approach is detailed during dataset creation in Section 4.2.1).

4.1.3. Spoiler Annotations

Both types of spoilers, inline and post based, as introduced at the beginning of Section 4.1, have been used by users long before Reddit actively supported such annotations. They were implemented by the users using different workarounds. Post spoiler annotations have been an official Reddit feature since early 2017.[6] In mid-2018 Reddit introduced official inline spoiler annotations.[7] Due to official options only becoming available recently, there is a large variety of workarounds that were, historically, used on Reddit.

For **post spoiler annotations** we found there to be two workarounds that are commonly used:

1. Users mark the post as NSFW (not safe for work), which is a long-standing Reddit feature that, among other things, prevents a thumbnail picture of a link's content from being shown.
2. Users start the title of the post with an explicit mention of spoilers, often in brackets or parentheses.

[6] `https://www.reddit.com/r/announcements/comments/5or86n/spoilers_tags_for_posts/` [Retrieved: 20-02-12]

[7] `https://www.reddit.com/r/modnews/comments/8ybmnq/markdown_support_for_spoilers_in_comments_is_live/` [Retrieved: 20-02-12]

We found a large variety of different formats in our exploration of **inline spoiler annotations**; various formats were used in different communities and at different times. All annotations follow the schema of marking a section of text. Some annotations, optionally, allow for supplying a topic which the spoiler concerns. For example, the official notation that has recently been introduced marks spoilers like this: >!*spoiler*!<, with some communities advertising the option to provide the topic of the spoiler [*spoiler topic*] >!*spoiler*!<. See Appendix A for a full list of the notations we used for in our data gathering.

4.2. Building The New Dataset

Based on the historical data from Reddit published by Jason Baumgartner[8], we build a new dataset. Our dataset differs from previously published spoiler datasets in that it not only allows for a binary classification task but also for a token-based approach. Iwai et al. (2014) built a sentence based model (see Section 3.3), but their dataset is not available. Further, our dataset is much larger than existing datasets, covering millions of samples rather than thousands.

Much of the previous work relied largely on content manually annotated during the research process (Iwai et al., 2014; Jeon et al., 2013, 2016) to use as training data. Following Boyd-Graber et al. (2013), we work on user-generated content with user-generated annotations. As pointed out by Boyd-Graber et al. (2013), collaborative editing, as is performed on TV Tropes articles, means that "users will edit the post until a consensus [spoiler] annotation is achieved". On Reddit, no such collaborative editing takes place. Instead, users create posts and are expected to follow certain, subreddit-specific rules (see Section 4.1.2). No process of finding the correct balance is in place. Instead, users might opt to defensively annotate their comments as containing spoilers, to avoid repercussions. Such an approach would lead to a high number of false positives in spoiler annotations. An example of this would be this comment: "Not sure if this is a spoiler but **Can you heal your robot ride or is it better to just find a new one?**"[9] (where bold font indicates tagged spoilers) which tags a minor detail in a game as a spoiler. On the other hand, poor moderation and careless users could lead to many false-negative annotations. An example of this would be this excerpt from a longer comment, detailing the death of a Star Wars character: "Rey is too impulsive and Snoke dies. [..]"[10].

It is therefore conceivable that the Reddit data, due to the lack of collaborative editing, is less reliable. Another problem that we see is that of which words exactly spoiler annotations cover. It is not at all clear where a boundary should be placed (see Section 7.1.1). The problem of handling non-expert annotations in language processing has been explored before, for example, by Snow et al. (2008).

[8] `https://old.reddit.com/r/datasets/comments/3bxlg7/i_have_every_publicly_available_reddit_comment/` [Retrieved: 19-06-24]

[9] `https://old.reddit.com/r/PS4/comments/5wkws6/horizon_zero_dawn_launch_week_discussion/dee2zgp/` [Retrieved: 20-02-12]

[10] `https://old.reddit.com/r/StarWars/comments/8j1yb6/what_is_a_character_flaw_of_rey/dywylce/` [Retrieved: 20-02-12]

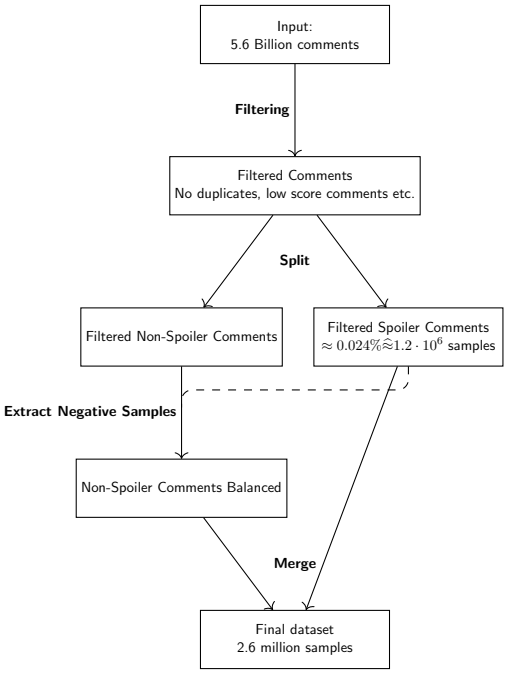


Figure 4.4.: A simplified representation of the dataset transformation process shows that a matching number of non-spoiler comments is collected.

We take several measures to improve the overall quality of the data. The basic procedure we followed when building the dataset is as follows:

- **Filter**:
  - Build a whitelist of subreddits deemed suitable, filter comments based on this whitelist.
  - Remove comments based on different criteria to improve the data quality (see Section 4.2.2).
- **Split**: Remove comments containing spoilers and build a separate set from them.
- **Extract negative samples**: For each spoiler-comment select a non-spoiler comment from the same subreddit.
- **Merge**: The extracted non-spoiler and spoiler comments form the final dataset

Figure 4.4, illustrates this procedure.

Unlike Boyd-Graber et al. (2013) we do not assign documents to the training, validation, and test sets based on their topic. Instead, we randomly distribute comments into the different splits. A temporal ordering (i.e., everything to a specific point in time is included in the training set, with subsequent data making up development and test set) would also be possible. It could yield interesting insights, especially when comparing the performance of models on shuffled data with those of the temporal data. Building datasets based on topics would also, at least to a limited extent, be possible by including only specific subreddits in each dataset, we leave this as an approach for further investigations of the topic.

4.2.1. Subreddit Whitelisting

As discussed in Section 4.1.2, we need to implement subreddit filtering to include only well-moderated communities with a reasonably strict spoiler policy. Broadly speaking, we encountered three different spoiler policies:

- None: Everything is allowed to be posted without tagging (e.g., `www.reddit.com/r/freefolk`)
- Limited: Some subreddits allow posting of information that would usually be considered a spoiler after a certain time. An example of this would be `www.reddit.com/r/survivor`, which has an express policy of prohibiting unmarked spoilers for a specific time after the airing of an episode. "There is a spoiler protection period in place for new episodes. It starts as the episode begins airing and lasts until Friday morning."[11].

  Another example would be specifically allowing older content to be posted. The "harrypotter" subreddit, for example, allows spoilers from the original book and movie series but requires those for newer movies in the franchise to be tagged.[12]

  We also consider subreddits that make spoiler tagging optional to be in this category if spoiler tagging is prevalent.
- Strict: Spoilers need to be marked, no matter how long ago the content released.

We exclude the first category from our dataset but include both the second and third. The reasoning for this is that we would like a model trained on the dataset, to be able to approximate human policies on spoilers. It is, therefore, useful to limit our dataset to subreddits, which, at least to a certain extent, limit the visibility of spoilers.

Practical concerns also limit us from ensuring all subreddits have a strict spoiler policy. Reviewing the policy and its application is already a time-consuming task for 200 subreddits, but validating this for the entire history of Reddit (as spoiler policies may change over time) would make it even more time-consuming. Another reason for excluding subreddits from our whitelist is the use of spoiler tags for other purposes. An example of this is the subreddit PictureGame[13], where spoilers are used to hide the solutions to picture-based riddles. Such uses of the tags could also be considered spoilers; we still decide to exclude them as they are very specific to Reddit. As laid out in the introduction for Chapter 4, we aim to make the dataset non-specific to Reddit.

We collected a list of the 200 subreddits with the most inline-spoilers. Based on the criteria outlined above (which we manually applied) we eliminated the following subreddits from this list:

[11] `https://reddit.com/r/survivor/wiki/spoilerpolicy?v=cee5368c-7a55-11e9-a64e-0e5b1453035a` [Retrieved: 20-02-12]

[12] `https://old.reddit.com/r/harrypotter/wiki/oursub?v=54bd1b34-9c12-11e9-9730-0ec7dedeebd0` [Retrieved: 20-02-12]

[13] `www.reddit.com/r/PictureGame/`

- alttpr
- chess
- geoguessr
- JapaneseInTheWild
- NFLstreamLives
- PictureGame
- puzzles
- riddles
- crosswords

All of these are removed for misappropriating spoiler tags. In many cases, spoiler tags are used for hiding solutions to specific riddles or questions. No subreddits are excluded for having no spoiler policy. While there are candidates for this (e.g., `/r/funny` which has no explicit policy) subreddits that show up in the top 200 list do by definition have a culture of actively using spoiler tags. The remaining subreddits make up our whitelist and are listed in Appendix B. It is important to note that just by taking the list of subreddits with the most spoiler annotations, we are already likely to select subreddits with a stricter spoiler policy. Although, we are also biasing towards subreddits with a large number of comments since we are looking at absolute spoiler counts instead of the percentages of comments containing spoilers.

4.2.2. Comment Filtering

The task of ensuring good data quality can be separated into ensuring it for each of the classes, both non-spoilers, and spoilers. In the case of the spoiler class, we strive only to include comments which are marked as spoilers for their actual content constituting a spoiler. A counterexample, which we would like to exclude, are templated comments that instruct the user on how to use spoiler annotations (as these comments usually do not contain actual spoilers).

Moderator Comments

An example of what we encountered in terms of false-positive spoilers are instructional posts explaining, using examples, how a spoiler should be tagged.[14] Such comments often follow subreddit specific templates and therefore represent easily classifiable examples. They are not useful for our dataset, as a model, expected to work on unseen comments, would barley benefit from arbitrary and repetitive examples like this. They can, usually, be caught using the simple heuristic of blacklisting moderator's comments. More specifically, we filter all comments *distinguished* as moderator comments. Distinguished comments indicate that moderators are posting in their role as moderators.

Duplicates

Duplicate comments, both in the spoiler and non-spoiler examples, are a potential source of unwanted data. For example, posts explaining the usage of spoilers, as explained in Section 4.2.2,

[14] `https://old.reddit.com/r/manga/comments/3h9vzp/slany_groups_actively_translating_break_blade/cu5ohf0/` [Retrieved: 20-02-12]

might not always be marked as moderator comments. If they are not marked as moderator comments, they can be caught by removing all duplicates. Generally, user-created comments are unlikely to constitute duplicates, meaning any duplicates are unlikely to be real user comments. The exception to this are very short comments like "Awesome!".

We do not consider those to be relevant examples for our dataset; no meaningful language understanding is necessary if the exact comment has been seen before. As a result, duplicates likely represent examples not fit for our dataset.

Ideally, near duplicates could also be eliminated, for example, if a moderator comment explicitly addressed a specific user, but everything else was following a template. Since the full dataset's size is in the order of billions of samples, a full cross-product to find duplicates would be computationally very expensive. We considered two approaches to duplicate detection:

1. *Locality Sensitive Hashing* (LSH): One approach to avoid a full cross product would be to use a locality sensitive hashing algorithm like MinHash or SimHash. These approaches would enable us to find near matches without performing comparisons of all document content combinations. It, however, still requires comparisons on the full cross-product of hashes. In the case of 5.6 billion comments we consider this to be computationally infeasible.

2. *Conventional Hashing*: Conventional hashing functions aim to minimize the probability of collisions. Therefore they are only suitable to find exact duplicates. This does, however, significantly decrease the computational burden of finding duplicates as, after a hashing operation for each document, only a grouping operation is needed to find duplicates.

Given our computational resources, this effectively gave us the choice of two approaches:

1. *Use LSH*: Instead of using LSH on the whole comment corpus, it would also be possible to only apply it to the dataset that was already filtered to be balanced. This balanced dataset contains 2.6 million comments, making the computation feasible. A drawback would be that the dataset could potentially become unbalanced as duplicates are unlikely to be evenly distributed across both classes.

2. *Conventional Hashing*: Due to the nature of exact matches, only a `groupBy` operation is needed, making this approach feasible to use on the entire comment corpus.

We decided to use a conventional hashing function. `md5` was chosen for its wide availability and relative speed. The chances of collision can, therefore, be calculated to be minimal; in fact (using an approximation of the birthday problem), a 50% risk of any collision occurring would only be reached at around $\sqrt{2 \cdot 2^{128} \cdot \ln(\frac{1}{1-0.5})} = \sqrt{2^{129} \cdot \ln(2)} \approx 2.17 \cdot 10^{19}$ documents. Cryptographic weaknesses of md5 are not relevant as we are not dealing with adversarial data.

4.2.3. Extracting Negative Samples

We collect non-spoiler comments to, in combination with the spoiler comments, build a balanced dataset. The naïve approach is to consider any comment that does not contain a spoiler

annotation to be a negative example. This approach is susceptible to including false negatives due to the, apparently, common practice of allowing untagged spoilers in comments, if the posts themselves are marked as spoilers. For this reason, we only sample comments from posts that are not marked as a spoiler. To avoid topic bias, for each comment containing a spoiler, a spoiler-free comment from the same subreddit is sampled. Biases could potentially be further reduced by sampling from the same time period; we did not choose to do this.

4.2.4. Dataset Format

The document dataset is provided as a simple collection of JSON documents with a field indicating the spoiler status and another containing the text. For the token classification task, tokenization has to be provided to enable token-specific metrics to be consistent across different approaches. We, therefore, build a dataset in the CoNLL-U Plus format[15]. As detailed in Section 1.3, different spoiler annotation schemes were and are in use on Reddit, all of these are unified into a single, token-level, binary feature in the dataset.

4.3. Summary and Final Dataset

We built a dataset based on collected Reddit comments. After finding all spoiler comments (roughly 2.4 million), we excluded many of them based on different quality criteria, resulting in around 1.3 million comments. Finally, we joined each spoiler comment with a non-spoiler one to form a balanced dataset. This balanced dataset consists of 2.6 million comments from 191 subreddits. 79% of documents are longer than 10 (white-space separated) tokens.

[15] `https://universaldependencies.org/ext-format.html`

5. Implementation

This chapter first discusses the approach we took to implementing the data processing pipeline for creating the spoiler dataset. It continues by giving some practical information related specifically to working with the Pusshift dataset by Jason Baumgartner[1].

The majority of this chapter targets those who want to work with the Pushshift dataset themselves, regardless of which aspect they want to analyze. We aim to provide practical insights to avoid pitfalls in the early stages of developing programs to analyze the dataset. Finally, we provide some insight into the machine learning tools we used.

5.1. Software Toolkit

Initially, we took the approach of importing the data into a PostgreSQL (Stonebraker and Rowe, 1986) database instance. This has the benefit of potentially allowing for a much smaller database size when committing to a fixed schema and only extracting the desired fields. In the process of analyzing the data, however, requirements for additional data fields and different data ingestions logic emerged numerous times. For example, we initially did not import the comment score but later decided to add it as a criterion for filtering. The parsing of spoiler tags had to be iterated on as we discovered different spoiler tagging syntax. While importing the dataset initially was a process of a few days on a single desktop machine, the prospect of rerunning the entire data ingestion when encountering any additional issues led us to look at different options.

In the end, we used PySpark (Zaharia et al., 2016) on a Hadoop[2] cluster for our processing instead, relying on PySpark's SQL abstractions to perform operations on the dataset. While this approach is resource-intense, it allowed for relatively quick iteration on the data ingestion logic.

5.2. Working with the Pushshift Dataset

The dataset is split into separate compressed files on a monthly basis, with comments and posts being split into individual files. The monthly datasets include some comments and posts that, according to the creation timestamp, were actually posted in the next month. When working with one or several monthly subsets of the data, it is also important to realize that

[1] `https://old.reddit.com/r/datasets/comments/3bxlg7/i_have_every_publicly_available_reddit_comment/` [Retrieved: 19-06-24]

[2] `https://hadoop.apache.org/`

comments from the beginning of the given month are usually in response to posts from the previous month.

The dataset comes compressed using different compression schemes (bz2, xz, and zstd), which are all supported by recent HDFS (Hadoop Distributed File System)[3] versions, making it is a good fit for an HDFS based storage system.

When working with the content of the individual JSON documents, the following facts might be helpful:

- Ids of the form 't1_19az' are, in fact, base 36 encoded integers, prefixed with the item type. `t1_` represents the item type, in this case, a comment. Further item types are specified in the official Reddit documentation[4] This way, each element can actually be identified by an integer instead of a string.
- The characters $<$ and $>$ are represented using their HTML escaped form (`<` and `>`).
- There are posts with a `subreddit_id` that is not present in the file `subreddits.csv`, which is also offered as part of the dataset. We believe that this is due to subreddit ids having been changed at one point in time. As a result, it is advisable to rely on the subreddit names instead of their ids.
- Some posts come without subreddit information; these appear to be advertisements. For our application, we just ignore them.
- Deleted and removed comments are included in the dataset, their text and author are replaced by "[deleted]" or "[removed]."
- As is to be expected in a document-based representation, the format of comments and posts does, in fact, change slightly over time. New fields for new concepts on the page are introduced which sometimes replace old fields.
- While examples of this are exceedingly rare, there are cases of strange data inside the text field. We did, for example, encounter a comment including a null byte as well as a comment including ASCII control characters.

We operated on 5.6 billion comments, starting from the inception of Reddit, up to and including April 2019. This data comes up to a total of 614.026 GB in size when bz2 compressed.

Our implementation for the data processing is available on GitHub[5].

5.3. Machine Learning Models

The BERT models we use are based on the Transformers library (Wolf et al., 2019), which integrates with PyTorch (Paszke et al., 2019). Our neural network implementation, based

[3] `https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html` [Retrieved: 20-02-12]
[4] `https://www.reddit.com/dev/api/` [Retrieved: 20-02-12]
[5] `https://github.com/hatzel/spoiler-data-processing`

on the Transformers library and PyTorch, is also available on GitHub[6]. It includes code paths for mixed-precision learning using apex[7]. While mixed-precision could yield performance improvements, we did not end up using it for the final results. Our repository also includes implementations for the WindowDiff and WinPR metrics. Sklearn (Pedregosa et al., 2011) was used for many of the evaluation metrics, as well as for the Naïve Bayes baseline model.

[6] https://github.com/hatzel/neural-spoiler-detection
[7] https://github.com/NVIDIA/apex

6. Methodology

This chapter discusses the models we use and their architecture. We make comparisons of results possible by applying our model architecture to the dataset previously introduced by Boyd-Graber et al. (2013). This means our models are trained on two datasets, the TV Tropes dataset and our own Reddit dataset. We additionally train a baseline model on each of the datasets. The two main tasks derived from our research questions are sequence classification and token classification. We will also consider story-document-supported models for document classification. To facilitate the exploration of model internals, we discuss our methodology for visualizing attention as well as possible shortcomings of such an approach.

6.1. Document Classification

The document classification task involves classifying entire documents into two classes: spoilers and non-spoilers. The same classification has been applied previously to the TV Tropes dataset by Boyd-Graber et al. (2013) and Chang et al. (2018).

6.1.1. Baseline Classification Model

We hypothesize that specific marker words are very indicative of a comment containing spoilers, making it a suitable and simple feature for a baseline model. If a sentence contains the word "killed," for example, we presume this makes it vastly more likely to be a spoiler. This is supported by the fact that Jeon et al. (2016) successfully used specific verbs as features (see Section 3.3). Boyd-Graber et al. (2013) published a list of unigrams that had the most information gain with respect to spoiler classification in their dataset.

Our implementation is based on the multivariate Naïve Bayes classifier, with *Laplace Smoothing*, developed by Pedregosa et al. (2011). We use their simple regular expression-based tokenizer to extract the tokens. Any sequence of two or more alphanumeric patterns is considered a token, while punctuation is ignored. Our baseline model works using both unigrams and bigrams, meaning not only individual terms, as discussed in Section 2.4, are used as features, but also pairs of consecutive words. This has the potential of capturing some contextual information of words.

6.1.2. BERT Classification Model

To assess the performance of current Transformer-based neural approaches for the spoiler detection task, we make use of the BERT model (Devlin et al., 2019). We use the pre-trained

BERT model, as provided by Devlin et al. (2019), and apply additional fine-tuning. While there are other Transformer-based pre-trained models available (Radford et al., 2018; Yang et al., 2019; Radford et al., 2019), we use BERT for its wide adoption and available tooling.

Our model is based on the cased variant of the BERT model. We chose this variant as previous work has shown named entities to be a salient feature for spoiler detection (Jeon et al., 2013) and due to the fact that BERT's cased variant was used for named entity recognition tasks by Devlin et al. (2019). Capitalization of words is a good feature in named entity recognition due to named entities (in English) generally being capitalized. The BERT model is extended by a final classification layer with a single output value. This way, binary spoiler annotation on the document level can be performed. Figure 6.1 shows the general architecture of our model. In the graphic, the blue box represents the standard BERT architecture; the dashed arrows represent the fact that all inputs affect all outputs, albeit over several encoder steps. w represents the linear layer that transforms the sequence representation `C` into a single real-valued output, which is converted to a class by comparing it to a decision boundary. BERT's application to the sequence labeling problem is discussed in Section 6.2.2.

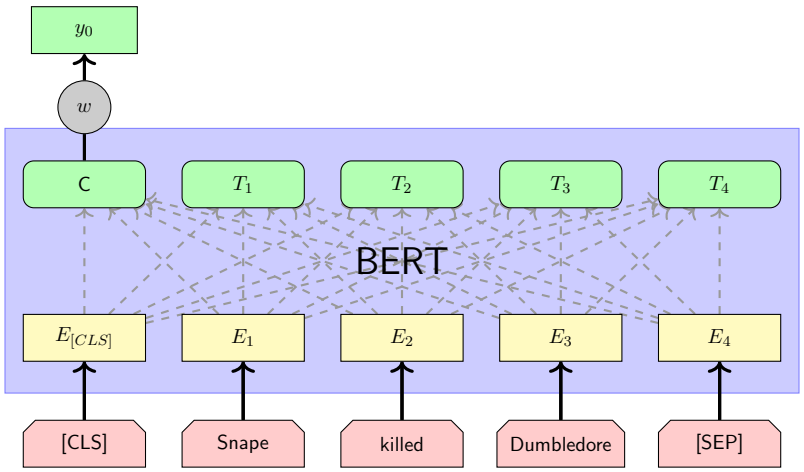


Figure 6.1.: We add a single linear layer on top of the BERT architecture. In the case of document classification, the linear layer operates on the `C` representation. (Graphic inspired by Devlin et al. 2019)

Data Preparation

We use the BERT tokenizer for preparing data for all of our BERT models. Additional preprocessing, such as parsing of Reddit's Markdown syntax, is discussed in Chapter 4.

Sun et al. (2019) explore multiple methods of handling documents that surpass the model's token limit. On sentiment data, they found the best method to use the first 128 and last 328 tokens. The TV Tropes dataset does not contain any documents that exceed BERT's 512-token limit, so this is only relevant for the Reddit dataset. Other approaches to deal with longer sequences have the potential to improve performance; however, even on the Reddit dataset, only roughly 2% of the documents are affected by the token limit. While it is not

immediately clear if the exact splitting approach by Sun et al. (2019) transfers well to our dataset, we take this approach for the 2% of documents that exceed the limit.

Training Process

Our chosen error metric is binary cross-entropy, which is well suited for binary classification tasks. No class weighting needs to be applied, as we are dealing with balanced data (both in our own Reddit dataset as well as the TV Tropes dataset). During fine-tuning, the output layer, as well as the original network's components, are updated using backpropagation. The original paper by Devlin et al. (2019) suggests specific hyperparameters, which we generally use as a starting point and change where necessary.

We use a dropout probability of 0.1 for all hidden layers, as suggested, while the final layer uses no dropout. For the activation function, we adhere to their training setup and use the GELU activation function. Due to memory limitations, we initially had to set the batch size, which Devlin et al. (2019) suggest setting to 16 or 32, to just 8. This approach was used for the TV tropes dataset. For Reddit models, different hardware was used, enabling us to make use of larger batch sizes. Batch size has previously been shown to have an impact on the ideal learning rates (Goyal et al., 2017). We performed an informed hyperparameter search to find the appropriate learning rate given our batch size and specific task. We employ early stopping to be able to perform a hyperparameter search across fewer parameters. Whenever the last three epochs yield no improvement in validation loss, we stop early, saving the best performing model from any previous epoch.

For training on the data by Boyd-Graber et al. (2013), we only use early stopping to find a good value for the number of epochs. We felt that we could not afford to expend a validation set purely for stopping, as additional data yielded large performance increases. So early stopping is performed in a first run on the validation set to find an appropriate number of epochs. In our final run to produce the final model, we instead include the validation set in the training data and perform no early stopping. On our own dataset, however, we felt that performing early stopping using a validation set was a good choice due to the larger dataset size.

We schedule learning rates based on **Slanted Triangular Learning Rates**, which was introduced by Howard and Ruder (2018) and validated to work well on BERT by Sun et al. (2019). Our parameters are $cut_{frec} = 0.1$ and $ratio = 32$ with η_{max} being our learning rate, determined via an informed search. It is worth noting that if early stopping is performed, the schedule can be interrupted at any point. In the case of training on the TV Tropes dataset, we ensure that in the final run, the learning rate schedule is applied consistently with the early stopping run. This means that even if early stopping terminated training after 2 epochs, with the schedule being specified for 6, the final training run would also be run using the first 2 epochs of a 6-epoch training schedule.

We considered implementing a scheme to decrease learning rates of lower layers in BERT but found no clear indication in any previous experiments (Sun et al., 2019) that it would yield improved results, and thus decided against implementing this approach. The hope would be

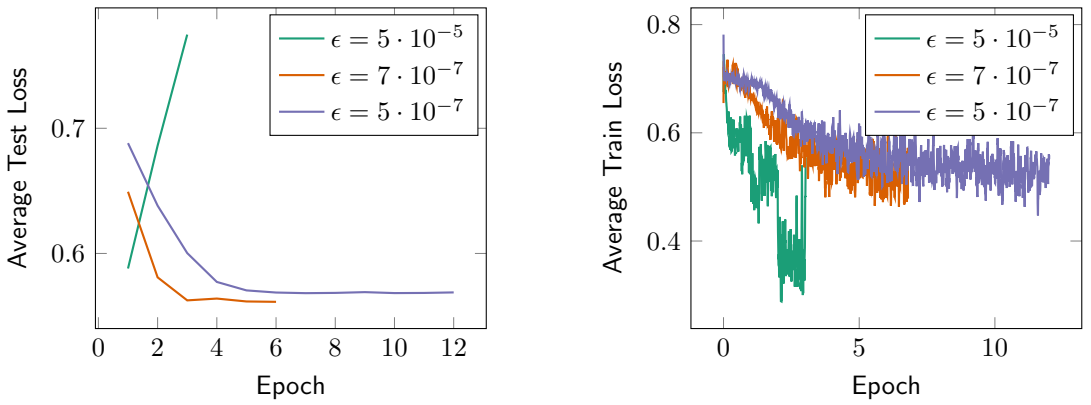
for those schemes to allow training for more epochs without breaking BERT's fundamental knowledge, which we would intuitively assume to be located in lower layers.

During our hyperparameter search, we found that we needed small learning rates compared to those in Devlin et al. (2019) to prevent overfitting. Our final hyperparameters are listed in Table 6.1.

Table 6.1.: The fine-tuning parameters were chosen similarly for all tasks after hyperparameter exploration. All our models are based on the BERT "base-cased" pre-trained model.

Task	Learning Rate	Epochs	Schedule Epochs	Batch Size
Reddit Sequence Classification	$7 \cdot 10^{-7}$	6	10	16
Reddit Token Classification	$7 \cdot 10^{-7}$	6	10	16
TV Tropes	$7 \cdot 10^{-7}$	5	10	8
Story Documents	$7 \cdot 12^{-7}$	8	10	8

Figure 6.2 illustrates the effect of different learning rate parameters on training using the TV Tropes data. While Figure 6.2a shows that only the smaller two learning rates actually converge, Figure 6.2b shows that the larger learning rate is not unstable but instead just overfits on the training data. We are not sure why our learning rates had to be much lower than those suggested by Devlin et al. (2019). One possible explanation could be the much smaller size of the TV Tropes dataset in comparison with the down-stream tasks evaluated by Devlin et al. (2019). Additionally, there is a relationship between learning rate and batch size (Goyal et al., 2017), which could contribute to us requiring smaller learning rates, given our relatively smaller batch sizes.



(a) Binary cross-entropy loss on the development set after each epoch of training

(b) Binary cross-entropy loss on training for a moving average of the last 50 batches

Figure 6.2.: Depending on the learning rate, we observe different convergence speeds or even divergence.

6.2. Token Model

For a token-based model, given the lack of previous work on this problem, we decided to use a strong baseline model based on the NCRF++ framework (Yang and Zhang, 2018). We expected this baseline to perform well, potentially on par with the BERT model, due to NCRF++ being a modern architecture and even simple models showing good results in the domain of spoiler detection, in previous work. In Chapter 7, the results will be analyzed validating or invalidating this hypothesis.

6.2.1. NCRF++-based Model

We build a baseline model using the NCRF++ toolkit (see Section 2.3). NCRF++ is a toolkit for neural language modeling, which allows for performing token classification without significant software development effort.

We configure the NCRF++ model to be trained using SGD. SGD used a learning rate of 0.015, with a learning rate decay of 0.05. The learning rate was determined using manual experiments. We use a bidirectional LSTM as the word level recurrent network with CNNs for the character level networks. For the final layer, we make use of the framework's CRF.

6.2.2. BERT Sequence Model

Just as in Section 6.1.2, we rely on a pre-trained BERT model. Now, instead of classifying entire documents, we instead classify each token individually. In the case of the sequence model, a single linear layer is applied to each token's representation, such that each token is classified based on if it is part of a spoiler. The layer weights are the same for all token positions. Figure 6.3 visualizes this process. As mentioned in Section 6.1.2, around 2% of comments in our dataset are longer than BERT's 510 content tokens. In the document classification task, this was solved by using only a subset of the input. Seeing that we now need to produce a result for each token, the same approach is not possible. Instead, for any tokens outside the model's output, we pick the majority class, thereby classifying them as non-spoilers.

6.3. BERT Story-Document-Supported Model

We build a story-document-supported model based on the TV Tropes data. The aim of this model is, for a given story document and a given text, to find out if the text contains spoilers for this document.

As show names and genres are not present in the dataset by Boyd-Graber et al. (2013), we have to perform our own extraction. Like Chang et al. (2018) we extract additional information using the work's name. Due to limitations on IMDb's usage, we make use of the OMDb API[1]. We perform an automated search on a title basis.

[1] `http://www.omdbapi.com/`

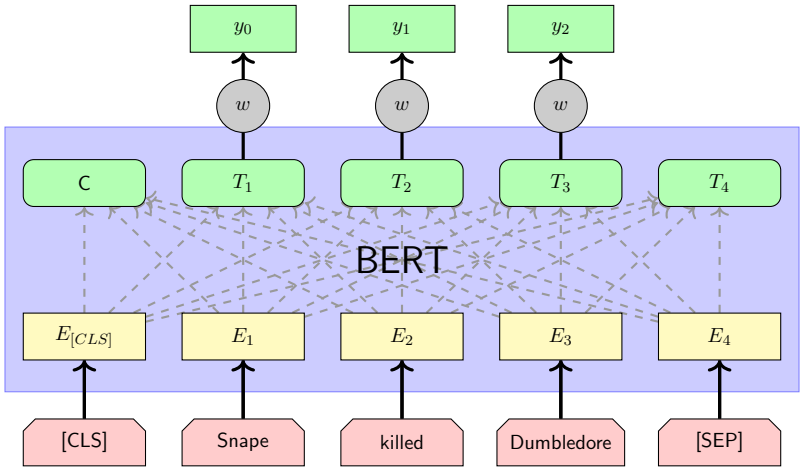


Figure 6.3.: In the case of token classification, a single linear layer operates on each individual token representation. The weights for transforming each token representation to the class output are shared across all token positions. (Graphic inspired by Devlin et al. 2019)

The samples in the TV Tropes dataset, as mentioned, do not exceed BERT's token limit. With the addition of story documents, however, we found many samples to exceed the token limit. This presented us with two options of either using a larger model or performing input sub-sampling as before. The approach we chose is to always include the full potential spoiler in the model's input and to cut off tokens from the end of the story document. It is worth noting that previous work suggests (see Section 3.2) this could be improved upon by sampling the end of the story document instead.

It is important to point out that we are not performing a full information retrieval task. To train a model that actually evaluates whether a given text is a spoiler with regard to a document–which would be an information extraction task—we would need to add training samples that are spoilers for different documents than the one they are associated with. These samples would serve as negative samples, making the task one of detecting spoilers relating to the specific document, rather than generally detecting spoilers.

We are interested in improving the spoiler detection model, similar to the genre approaches taken by Chang et al. (2018) and Boyd-Graber et al. (2013). Instead of using structured genre information, we use story documents as an information source. The approach we take has severe limitations in that some story documents are not available in the OMDb API, and others might be incorrectly associated. To be specific: out of roughly 16,000 documents, we were unable to retrieve 1333 from the API with an additional 666 works not having a summary available via the API. This means that roughly 12.3% of the documents come without a summary. We are unsure of the amount of noise with respect to mismatches of story documents and works referred to by the posts. Another potential issue is the quality of story summaries, from our manual inspection, it is clear that some "summaries" are more akin to tag lines and only describe the premise. An example of this is the summary of the comedy series "Yo soy Betty, la fea": "An outcast in a prominent fashion company, a sweet-hearted and unattractive assistant

falls hopelessly in love with her boss."[2] The problem, in this case, could also be that we only retrieved a single plot description for the whole series as opposed to one for each episode. These limitations, while significant, do not, in principle, prevent us from showing what we set out to, seeing that any significant improvement would indicate the utility of story documents in these classification tasks.

6.4. Extracting and Visualizing Attention

We would like to gain some insight into the decision process of the model. One approach to this is to visualize the attention paid to different tokens.

We extract attention similarly to Kovaleva et al. (2019). For the document classification task, this means that the attention corresponding to the final `[CLS]` token is extracted for each token in the input sequence. The final `[CLS]` token is what the final linear classification layer operates on, which means that in the document case, only the attention associated with `[CLS]` is relevant to the final result. Our visualization omits the attention relating to the special `[CLS]` and `[SEP]` tokens. The weights associated with these special tokens typically (at least in the last encoder layer which we are considering here) heavily outweigh those of content tokens. As a result, a large portion of the representation that is built up in the lower layers of the model is not visualized. The insight we can gain from them is, however, limited, as they are not easily interpretable. An input token that has no attention associated with it can still be represented in the internal state of other tokens, so this visualization is by no means perfect. The fact that such internal states can not easily be inspected has been pointed out as a potential issue when analyzing attention (Jain and Wallace, 2019).

For the token classification case, it is important to note that the attention associated with all tokens is now relevant. In an effort to simplify visualization, we average the attention across all output tokens, which does involve a loss of information and potentially makes it harder to attribute the classification of a specific token. Using attention as a method of explaining models has been called into question by Jain and Wallace (2019). Wiegreffe and Pinter (2019) argue that information useful for interpreting a model can be gained from analyzing attention, rejecting some of the claims by Jain and Wallace (2019). Additionally, experiments specific to BERT have also shown that attention does correlate with meaningful features (Kovaleva et al., 2019). Given these three results, we feel attention visualization is an appropriate approach to understanding the model, but care needs to be taken to not over-interpret the results. We visualize attention by highlighting words in blue, with increasing intensity indicating increased attention. An example of our highlighting scheme can be seen in Figure 6.4.

To systematically analyze attention, we can consider the attention score (between 0 and 1) and its correlation with specific token types. For this, we consider the token type to be a binary feature on the token level (e.g., "is the token a noun?"). We determine the Pearson correlation of such a feature with the attention score (Benesty et al., 2009).

[2] `http://www.omdbapi.com/?t=Yo+Soy+Betty+La+Fea&plot=full` [Retrieved 20-02-13]

> Great movie , nauseatingly cliche ending . I can ' t get pregnant , I ' ll never get pregnant . . . oh , I ' m pregnant now !
>
> https://old.reddit.com/r/movies/comments/1niy4a/what_was_the_last_movie_you_watched_and_what_did/ccj7x9a/ [Retrieved: 20-02-12]

Figure 6.4.: The attention a token receives is shown using the intensity of the blue background. This comment is a spoiler incorrectly classified as a non-spoiler.

7. Evaluation

In this chapter, we evaluate our models by producing results and comparing them to those of previous work. Additionally, we explore the classification network's internal decision-making process by analyzing the models' attention. Section 7.2 will list results for models trained on the TV Tropes dataset, whereas in Section 7.3 the results of different models on the Reddit dataset are presented.

7.1. Evaluation Metrics

For the classification task, we will be using two different metrics: accuracy and F_1 score. Accuracy is used as the primary metric for our binary classification, mostly to enable comparisons with Boyd-Graber et al. (2013) and Chang et al. (2018).

The impact of misclassifications on the perceived quality of the model is potentially very different for different classes. When using the model, a false positive is just a minor annoyance, whereas a false negative reveals a spoiler to a reader. We look at precision and recall (in the form of F_1 score) over accuracy, although the exact weighting of false negatives compared to false positives is not clear and out of scope for this thesis. We follow previous work in using F_1 score as a metric, allowing us to use a single point of comparison.

7.1.1. Token Model

Due to the class imbalance on the token model, we will not be using accuracy as a metric for it. While the documents themselves still have an equal class distribution, the tokens inside them do not. In fact, when looking at the tokens, we encounter class-imbalance of roughly 1:10, meaning there are ten times more non-spoiler tokens than spoiler tokens. This means that F_1 score on the token level is our metric of choice. In this case, accuracy (when not class balanced) is a poor metric, because a model always choosing the majority class would yield a 90% accuracy.

Poorly Defined Boundaries

An additional problem we encounter in this task is that of poorly defined boundaries. This task potentially has poorly defined class boundaries. See Figure 7.1 for an example to illustrate this. The actual spoiler information is covered by the model, and exact boundary placement is, in this case, debatable. A desirable trait of any metric we use would, therefore, be to recognizes near misses and assign some reward to them.

In Section 2.5.2, we discussed two error metrics: **WindowDiff** and **WinPR**, which can be used for segmentation tasks. They both fulfill the requirement of treating near misses differently from complete misclassifications. This is also the case for metrics like F_1 score on a token level. Metrics like the F_1 score do not, however, allow for adjusting the tolerance of near misses. We can easily adapt the tolerance with regards to near misses by changing the window size when using WindowDiff or WinPR.

> no i watched the show and movie
> it was just years ago forgot some bits
> **does kyoko kill herself and take sayaka with her? i knew kyoko died just not sure at what exact time her taking sayaka with her makes sense**
>
> https://old.reddit.com//r/anime/comments/82bton/what_is_your_ultimate_im_watching_something/dvbgpkn/

Figure 7.1.: This comment shows that it is not clear at which token the spoiler annotation should start. Boldness indicates the target label and underlined text the model's prediction.

As explained in the introduction to this section, different error classes could have a very different impact on the utility of the model in real-world scenarios. For this reason, we chose to focus on WinPR, which allows differentiating between different error classes. Both metrics have a window size parameter. While the WindowDiff score is sensitive to changes in the window size, WinPR is less so. Due to our dataset containing samples without spoiler sections, we run into a potential class imbalance issue when using WindowDiff. We apply WinPR to all results by adding up the resulting confusion matrices for each document and calculating WinP and WinR from the resulting matrix. WindowDiff is averaged over all windows across the dataset. (We do not include windows spanning multiple documents.) This is a form of micro averaging, which means that long documents will have a comparatively larger impact than smaller documents. Half of the average sequence length is the recommended value for the window size (Pevzner and Hearst, 2002). For the Reddit dataset, the average sequence length for spoilers is ≈ 32.2 and ≈ 46 for non-spoilers. Following the literature, this means a window size of 16 or 23 would be appropriate, with the weighted average of the classes being ≈ 41.42. We evaluate the following window sizes: 41, 16, and 5. The lowest number is chosen as we feel it does a good job of representing the uncertainty boundary of classes in our task.

7.2. TV Tropes Dataset

The test dataset provided by Boyd-Graber et al. (2013) shows a small class imbalance in some of the splits. We still use an accuracy measurement over all samples rather than a class-balanced accuracy. This is required to retain comparability to the previous results, as they use the same variant of the metric (Boyd-Graber et al., 2013; Chang et al., 2018).

Table 7.1 shows the results different models achieve on the TV Tropes dataset. Our baseline model, Naïve Bayes document classification, reaches an accuracy of 67.6% on the TV tropes

Table 7.1.: Classification results on the TV Tropes dataset by Boyd-Graber et al. (2013) indicate that more recent neural approaches yield the best performance. Naïve Bayes is shortened to NB.

Model	Accuracy	F_1 Score
Boyd-Graber et al. Baseline SVM	0.6019	0.6947
Chang et al. CNN	0.7082	0.7351
Chang et al. Sentence Encoder of HAN	0.7231	0.7480
Chang et al. Sentence Encoder	0.7183	**0.7584**
Our NB Model (Unigrams)	0.6764	0.7042
Our NB Model (Unigrams + Bigrams)	0.6635	0.7145
Our Bert Model	**0.7422**	0.7438

dataset, with an F_1 score of 0.70. Boyd-Graber et al. (2013) reached an accuracy of 60% without employing metadata (the model for this result also used bigram features). It seems surprising that our Naïve Bayes classifier would beat the SVM based model by Boyd-Graber et al. (2013). These results might be explained by their use of an SVM instead of Naïve Bayes classification, which has more hyperparameters that need to be optimized for optimal results. Further reasons for this result are explored in Section 7.4.2. Additionally, as we detail in Section 7.4.3, stop words are a salient feature when using a Naïve Bayes model for this task. The approach by Boyd-Graber et al. excluded those, potentially negatively impacting the result.

Our BERT Model is slightly outperformed in terms of F_1 score but beats the model by Chang et al. (2018) in terms of accuracy. Both neural approaches significantly outperform the baseline results.

7.2.1. Story Document Supported Model

The story-document-supported model operates on the TV Tropes document as well as a summary for the work in question. Table 7.2 shows that our BERT story document model does outperform our BERT model without story documents from the perspective of the F_1 score. The clear improvement in F_1 score illustrates that the inclusion of the story document feature can improve the model, even considering the accuracy drops slightly for that model.

Table 7.2.: Classification approaches using meta-information perform better than those that do not.

Model	Meta Information	Accuracy	F_1 Score
Boyd-Graber et al. SVM with Genre	Genre	0.6777	0.6327
Chang et al. best Model[1]	Genre	**0.7556**	**0.7847**
Our BERT Story-Document model	Story Document	0.7393	0.7562
Chang et al. Sentence Encoder	–	0.7183	0.7584
Our Bert Model	–	0.7422	0.7438

The usage of story documents does not quite yield the same improvement as the usage of genre in Chang et al.'s work. There could be several reasons for this related to our specific implementation:

- Our automated story document search could yield incorrect results
- The model's token limit is exceeded in 3.7% of cases
- For 12.3% of documents, no story document was retrieved

We know that at least $\approx 16\%$ of documents have some limitations in their usage of story documents; we are not sure how many come with the wrong one. This does indicate that story documents help in the classification, but at least in our implementation, they do not reach the same improvement as the inclusion of genre meta-information.

7.3. Reddit Dataset

After exploring the results on the TV Tropes dataset, we now show the results achieved on the Reddit dataset.

7.3.1. Classification Model

For training, we took a hyperparameter optimization approach. Due to the huge amount of data, we performed optimization on a random subset of 75 thousand randomly sampled documents from the Reddit-spoiler dataset. The same learning rates as for the TV Tropes data emerged as the best ones.

Table 7.3.: Classification results on the Reddit dataset show neural models outperforming the baseline.

Model	Accuracy	F_1 Score
NB Model (Unigram)	0.7341	0.7295
NB Model (Unigram + Bigram)	0.7381	0.7650
BERT Based Model	**0.8213**	**0.8134**

Attention

To understand the decision process of the model, we employ the analysis of attention in the final layer. Consider the examples in Figure 7.2; they seem to indicate that named entities are focused on. The words "Josh," "Stark," "Asuka," and "Shinji" receive a relatively large share of attention, indicating that named entities are a feature the model focuses on. At the same time, the marker word "kills," which we expect to receive a lot of attention, receives relatively little. Overall the attention to named entities aligns, at least partially, with our expectations.

Previous work has found the presence of named entities to be a good feature for this task (see Section 3.3). Additionally, specific frequently used words and tenses were recognized as

Ok , this could be sort of spoilerish . I just had a look at the outcomes of that decision and The saw kills Josh no matter what .

https://old.reddit.com/r/TwoBestFriendsPlay/comments/3k4ncv/two_best_friends_play_until_dawn_part_6/cuuuhb5/ [Retrieved: 20-02-12]

(a) This spoiler comment is incorrectly classified as a non-spoiler.

Call me sappy , but I want the Stark siblings to get back together again .

https://old.reddit.com/r/gameofthrones/comments/4b0k6j/allwhich_two_characters_would_you_like_to_see/d158vbw/ [Retrieved: 20-02-12]

(b) This comment is classified as a non-spoiler, in agreement with the label.

Turns out that Asuka knew about Shinji coming .

https://old.reddit.com/r/anime/comments/8f7f5r/free_talk_fridays_week_of_april_27_2018/dy3qvaz/ [Retrieved: 20-02-12]

(c) This comment is correctly classified as a spoiler.

Figure 7.2.: Named entities and specific marker words, in this case, "spoilerish" appear to receive more attention than many other tokens.

salient features. The significance of the frequently used word feature is supported by the good performance of Naïve Bayes models on the task TV Tropes task (see Section 7.2). In an effort to understand whether our model makes use of these features, we can inspect which features the model focuses on. To systematically analyze which features the model focuses on, we calculate correlations of the attention score with different token-level binary features. In order to ascertain if the focus on a feature is specific to our task, we compare the correlations on the fine-tuned model with those on the base model.

Table 7.4 shows different features and their correlation with a token's attention. Marker words are words in a list of words that have a high information gain associated with them (specifically these words: *end, ending, spoiler, also, way, story, like, spoilers, made, get*).

Table 7.4.: Of all tested features, a token being a named entity has the highest Pearson correlation with the attention it receives.

Feature	Correlation base model	Correlation fine-tuned
Named entity	0.017	**0.130**
Common noun	−0.040	0.004
Marker word	−0.017	0.055
Verb with past tense	−0.040	−0.044

We see that the model does, in fact, place a lot of focus on named entities. Somewhat surprisingly, these are focused on even more than our marker words. Interestingly, there appears to be no correlation of a verb's tense with its associated attention. This presumably has to do with the fact that earlier layers extract such grammatical features as tense. We speculate that the information is, at such a late stage in the encoding process, associated with the whole

sentence (i.e., the [CLS] or [SEP] tokens), rather than individual words.

7.3.2. Sequence Model

This approach is only evaluated on our own dataset, as the TV Tropes dataset offers no sequence annotations. We focus on the F_1 score for this model as it deals well with the class imbalance we encounter in this task. Additionally, the metrics WindowDiff and WinPR are used.

Table 7.5.: Different window sizes can impact which model yields better performance scores. For everything but WindowDiff, larger values are better.

Model	k	F_1 Score	$WinP_k$	$WinR_k$	$WindowDiff_k$
NCRF++	3	0.6485	0.2957	0.6559	**0.0481**
BERT based model		**0.6684**	0.3475	0.3970	0.0610
NCRF++	16	-	0.3236	0.7179	0.1347
BERT based model		-	0.3985	0.4552	0.1434
NCRF++	41	-	0.3481	**0.7723**	0.2457
BERT based model		-	**0.4385**	0.5009	0.2382

The results illustrate WinPR's stability with regard to adjusting window sizes. The results also show that the BERT model generally is much more precise, whereas the NCRF++ model achieves a better recall. This results in F_1 scores that are relatively close to each other for both models.

We hypothesize that the NCRF++ (Yang and Zhang, 2018) model's advantage could be explained by the explicit modeling of subsequent output labels through the use of CRFs. In our dataset, a token is vastly more likely to be a spoiler if it is preceded and followed by likely spoiler tokens. The BERT based model, however, can, in principle, learn such interdependencies as well, and, given the size of the dataset, it seems likely that it would.

> Did you watch End Of Evangelion? It's the movie the real ending of the show
>
> https://old.reddit.com/r/anime/comments/9xmkv0/anime_noob_here_just_finished_neon_genesis/e9tsala/ [Retrieved: 20-02-12]

(a) A vague comment is, in part, classified as a spoiler while it is not tagged as such.

> Tip that will make smelter demon much easier: **Gyrm Greatshield has 100% fire resistance.**
>
> https://old.reddit.com/r/DarkSouls2/comments/24zlg0/dear_everybody_who_summons_a_dragon_bro/chcjf7t/ [Retrieved: 20-02-12]

(b) The second sentence is correctly predicted to not be a spoiler. Interestingly, the named entitiy does not recieve a lot of attention.

Figure 7.3.: Attention on the sequence model shows which tokens receive the attention during final classification.

Figure 7.3 shows the attention and labels for two Reddit comments. As previously, bold text

indicates gold labels, with underlining showing the prediction. Additionally, the intensity of red now indicates the model's certainty for a specific token; thick underlining indicates the certainty clears the threshold for the token to be designated as a spoiler. In the case of Figure 7.3a, a spoiler annotation is not present in the dataset, but the model predicts one. Likely this misclassification can likely be attributed to the marker word "ending," which often indicates spoilers. The example Figure 7.3b shows a spoiler that is correctly identified. Surprisingly the named entity receives little attention in this case. Generally, noisy results are to be expected and why we performed a systematic analysis (see Table 7.4).

7.4. Discussion

In this section, we will summarize the results, exploring the model's fitness for real-world applications. We will discuss the class imbalance in real-world data and the implications for our model.

Generally, our results show that Transformers represent a promising approach to the task of spoiler detection. At the same time, we have not seen significant improvements over previous neural network approaches using recurrent neural networks for the TV Tropes dataset. In the case of Reddit, this evaluation is more difficult, but we suspect that existing models, when trained on our dataset, would perform similarly to ours. The results for the sequence model also seem promising but are hard to set in context due to the lack of previous token-based approaches.

Story documents showed some ability to improve the results but lacked behind our expectations, given the success of previous metadata-supported approaches. It is feasible that this could be explained by specific issues with our approach, such as misattributed and missing story documents for many examples in the dataset.

We found our baseline models to perform surprisingly well. In the case of the sequence model, this was to be expected due to our choice of baseline model. For the Naïve Bayes model, however, the performance was better than expected; we discuss possible reasons for this in Section 6.1.2.

Given these results and the examples, it is, however, also important not to unduly emphasize the ability of attention to explain our model's decision, at least when visualized in this limited manner. While systematic analysis shows a correlation, we have also seen examples (e.g., the named entity in Figure 7.3b), which seem to contradict these results.

7.4.1. Real-World Application

Spoilers are very rare in real-world data. In fact, only 0.042% of comments on Reddit contain spoilers (see Section 4.1.1). To illustrate this, we built a subset of the dataset in accordance with this distribution. This dataset of 20,000 random samples from our test set contains five spoilers. Figure 7.4 shows the confusion matrix on this dataset. Our model on this distribution yielded a precision of 0.0015 with a recall of 0.8. While the sample is too small to draw any real

conclusions with regard to the recall, we can observe that the precision drops significantly. We have no clear data to indicate which recall or precision would be acceptable for the application of the model. It seems clear, however, that having more than 10% of comments erroneously marked as containing a spoiler would be unacceptable.

		Predicted Class	
		Spoiler	Non-Spoiler
True Class	Spoiler	4	1
	Non-Spoiler	2734	17266

Figure 7.4.: A confusion matrix of documents distributed in accordance with the actual data shows that the precision drops considerably.

7.4.2. Overperformance of our baseline model

Our baseline model for the TV Tropes dataset performs on par with the models created by Boyd-Graber et al. (2013). This is an unexpected result, considering that their model also makes use of genre information, which significantly improves results. One contributing factor to this was that Boyd-Graber et al. performed stop word removal. Our unigram- and bigram model's performance decrease to an F_1 score of 0.68 and 0.70, respectively, when removing stop words. This might be explained, in part, by stop words working as a substitute feature for the length of documents. Naïve Bayes, which does not consider document length as a feature, can instead approximate length by relying on the frequency of very common words (which often are stop words). The salience of document length, as well as the significance of different tokens as features, are explored in Section 7.4.3. Another observation that is important in this context is that Iwai et al. (2014) also found Naïve Bayes to perform best on their data, having compared it with four other approaches, including an SVM (see Section 3.3). Ultimately, while the application of Naïve Bayes is simple and very appropriate for the unigram and bigram feature set, the large improvement over an SVM approach remains surprising.

7.4.3. Comparative Difficulty of the Task

Our models perform better on Reddit data than they do on TV Tropes data. There could be two obvious reasons for this: the size of the dataset, and the inherent properties of the documents making classification easier. We focus on exploring the latter aspect. From a manual review, we found many Reddit comments to contain explicit warnings of spoilers (see Figure 7.5 for an example) in addition to the actual notation. As a result, we hypothesize that specific features are much more indicative of the spoiler class in the Reddit dataset when compared to the TV Tropes dataset.

To understand if this really is the case or if the dataset size is the only contributing factor, we calculated the information gain (see Section 2.6) of individual words (i.e., unigrams) with regards to the class of a document. The information gain for a random sample of 10,000 documents from our Reddit training set is shown in Table 7.6.

> Spoiler: **So were the mirrors trapping the thing and when she broke them, it let it out?** This makes me want to create!
>
> https://old.reddit.com/r/pics/comments/adj6fr/i_flew_all_the_way_from_canada_to_la_and_won_my/edi50to/

Figure 7.5.: While the bold text is marked using spoiler tags, an additional explicit spoiler warning is included in this comment.

Table 7.6.: The top 10 words by their information gain (given in bits) in a 10,000-word subset of the Reddit dataset contain many stop words and show a high information gain.

Word	Information Gain
the	0.1244
to	0.0886
and	0.0835
was	0.0721
that	0.0629
of	0.0610
it	0.0581
in	0.0573
but	0.0532
for	0.0478

(a) (including stop words)

Word	Information Gain
end	0.0323
ending	0.0317
spoiler	0.0267
also	0.0264
way	0.0263
story	0.0258
like	0.0244
spoilers	0.0234
made	0.0221
get	0.0214

(b) (stop words removed)

Table 7.7.: The top 10 words by their information gain in the TV Tropes dataset contain many stop words but yield significantly less information gain.

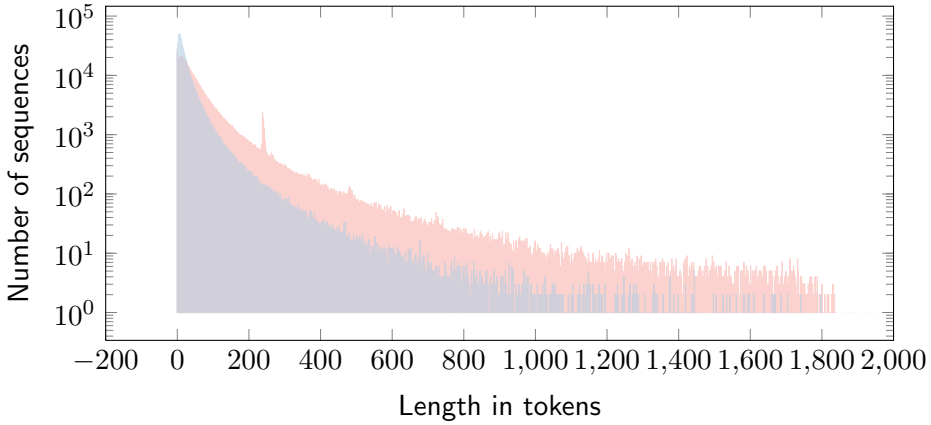
Word	Information Gain
he	0.0107
show	0.0066
his	0.0063
to	0.0058
finale	0.0058
after	0.0058
her	0.0055
the	0.0053
out	0.0052
killed	0.0052

(a) (including stop words)

Word	Information Gain
show	0.0066
finale	0.0058
killed	0.0052
death	0.0045
turns	0.0045
season	0.0043
end	0.0038
kills	0.0033
dead	0.0033
revealed	0.0032

(b) (stop words removed)

Figure 7.6.: The length distribution in sequences of spoiler and non-spoiler tokens differ.



The abundance of words that are filtered by a stop word list is interesting: We suspect it to be caused by the correlation of comment length with the class. The point-biserial correlation (Tate, 1954) of the class with the document length for our Reddit dataset, specifically the same 10,000 document subset as used for the information gain calculation, is 0.3132, meaning longer comments are in fact much more likely to contain spoilers. This correlation is illustrated in Figure 7.6, which shows the frequency for different lengths in tokens for both classes. Tokens were created using white space tokenization. The single spike at around 240 is caused by a specific comment format that was not excluded in our dataset creation. Many of the posts of this length are a table summarizing e-sports match results.[2] The TV Tropes dataset, by comparison, only shows a correlation of 0.1206 for comment lengths.

As proved by the correlation values, comment length is a good feature in spoiler classification on both datasets. Our baseline model does, however, not directly use this information. We suspect the counts of stop words to serve as a substitute feature that is captured by the Naïve Bayes model, potentially further increasing the value of stop words as a feature.

These results indicate that classification on our dataset is easier than on the TV Tropes dataset and that performance differences in the models are not merely caused by a difference in the size of the training data. Due to this fact and the presence of more training examples, it is unsurprising that all models produced better results on the Reddit dataset.

[2] `https://old.reddit.com/r/DotA2/comments/5b00bz/for_those_interested_in_why_kuro_played_carry_for/d9lnoon/` [Retrieved: 20-02-12]

8. Conclusion

In this thesis, we built a new dataset for spoiler classification. The dataset, to our knowledge, is the first to offer token-based spoiler coverage where previous datasets have only operated on sentences or documents. It also represents the largest spoiler dataset known to us. This larger dataset should allow for new approaches that were previously infeasible due to small training sets.

We used BERT, a Transformer-based, neural architecture to perform spoiler classification on a document basis, building on existing pre-trained models. Our approach for classifying documents outperforms our baseline model and performs on par with previous publications that also use neural methods. These results indicate to us that generally, Transformer architectures are a good fit for the problem but are not a substantial improvement over existing approaches. We did encounter the issue of sequences being too long for BERT to operate on, which we identify as a potential issue with the approach. Overall we can answer our initial Question 1 in the affirmative: Transformers—specifically BERT—can be used to classify comments with regard to their spoiler status effectively. Currently, however, real-world application of the model is infeasible due to the severe class imbalance encountered in real data. Many smaller optimizations to the model and training could—applied together—probably yield a substantial performance increase. Nevertheless, a clear path to improving the model dramatically enough to make real-word usage feasible is not apparent to us.

We established that sequence classification using BERT works well on the dataset, although it was still, depending on the exact evaluation method, outperformed by the NCRF++-based approach. It is, however, hard to find any specific points of comparison as no previous work we know of operated on token level annotations. Segmentation metrics were used to show that different models perform the best, depending on how tight of a boundary is required. So in response to our Question 2, we can say that sequence classification works well in principle, but in order to assess the quality in detail, more points of comparison are required.

Since we were able to show a significant increase in the model's performance when augmenting it using story documents, we can answer Question 3 positively as well. However, we did not reach the same levels of improvements enabled by the use of genre information in previous approaches. As there are numerous ways our approach to this task could be improved, we are hopeful that the use of story documents could bring more significant benefits and outperform genre information as a feature.

Inspecting the model's attention indicates that specific marker words, as well as named-entities, are an important feature for classification. These features have previously been used in manual feature engineering efforts in the domain of spoiler detection, indicating that our

model does, in part, rely on the same information. Our inspection of the model did not allow us to confirm that more complicated linguistic clues, like tense, were used. We suspect that generally, the attention visualization method, at least when only inspecting the last layer, can not yield such information. Given the results of the neural architecture outperforming other approaches, it seems very likely that some additional information like tense is taken into account. In conclusion, and as an answer to Question 4, we were able to gain some insight into the model with clear indications that the models rely on features that have previously also been used in human feature engineering efforts.

8.1. Future Work

Some measures of the quality of the user-generated annotations on our dataset would be desirable. Such measures would allow for assessing the expected limits to any machine learning model's performance. The performance of human annotators on a subset of the data would have to be measured to enable this comparison. Similarly, an understanding of the minimum performance scores required for users to perceive automatic spoiler detection as useful would also be desirable.

Some comments require the comment tree as context to classify them accurately. For example, a comment answering "Yes" to another comment may or may not be a spoiler depending on the previous comment. Including parent comments or the post's text could enable the classification and sequence models to make use of the context.

An intriguing piece of information in the Reddit dataset that we have paid little attention to is the topic information. Using this topic tag could potentially enable some classification topic modeling. It seems likely that many people only care about spoilers on specific topics, so adding this aspect could be vital for real-world applications. Additionally, the tags could aid in extracting relevant story documents, thereby enabling the application of the document supported model. The quality of tags has, however, not been throughly explored and might pose a problem.

8.1.1. Potential Model Improvements

Sun et al. (2019) investigated the effect of further in-domain pre-training, meaning continuing non-task-specific, pre-training on data from the target domain. In our case, this would mean further pre-training on Reddit data. Generally Sun et al. (2019) found this pre-training step to have a positive impact, so we would expect that our results could be improved by employing this technique. We believe that this approach, when applied to TV Tropes data, could also yield better results on the dataset by Boyd-Graber et al. (2013). Given the size of our dataset, the impact would probably be greater for the TV Tropes data.

In addition to the in-domain pre-training, Sun et al. (2019) also suggest using in-task pre-training. In our case, this would mean applying the results of either the Reddit or the TV Tropes task to the other task. Especially in the case of the TV Tropes data, we suspect that a

model retrained on the Reddit data could yield improvements.

A relatively simple way to slightly improve the token model would be to handle documents that are too long for BERT's 512-token limit (roughly 2% of the data) in a better way. Currently, the majority class is picked for any token in the document beyond the limit. A simple approach might be to split the document into several, possibly overlapping, windows each being small enough for the model. These sub-documents would then be classified as usual, with overlapping sections (those that are present in multiple sub-documents) being combined using some pooling operation (e.g., max pooling). A simpler alternative still might be the usage of large pre-trained architectures. This would only affect roughly 2% of documents in the Reddit dataset, which places an upper bound on the possible improvement.

Bibliography

Jorge Abreu, João Nogueira, Valdecir Becker, and Bernardo Cardoso. 2017. Survey of catch-up TV and other time-shift services: a comprehensive analysis and taxonomy of linear and nonlinear television. *Telecommunication Systems*, 64(1):57–74.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1706.0473.

Doug Beeferman, Adam Berger, and John Lafferty. 1997. Text segmentation using exponential models. In *Second Conference on Empirical Methods in Natural Language Processing*, pages 35–46, Providence, RI, United States.

Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. 2009. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer Nature Switzerland, Cham, Switzerland.

David M. Blei, Andrew Y. Ng, Michael I. Jordan, and John Lafferty. 2003. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022.

Joseph K. Blitzstein and Jessica Hwang. 2019. *Introduction to probability*, 2nd edition. Chapman and Hall/CRC, Boca Raton, FL, United States.

Jordan Boyd-Graber, Kimberly Glasgow, and Jackie Sauter Zajac. 2013. Spoiler alert: Machine learning approaches to detect social media posts with revelatory information. In *Proceedings of the 76th ASIS&T Annual Meeting: Beyond the Cloud: Rethinking Information Boundaries*, pages 1–9, Montréal, Canada.

Buru Chang, Hyunjae Kim, Raehyun Kim, Deahan Kim, and Jaewoo Kang. 2018. A deep neural spoiler detection model using a genre-aware attention mechanism. In *Advances in Knowledge Discovery and Data Mining*, pages 183–195, Melbourne, Australia.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of*

the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 4171–4186, Minneapolis, MN, United States.

Tomasz Downarowicz. 2007. Entropy. *Scholarpedia*, 2(11):3901. Revision #126991.

G David Forney. 1973. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278.

Devin Gaffney and J. Nathan Matias. 2018. Caveat emptor, computational social science: Large-scale missing data in a widely-published Reddit corpus. *PLOS ONE*, 13(7):1–13.

William A. Gale and Kenneth W. Church. 1994. What's wrong with adding one? In *Corpus-Based Research into Language*, pages 190–198, Amsterdam, Netherlands. Rodolpi.

Felix A. Gers and Jürgen Schmidhuber. 2001. LSTM recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340.

Njagi Dennis Gitari, Zhang Zuping, Hanyurwimfura Damien, and Jun Long. 2015. A lexicon-based approach for hate speech detection. *International Journal of Multimedia and Ubiquitous Engineering*, 10(4):215–230.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*, 1st edition. MIT Press, Cambridge, MA, United States; London, England.

Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677.

Ralph Grishman and Beth Sundheim. 1996. Message understanding conference- 6: A brief history. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*, pages 466–471, Copenhagen, Denmark.

Sheng Guo and Naren Ramakrishnan. 2010. Finding the storyteller: Automatic spoiler tagging using linguistic cues. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 412–420, Beijing, China.

Stephen Harrington, Tim Highfield, and Axel Bruns. 2013. More than a backchannel: Twitter and television. *Participations: Journal of Audience & Reception Studies*, 10(1):405–409.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia.

Hidenari Iwai, Yoshinori Hijikata, Kaori Ikeda, and Shogo Nishida. 2014. Sentence-based plot classification for online review comments. In *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, pages 245–253, Washington, D.C., United States.

Robert A Jacobs. 1988. Increased rates of convergence through learning rate adaptation. *Neural networks*, 1(4):295–307.

Sarthak Jain and Byron C. Wallace. 2019. Attention is not explanation. *CoRR*, abs/1902.10186.

Sungho Jeon, Sungchul Kim, and Hwanjo Yu. 2013. Don't be spoiled by your friends: Spoiler detection in TV program tweets. In *International AAAI Conference on Web and Social Media*, pages 220–235, Boston, MA, United States.

Sungho Jeon, Sungchul Kim, and Hwanjo Yu. 2016. Spoiler detection in TV program tweets. *Information Sciences*, 329:220–235.

Benjamin K. Johnson and Judith E Rosenbaum. 2018. (Don't) tell me how it ends: Spoilers, enjoyment, and involvement in television and film. *Media psychology*, 21(4):582–612.

Alfons Juan and Hermann Ney. 2002. Reversing and smoothing the multinomial naive bayes text classifier. In *Pattern Recognition in Information Systems, Proceedings of the 2nd International Workshop on Pattern Recognition in Information Systems, PRIS 2002*, pages 200–212, Ciudad Real, Spain.

John D Kelleher, Brian Mac Namee, and Aoife D'arcy. 2015. *Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies*, 1st edition. MIT press, Cambridge, MA, United States; London, England.

Olga Kovaleva, Alexey Romanov, Anna Rogers, and Anna Rumshisky. 2019. Revealing the dark secrets of BERT. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4364–4373, Hong Kong, China.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, San Francisco, CA, United States.

Peter. E. Latham and Yasser Roudi. 2009. Mutual information. *Scholarpedia*, 4(1):1658. Revision #186917.

Ling Li, Huawen Liu, Zongjie Ma, Yuchang Mo, Zhengjie Duan, Jiaqing Zhou, and Jianmin Zhao. 2014. Multi-label feature selection via information gain. In *Advanced Data Mining and Applications*, pages 345–355, Guilin, China.

Kyosuke Maeda, Yoshinori Hijikata, and Satoshi Nakamura. 2016. A basic study on spoiler detection from review comments using story documents. In *2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 572–577, Omaha, NE, United States.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, England.

Michael C. Mozer. 1989. A focused backpropagation algorithm for temporal pattern recognition. *Complex Systems*, 3:349–381.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035, Vancouver, Canda.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexxandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Lev Pevzner and Marti A. Hearst. 2002. A critique and improvement of an evaluation metric for text segmentation. *Computational Linguistics*, 28(1):19–36.

Lutz Prechelt. 2012. Early stopping — but when? In *Neural Networks: Tricks of the Trade: Second Edition*, pages 53–67. Springer Berlin Heidelberg.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. Technical report. `https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf`.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. Technical report. `https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf`.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. 2017. Searching for activation functions. *CoRR*, abs/1710.05941.

Raúl Rojas. 1996. *Neural networks: a systematic introduction*. Springer-Verlag, Berlin; Heidelberg.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Claude Sammut and Geoffrey I. Webb. 2010. *Encyclopedia of Machine Learning*, 1st edition. Springer US, Boston, MA, United States.

Martin Scaiano and Diana Inkpen. 2012. Getting more from segmentation evaluation. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 362–366, Montréal, Canada.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and Korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152, Kyoto, Japan.

Yusuxke Shibata, Takuya Kida, Shuichi Fukamachi, Masayuki Takeda, Ayumi Shinohara, Takeshi Shinohara, and Setsuo Arikawa. 1999. Byte pair encoding: A text compression scheme that accelerates pattern matching. Technical report, Technical Report DOI-TR-161, Department of Informatics, Kyushu University.

Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Ng. 2008. Cheap and fast – but is it good? evaluating non-expert annotations for natural language tasks. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 254–263, Honolulu, Hawaii. Association for Computational Linguistics.

Michael Stonebraker and Lawrence A. Rowe. 1986. The design of POSTGRES. *SIGMOD Record*, 15(2):340–355.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune BERT for text classification? In *Chinese Computational Linguistics*, pages 194–206, Kunming, China.

Robert F. Tate. 1954. Correlation between a discrete and a continuous variable. point-biserial correlation. *The Annals of mathematical statistics*, 25(3):603–607.

Atsushi Ueno, Yuu Kamoda, and Tomohito Takubo. 2019. A spoiler detection method for japanese-written reviews of stories. *International Journal of Innovative Computing Information and Control*, 15(1):189–198.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008, Long Beach, CA, United States.

Sarah Wiegreffe and Yuval Pinter. 2019. Attention is not not explanation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 11–20, Hong Kong, China.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771.

Vikas Yadav and Steven Bethard. 2018. A survey on recent advances in named entity recognition from deep learning models. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2145–2158, Santa Fe, NM, United States. Association for Computational Linguistics.

Jie Yang and Yue Zhang. 2018. NCRF++: An open-source neural sequence labeling toolkit. In *Proceedings of ACL 2018, System Demonstrations*, pages 74–79, Melbourne, Australia.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32*, pages 5754–5764, Vancouver, Canda.

Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. 2016. Apache Spark: A unified engine for big data processing. *Communications of the ACM*, 59(11):56–65.

A. Reddit's Inline Spoiler Annotations

As discussed in Section 4.1 a variety of inline spoiler annotation formats were used on Reddit over time and in different communities. We found this list of different annotations in our exploration, where `<spoiler topic>` denotes where the topic the spoiler is on is to be put in and `<spoiler text>` denotes where the actual text that is considered a spoiler should go.

- `(<spoiler topic>)[/s "<spoiler text>"]`
- `(<spoiler topic>)[/spoiler "<spoiler text>"]`
- `(<spoiler topic>)[#s "<spoiler text>"]`
- `(<spoiler topic>)[#spoiler "<spoiler text>"]`
- `(spoiler)[/s "<spoiler text>"]`
- `(<spoiler text>)[/s]`
- `>!<spoiler text>!<`
- `[<spoiler topic>] >!<spoiler text>!<`

B. Whitelisted Subreddits

As laid out in Chapter 4 we built our whitelist from the top 200 subreddits containing the most spoiler comments. The remaining subreddits are listed below.

- 40kLore
- AceAttorney
- anime
- anime_irl
- Animemes
- Animesuggest
- araragi
- arrow
- AskReddit
- AskScienceFiction
- asoiaf
- assassinscreed
- awwnime
- batman
- bindingofisaac
- Bioshock
- blackmirror
- boardgames
- BokuNoHeroAcademia
- books
- Borderlands
- breakingbad
- comicbooks
- Cosmere
- criticalrole
- danganronpa
- dankmemes
- darksouls
- DarkSouls2
- darksouls3
- dbz
- DCcomics
- DDLC
- DestinyTheGame
- Deusex
- Diablo
- dishonored
- DivinityOriginalSin
- doctorwho
- DotA2
- dragonage
- Falcom
- fallenlondon
- Fallout
- Fantasy
- FanTheories
- fatestaynight
- FFBraveExvius
- ffxiv
- FFXV
- FinalFantasy

- fireemblem
- fireemblemcasual
- FireEmblemHeroes
- FlashTV
- fo4
- FreeKarma4U
- funny
- gallifrey
- GameDeals
- gameofthrones
- Games
- gamindustri
- gaming
- Gamingcirclejerk
- girlsfrontline
- Gloomhaven
- grandorder
- GrandTheftAutoV
- gravityfalls
- Guildwars2
- Gundam
- halo
- harrypotter
- HollowKnight
- horror
- HPfanfiction
- HPMOR
- india
- investing
- Jokes
- JRPG
- Kaguya_sama
- katawashoujo
- KDRAMA
- KingdomHearts
- koreanvariety
- KotakuInAction
- leagueoflegends
- lifeisstrange
- LightNovels
- loodborne
- magicTCG
- Malazan
- manga
- Marvel
- marvelstudios
- masseffect
- MassEffectPhoenix
- Megaten
- metalgearsolid
- mindcrack
- Minecraft
- MLPLounge
- movies
- MrRobot
- mylittlepony
- MysteryDungeon
- Naruto
- NetflixBestOf
- nfl
- nier
- nintendo
- NintendoSwitch
- noveltranslations
- OnePiece
- OnePunchMan
- otomegames
- overlord
- Parahumans
- Pathfinder_RPG
- patientgamers
- pcgaming
- pcmasterrace
- Persona5
- pics
- pokemon
- printSF
- projecteternity
- PS4
- Random_Acts_Of_Amazon
- rational
- reddeadredemption
- residentevil
- roosterteeth
- rupaulsdragrace
- RWBY
- scifi
- Sekiro
- shield
- shieldbro
- ShingekiNoKyojin
- ShitPostCrusaders
- skyrim
- smashbros
- SquaredCircle
- StardewValley
- StardustCrusaders
- startrek
- StarWars
- steinsgate
- stevenuniverse
- Stormlight_Archive
- subnautica
- Supernatural
- survivor
- SwordOrSheath
- swtor
- syriancivilwar
- tales
- teenagers
- television
- thebachelor
- TheExpanse
- TheLastAirbender
- thelastofus
- thewalkingdead
- TheWitness
- titanfolk
- todayilearned
- TokyoGhoul
- Toonami
- TrueAnime
- truegaming

- twitchplayspokemon
- TwoBestFriendsPlay
- Undertale
- videos
- visualnovels
- vita
- Warframe
- WebGames
- westworld
- whowouldwin
- witcher
- WoT
- wow
- Xcom
- Xenoblade_Chronicles
- zelda
- ZeroEscape

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Masterstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel — insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen — benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Hamburg, den 14.2.2020 Hans Ole Hatzel

Veröffentlichung

Ich stimme der Einstellung der Arbeit in die Bibliothek des Fachbereichs Informatik zu.

Hamburg, den 14.2.2020 Hans Ole Hatzel