# Assignment 2

- Renjie Shao (reshao@eng.ucsd.edu)
- Hongyi Ling (holing@eng.ucsd.edu)
- Chenning Yu (chy010@eng.ucsd.edu)

# Section 1: Identify Dataset

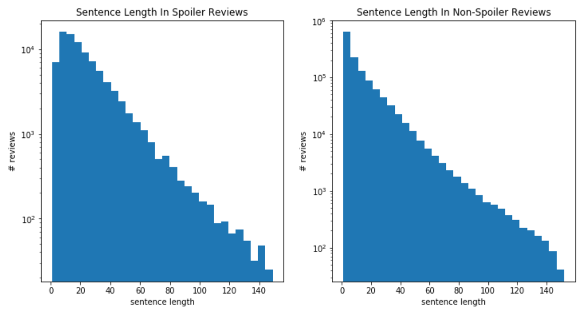
## Section 1.1: Dataset Clarification

Our dataset is the Goodreads Book Review Dataset, which can be found from https://github.com/MengtingWan/goodreads.

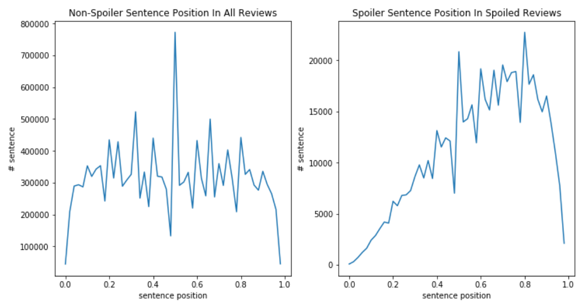
In this dataset, all the reviews are written in English. In total, there are 25,475 various book IDs and 18,892 various user IDs, and each book/user has at least one associated spoiler review according to the description in Wan et al.[1]. On review level, there are 1,378,033 reviews, and 89,627 of them include 'spoiler sentences', which means that 6.50% reviews contain spoilers. On sentence level, there are 17,672,655 sentences in total, and 3.22% sentences are spoilers. In short, this is a highly unbalanced dataset.

## Section 1.2: Exploratory Analysis

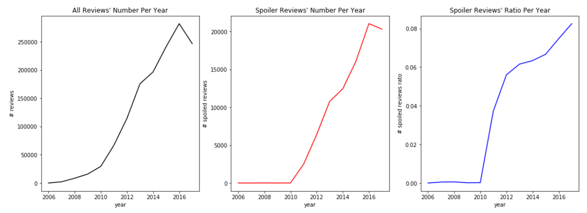
There are several data analyses done on the dataset:



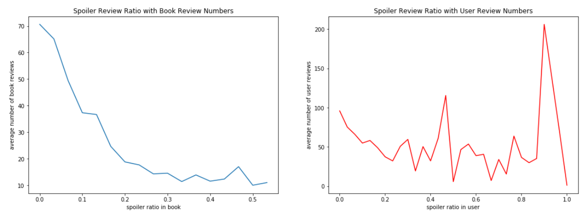
Firstly, we analyze sentence length distributions on both spoiler reviews and non-spoiler reviews. We observe that the two distributions are similar, which implies that sentence length has no correlation with spoiler appearance.



We compare the position of spoiler sentences and non-spoiler sentences, and observe that the spoiler sentence tend to be in the later part of a review and the non-spoiler sentences appears like a uniform distribution. A noticeable thing is that there is a spot lying relatively high on the x=0.5 point in the distribution in the first figure. This is because there are many reviews only containing one sentence in the data, which the implementation of our visualization algorithm calculates the position of such sentences to be 0.5.

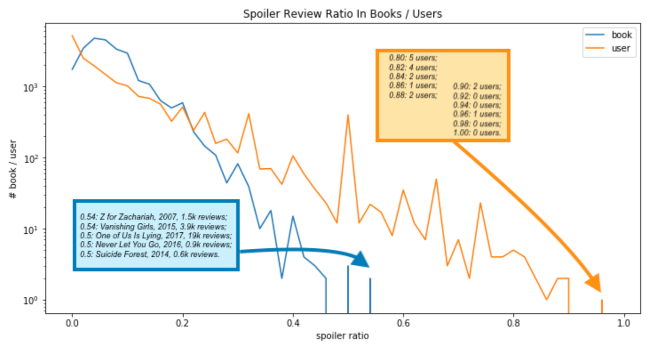


Furthermore, we try to find the relationship between the appearance of spoiler reviews and time. Surprisingly, we find that there is no spoiler review before 2011. We tried to explain this phenomenon, and find out a surprising fact: GoodReads didn't support spoiler tag until 2011! This infers a great deficiency in the original data, which is that instead of regarding no spoiler review before 2011, these reviews before 2011 are indeed unlabeled data! Therefore, the reviews before 2011 are filtered out in our experiments. We also find that the ratio between spoiler reviews and total reviews increases year by year. This might indicate that the users nowadays are getting more and more familiar with using the spoiler tags as time passed by.

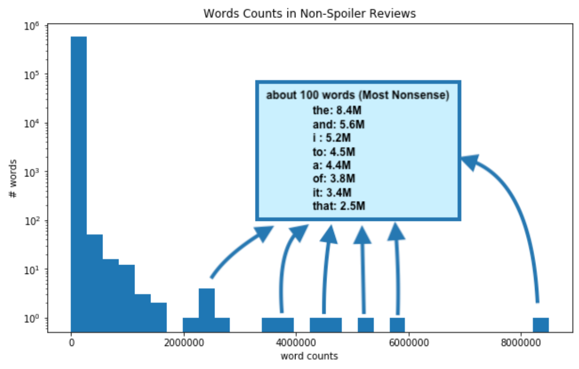


Are there going to be more spoilers when there are more reviews? We find that there is a negative correlation between the spoiler ratio of book reviews and the number of book reviews. It is equivalent to: when there are more reviews, the users tend not to write spoiler reviews. Also, there might be another explanation: the average of users' behaviour tends not to write spoiler reviews, and when the review number for a certain book grows, the spoiler ratio will go down to match with such an averaged user behaviour.

On the other side, we find that there are no certain pattern to define the correlation between the spoiler review ratio of one certain user and the number of reviews the same user has written.



We investigate the spoiler review ratio of book/user to find out the spoiler appearance tendencies for each book/user. It is observed that "user behaviors" are very different between two users/books. Also, there are no more than a half spoiler reviews for all the books, but a user may "only" write spoiler reviews. It is worth mentioning that there are no fixed pattern between the spoiler ratio of a book ,the publish date of it and the number of reviews of it, when the spoiler ratio is relatively high. We assume that it is hard to predict whether the user is going to write a spoiler review on a certain book, if there is only information about review numbers of the book and the publish date of the book. Thus, we only use what the user says to predict whether the review contains a spoiler instead of considering the information about the book. Because the users' id are uninvertible hash values, we cannot track their data. As a result, no further case study on specific user is exploited.



As is known to all, there are a lot of nonsense words in sentences. To further explain this, we count word frequencies for words in all reviews. After sorting the words based on their frequencies, we do a manual check on the top 100 popular words. We find that the top eight words are all nonsense (in a way as how human gains information from text). Furthermore, we find that most words among the top 100 words have no typical meaning for classifying a spoiler. To solve this problem, we import the stop words from nltk python package, which contains the pre-defined stop words. We filter out these stop words out of the popular words candidates.

## Section 2: Identify Predictive Tasks

| | |
|---|---|
| Model | Logistic Regression |
| Evaluation | AUC |
| Feature | Word Count, DF-IIF, WF, NWF, NWF-D |

In short, our task is to predict whether a review contains spoilers. We evaluate use AUC of ROC curve. For comparison, we use logistic regression of word count vector, with 100 most popular words, as the baseline. We assess the validity by separating the data into training, validation, test data, and assess the validity by training on training data, evaluating on test data. Several various features are applied, which will be described later in this section.

For the evaluation, We use area under the ROC curve(AUC) to evaluate our model since the dataset is severely imbalanced. In an unbalanced dataset, even a naive classifier can perform very good if it only predicts the result to be the label which has the largest samples. As a result, we use AUC because a random classifier can only produce the result around 0.5 in the AUC predicts.

We use feature engineering techniques to find the features which are best fit to tackle this problem. We will describe our mental paths to find the best feature in Section 3. In this section, we focus on how each feature is expressed as mathematical formulas, and how to process the data to obtain it.

### Section 2.1: Word Count

Word count is simple: count the emergence of each word in a certain review. The length of word count feature vector is determined by the number of popular words we chose from reviews.

### Section 2.2: DF-IIF

DF-IIF is a feature proposed by Wan et al.[1]

$$DF_{i,w} = \frac{\#review\ of\ book\ i\ containing\ word\ w}{\#review\ of\ book\ i}$$

$$IIF_{w} = log\frac{\#book\ containing\ review\ covering\ word\ w}{\#book}$$

$$DF{\cdot}IIF_{i,w} = DF_{i,w}{\cdot}IIF_{w}$$

## Section 2.3: Word Frequency (WF)

$$WF_{i,w} = \frac{\#word\ w\ count\ in\ spoiler\ reviews\ of\ book\ i}{\#word\ w\ count\ in\ all\ reviews\ of\ book\ i}$$

This counts for the frequency of each word shown in each book.

The dimension of WF is determined by the number of words model choose.

## Section 2.4: Normalized Word Frequency (NWF)

$$NWF_{i,w} = \frac{\#word\ w\ count\ in\ spoiler\ reviews\ of\ book\ i}{\#word\ w\ count\ in\ all\ reviews\ of\ book\ i} * \frac{N_i}{SN_i}$$

, where $SN_i$ is #spoiler reviews associated with book $i$ and $N_i$ is # reviews associated with i.

The dimension of Normalized WF will be a vector with fixed length, which is determined by the number of popular words we chose for books.

## Section 2.5: Normalized Word Frequency Distribution (NWF-D)

The feature we use here is called Normalized item-wise Spoiler Word Frequency Distribution, and for short, it is called as Normalized WF distribution. And we describe it as the following equation:

$$NWFD_{review} = Hist(\{Normalized\ WF_{i,w} \mid w \in\ W_{review}\})$$

where $i$ means the id of the book of the current review, and $w$ is the word we chose from the word candidates $W_{review}$.

The dimension of Normalized WF Distribution will be a vector with fixed length, which is determined by the number of bins for the distribution.

The histogram function is a function which transforms the word vector into the proximity distribution of the words. In particular, the range of $Normalized - WF_{i,w}$ is between [0, 150] for any word. For example, we want to produce a histogram vector with 30 dimensions, we count the number of words $w$ in $W_{review}$, whose $Normalized - WF_{i,w}$ value is in [0,5], and this count is the first entry of $Normalized\ WF\ Distributionn_{review}$. Furthermore, we observe that such distribution is highly unbalanced, because most words tend to stay in the lower interval. As a result, we took the logarithmic number of each entry in the histogram, and then normalized them to be 1, since we want it to be proximity distribution, and let it to be a general feature in various books.

## Section 2.6: How to Process Data and Get Features

To process the data to obtain our feature, we firstly delete the lines in the original data whose date equals to or earlier than 2010, because they are indeed unlabeled data since the spoiler tag on GoodReads did not launch until January, 2011.

For word count and DF-IIF, we use the most 500 popular words in all reviews which are not covered in the stop words defined by nltk library.

For WF, NWF, and NWF-D, instead of using the top-ranked popular words among all the books, we design the specified popular word for each book. The logic here is that different book should have different popular keyword for detecting spoilers. To filter out the nonsense words which are not defined by nltk library, we try to remove the most popular words in all the reviews from these specified words, since according to our observations on the original data, the 100 top-ranked popular words in all the reviews are almost nonsense. As a result, for each book, we choose the 500 top-ranked words in the reviews of the current book, and these words should not appear in the 100 top-ranked popular words in all the reviews.

After we got the top 500 most popular words for each feature, we calculate feature vectors using the equation described above.

For the pipeline of training, validation, testing, we firstly shuffle the data to get an unbiased distribution. Secondly, we split our data using the approach suggested in Wan et al.[1]: use 20% samples for test set, and 10000 samples for validation set, and let other samples to be the train set. This means that there are 1,047,857 samples for training, 10,000 samples for validation, and 264,464 for testing. We firstly extract the featurized samples based on the train set. We train our model based on these training samples. If there is any hyper-parameter, we tune based on the validation dataset. Finally, after the model has finished learning all the parameters, we evaluate the model on the test dataset, with the evaluation of AUC.

Readers might be wondering how we think through this problem and get this formula. We will address this problem in Section 3.

## Section 3: Model Description

### Section 3.1: Model Selection

Because this is a large dataset with a lot of bias from user and book, and a lot of noise introduced by the subjectivity of the definition of a spoiler, based on our limited computational power, we have two choices: Support Vector Machine and Logistic Regression. After some comparison experiments, we decided to use the logistic regression, because SVM needs a huge amount of time to train. At one night, we leave our laptop, begin training SVM before we went to bed, and the training procedure had not finished yet, when we wake up and check the laptop on the next morning. At minimum, SVM needs 9 hours to train. On the other hand, logistic regression is fast to train, which takes only about 10 minutes on the samples with 500 feature dimensions and the same sample number. As a result, we decided to use logistic regression as our model.

A brief description on logistic regression: given feature vector $f$ (including bias term) and label $y$, logistic regression tries to find the parameter  , which makes the prediction $(\cdot f)$ best match with label $y$, where $\sigma$ is the sigmoid function.

## Section 3.2: How We Get These Weird Features

To solve this problem, we made three feature design in toal: item-wise spoiler word frequency(WF), normalized WF(NWF), normalized WF distribution(WFD). Word bags and DF-IIF, a feature proposed by Wan et al.[1], are used as baseline.

For word w, item i, DF-IIF is defined as $DF_{i,w} * IIF_w = \frac{|D_{w,i}|}{|D_i|} * log\frac{|I| + \varepsilon}{|I_w|+\varepsilon}$ , where $|D_i|$ is #reviews associated with i , $|D_{w,i}|$ is #reviews containing word w, $|I_w|$ is #items containing w, $|I|$ is the total number of items.

Although DF-IIF has ability to detect spoilers, it focuses on the connection between word and reviews of specific book rather than connection between word and spoiler reviews. But in the task of spoiler detection, we want a feature to reflect the relationship between words and spoilers. For example, *Harry* and *Hermione* may both appear a lot in the reviews of *Harry Potter* but *Hermione* is more associated with spoilers. Motivated by that, we design $WF$ to identify the spoiler attribute of a word.

**WF**: For word $w$, item $i$,
$WF_{i,w} = \frac{\#word\ w\ count\ in\ spoiler\ reviews\ of\ book\ i}{\#word\ w\ count\ in\ all\ reviews\ of\ book\ i}$

$WF_{i,w}$ is large if word $w$ tends to appear in spoiler reviews of item $i$. One problem of $WF$ is that the magnitude of WF is dependent on i. For example, $WF_{i,w} = 0.1$ can mean word $w$ appears a lot in spoiler reviews of $i$ but $WF_{j,w} = 0.1$ can mean word $w$ seldom appears in spoiler reviews of $j$. To address this problem, we design Normalized WF.

**Normalized WF**: For word $w$, item $i$,
$NWF_{i,w} = \frac{\#word\ w\ count\ in\ spoiler\ reviews\ of\ book\ i}{\#word\ w\ count\ in\ all\ reviews\ of\ book\ i} * \frac{N_i}{SN_i}$
where $SN_i$ is #spoiler reviews associated with i and $N_i$ is # reviews associated with i. $Normalized - WF_{i,w} > 1$ indicates that word $w$ tends to appear in spoiler reviews. Because the words which appear frequently in spoiler reviews are associated with item $i$ , so there will be problems to handle different “spoiler words” in reviews of different item i. To solve this problem, we decide to care about the distribution of Normalized WF.

**Normalized WF distribution**: According to the magnitude of Normalized WF for all reviews and word, we split into several intervals. For item $i$ ,we calculate Normalized WF values for all words in the dictionary.Then we count the number of Normalized WF values in each interval and get a feature vector $[F_1, F_2, ..., F_n]$ :
$F_i = \#Normalized\ WF \in [x_i, x_{i+1}]$ , where $[x_i, x_{i+1}]$ is the interval we defined. In the experiment, we find that Normalized WF is distributed between [0, 150]. We want to produce a feature with 30 dimensions. As a result, we count the word in $[x_i, x_{i+1}]$ , and normalized the distribution into sum of 1. For example, if we want to have a dimension of 30, we count the words within the range [0, 5] for the first entry of the feature, and count the words within the range [5, 10] for the second entry of the feature, and so on. Furthermore, we observe that such distribution is highly unbalanced, because most words lie between the range with a relatively small number, and higher number range has

fewer words, and it forms to be a logarithmic distribution. As a result, we firstly get the log of every entry in the feature vector, and then normalize them to be sum of 1.

The assumption behind such "Histogram"-based feature vector is that although the key word for different book to detect spoilers varies from each other, the distribution for highly related keywords shares among different books. For example, "Harry" might be a spoiler key word for the reviews of "Harry Potter", and "Wenjie" might be a spoiler key word for the reviews of "Three Body". Even if they will lie in the different entries in the their own corresponding feature vector, they will contribute to similar weights when they are reorganized into the histogram. In such a manner, histogram is a more general feature, which stays stable for the various categories of books, and more robust for different words with the same semantics under spoiler detection scenario.

In our dataset, number of reviews varies among different books. If we choose the most frequent words in the whole dataset as the dictionary, we may ignore keywords of a book which has few reviews. Besides, all the features we design are item-specific. Therefore, we decide to collect different dictionaries for each book. First we calculate 100 most frequent words(except stopwords) $W_{common}$ in all reviews. Words in $W_{common}$ does not provide much information to help spoiler detection since they appear in all kinds of reviews. Then for each book $i$ , we calculate 500 most frequent words which is not in $W_{common}$ from the reviews of $i$ as the dictionary $D_i$ of book $i$. Words in $D_i$ are sorted in order of their frequency.

For a review $r$ of book $i$, its feature will be $[\#w * F_{w,i} \ for \ w \ in \ D_i]$ where F is one of the features discussed above.

# Section 4: Literature Exploration

## Section 4.1 Brief History on Spoiler Detection Literatures

Although spoiler analysis and detection is quite a new topic, there are many related work. Wan et al.[1] proposed a neural network to detect spoiler sentences on this dataset. They leverage hierarchical attention network, item-specificity feature and Item/User Spoilers & Self-Reporting Bias. TV Tropes is a widely used dataset to do spoiler detection. Boyd-Graber et al.[2] created TV Tropes dataset and incorporated additional metadata to improve the ability to detect spoilers. Several models have been proposed to automatically detect spoilers on similar datasets. Guo et al. developed a topic model based on Latent Dirichlet Allocation combining linguistic dependency information. Jeon et al.[3] crawled about 17,000 tweets about a reality TV show and manually labeled them. They design four features that are significant in the classification: Named Entity, frequently used verb, objectivity and URL and main tense of Tweet. Then they applied a spoiler detection model based on SVM. Most of state-of-art methods use neural network and deep learning. Chang et al.[4] proposed a genre-aware attention mechanism to focus on spoiler words for each genre. They also used a bidirectional gated recurrent unit to extract sentence feature and determine whether a given sentence is a spoiler with the genre-aware attention mechanism. Yang et al.[5] proposed a novel similarity-based network with interactive variance attention to detect spoilers in time-sync comments.

| | |
|---|---|
| Dataset | GoodReads, TV Tropes |
| Method | LDA, SVM, Neural Networks |

## Section 4.2 Why Neural Networks Become the Majority?

In short, neural networks have great ability to approximate any kind of function, once a bunch of large datasets is given.

Furthermore, nowadays NN-based models can learn the language model from unsupervised data (in terms of pre-train phase). Then it can transfer to the target domain using the data aligned with the distribution in the target domain. It makes the generalization criteria simple for neural networks: just give them big data.

Another important reason is that, rather than to use bag-of-word or any single-word-based methods, neural networks can extract the context in the data, and the context can be a variable length of words. This is a great property, because the text data is exactly the type of data with variable length.

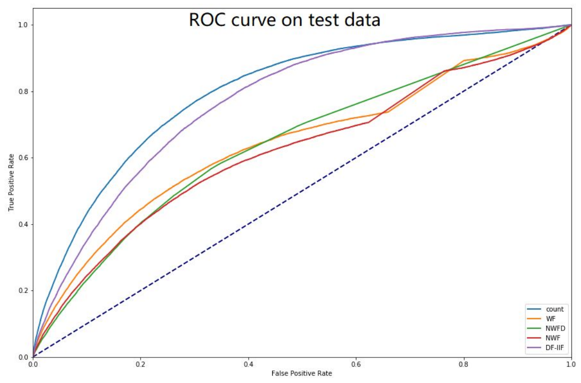
On the other hand, we observe that there are still some good works based on LDA and SVM. It shows that it is still possible to apply other methods to tackle spoiler detection problems, as long as we do not limit our imagination.

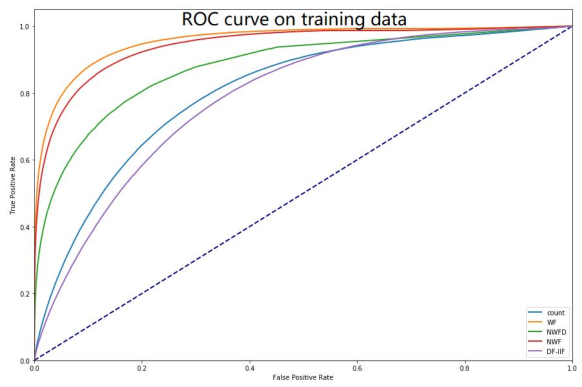
## Section 4.3 Related Datasets

In these related work, we find two useful datasets: *TV Tropes and Good reads*. Because Goodreads is a larger and newer dataset, so we choose to use it. We download it from https://github.com/MengtingWan/goodreads.

## Section 5: Performance Analysis

Firstly, we demonstrate our results with logistic regression based on each feature design, and use the same training set and test set for getting comparable results. Furthermore, rather than to view our model as a black box, we try to explain the parameter of our model. We do a thorough research on what the model learns and make some explanation on the learnt parameters, which will be demonstrated in the following several “Specific Focus” subsections.





The above curve is the AUC curve for the performance of logistic regression accompanied with each feature we try: (1) word count, (2) Word Frequency, (3) Normalized Word Frequency Distribution, (4) Normalized Word Frequency, (5) DF-IIF introduced by Wan et al.[1] For every feature, we use the regularization parameter C to be 0.01, and 500 dimension + 1 bias term for all the feature except Normalized Word Frequency Distribution, and 30 bins + 1 bias term for NWFD.

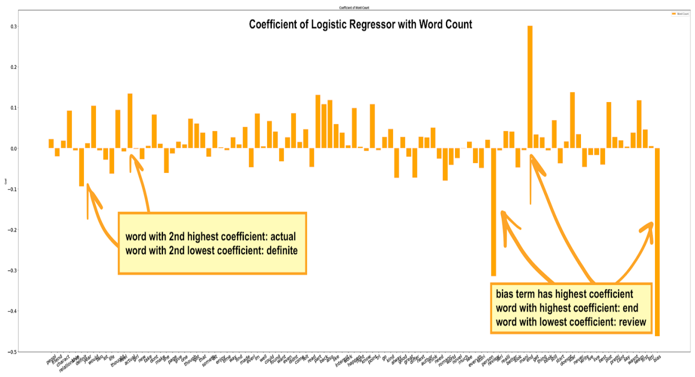
Though a little frustrated, it is surprising to see that the simple feature, word count, outperforms every other fancy feature, which the predecessors and we spent hundreds of hours on designing. As a result, rather than blowing our own horn about the success, we focus on this pain, find and reason out what makes such huge gap between imagination and reality. Is it because we design the wrong feature, or the model does not converge, or the model overfits on the training data?

## Section 5.1: Why It Fails? Big Issue: Generalization

| Model | AUC on training set | AUC on test set |
|---|---|---|
| WF | 0.897 | 0.664 |
| NWF | 0.8818 | 0.6288 |
| NWF distribution | 0.8799 | 0.6412 |

Comparing performance on training set and test set, we find that features WF, DF-IIF, NWF have very high AUC on training set and poor AUC on test set. The reason may be the generalization ability of such feature designs is poor. These features pay much attention to the words that appear in spoiler reviews on training set. Words that tend to appear in spoiler reviews are supposed to help spoiler detection and we do obtain a higher AUC on the training set. In our dataset, only 6.50% reviews contain spoilers. Because of this, spoiler words distribution may be different between training set and test set.

## Specific Focus 1: What has logistic regression learnt from data?



To focus on what the logistic regression learnt from the data, we train a logistic regression based on word-count feature extracted from 100 most popular words. We show the coefficient of the logistic regressor. The x-axis is the corresponding popular word for that entry. The y-axis is the corresponding coefficient of the regressor for each entry. We observe that the range of the coefficients is within [-0.5, 0.5]. Because there is a bias term added to the vector, there are 101 elements in the x-axis.

By watching this histogram, it is easy to find out that the entry with the highest coefficient is the bias term. The reason is that our data is a highly unbalanced set, where most samples are labeled as 0. To match with this most-zero distribution, the bias term is a large negative number so that the output of logistic regression will go to zero in most cases.

Why does 'end' have the highest coefficient? We filtered out sentences containing 'end', and we reason out that many spoilers have a tendency to tell the story with phrases such as 'happen at the end', 'toward the end'!

Why does 'review' have the lowest coefficient? Maybe because review means that the reader has some personal thoughts on the story of the book, which tends not to be a spoiler because they have already digested the content and convert the storyline into more abstracted conclusions, as the function a "review" provides.

### Specific Focus 2: Does feature ensemble help?

| Single / Ensemble Model | AUC |
|---|---|
| Word bags(baseline) | 0.7577 |
| Word bags + DF-IIF | 0.7589 ↑ |
| Word bags + WF | 0.7594 ↑ |
| Word bags + NWF | 0.6761 ↓ |
| Word bags + NWF distribution | 0.7564 ↓ |

In this section, we focus on one question: what will the result be if we ensemble these features? Will it be better (because they catch different information from texts), or worse (because of the influence of features with poor generalization ability) ?

We observe that AUC is higher compared with word bags after ensembling features DF-IIF, WF with Word Bags though only use one of them will not achieve a better result. WF+Word Bags has the best AUC of 0.7594. By ensembling word bags, the model's generalization ability is improved.

Surprisingly, ensembling NWF with word bags get a worse performance. We believe the reason is that NWF perform so good on the training set and affect word bags. When testing on the test set, NWF is useless and get a worse performance.

## Section 6: Conclusion

In this assignment, we exploit our model, logistic regression, with five designed different features to predict the existence of spoilers in reviews on GoodReads.

However, we experience failure that we cannot simply beat our baseline with applying word count with logistic regression. We deeply analyze our failure, and find out that the feature we designed has low ability to generalize because of the heavy dependence on the word occurrences in the spoiler reviews, which changes severely because of the synonym phenomena, and the highly unbalance of the dataset, which means the features we design themselves are highly overfitted on the training dataset.

To solve the generalization problem, we use ensemble techniques, and successfully prove minor improvement based on the word count feature.

## Reference

[1]Wan, Mengting, Rishabh Misra, Ndapa Nakashole, and Julian McAuley. "Fine-Grained Spoiler Detection from Large-Scale Review Corpora." *arXiv preprint arXiv:1905.13416* (2019).

[2]Boyd-Graber, Jordan, Kimberly Glasgow, and Jackie Sauter Zajac. "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*, p. 45. American Society for Information Science, 2013.

[3]Jeon, Sungho, Sungchul Kim, and Hwanjo Yu. "Spoiler detection in TV program tweets." *Information Sciences* 329 (2016): 220-235.

[4]Chang, Buru, Hyunjae Kim, Raehyun Kim, Deahan Kim, and Jaewoo Kang. "A deep neural spoiler detection model using a genre-aware attention mechanism." In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 183-195. Springer, Cham, 2018.

[5]Yang, Wenmian, Weijia Jia, Wenyuan Gao, Xiaojie Zhou, and Yutao Luo. "Interactive Variance Attention based Online Spoiler Detection for Time-Sync Comments." In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 1241-1250. ACM, 2019.