Clickbait Detection using Natural Language Processing and Machine Learning

Varun Shah

March 23, 2018

Abstract

Clickbait refers to social media posts designed to entice the clicking of an accompanying link in order to increase online readership. Clickbait detection is important for the preservation of quality and ligitimacy in social media and news publishers. In this paper, I present a model for clickbait classification using Natural Language Processing and Machine Learning. The data used was found on the "Clickbait Challenge 2017" [4] website. I utilized Python's Natural Language Tool Kit [5] and Stanford CoreNLP [8] parts-of-speech tagger to develop and implement features that would help the model classify clickbait. Results showed that once the features are added to the model, *RandomForest* achieves the highest classification accuracy of 88.2051% with an ROC–AUC of 0.928 for clickbait instances.

Contents

List of Figures

List of Tables

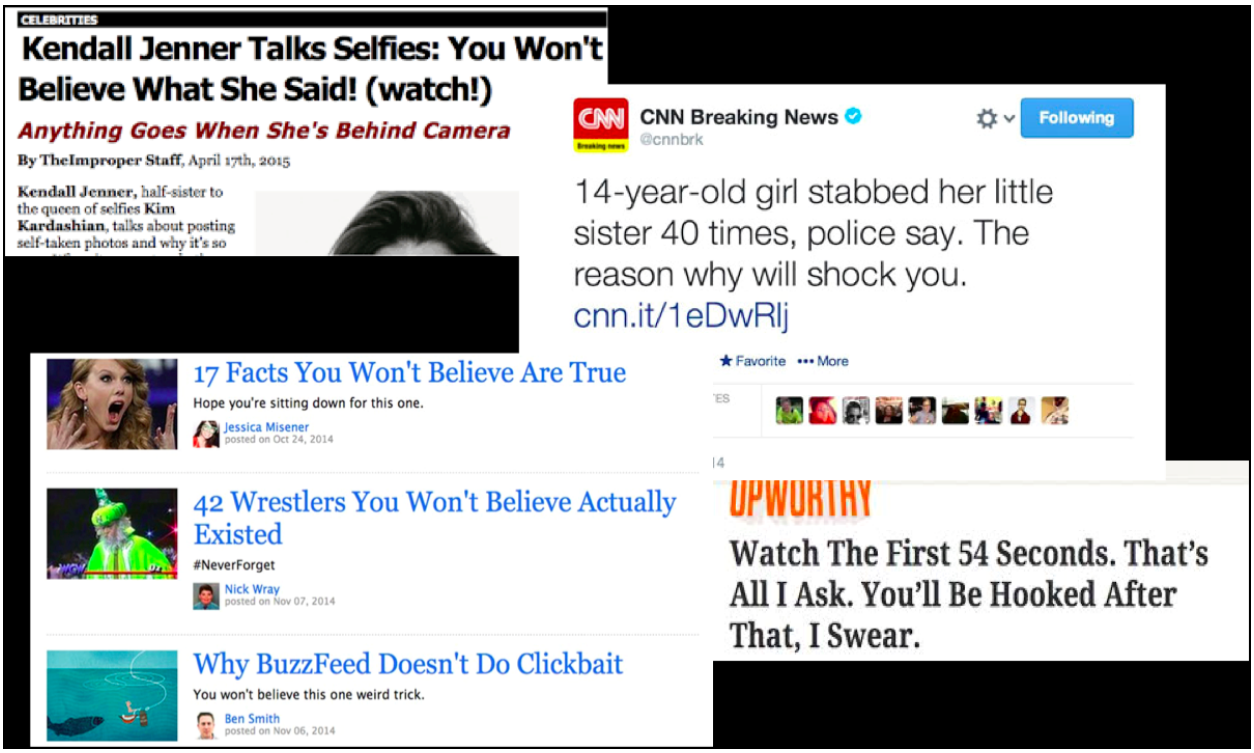

Figure 1: Examples of clickbait. *Source: Google images*

1 Introduction

Clickbait refers to social media posts that are designed to entice the clicking of an accompanying link in order to increase online readership. This is done by exploiting the "curiosity gap" – providing enough information to make readers curious, but not enough to satisfy that curiosity without requiring them to click the link accompanying the post [10]. Such posts are meant to mislead readers by exaggerating the content of the accompanying link. Figure 1 gives us some examples of clickbait where crutial information about the post is intentionally shielded in order to spark curiosity among readers, making them ask questions like "What did Kendall Jenner say?", "Why did the 14-year-old stab her little sister?", and so on. These are important pieces of information that any good journalist would mention in the title itself. More examples of clickbait include:

- A Woman Dropped Her Purse, Scattering All Her Belongings. What We Found Inside Will Blow Your Mind. $< link >$
- 9 Out Of 10 Doctors Will Not Recommend This Medicine That Could Cure Cancer. $< link >$
- Here's What Will Actually End Racism. $< link >$

Clickbait usage has increased significantly in recent years [7] and is being adopted by news publishers, violating journalistic ethics and decreasing the quality of content on one's news feed [7]. Therefore, clickbait detection is extremely important for the preservation of quality and legitimacy in social media and news publishers.

This motivates the "Clickbait Challenge 2017" [4], organized by Tim Gollub, Martin Potthast, Matthias Hagen, and Benno Stein of Bauhaus-Universität Weimar. This is an open challenge to develop a classifier that can detect clickbait in social media posts. The data is provided in the form of a Twitter corpus compiled by the founders. Each post is rated by five annotators and their mean score determines the class label, "clickbait" or "no-clickbait." The data is described in greater detail in Section 3.

In this paper, I will describe my attempt at building a clickbait classifier model. I begin by describing past attempts at clickbait analysis and detection in Section 2, then move on to describing the data used in my experiments in Section 3, the text processing and filtering involved before choosing the appropriate machine learning algorithm was applied, and the Natural Language Processing techniques that helped me add features to the model to improve its classification accuracy. For my experiments, I used Weka 3.8.1 – an open source machine learning tool.

2 Background and Related Work

The first known Machine Learning approach to clickbait detection was made by Martin Potthast, Sebastian Köpsel, Benno Stein and Matthias Hagen [7]. They compiled the first clickbait corpus, containing 2,992 Twitter tweets, 767 of which are clickbait. Their clickbait model is based on 215 features which divide into 3 categories: the teaser message, the linked web page, and the meta information. The teaser messages themselves are divided into three subcategories: the first comprises basic text statistics (number of tokens, length of message, etc.) and the other two comprise dictionary features (parts-of-speech usage, word patterns or templates, etc.). To analyze the linked web page, the readability and length of the main content is measured when extracted with the open source Boilerpipe library. Meta information comprises information about a tweet's sender, media attachments, retweets, and the time of tweeting. The corpus is randomly split into training and testing datasets in the ratio 2:1. Results showed that the $RandomForest$ classifier performs better than other algorithms, such as $LogisticRegression$ and $NaiveBayes$, acheiving an ROC–AUC [9] of 0.76 [7].

The clickbait corpus developed for their paper focuses only on Twitter posts. It also only takes into account tweets posted by the 20 most popular (in terms of retweets) publishers. Each post is assessed by 3 individuals and is labeled either clickbait or not. More so, these judgments were made only on the basis

of the teaser message of each post and not by clicking on the accompanying link. The Clickbait Challenge dataset uses judgments made by 5 individuals. This is further elaborated in Section 3 where we describe the dataset obtained from the Clickbait Challenge 2017 website [4].

Alex Peysakhovich and Kristin Hendrix [6] describe their efforts in detecting and reducing clickbait in Facebook's News Feed. They describe a system that determines whether a post is clickbait based on two key points: if the title withholds information necessary to understand the article and if the title exaggerates, creating misleading expectations of the article for the reader. The system not only identifies posts that are clickbait but also identifies the web domains and Facebook Pages that frequently post clickbait so that links posted from these Pages appear lower down in the News Feed. This change can be reversed if the Page stops posting clickbait. Their motivation is to ensure that Facebook is a place for authentic communication.

Based on research conducted by Jonas Blom and Kenneth Hansen, clickbait (or "forward-reference") posts have several key grammatical traits that help identify them as such [2]. The key traits cited are:

- Demonstrative pronouns, such as "these", "they", "them", "this", etc. For example, "*These* were Chavez's last words" [2] leaves the reader wondering what his last words were.
- Personal pronouns, such as "he", "him", "she", "her", etc. For example, "*He* wants to make the national team wear EU clothes" [2] leaves the reader wondering who "he" refers to.
- Adverbs, such as "here", "there", "somewhere", etc. For example, "*Here* you can use 4G with iPhone 5" [2] leaves the reader wondering where 4G can be used.
- Definite articles, such as "the", "a", "an", etc. For example, "In a few seconds *the* terror bomb explodes" [2] leaves the reader wondering what terror bomb is being referred to.
- Imperatives with implicit discourse diectic reference, for example in "*See* if your bank is at risk of collapsing" the word "see" refers to something revealed by the full text [2].
- Interrogatives referring to an answer given in the full text, for example, "Do you live in a violent municipality?" is not a rhetorical question and implicitly refers to an answer available in the full text [2].
- General nouns with implicit discourse diectic reference, for example, "*VIDEO*: Gigantic baby born in Texas" refers to a part of the upcoming discourse if the link is clicked [2]. Another example of this is, "*10 good reasons* to eat chocolate" implicitly refers to an upcoming discourse (a list of reasons) [2].

This gives me reason to believe that analyzing the parts-of-speech usage could be a good way to detect clickbait. My intuitions are described in Section 5.

Figure 2: Example of $postText$ [4].

3 Data

For preliminary results and feature development the *clickbait17-train-170331* dataset found on the "Clickbait Challenge 2017" [4] website is used. Unfortunately, the dataset is poorly documented and we do not know the where these posts come from (geographically) or when they were collected, but we do know that these posts were made in 2017. This dataset has 2459 instances with the following attributes:

- id: Unique integer assigned to identify each post
- $postTimestamp$: Time of the post in GMT
- $postText$: Text of the post without the link (teaser message) – see Figure 2
- $postMedia$: Image/Video accompanying post
- $targetTitle$: Title of the article linked in the post – see Figure 3
- $targetDescription$: Short description of linked article
- $targetKeywords$: Keywords used in the linked article
- $targetParagraphs$: Selected paragraphs from the linked article
- $targetCaptions$: Caption(s) of the image(s) in linked article
- $truthJudgements$: List of clickbait scores in $[0, 1]$ given to each post – see Figure 4
- $truthMean$: The mean of the scores given in $truthJudgements$

Figure 3: Example of $targetTitle$ [4].

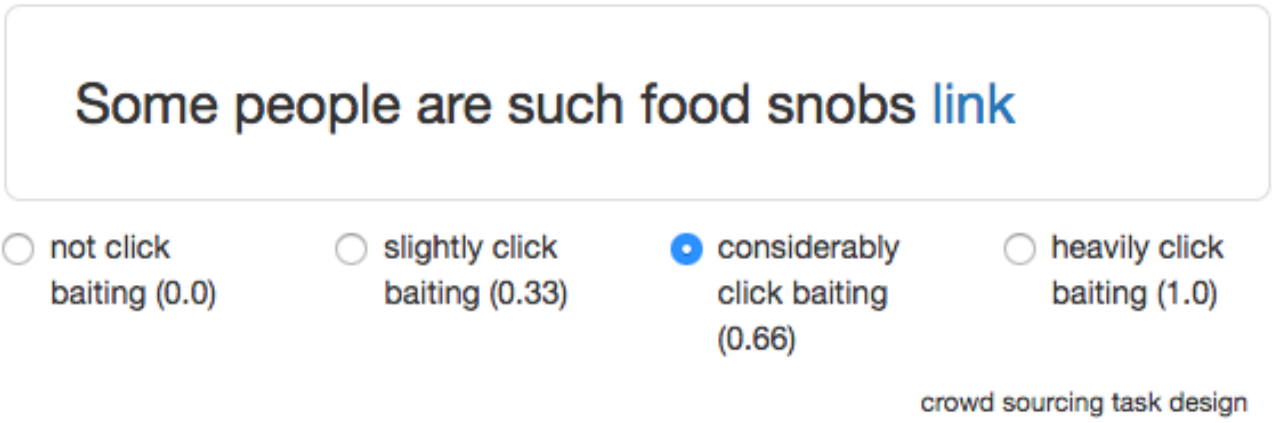

Figure 4: 4-point scale for crowdsourced annotators [4].

- $truthMedian$: The median of the scores given in $truthJudgements$
- $truthMode$: The mode of the scores given in the $truthJudgements$
- $truthClass$: Whether a post is "clickbait" or "no-clickbait" (class attribute)

Figure 2 is an example of the $postText$, or teaser message from the dataset. It is important to note that the accompanying link is not part of the $postText$ and is not something we will analyze in this paper. Figure 3 is an example of the $targetTitle$ of a post – it is the article title of the linked article in the $postText$.

The $truthJudgements$ attribute is made up of crowdsourced clickbaiting scores provided by five annotators on a 4-point scale as illustrated in Figure 4. The $truthClass$ is determined by the $truthMean$ – if the mean score of the five annotators is greater than or equal to 0.5, the $truthClass$ is "clickbait", otherwise "no-clickbait".

Out of the 2451 instances, 762 are "clickbait". Some examples from the dataset include:

- What India's microloan meltdown taught one entrepreneur
- 31 Accessories Every 90s Girl Will Recognize

The remaining 1697 instances are "no-clickbait". Examples from the dataset include:

- Prince Harry meets Lady Gaga at the Royal Albert Hall
- Apple debuts iOS 9: Battery enhancements, smarter Siri and more

Figure 5 illustrates the imbalance in the $truthClass$. We see that the number of "clickbait" instances is less than half the number of "no-clickbait" instances.

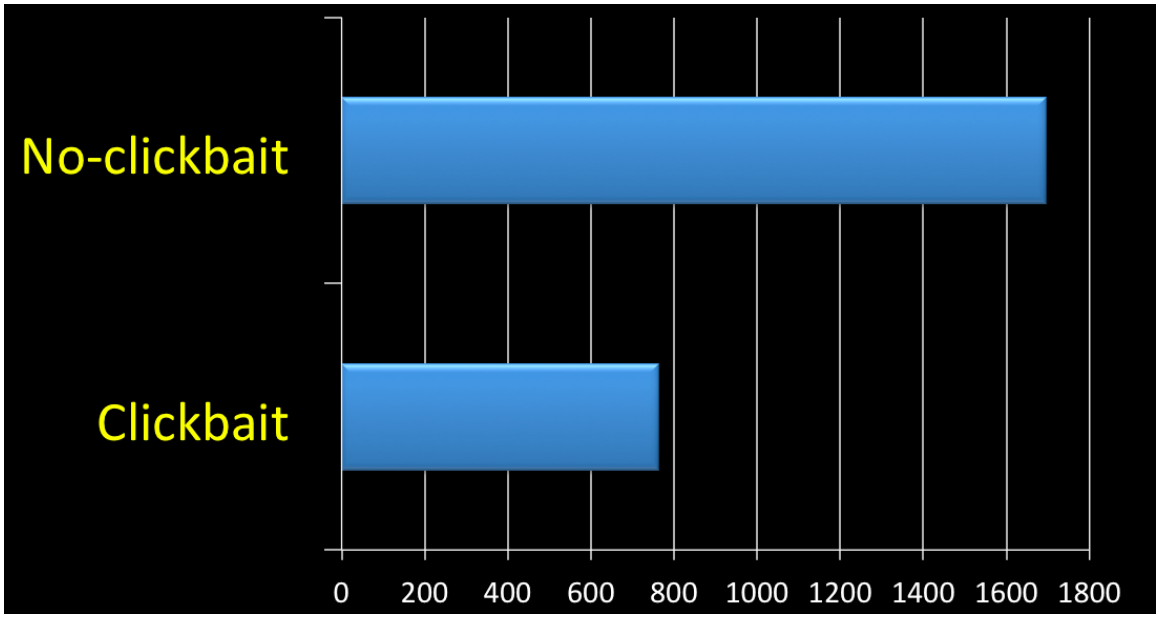

Figure 5: Class distribution in *clickbait17-train-170331* data.

The *clickbait17-train-170630* dataset is used to evaluate the performance of our clickbait detector. This dataset is also obtained from the "Clickbait Challenge 2017" [4] website and has the same attributes as the one just described, except that it comprises 19538 instances. Out of these, 4761 are "clickbait" and 14777 are "no-clickbait".

4 Preprocessing and Preliminary Results

Since the class attribute is not evenly distributed, the data was downsampled 70% to ensure that the classifier does not train itself on such a heavily biased dataset. The reason we downsampled to 70% is so that we do not repeat any "clickbait" instances in order to balance the dataset. This is because we want the classifier to familiarize itself with as many "clickbait" instances as possible while still being able to distinguish them from "no-clickbait" instances. The reason we choose to downsample instead of upsample is so that the "clickbait" instances are not repeated in order to balance the class distribution.

For preliminary experiments, we include the following attributes in our model:

- $postTimestamp$ (Nominal)
- $postText$ (String)
- $targetTitle$ (String)
- $truthClass$ (Nominal)

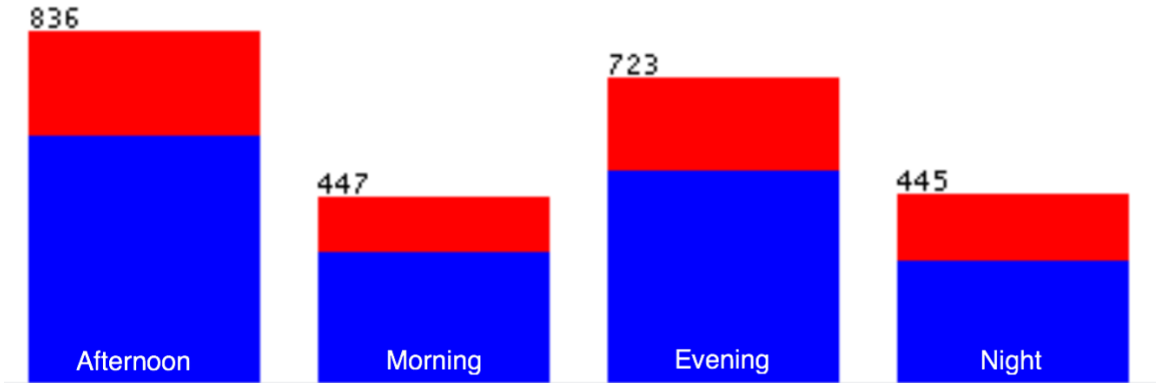

Figure 6: Distribution of *postTimestamp*. Red = *clickbait*, Blue = *no-clickbait*.

In each experiment, I assess the accuracy of the classifier in predicting clickbait by running a stratified 10-fold cross-validation [1]. In this technique, the data is randomly divided into 10 subsamples of equal size. 9 of these subsamples are used to train the model, while the remaining subsample is used to test the model. This process is repeated 10 times with every subsample acting as the testing data exactly once. Finally, the results obtained in each fold are averaged to obtain the classification accuray of the model. Stratification ensures that each fold contains roughly the same proportions of class labels [1].

These attributes required some preprocessing before any experiements could be run.

4.1 Discretizing *postTimestamp*

In its raw form, $postTimestamp$ gives us the exact time of each post normalized to GMT. The values in this attribute are too scattered and unique to run any meaningful analysis and so with Python code, they were divided into 6-hour bins as follows:

- Morning (6 am to 12 pm): 447 instances
- Afternoon (12 pm to 6 pm): 836 instances
- Evening (6 pm to 12 am): 723 instances
- Night (12 am to 6 am): 445 instances

Figure 6 is a graphical representation of the distribution of $postTimestamp$. The integers represent the number of instances in each 6-hour bin and we can use them to identify which bar represents what time of day. For example, the bar with 836 instances represents "Afternoon."

minTermFreq	**Classification Accuracy**
1	81.2814%
2	81.2814%
3	81.2814%
4	81.2814%
5	81.2814%
6	80.9045%
7	79.9623%
8	80.4648%
9	78.5804%
10	77.3869%

Table 1: Comparing classification accuracies with different *minTermFreq* values

4.2 Optimizing *StringToWordVector*

The unsupervised *StringToWordVector* filter converts String attributes into a set of attributes representing the information obtained from a word based on its occurrance and usage in the text we are analyzing. This is useful for text classification. The filter has multiple specifications including:

- *minTermFreq* – This sets the minimum frequency for a term to be included in a vector on a per-class basis.
- *outputWordCounts* – Output the word counts indicates the presence or absence of a word.
- *normalizeDocLength* – Whether the word frequencies for an instance should be normalized.
- *stemmer* – Sets the stemming algorithm to be used in order to reduce each word into its root form. For example, "stemming" would become "stem", "writing" would become "write", and so on.
- *stopwordsHandler* – Sets the stopwords handler to be used. Stopwords refer to the most common words used in a language, so for our case it includes words like "the", "an", "a", and so on.
- *tokenizer* – Sets the tokenizer algorithm to be used. Tokenizing breaks up a string into pieces (words, keywords, phrases, etc.) called "tokens". This is useful for analyzing the structure of the text.

I conducted experiments to analyze the effect of each of these specifications on the String attributes and to assess which values to assign to them. A Support Vector Machine (SVM) [3] is used as the classifier in this experiment. It is a supervised learning model that analyzes data and classifies it given the class attribute. Since we are analyzing word vectors, a SVM works well for our purpose. To optimize *StringToWordVector* I did the following:

outputWordCounts	**Classification Accuracy**
$False$	81.2814%
$True$	$82.2864\%^*$

Table 2: Analyzing *outputWordCounts*

normalizeDocLength	**Classification Accuracy**
No normalization	$82.2864\%^*$
Normalize all data	81.4698%
Normalize test data only	81.4698%

Table 3: Analyzing *normalizeDocLength*

- I analyzed the effect of *minTermFreq* by applying the filter with *minTermFreq* starting from 1, and going up to 10, running a 10-fold cross-validation. The results of this experiment are shown in Table 1. The classification accuracy remains the same between $0-5$. At 6, the accuracy reduces, but not statistically significantly. In fact, the only time the classification accuracy decreases significantly from 81.2814% is at 10. These results suggest that the *minTermFreq* setting affects classification accuracy and, for this data, better accuracies are achieved when it is set to $<= 5$. This is why I set *minTermFreq* to 5 for the rest of my experiments.

- After this, I shifted my attention to *outputWordCounts*. The default value is False. With *minTermFreq* still at 5, I ran a 10-fold cross-validation with the default value and with *outputWordCounts* $= True$. Table 2 describes the results. The classification accuracy increased to 82.2864% when *outputWordCounts* was set to $True$. This increase was statistically significant at the 95% confidence interval level (as indicated by the * symbol). Therefore, I keep *outputWordCounts* set to $True$ for the rest of the experiments.

- The default value for *normalizeDocLength* is "No normalization". I ran 10-fold cross-validations with the default value, as well as with "Normalize all data" and "Normalize test data only". Table 3 describes the results. The default setting achieved the highest classification accuracy and so I keep it at that for the rest of the experiments.

- I then ran 10-fold cross-validations using different *stemmer* values. The default value is *NullStemmer*, meaning no stemming. Table 4 indicates that none of the classification accuracies are statistically significantly different from one another. Therefore, I keep the default stemmer, *NullStemmer* for the rest of the experiments.

- I also ran 10-fold cross-validations using various *stopwordsHandler* values, the default being *Null*,

stemmer	**Classification Accuracy**
$NullStemmer$	82.2864%
$IteratedLovinsStemmer$	81.8467%
$LovinsStemmer$	81.0930%
$SnowballStemmer$	82.2864%

Table 4: Analyzing *stemmer*

stopwordsHandler	**Classification Accuracy**
$Null$	82.2864%
$MultiStopwords$	82.2864%
$Rainbow$	81.1558%
$RegExpFromFile$	82.2864%

Table 5: Analyzing *stopwordsHandler*

meaning no handling of stopwords. Table 5 indicates that, once again, none of the results are statistically significantly better than the others and so I keep the default value, *Null*.

- Finally, I analyzed *tokenizer* by running a 10-fold cross-validation using each tokenizer, the default being *WordTokenizer*. Table 6 describes the reaults of this experiment. Since, the default tokenizer is the only one that achieves a statistically significant classification accuracy, I keep it at that.

Figure 7 depicts the specifications of my optimized *StringToWordVector* filter.

4.3 Selecting attributes for the model

After preprocessing, I conducted experiments to guage the effect of each attribute on the model's classification accuracy. Again, we use a SVM as the primary classification algorithm for its ability to classify word vectors.

In order to gauge the effect of removing an attribute from the model, we first run a 10-fold cross-validation on the complete model with all 4 attributes: *postText*, *targetTitle*, *postTimestamp*, and *truthClass* (class attribute). We then remove one of the attributes (never the class attribute) and run a 10-fold cross-validation comparing the classification accuracy achieved to that of the complete model. Unless there is

tokenizer	**Classification Accuracy**
$WordTokenizer$	$82.2864\%^*$
$AlphabeticTokenizer$	81.3442%
$CharacterNGramTokenizer$	81.2814%
$NGramTokenizer$	75.6281%

Table 6: Analyzing *tokenizer*

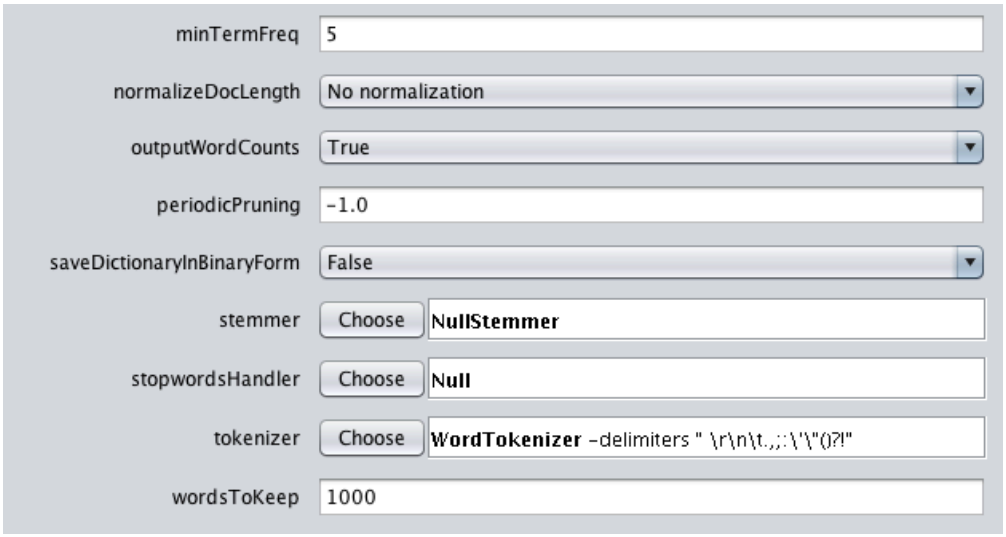

Figure 7: Specifications of optimized *StringToWordVector*.

a statistically significant decrease in classification accuracy, we conclude that the attribute removed is not useful in predicting the class attribute. This is repeated so that each attribute is excluded from the model at least once (except the class attribute). Results of these experiments were:

- I removed *targetTitle* from the model and ran a 10-fold cross-validation. The classification accuracy dropped statistically significantly to 77.0101% and so I include *targetTitle* in the model.
- I removed *postText* from the model and ran a 10-fold cross-validation. The classification accuracy drops statistically significantly to 80.1508% and so I include *postText* in the model.
- I first removed *postTimestamp* from the model and ran a 10-fold cross-validation. The classification accuracy decreases to 82.1608%. This decrease is not statistically significant. This is largely because all the timestamps are normalized to GMT, whereas all these posts were not made from the same location, which means that the $postTimestamp$ does not accurately represent the time of a given post. Therefore, I exclude *postTimestamp* from the model.

Therefore, 3 of the attributes selected for preliminary experimentation proved to affect the classifier's ability to predict whether a post is clickbait. The model achieves the highest statistically significant classification accuracy when $postText$, $targetTitle$, and $truthClass$ are included.

4.4 Selecting a classification algorithm

Now that we have optimized our filters and the model, we shift our attention to finding the best classifier. We use *ZeroR* as our baseline predictor. *ZeroR* always chooses the majority class. This would have been

Classification Algorithm	Classification Accuracy	ROC–AUC
$ZeroR$	50.0%	0.503
$J48$	$76.3819\%^*$	0.802
$LibSVM$	$82.1608\%^*$	0.823
$RandomForest$	$86.3065\%^*$	0.945

Table 7: Comparing accuracies of various classification algorithms.

"no-clickbait", but since we have balanced the class distribution, we expect a classification accuracy of roughly 50%. Our goal is to find classification algorithms that perform statistically significantly better than our baseline classifier, *ZeroR*.

The algorithms I analyze in the preliminary stages are:

- *LibSVM* – Library for support vector classification that supports multi-class classification [3]. We have already talked about SVMs in Section 4.2.
- *J48* – Java implementation for generating a pruned or unpruned *C4* decision tree
- *RandomForest* – Classification algorithm that constructs a forest of random decision trees

Table 7 describes the results of the experiments conducted on each algorithm. All the algorithms perform statistically significantly better than the baseline classifier, *ZeroR*.

I used ROC–AUC as the evaluation metric to evaluate the performance of my classifier as in "Clickbait Detection" [7] described in Section 2. Its values range from $0 - 1$ and the closer to 1, the better the classifier performs. It is a measure of how well our classifier can distinguish between the two classes.

- *J48* achieved an ROC–AUC of 0.802. This indicates that *J48* performs fairly well [9].
- *LibSVM* achieved an ROC–AUC of 0.823. This indicates that *LibSVM*'s performance is good [9]. It performs statistically significantly better than *J48*.
- *RandomForest* performs the best achieving an ROC–AUC of 0.945. This indicates that *RandomForest* performs excellently [9].

Therefore, *RandomForest* performs better than the other algorithms and so we use this as our classification algorithm for the remainder of our experiments. We also set our new baseline to 86.3065%. For every experiment, we check to see if the classification accuracy achieved is statistically significantly better than this baseline.

5 Adding Clickbait Indentification Features

In order to improve the classifier's performance I came up with 25 features that I thought would be useful for clickbait indentification based on intuition and the background work described in Section 2. As expected, not all of them were useful. In this section I will discuss the features that turned out to be useful for clickbait detection and some of the features that were not useful. The 25 features I came up with were:

1. The length of the longest word in $postText$.
2. The number of tokens in $postText$.
3. The number of words in $postText$.
4. The number of proper nouns in $postText$.
5. The number of prepositions in $postText$.
6. The number of subordinate conjunctions in $postText$.
7. The number of personal pronouns in $postText$.
8. The number of determiners in $postText$.
9. The number of possessive endings (apostrophe) in $postText$.
10. The number of Wh-pronouns in $postText$.
11. The number of Wh-adverbs in $postText$.
12. The number of overlapping words between $postText$ and $targetKeywords$.
13. The number of overlapping words between $postText$ and $targetTitle$.
14. Whether $postText$ begins with "Who", "What", "When", "Where", "Why", "Which", or "How".
15. Whether $postText$ begins with a number.
16. Whether $postText$ has a number at all.
17. Whether $postText$ has a superlative.
18. Whether $postText$ has a question mark ("?").
19. Whether $postText$ has a 2-gram $Noun + Verb$ pattern.

20. Whether $postText$ has a 2-gram $Preposition + Noun$ pattern.

21. Whether $postText$ has a 2-gram $SubordinateConjunction + Noun$ pattern.

22. Whether $postText$ has a 2-gram $Determiner + Noun$ pattern.

23. Whether $postText$ has a 2-gram $PersonalPronoun + Verb$ pattern.

24. Whether $postText$ has a $Number + NounPhrase + Verb$ pattern.

25. The likelihood that the n-gram parts-of-speech pattern appears in $postText$ of "clickbait" instances.

It is important to note that "2-gram" refers to a two-word continuous sequence in the text. Out of these features, 3, 13, and 25 turned out to be useful additons to the model. They are described in greater detail in Section 5.2. Section 5.1 describes some interesting features that did not improve the model. These features were implemented using Python's NLTK library [5]. I used the StanfordCoreNLP tagger [8] to tag and analyze the parts-of-speech of $postText$ and $targetTitle$.

5.1 "Failed" Features

As mentioned eariler, 22 out of the 25 features failed to improve the model's performance. In other words, adding these features to the model did not statsitcially significantly improve the classification accuracy with respect to our baseline – 86.3065%. This section outlines some of the interesting "failed" features.

5.1.1 *hasSuperlative*

This feature utilizes the StanfordCoreNLP parts-of-speech tagger to analyze the $postText$ and indicates whether it contains at least one superlative. The intuition behind this feature is that superlatives can be used to exaggerate the degree to which a statement is true without providing any mathematical justification and so the use of superlatives could be indicative of clickbait. For example:

- The *greatest* Italian dishes of all time! $< link >$
- Here is the *worst* way to spend your weekend. $< link >$
- Top 10 *most* annoying nicknames for pets. $< link >$

Table 8 illustrates the distribution of this feature in the *clickbait17-train-170331* dataset. The poor representation of this feature in "clickbait" instances (Only 60 out of 762) could be one of the reasons why this feature did not turn out to be useful.

hasSuperlative	# Clickbait	# No-clickbait
Yes	60	87
No	702	1610

Table 8: Distribution of *hasSuperlative* feature

hasNum	# Clickbait	# No-clickbait
Yes	191	416
No	571	1281

Table 9: Distribution of *hasNum* feature

5.1.2 *hasNum*

This is also a binary (Yes/No) feature which indicates whether the $postText$ contains at least one number or not. Intuitively, clickbait posts tend to use numbers as pseudo statistics in order to lure readers into clicking the accompanying link. Therefore, the use of numbers in the $postText$ could be indicative of clickbait. For example:

- *10* ways to get a raise in your salary. $< link >$
- Top *five* holiday destinations within your budget. $< link >$
- *20* shocking images you won't believe are real! $< link >$

Table 9 shows the distribution of this feature in the *clickbait17-train-170331* dataset. The representation of this feature in "clickbait" instances is higher than *hasSuperlative*, but it is still low, specially given that the class distribution in the dataset is already heavily imbalanced. In fact, unlike *hasSuperlative*, the representation of this feature in "no-clickbait" instances is much higher than "clickbait" instances. This means that, given the data, it is more likely that a post containing a number will be "no-clickbait", which could explain why it does not positively contribute to the model.

5.2 Useful Features

3 out of the 25 features proved to be useful for our model. This means, including these features in the model achieved the highest classification accuracy. Removing either one of these features brings the classifcation accuracy down statistically significantly. So, while each of these features individually statistically significantly improves the model's accuracy compared to the baseline of 86.3065%, the model performs the best when all them are included.

Overall Mean	Clickbait Mean	No-clickbait Mean
12.662	11.901	13.002

Table 10: Mean number of words in $postText$ by class attribute

Figure 8: Illustration of *numOverTitle*.

5.2.1 *numWords*

This feature indicates the number of words contained in $postText$ (which excludes the accompanying link). Intuitively, clickbait posts are short and snappy in the hope that they will catch the readers' attention. Therefore, we expect a low number of words in $postText$ to be associated with "clickbait" instances. Longer $postText$ could mean that less information is intentionally shielded as these types of $postText$ would be less ambiguous.

Table 10 illustrates the mean number of words in $postText$ by class attribute in the *clickbait17-train-170331* dataset. As expected, "clickbait" instances have a lower mean number of words in $postText$ than "no-clickbait" instances. The results of adding this feature are discussed in Section 6.

5.2.2 *numOverTitle*

This feature measures the similarity between $postText$ and $targetTitle$. The intuition behind this is that the more similar these two attributes are, the more likely it is that $postText$ accurately reflects $targetTitle$ and the more likely it is that the post is not clickbait. Therefore, we measure the number of overlapping words – the number of words in $postText$ that are also in $targetTitle$. In the examples illustrated in Figure 8, the number of overlaps is 4. This because, we treat "scientists" and "science" as an overlap because they come from the same root word, "science". This technique of breaking down every word to its root form is called 'stemming' and helps account for difference in tenses, singularity and plurality, and so on, giving us a more accurate picture of how similar $postText$ and $targetTitle$ truly are.

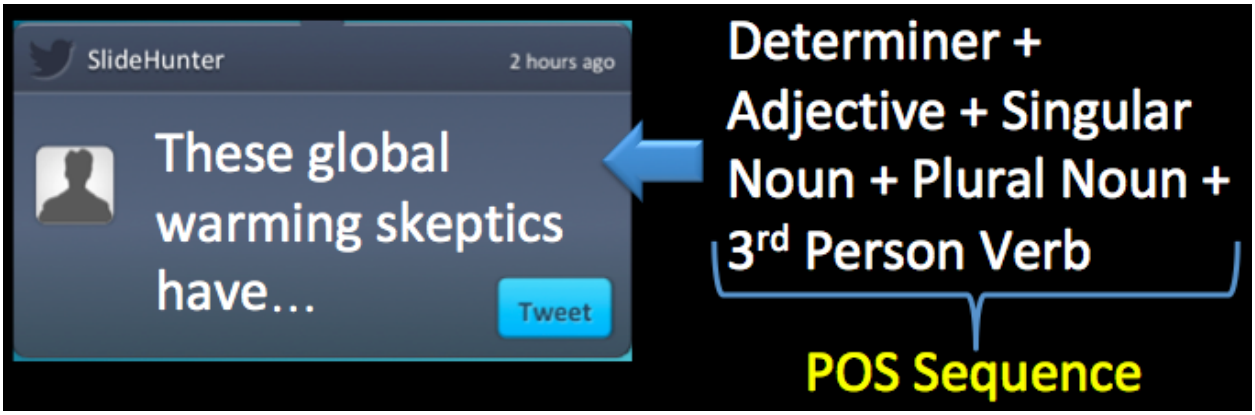

Figure 9: Example of a POS Sequence.

The mean number of overlaps between $postText$ and $targetTitle$ in the dataset is 3.997. For "clickbait" instances, the mean number of overlaps is 3.150, and for "no-clickbait" instances, the mean number of overlaps is 4.378. There are a few instances where the $postText$ and $targetTitle$ are the same, which makes sense since a post is often made by the author of the article. These two being the same does not tell us anything regarding the class label of this post – that is, it is not the case that these two being the same makes it more or less likely for a post to be clickbait. The results of adding this feature to the model are described in Section 6.

5.2.3 *posRatio*

This was the most useful feature that was implented in the model. It measures the likelihood that a given post's parts-of-speech (POS) sequence appears in "clickbait" instances. A POS sequence is the sequence of parts-of-speech tags of the words in $postText$. Figure 9 shows us an example of a parts-of-speech sequence. The intuition behind this feature is that the more likely it is for a given POS sequence to appear in "clickbait" instances, the higher the chances that that given post is clickbait. The likelihood of a POS seqence appearing in "clickbait" instances (called $posRatio$) is measured by the following formula:

posRatio = (#clickbait) / (#clickbait + (#no-clickbait/overallRatio))

where,

- #clickbait = the number of times a POS sequence appears in "clickbait" instances,
- #no-clickbait = the number of times a POS sequence appears in "no-clickbait" instances,
- overallRatio = Total number of "no-clickbait" instances in dataset divided by the total number of "clickbait" instances.

Attributes Included	Classification Accuracy
$postText + targetTitle + numWords + numOverTitle$	82.2864%
$postText + targetTitle + posRatio$	86.6860%
$postText + targetTitle + numWords + numOverTitle + posRatio$	88.2051%

Table 11: Results of adding features to model

The reason we divide the "no-clickbait" instances by the overallRatio is to account for the difference in class distribution. Intuitively, the higher the $posRatio$ of a post, the more likely it is that the post is clickbait. The posRatio of the first 5 words in the example illustrated in Figure 9 is 0.8698, indicating that it has a high chance of being clickbait. After running 1-fold cross-validations using the posRatio of the first $1, 2, 3, 4, 5, 6$ words in $postText$, we conclude that the feature works best when analyzing the first 5 words in $postText$. This could be because analyzing more than the first 5 words makes the POS sequence too specific to that particular instance, whereas the POS sequence of less than 5 words is not meaningful enough.

The most common POS sequence in the "clickbait" instances of the data set was *Cardinal Number + Adjective + Noun (plural) + To + Verb*. For example, "10 crazy ways to cure cancer!" falls under this POS sequence. Not surprisingly, the posRatio of these instances was 1. The results of adding this feature to the dataset is described in Section 6.

6 Results

Now that we have identified features useful for clickbait detection, we evaluate our model on the bigger *clickbait17-train-170630* dataset, described in Section 3. The reason we do this is to test how well our classifier performs on unseen data. Therefore, our posRatio is calculated based on the POS sequences observed in the smaller dataset, which means that there will be missing values since there are POS sequences in the bigger dataset that we have not yet seen.

We start with our baseline as $ZeroR$'s classification accuracy – 75.8553%. Figure **??** shows us the results of adding these features to our model. All of these accuracies are statistcially significantly better than our baseline. Moreso, each subsequent accuracy is statisically significantly better than the previous. In other words, simply adding posRatio to the model gives us a statistcally significantly higher accuracy than adding $numWords$ and $numOverTitle$. Our model performs the best when all these features are added, giving us a classification accuracy of 88.2051%.

```
Correctly Classified Instances        8585               88.2051 %
Incorrectly Classified Instances      1148               11.7949 %
Kappa statistic                          0.6305
Mean absolute error                      0.2012
Root mean squared error                  0.2965
Relative absolute error                 54.9362 %
Root relative squared error             69.2725 %
Total Number of Instances             9733

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.982    0.433    0.877      0.982    0.927      0.658    0.928     0.973     no-clickbait
                 0.567    0.018    0.911      0.567    0.699      0.658    0.928     0.850     clickbait
Weighted Avg.    0.882    0.333    0.885      0.882    0.872      0.658    0.928     0.943

=== Confusion Matrix ===

    a    b   <-- classified as
 7252  131 |    a = no-clickbait
 1017 1333 |    b = clickbait
```

Figure 10: Detailed performance analysis of model on unseen data.

7 Conclusion and Future Work

Figure 10 shows us an analysis of our classifier's performance on unseen data. From it, we can conclude that our classifier performs better than the one presented by Martin Potthast, Sebastian Köpsel, Benno Stein and Matthias Hagen [7], acheiving an ROC–AUC of 0.928 for "clickbait" instances. From the confusion matrix we see that only 131 "no-clickbait" instances were wrongly classified as "clickbait", which is good because we would rather wrongly classify "clickbait" instances as "no-clickbait" than dismiss a legitimate article as clickbait.

The winners of the clickbait challenge achieved a classification accuracy of 85.6%. Our classifier performs better, however, these two accuracies were evaluated on different datasets so they are not directly comparable.

In conclusion, we can say that we have successfully identified and implemented features that are useful for clickbait detection. Our classifier performs very well compared to past machine learning attempts, including participants of the "Clickbait Challenge 2017" [4].

As future work, it would be interesting to analyze the images in each article. It would also be interesting to analyze how many ads are present on the linked webpage since clickbait is normally used to increase viewrship on webpages in order to earn money from advertisements. Therefore, the number of ads would be a strong indicator of whether a post is clickbait.

References

[1] *10-fold Cross-validation*. URL: https://www.openml.org/a/estimation-procedures/1.

[2] Jonas Nygaard Blom and Kenneth Reinecke Hansen. "Click bait: Forward-reference as lure in online news headlines". In: *Journal of Pragmatics* 76 (2015), pp. 87 –100. ISSN: 0378-2166. DOI: https://doi.org/10.1016/j.pragma.2014.11.010. URL: https://www.sciencedirect.com/science/article/pii/S0378216614002410.

[3] Chih-Chung Chang and Chih-Jen Lin. "LIBSVM: A library for support vector machines". In: *ACM Transactions on Intelligent Systems and Technology* 2 (3 2011). Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm, 27:1–27:27.

[4] *Clickbait Challenge 2017*. http://www.clickbait-challenge.org/. Accessed: 06/07/2017. URL: http://www.clickbait-challenge.org/.

[5] *Natural Language Toolkit*. Accessed: 03/23/2018. URL: https://www.nltk.org/.

[6] *News Feed FYI: Further Reducing Clickbait in Feed*. https://newsroom.fb.com/news/2016/08/news-feed-fyi-further-reducing-clickbait-in-feed/. Accessed: 06/07/2017. URL: https://newsroom.fb.com/news/2016/08/news-feed-fyi-further-reducing-clickbait-in-feed/.

[7] Martin Potthast et al. "Clickbait Detection". In: *Advances in Information Retrieval. 38th European Conference on IR Research (ECIR 16)*. Ed. by Nicola Ferro et al. Vol. 9626. Lecture Notes in Computer Science. Berlin Heidelberg New York: Springer, 2016, pp. 810–817. DOI: http://dx.doi.org/10.1007/978-3-319-30671-1_72.

[8] *Stanford CoreNLP - Natural Language Software*. Accessed: 03/23/2018. URL: https://stanfordnlp.github.io/CoreNLP/.

[9] *The Area Under an ROC Curve*. URL: http://gim.unmc.edu/dxtests/roc3.htm.

[10] Katy Waldman. *Mind the 'curiosity gap': How can Upworthy be 'noble' and right when its clickbait headlines feel so wrong?* 2014.