Lucino Chiafullo
Arjuna Mahenthiran
Molly Moran
Samantha Richards

LING 131A Final Project Write-Up

Background and Motivation

The advent of internet-based news sources has brought "clickbait" to the fore, a phenomenon defined as 'content whose main purpose is to attract attention and encourage visitors to click on a link'. We see this effect most often in headlines, which normally double as the link from an external source to an article. Clickbait-style journalism is most often associated with internet-media giants like Buzzfeed, but is also an increasingly common tactic used by more established news sources. Our aim with this project was to build a classifier that could perform two separate but related functions: classify a given headline as "bait or "not bait", and return the percentage of headlines of a given news source that are classified as "bait".

This may be of particular use to news publications interested in determining the makeup of their headlines. Our program does not involve any document-level classification - we focus solely on the content of headlines - and makes no judgment about the quality of content contained in articles with clickbait titles. However, the analysis we perform may serve as a launching point for investigating the relationship between clickbait-style journalism and overall content quality in terms of informativeness, bias, etc. This may offer downstream information extraction applications (such as filtering out clickbait from a corpus of news articles.)

Data Collection and Labeling

When it came to training our classifiers, we unfortunately had no gold-standard data to work with; we were unable to find a freely-available corpus of headlines pre-classified as clickbait/ not-clickbait. The lack of any such resource, however, confirmed for us that our project was worth undertaking. Given these circumstances, we had to build and label our own set of training data.

Our first instinct was to train the classifiers to recognize clickbait using headlines from Buzzfeed. This wasn't, however, an entirely subjective choice: we consulted many sources on the phenomenon of clickbait, all of which seemed to confirm that Buzzfeed is considered something of a "gold-standard" of the journalistic style. Stylistically, Buzzfeed headlines are written in a very distinctive way that let us justify labeling this set of headlines as clickbait - most, if not all, were prime examples of the phenomenon to be classified.

Training the classifiers to recognize "non-clickbait", however, was trickier. Recognizing the use of particular journalistic tactics is one thing; recognizing the *lack* of such tactics is another beast entirely. We ended up settling on Reuters and Associated Press as the best resources for this task for several reasons. Clickbait has mostly risen in a pay-per-click journalistic landscape, so Associated Press, as a not-for-profit news agency, is far less likely to

employ these tactics. Reuters is universally known for its self-described "value-neutral approach".

We considered using NLTK's built-in Reuters corpus, but quickly decided that it did not properly meet our needs - we were looking solely at headlines, whereas the Reuters corpus is built more for document-level classification. We decided instead, as discussed, to rely exclusively on NewsAPI (newsapi.org) as a source to pull recent news data; NewsAPi provided an easy way to get a large data set in bulk, granting us access to real-time news content from 30,000 different news sources around the world. We collect a month's worth of headlines (between 100 and 1000) from Buzzfeed, Reuters, and Associated Press for our training and test data.

We wrote the file *data_collector.py* to create this list of headlines; this calls the API on a number of sources and labels all headlines from Buzzfeed News as clickbait, and adds these headlines to a list of clickbait headlines. Afterwards, it labels headlines from Reuters and AP as non-clickbait, then adds these to a list of such headlines. At the end of the central function in this file, the lists are combined and pickled, so that the list of headlines can be accessed in various files, and the program need not call the API each time it runs.

Choice of Classifier

Our project contains two separate classifiers, which we use for different purposes. The first is NLTK's built-in Naive Bayes classifier, which we use to classify entire corpora of headlines from a particular news source. In terms of NLTK's basic built-in classifiers, we chose to use Naive Bayes over a Decision Tree for several reasons. First: feature extraction was something of a trial-and-error process. We didn't know exactly what we were looking for, nor did we know which features would end up being the most informative; using a naive classifier allowed us to observe and track the performance of every feature as we tested and updated our model, whereas a Decision Tree would have thrown out the least informative features. As we continued to update the model and check its most informative features, we saw many features vying for the top spot, so it was especially important that every feature mattered. Second: Decision Trees are much more likely to overfit, and we wanted to avoid falling into this trap as much as possible, especially given that we chose to train our classifier to recognize clickbait using only data from Buzzfeed.

The classifier was trained on the list of headlines gathered by the aforementioned data_collector file, which was about 3000 headlines long. This was not a perfect size, but was the largest possible given the limitations of the API we used for data collection. We were able to generate feature sets for each headline (detailed in a later section) and trained the classifier on roughly 90% of this dataset, saving the rest for testing. Throughout development, we noted the accuracy of the classifier given certain features and fine-tuned the functions that returned poor results. This process was fairly normal for classification.

Lucino Chiafullo
Arjuna Mahenthiran
Molly Moran
Samantha Richards

After determining that a Decision Tree may not be the best route for creating a second classifier, it became clear that more research had to be done in order to determine what the next best option might be. After deciding that a Maximum Entropy classifier may lead to too disorganized of data classifications, a Nearest Neighbors classifier was settled upon. We use the Nearest Neighbors classifier to classify individual headlines. A nearest neighbors classifier is trained by taking a collection of data values as a parameter, and then fitting each collection of values to its corresponding classification. For example, if the headline "Senate leader meets President of Moldova" should be classified as "not clickbait", then the classifier is created using the feature values of the headline as a parameter. In order to create this classifier, Sci-Kit Learn, a machine-learning package in Python, was utilized. Then, the fit() method is called on a collection containing "not bait", matching the data point in the first collection to that of the same index of the latter. The accuracy of the classifier can then be determined by calling the .score() method and passing in the same data used to create the classifier. This returns accuracy score of the classifier at classifying headlines, not unlike the .accuracy() method of a Naive Bayes classifier.

Just as with the .classify() method of a Naive Bayes classifier or Decision Tree, a Nearest Neighbors classifier can also be used to provide a classification given a headline. However, because the classifier takes in a collection of values as a parameter rather than a string, the string must first be converted into these values through a helper method before being passed into the .predict() method of the classifier. The classifier then uses the parameter of number of neighbors passed into it when it was created, and searches for this same number of data points within the training data that are closest to the data point of interest. Finally, it uses a majority vote of the classifications of these data points to determine what classification to assign the headline of interest. Critically, this means that there are two points at which a Nearest Neighbor classifier's accuracy can be manipulated: the feature set of extracted features, similar to a Naive Bayes classifier, and the number of neighbors evaluated when determining the classification of novel data. Because the former is already examined elsewhere in this document regarding Naive Bayes classifiers, a table of receive results when changing the number of neighbors of the classifier is provided below:

N (Neighbors)	Accuracy
1	0.8916797488226059
11	0.8951334379905809
51	0.8904238618524333
101	0.8737833594976452

Generally, as the number of neighbors increases, the accuracy the classifier decreases, as more potentially incorrectly labeled neighbors are considered. Furthermore, the program is significantly slower than a Naive Bayes classifier at providing a classification based on a string, and as such, is not to be used for classifying large numbers of headlines.

Object Oriented Approach

We initially implemented our classifier using a simple data structure consisting of a list of tuples, where each tuple was a headline string followed by a label ('bait' or 'not bait'). However, we found that many of functions that implemented the features were redundantly looping and re-looping over the headline to extract the same information using low-level token-based processing: namely, a list of tokens, a list of lowercase tokens, the number of tokens in the headline, a list of tuples containing the tokens and their POS tags, and a list of just POS tags. We therefore created the Headline class to store this information for each headline, and our procedure now begins with instantiating a list of headline objects which would calculate and store these values and data structures upon initialization, so that each function only has to access the attribute rather than looping through a data structure or calling a method. We found that this greatly eliminated redundancy in the code and decrease the runtime significantly (cutting it in half at the time).

Similarly, because the code required to pull headline data from NewsAPI was long and cumbersome, we found that creating Newscorpus objects was a cleaner and more efficient way to extract and store headline data for particular news sources. This came especially in handy given the way we designed our client code.

Client Code

We designed our main program to be an interactive way to test the percentage of clickbait headlines in a given news source, or to classify individual headlines as either clickbait or non-clickbait: the user can choose which functionality she wants to use. This was a way to introduce the classifier to unseen data: the option to classify individual headlines provides a quick and fun way for users to test the interface and get a sense of the program, while the option to classify a corpus of headlines from a given source accomplishes a potentially more useful task, measuring how often a source publishes stories with baiting headlines. For this functionality, a list of relevant sources is included as sources.txt in the repository, which includes the names of news sources in the format in which they must be entered for the program to work.

When the user runs *clickbaitoop.py*, a prompt appears asking which functionality they would like to try. Given an individual headline, the program will run the SKLearn Neighbors Classifier and return a prediction about the headline's label. Given a news source, the program will create a Newscorpus object based on the user's chosen source, and run the NLTK Naive Bayes classifier on the headlines of that Newscorpus object, returning a value corresponding to

the percentage of headlines in the corpus classified as clickbait. During testing, we looked at sources with high expected clickbait values such as The Huffington Post and Business Insider. Our classifier, however, returned values around 25% and 30%, respectively. We reasoned that these sources still have a comparably large amount of clickbait content as opposed to sources with comparably low percentages such as The Financial Times or Daily Mail, which hover around 10-15% clickbait. Interestingly, The New York Times has quite a high percentage of clickbait according to our classifier, comparable to the Huffington Post (in between 25 and 30% of headlines were tagged as 'bait').

These results suggest that while most sources are not pure examples of clickbait compared to Buzzfeed (which was assumed and documented previously), some can be characterized as 'more likely' to publish clickbait headlines (such as The Huffington Post or Business Insider).

It would be worthwhile to try to limit the sections of a given source to curtail potential clickbait classification (e.g., eliminate the editorial section from the list of headlines). This addition could reduce the percentage of clickbait from sources that claim to avoid the common clickbait style. Perhaps, however, these results demonstrate that the digital age has driven most publishers to produce content in a certain style to generate revenue.

Feature Extraction

One of the main challenges we faced was developing a guiding principle for feature extraction. At first glance, classifying clickbait seems like a fairly straightforward task: we are all familiar with some of the catchphrases and flagwords frequently associated with clickbait headlines: ("You won't believe…", "...will surprise you.", "Here's why…", etc.) If we were to simply search headlines for a predetermined class of frequent n-grams, and classify them based on the presence or absence of those n-grams, the project would involve little to no analysis. To make the task more challenging, we began by avoiding relying on specific lexical items as flags, focusing instead on higher-level syntactic and stylistic features to see what we could glean from a more big-picture analysis. Our bigrams feature extracts the 10 most frequent bigrams from our training dataset, and serves as a baseline. Working with only the most frequent bigrams, our classifier's baseline accuracy was around 79%. From there, we worked on extracting various additional features to improve the classifier's accuracy.

Several of our features are fairly self-explanatory: procount and wh scan a headline for the use of first- and second-person pronouns and wh-words, respectively - two common tactics used to make the reader feel personally engaged and incite curiosity - and both return True if any are found. Startswithnum checks to see whether the first tag in a headline corresponds to a cardinal digit: this feature allowed us to isolate articles in "listicle" format and classify them as clickbait. Superlative scans the POS-tagged headline to check for instances of superlative adjectives, which are commonly used in hyperbolic journalism. Punct calculates the number of sentence-ending punctuations marks found within the headline; we found that, typically,

clickbait headlines use a punchy style, often including several short sentences in a single headline. Initially, this feature was designed to return a count; however, during testing, we discovered that any number of sentence-ending punctuation greater than 0 was highly correlated with clickbait, and so we altered the feature to return True if the count was greater than 0.

The mostcommontag feature was originally designed to scan the POS-tagged headline and return the most frequent part of speech occurring in the headline. However, once again, through testing we discovered that only one particular most common tag was showing up again and again under our most informative features: 'NN'. As a result, we edited the mostcommontag feature to simply return True if a headline's most common tag is a noun. Headlines whose most frequent tag was a noun were highly likely to be classified as non-clickbait. This result initially surprised us; however, after some reflection it became clear that a headline whose most common tag is a noun is likely contentful and informative, because it makes reference to more discrete entities. Clickbait is known for its vague nature: less informative than imperative.

Stemming from this observation, we then developed the imperative feature, which returns true if the headline begins with a bare-form verb: a good indication that the headline begins with an *imperative*. This turned out to be another reliable feature for predicting clickbait, which makes sense, as clickbait headlines are often unsubtle calls to action: *Watch, check out, see, find out...*etc.

Previous investigations into clickbait headlines have revealed that this phenomenon tends to involve more words of shorter length than "non-clickbait" counterparts. As such, it seemed important that the average word length of the headline was an important feature for our classifier to take into account. First, it was necessary to not count words that tend to appear over and over again in headlines, regardless of whether they are "clickbait" or not. This way, only the average word length of the more important content words was calculated. Then, rather than return an integer of the average word length within the title, a boolean of whether the average word length is above or below 4 characters is returned. 4 is used because according to Chakraborty et al., clickbait headlines have an average word length of 4.5 characters. However, this feature was found to not be extremely helpful, contributing among the least of all our features.

Relatedly, we also hypothesized that non-clickbait headlines would be more likely to contain a higher proportion of "unknown" or at least rare words, as a function of use of a broader vocabulary overall, reflecting the likely more educated readership and broader range of subjects. We thus implemented the feature rare_words, which gathered a list of the 50 most common words from the training corpus of news headlines (both bait and non-bait) and then sorted words in the focal headline into a "rare word" list if they were *not* found on this list of common words. Initially, we returned the count of rare words as a feature, but found that if the count of rare words was above 17, this proved just as useful for determining whether the headline was bait. Interestingly, however, this relationship was the opposite direction from what we had

hypothesized: headlines with greater than 17 rare words were much *more* likely to be clickbait (around 16:1).

The function_words feature takes advantage of nltk's list of English stopwords, creating a list of words in a given headline that are also in the 'stopwords' list, the titular 'function words'. Initially, we designed this feature as proportion of words in a headline that were function words, but after testing the feature as a ratio we found that the highest accuracy was contributed by the feature when its value was exactly .5 (that is, when half of the words in the headline were from the stopwords list). To capture this, the feature was converted into a boolean returning True if half of the words in the headline are on the stopword list. Since its implementation, this feature has frequently topped the list of informative features. However, interestingly, the overall accuracy of the classifier seems to be slightly higher *without* function_words, perhaps due its interdependence with another more informative feature (such as bigrams). We decided to remove function_words for this reason, and it is commented out in the code. One can see below screenshots of three different versions of the accuracy statistics: before implementing function_words, implementing funcion_words as a double (a ratio), and implementing function_words as a boolean:

```
/usr/local/bin/python3.7 /Users/samantharichards/Documents/Git/nlp-final-slam/clickbaitoop.py
0.9402061855670103
Most Informative Features
                bigrams = True              bait : not_ba =     67.0 : 1.0
           mostcommontag = True           not_ba : bait   =     30.8 : 1.0
                procount = True              bait : not_ba =     29.5 : 1.0
           startswithnum = True              bait : not_ba =     17.7 : 1.0
              rare_words = True              bait : not_ba =     16.6 : 1.0
              flag_words = True              bait : not_ba =     13.9 : 1.0
                      wh = True              bait : not_ba =     10.1 : 1.0
      averagewordlength = True              bait : not_ba =      6.7 : 1.0
                  punct = True              bait : not_ba =      5.0 : 1.0
               procount = False          not_ba : bait   =      2.1 : 1.0

Process finished with exit code 0
```

```
/usr/local/bin/python3.7 /Users/samantharichards/Documents/Git/nlp-final-slam/clickbaitoop.py
0.9278350515463918
Most Informative Features
          function_words = True              bait : not_ba =    106.2 : 1.0
                 bigrams = True              bait : not_ba =     71.9 : 1.0
            mostcommontag = True           not_ba : bait   =     30.6 : 1.0
                 procount = True              bait : not_ba =     27.6 : 1.0
            startswithnum = True              bait : not_ba =     21.5 : 1.0
               rare_words = True              bait : not_ba =     17.4 : 1.0
               flag_words = True              bait : not_ba =     16.4 : 1.0
                       wh = True              bait : not_ba =     10.0 : 1.0
       averagewordlength = True              bait : not_ba =      7.6 : 1.0
                   punct = True              bait : not_ba =      4.6 : 1.0

Process finished with exit code 0
```

Lucino Chiafullo
Arjuna Mahenthiran
Molly Moran
Samantha Richards

```
/usr/local/bin/python3.7 /Users/samantharichards/Documents/Git/nlp-final-slam/clickbaitoop.py
0.931958762886598
Most Informative Features
                 bigrams = True              bait : not_ba =     75.4 : 1.0
          function_words = 0.5               bait : not_ba =     63.8 : 1.0
          function_words = 0.0            not_ba : bait   =     47.3 : 1.0
           mostcommontag = True           not_ba : bait   =     43.9 : 1.0
          function_words = 0.375             bait : not_ba =     38.4 : 1.0
          function_words = 0.4               bait : not_ba =     34.4 : 1.0
          function_words = 0.1111111111111111 not_ba : bait   =     28.9 : 1.0
                procount = True              bait : not_ba =     27.5 : 1.0
           startswithnum = True              bait : not_ba =     22.5 : 1.0
          function_words = 0.42857142857142855   bait : not_ba =     22.4 : 1.0

Process finished with exit code 0
```

Finally, although we were able to capture many general patterns in the data using the features described above, we also made a return to the basics, asking whether those certain words which, as human observers, ‘flagged’ a headline as very likely to be clickbait were already captured by our other features, or were in fact not. A feature flag_words was added so that the appearance of any one of a small list of words, [‘this’, ‘these’, ‘will’, ‘ll’, ‘believe’, and ‘surprise’] would trip the feature to return True. This feature proved to indeed contribute a small weight to the overall accuracy, classifying clickbait accurately at a rate of 16:1, on par with our rare_words feature, though as predicted, vastly less informative than more sophisticated measures such as bigrams.

Figure 1 below illustrates the classifier’s accuracy using various subsets of features:

Subset	Accuracy	Most informative Feature
Baseline (Bigrams only)	79.2	(bigrams)
Complement of Baseline	92	function_words
Non-Lexical	74.4	mostcommontag
Lexical	89.9	bigrams
All	94.2	bigrams

Testing the classifier on all of our features *besides* the bigram scanner (the feature subset labeled “complement of baseline”), we were pleasantly surprised to find that our accuracy was about 92% - a great improvement from our baseline accuracy using only bigrams, which settled in around 79%. This confirmed the suspicion we had entering the project: that the most telltale

features of clickbait-style journalism are more complex than simply the inclusion of particular phrases.

We also tested the classifier on two other subsets of features - what we called “lexical” and “non-lexical” - to get a better sense of where the most informative features lie. The “lexical” features included all those that searched for particular tokens in the headline - this included procount, wh, and flagwords. The “non-lexical” features were those that represented a level up in abstraction - this included any features that scanned for particular parts of speech, calculated word lengths or ratios of tokens, and counted punctuation (although punctuation marks are technically tokens, we considered this type of feature more of a syntactic concern than a lexical one.) When we tested the classifier solely on the lexical subset, the accuracy was higher than when we tested the classifier on the non-lexical subset. This is to be expected, as totally ignoring lexical choice leaves out a lot of crucial information.

Ultimately, however, we achieved our highest accuracy rates when using all of our features (with the exception of functionwords, which did not make the cut for reasons explained above.)

Looking to the Future

As linguists, we are aware that language is ever-changing. However, combined with the advent of new technologies that increase the speed of discourse, some might argue that this change is happening at an pace never before seen. The “clickbait-ification” of headlines over time is but one example of this. The value in technologies such as the classifiers we have created, then, lies in their ability to accurately and reliably discern when news headlines are maybe overly-hyperbolic simply as a means of getting attention, or when news headlines are appropriately titled.

For writers and journalists, our classifier may be a useful tool in determining whether their headlines are out-of-sync with their content, or are likely to convey a different kind of content than desired. For consumers of news, this technology could be a preliminary boon in helping to disambiguate which news sources may be more reliable or honest than others.

Works Cited

Chakraborty, Abhijnan & Paranjape, Bhargavi & Kakarla, Sourya & Ganguly, Niloy. (2016). Stop Clickbait: Detecting and Preventing Clickbaits in Online News Media.

Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.