# A Simple Approach to Clickbait Detection

Jugoslav Stojcheski
stojcheski.jugoslav@students.finki.ukim.mk

Ss. Cyril and Methodius University
Faculty of Computer Science and Engineering
Skopje, Republic of Macedonia

30.09.2018

**Abstract**

Nowadays, some of the biggest social networks/platforms already "declared war" against those posts that decreases the reader's/viewer's enjoyment of using their products by distracting them. Those posts are called clickbaits and their purpose is to catch the attention of the readers/viewers and to drag them to the website to which the clickbaits lead.

My primary motivation to try to detect clickbait headlines was the reason that I was (and probably still am) their "victim". In this article, I propose some simple approaches to successfully detect clickbait headlines. The classifier that achieved the highest level of accuracy overperformed each classifier proposed at the Clickbait Challenge 2017 workshop. How I've done that? As the best clickbait creators would say: "This man found a MAGICAL approach to detect clickbait headlines and his solution will SHOCK you!".

# Contents

# 1 Introduction & Motivation

We live in an era in which a lot people (are trying to) make money on-line on the Web. There are a lot of people running their own websites in order to sell their advertisement space either directly by promoting a product or indirectly by having banner advertisements or sponsored content. Since they get paid based on the number of clicks/views the advertisements get, a completely expected goal would be to maximize the number of clicks/views (customers/consumers). The quest for a method to attract more consumers sometimes lead the creators of these websites into "dirty" business practices. Writing clickbait articles is the technique to gain people's attention by exploiting particular human weaknesses (such as curiosity), since it is extremely predictable what most people will be curious about and would want to know more about, or what will activate some person's "fear of missing out" feeling.

Clickbait is a website link designed to entice users to go to a certain web page or video. Clickbait headlines typically aim to exploit the "curiosity gap", providing just enough information to make readers of news websites curious, but not enough to satisfy their curiosity without clicking through to the linked content. Most of the clickbait headlines and articles present little or no legitimate well-researched news and instead use eye-catching headlines that include exaggerations of news events (using controversial topics), scandal-mongering, promise of big reward, emotionally charged headlines or sensationalism. Clickbait headlines deliver non-valuable or low-quality information, which leads to a journalism of low quality and suspicion over the reliability of the media.

In the next sections, I propose some machine learning methods for clickbait detection only by examining the headlines, without reading the whole article or viewing the images and videos. In order to do that, I used the **bag-of-words** and **word2vec**[3, 4] models to evaluate the accuracy of several classifiers, and compared the results of their evaluation.

# 2 Related work

Most of the work done on this problem is presented as part of the Clickbait Challenge 2017 workshop at Bauhaus-Universität in Weimar (Germany). The task of this workshop was to develop a classifier that rates how (much) click-baiting a social media post is, given a dataset consisted of information about the articles (headlines, image captions etc.) and images accompanying the articles. The classifiers used on this workshop had to output a clickbait score in the range [0,1], where a value of 1.0 denotes that a post is heavily click-baiting. As primary evaluation metric Mean Squared Error (MSE) with respect to the mean judgments of the annotators is used, and accuracy, $F_1$-score, precision and recall, as one of the secondary evaluation metrics (used only for informational purposes).

The dataset of this workshop was composed of three subsets (labeled as: "Training", "Training / Validation", "Unlabeled"). The details about the distribution of the subsets are given in Figure 1.

| Headline type | Training | Training / Validation | Unlabeled | Σ |
|---|---|---|---|---|
| Clickbait | 762 (0.74%) | 4761 ( 4.68%) | ? | 5532 + ? (? %) |
| News | 1697 (1.66%) | 14777 (14.48%) | ? | 16474 + ? (? %) |
| Σ | 2459 (2.41%) | 19538 (19.15%) | 80012 (78.44%) | 102009 (100.00%) |

Figure 1: Contingency table (distribution) of the headlines of the Clickbait Challenge 2017 workshop dataset.

After the finish of the workshop, the results were published on-line[1]. *Omidvar et al.* (team *albacore*) achieved the lowest value of **MSE** (**0.032**) by applying different kind of deep learning architectures[5]. *Zhou* (team *zingel*) achieved the highest **accuracy** (**85.6**%), and the highest value of **$F_1$-score** of (**0.683**) by applying a token-level, self-attentive mechanism on the hidden states of bi-directional Gated Recurrent Units (biGRU; of a Recurrent Neural Network)[7], *Papadopoulou, et al.* (team *snapper*) achieved the highest value of **recall** (**0.893**) by using a committee of classifiers, each trained on a different class of text features [6], and team *houndshark* achieved the highest value of **precision** (**0.779**).

[1] More details and other results are available at the official website of Clickbait Challenge 2017.

# 3 Data description & sources

## 3.1 Sources

The dataset that I used consisted only headlines of news and clickbaits, and is combined from multiple sources:

- Clickbait Challenge 2017 [Workshop at Bauhaus-Universität in Weimar, Germany].
- Datasets collected by other people on **GitHub** for their own projects:
  - clickbait [by: bhargaviparanjape]
  - Clickbait-Detector [by: LorenzoNorcini]
  - clickbait-detection [by: pfrcks]
  - clickbait-detector [by: saurabhmathur96]
  - clickbait-ml [by: ventodiventu]
- Reddit:
  - In the News (/r/inthenews) [News headlines]
  - News (/r/news) [News headlines]
  - Saved You a Click (/r/savedyouaclick) [Clickbait headlines]

## 3.2 Getting Reddit data by using PushShift.io

In order to get all the informations about the Reddit submissions, I used the PushShift.io API in combination with a parser that I wrote in Python[2].

## 3.3 Labeling

The dataset from the Clickbait Challenge 2017 workshop and the datasets from the GitHub repositories were already manually labeled. The datasets that I collected from Reddit were not labeled, so I labeled them by using their own description, i.e. the titles from the "In the News" and "News" subreddits as news headlines, and the titles from "Saved You a Click" as clickbait headlines.

## 3.4 Dataset composition

I composed my own dataset by merging all the clickbait and news headlines from the manually-labeled datasets (Clickbait Challenge 2017 workshop; GitHub repositories). After that, I deleted all the duplicate headlines, and as a result I got an unbalanced dataset (14984 more clickbait than news headlines). To balance the dataset, I used the Reddit submissions with the highest scores of the news subreddits ("In the News" and "News"). After all, the final dataset consisted **116780 unique headlines** (58390 items of each class). (More information about the distribution of the headlines are presented as a contingency table in Figure 2).

| Headline type | CC2017 | GitHub | Reddit | Σ |
|---|---|---|---|---|
| Clickbait | 5335 (4.57%) | 53055 (45.43%) | 0 (0.00%) | 58390 (50.00%) |
| News | 15881 (13.60%) | 27525 (23.57%) | 14984 (12.83%) | 58390 (50.00%) |
| Σ | 21216 (18.17%) | 80580 (69.00%) | 14984 (12.83%) | 116780 (100.00%) |

Figure 2: Contingency table (Distribution) of the headlines of my own (self-composed) dataset. | CC2017: Clickbait Challenge 2017

[2]The complete project (code & data) is available here - https://github.com/jStojcheski

Some headline examples from the composed dataset:

- Labeled as "Clickbait":
  - Source: Clickbait Challenge 2017
    - "Here's How Trump Could Make Mexico Pay for the Border Wall"
    - "Want To Surf Down An Active Volcano? Here's How To Do It"
  - Source: GitHub
    - "This Is Why You Should Always Use Your Your Right Hand To Make Calls"
    - "We Asked A Married Couple To Try Each Other's Hobbies And This Is What Happened"
- Labeled as "News":
  - Source: Clickbait Challenge 2017
    - "Germany axe attack at train station leaves five injured, suspect arrested"
    - "Stockholm truck attack: Man arrested after four killed in shopping street, Sweden police say"
  - Source: GitHub
    - "Macedonia says compromise with Greece over name dispute possible"
    - "Tesla allows self-driving cars to break speed limit, again"
  - Source: Reddit
    - "Elon Musk cheers as Norway moves toward ban on petroleum-powered cars"
    - "SpaceX is about to make history by relaunching a used Falcon 9 rocket"

# 4 Preprocessing

The next step was to do basic preprocessing by checking whether there are inappropriate characters from different encodings. I examined all the headlines containing inappropriate characters and composed a dictionary to automatically decode them into their appropriate characters. (Some translation examples are given in Figure 3.) After that, I used a special tag `<number>` to tag all the numbers, removed all the non-letters/-numbers from the headlines, and switched all the uppercase characters to lowercase characters.

| From | &amp | " | Ã¡ | Ã± | Ã³ | Ã¸ |
|---|---|---|---|---|---|---|
| To | & | " | á | ñ | ó | ø |

Figure 3: Some examples from the self-defined dictionary.

Additionally, I used some modules from NLTK (Natural Language Toolkit) such as: English stop words, WordNet Lemmatizer, and Porter Stemmer; and combined them into the following additional preprocessing methods:

- **No additional preprocessing** method (none of the modules above used).
- Remove the **English stop words** from the sentences. (**Without stop words**)
- **Lemmatize** the words of the sentences. (**Lemmatization**)
- **Stem** the words of the sentences. (**Stemming**)
- Remove the **English stop words**, and **lemmatize** the words of the sentences. (**W/O stop words + Lemm.**)
- Remove the **English stop words**, and **stem** the words of the sentences. (**W/O stop words + Stem.**)

- **Lemmatize** and **stem** the words of the sentences. (**Lemm. + Stem.**)
- Remove the **English stop words**, **lemmatize**, and **stem** the words from the sentences. (**W/O stop words + Lemm. + Stem.**)

Note: The phrases between the parentheses are the phrases used in the figures.

# 5 Approach to the problem & evaluation

Obviously, this problem can be defined as a binary classification problem. I decided to use some of the most fundamental models (**bag of words** and **word2vec**) to evaluate several classifiers. I split the dataset into 10 equally-sized subsets by randomly selecting the headlines (without replacement) to evaluate the classifiers by using a 10-fold cross-validation method. In the results presented in this article, the **accuracy** (%) of each classifier is used as primary evaluation metric and is measured as the mean value of the accuracies for each of the 10 cross-validation iterations, and the **time** (seconds) is measured as the sum of the time needed by each of the 10 cross-validation iterations. In other words (mathematically written),

$$Accuracy(Classifier) = \frac{1}{10}\sum_{k=1}^{10} Accuracy(Classifier)_k \tag{1}$$

$$Time(Classifier) = \sum_{k=1}^{10} Time(Classifier)_k \tag{2}$$

As expected, accuracy is defined as:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{3}$$

As secondary evaluation metrics, I decided to use **precision**, **recall**, and **$F_1$-score**, defined as:

$$Precision = \frac{TP}{TP + FP} \tag{4}$$

$$Recall = \frac{TP}{TP + FN} \tag{5}$$

$$F_1\text{-}score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{6}$$

- TP: true positive | TN: true negative | FP: false positive | FN: false negative

In the next subsections, I present the models that I used, and the conclusions and results I derived by evaluating several classifiers on these models.

## 5.1 Bag of words

### 5.1.1 Initial results

The simplest approach to this problem was to use the bag-of-words model combined with `scikit-learn`'s `CountVectorizer` (to convert a collection of text documents to a matrix of token counts) and several classifiers (Extra Trees[2], Logistic Regression, Naïve Bayes, Neural Network, Random Forests[1]) with their default parameter settings in `scikit-learn` and to run them over all of the additional preprocessing methods (defined in section 4). The results that I got on My machine are presented graphically in Figure 4 and numerically in Figure 5. The conclusions that I derived from the results are:

- The accuracy drastically drops by removing stop words from the headlines. This is somehow an expected phenomenon, since a lot of clickbait headlines are composed of words like "how", "what", "you" etc. To confirm my suspicions about the content of the headlines, I examined them and found out some patterns that occur very often as clickbait headlines, and very rarely as news headlines. Such examples (and the number of occurrences in my dataset) are presented in Figure 6.

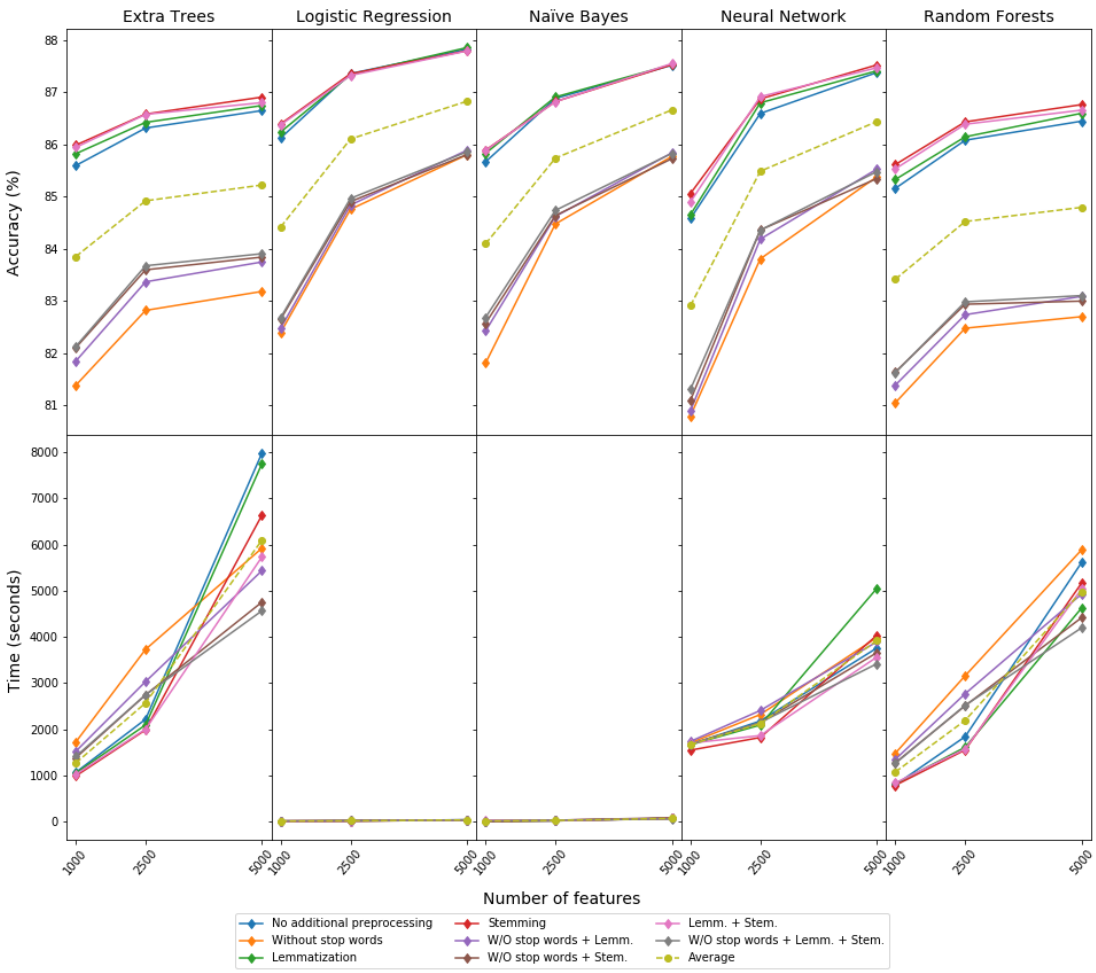


Figure 4: Graphical comparison of several classifiers using the bag-of-words model.

| Preprocessing methods | ET | LR | NB | NN | RF |
|---|---|---|---|---|---|
| No additional preprocessing | 86.6501 | **87.8275** | 87.5210 | 87.3797 | 86.4472 |
| Without stop words | 83.1803 | **85.8015** | 85.7707 | 85.3785 | 82.6957 |
| Lemmatization | 86.7469 | **87.8584** | 87.5321 | 87.4114 | 86.5987 |
| Stemming | 86.9087 | **87.7984** | 87.5295 | 87.5253 | 86.7649 |
| W/O stop words + Lemm. | 83.7455 | **85.8880** | 85.8383 | 85.5258 | 83.0853 |
| W/O stop words + Stem. | 83.8388 | **85.8066** | 85.7270 | 85.3442 | 82.9945 |
| Lemm. + Stem. | 86.8025 | **87.7941** | 87.5518 | 87.4713 | 86.6587 |
| W/O stop words + Lemm. + Stem. | 83.9039 | **85.8580** | 85.8289 | 85.4770 | 83.0998 |

Figure 5: Numerical comparison of several classifiers using the bag-of-words model. | **Bold**: Best result by preprocessing method (*row*); **Bold + Underline**: Best (overall) result | ET: Extra Trees; LR: Logistic Regression; NB: Naïve Bayes; NN: Neural Network; RF: Random Forests. | Measurement unit: accuracy (%)

.

| Clickbait-headline pattern | Clickbait | News |
|---|---|---|
| "Can you..." | 1377 | 5 |
| "How much do you..." | 53 | 0 |
| "How well do you..." | 458 | 1 |
| "Which (<adjective>/<noun>)..." | 1821 | 4 |
| "You (will/'ll) never..." | 35 | 0 |
| "You (will not/won't) believe..." | 39 | 0 |

Figure 6: Some clickbait-headline patterns and the number of occurrences in my clickbait/news headlines subsets.

- The accuracy (and the time needed to evaluate the classifiers) is directly proportional to the number of features used. (I was unable to use more than 5000 features due to computational-power limits.)
- The most accurate classifiers (with similar accuracy) are: Logistic Regression, Naïve Bayes, and Neural Network. (In fact, for each of the additional preprocessing methods, the Logistic Regression classifier is more accurate than each of the other classifiers.)
- The most accurate classifier during this step is the **Logistic Regression** classifier combined with **Lemmatization** as additional preprocessing method (named **BOW-Init**). This classifier achieved accuracy of **87.8584**%.
- The fastest classifiers (with similar temporal demand) are: Logistic Regression and Naïve Bayes.

Since the Logistic Regression and the Naïve Bayes classifiers are the most accurate and the fastest ones (the Neural Network classifier achieves similar accuracy, but it is a lot slower than these ones), I decided to run these two classifiers once again by using different parameters for them. The results that I got are presented in the next subsection (5.1.2).

### 5.1.2 Additional results

The additional evaluation was done by using the best additional preprocessing methods (No add. preproc.; Lemmatization; Stemming; Lemm. + Stem.) and the best classifiers (Logistic Regression (LR); Naïve Bayes (NB)), using different values for their main parameters:

- C: inverse of regularization parameter for LR
- $\alpha$: additive smoothing parameter for NB

The results that I got on My machine are presented graphically in Figure 7 and numerically in Figure 8. The conclusions that I derived from the results are:

- The Logistic Regression classifiers are better than each of the Naïve Bayes classifiers compared on the time needed to evaluate the classifiers, and better than all of the Naïve Bayes classifiers (except when $C \approx 0.00$) compared on the accuracy they achieved.
- Although each of the Logistic Regression classifiers achieved similar accuracy (except when $C \approx 0.00$), the best classification is done by using the **Logistic Regression** classifier with **C = 0.25** and only **Lemmatization** as an additional preprocessing method (named **BOW-Addt**). This classifier achieved accuracy of **87.9825**%.

### 5.1.3 Improving the accuracy by using ensemble methods

A little interesting experiment in attempt to improve the accuracy was to compose multiple ensemble classifiers from the previously-trained classifiers and to evaluate them by using an unweighted majority voting. I assigned probability to all the classifiers by applying the softmax function (Equation 7) on the accuracies of the classifiers, such that each classifier has a chance to

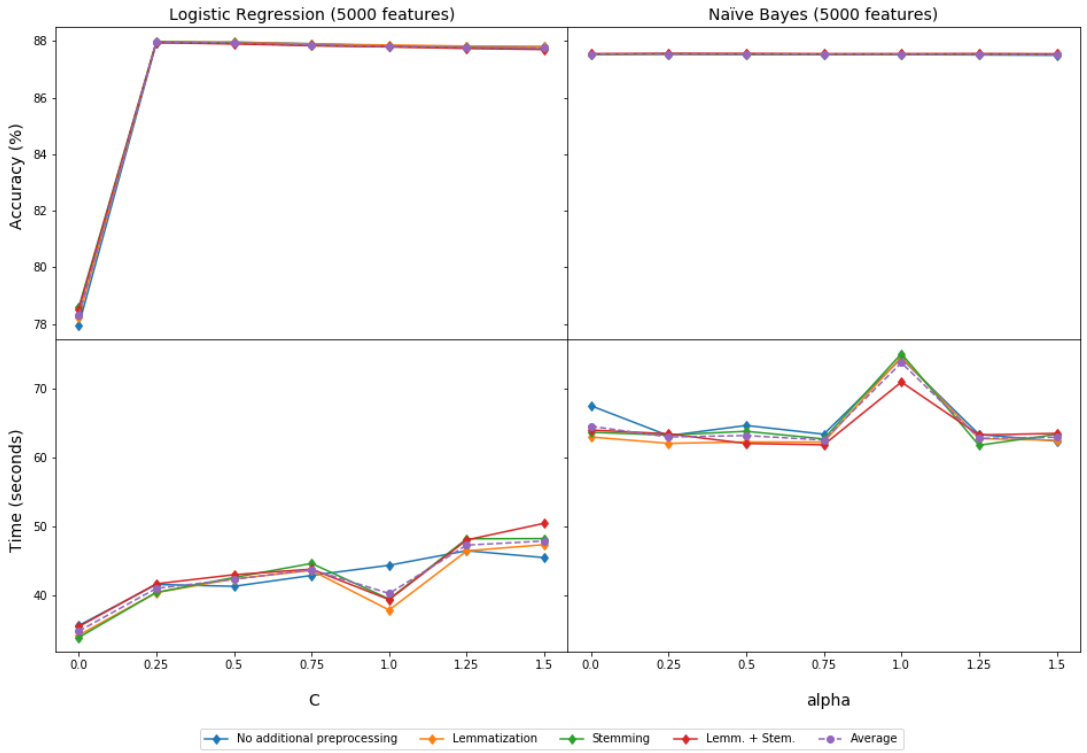


Figure 7: Graphical comparison between the Logistic Regression (LR) and the Naïve Bayes (NB) classifiers for different values of their main parameters, using the bag-of-words model.

| Accuracy (%) | No add. preproc. | Lemmatization | Stemming | Lemm. + Stem. |
|---|---|---|---|---|
| LR ($C \approx 0.00$) | 77.9466 | 78.2120 | 78.5725 | 78.5083 |
| LR ($C = 0.25$) | 87.9389 | **87.9825** | **87.9740** | **87.9431** |
| LR ($C = 0.50$) | **87.9611** | 87.9543 | 87.9320 | 87.8952 |
| LR ($C = 0.75$) | 87.9046 | 87.8978 | 87.8387 | 87.8370 |
| LR ($C = 1.00$) | 87.8275 | 87.8584 | 87.7984 | 87.7941 |
| LR ($C = 1.25$) | 87.8070 | 87.8198 | 87.7616 | 87.7359 |
| LR ($C = 1.50$) | 87.7933 | 87.7993 | 87.7171 | 87.6999 |
| NB ($\alpha = 0.00$) | 87.5372 | 87.5338 | 87.5261 | 87.5518 |
| NB ($\alpha = 0.25$) | 87.5355 | 87.5415 | 87.5458 | 87.5604 |
| NB ($\alpha = 0.50$) | 87.5227 | 87.5355 | 87.5321 | 87.5612 |
| NB ($\alpha = 0.75$) | 87.5261 | 87.5330 | 87.5270 | 87.5510 |
| NB ($\alpha = 1.00$) | 87.5210 | 87.5321 | 87.5295 | 87.5518 |
| NB ($\alpha = 1.25$) | 87.5098 | 87.5210 | 87.5210 | 87.5561 |
| NB ($\alpha = 1.50$) | 87.4970 | 87.5158 | 87.5133 | 87.5475 |

Figure 8: Numerical comparison between the Logistic Regression (LR) and the Naïve Bayes (NB) classifiers for different values of their main parameters, using the bag-of-words model. | **Bold**: Best result by preprocessing method (*column*); **Bold + Underline**: Best (overall) result | Measurement unit: accuracy (%).

be chosen from the set of classifiers according to its assigned probability. I ran the experiment for 1000 iterations, for several (odd) numbers of sample size (3, 5, 7, ... 21), by choosing the classifiers with and without replacement.

$$P(Classifier_c) = \frac{\exp\{Accuracy(Classifier_c)\}}{\sum_{i=1}^{C} \exp\{Accuracy(Classifier_i)\}} \tag{7}$$

The best result achieved, by using a combination of 7 different (without replacement) classifiers (named **BOW-Ens**), was an accuracy of **88.7952**%, which is an improvement of 0.8127% over the result of the best single classifier described in subsubsection 5.1.2.

## 5.2 word2vec

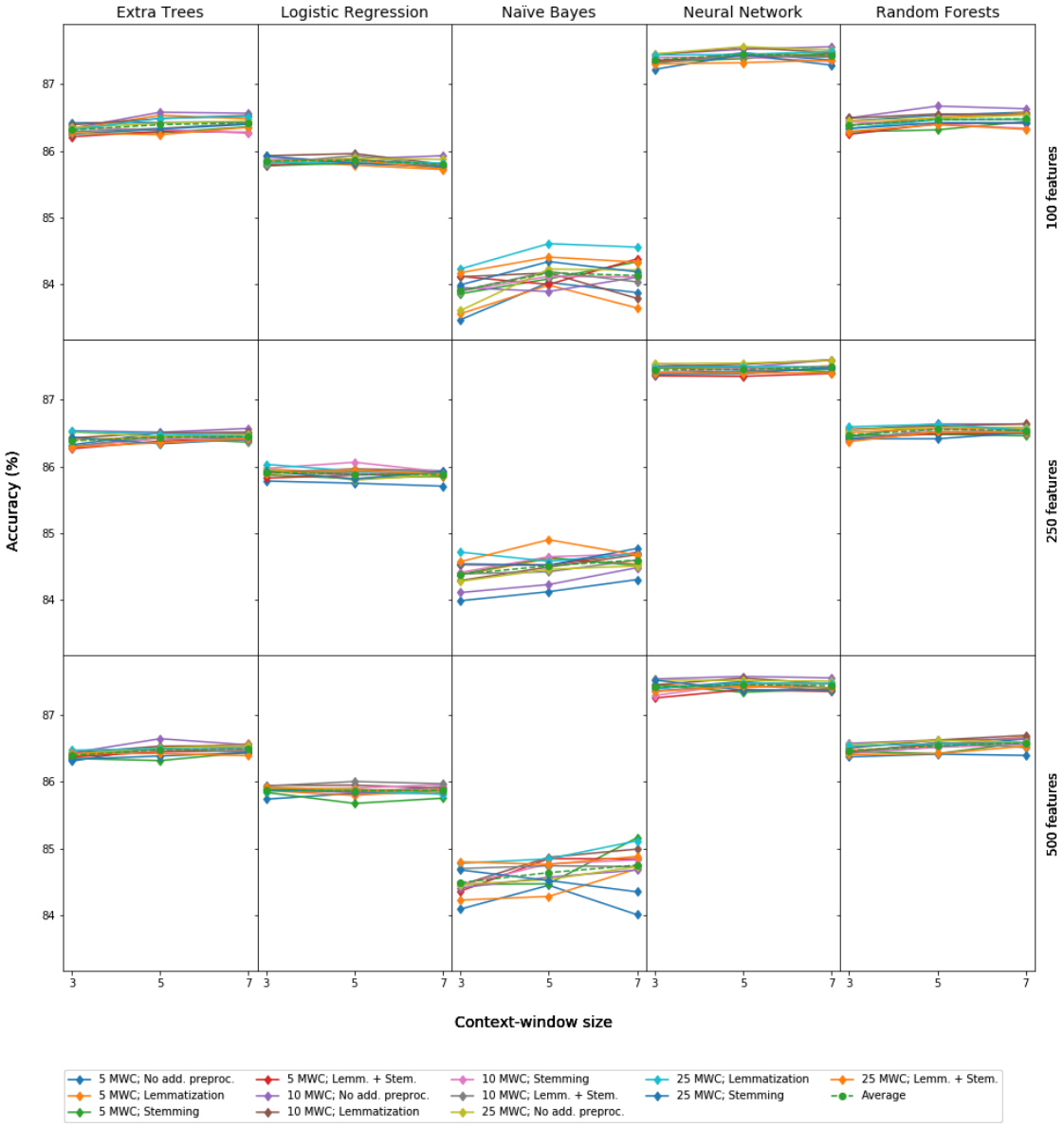


Figure 9: Graphical comparison of the accuracy achieved by several classifiers using the word2vec model. | MWC: Minimum word count.

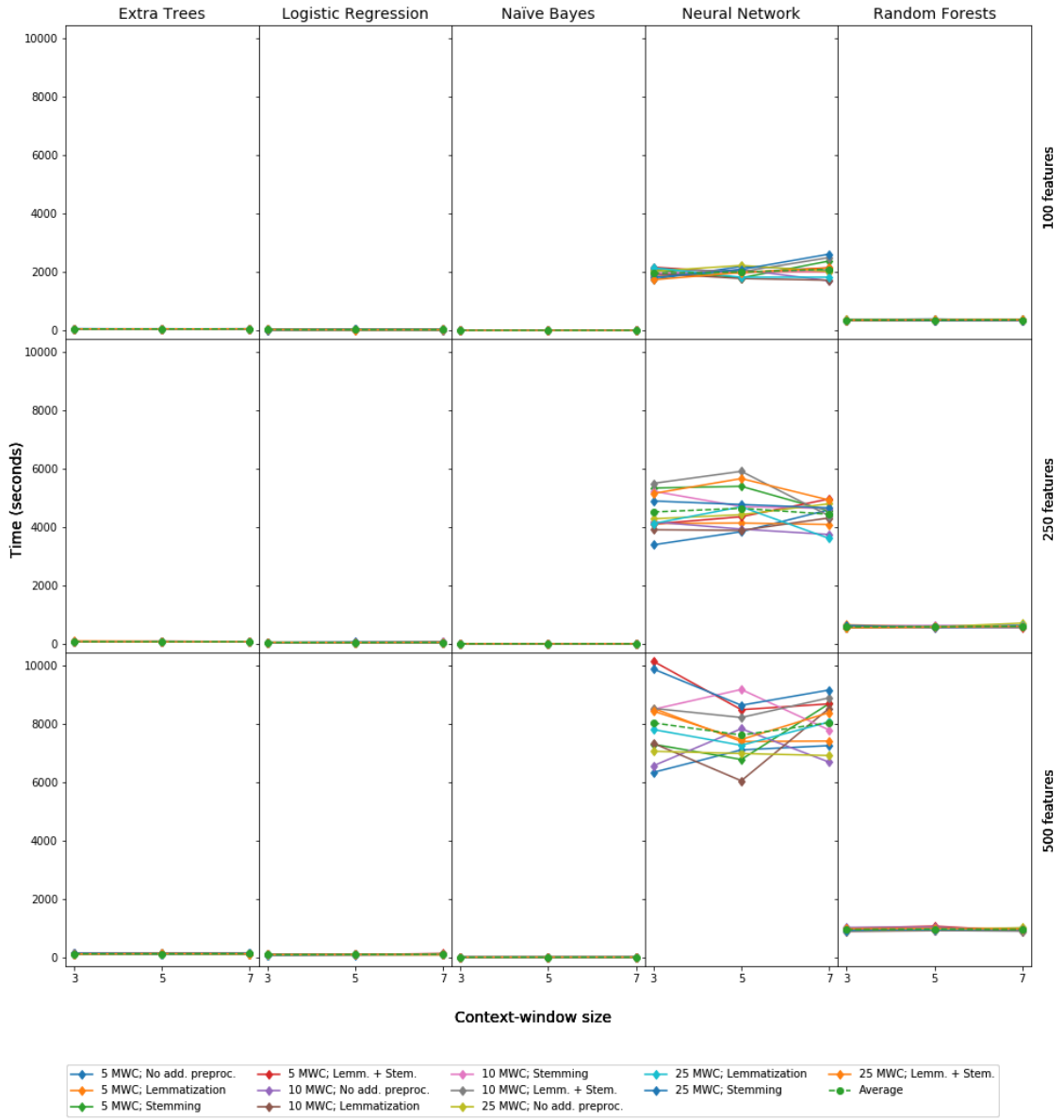


Figure 10: Graphical comparison of the time needed by several classifiers using the word2vec model. | MWC: Minimum word count.

### 5.2.1 Initial results

The results in the previous subsection hallmarked the benefits of using stop words in the model. Considering that, I used the word2vec model with different parameters ({100, 250, 500} features; {5, 10, 25} minimal word count; {3, 5, 7} context-window size) to evaluate the same classifiers as those defined in subsubsection 5.1.1. The results that I achieved by evaluating these classifiers on My machine are graphically presented in Figure 9 (achieved accuracy) and Figure 10 (time needed). Obvious conclusions that could be derived by quick inspection of these figures are:

- The Neural Network (NN) classifiers achieve higher accuracy than each of the other types of classifiers, but they are also a lot slower then them. The most accurate of them (and in this step, overall) is the **NN** classifier (with its default parameters) by using **no additional preprocessing method**, **250** features, **10** minimum word count, and **7** context-window size (named **W2V-Init**). This classifier achieved accuracy of **87.5963**%.
- The temporal demand to evaluate the Neural Network classifiers is directly proportional to the number of features used (without some significant ).

- There's no significant benefit regarding the level accuracy from using more features, i.e. the time-accuracy trade-off is not justified.

An important thing to mention is that minor differences between the accuracies (or the time needed/number of iterations, for that matter, since the NN classifier algorithm stops automatically if the last step was of length less than the tolerance threshold level) are due to the random initialization of the weights, so some minor differences between the classifiers in the group of NN classifiers are *not* statistically significant.

Considering all observations, it is hard to decide which values of the parameters (Number of features; Minimum word count; Context-window size), as well as additional preprocessing method, are preferred to choose in order to re-run the NN classifier by using different values for the parameters of the classifier. However, since the time-accuracy trade-off is not justified, I decided to use 100 features during the additional evaluation step (described in subsubsection 5.2.2).

### 5.2.2 Additional results

Since a minor difference for the accuracy level between a pair of NN classifiers is not statistically significant, I decided to run a bunch of NN classifiers once again by using fixed values for the following parameters:

- Number of features: 100
- Minimum word count: 10
- Context-window size: 5
- Additional preprocessing method: Lemmatization

and using different values for the following parameters:

- Activation function: Logistic (Equation 8), ReLU (Equation 9);
- Exponent for inverse scaling learning rate (`power_t`): 0.25, 0.50, 0.75, 1.00;
- Tolerance (for the optimization): $10^{-4}$, $10^{-5}$.

$$\sigma(x) = (1 + e^{-x})^{-1} \tag{8}$$

$$ReLU(x) = x^{+} = \max(0, x) \tag{9}$$

The results that I got on My machine are presented numerically in Figure 11. The conclusions that I derived from the results are:

- Using the ReLU activation function achieves higher level of accuracy than the Logistic activation function for the same values of the other two parameters (exponent for inverse scaling learning rate; tolerance).
- Although each of the NN classifiers using the ReLU activation function achieved similar accuracy, the best classification is done by using the **NN** classifier with **ReLU** activation function, exponent value of **0.75** for inverse scaling learning rate and tolerance of $10^{-4}$ (named **W2V-Addt**). This classifier achieved accuracy of **87.6332**%.

| Accuracy (%) | `power_t` = 0.25 | | `power_t` = 0.50 | | `power_t` = 0.75 | | `power_t` = 1.00 | |
|---|---|---|---|---|---|---|---|---|
| Tol. = $10^{-4}$ | 86.9404 | 87.5595 | 86.9130 | 87.4687 | 86.5996 | **87.6237** | 86.6612 | 87.4396 |
| Tol. = $10^{-5}$ | 87.1117 | 87.5278 | 87.1939 | 87.5372 | 87.0748 | **<u>87.6332</u>** | 87.1365 | 87.5398 |

Figure 11: Numerical comparison of the results achieved after the additional evaluation process by using different values for the parameters of the NN classifier, using the word2vec model. | **<u>Bold + Underline</u>**: Best (overall) result ; **Bold**: $2^{nd}$-best result | Colored cells: Logistic ReLU .

### 5.2.3 Improving the accuracy by using ensemble methods

The same approach (as in subsubsection 5.1.3), in attempt to improve the accuracy by composing multiple ensemble classifiers, was also applied to the classifiers evaluated in combination with the word2vec model.

The best result achieved, by using a combination of 13 different (without replacement) classifiers (named **W2V-Ens**), was an accuracy of **88.0784%**, which is an improvement of 0.4452% over the result of the best single classifier described in subsubsection 5.2.2.

# 6 Comparison of the best classifiers

## 6.1 My own classifiers

After the finish of the evaluation process, a comparison between the best classifiers was needed to decide which one of them is the best one. As already mentioned, I used the accuracy as primary evaluation metrics and the temporal demand, precision, recall and $F_1$-score as secondary evaluation metrics. The results are numerically presented in Figure 12.

| | BOW-Init | BOW-Addt | BOW-Ens | W2V-Init | W2V-Addt | W2V-Ens |
|---|---|---|---|---|---|---|
| Accuracy (%) | 87.8584 | 87.9825 | **88.7952** | 87.5963 | 87.6332 | 88.0784 |
| Time (s) | **37.93** | 40.46 | 5611.65 | 3751.88 | 2946.42 | 10151.80 |
| $F_1$-Score | 0.8766 | 0.8773 | **0.8861** | 0.8731 | 0.8735 | 0.8774 |
| Precision | 0.8910 | 0.8962 | 0.9008 | 0.8934 | 0.8938 | **0.9029** |
| Recall | 0.8628 | 0.8591 | **0.8719** | 0.8537 | 0.8542 | 0.8533 |

Figure 12: Comparison between the best classifiers for each step in section 5. | **Bold**: Best result by evaluation metric (*row*); | Note: the time needed to evaluate the ensemble classifiers (BOW-Ens; W2V-Ens) is measured as the maximum (longest) time of each of the sub-classifiers, by assumption that all sub-classifiers can be evaluated in parallel (at the same time).

Since accuracy is the most important evaluation metric, the best classifier is the **BOW-Ens** classifier (evaluated in subsubsection 5.1.3). This classifier achieved accuracy of **88.7952**%, but also achieved the highest value of **$F_1$-score** (**0.8861**) and **recall** (**0.8719**). The confusion matrix of this classifier is given in Figure 13.

| BOW-Ens | | Predicted class | |
|---|---|---|---|
| | | T | F |
| Actual class | T | 50912 | 7478 |
| | F | 5607 | 52783 |

Figure 13: Confusion matrix of the BOW-Ens classifier. | T: clickbait headline; F: news headline.

Further, in the next section (section 7), I compare this classifier with the best classifiers from the Clickbait Challenge 2017 workshop on the following evaluation metrics: accuracy, $F_1$-score, precision, and recall.

## 6.2 My own classifiers and the classifiers from Clickbait Challenge 2017

Figure 14 clearly shows that my **BOW-Ens** classifier outperforms each of the classifiers presented at the Clickbait Challenge 2017 workshop on the following evaluation metrics: accuracy (by 3.1952%), $F_1$-score (by 0.2031) and precision (by 0.1218). The only evaluation metric on which the BOW-Ens classifier is outperformed is recall (by 0.0211).

| | BOW-Ens | zingel | houndshark | snapper |
|---|---|---|---|---|
| Accuracy (%) | **88.7952** | 85.6 | 76.4 | 44.6 |
| $F_1$-Score | **0.8861** | 0.683 | 0.023 | 0.434 |
| Precision | **0.9008** | 0.719 | 0.779 | 0.287 |
| Recall | 0.8719 | 0.650 | 0.012 | **0.893** |

Figure 14: Comparison between my best classifier (BOW-Ens) and the best classifiers from Clickbait Challenge 2017. for each step in section 5. | **Bold**: Best result by evaluation metric (*row*); Underline: Best result by evaluation metric between the classifiers from Clickbait Challenge 2017.

# 7 Summary

As it was already presented in the previous sections, my BOW-Ens model achieved accuracy of 88.7952%. This is an improvement of 3.1952% over the highest level of accuracy (85.6%) achieved by the classifier proposed by *Zhou* (team *zingel*)[7] at the Clickbait Challenge 2017 workshop.

Although the achieved results are satisfactory, I believe that I could achieve even better results if I:

- **Re-edit and re-label the headlines manually**. (This was impossible to do due to the (relatively) huge amount of data.)
- Implement **Deep Learning** algorithms or **Support Vector Machines**. This is definitely worth trying, but also very costly since better hardware and/or a lot time is needed.
- Had **better hardware components**, so I could use them to run the classifiers by using a greater number of features.
- Could do **better parameter tuning**. Any unspecified parameters (in the text above) are set to their default values assuming that they are close to optimal. "Playing" with those parameters of the classifiers could potentially make them achieve better accuracy.

# 8 My machine

- CPU: Intel Core i7-4710HQ (4-core, 2.50 – 3.50 GHz, 6MB cache)
- GPU: NVIDIA GeForce GTX 850M (4GB DDR3)
- RAM: 16GB (2 x 8192MB) – DDR3, 1600MHz
- SSD: Samsung 960 EVO NVMe M.2 250GB
- OS: Manjaro Linux version 17.1.10 (Hakoila) | Kernel version 4.14.69-1

# References


[1] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[2] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[3] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[4] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[5] Amin Omidvar, Hui Jiang, and Aijun An. Using neural network for identifying clickbaits in online news media. *arXiv preprint arXiv:1806.07713*, 2018.

[6] Olga Papadopoulou, Markos Zampoglou, Symeon Papadopoulos, and Ioannis Kompatsiaris. A two-level classification approach for detecting clickbait posts using text-based features. *arXiv preprint arXiv:1710.08528*, 2017.

[7] Yiwei Zhou. Clickbait detection in tweets using self-attentive network. *arXiv preprint arXiv:1710.05364*, 2017.