Clickbait Article Detection Using Deep Learning: These Results Will Shock You!

Kali Cornn
Department of Computer Science, Stanford University

1 Introduction

Social networking websites such as BuzzFeed often utilize clickbait headlines and "teaser messages" in order to lure users to click and read an article [1]. Therefore, clickbait is an effective method used by content creators that draws users to websites.

While the article may not necessarily have false information, the headline can be greatly exaggerated and misleading. Such article can therefore spread misinformation.

Examples of clickbait article titles:

- "Tesla CHARGES You For THIS??"
- "McDonald's Just Made a Stunning Announcement That Will Completely Change the Future of Fast Food".

This project aims to solve the following problem: given an article headline or social media post, is it considered "clickbait" or "non-clickbait"?

The best approach for this problem, as described in this paper, was found to be a recurrent neural network with GloVe embedding weights.

2 Previous Work

Potthast et. al developed the first approach to clickbait detection by utilizing such machine learning methods as logistic regression, Naïve Bayes, and Random Forests on Twitter data. However, the features used in their models, such as character and word n-grams, were tediously created by hand [1].

Anand et. al utilized variants of a bidirectional recurrent neural network with distributed word embeddings to detect clickbait. Models used were RNN, LSTM, and GRU, with LSTM having the highest accuracy overall [2]. However, little previous work has utilized both convolutional neural networks and recurrent neural networks in analysis of clickbait.

3 Dataset

3.1 Dataset Collection and Overview

A custom dataset of 11,504 headlines was used for analysis. There was approximately a 50%-50% split between clickbait (5,751) and non-clickbait (5,753).

9,522 headlines were from The Clickbait Challenge dataset, a collection of social media posts that are pre-classified as either clickbait or non-clickbait, based on crowd-sourced user data [3].

1,982 headlines were scraped using a custom Python screen from three Reddit subreddits (/r/savedyouaclick, /r/news, /r/worldnews). These headlines were classified as clickbait or non-clickbait based on the subreddit they originated from (/r/savedyouaclick was considered clickbait; /r/news and /r/worldnews were considered non-clickbait).

4 Unigram Word Clouds

The **WordCloud** package was used on clickbait and non-clickbait headlines as a method of data exploration to determine the types of words that are frequently used in clickbait headlines.

Figure 1: Top 20 words present in clickbait titles

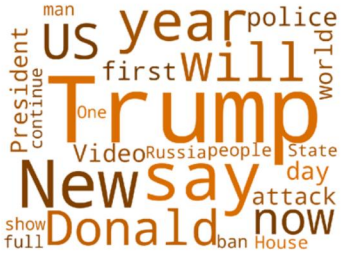

Figure 2: Top 20 words present in non-clickbait titles

5 Approach and Results

Data were randomly split into train, development, and test sets by using a 60%-20%-20% ratio.

All models were evaluated by looking at accuracy, precision, recall, and F1-score on the test set.

5.1 Baseline Models

Baseline models used were Multinomial Naïve Bayes (MNB), Support Vector Machines (SVM), and Random Forest Classifier (RFC).

All baseline models used Term Frequency-Inverse Document Frequency (**tf-idf**) as features. To do so, the input text was converted to a matrix of Tf-idf features by using the **TfidfVectorizer** in the **sklearn feature_extraction** package. Tf-idf weighs words based on their frequency and importance within a document: common words such as "the" are weighed less and rarer terms are weighed more.

Multinomial Naïve Bayes

A supervised learning method, multinomial Naïve Bayes is a common method, based on Bayes Theorem used in NLP, particularly text classification. It relies the assumption that features used for classification are independent.

This classifier achieved 72.0% accuracy.

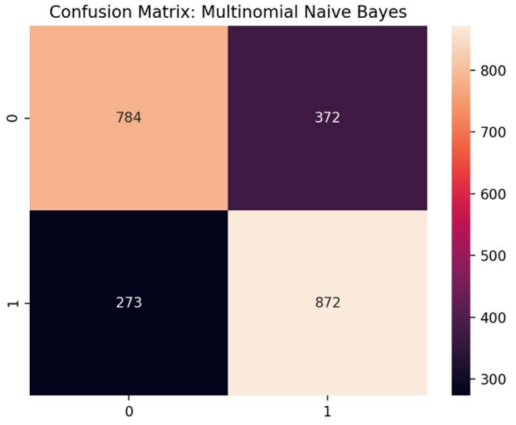

Figure 3: Confusion matrix for the output of the MNB classifier

Support Vector Machines

This classifier aims to create an optimal hyperplane that splits all data points into two perfectly separable classes.

This was implemented using **LinearSVC** in the **sklearn svm** package; **LinearSVC** allows for flexibility in the choice of parameters and various loss functions. The multi-class strategy used was **ovr (one-versus-rest**), and used **L2** penalty.

This classifier achieved 69.8% accuracy.

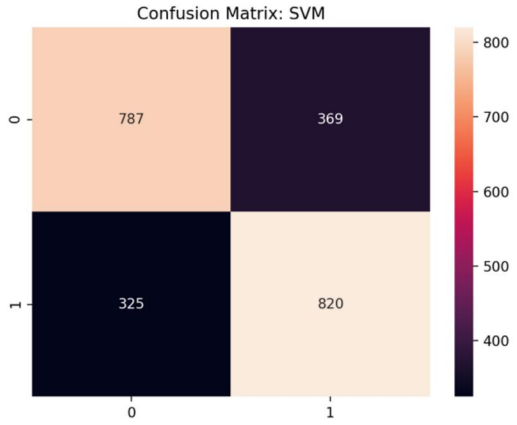

Figure 4: Confusion matrix for the output of the SVM classifier

Random Forest Classifier

Random forest classifiers create de-correlated trees using a randomly selected set of features, often with bootstrapping (sampling with replacement), and takes the average of these trees.

This classifier used the **gini** criterion, 1000 trees (n_estimators = 1000), a maximum of 3 features when looking for a tree split (max_features = 3), and bootstrapping.

This classifier achieved 72.1% accuracy, and also had the highest precision, recall, and F1-scores of all baseline models. However, this was at the expense of a significantly slower training time.

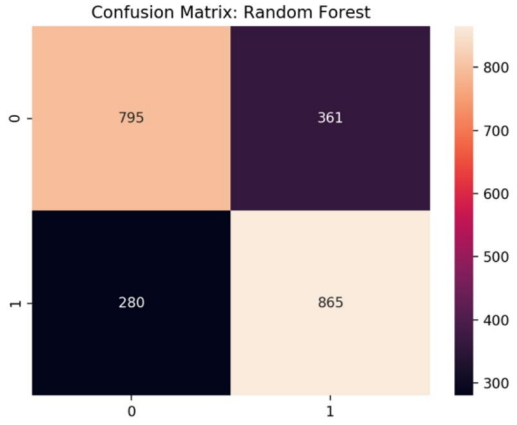

Figure 5: Confusion matrix for the output of the RFC classifier

5.2 Neural Network Models

All neural network models used Global Vectors for Word Representation (**GloVe**).

GloVe represents words as vectors, in which the cosine similarity between two word vectors measures

the similarity of corresponding wods. 100-dimensional word vectors trained on Gigaword and Wikipedia data were used for the neural network models [4].

Adam optimization was used with a learning rate of 0.0005.

5.3 Convolutional Neural Network (CNN)

Convolutional neural network (CNN) models apply convolutional filters to inputs, followed by fully-connected (dense) layers. While mainly used in imaging, CNNs are also used for NLP tasks by applying convolutional filters to documents represented as matrices.

The CNN models for this project were created with the **keras** library.

Three models were created:

- CNN without GloVe embedding weights,
- CNN with GloVe embedding weights + 1 LSTM layer,
- CNN with GloVe embedding weights,

CNN with GloVe embedding weights

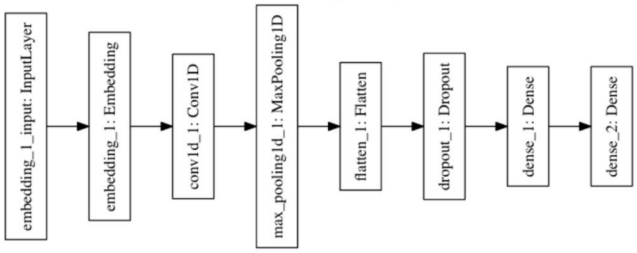

Figure 6: Architecture of CNN model, utilizing embedding weights

Layers

- **Input:** Text was tokenized and padded to a maximum length of 20.
- **Embedding:** The input was embedded into a GloVe matrix.
- **Convolution:** 64 filters and a kernel size of 2 were used, with **ReLU** activation.
- **Max Pooling:** The size of the max pooling windows was 2.
- **Dropout:** The dropout rate was 0.2.
- **Dense:** The output of convolution + max pooling, after dropout, was fed through two dense (fully-connected) layers. The first layer (parameter units = 128) had **ReLU** activation and the second layer (parameter units = 1) had **sigmoid** activation due to the binary output (classifying as clickbait or non-clickbait). **L2 regularization** with a 0.01 was also used.

The model was trained fairly quickly (6 epochs) and used a batch size of 64. The binary cross entropy loss function was used. This model achieved 72.9% accuracy.

Two other CNN models were created, as briefly described below, but they did not perform as well as the above model.

CNN without GloVe embedding weights

Without the use of GloVe embedding weights, the CNN had an accuracy of 63.5%, which is lower than that of all other models, including the baseline models.

CNN with GloVe embedding weights and one LSTM layer

This model used the same parameters for the Convolution and Max Pooling layers as in the previous model, with the addition of single long short-term memory (LSTM) layer of 128 units and **tanh** activation. This model was trained with 1 epoch and used a batch size of 64. This model achieved 71.0% accuracy.

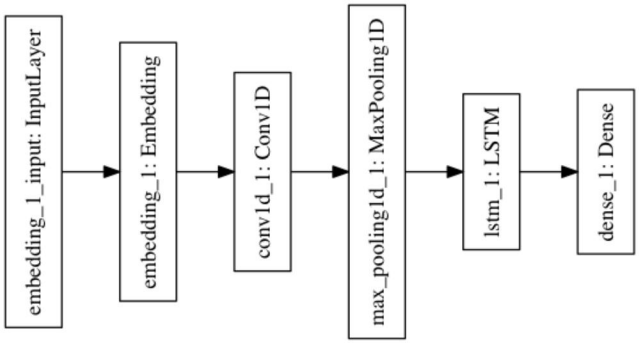

Figure 7: Architecture of CNN model, utilizing embedding weights and one LSTM layer

5.4 Recurrent Neural Network (RNN)

Recurrent neural network (RNN) models utilize "memory" to store information about inputs that have been previously perceived in time. While training, RNNs often suffer from the vanishing gradient problem, which means that the gradients of the neural network's output can become very small, so even a large change in a parameter can have a marginal effect on the gradient. So, when the gradients are too small, they are at risk of "vanishing." This problem is addressed by using LSTMs, which use gating mechanisms to "remember" values over time to create new hidden states, thereby improving gradient flow.

One RNN model was created with the **keras** library. This model utilized GloVe embedding weights.

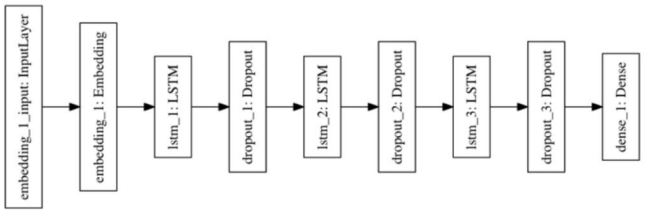

Figure 8: Architecture of RNN model, utilizing embedding weights

Layers

- **Input:** Text was tokenized and padded to a maximum length of 20.
- **Embedding:** The input was embedded into a GloVe matrix.
- **LSTM + Dropout:** Three LSTM layers were used, with a dropout layer in between each. The LSTM layers had 256 units, 128 units, and 64 units, respectively. Dropout rate was 0.2. Dropout was used to prevent the LSTM layers from overfitting.
- **Dense:** The output was fed through a single dense layer (parameter of units = 1) with **sigmoid** activation.

This model was trained for 5 epochs and used a batch size of 128. The binary cross entropy loss function was used.

6 RESULTS

Model	Accuracy	Precision	Recall	F1
MNB	0.720	0.721	0.720	0.719
SVM	0.698	0.700	0.698	0.698
RFC	0.721	0.723	0.721	0.721
CNN (w/o embed)	0.635	0.638	0.641	0.630
CNN (w/ embed)	0.729	**0.731**	0.728	0.723
CNN + LSTM	0.710	0.705	0.723	0.707
RNN	**0.738**	0.723	**0.770**	**0.742**

Table 1: Results from all baseline and neural network models

7 DISCUSSION

Baseline accuracy, from randomly guessing whether a headline is clickbait or non-clickbait, can be considered to be 50%. All models evaluated in section 6 performed above this baseline accuracy.

Overall, the RNN model performed the best, with an accuracy of approximately 73.8%. Though typically used for imaging-related tasks, the CNN model also performed well: its precision of 73.1% was higher than that of the RNN model.

Though this RNN model did not perform nearly as well as did Anand et. al's bidirectional model (98.19% accuracy), the LSTM model in this project, like in Anand et. al's findings, was also the best-performing model.

However, the higher accuracy of the RNN model was at the expense of a longer training time than that of the CNN models.

8 FUTURE WORK

The current neural network models could be improved with further hyperparameter tunings. In addition, 10-fold cross validation could be used to maintain consistency among train, development, and test datasets.

To further improve detection accuracy, a more diverse dataset could be used, one that not only utilizes headlines, but also other features of an article, such as post text and the presence of images.

9 GITHUB REPO

https://github.com/kcornn/cs230-project

REFERENCES

[1] Potthast M., Köpsel S., Stein B., Hagen M. (2016) Clickbait Detection. In: Ferro N. et al. (eds) Advances in Information Retrieval. ECIR 2016. Lecture Notes in Computer Science, vol 9626. Springer, Cham

[2] Anand, A., Chakraborty T., Park N.. (2016) We used neural networks to detect clickbaits: You won't believe what happened next! CoRR, abs/1612.01340, 2016. http://arxiv.org/abs/1612.01340.

[3] Clickbait Challenge, 2017, www.clickbait-challenge.org/.

[4] Pennington, Jeffrey. "GloVe: Global Vectors for Word Representation." GloVe: Global Vectors for Word Representation, nlp.stanford.edu/projects/glove/.