Beyond Buzzwords: Clickbait Detection using Machine Learning and Simple Transformers

Șerpar Ariana-Andra
Department of Computer Science
West University of Timișoara
Timișoara, Romania
ariana.serpar00@e-uvt.ro

I. What is clickbait and why is it harmful?

The term "clickbait" is quite straightforward in its meaning, showcasing an allegory where the audience is the prey and the creators are predators who set traps to lure us into clicking, using whatever means possible. From a technical standpoint, clickbait is characterized by attention-grabbing headlines, provocative thumbnails, or intriguing descriptions designed to pique curiosity. However, the actual content often fails to create the same response [1]. Utilizing capitalized words and sensational language are among the most common tactics in clickbait, intending to exaggerate the drama and entice users into clicking.

In essence, clickbait functions as a marketing strategy to boost views or clicks without the effort of delivering substantial and quality content [2]. The incentive is clear, as increased views lead to more revenue, making it an attractive prospect without any demanding requirements.

Despite its supposed initial effectiveness, clickbait is, in its essence, a double-edged sword. While it may work the first several times, audiences eventually become aware of the tactic. Clickbait not only elicits frustration and disappointment but also creates distrust between the audience and the media. Although it may sound harsh, clickbait is fundamentally a deceptive practice rooted in making exaggerated promises that cannot be realistically fulfilled. Despite all its drawbacks, it has become so widespread, that its prevalence has become a cause for concern. A global study on Statista [3] reveals that 30% of respondents avoid news sources due to perceived untrustworthiness and bias. This rift between the creators and the audience will only widen if clickbait-like tactics continue to be employed.

What can be done in this situation? One approach is to educate ourselves on the tactics employed in clickbait, enabling us to recognize it ourselves and exercise self-restraint to abstain from clicking on such content. Another alternative is to train Machine Learning (ML) to do it for us instead.

II. What would be the first step?

To train classifiers for clickbait detection, the initial step involves finding an appropriate dataset, as creating one ourselves would be a bit complicated. Fortunately, the Kaggle platform, tailored for data scientists and ML enthusiasts, provides readily available datasets covering a wide array of topics.

For the scope of this report, the Clickbait Dataset [4] proves to be very comprehensive, offering 32,000 classified news titles to choose from. However, due to the diverse classifiers we are going to test on the dataset, such a large number of instances would not be very effective, computationally speaking. Therefore, the initial dataset was downsized to 500 instances, which is enough for accurate results with acceptable training speed.

Table I shows five examples of instances that may be found in this particular dataset, to aid understanding.

TABLE I
Dataset sample examples

Text	Label
Zimbabwean unity talks fail	0
Your Zodiac Sign Will Tell You What Music To Listen To	1
Wreckage of crashed plane in Africa found	0
Which Type Of Swearer Are You	1
12 Signs You Grew Up In San Francisco	1

III. What classifiers could be used?

Having settled on our dataset, the next step involves the selection of classifiers with the potential for optimal performance. There is obviously a huge amount of classifiers out there, so it is not feasible to test all of them. However, considering that our goal is to achieve the highest possible performance metrics, it is necessary to at least cover the most important categories. This exploration spans from traditional ML models and ensembles to advanced models like Neural Networks and Transformers. The following paragraphs will delve into the specific characteristics of each classifier that was applied to the dataset.

A. Multinomial Naïve Bayes

Multinomial Naïve Bayes (MNB) is a probabilistic algorithm that utilizes the multinomial distribution of features to make predictions. This distribution is an extension of the binomial distribution, covering scenarios with more than two possible outcomes or categories [5]. Widely used in Natural Language Processing (NLP), MNB is particularly well-suited for classification tasks involving discrete features, such as text classification.

The multinomial model focuses especially on determining term frequency, representing the number of term occurrences in a document. Given the importance of certain terms in identifying clickbait titles, this feature makes the model an appropriate choice for clickbait detection. However, caution is necessary, as certain terms with high frequency may be stopwords that contribute nothing to the meaning of the document. Therefore, preprocessing is crucial for enhancing accuracy, which includes the removal of stopwords [6].

The "naïve" aspect of MNB comes from its assumption that each feature is independent, and therefore not influenced by other features [7]. However, this assumption may not remain true in practice, leading to potential impact from hidden correlations.

Despite the presence of hidden correlations, the MNB algorithm exhibits irreproachable efficiency and finds applications across diverse datasets, including those with high complexity. Notably, it demonstrates the ability to make accurate predictions even when confronted with limited training data, a significant advantage in several contexts.

B. Logistic Regression

Logistic Regression (LR) is a statistical model extensively used in applications of ML. Its versatility extends across fields such as statistics and epidemiology, while also serving as a benchmark for NLP tasks [8]. Specifically designed for situations involving binary outcomes, LR is an appropriate and predictable choice for our implementation, given the binary classification characteristics of our dataset.

Within the realm of ML, the classifier is categorized as a method that operates by taking input and multiplying it with weight values. It technically serves as a classifier that discerns which features from the input are most useful in distinguishing between different possible classes [9].

Contrary to common belief, LR does not assume a linear correlation between features and the target, which is an added benefit to a classifier that is already fast, adaptable and interpretable. While LR may exhibit limitations in handling imbalances, missing values, and outliers, these challenges are not encountered in our selected dataset.

C. Support Vector Machines

Support Vector Machine (SVM) operates on the idea of utilizing a hyperplane to delimit classes within a labeled dataset. While an endless number of hyperplanes can accomplish this task, SVM seeks to identify the one that maximizes the distance between the classes. The inclusion of various kernel functions, such as the Gaussian (RBF) kernel, enhances SVM's capability to handle instances with non-linear separability. The prediction mechanism of SVM involves determining on which side of the hyperplane an instance falls in [10].

Achievements of SVM in pattern recognition and text classification tasks underscore its high accuracy [11]. Similar to neural networks, SVM may demand meticulous parameter tuning, including kernel selection. The robustness of SVM is affected by outliers, as they can have a considerable impact on its performance by altering the position of the decision boundary.

Despite its apparent simplicity, SVM exhibits good performance on large high-dimensional datasets and resistance to overfitting, due to its focus on margin maximization. The strategic balance it strikes between complexity and effectiveness makes SVM a valuable choice in ML scenarios, including NLP.

D. k-Nearest Neighbours

The K-Nearest Neighbor (kNN) classifier operates under the assumption that similar data points are closer together [7]. Proximity is computed by employing mathematical distances, commonly utilizing the Euclidean distance metric.

Upon loading the data, we determine the number of neighbors considered for label assignment, a critical decision influencing model performance. While opting for multiple neighbors enhances prediction stability, a high neighbor count may also elevate errors [10]. It is customary to perform cross-validation and select an odd number of neighbors for accurate results.

Once the optimal number of similar data points is established, each instance undergoes analysis by inspecting its closest K neighbors. The count of neighbors with different labels dictates the most frequent category, likely becoming the assigned label for the data point.

The outcome varies depending on the task at hand, as kNN serves both classification and regression. For classification, the established label is returned, while for regression, the mean of the neighbors is computed. It is not a very common classifier when it comes to text classification, but there are some notable studies that use it with varying degrees of success [12].

Noteworthy advantages of this classifier include its simplicity and straightforward implementation, requiring minimal assumptions and parameter tuning. The algorithm's versatility is evident in its satisfactory performance across various tasks. However, its speed encounters challenges as the number of predictors or variables increases.

E. Random Forest

Random Forest (RF) stands out as a popular ensemble learning technique applied to both classification and regression tasks. It builds numerous Decision Trees and merges their predictions to yield a more precise and robust model. As part of the bagging algorithms family, RF is renowned for its simplicity, adaptability, and superior predictive performance.

The uniqueness of RF lies in the addition of randomness throughout the training process. It achieves this by generating multiple subsets of the training data through bootstrapping, a technique also known as "sampling with replacement". Additionally, when expanding each tree, only a random subset of features is considered for every split. This deliberate introduction of randomness helps reduce unwanted correlations and is a preventive measure against overfitting.

In classification scenarios, the final prediction often results from a majority vote among the individual tree predictions.

Surpassing the efficiency of a standalone Decision Tree, RF finds applications in NLP contexts. Notably, research has showcased its unexpected accuracy and its ability to boost the performance of other classifiers based on Transformers [13].

F. Gradient Boosting

Gradient Boosting (GB) is an ensemble learning technique that assembles the predictions of multiple weak learners, usually Decision Trees, to construct a robust predictive model. This ML algorithm is powerful and widely employed for both regression and classification tasks. Recognized for its ability to handle intricate data relationships, GB is held in high regard for its predictive performance.

Using a sequential construction of trees, GB distinctly addresses the errors of the collective predictions from the existing ensemble. This characteristic sets it apart from bagging methods like RF, where trees are built independently. Not unlike RF, GB offers insights into feature importance, elucidating which features wield greater influence in shaping predictions across the ensemble.

The optimization process within GB resembles gradient descent, aiming to minimize a loss function through iterative updates that shift the model toward reducing the gradient of the loss. The focal point of the model lies in minimizing residuals, representing the disparities between observed and predicted values. Each tree in the sequence is trained to predict the residuals of the ongoing ensemble.

Despite its remarkable achievements in various ML tasks, GB has not gained widespread popularity in NLP contexts, especially when compared to other ensembles like RF. Nevertheless, it is gradually gaining traction, supported by noteworthy research papers attesting to its stability as a classifier [14].

G. Long Short Term Memory Networks

Developed as an enhancement to the Recurrent Neural Network, Long Short-Term Memory (LSTM) networks are able to actively retain or discard information, which is particularly beneficial when dealing with extensive datasets where catastrophic forgetting may occur. Technically speaking, LSTMs incorporate a specialized memory cell, enabling the network to effectively preserve and update information across extended sequences.

The design of LSTMs encompasses three gate mechanisms: the input gate, the forget gate, and the output gate. These gates play crucial roles in directing the flow of information into, out of, and within the memory cell. The input gate determines which values from the input should be stored in the memory cell, while the forget gate decides what information to discard from the memory cell. Lastly, the output gate controls the information that will exit based on the current input and the memory cell's contents.

LSTMs are appropriate for sequence-to-sequence learning tasks where the input and output are sequences of data. Applications span diverse domains such as machine translation, image processing, as well as speech and handwriting recognition [15].

H. Simple transformers models

A Transformer is a sophisticated type of Deep Learning architecture extensively employed to address NLP challenges. By incorporating attention mechanisms, Transformers evaluate the significance of various segments within the input sequence, enabling the model to concentrate on important information and capture dependencies. Typically structured with an encoder-decoder architecture, transformers have demonstrated remarkable success, bolstering their promise and popularity.

In this particular implementation, the *simpletransformers* library was utilized, designed to ease the accessibility of Transformers with just a few lines of code. The library covers an array of tasks, including text classification, named entity recognition, question answering, and language modeling. It offers support for various model types, extending beyond English to other languages. Within this report, three models were tested for text classification on the designated dataset:

- BERT, also known as "Bidirectional Encoder Representations from Transformers" was introduced by Google researchers in 2018. BERT is trained on an extensive corpus and excels across numerous NLP tasks, even though its considerable parameter count renders it computationally demanding.
- RoBERTa, or "Robustly Optimized BERT Approach" is regarded as a more efficient version of BERT. Along with Electra, RoBERTa is theoretically expected to maintain or surpass BERT's performance.
- Electra, short for "Efficiently Learning an Encoder that Classifies Token Replacements Accurately", is less computationally expensive and uses fewer parameters. It is considered to be a basic version of BERT.

IV. How should we approach the implementation?

A. Preprocess

The initial step involves preprocessing the dataset to prepare it for subsequent application of classifiers. This stage holds particular significance for traditional ML and ensemble models, whereas minimal preprocessing is required for LSTM, and the transformer models demand close to no preprocessing.

Firstly, the *simpletransformers* library needs to be downloaded into our Google Colaboratory Notebook. Following that, we proceed to import all the essential libraries, as detailed in Table II, along with their corresponding versions. Having other versions may cause complications and the inability to properly execute the implementation.

Upon importing the mandatory libraries, the dataset of interest is loaded from Google Drive. Loading the dataset from Kaggle is also possible, but in this instance we will utilize a more compact version of the dataset, which was previously saved into Google Drive. The index column is subsequently removed from the resulting dataframe, as it is non-informative for the majority of predictive tasks, including this one. Before we begin the text preprocessing, an exploratory analysis of the dataset is conducted. This analysis involves familiarizing

TABLE II
PACKAGE VERSIONS

Package	Version
google.colab	0.0.1a2
pandas	1.5.3
matplotlib	3.8.2
numpy	1.22.4
sklearn	1.2.2
nltk	3.7
seaborn	12.0b3
simpletransformers	0.60.3
tensorflow.keras	3.0.1
torch	2.1.1

with the data type of each column, identifying any instances of missing values, and ensuring the dataset balance.

The balance of the dataset is especially important, as an imbalanced dataset could, without doubt, introduce a bias towards the most frequent class, thus impacting the performance of our models. To assess the balance, a visualization of the values within the *label* column is executed using *seaborn*. The findings from Figure 1 affirm that the dataset is perfectly balanced, which indicates that we may proceed with text processing.

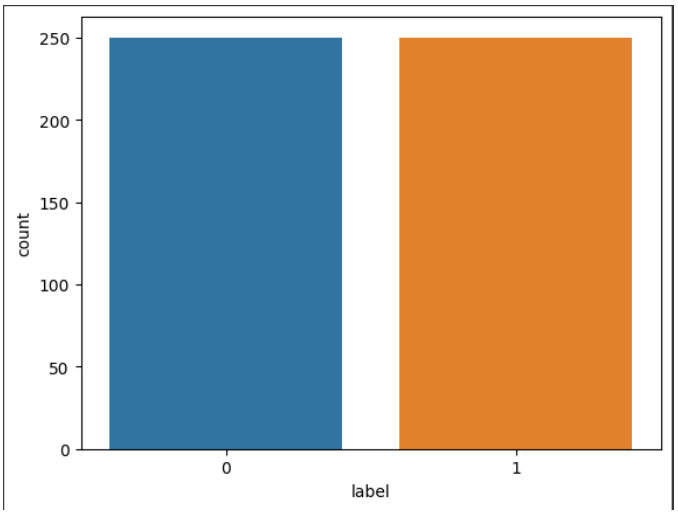

Fig. 1. Dataset balance visualization using Seaborn

The subsequent step involves the segmentation of our dataset into features and target variables. Given the fact that we have only 2 columns in our dataset, the X array encompasses the title names, while the Y array represents the binary labels. After the split, the text processing phase, featuring the X array, can be started.

Tokenization is the first processing step, a method that breaks a sentence into words and paragraphs into sentences. Given the single-sentence nature of our titles, tokenization entails the segmentation of each sentence into individual words.

Following tokenization, all individual words, including proper nouns, are transformed into lowercase. This preprocessing step is integral for the follow-up tasks, particularly the identification and removal of stopwords. Stopwords are commonly used words that lack specific meaning and serve to connect other words. Notable examples include ”the,” ”you,” ”a,” and others. Removal of stopwords is facilitated by the *nltk* library, which supplies a comprehensive stopword list.

In addition to stopwords, any numerical digits and punctuation marks are removed, as they lack distinctive characteristics aiding in the differentiation between clickbait and non-clickbait titles. A check for additional spaces is also conducted to avoid future issues that may arise. The remaining task involves lemmatizing the words, reverting them to their base form, with no change in part of speech during the transformation.

For the utilization of ML classifiers, the final step in this endeavor is to convert the extracted words into features. The *CountVectorizer* function is employed for this purpose. The resulting dataset features the extracted words as column headers, with binary values (0 and 1) attesting their presence or absence in a given title.

This concludes the preprocessing procedure applied for traditional and ensemble classifiers. In the case of the LSTM network, superior results were achieved by merely tokenizing the results without further processing. Transformer classifiers, on the other hand, necessitated the dataset to be split into training and testing sets without segregating the targets, as required in the previous instances.

B. Apply the classifiers

To facilitate the application of classifiers, data separation is imperative. A consistent testing and training split approach was employed across all classifiers, allocating 25% of the data for testing purposes. No validation set was created, as the small number of instances did not permit it. The division incorporated shuffling and stratification based on the label to preserve balance within both training and testing sets.

For classical and ensemble models, a cross-validation technique involving random subsampling with 30 repeats was implemented. All classifiers were imported from the *sklearn* library. Performance metrics for this category encompassed accuracy, f1-score, plotting the confusion matrix, and ROC-AUC values.

MNB and LR were employed without customized parameters. For SVM, the addition of the *probability=True* parameter was necessary to enable ROC curve plotting. KNN was executed with 5 neighbors, instead of the default 3. Grid Search was conducted for RF and GB to ensure optimal model parameters. While RF yielded prompt results, GB necessitated a substantial 4.5-hour duration for parameter identification, ironically exhibiting unsatisfactory performance despite the extended computation time.

In the context of the LSTM architecture, minimal customizations were applied, mainly involving consistent input dimensions at each step to foresee any errors. The model underwent training with a batch size of 32 over 12 epochs, a choice made to eliminate the possibility of overfitting. The

structural depiction of the sequential model is illustrated in Figure 2.

Layer (type)	Output Shape	Param #
embedding_15 (Embedding)	(None, 32, 100)	258400
lstm_15 (LSTM)	(None, 32, 32)	17024
global_max_pooling1d_15 (GlobalMaxPooling1D)	(None, 32)	0
dropout_15 (Dropout)	(None, 32)	0
dense_15 (Dense)	(None, 1)	33

Total params: 275457 (1.05 MB)
Trainable params: 275457 (1.05 MB)
Non-trainable params: 0 (0.00 Byte)

Fig. 2. LSTM model structure and parameters

Concerning the transformer classifiers, the implementation of random subsampling with 30 repeats was too computationally intensive, requiring an estimated duration of nearly 5 hours, as per calculations. In light of this, a faster cross-validation method, namely the k-fold cross-validation with 5 folds, was adopted. The default number of epochs for these models is set to 1, given their prolonged training duration. Notably, for the optimal model, which demonstrated superior performance, the training regimen entailed k-fold cross-validation across 3 epochs. Despite the computational expense and time consumption associated with this approach, the resulting performance surpassed expectations.

V. What results should we expect?

The obtained results are deemed commendable, showing average to high performances across nearly all classifiers. Notably, kNN model emerged as the least effective, demonstrating a mean accuracy of 55.09%, accompanied by a standard deviation of 0.014 and an f1-score of 0.69. An examination of the confusion matrix, as depicted in Figure 3, reveals a prominent issue generated by the False Positive labels. This signifies a notable prevalence of non-clickbait titles being misclassified as clickbait, while conversely, there are no instances of clickbait titles being inaccurately labeled as non-clickbait by the model. The ROC curve presented in Figure 4 portrays above-average performance, yielding an Area Under the Curve (AUC) value of 0.63. Although indicative of reasonable performance, the pursuit of a value closer to 1 remains the main objective. Encouragingly, the subsequent models all exhibit improved efficacy.

Regrettably, the second least effective classifier is GB, marking an unexpected disappointment despite the considerable training duration invested. Nevertheless, the results exhibit a noteworthy advancement in comparison to kNN. The classifier yields an average accuracy of 81.96%, accompanied by a standard deviation of 0.026 and an f1-score of 0.84. In examining the confusion matrix presented in Figure 5, a recurrent observation is the prevalence of False Positive

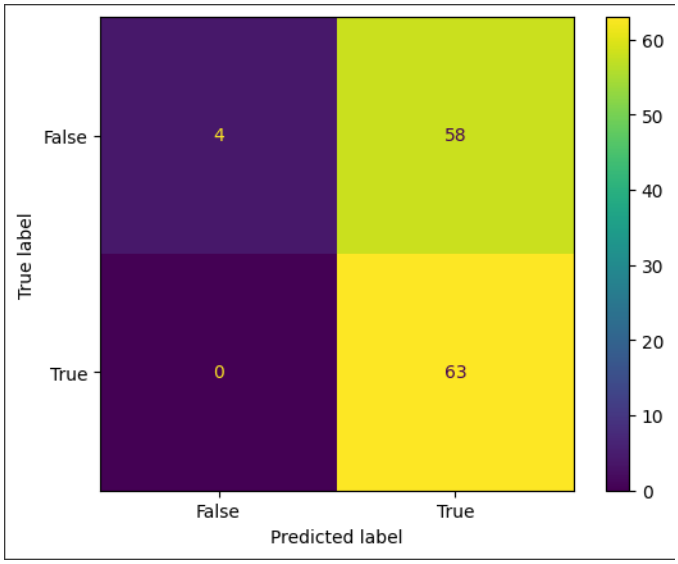

Fig. 3. k-Nearest Neighbours Confusion Matrix

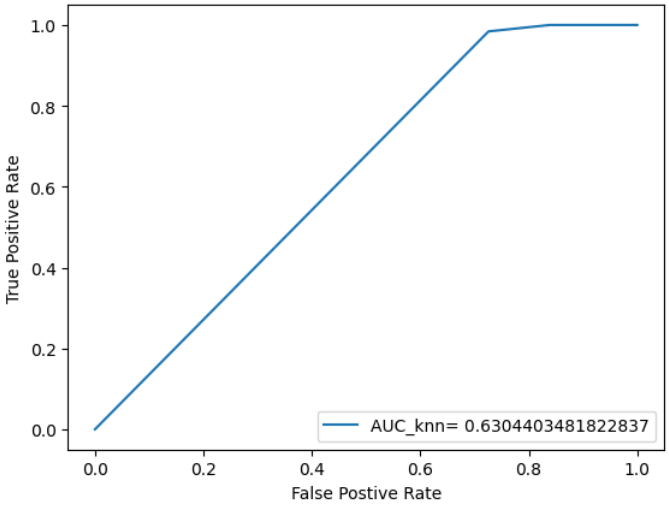

Fig. 4. Area Under the Curve (AUC) for k-Nearest Neighbours

values, although False Negative values are also evident in this instance. The ROC curve, illustrated in Figure 6, underscores a substantial enhancement, featuring an AUC value of 0.92.

These outcomes are surprising, given GB's typical high-performance attributes, particularly on extensive and intricate datasets. The suboptimal results may be attributed to the Grid Search process, which occasionally fails to identify the optimal parameters. It is noteworthy that the test accuracy projected by the Grid Search is 88%, exhibiting quite a difference from the cross-validated accuracy.

The subsequent classifier in our rank is SVM, presenting results that can be characterized as moderately satisfactory. The accuracy surpasses the preceding GB case by a small margin, registering at 83.54%. The standard deviation exhibits a slight increase, measuring 0.027, while the f1-score maintains a value of 0.84. An examination of the confusion matrix,

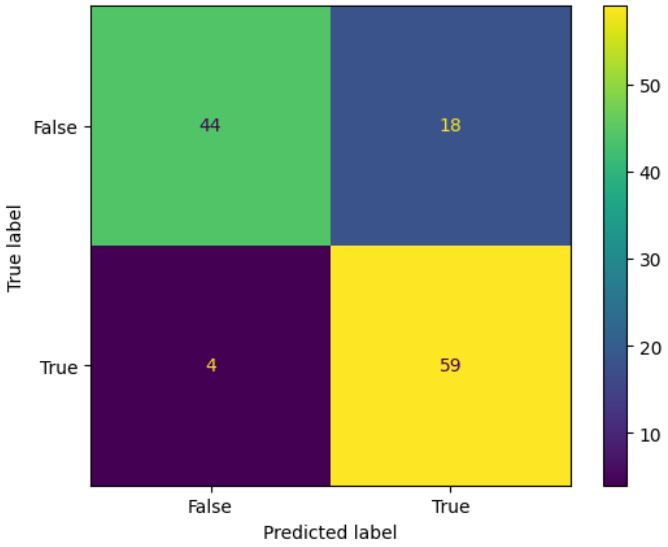

Fig. 5. Gradient Boosting Confusion Matrix

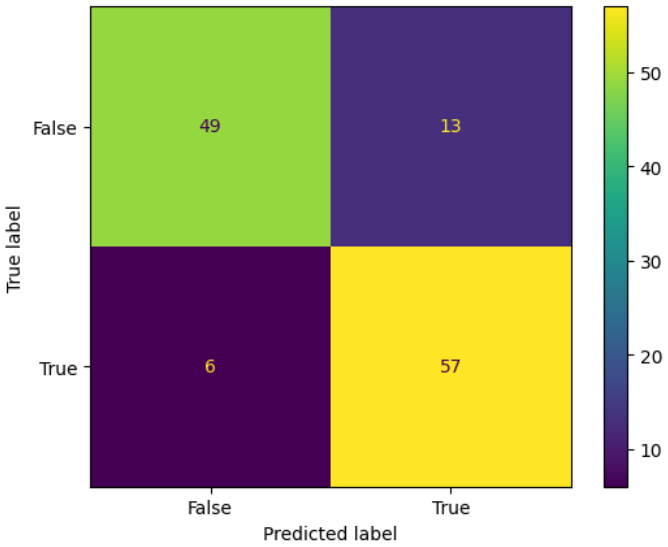

Fig. 7. Support Vector Machine Confusion Matrix

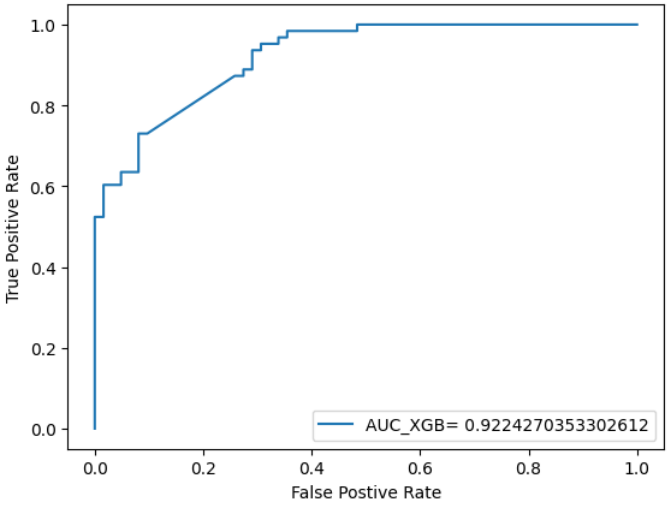

Fig. 6. Area Under the Curve (AUC) for Gradient Boosting

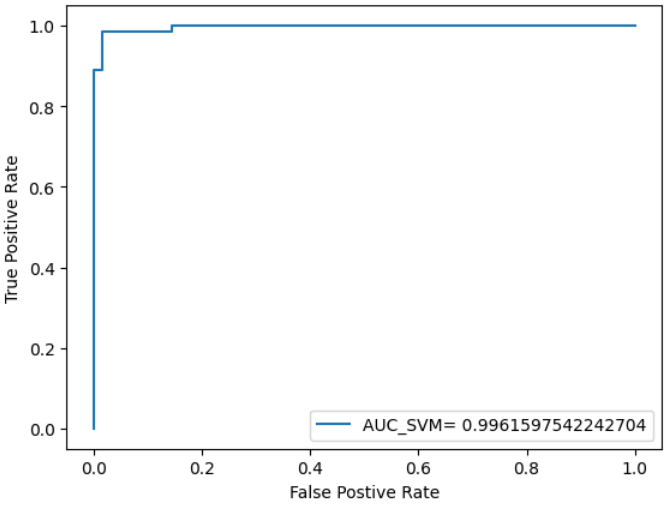

Fig. 8. Area Under the Curve (AUC) for Support Vector Machine

depicted in Figure 7, reveals incremental enhancements in the classification of False Positive instances, albeit accompanied by a noticeable rise in False Negative instances.

The most evident advancement can be observed in the ROC curve, showcased in Figure 8, wherein the AUC escalates from 0.92 (GB) to an impressive 0.99. No alternative kernels were explored, as the default ”RBF” kernel typically yields optimal results, while the ”linear” kernel is ill-suited for this particular scenario. Despite SVM’s relative underrepresentation in tasks of this nature, it upholds its standing by delivering an average performance coupled with speedy training times.

Advancing to classifiers with performance metrics exceeding 85%, we encounter the RF classifier, which notably outperforms GB by delivering superior and more expeditious results. The classifier attains an average accuracy of 87.68%, accompanied by an f1-score of 0.88 and a standard deviation of 0.033, representing the most substantial deviation computed thus far. Examination of the confusion matrix, illustrated in Figure 9, reveals a total of 16 misclassifications, with only one instance of False Negative classification.

However, a marginal decline is discernible in the ROC curve, depicted in Figure 10, where the AUC value for RF is measured at 0.96, in contrast to the 0.99 observed for SVM. Notably, it is pertinent to acknowledge that the Grid Search once again overestimates the test accuracy, promising a value of 0.92 without fulfilling it in the least. This deviation underscores the inclination of cross-validation to provide more realistic and often conservative results. In spite of this negative evolution in accuracy estimation, RF secures its position as the third-best classifier within the traditional and ensemble categories, a noteworthy accomplishment in its own right.

In the realm of traditional and ensemble classifiers, the

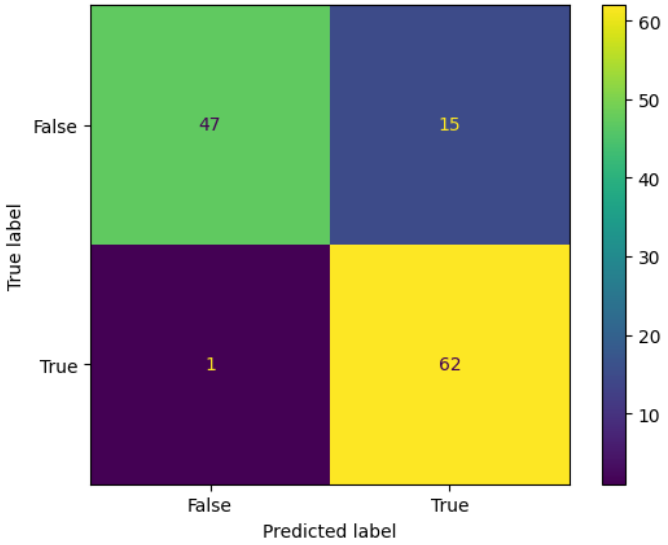

Fig. 9. Random Forest Confusion Matrix

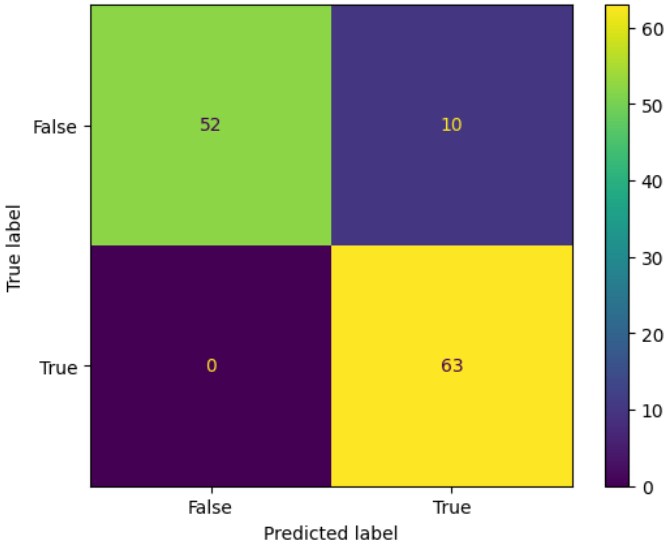

Fig. 11. Logistic Regression Confusion Matrix

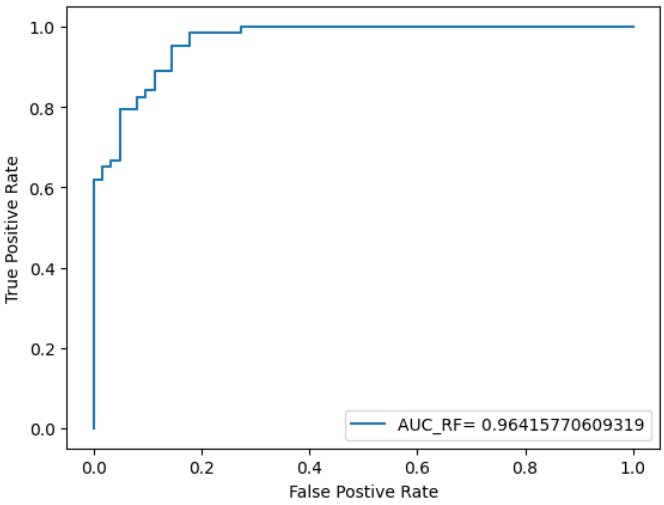

Fig. 10. Area Under the Curve (AUC) for Random Forest

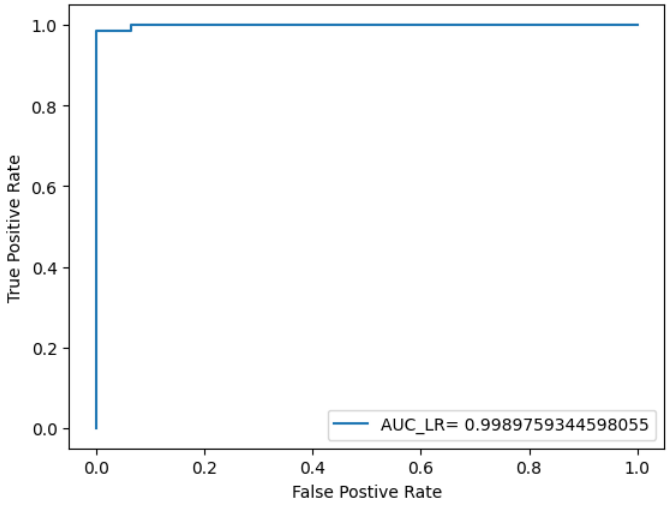

Fig. 12. Area Under the Curve (AUC) for Logistic Regression

penultimate contender is LR, showcasing performance metrics that nearly rival the chart-topping classifier. LR demonstrates an impressive f1-score of 0.92, a standard deviation of 0.027, and an accuracy of 92.13%. The accompanying confusion matrix, depicted in Figure 11, highlights a total of 10 misclassifications, all attributed to the False Positive category.

A noteworthy distinction for LR is its AUC value of 0.998, as evident in Figure 12. The results unveil an intriguing observation, as the more intricate traditional classifiers manifest a less proficient performance in this specific classification task.

Claiming the title of the most adept classifier within the traditional models and ensemble category is MNB. Remarkably, MNB achieved this distinction while concurrently exhibiting the swiftest training time, completing the training process in a mere second, and quickly outputting the results. Surpassing the 90% threshold, MNB attains an impressive accuracy of 92.83%, an excellent f1-score of 0.93, and a slight standard deviation of 0.022.

The confusion matrix portrayed in Figure 13 unveils a sparse tally of 6 misclassifications out of a total of 125 predictions, a testament to MNB's robust classification performance. As anticipated, False Positives constituted the predominant misclassification category. MNB further solidifies its prowess with an exemplary AUC value of 0.99, as depicted in Figure 14. As we transition to evaluating the LSTM and transformers, the objective is to surpass the good scores achieved by MNB.

As previously mentioned, the LSTM network was applied first on the fully preprocessed dataset, along with the traditional classifiers. However, it yielded subpar performances, that indicated a need for a change in the preprocessing. Therefore, the data for the second LSTM network underwent only tokenization.

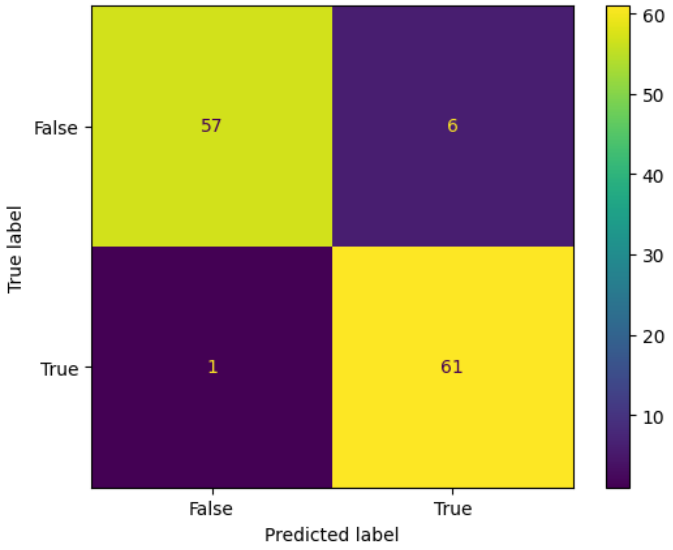

Fig. 13. Multinomial Naive Bayes Confusion Matrix

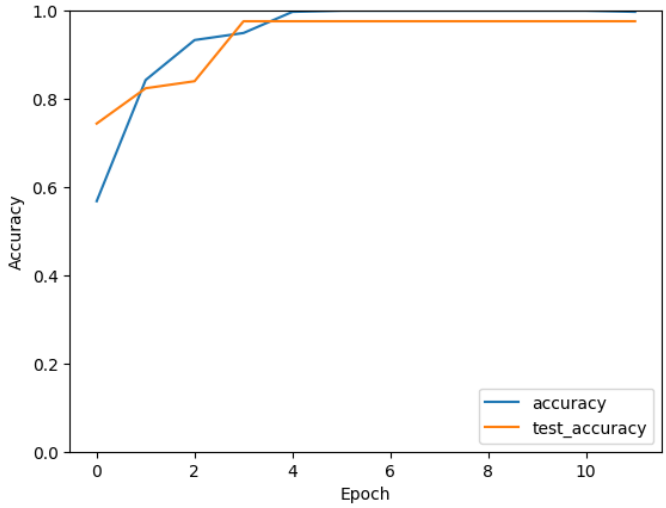

Fig. 15. Testing and training accuracy on LSTM

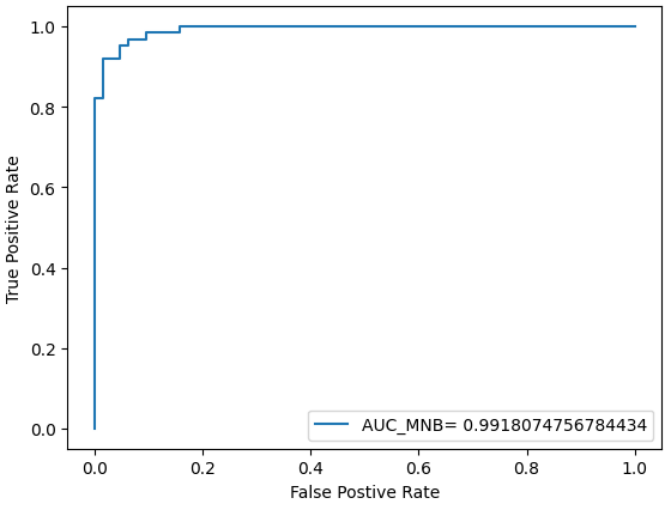

Fig. 14. Area Under the Curve (AUC) for Multinomial Naive Bayes

The second LSTM network exhibits notable proficiency in accuracy over the course of 12 training epochs. Emphasizing accuracy as the primary evaluation metric, the initial epoch showcases a training accuracy of 56.80%, juxtaposed with a testing accuracy of 74.70%. Subsequent epochs witness a swift ascent in values, with the training accuracy escalating to 100% while maintaining resistance to overfitting.

Culminating in a testing accuracy of 97.60%, the LSTM network attains outstanding performance. To mitigate concerns of overfitting, a visualization of the training and testing accuracies is presented through the utilization of *matplotlib*, as depicted in Figure 15. This graphical representation serves as a tool for analyzing the network's generalization capacity and performance stability across epochs.

Last, but certainly not least, the transformer models harnessed from the *simpletransformers* library were tested on the dataset. The first classification model that was tested leverages RoBERTa as its foundation and is executed without cross-validation, providing a snapshot of results without entailing an extensive training process. Impressively, this model attains a flawless accuracy of 100% and an AUC value of 1, ostensibly representing an ideal model. However, the absence of cross-validation shatters the credibility of this result, necessitating a more rigorous assessment. Adding a 5-fold cross-validation provided more realistic results, with 99.3% accuracy when training on 3 epochs.

Subsequently, a classifier utilizing Electra as its base undergoes evaluation without cross-validation. Regrettably, its performance quickly falls short of RoBERTa, yielding an accuracy of 99.18%. Therefore, the 2-hour cross-validation process was considered unlikely to provide better results than the ones already obtained with RoBERTa.

In pursuit of another robust evaluation, cross-validation is introduced to another model, this time adopting BERT as the foundational transformer. Employing k-fold cross-validation with 5 folds and only one epoch, the average accuracy registers at 99.3%, the same as RoBERTa. Further refinement entailed applying k-fold cross-validation with an extended 3 epochs, again with BERT as the transformer base. This culminates as the best performance among all the tested models, with a remarkable accuracy of 99.8%.

While the ultimate goal would have been to achieve 100% accuracy, we do not consider that to be realistic in a practical context, because of the appearance of various unforeseen factors. The attained 99.8% accuracy defies all expectations and stands as a testament to the formidable capabilities of these sophisticated Transformer models.

To close this section, Figure 16 supplies a comprehensive comparison between the accuracies of all the models.

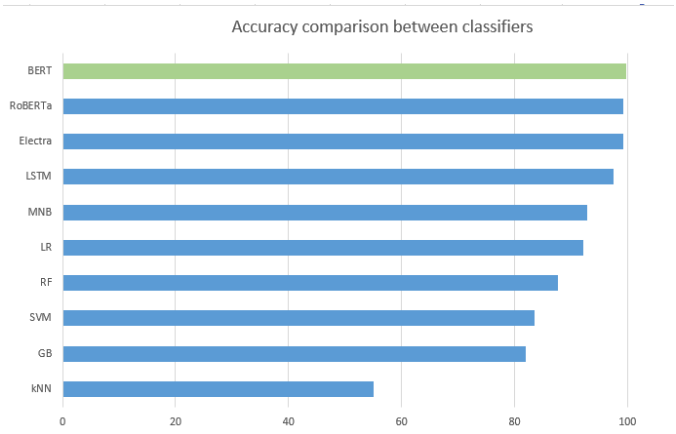

Fig. 16. Comparison between classifiers based on accuracy

VI. What conclusions can be drawn?

This report systematically tackled the task of clickbait classification, aiming to present a comprehensible yet rigorous analysis. The investigation centered on the deployment of diverse models, encompassing traditional and ensemble classifiers, as well as LSTM and Transformers. The outcomes, predominantly surpassing the threshold of average performance, underscore the efficacy and adaptability of the employed models.

The best models were without a doubt those based on Transformers. The best performer emerged as a Transformer model based on BERT, which attained a nearly impeccable accuracy of 99.8% during cross-validation with three epochs.

Other outstanding models included the models based on RoBERTa and Electra, the LSTM architecture and the MNB approach, which surprised with its simplicity and effectiveness.

References

[1] A. Chillingworth, "What is clickbait and should you use it?."
[2] Juxtathinka, "Why clickbait is bad for you."
[3] Statista, "Reasons for avoiding news worldwide as of february 2022."
[4] A. Anand, "Clickbait dataset."
[5] D. Labhade, N. Lakare, A. Mohite, S. Bhavsar, S. Vispute, and G. Mahajan, "An overview of machine learning techniques and tools for predictive analytics," *Asian Journal For Convergence In Technology (AJCT) ISSN-2350-1146*, vol. 5, no. 3, pp. 63–66, 2019.
[6] G. Singh, B. Kumar, L. Gaur, and A. Tyagi, "Comparison between multinomial and bernoulli naïve bayes for text classification," in *2019 International Conference on Automation, Computational and Technology Management (ICACTM)*, pp. 593–596, IEEE, 2019.
[7] C. C. Aggarwal *et al.*, *Data mining: the textbook*, vol. 1. Springer, 2015.
[8] A. Prabhat and V. Khullar, "Sentiment classification on big data using naïve bayes and logistic regression," in *2017 International Conference on Computer Communication and Informatics (ICCCI)*, pp. 1–5, IEEE, 2017.
[9] S. Indra, L. Wikarsa, and R. Turang, "Using logistic regression method to classify tweets into the selected topics," in *2016 international conference on advanced computer science and information systems (icacsis)*, pp. 385–390, IEEE, 2016.
[10] M. Cakir, M. A. Guvenc, and S. Mistikoglu, "The experimental application of popular machine learning algorithms on predictive maintenance and the design of iiot based condition monitoring system," *Computers & Industrial Engineering*, vol. 151, p. 106948, 2021.
[11] M. Goudjil, M. Koudil, M. Bedda, and N. Ghoggali, "A novel active learning method using svm for text classification," *International Journal of Automation and Computing*, vol. 15, pp. 290–298, 2018.
[12] K. Shah, H. Patel, D. Sanghvi, and M. Shah, "A comparative analysis of logistic regression, random forest and knn models for the text classification," *Augmented Human Research*, vol. 5, pp. 1–16, 2020.
[13] H. Chen, L. Wu, J. Chen, W. Lu, and J. Ding, "A comparative study of automated legal text classification using random forests and deep learning," *Information Processing & Management*, vol. 59, no. 2, p. 102798, 2022.
[14] R. A. Stein, P. A. Jaques, and J. F. Valiati, "An analysis of hierarchical text classification using word embeddings," *Information Sciences*, vol. 471, pp. 216–232, 2019.
[15] R. C. Staudemeyer and E. R. Morris, "Understanding lstm–a tutorial into long short-term memory recurrent neural networks," *arXiv preprint arXiv:1909.09586*, 2019.