Understanding Machine vs Human Generated Text in News

INTRODUCTION

The average consumption of news from non-traditional sources such as social media, blogs and instant communications groups has received much attention in the last few years. Social media is one of the key sources of news information on which people today rely. There are a variety of problems associated with the release of a new data source. In 2016, considerable attention was devoted to the prominence of misinformation within America's political discourse, particularly following President Trump's elections. The word "false news" was widely used to describe factually inaccurate and misleading articles often published for money purposes through page reviews. Moreover, the issue of legitimacy is difficult: what 'legitimizes' a news site? Can this be determined in an objective way? The generation of fake texts and tweets is deliberate, based on unknown sources, which are trivial and current methodologies in which consumer trustworthiness, truthfulness of news and social media participation are individually validated. In this project, we are trying to produce a model that can predict the likelihood of a fake tweet which might have been written by a machine in a given set of tweets while also validating the same on news articles scraped from the internet.

PROBLEM DESCRIPTION

The problem statement of this research revolves around extracting statistical measures from the language distribution of large language models and incorporating them to build a classification model that classifies a given tweet as ‘human’ generated or ‘machine’ generated and maximizing its accuracy by performing hyper tuning on the parameters selected to build the detector model. The necessity of the system is to truncate the rapid dissemination of the falsified information that pose a threat to social media as a platform. In this project, we strive to utilize social media to classify the news content, particularly Twitter. The aim of the project is to explore and advocate on the selection of determinant statistical features that help detect/classify a news as either ‘human’ or ‘machine’ generated.

MODEL DESCRIPTION

As mentioned in the problem statement, the project deals with identifying fake tweets/news from the given dataset of Tweepfake and Project1-dataset. The implementation involves tasks such as

data preprocessing, feature extraction, training classification models etc. The architecture diagram of the implementation is provided in Fig 1.

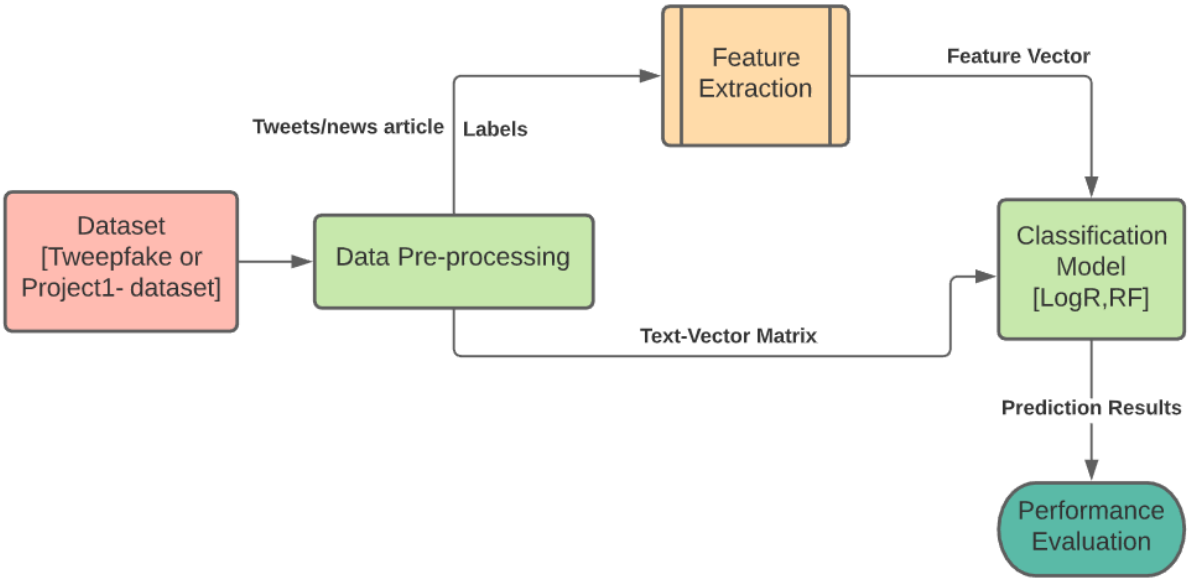

Fig 1: Architecture Diagram

The detailed explanation of each of the steps in the diagram is provided below:

a. Data Preprocessing: The datasets used in this project are: 1) Tweepfake from Kaggle, 2) Project1-dataset. For feature generation, we rely on the Sklearn package's feature extraction functionalities to conduct tokenization, part-of-speech tagging, syntactical parsing, and named entity recognition. Properties like 'CountVectorizer', 'TfidfVectorizer' and 'TfidfTransformer' were used that not only allows for very fast performance compared to other NLP packages such as NLTK but also produces sparse representation of counts and feature selection and helps produce TF-IDF features from raw text.

b. Feature Selection and Extraction: For the machine generated text detection, the actual textual data is being considered as features. However, the data is in text type. It is understood that text data does not perform well for machine learning analysis. So the text data has to be converted into any form of numerical representation. This process is called vectorization. Every record (i.e. Tweets and news articles in this case) should be converted into a vector. There are several techniques/algorithms which can convert text to vectors. Below is the list of such techniques:

    1. Bag of Words (BoW) model: This model converts text to numbers and has a prebuilt set of significant words where it considers the count of each of these words equal to 'n'(number of components). A vector is generated for the word of length n and if the ith word is present in the text(tweet/news) then the ith component in

the corresponding vector will be 1,0 otherwise. This way vectors are generated for all the tweets.

2. Word2Vec: This model helps take the context of the word into consideration . For example, the same word appearing in a text at different locations could have different meanings and Word2Vec takes this into account. This model takes the surrounding words into context to generate the target word which makes it a really interesting approach towards text classification especially with texts related to news.
3. TF-IDF: This model helps assign a weight based on the frequency of appearance and the term frequency component adjusts the weight proportionally with the number of times a word appears in the text with respect to the total number of words in that text. The inverse document frequency variable recognizes unique words in the document set and accordingly increases weight. Sklearn's 'TfidfVectorizer' and 'TfidfTransformer' help convert the raw documents to a matrix of TF-IDF features. 'TfidfTransformer' helps generate scores of words in the document, while 'CountVectorizer' generates features based on the count of words and performs other operations like limit your vocabulary size, apply stop words.

d. Training Classification Model: As explained in the previous sections, the adjacency matrix, feature vectors and the labels form the input for the classification model. We had employed three classification models - Logistic Regression, Support Vector Machines and Random Forest Classifier method for our project using the 'Bag of Words' and 'N-gram' techniques where we saw that the best performing classifier overall was Logistic Regression over the other two.

1. N-gram based categorization: N-grams are N-character slices of a string which include characters found next to each other in a word and include blanks before and after the word. N-gram based categorization. N-gram based categorization calculates and compares profiles of N-gram frequencies where training sets are used as the baseline and this system creates profiles for any documents that need to be classified and measures the distance between them and the category profiles. We used this technique because it yielded a 99.8% accuracy rate across 8 languages in one study [1].
2. Bag of Words model: As discussed above, BoW model uses the occurrence of each word as a feature for training a classifier. It describes the occurrence of words, vocabulary of known words and measure of the presence of known words.

DATASET

1. *TweepFake Dataset*

   The primary dataset we used for the project was the provided TweepFake dataset which is a twitter deep fake dataset which is a collection of human and deep-fake messages. It contains a variety of different columns but the only columns that we were considering in this project were:

   - Tweet text
   - Category of the tweet (human/ machine)

   This dataset contained a total of 20926 equally categorized tweets in the training dataset and 2584 tweets in the test dataset.

2. *Project 1 Dataset*

   The other dataset that we used was the dataset that we created in the first phase of the project. It was a combination of news articles by humans scraped from various news websites and machine generated news articles which were made using HuggingFace and DeepAI. This dataset was modified for it to only contain two columns:

   - News text
   - Category of the news (human/machine)

This dataset contained a total of 198 equally categorized news articles in the training dataset and 49 news articles in the test dataset.

RESULTS

Metrics	**Logistic Regression**	**Support vector machine**	**Random Forest Classifier**
Accuracy	76%	73%	73%
Precision	0.78	0.77	0.82
Recall	0.78	0.77	0.78
F1-score	0.78	0.77	0.77

Table 1: Bag of Words technique(dataset: Tweepfake)

Metrics	**Logistic Regression**	**Support vector machine**	**Random Forest Classifier**
Accuracy	74%	76%	65%
Precision	0.78	0.78	0.79

Recall	0.77	0.78	0.73
F1-score	0.78	0.78	0.72

Table 2: N-grams technique(dataset: Tweepfake)

Using the different classification models to train our data we found out different results for both the datasets:

As we can see in Table 1, For the TweepFake dataset the highest accuracy was achieved by using the bag of words features into the logistic regression classifier.

- Logistic Regression using bag of words gave us 78% accuracy with the highest F1-score of 0.77 followed by Support Vector Machine using the n-gram categorization with an accuracy of 78% and F1-score of 0.765.
- The worst performing algorithm for this dataset was Random Forest with 73% accuracy and 0.65 F1-score.

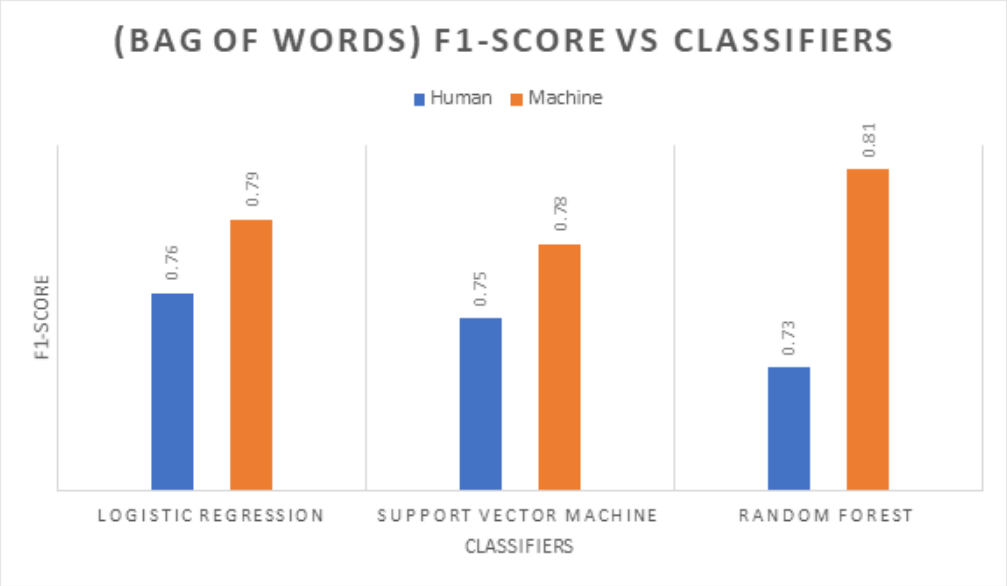

Fig 2: TweepFake: F1-score vs Classifiers (Bag of Words)

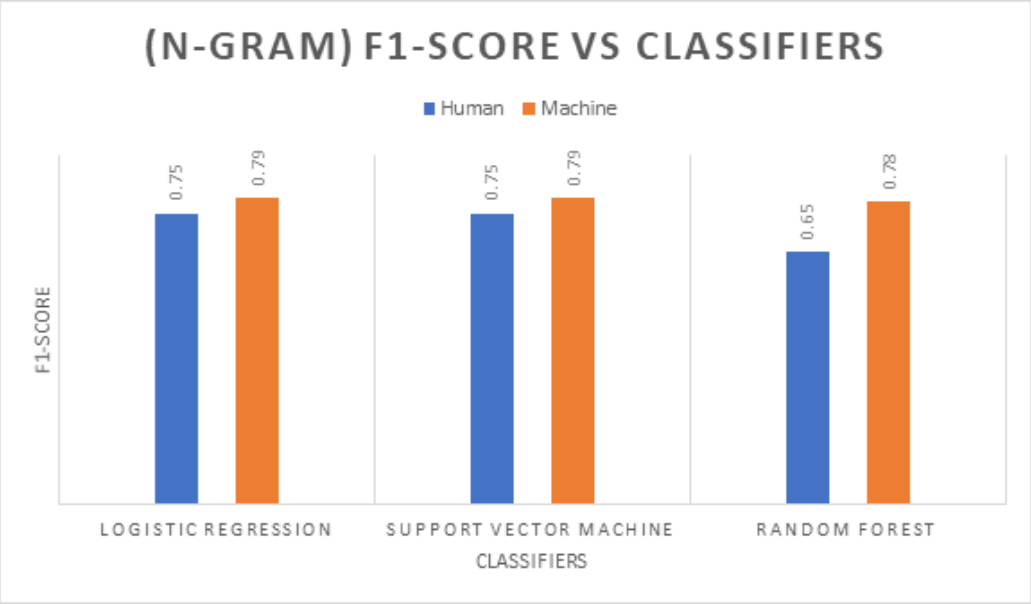

Fig 3: TweepFake: F1-score vs Classifiers (n-gram)

For the Project 1 dataset about machine generated and human written news articles we see a different winner.

- The random forest classifier using the n-grams categorization algorithm gives us the highest accuracy of 100% and a F1-score of 1, followed by Logistic Regression using bag of words with 94% accuracy and a F1-score of 0.945.
- The worst performing algorithm for this dataset was Support Vector Machine using the bag of days with 82% accuracy and a F1-score of 0.8.

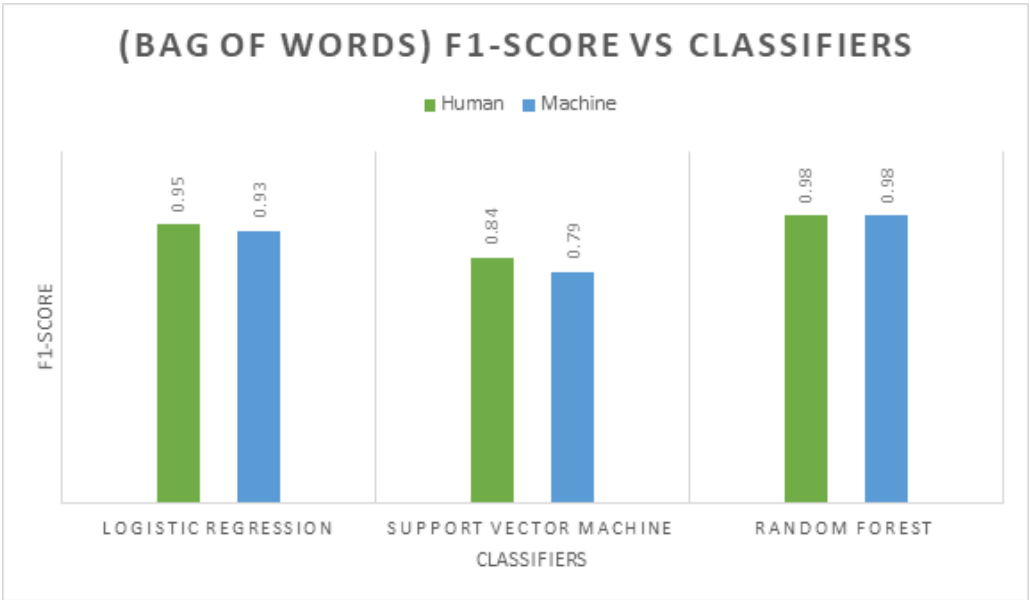

Fig 4: Project-1: F1-score vs Classifiers (Bag of Words)

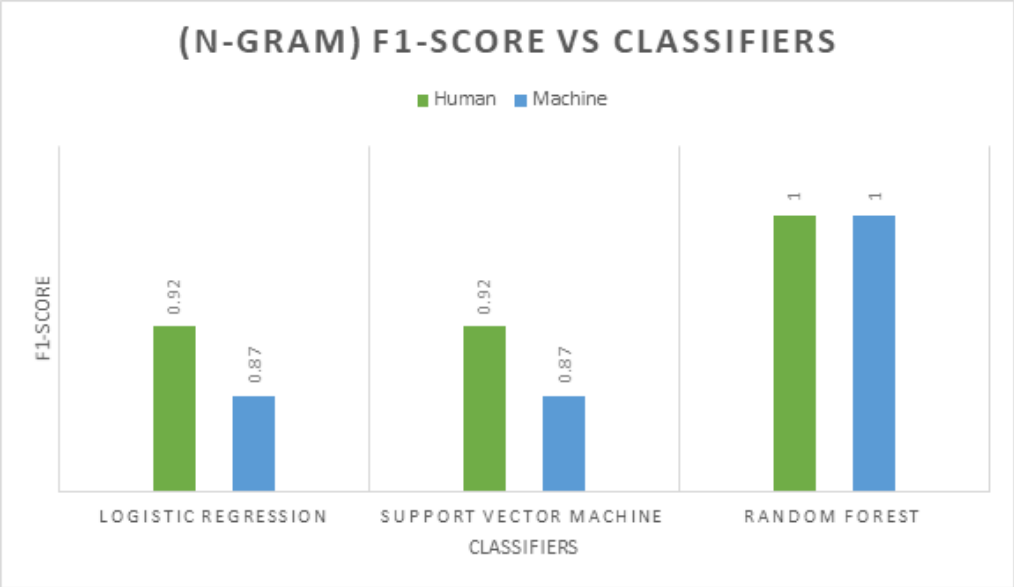

Fig 5: Project-1: F1-score vs Classifiers (n-gram)

DISCUSSION OF RESULTS

The given results for both of the datasets provide us with many interesting insights:

- Our project deals with text classification, which with the help of vectorization techniques, the text is firstly converted to numerical values. Generally, Random Forest performs better than Logistic Regression when the values are categorical and unbalanced, but this is not our case. Hence, this explains why Logistic Regression performs better than random forest here irrespective of the technique used.
- For smaller datasets like that of the Project-1 dataset, random forest performs the best and returns a 100% accuracy over logistic regression and support vector machine. As impressive as it may sound, we can easily infer that this is a case where the dataset has been overfitted into the model. We can support this by inferring from our results that Random Forest performs the worst when it comes to a larger size 'TweepFake' dataset but performs the best when the size of the dataset is smaller, which is in case of our Project1-dataset. Hence, even with lesser accuracy, logistic regression does not completely overfit the data. Being a linear model, it is less prone to overfitting when compared to random forest. Hence, logistic regression is our choice of model.
- Generally, we observed that random forest has a longer execution time in comparison to Logistic Regression and we observed that as part of our experiment as well. Hence, based on as little 20000 values we see that random forest is much slower which might increase as the data size increases.
- We also observed that overall logistic regression is more balanced in comparison to other algorithms(SVM, RF). Through the graphs and tables (Fig 2 and Table 1), we notice that the best classifier to classify only the machine generated text properly is Random Forest

using the “bag of words” technique but that is not balanced and does not provide proper classification results for human written text. This balanced classification result of logistic regression is why it is chosen over Random Forest.

With all of the above-mentioned points into consideration, we can say that for machine generated text vs human written text classification, after performing parameter tuning, Logistic Regression using a bag of words performs the best and is our chosen model among all the other models for text classification.

PARAMETER TUNING [ADDITIONAL]

We incorporated the aspects of tuning the parameters for our project, so we tried manual parameter tuning as well as fine tuning the parameters using the Grid Search. We used GridSearchSV for this purpose.

1. Manual Parameter Tuning:
    a. *Logistic Regression*
    We tried changing the penalty parameter to using l1 instead of l2 but instead changing this led to a lower accuracy evaluation. We also tried changing the solver but none of them provided better accuracy compared to the “lbfgs” solver. Also, the max_iterations were altered as the algorithm did not converge when a lower value than 10,000 was used.

    b. *Support Vector Machine*
    We changed the max_iterations in this case too because the program would not converge on lower values. On changing the shrinking heuristic to False and the degree as 4 we found that the program ran for longer and had lower accuracy of 85% for the TweepFake dataset and hence we decided to have a shrinking heuristic and keep the degree of the polynomial for the vector as 3.

    c. *Random Forest*
    For random forest initially we tried looking into changing the max_depth as we thought that a larger depth value might provide us with a model that is better but instead, we found ourselves with a model that was overfitting as it provided high accuracy on the training dataset but really low accuracy on the test dataset. Changing the criterion from entropy to “gini” did provide us better results, hence we chose gini as the measure of quality for this classifier.

2. Fine Tuning using GridSearch:
    Fine tuning was a new area to explore so we went with GridSearchCV which exhaustively generates different candidates from a grid of parameter values that we provided to the algorithm. For logistic regression, we checked parameters like range of the idf values, to use or not to use idf which means to use the term frequency only or to use both term

frequency and the inverse document frequency and to smooth and not too smooth ‘idf’ values which is to allow unknown words to be processed which was not earlier seen in the training dataset. Similarly, we explored the depth of the tree for random forest to see at what point the model starts overfitting and concluded that 12 depth works well.

REFERENCES

1. Cavnar, William & Trenkle, John. (2001). N-Gram-Based Text Categorization. Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval.
2. Chervonenkis, Alexey. (2013). Early History of Support Vector Machines. 10.1007/978-3-642-41136-6_3.