Understanding Machine vs Human Generated Text in News – Phase 2

Building a Classifier Model

GROUP 10

Raakesh Sureshkumar
Computer Informatics and Decision Systems Engineering
Arizona State University
rsures14@asu.edu

Dunchuan Wu
Computer Informatics and Decision Systems Engineering
Arizona State University
dunchuan@asu.edu

***Abstract*—With the advent of AI adversaries, there is mass generation of good quality fake news around different topics of interest. These differences are quite subtle and can be identified only by the models that generated it which is quite counter-intuitive. So, in this project focuses on extracting human written news articles by journalists and comparing them with Artificial Intelligence model generated news articles with the prompt as the news article headings.**

Keywords—adversary, artificial neural network, Recurrent Neural Networks, tokenization, Rectified linear unit, Long short-term memory, Summarization, Auto-regressive language generation,

I. INTRODUCTION

The objective of this project is to create a means of classification of the human generated text versus the machine-generated text. There are various parameters of dissimilarities between the human-generated and the machine-generated text. These include the randomness in the text but maintaining the context and hence the flow. The classification of such text has become a major concern now as the machine generation of text has become more advanced which sounds more human. Some of the models that indicate this are the GPT-3 (which is set to release 2020 with training on 3 major datasets) and the huggingface.

This has lead to simplify generation of neural fake news and increased difficulty in identifying the difference between the human edited one and the machine generated one.

Here we use some of the contemporary methods to classify a set of machine generated text and human generated text that was obtained from the previous phase of the project. The main objective of the previous phase of the project was to use web automation technology like Selenium to scrape through the news websites to get genuine articles and also use the article headings to generate adversaries using recent technologies like the GPT-2. In an abstract sense, The text dataset is supposed to be preprocessed, converted to word vectors to be processed and fed into Deep neural nets to classify the text.

II. LITERATURE REVIEW

A. *Jeffrey Pennington, Richard Socher, Christopher D. Manning*

GloVe: Global Vectors for Word Representation

GloVe is an unsupervised learning algorithm, developed at Stanford, for obtaining vector representations for words. The statistics of word occurrences in a corpus is the primary source of information available to all unsupervised methods for learning word representations.

Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase linear substructures of the word vector space. The corpus mentioned here is a pre-defined corpus that this taken from a plethora of topics with millions of words.

One of the best examples to explain the relative co-occurence of probability between the words is explained in an efficient manner using the below example. the appropriate starting point for word vector learning should be with ratios of co-occurrence probabilities rather than the probabilities themselves.

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

where w corresponds to a vector of dimension ‘d’ are word vectors and w^~ corresponds to a vector of dimension ‘d’ are separate context word vectors Further to encode the information present in the ratio P_ik / P_jk. We can restrict our consideration to those functions F that depend only on the difference of the two target words. This is valid because vector spaces are inherently linear in structure.

$$F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

The final equation to solve this can be given by,

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik})$$

Where the w represents the weights and b represents the bias as in a simplified linear model. The X represents the function F as

$$w_i^T \tilde{w}_k = \log(P_{ik}) = \log(X_{ik}) - \log(X_i)$$

However, there seems to exist a drawback to this model. The main drawback would be that it weighs all the co-occurrences equally, eventhough some occur rarely or never at all. This according to the paper can be mitigated by a new weighted least squares approach within the regression models during the error correction. The following is the cost function pertaining to the model.

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

The function f(x) pertains to (x / x_max)^\\alpha when x < x_max and 1 otherwise. Here the value of \alpha was found out to be ¾ on experiment. It is noted that all the unsupervised methods for learning word vectors are ultimately based on the occurrence statistics of the words within the corpus. So, the overall cost function can be approximated to be,

$$J = -\sum_{i=1}^{V} X_i \sum_{j=1}^{V} P_{ij} \log Q_{ij} = \sum_{i=1}^{V} X_i H(P_i, Q_i)$$

Where Q_ij is the softmax function for the error calculation and J is the cost function. H represents the cross entropy for the approximation of the errors.

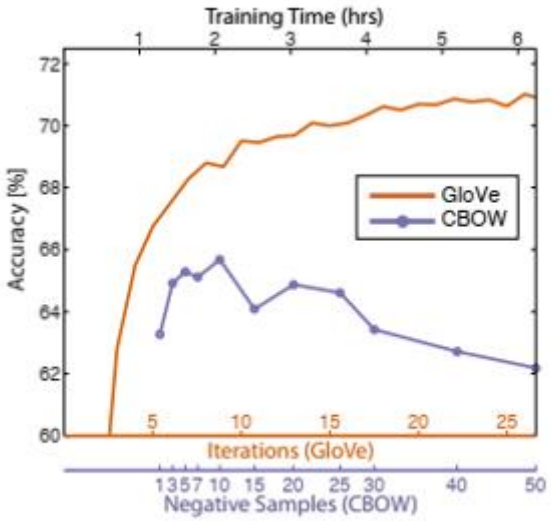
$$Q_{ij} = \frac{\exp(w_i^T \tilde{w}_j)}{\sum_{k=1}^{V} \exp(w_i^T \tilde{w}_k)}$$

The evaluation of the model included the parameters like word analogies, which could test the textual aptitude of the system to respond. The word analogy task consists of questions like, “a is to b as c is to ?”

The dataset contains 19,544 such questions, divided into a semantic subset and a syntactic subset. GloVe records the highest with a corpus of 42B with an accuracy of 81.9 in Semantic analysis and in syntactic definition it has an accuracy of 69.3 with a dataset of 42B as corpus. This was the main focus of this project application. GloVe performed good in other aspects like word similarity, Named Entity Recognition (NER)

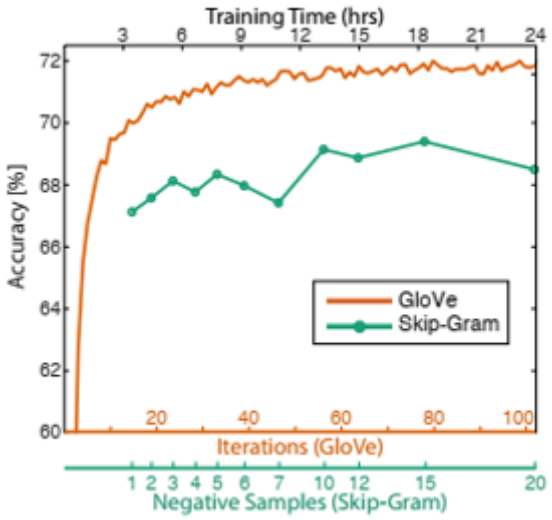
The datasets included used a corpus from Gigaword5 and Wikipedia2014 containing 6 billion tokens and 42 billion tokens from the Common Crawl. The experimental setup included an optimum \alpha of 0.75 as specified in the cost function for optimization. Using the spearman rank correlation task on word similarity, all vectors being of 300 dimensions, the GloVe seems to be on par with the training of word2vecby Google. The skip-gram (SG) and continuous bag-of-words (CBOW) models were trained on the 6 billion token corpus (Wikipedia 2014 + Gigaword 5) with

a vocabulary of the top 400,000 most frequent words and a context window size of 10.



(a) GloVe vs CBOW

FIGURE 1 TRAINING ACCURACY FOR BAG OF WORDS VS GLOVE



(b) GloVe vs Skip-Gram

FIGURE 2 TRAINING ACCURACY FOR SKIP-GRAM VS GLOVE

B. Shervin Minaee, Nal Kalchbrenner, Amsterdam
Erik Cambria, Narjes Nikzad, Tabriz Jianfeng Gao, Redmond.

Deep Learning Based Text Classification: A Comprehensive Review

arXiv:2004.03705v2 [cs.CL] 17 Nov 2020

Overview: Text classification, also known as text categorization, is a classical problem in natural language processing (NLP), which aims to assign labels or tags to textual units such as sentences, queries, paragraphs, and documents. It has a wide range of applications including question answering, spam detection, sentiment analysis, news categorization, user intent classification, content moderation, and so on. Text data can come from different sources, including web data, emails, chats, social media, tickets, insurance claims, user reviews, and questions and answers from customer services, to name a few. Text is an extremely rich source of information. But extracting insights from text can be challenging and time-consuming, due to its unstructured nature.

Text Classification (TC) is the process of categorizing texts (e.g., tweets, news articles, customer reviews) into organized groups. Typical TC tasks include sentiment analysis, news categorization and topic classification. Recently, researchers show that it is effective to cast many natural language understanding (NLU) tasks (e.g., extractive question answering, natural language inference) as TC by allowing DL-based text classifiers to take a pair of texts as input. This section introduces five TC tasks discussed in this paper, including three typical TC tasks and two NLU tasks that are commonly cast as TC in many recent DL studies.

In the top-k sampling, a kth token is defined, which forms the threshold for filtering out the sorted probabilities that are below the kth token. This ensures an improvement in quality and accuracy of text generation which sticks to the context. By quality, here we refer the coherence in the text. The problem here is that we are not sure at what optimum value for the kth token, there are words with reasonable sampling that form a broad distribution or in some others there are none with a narrow distribution. In our example, we have taken this value as 200.

RNNs are trained to recognize patterns across time, whereas CNNs learn to recognize patterns across space .RNNs work well for NLP tasks such as POS tagging or QA where the comprehension of long-range semantics is required, while CNNs work well where detecting local and position-invariant patterns is important.

These patterns could be key phrases that express a particular sentiment like “I like” or a topic like ”endangered species”. Thus, CNNs have become one of the most popular model architectures for text classification. One of the first CNN-based models for text classification is proposed by Kalchbrenner et al. This model uses dynamic *k*-max pooling, and is called the Dynamic CNN (DCNN). The first layer of DCNN constructs a sentence matrix using the embedding for each word in the sentence. Then a convolutional architecture that alternates wide convolutional layers with dynamic pooling layers given by dynamic *k*-max pooling is used to generate a feature map over the sentence that is capable of explicitly capturing short and long-range relations of words and phrases. The pooling parameter *k* can be dynamically chosen depending on the sentence size and the level in the convolution hierarchy.

Character-level CNNs have also been explored for text classification [50, 51]. One of the first such models is proposed by Zhang et al. [50]. As illustrated in Fig. 6, the model takes as input the characters in a fixed-sized, encoded as one-hot vectors, passes them through a deep CNN model that consists of six convolutional layers with pooling operations and three fully connected layers. Prusa et al. [52] presented a approach to encoding text using CNNs that greatly reduces memory consumption and training time required to learn character-level text representations. This approach scales well with alphabet size, allowing to preserve more information from the original text to enhance classification performance.

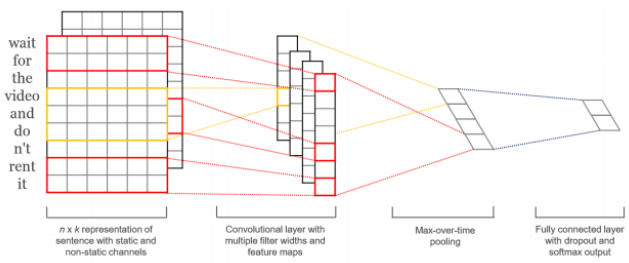


FIGURE 3 CNN LAYER DESCRIPTION INVOLVING 1D CONV ; MAX POOL AND FULLY CONNECTED LAYER

III. MODEL DESCRIPTION

In this section a brief describe is given on how the data is collected, pre-processed for feature extraction and sent to a classifier model to obtain the proper binary classification of a human or machine generated news article or text.

A. *Text Preprocessing*

Text normalization is an important step in any kind of Natural Language Process. It involves extracting meaning information from the text just like humans perceive. A text could have multiple disparities like missing words, abbreviations, shortforms etc. Also, the text needs to be converted to an interpretable format for further processing. This action can be performed by text pre-processing techniques like stemming, lemmatization, stop words removal, noise reduction etc.

Tokenization

To begin with, the process of text preprocessing takes the original data we read to transform data in a form that we can easily manipulate. In a word, in this phase, we extract every single word from human-generated and machine-generated articles. The extracted word count is similar to the result returned by the function “word count” that comes within the text platform like microsoft word or google doc online. The content of each word is a mixture of words that have a regular and irregular combination of letters and symbols. To be specific, a normal word is regular and a word that contains symbols like “He’s”, ““really””, “(maybe)”. This is expected to happen and we will have various methods to clean the data so that every word becomes regular. Once we have the nice-looking words, we can count again and compute their frequency as helpful indicators.

Beyond this, we take into account that a simple word like “a”, “an”, “the” may contribute to a large amount of word count and have less meaningful content.

Uncleaned:

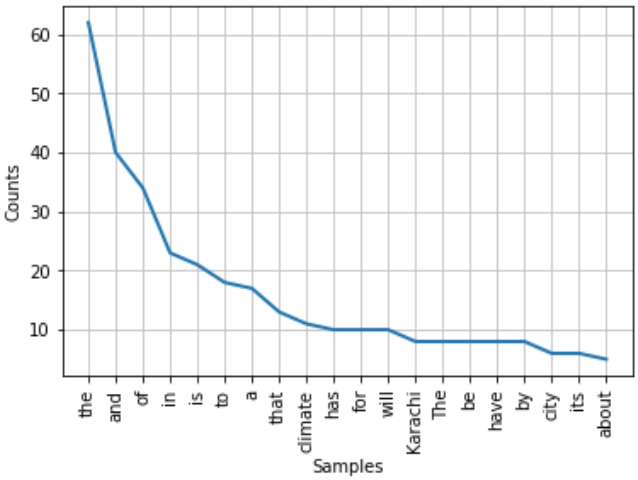


FIGURE 4 TOKENIZATION : UNCLEANED SAMPLES VS COUNTS

Cleaned:

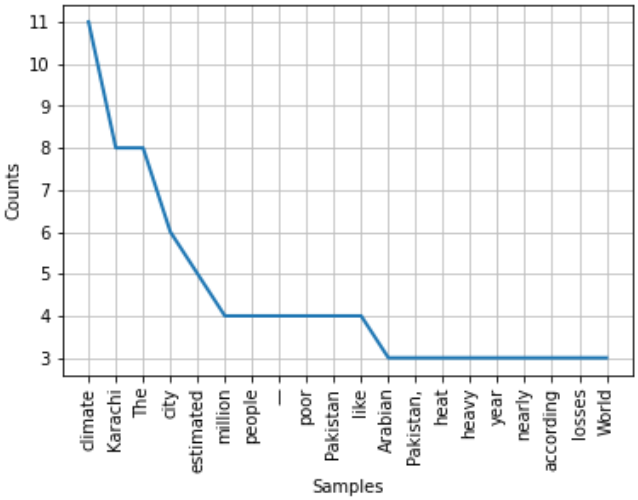


FIGURE 5 TOKENIZATION : CLEANED SAMPLES VS COUNTS

Lemmatization

This task takes place after we successfully processed data in the form of tokens. Considering the word list size in practical or business-level use may be up to thousands of times to ours, the processing time is a big issue. However, although CPU performance on the consumer's markets is impressive enough, compressing the data is one of the most important steps in text preprocessing.
Some common words like "talked", "talks" and "talking" can be counted multiple times more than "talk" in word list size, which is not a desirable way to count. Lemmatization can efficiently reduce it by simplifying those words as one single word.

Stemming

This step is quite similar with Lemmatization, but the difference between them is the result. A word "courses" might be transformed into "cours" which is not a valid word. However, it takes less time to process words at this step. On the other hand, lemmatization takes every word as input and produces a valid word ("courses" to "course"), but it takes longer time. Both lemmatization and stemming can reduce the data size significantly.

Noise Removal

This is a very important note-taking characteristic where people like journalists and physics scribble out symbols that are recognizable only by them. For processing the noise we can use regular expressions as a filter. These symbols though may contain some meaning for humans might not be useful for processing the data and could cause only confusion to the system while processing. It is also highly domain dependent. For example, in Tweets, noise could be all special characters except hashtags as it signifies concepts that can characterize a Tweet. The problem with noise is that it can produce results that are inconsistent in your downstream tasks. In our case, the noise could be due to punctuation marks and scientific symbols that could explain climate related effects (especially in science articles. But, as our database is mainly a computer-generated and typed database, there won't be much noise as in tweets and hand taken notes.

In case of news articles, quotes from reporters and scientists could be of some significance too. This may not be of greater importance during the tokenization. But, some characters like an exclamation could be of significance as it could mean, it has been edited by a journalist and involves the expression of a person. Also, accented characters need to be removed.

Approach 1: Stemming as discussed in the other section can entertain noise removal but can take away useful information which could affect tokenization and hence give out non-contextual words which could result in a poor performance.

Approach 2: Regex based word analysis and cleaning. This approach helps keep the word

intact and also provides an actual word that exists in the english vocabulary.

	raw_word	cleaned_word	stemmed_word
0	..trouble..	trouble	troubl
1	trouble<	trouble	troubl
2	trouble!	trouble	troubl
3	<a>trouble	trouble	troubl
4	1.trouble	trouble	troubl

FIGURE 6 STEMMING EXAMPLE

Stop words

These are the set of words that exist in every language and do not provide context to a speech, an article or any other kind of communication content. A wonderful example would be that of a search engine. If our search query is “how to create a feature map on english vocabulary”, the search engine tries to find web pages that contained the terms “how”, “to” “develop”, “information”, ”retrieval”, “applications” the search engine is going to find a lot more pages that contain the terms “how”, “to” than pages that contain information about developing information retrieval applications because the terms “how” and “to” are so commonly used in the English language. If we disregard these two terms, the search engine can actually focus on retrieving pages that contain the keywords: “develop” “information” “retrieval” “applications” – which would bring up pages that are actually of interest. This is just the basic intuition for using stop words.

Similarly, in our case of news content, the textual data may contain words like “is, an, the, many, more, about etc” which can be considered meaningless as far as the context is concerned. There are 2 disadvantages for our application. The first being increased input feature vectors that could result in increased requirement for processing power and hence reduced speed. Also, it could affect the accuracy of the classification as these words are not supposed to contribute significant information to the final results.

Application: We are using the stop words vocabulary provided by the CountVectorizer provided by sklearn machine learning library. It has a list of 318 words. Now, not all words in these 318 would be meaningless for our application. For example, in our case, we have a topic of climate change. So, we might have to keep some of the words like location-based (anywhere, there), temporal (when, afterwards, whenever), numbers (one through 10, 20,30...100 in words) are of importance as they are closely related to the topic climate. So, they have a chance of providing context to the words and hence help us out in creating meaningful feature vectors.

Types:

- Determiners – Determiners tend to mark nouns where a determiner usually will be followed by a noun
examples: the, a, an, another
- Coordinating conjunctions – Coordinating conjunctions connect words, phrases, and clauses
examples: for, an, nor, but, or, yet, so
- Prepositions – Prepositions express temporal or spatial relations
examples: in, under, towards, before

Our stop words list: 'a', 'about', 'above', 'across', 'again', 'against', 'all', 'almost', 'alone', 'along', 'already', 'also', 'although', 'am', 'among', 'amongst', 'amount', 'an', 'and', 'another', 'any', 'anyhow', 'anyone', 'anything', 'anyway', 'are', 'as', 'at', 'back', 'be', 'became', 'because', 'become', 'becomes', 'been', 'behind', 'being', 'below', 'beside', 'besides', 'between', 'bill', 'both', 'but', 'by', 'call', 'can', 'cannot', 'cant', 'co', 'con', 'could', 'couldn’t, 'cry', 'de', 'describe', 'detail', 'do', 'done', 'down', 'due', 'during', 'each', 'eg', 'either', 'else', 'empty', 'enough', 'etc', 'even', 'ever', 'every', 'everyone', 'everything', 'except', 'few', 'fill', 'find', 'fire', 'for', 'former', 'formerly', 'found', 'from', 'front', 'full', 'further', 'get', 'give', 'go', 'had', 'has', 'hasn’t, 'have', 'he', 'hence', 'her', 'here', 'hereby', 'herein', 'hereupon', 'hers', 'herself', 'him', 'himself', 'his', 'how', 'however', 'hundred', 'i', 'ie', 'if', 'in', 'inc', 'indeed', 'interest', 'into', 'is', 'it', 'its', 'itself', 'keep', 'last', 'least', 'less', 'ltd', 'made', 'many', 'may', 'me', 'meanwhile', 'might', 'mill', 'mine', 'more', 'moreover', 'most', 'mostly', 'move', 'much', 'must', 'my', 'myself', 'name', 'namely', 'neither', 'never', 'nevertheless', 'nobody', 'none', 'nor', 'not', 'nothing', 'now', 'of', 'off', 'often', 'on', 'onto', 'or', 'other', 'others', 'otherwise', 'our', 'ours', 'ourselves', 'out',

'over', 'own', 'part', 'per', 'perhaps', 'please', 'put', 'rather', 're', 'same', 'see', 'seem', 'seemed', 'seeming', 'seems', 'serious', 'several', 'she', 'should', 'show', 'side', 'since', 'sincere', 'so', 'some', 'somehow', 'someone', 'something', 'sometime', 'sometimes', 'somewhere', 'still', 'such', 'system', 'take', 'than', 'that', 'the', 'their', 'them', 'themselves', 'then', 'thence', 'there', 'thereafter', 'thereby', 'therefore', 'therein', 'thereupon', 'these', 'they', 'thick', 'thin', 'this', 'those', 'though', 'through', 'throughout', 'thru', 'thus', 'to', 'together', 'too', 'top', 'toward', 'towards', 'un', 'under', 'until', 'up', 'upon', 'us', 'very', 'via', 'was', 'we', 'well', 'were', 'what', 'whatever', 'whence', 'whereafter', 'whereas', 'whereby', 'wherein', 'whereupon', 'wherever', 'whether', 'which', 'while', 'whither', 'who', 'whoever', 'whole', 'whom', 'whose', 'why', 'will', 'with', 'within', 'without', 'would', 'yet', 'you', 'your', 'yours', 'yourself', 'yourselves'

Word Contractions

This is a very important note-taking characteristic where people like journalists and physics scribble out symbols that are recognizable only by them. These symbol though may contain some meaning for humans might not be useful for processing the data and could cause only confusion to the system while processing.

B. Vectorization

This step is one of the important steps for further processing of the text. The text cannot be processed as it is for classification as it involves numerical functions that need to be evaluated in order to be optimized and obtain the desired result. So, the given text text is converted into a set of meaningful vectors that are correlated based upon the word count and their relation with the context and the document frequency and the word occurrences.

Count Vectorizer

In our case, scikit-learn's CountVectorizer is used to convert a collection of text documents to a vector of term/token counts. It also enables the pre-processing of text data prior to generating the vector representation. This functionality makes it a highly flexible feature representation module for text.

The lowercase in text also matters in our case because. Neural and LSTM text generators can differ in sentence construction and the capitalization of the text wherever required. For example, to mention an exclamation in a sentence, that particular phrase can be capitalized in case of a human-generated text. But, in terms of a machine generation still there is a random probability of generating such and still less probability for even a Camelcase to occur. So, it can be important to disable the lowercase in order to maintain the case sensitivity while training the model.

An n-gram model is a type of probabilistic language model for predicting the next item in such a sequence in the form of a $(n-1)$–order Markov model. Two benefits of n-gram models (and algorithms that use them) are simplicity and scalability – with larger n, a model can store more context with a well-understood space–time tradeoff, enabling small experiments to scale up efficiently. The LSTMs are inherently n-gram models as they rely on the preceding and forth-coming words. Though this model is widely used in text generation, it is also used in vectorization to provide higher context to a word in a dictionary based upon the underlying corpus.

Parameters related to the n-grams includes an analyzer that decides whether the n-grams should be based on a word or a character n-gram. In our case, we have chosen a word n-gram and there might not be a necessity to check for the text character by character for the vectorization while it may be required in case of a chatbox. For the range, we go ahead with the bigram, as it may provide more insight into the context of the text.

Max_df and Min_df are used when building the vocabulary to ignore terms that have a document frequency strictly higher than the given threshold (corpus-specific stop words). Where any word within this threshold is taken into considered. Generally according to the TfIdf, the higher the value, the more common it is and the less contextual it is. So, it is advisable to remove the words that have a huge chance of occurrence. So, we give the Max_df as 0.8 and all the text below this value are prompt removed before the tokenization process. This also saves time and confuses the model less thereby improving the accuracy.

Term frequency Inverse text document frequency Vectorizer

This is a step ahead of the count vectorizer. Apart from the count vectorizer functionality, it also has the inverse document frequency parameters that can be used to filter out the text that appear frequently in a document and do not provide context to it.

The formula that is used to compute the tf-idf for a term t of a document d in a document set is tf-idf(t, d) = tf(t, d) * idf(t), and the idf is computed as idf(t) = log [n / df(t)] + 1 (if smooth_idf is False), where n is the total number of documents in the document set and df(t) is the document frequency of t; the document frequency is the number of documents in the document set that contain the term t. The effect of adding "1" to the idf in the equation above is that terms with zero idf, i.e., terms that occur in all documents in a training set, will not be entirely ignored. (Note that the idf formula above differs from the standard textbook notation that defines the idf as idf(t) = log [n / (df(t) + 1)]). There is also an added feature that the formulas used to compute tf and idf depend on parameter settings that correspond to the SMART notation used in IR. Tf is "n" (natural) by default, "l" (logarithmic) when sublinear_tf = True. Idf is "t" when use_idf is given, "n" (none) otherwise. Normalization is "c" (cosine) when norm=l2, "n" (none) when norm=None.

C. *Choosing between pretrained vectorizers for word embeddings*

Word2Vec

When researchers of Google Tomas Mikolov et.al published this paper, they proposed an approach, famously known as Word2Vec. It uses small neural networks to calculate word embeddings based on words' context. There are two approaches to implement this approach.

First, there is the continuous bag of words or CBOW. In this approach, the network tries to predict which word is most likely given its context. Words that are equally likely to appear can be interpreted as having a shared dimension. If we can replace "cat" with "dog" in a sentence, this approach predicts a similar probability for both. Therefore, we infer that the meaning of these words is similar on at least one level.

The second approach is skip-gram. The idea is very similar, but the network works the other way around. That is, it uses the target word to predict its context. See the links at the end of this post for more details.

When Word2Vec came out in 2013, the results were unprecedented but also hard to explain from a theoretical point of view. It worked, but there was some confusion why.

Global Vector representation (GloVe)

One year later researchers of Stanford published GloVe. As Word2Vec learns embeddings by relating target words to their context, it ignores whether some context words appear more often than others. For Word2Vec, a frequent co-occurrence of words creates more training examples, but it carries no additional information.

In contrast, GloVe stresses that the frequency of co-occurrences is vital information and should not be "wasted "as additional training examples. Instead, GloVe builds word embeddings in a way that a combination of word vectors relates directly to the probability of these words' co-occurrence in the corpus. GloVe is not a trained model in the same sense that Word2Vec is. Instead, its embeddings can be interpreted as a summary of the training corpus with low dimensionality that reflects co-occurrences.

For our project, we have chosen to use this particular model as it has data adherent to the relative probability of the text within the corpus. It is a step further to the Word2Vec mode. Also, this model is divided into a wide range of corpus which is memory friendly and can be run on out local system.

FastText

In 2016, artificial neural nets had gained quite some traction, and Word2Vec has proven its usefulness in many areas of NLP. However, there was one unsolved problem: generalization to unknown words. FastText — a development by

Facebook released in 2016 — promised to overcome this obstacle.

The idea is very similar to Word2Vec but with a major twist. Instead of using words to build word embeddings, FastText goes one level deeper. This deeper level consists of part of words and characters. In a sense, a word becomes its context. The building stones are therefore characters instead of words.

The word embeddings outputted by FastText look very similar to the ones provided by Word2Vec. However, they are not calculated directly. Instead, they are a combination of lower-level embeddings.

There are two main advantages to this approach. First, generalization is possible as long as new words have the same characters as known ones. Second, less training data is needed since much more information can be extracted from each piece of text. That is why there are pre-trained FastText models for way more languages than for every other embedding algorithm.

D. *Basic classifier models*

Linear support vector classifier

The objective of a Linear SVC (Support Vector Classifier) is to fit to the data that is provided, to return a "best fit" hyperplane that divides, or categorizes our training data. From there, after getting the hyperplane, we could then feed our text features to our classifier to see what the "predicted" class is. This makes this specific algorithm rather suitable for our use.

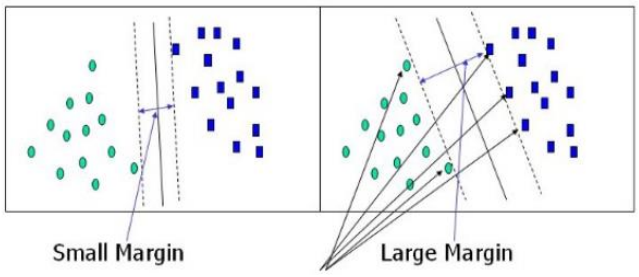


FIGURE 7 HYPER-PLANE PLOT FOR SVC

The sklearn library in python is used in our project to create this particular model.

Naïve Bayes classifier

In statistics, Naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features. They are among the simplest Bayesian network models, but coupled with Kernel density estimation, they can achieve higher accuracy levels.

Naïve Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) [in our case the pre-processed word sequences] in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

The NaiveBayes() functionality from the sklearn library in python is used in our project to create this particular model.

Logistic regression classifier

In statistics, the logistic model (or logit model) is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick. In our case, we need to detect if a particular news item is human created or machine generated.

In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labeled "0" and "1" [in our case 0 being the human news article and 1 being the machine generated news article.

This model is a pre-defined function in the sklearn package in python which has been used in our project.

E. Neural network models

Neural networks form the base of the customized model starting from a perceptron to a much Deep network consisting of huge number of layers and input / output nodes within every layer. Every layer carries I/O nodes which carry a weight and a bias. These weights and bias are adjusted for every back propagation in a neural net. So, in essence every node in a neural network acts as a simple linear regression model.

Every function, including the initial neuron receives a numeric input, and produces a numeric output, based on a internalized function, which includes the addition of a bias term, which is unique for every neuron. That output is then converted to the numeric input for the function in the next layer, by being multiplied with an appropriate weight. This continues until one final output for the network is produced. The difficulty lies in determining the optimal value for each bias term, as well as finding the best weighted value for each pass in the neural network. To accomplish this, one must choose a cost function.

A cost function is a way of calculating how far a particular solution is from the best possible solution. There are many different possible cost functions, each with advantages and drawbacks, each best suited under certain conditions. The entire neural network converges to a global optimum based on a cost function which determines the error generated for every iteration throughout all the layers. The final decision lies in the optimization of the weights and bias in such a way that the error generated by the cost function is minimized to the least.

Choice of neural network

Recurrent neural network

Recurrent Neural Network remembers the past and it's decisions are influenced by what it has learnt from the past. Note: Basic feed forward networks "remember" things too, but they remember things they learnt during training. For example, an image classifier learns what a "1" looks like during training and then uses that knowledge to classify things in production.

While RNNs learn similarly while training, in addition, they remember things learnt from prior input(s) while generating output(s). It's part of the network. RNNs can take one or more input vectors and produce one or more output vectors and the output(s) are influenced not just by weights applied on inputs like a regular NN, but also by a "hidden" state vector representing the context based on prior input(s)/output(s). So, the same input could produce a different output depending on previous inputs in the series. a network becomes "recurrent" when you repeatedly apply the transformations to a series of given input and produce a series of output vectors. There is no pre-set limitation to the size of the vector. And, in addition to generating the output which is a function of the input and hidden state, we update the hidden sate itself based on the input and use it in processing the next input.

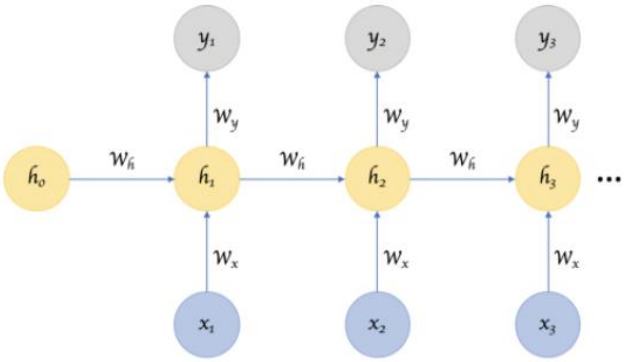


FIGURE 8 RNN SEQUENCING EXAMPLE

Here "w"s represent the weights, "h" represent the activation functions and x represent the inputs. These are the base neural networks in the transformer-based models that have emerged in the recent years. Around 2016, BERT (Bidirectional Encoder Representation from Transformers) was introduced which had the capability of LSTMs (Lon-short term memory). This helped reshape the world of Natural Language Processing and involved deeply in the development of the sequence models.

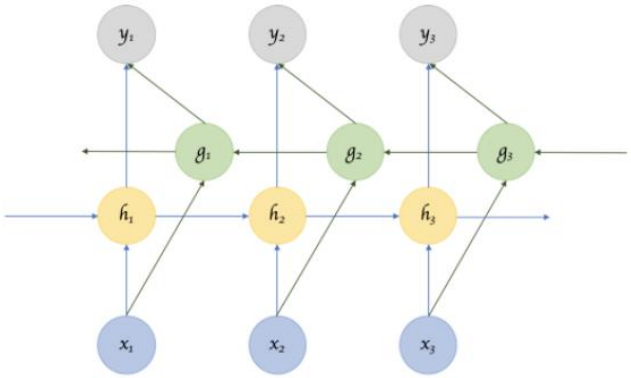


FIGURE 9 RNN LSTM EXAMPLE

Convolution neural network

While RNNs are good at Lon-Short Term Memory models, a variation of the vanilla neural network is the convolution neural network. ConvNets, as they are sometimes known offer some significant advantages over normal neural nets, especially when it comes to image classification.

In our case, we have tried using a CNN for text sequence / news article classification. This may be better than RNNs for certain reasons. In such a case, the initial inputs would be text sequences, made up of feature vectors. The traditional issue with news classification is that with huge inputs, with many embeddings, is that it quickly becomes computationally infeasible to train some models. What CNN tries to do is transform the text into a form which is easier to process, while still retaining the most important features. This is done by passing a filter over the initial matrix, which conducts matrix multiplication over a subsection of the vectors in the initial adversary, it iterates through subsets until it has considered all subsets. The filter aims at capturing the most crucial features, while allowing the redundant features to be eliminated. This passing of a filter over the initial feature matrix is known as the Convolution Layer.

After the convolution layer comes the pooling layer, where the spatial size of the convoluted features will be attempted to be reduced. The reduction in complexity, sometimes known as dimensionality reduction will decrease the computational cost of performing analysis on the data set, allowing the method to be more robust. In this layer, a kernel once again passes over all subsets of vectors of the text sequence. There are two types of pooling kernels which are commonly used. The first one is Max Pooling, which retains the maximum value of the subset. The alternative kernel is average pooling, which does exactly what you'd expect: it retains the average value of all the features in the matrix of the adversary.

F. *Optimizing neural network parameters*

This is an important step in any of the neural network optimization algorithm. Hyperparameter optimization or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are learned.

The same kind of machine learning model can require different constraints, weights or learning rates to generalize different data patterns. These measures are called hyperparameters, and have to be tuned so that the model can optimally solve the machine learning problem. Hyperparameter optimization finds a tuple of hyperparameters that yields an optimal model which minimizes a predefined loss function on given independent data. The objective function takes a tuple of hyperparameters and returns the associated loss. Cross-validation is often used to estimate this generalization performance.

The two main methods used in the industry are discussed below and the one chosen by us is also mentioned as follows.

Grid search

The traditional way of performing hyperparameter optimization has been grid search, or a parameter sweep, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a held-out validation set.

Since the parameter space of a machine learner may include real-valued or unbounded

value spaces for certain parameters, manually set bounds and discretization may be necessary before applying grid search.

Randomized search

Random Search replaces the exhaustive enumeration of all combinations by selecting them randomly. This can be simply applied to the discrete setting described above, but also generalizes to continuous and mixed spaces. It can outperform Grid search, especially when only a small number of hyperparameters affects the final performance of the machine learning algorithm.[2] In this case, the optimization problem is said to have a low intrinsic dimensionality.[5] Random Search is also embarrassingly parallel, and additionally allows the inclusion of prior knowledge by specifying the distribution from which to sample.

Our choice

We have chosen randomized search as it is "embarrassingly parallel" i.e performs all the operations in parallel by specifying the n_jobs which allocates batches and workloads to separate cores of the CPU and also indulges GPU cores to the processing if present. Furthermore it is more advanced compared to grid search and selects the parameters randomly than in order. The following are our randomized cross validation parameters:

```
# Grid search params
param_grid = dict(num_filters=[128, 256],
                  kernel_size=[7, 9, 15],
                  vocabulary_size=[vocab_size],
                  embedding_dim=[100],
                  maxlen=[max_len])
```

FIGURE 10 GRID SEARCH PARAMETERS

From the above, num_filters is the number of filters for the 1D convolution layer, kernel_size is the kernel size for performing the convolution operation on the feature vectors, vocabulary_size specifieis the dictionary size for the input feature vector from the embedding layer, embedding_dim gives the dimension of the input side of the embedding layer from the feature vectors and max_len is 1 added to the vocabulary size from the vectorized feature vector length.

In our case, we use the RandomizedSearchCV from the sklearn package.

IV. FINAL MODEL

Text pre-processing

Lemmatization is performed by using the WordNetLemmatizer class from the nltk.stem package. This brings the words to their root / etymological form. Stemming is not performed as it is found to be less accurate compared to a Lemmatizer though perform the same functions.

The strip_accents of the CountVectorizer from the sklearn package removes certain accented words that come under the latin scripts in the Unicode format.

As mentioned earlier, we have created our own list of stop words that are to be removed from the text and are statistically redundant in both the human and machine generated texts.

Texts are not converted to lowercase as it may affect the accuracy of prediction. There may be mistakes or differences in the machine generated text for case sensitivity which may be missed out during the classification.

For noise removal the following regex pattern is used:

r '\b[a-zA-Z]{3,}\b | [^a-zA-z0-9.,!?/:;\"\'\s] | [^a-zA-z.,!?/:;\"\'\s]'

where represents a raw string without escape sequences:

\b[a-zA-Z]{3,}\b – removes numbers and words less than 3 letters as they seldom convey any meaning to the text

[^a-zA-z0-9.,!?/:;\"\'\s] – removes special characters

[^a-zA-z.,!?/:;\"\'\s] – removes the numbers from the text

Feature extraction and vectorization

In our project this is tested for 2 of the packages i.e the TfIdf vectorizer and the Count vectorizer from sklearn library. This typically converts the words to feature vectors based on their frequency of occurrence within the word document or the input text.

As this might not be fruitful for the training due to the small size of the corpus, we have also used the pre-trained models that have around 6 billion words in their corpus. On researching on the 2 best trained models for the word embedding, we found that GloVe and Word2Vec are trending in the research. For our purpose and reasons mentioned in the previous sections we use GloVe word embedding as the input layer for the neural network.

Classifier model

We have trained and tested with multiple models both neural and non-neural as discussed above. However, the one to focus on would be the convolutional neural network with the following summary obtained using the summary() functionality of the keras toolkit.

Layer (type)	Output Shape	Param #
embedding_28 (Embedding)	(None, 100, 100)	786100
conv1d_27 (Conv1D)	(None, 96, 32)	16032
global_max_pooling1d_27 (Glo	(None, 32)	0
dense_81 (Dense)	(None, 20)	660
dense_82 (Dense)	(None, 10)	210
dense_83 (Dense)	(None, 1)	11

Total params: 803,013
Trainable params: 803,013
Non-trainable params: 0

FIGURE 11 CNN MODEL STRUCTURE WITH EMBEDDING LAYER ; INPUT LAYER ; DEEP LAYERS AND OUTPUT LAYER

The input being the embedding layer embedded using the GloVe model followed by 1 dimensional convolution layer, with the activation function as “Rectified Linear Unit” or relu for short. Then a global max pooling layer followed by 2 dense layers with relu as the activation function. The final is a softmax layer to perform the classification after optimization.

Choice of activation function

The ReLU function is a non-linear activation function that has gained popularity in the deep learning domain. ReLU stands for Rectified Linear Unit. The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time. Earlier sigmoids were used in its place, but ReLU seemed to produce better accuracy due to its slow convergence to the global optima and hence higher accuracy.

This means that the neurons will only be deactivated if the output of the linear transformation is less than 0.

$$f(x)=\max(0,x)$$

The following is a ReLU function plotted

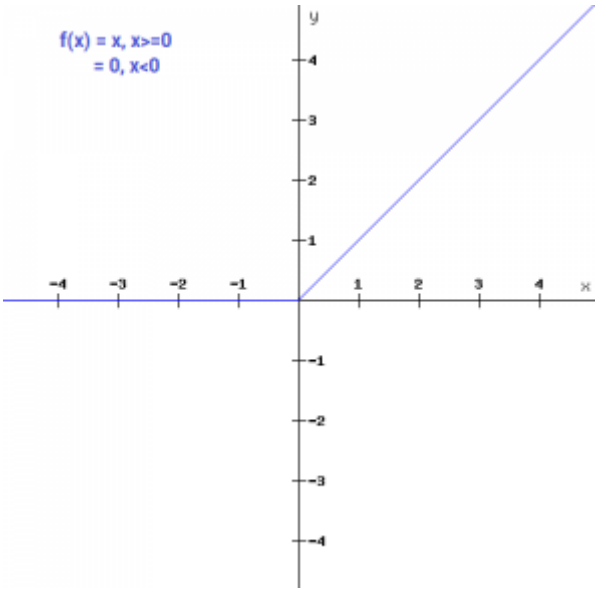


FIGURE 12 RELU ACTIVATION FUNCTION PLOT

Choice of optimizer

In our project, we use the adam optimizer. It realizes the benefits of both Adaptive Gradient and RMS Propagation which are the conventional optimization gradient techniques for convergence. Adam is different to classical stochastic gradient descent. Stochastic gradient descent maintains a single learning rate (termed alpha) for all weight updates and the learning rate does not change

during training. A learning rate is maintained for each network weight (parameter) and separately adapted as learning unfolds. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. The authors describe Adam as combining the advantages of two other extensions of stochastic gradient descent. Specifically:

Adaptive Gradient Algorithm (AdaGrad) that maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems).

Root Mean Square Propagation (RMSProp) that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). This means the algorithm does well on online and non-stationary problems (e.g. noisy).

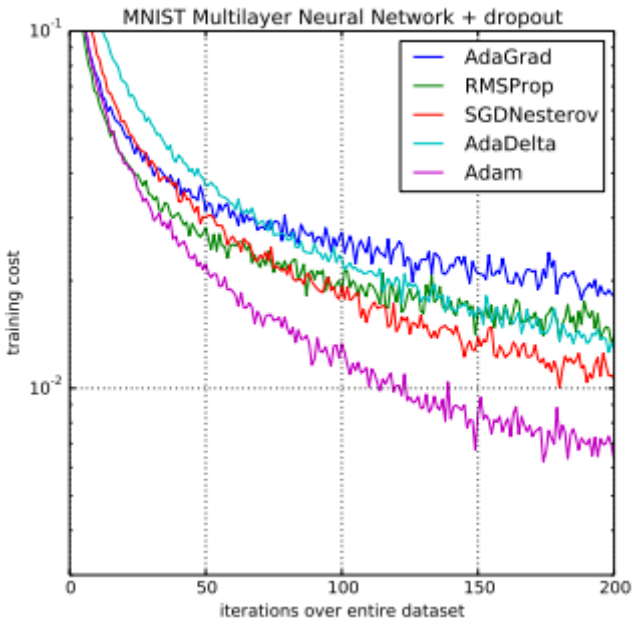


FIGURE 13 ADAM OPTIMIZER TRAINING COST COMPARED WITH THE OTHER CONTEMPORARY OPTIMIZERS

The below represents the overall view of the entire process represented above in the model.

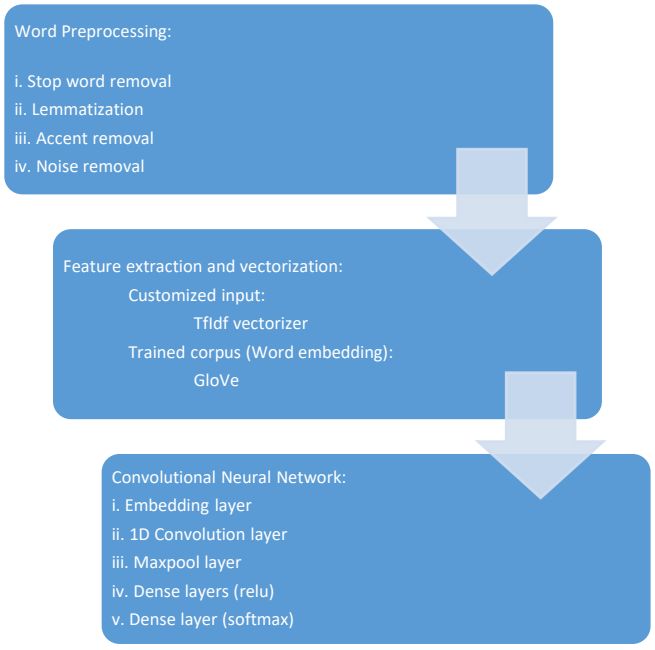


FIGURE 14 OVERALL FLOW

V. CODE REQUIREMENTS

Sci-kit learn

This library is one of the greatest of all times for a data scientists with all the required state-of-the-art tools on various topics involving machine learning, data science and natural language processing.

In our project, we use sci-kit learn for 2 purposes.

First, feature extraction is the one included in the text pre-processing. The following are the packages used and their utilities:

Model_selection => splitting the test and train dataset

Pipeline => Create a pipeline for processes for a single input and a single output.

TfIdfVectorizer => Performs the Term frequency Inverse document frequency to analyze the text count within the corpus

CountVectorizer => Performs a stand count on the number of times a word appears in a particular input word sequence and converts it into a vector based on a pre-defined vocabulary.

NaiveBayes => A classifier based on Baye's theorem

SVM => For Linear Support vector machine classifier

Linear_model => For logistic regression model

Metrics => To calculate the accuracy and the predictions from a trained model

RandomizedSearchCV => A hyper-parameter optimization algorithm for randomized search for the optimum parameters

Keras - Tensorflow

This is a deep learning library by Google Inc. It is merged with yet another deep learning library called the TensorFlow owned and developed by Google too. Similar to sci-kit-learn this open source toolkit is a robust, strong tool that contains state-of-the-art libraries to create any form of deep neural networks from basic perceptrons to complex CNNs and LSTMs.

The following are the functionalities used from Keras in out project:

Tokenizer => Customize a tokenizer based on the user's program including a pre-defined vocabulary

pad_sequences => Pads text sequences to be fed into the neural network that is constrained for the size of the input fed.

Sequential => The base for the deep learning model involving neural networks

Layers => To create the required layers within the CNN

NLTK

Natural Language TooKit is a state of the art toolkit for Natural language processing. It is slower than spaCy but better in some of the functionalities and contains many more functionalities compared to spaCy.

In our project, we use it for:

WordNetLemmatizer => This is a tool for lemmatization of the input text which can be used as a callable tokenizer in the CountVectorizzer function.

spaCy

Eventhough, this has not been used in the final implementation, this is the fastest tool for Natural Language Processing and we had used to test out all the text pre-processing techniques using this toolkit.

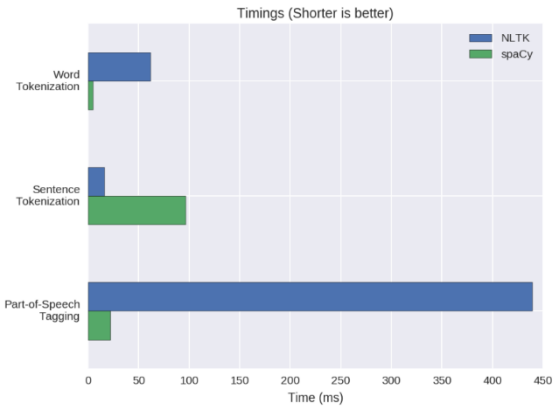


FIGURE 15 PERFORMANCE OF SPACY COMPARED WITH NLTK

Pandas

This toolkit is a widely used data retrieval and raw data processing tools. We have used this for removing the null spaces right after reading the input CSV.

Libraries and tools used

Library	Tool
numpy	-
pandas	-
sci-kit learn	Model_selection.train_test_split
sci-kit learn	Pipeline.pipeline
sci-kit learn	Naïve_bayes.MultinomialNB
sci-kit learn	Svm.LinearSVC
sci-kit learn	linear_model import LogisticRegression
sci-kit learn	sklearn import metrics
sci-kit learn	sklearn.metrics .plot_confusion_matrix
sci-kit learn	feature_extraction.text. CountVectorizer
sci-kit learn	feature_extraction.text .TfidfVectorizer
matplotlib	pyplot
tensorflow.keras	Models.Sequential
tensorflow.keras	layers

keras	preprocessing.text import Tokenizer
keras	preprocessing.sequence import pad_sequences
keras	wrappers.scikit_learn import KerasClassifier
keras	model_selection import RandomizedSearchCV, train_test_split
nltk	word_tokenize
nltk	WordNetLemmatizer

TABLE 1 LIBRARY REQUIREMENTS

VI. DATASET USED

TweepFake dataset: The dataset is obtained from Kaggle. Examples of deep-fake messages can already be found in social media (as our dataset shows), there is still no episode of misuse on them; however, the language models' generative capability deeply worries: it is, therefore, necessary to quickly raise shields against this threat as well. Some deep-fake text detection techniques have already been investigated, but there is still a lack of knowledge on how those state-of-the-art deepfake text detection techniques perform in a "real-social-media-setting", in which the text generation method is unknown and the text content is often short (especially on Twitter).

A dataset of deep-fake social media messages is required to start the research. Unfortunately, most have not created a properly labelled social media dataset containing only human and deep-fake messages (thus excluding cheap-fake texts that employ simple generative techniques as gap-filling and search-and-replace methods) that can already be found on social media timelines.

Focusing on Twitter, Kaggle has collected human and deepfake tweets to support the research on deepfake social media text detection in a "real-setting". The dataset includes the following.

Split	# bot tweets	# human tweets	total
Training set	10463	10463	20926
Validation set	1163	1163	2326
Test set	1292	1292	2584

FIGURE 16 DATA SAMPLES FROM TWEEPFACE

To collect machine-generated tweets, the only known way is to heuristically search for Twitter accounts on the web (especially on Github and Twitter, of course) looking for keywords related either to automatic or AI text generation, deep-fake test/tweets, or to specific technologies such as "GPT-2", "RNN", "LSTM" and so on.

Apart from the above dataset, to verify the model, a separate dataset that was collected and generated in the previous phase has been used. This includes 100 human written text and corresponding machine generation news articles based on the headings provided for the news articles.

I. RESULTS

Model / Accuracy	**Training accuracy**	**Testing Accuracy**
Linear Support Vector Classifier	99.89%	72.53%
Naïve Bayes	97.71%	70.31%
Linear Regression	95.75%	73.83%
Perceptron (Tfldf vectorizer)	99.94%	71.61%
Perceptron (Count vectorizer)	99.94%	69.53
DNN with embedding layer and maxpool	100% (overfit)	71.35%

Convolutional Neural Network (Without GloVe embedded training)	51.62%	50.26%
Convolutional Neural Network (GloVe embedded training)	55.87	54.17

TABLE 2 ACCURACY FOR VARIOUS TRAINED MODELS FOR BOTH THE TRAINING AND THE TESTING SET

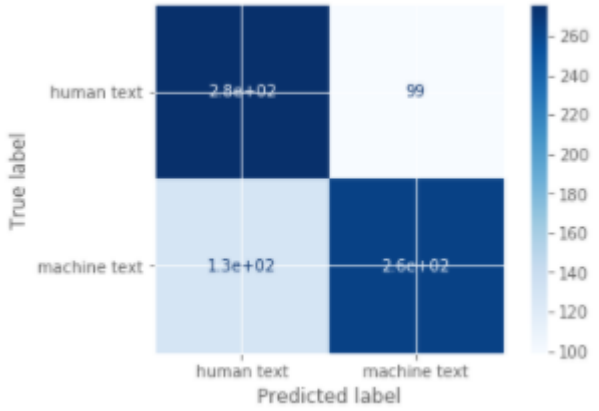


FIGURE 18 HEATMAP AND CONFUSION MATRIX FOR NAÏVE BAYES

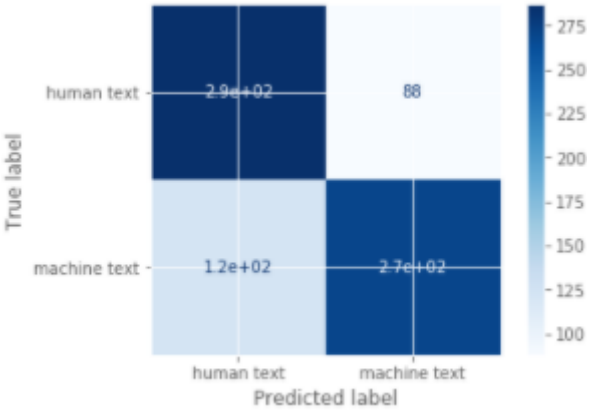


FIGURE 19 HEATMAP AND CONFUSION MATRIX FOR LINEAR SVC

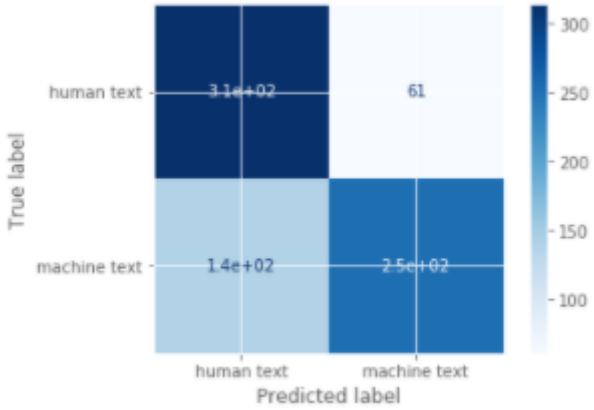


FIGURE 20 HEATMAP AND CONFUSION MATRIX FOR LOGISTIC REGRESSION

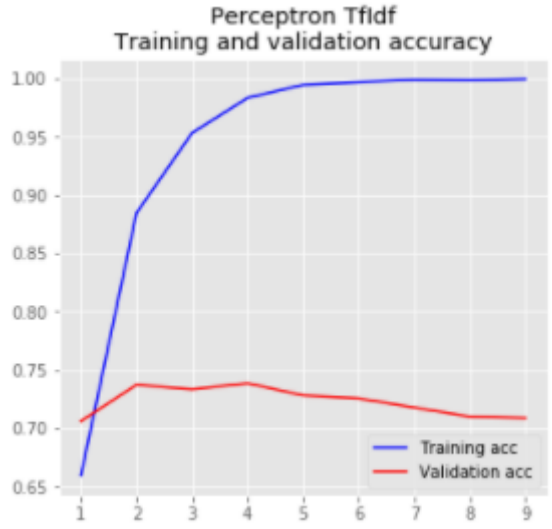


FIGURE 21 PERCEPTRON TFIDF BASED TRAINING VS TESTING ACCURACY PLOT

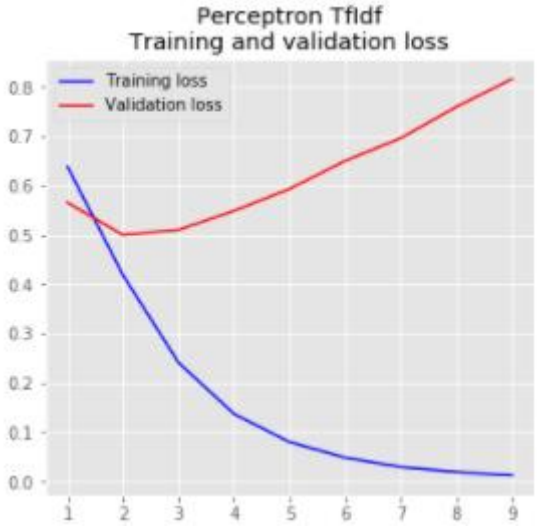


FIGURE 22 PERCEPTRON TFIDF BASED TRAINING VS TESTING LOSS PLOT

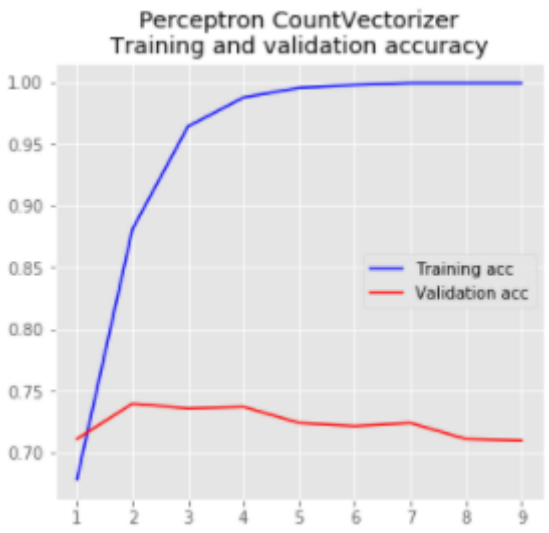


FIGURE 23 PERCEPTRON COUNT VECTORIZER BASED TRAINING VS TESTING ACCURACY PLOT

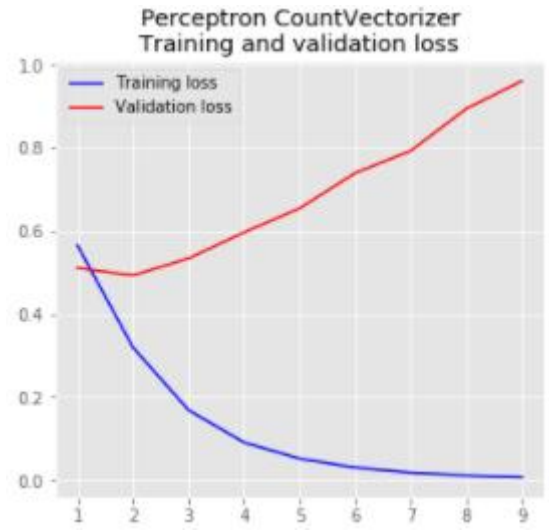


FIGURE 24 PERCEPTRON COUNT VECTORIZER BASED TRAINING VS TESTING LOSS PLOT

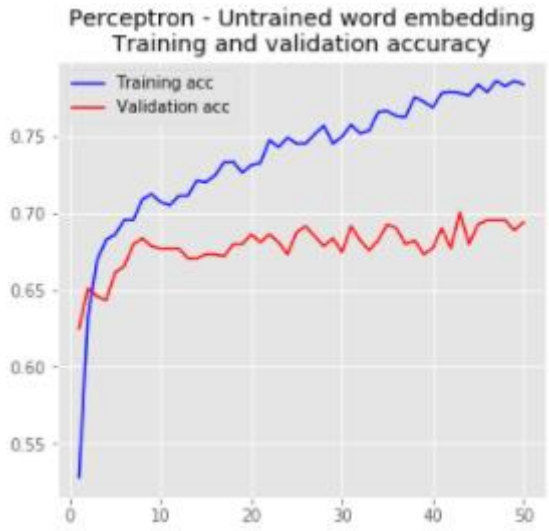


FIGURE 25 PERCEPTRON UNTRAINED WORD EMBEDDING BASED TRAINING VS TESTING ACCURACY PLOT

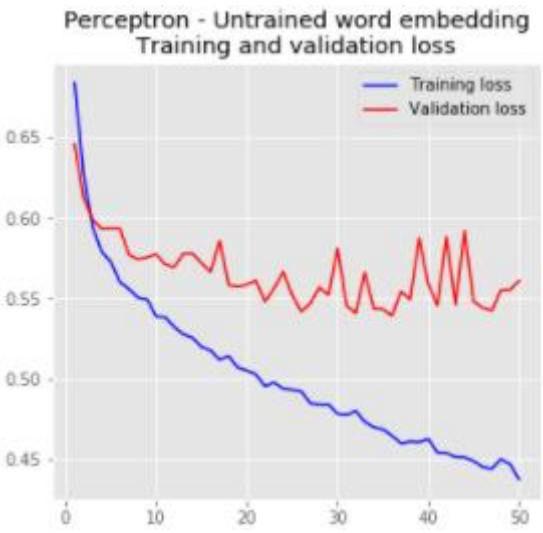


FIGURE 26 PERCEPTRON UNTRAINED WORD EMBEDDING BASED TRAINING VS TESTING LOSS PLOT

FIGURE 27 PERCEPTRON TRAINED GLOVE WORD EMBEDDING BASED TRAINING VS TESTING ACCURACY PLOT

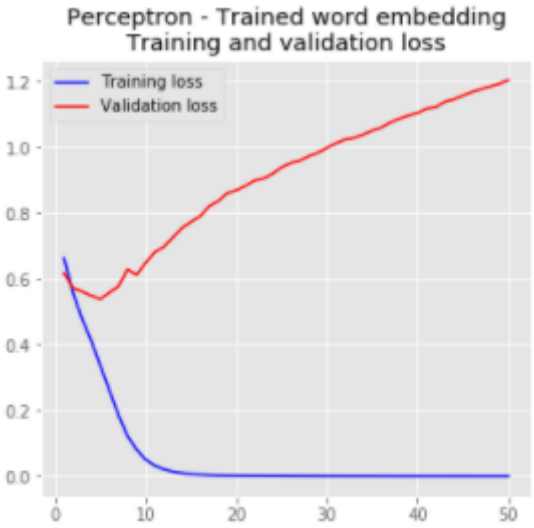


FIGURE 28 PERCEPTRON TRAINED GLOVE WORD EMBEDDING BASED TRAINING VS TESTING LOSS PLOT

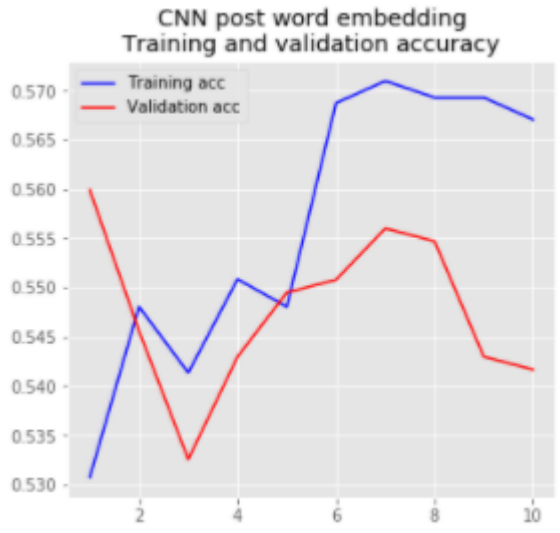


FIGURE 29 CNN POST WORD EMBEDDING BASED TRAINING VS TESTING ACCURACY PLOT

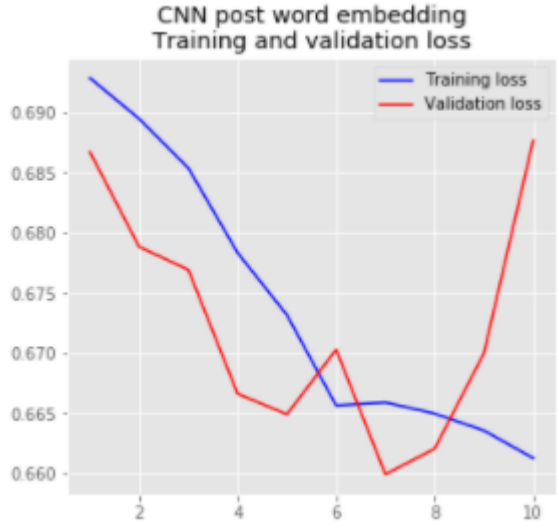


FIGURE 30 CNN POST WORD EMBEDDING BASED TRAINING VS TESTING LOSS PLOT

As per the experiment, the best output so far has been obtained in the linear support vector classifier model. The convolutional neural network seems to perform poorer due to lack of the memory from the previous layers. It misses the inter-dependency of the textual content with the previous words. This provides the continuity of the word sequences and meaning to any sentence structure.

The machine generated word sequences miss this meaning. This is hugely taken care of by the LSTMs (Lon short term memory) models using RNNs and transformer architecture to provide extra-ordinary performance with the text classification.

II. REFERENCES AND LINKS

[0] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean, ***Distributed Representations of Words and Phrases and their Compositionality***

[[1 Tiziano Fagni, Fabrizio Falchi, Margherita Gambini, Antonio Martella, Maurizio Tesconi. ***TweepFake: about Detecting Deepfake Tweets*** *arXiv:2008.00036v1 [cs.CL] 31 Jul 2020*]

[2] Shervin Minaee, Nal Kalchbrenner, Amsterdam Erik Cambria, Narjes Nikzad, Tabriz Jianfeng Gao, Redmond.. ***Deep Learning Based Text Classification: A Comprehensive Review*** *[*arXiv:2004.03705v2 [cs.CL] 17 Nov 2020*]*

[3] Muhammad Zain Ami, Noman Nadeem. ***Convolutional Neural Network: Text Classification Model for Open Domain Question Answering System***.

[4 Jeffrey Pennington, Richard Socher, Christopher D. Manning, ***GloVe: Global Vectors for Word Representation***

ONLINE REFERENCES

1. https://www.kdnuggets.com/2019/04/text-preprocessing-nlp-machine-learning.html#:~ [General]
2. http://kavita-ganesan.com/what-are-stop-words/#.X7hK4GhKiUk [Stop words]
3. https://www.kdnuggets.com/2018/08/practitioners-guide-processing-understanding-text-2.html [Word contractions]
4. https://pypi.org/project/pycontractions/ [Word Contractions]
5. https://en.wikipedia.org/wiki/Convolutional_neural_network [CNN]
6. https://realpython.com/python-keras-text-classification/ [CNN keras overview]
7. https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/ [Display history]
8. https://papers.nips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-

Paper.pdf [Pre-trained word embeddings - Woord2Vec]

9. https://nlp.stanford.edu/projects/glove/ [GloVe]
10. https://nlp.stanford.edu/pubs/glove.pdf
11. https://towardsdatascience.com/the-three-main-branches-of-word-embeddings-7b90fa36dfb9
12. https://towardsdatascience.com/nlp-building-text-cleanup-and-preprocessing-pipeline-eba4095245a0 [Text pre-processing]
13. https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html [sklearn tutorial]

III. APPENDIX

INDIVIDUAL CONTRIBUTIONS

TABLE I. REPORT CONTRIBUTIONS

Sl. No	Package name	Author
1	1. Introduction	Raakesh
3	2a. Literature Review (GloVe)	Raakesh
4	2b. Literature Review (Deep learning based text classification: A comprehensive review)	Raakesh
5	Model Description – Text preprocessing	Raakesh
6	Tokenization	Dunchuan
7	Lemmatization	Dunchuan
8	Stemming	Dunchuan
9	Noise removal	Raakesh
10	Stop words	Raakesh
11	Word contractions	Raakesh
12	Vectorization	Raakesh
13	Count Vectorizer	Raakesh
14	TfIdf Vectorizer	Raakesh
15	Word2Vec	Raakesh
16	GloVe	Raakesh
17	FastText	Raakesh
18	Linear support vector classifier	Raakesh
19	Naïve Bayes Classifier	Raakesh
20	Logistic regression classifier	Raakesh
21	Neural network models	Raakesh
22	Optimizing neural network parameters	Raakesh
23	Grid search	Raakesh
24	Randomized search	Raakesh
25	Final Model	Raakesh
26	Choice of activation function	Raakesh
27	Code Requirements	Raakesh
28	Scki-kit-learn	Raakesh
29	Keras Tensorflow	Raakesh
30	NLTK	Raakesh
31	spaCy	Raakesh
32	Dataset used	Raakesh
33	Results	Raakesh
34	References and Links	Raakesh
35	Report formatting	Raakesh

TABLE II. CODE BASE CONTRIBUTIONS

Sl. No	Code module	Author
1	Dataset collection	Raakesh
2	ML Classifiers using sci-kit-learn pipelines	Raakesh
3	Function to plot accuracy and loss	Raakesh
4	Feature Extraction	Raakesh
5	Perceptron - Using TfiDf vectorizer	Raakesh
6	Perceptron - Using Count vectorizer	Raakesh
8	Word Embeddings - GloVe	Raakesh
9	Training model embedding layer - training parameter - False	Raakesh
10	Training model with embedding layer - training parameter - True	Raakesh
11	# Hyper-parameter optimization - RandomizedCV	Raakesh
12	CNN Optimized	Raakesh

TABLE III. POWERPOINT PRESENTATION

SL. NO	MODULE	AUTHOR
1	Power point slides	Raakesh
2	Presentation	Dunchuan, Raakesh