Empirical Study for bot recognition Using BERT and BiLSTMs

Alejo Pérez Gómez
Department of Statistics, University of Linköping

15 January 2022

Abstract

Pre-trained language models together with the fine-tuning practices have been experiencing a remarkable improvement at NLP tasks during the last years. These models are learnt from well-written text corpora that share characteristics of academical and literary grammar, syntax, and lexicon. Literary works, Wikipedia and News are a few examples of these. When it comes to Tweet Spambot Recognition, we must deal with a big drawback: the difficulty of leveraging the prior knowledge of language models when applied on data that belong to an especially different domain. Social media text, such as Twitter tweets, features many peculiarities (own memes, emojis, hashtags, "at" symbols, misspellings, grammar incorrectness, etc.).

In order to explore how these language models generalize on these web data distributions, we explored some Deep Learning language model architectures to research how they perform at the tweet Spambot Classification task. We used a fine-tuning approach to test the classification performance of BERT, BiLSTMs, and other classic Machine Learning techniques at several tweet data datasets. Our results show several experiments that yielded satisfactory performance. Hence, further research is needed to exploit the text-level classification against the traditional meta-data techniques that spend a lot of time and computational resources.

Introduction

Social spam bots are computer algorithms that automatically produce content and interact with humans and social media [1]. Whereas many of them are intended to conduct a harmless behavior, others are acknowledged of endangering democracy procedures [2], [3], causing panic during emergencies (as in the Covid-19 pandemic [4]) , and affecting the stock market [5], even violating private sensitive data such as phone numbers and addresses [6]. In a similar vein, they generally tend to impoverish the user experience of Twitter, the famous micro-blogging social network that is, in turn, a central matter in this work. This can be done via the aforementioned ways or more roughly by spreading hate speech tweets (as in the case of spamming arabic religious hatred [7]) and inflammatory and violent language towards concrete social groups [8].

In this regard, the scientific literature shows a wide, vast variety of experiments conducted in order to identify these deceptive social spam bots over Twitter. These works leverage Twitter-provided data of different natures to feed Machine Learning (ML) text recognition supervised and unsupervised techniques.

State of the Art

Among the State-of-the-Art of supervised techniques, prominent examples showcase successful results in this binary classification task. These techniques aim at performing the discernment between bot and genuine human tweets. They do it by learning from a dataset made of features-labels tuples. This dataset serves as a learning set in order to further generalize the classification when we apply the learnt algorithm to unforseen data [9].

Classic machine learning algorithms such as Random Forest (RF), Logistic Regression (LR), AdaBoost (AB), Decision Tree (DT), Support Vector Machine (SVM) or Naïve Bayes (NB) have been used widely for this task [10].

Botometer [11] is an example of this approach, making use of more than 1000 hard-engineered features (primarily numerical and account-metadata-related such as number of friends, retweets, mentions, posting rates, etc). It builds on random forests to provide bot-likelihood score, yielding an AUC score of 0.99 on the dataset generated by Lee et al. [12].

Pozzana et al. [13], in turn, used DTs, RF, AB and Extra-Trees (ET), to achieve an AUC score of 0.97 for all models except for the AB experiment (0.84). They applied the algorithms to an own generated dataset

labeled by Botometer and the Cresci et al. dataset [14].

Other authors combined the metadata-engineered features with text-level tweet data. Kudugunta et al. [9] used deep learning with prior preprocessing/tokenization tools attaining a 96% of accuracy. Global Vectors for Word Representation (GloVE) was used to transform the tweets into word embeddings to further feed Long Short Term Memory Models (LSTMs) and contextual LSTMs alongside the user metadata.

While account-level data in combination (or not) with text-level data seems to yield remarkable results, our work will be centered on tweet text-level-only detection. In this line, Garcia-Silva et al. [15] experimented with various Pre-trained Embeddings and Deep Learning (DL) techniques in a Fine-Tuning (FT) workframe.

Building on the approach of Howard & Ruder [16], they used pre-trained language models and embeddings to leverage inherent knowledge and further fine-tune them into the new problem domain. Their approach consisted in fine-tuning pre-trained language models (Open AI GPT, BERT, ULMFit) and combining them with CNN and Bidirectional LSTMs (Bi-LSTMs). They used a generated dataset out of the database of human and bot accounts published by Gilani et al. [17] whose scraping procedures are in their github repository [18].

Our work

Language models such as ULMFiT, Open AI GPT, BERT and ELMo, learn from grammaticaly correct text like Wikipedia, BookCorpus, Wikipedia and BookCorpus, and News respectively. When applying these models to Twitter data, the first issue we encounter is that this special case of text features many intrinsic peculiarities: hashtags (#) to tweet about popular topics, concerns or events, mentions (@) to include another account, shortened URLs (http://t.co) and retweets (RT) to share other people's tweets, etc. Therefore, the text-level recognition has to meet the problem of domain change when applying language models (pre-trained on clean language corpora) in real web Twitter data (sometimes grammatically incorrect, in different languages, full symbols and emojis, etc.)

Our main purpose is to help in reducing this domain change problem in the context of twitter social spambot recognition task. We aimed at doing it by trying three different approaches. Firstly, we used baseline ML techniques: Dummy Classifier (DC), Naïve Bayes (NB), Logistic Regression (LR), Decision Tree (DT) and Support Vector Machine (SVM). Secondly, we used a pre-trained instance of a BERT model [19]. Thirdly, a custom architecture with a base layer of BERT followed by 2 layers of Bi-LSTM [20]. All the code is visible at our project github repository [21].

We used 3 datasets for this task. The 2 first ones are included inside the Cresci et al. database [14], their names are Social Spambots 2 (SSB2) and Social Spambots 3 (SSB3). The former consists of spam tweets of paid apps for mobile devices and the latter corresponds to spam tweets of products on sale at Amazon.com. The third dataset (Scraped) is similar to the one Garcia-Silva et al. used in [15], because we scraped the data by means of the scripts provided in their code repository [18] visible in our repository.

We run the trials of ML techniques on both a pre-processed (custom data cleaning + custom tokenization) and a raw version of the SSB 2 and 3 datasets (default tokenization). For SSB1 and SSB2, we used a modification of the technique proposed by Kamps et. al [22] found successful in their work when pre-processing specific Twitter for sentiment analysis task (**/text-preprocessing-techniques/preprocess.py**). In the case of the Scraped dataset, the experiments were run on a pre-processed version of them resulting of the technique recommended in the work of Garcia-Silva et al.

Our executions with the baselines generally outperformed the BERT and BERT+Bi-LSTM runs, achieving an accuracy of **0.82** and **0.93** with NB at SSB2 (pre-processed) and SSB3 (pre-processed) respectively. On the Scraped dataset, we attained a **0.75** of accuracy with LR and NB. The only remarkable results achieved with BERT+Bi-LSTMS were **0.83** on the SSB3 (pre-processed) dataset, and for BERT a **0.73** on the same dataset. The rest of runs with these architectures showed a strong overfitting on the test sets, presenting a clear bias predicting towards the *human* category for all tweets. Thus, all the rest of accuracy scores were mostly about the 0.5 aleatory accuracy, as datasets were downsampled to be balanced.

The rest of the report is structured as follows: The next point is about the *Theory* behind the ML and DL models used, the *Datasets* point explains how the datasets for the experiments were created and preprocessed and refers to the creation scripts in the Github. The *Method-Experiments* describes the procedures used to conduct the experiments in detail. Then we show *Results*, *Discussion* and a summarization in the form of *Conclusion*.

Theory

RNN is a class of artificial neural sequence model (see Fig. 1), whose connections between units constitute a directed cycle. It gets arbitrary embedding sequences $x = (x_1, ..., x_T)$ as inputs and uses its internal memory network to exhibit dynamic temporal behavior [23]. At each time step t, the hidden state h_t of the RNN is computed based on the previous hidden state h_{t-1} and the input at the current step x_t:

$$h_t = g(U_{x_t} + W h_{t-1}) \tag{1}$$

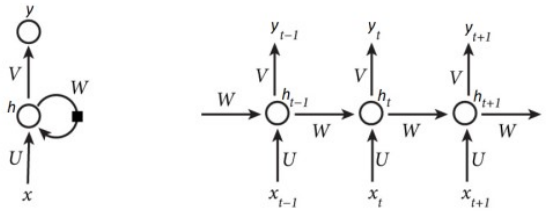

Figure 1: Left: Recurrent neural network; Right: Recurrentneural network unfolded [20]

The **LSTM** is builds on top of the RNN to solve the vanishing gradient problem (gradients drifting to 0 due to lost of memory in short sequences). It uses a memory cell to store information (Fig. 2). σ represents the *softmax* function, W, V, U are weight matrices of the model and i, o, f, c are the gates [24].

$$i_t = \sigma(U_i x_t + W_i h_{t-1} + V_i c_{t-1}) \tag{2}$$

$$f_t = \sigma(U_f x_t + W_f h_{t-1} + V_f c_{t-1}) \tag{3}$$

$$c_t = f_t c_{t-1} + i_t tanh(U_c x_t + W_c h_{t-1}) \tag{4}$$

$$o_t = \sigma(U_o x_t + W_o h_t - 1 + V_o c_t) \tag{5}$$

$$h_t = o_t tanh(ct) \tag{6}$$

Following this chain of complexity built-up, the BiLSTM uses two LSTMs to be fed a sequence of tokens and capture the relations between tokens before and after them. See in Fig. 3, the learning procedure from left to right and viceversa. A hidden layer $\vec{h}$ produced by x_t and $\vec{h}_{t-1}$ generates an activation and a backward hidden $\overleftarrow{h}$ layer does likewise in the opposite direction, see Figure 3.

The **BERT** language model [19], using this bidirectional fashion is made of two stages. The first one masks tokens randomly (by default 15% of sentences) and predicts them, Masked Language Modeling (MLM). The second one uses a corpus to predict the next sentence with the aim of learning relations

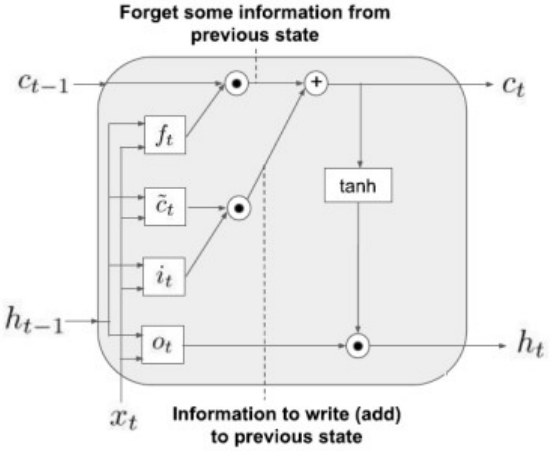

Figure 2: Long-Short Term Memory Cell [20]

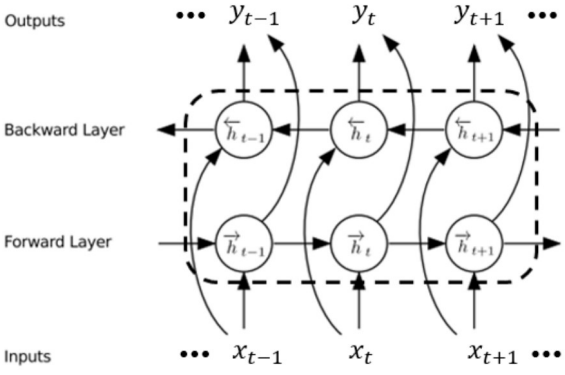

Figure 3: Bidirectional LSTM [20]

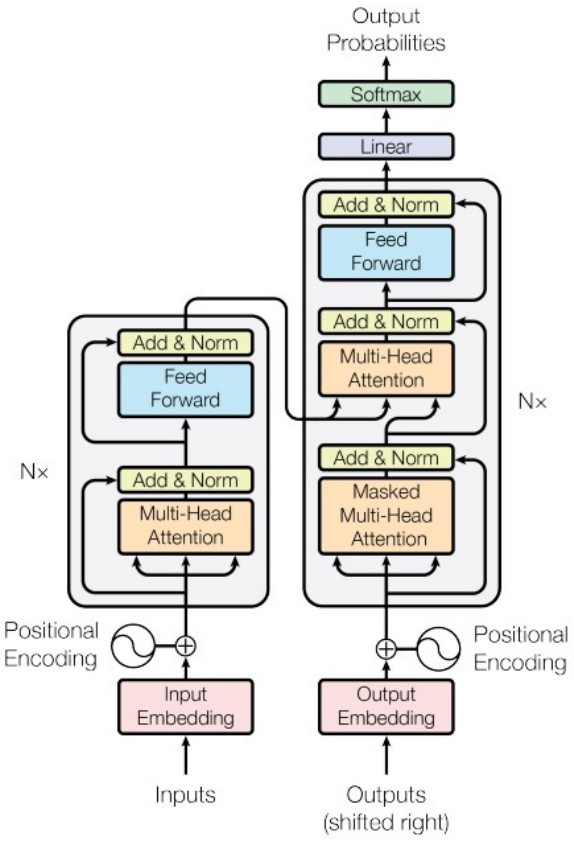

Figure 4: Transformer architecture [25]

between sentences, Next Sentence Prediction (NSP) [10]. See architecture in Fig. 4.

This architecture relies on 2 other remarkable concepts in NLP, the so-called **Word Embeddings** and the **Transformers**. The former stands for the representation of words as real-valued vectors by encoding their meaning, so as words with similar meanings are closer in the vector space than dissimilar ones [10]. **Transformers** use **attention mechanisms** avoiding recurrence and convolutions while relying on parallelization and [25] of computation. Thus, BERT stacks multilayer bi-directional Transformer encoders. The attention mechanisms account to the strength of relation between words intrasentence, see Fig. 5.

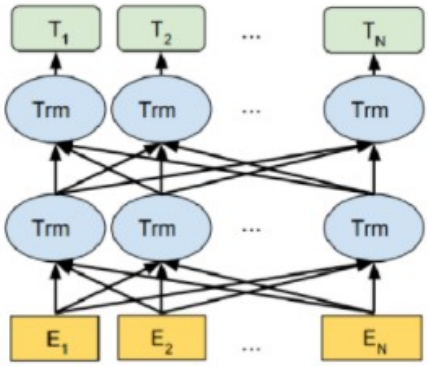

Figure 5: BERT architecture, the base layer creates an embedding representation, each subsequent layer computes another word representation to be fed to the next stage in a multi-head attention fashion. [19]

Our Models

As baselines for our experiments we used several simplistic models. In this approach, be it with preprocessing or not, a **CountVectorizer** transforming step provided by the open source library **Scikit-learn** was applied. This turns the collection of tweets into numerical feature vectors. This specific strategy (tokenization, counting) is called the Bag of Words or "Bag of n-grams" representation [26]. In short, this stage converts the tweets into a vector of counts corresponding to the count of all tokens. Now, we depict the baseline models we used.

Dummy Classifier, that ignores the features and predicts the most common class.

The **Naïve Bayes classifier** [27] was used as the first baseline model, following this function to obtain the class-belonging probability derived by Maximum Likelihood Estimation (MLE). Let $P(x_i \mid y)$ be the

probability of having feature $x_i = X$ given the class $Y = y$ and $P(y)$ the prior probability of belonging to that class.

$$P(y \mid x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} \tag{7}$$

$$P(x_i \mid y)\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y) \tag{8}$$

Logistic Regression [28] was used, simply linearly regressing with the features and applying a softmax function to map the output to a binary classification probability (MLE). Let θ be the parameters of the regression and X the vectorized data, then it is a problem of optimizing the θ in the training set.

$$P(y \mid x_) = \frac{e^{\theta^t X}}{1 + \theta^t X} \tag{9}$$

Decision Trees [29], that learn decision rules inferred from the training set that separate the data according to the features. Classification is performed flowing through the root towards the bottom leaves of the tree.

Support Vector Machine [30], that learns to establish a Maximum Marginal Hyperplane (MMH) that best divides the dataset into classes inferring from the feature space. It has been proved effective in non-linearly separable problems by using the kernel trick (transformation the input space to a high dimensional expansion space out of the features). We applied a grid-search hyperparameter searching with the last 4 aforementioned models.

The next model applied was a pre-trained **BERT** whose diagram is in Fig. 6. The model can be downloaded from the **Tensor Flow API** (*'https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/2'*).

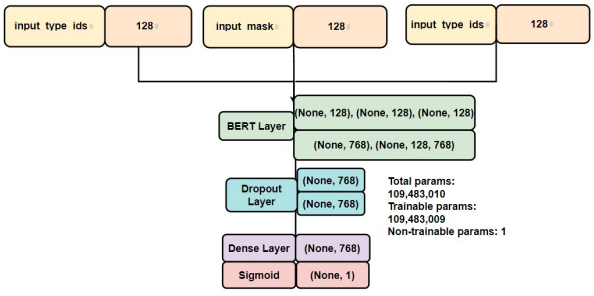

Figure 6: BERT Model Architecture

The last model consisted of a BERT layer whose output in form of word representations (the output embeddings) are forwarded to feed 2 Bi-LSTM layers. We named it **BERT+Bi-LSTM**. The 2 Bi-LSTM layers have integrated a *LSTM Keras object*. See model diagram in Fig. 7.

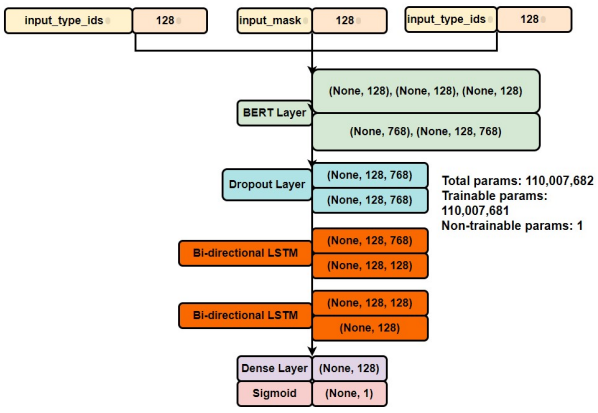

Figure 7: LSTM (Bi-LSTM) Model Hyperparameters

Datasets

For the dataset generation we implemented several sampling functions that output training and validation sets. They are at the Github *Utilities.py* (Scrape dataset) and *Utilities2.py* (SSB2 & SSB3), having analogous functions. More in depth, the function *generate_data()* samples the input datasets with parametrizable proportions of bot tweets, and human tweets. *generate_train_test()* Takes the output of the previous function and generates the training and validation sets (0.7 and 0.3 in proportion respectively). The function *downsample()* balances the datasets towards the less populated category and *downsample_prop()* reduces the dataset sampling the with custom proportion introduced by the user. All datasets where downsampled to be balanced. In the *Preprocessing_test_set.ipynb* we sample the SSB2 and SSB3 datasets to generate a testing set that did not overlap with neither the training sets nor validation. We did not sample a testing set for the Scrape Dataset (Garcia-Silva et al.) due to the validation set results were already extremely overfitted.

SSB2 and SSB3

Cresci et al. gave us permission access their datasets. Therefore, the baseline models were fed with 299755 bot/human tweets (SSB2-training), 38521 bot/human tweets (SSB2-valid), 39884 bot/human tweets (SSB3-training) and 17093 bot/human tweets (SSB3-valid). Mind the notation bot/human means

ther is the same number of tweets for each category. For the grid-search (Baselines) the datasets were downsampled considerably due to the computational limitation.

To feed BERT and BERT+Bi-LSTM the downsampling was stronger do to computational limitations.

For the BERT-BiLSTM We used 2297 bot/human tweets (SSB2-training), 297 bot/human tweets (SSB2-valid), 1088 bot/human (SSB2-testing) 2275, bot/human tweets (SSB3-training), 976 bot/human tweets (SSB3-valid) and 1424 bot/human tweets (SSB3-testing).

For the BERT We used 2253 bot/human tweets (SSB2-training), 310 bot/human tweets (SSB2-valid), 1165 bot/human tweets (SSB2-testing), 2275 bot/human tweets (SSB3-training) and 976 bot/human tweets (SSB3-valid) and 1456 bot/human tweets (SSB3-testing) .

The preprocessing of this datasets is visible at */text-preprocessing-techniques/preprocess.py & techniques.py*, and draws on the work of [22] with *NLTK* package. The pipeline of this pre-processing is the following: Tokenization -> Antonym Replacement if preceeds "not" -> Translation if not English -> All capitals special Token addition -> Lower Case -> Elongated Replacement - Lemmatization -> Stemming (similar to lemmatization but more aggressive).

After the preprocessing, some tweets went to NaN possibly due to the translation or stemming, so we lost some tweets.

Scrape Dataset

We scraped this dataset using the Garcia-Silva et al.s' ruby scripts (**/An-Empirical-. . . /dataset generation/program-bots.rb & program-humans.rb**). We also pre-processed them as they recommended (**/An-Empirical-study. . . preprocess-twitter.rb**). Their preprocessing pipeline consists of: Tweet specific Tokenization taking into account emojis, memes and twitter interaction syntax (@, #, RT) -> Force splitting words appended with slashes -> Tokenization of repeated punctuation -> Special Elongation Tokenization -> All Caps Tokenization.

Method - Experiments

Baselines

For the baselines, we simply trained on the respective training sets and further evaluated the performance on the validation test. No further testing set was included here since they are baseline models. For the NB, LG, DT and SVM we run a grid-search to perform hyper-parameter selection. This featured 4 or 5 folds depending on the running time (computational time was limiting). See the hyperparameter grid at Table 1. As for the CountVectorizer we varied the n-grams over all the models and in the NB the the *binary* parameter (set count 0 to 1). We also tried typical intrinsic parameter grids for each model.

	vectorize__binary	vectorize__ngram_range	NB__alpha	
NB	(True, False)	[(1,1), (1,2)]	(1,0.1)	
	vectorize__ngram_range	LG__C		
LG	[(1,1), (1,2), (2,2)]	[0.1, 0.5, 1]		
	vectorize__ngram_range	DT__max_depth		
DT	[(1,1), (1,2), (2,2)],	[30, 60, 120]		
	vectorize__ngram_range	SVC__C	SVC__gamma	SVC__kernel
SVM	[(1,1), (1,2), (2,2)]	[10, 1000]	[0.001, 0.0001]	rbf

Figure 8: Table 1. Baseline's Grid Hyperparameters

BERT

We followed the tutorial provided by Wandb [31]. In order to feed the datasets to the input layer of BERT, it was necessary to transform them into Tensor Flow *Prefetch Dataset Objects*, therefore passing them to GPU. In order to generate these, we used the BERT tokenizer that draws on the BERT specific corpus and we generated 3 features for each tweet. For this, we coded functions *create_feature_map* and *create_feature*, visible at all experiments with BERT in the Github. The three features for each tweet are:

- Token ID's: position in the sentences
- Padding generated Masks (sparse 1-0 vectors)
- Label ID's

We also shrank the sequences to 128 characters, as that is maximal for the model downloaded. We use the hyperparameter configuration specified in Table 2.

Hyperparameters	Value
Label List	[0,1]
Max Sequence Length	128
Train Batch Size	32
Learning Rate	$2 \cdot 10^{-5}$
Epochs	3
Optimizer	adam
Dropout	0.5

Figure 9: Table 2. BERT Model Hyperparameters

We trained the model with the different training datasets and tested them on the respective created

test sets. The test set served the purpose of extracting performance metrics as well as annotating the capability of generalization. The number of epochs was selected empirically beginning with 2 or 3 epochs and repeating the trials until we met an early stopping criterion (the validation set shows no overfitting signs). The output BERT object fed to the dense layer function was the *pooled_output*. This is a pooling over all the embedding representations for each token (1,128) and gets the ideal dimensions to be passed to a dense layer.

BERT and Bi-LSTMs combination

We applied the same feature transformation as in the last experiment. For the LSTMs integrated inside the Bi-LSTM Layers we used the hyperparameter configuration of the Table 3. We used the Wandb API functions in order to save and upload model executions in the cloud Wandb private instance. The BERT's output object selected forwarded to the Bi-LSTM was the *sequence_output* (128, 278), and fits the next input stage's dimension.

Hyperparameters	Value
Activation	Tanh
Recurrent Activation	Sigmoid
Bias	True
Kernel Initializer	Glorot Uniform
Recurrent Initializer	Orthogonal

Figure 10: Table 3. LSTM (Bi-LSTM) Model Hyperparameters

The process of training was identical to the previous model tested, training with the training set and further validation in the unseen data (testing set).

Results

The results are shown in the following tables, precision, recall and precision at recall are taken out of the positive class (bot). Precision at recall is the result of the computation of the best precision where recall ranges from an vector of 200 thresholds between [0,1]. We can observe a clear outperformance of the grid-searched baseline models in all datasets with respect to BERT and BERT+Bi-LSTM. It is important to remark that the baselines' results are based on the evaluation of a validation set (however unseen by the model during the training step) that does not coincide with the Table 4 testing set (BERT & BERT+BiLSTM results).

	precision	recall	precision_at_recall	accuracy
BERT-SSB2	0.584	0.449	0.608	0.568
BERT+SSB3	0.494	0.869	0.499	0.491
BERT-Preprocessed-SSB2	0.0	0.0	0.505	0.494
BERT-Preprocessed-SSB3	0.978	0.478	0.976	**0.733**
BERT+BiLSTM_SSB2	0	0	0.554	0.5032
BERT+BiLSTM_SSB3	0.5	0.894	0.506	0.5017
BERT-BiLSTM-Preprocessed-SSB2	0	0	0.701	0.495
BERT-BiLSTM-Preprocessed-SSB3	0.678	0.985	1.	**0.834**
BERT+BiLSTM_Scraped (valid)	0	0	896	496

Figure 11: Table 4. BERT & BERT+ Bi-LSTM Results

In overall, all the runs of these 2 architectures show strong signs of overfitting since they perform poorly with unseen data. That is not the case of BERT+BiLSTM (0.834) on the pre-processed SSB3 and 0.733 of BERT on the same dataset.

	precision	recall	precision_at_recall	accuracy
NB	0.65	0.77	0.70	0.68
LG	0.69	0.62	0.65	0.67
DT	0.67	0.79	0.72	0.70
SVM	0.64	0.75	0.69	0.66

Figure 12: Table 5. Baselines Results - SSB2 Raw

	precision	recall	precision_at_recall	accuracy
NB	0.70	0.89	0.78	0.76
LG	0.84	0.83	0.84	**0.84**
DT	0.79	0.76	0.77	0.77
SVM	0.84	0.81	0.82	0.83

Figure 13: Table 6. Baselines Results - SSB3 Raw

In bold letters are marked the best results reaching even to 0.93 of accuracy with such a simple technique as NB at the dataset SSB3 pre-processed.

However, the Scrape dataset does not experiment any improvement compared to the best result at the baseline (0.75 with NB and LG).

Discussion

We remark the performance of the BERT+BiLSTM (0.834) on the pre-processed SSB3 set as well as the 0.733 accuracy of BERT on the same dataset. They are still 20 and 30 percentage points than below Pozzana et al. [13], but we do not have the same computational resources to sample such a large dataset of SBB2 and SBB3. In overall, all the runs of these 2 architectures show strong signs of overfitting since they perform poorly with unseen data.

The baselines tended to perform better although their validation sets are not comparable enough to the other models' testing set. We cannot claim it is a resounding outperformance.

	precision	recall	precision_at_recall	accuracy
NB	0.80	0.84	0.82	**0.82**
LG	0.73	0.62	0.67	0.70
DT	0.72	0.49	0.58	0.65
SVM	0.76	0.56	0.64	0.69

Figure 14: Table 7. Baselines Results - SSB2 preprocessed

	precision	recall	precision_at_recall	accuracy
NB	0.90	0.98	0.94	**0.93**
LG	0.90	0.83	0.86	0.87
DT	0.88	0.79	0.83	0.84
SVM	0.89	0.79	0.84	0.85

Figure 15: Table 8. Baselines Results - SSB3 preprocessed

Other works like the Botometer [11] reached to an almost perfect classification. Although our methods are not completely comparable to theirs since they are professionals with cutting-edge equipment, we want to highlight that the use of account metadata is sometimes crucial to get an almost perfect accuracy. This is observable in many works such as [12], [9].

Garcia-Silva et al. showcase a much better performance in their work (they used only text-level modeling) [15]. We used tweets similar to theirs as we scraped the same accounts and preprocessed equally. The difference in performance compared to them (our 0.5 and 0.75 vs their solid 0.8229) can be explained because they used a better practice of fine-tuning freezing the first layers in the training.

Conclusion

In this work we chose a classification problem to research the generalisation of pre-trained language models comparing it with simpler models that rely on simple token count representations. The complication for the pre-trained models is their knowledge is taken out of clean, mostly academical text, whereas web data, and concretely Twitter data reports generally a different domain distribution in terms of grammar, lexicon and syntax.

	precision	recall	precision_at_recall	accuracy
NB	0.78	0.71	0.74	0.75
LG	0.77	0.71	0.74	0.75
DT	0.70	0.68	0.69	0.69
SVM	0.72	0.64	0.68	0.70

Figure 16: Table 9. Baselines Results - Scrape dataset

The objective was to leverage the prior language knowledge of language models and use transfer-learning by fine-tuning in our datasets. More specifically, we wanted to instruct the architectures in a new task without loss of performance in matters of their already acquired knowledge.

Our research suggest that more basic models show a general better performance when compared to the approach we used in other with complex models.

Not all the results are a failure but it is evident that something failed in our approach and we have some insights of what could be the reason these models overfitted drastically in many most of the occasions.

- **Unsufficient pre-processing**: The pre-processing + tokenization pipeline has been proven to be prone to fail. The NLTK translator function did not work on a lot of occasions and that might be due to the uncleanness of the data. The stemming is perhaps too much of a excessive pre-processing tool. In general, we believe that getting rid of more foreign language tweets would have improved the results.
- **Training with frozen layers should be tried**: Our approach did not leverage transfer learning in most of the occasions. It looks that BERT forgot most of its prior acquired knowledge and we contemplate that freezing the first layers of the architectures would prevented this loss, helping the fine-tuning to be performed more swiftly.
- **Change in activation functions**: As in most occasions the models have a drastic bias towards the human category, we suspect that gradients have saturated towards the negative class. A change of the activation functions inside the Bi-LSTM hidden layers and/or in BERT could be done to explore whether this is a plausible solution.
- **Augment of the computational resources**: The GPU should be upgraded in order to cope with larger amounts of data so as we increase the capability of generalization.
- **More solid dataset splits:** As some models were provided with very limited hardware performance, it has not been possible run our trials in identical data splits. This is because some models tolerated the size of the training data better and it was not necessary to downsample the dataset, while others needed a downsample. That was more intense in the case of BERT and BERT+BiLSTM.

To conclude, we want to state that it is crucial for our society to keep the research on this matter for the reasons explained in the introduction. The text level bot recognition approach should be further explored due to two main reasons:

- It relies on NLP and we can avoid to scrape account's and sensitive data, on top of avoiding costly creation of hard-engineer features.
- The amount of time spent in engineering features and preprocessing the data could be diminished significantly with a correct use of transfer learning.

References

[1] E. Ferrara, O. Varol, C. Davis, F. Menczer, and A. Flammini, "The Rise of Social Bots," *Communications of the ACM*, vol. 59, no. 7, pp. 96–104, Jun. 2016, doi: 10.1145/2818717. [Online]. Available: http://arxiv.org/abs/1407.5225. [Accessed: 15-Jan-2022]

[2] E. Ferrara, "Bots, elections, and social media: A brief overview," *arXiv:1910.01720 [cs]*, Oct. 2019 [Online]. Available: http://arxiv.org/abs/1910.01720. [Accessed: 15-Jan-2022]

[3] A. Shevtsov, C. Tzagkarakis, D. Antonakaki, and S. Ioannidis, "Identification of Twitter Bots Based on an Explainable Machine Learning Framework: The US 2020 Elections Case Study," *arXiv:2112.04913 [cs]*, Dec. 2021 [Online]. Available: http://arxiv.org/abs/2112.04913. [Accessed: 15-Jan-2022]

[4] E. Ferrara, "What Types of COVID-19 Conspiracies are Populated by Twitter Bots?" *First Monday*, May 2020, doi: 10.5210/fm.v25i6.10633. [Online]. Available: http://arxiv.org/abs/2004.09531. [Accessed: 15-Jan-2022]

[5] J. Bollen, H. Mao, and X. Zeng, "Twitter mood predicts the stock market," *Journal of Computational Science*, vol. 2, no. 1, pp. 1–8, Mar. 2011, doi: 10.1016/j.jocs.2010.12.007. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S187775031100007X. [Accessed: 15-Jan-2022]

[6] Y. Boshmaf, I. Muslukhov, K. Beznosov, and M. Ripeanu, "Design and analysis of a social botnet," *Computer Networks*, vol. 57, no. 2, pp. 556–578, Feb. 2013, doi: 10.1016/j.comnet.2012.06.006. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1389128612002150. [Accessed: 15-Jan-2022]

[7] N. Albadi, M. Kurdi, and S. Mishra, "Hateful People or Hateful Bots? Detection and Characterization of Bots Spreading Religious Hatred in Arabic Social Media," *Proceedings of the ACM on Human-Computer Interaction*, vol. 3, no. CSCW, pp. 1–25, Nov. 2019, doi: 10.1145/3359163. [Online]. Available: http://arxiv.org/abs/1908.00153. [Accessed: 15-Jan-2022]

[8] M. Stella, E. Ferrara, and M. De Domenico, "Bots increase exposure to negative and inflammatory content in online social systems," *Proceedings of the National Academy of Sciences*, vol. 115, no. 49, pp. 12435–12440, Dec. 2018, doi: 10.1073/pnas.1803470115. [Online]. Available: http://www.pnas.org/lookup/doi/10.1073/pnas.1803470115. [Accessed: 15-Jan-2022]

[9] S. Kudugunta and E. Ferrara, "Deep Neural Networks for Bot Detection," *Information Sciences*, vol. 467, pp. 312–322, Oct. 2018, doi: 10.1016/j.ins.2018.08.019. [Online]. Available: http://arxiv.org/abs/1802.04289. [Accessed: 15-Jan-2022]

[10] A. Z. Kenyeres, "Social Media Bot Detection." Luleå University of Technology, Department of Computer Science, Electrical; Space Engineering, 2021.

[11] K.-C. Yang, E. Ferrara, and F. Menczer, "Botometer 101: Social bot practicum for computational social scientists," *arXiv:2201.01608 [cs]*, Jan. 2022 [Online]. Available: http://arxiv.org/abs/2201.01608. [Accessed: 15-Jan-2022]

[12] K. Lee, B. Eoff, and J. Caverlee, "Seven Months with the Devils: A Long-Term Study of Content Polluters on Twitter," 2011.

[13] I. Pozzana and E. Ferrara, "Measuring Bot and Human Behavioral Dynamics," *Frontiers in Physics*, vol. 8, p. 125, Apr. 2020, doi: 10.3389/fphy.2020.00125. [Online]. Available: https://www.frontiersin.org/article/10.3389/fphy.2020.00125/full. [Accessed: 15-Jan-2022]

[14] S. Cresci, R. Di Pietro, M. Petrocchi, A. Spognardi, and M. Tesconi, "The Paradigm-Shift of Social Spambots: Evidence, Theories, and Tools for the Arms Race," in *Proceedings of the 26th International Conference on World Wide Web Companion - WWW '17 Companion*, 2017, pp. 963–972, doi: 10.1145/3041021.3055135 [Online]. Available: http://dl.acm.org/citation.cfm?doid=3041021.3055135. [Accessed: 15-Jan-2022]

[15] A. Garcia-Silva, C. Berrio, and J. M. Gómez-Pérez, "An Empirical Study on Pre-trained Embeddings and Language Models for Bot Detection," in *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, 2019, pp. 148–155, doi: 10.18653/v1/W19-4317 [Online]. Available: https://www.aclweb.org/anthology/W19-4317. [Accessed: 14-Jan-2022]

[16] J. Howard and S. Ruder, "Universal Language Model Fine-tuning for Text Classification," *arXiv:1801.06146 [cs, stat]*, May 2018 [Online]. Available: http://arxiv.org/abs/1801.06146. [Accessed: 15-Jan-2022]

[17] Z. Gilani, E. Kochmar, and J. Crowcroft, "Classification of Twitter Accounts into Automated Agents and Human Users," in *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*, 2017, pp. 489–496, doi: 10.1145/3110025.3110091 [Online]. Available: https://dl.acm.org/doi/10.1145/3110025.3110091. [Accessed: 15-Jan-2022]

[18] A. Garcia-Silva, C. Berrio, and J. M. Gómez-Pérez, "An Empirical Study on Pre-trained Embeddings and Language Models for Bot Detection," in *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, 2019, pp. 148–155, doi: 10.18653/v1/W19-4317 [Online]. Available: https://github.com/expertailab/An-Empirical-study-on-Pre-trained-Embeddings-and-Language-Models-for-Bot-Detection

[19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv:1810.04805 [cs]*, May 2019 [Online]. Available: http://arxiv.org/abs/1810.04805. [Accessed: 15-Jan-2022]

[20] F. Wei and U. T. Nguyen, "Twitter Bot Detection Using Bidirectional Long Short-term Memory Neural Networks and Word Embeddings," *arXiv:2002.01336 [cs]*, Feb. 2020 [Online]. Available: http://arxiv.org/abs/2002.01336. [Accessed: 15-Jan-2022]

[21] P. G. Alejo, "BERT-Bi-LSTM-Bot-Recognition." Jan-2020 [Online]. Available: https://github.com/alejo-perez-upc-77/BERT-Bi-LSTM-Bot-Recognition.git

[22] D. Effrosynidis, S. Symeonidis, and A. Arampatzis, "A Comparison of Pre-processing Techniques for Twitter Sentiment Analysis," in *Research and Advanced Technology for Digital Libraries*, vol. 10450, J. Kamps, G. Tsakonas, Y. Manolopoulos, L. Iliadis, and I. Karydis, Eds. Cham: Springer International Publishing, 2017, pp. 394–406 [Online]. Available: http://link.springer.com/10.1007/978-3-319-67008-9_31. [Accessed: 16-Jan-2022]

[23] J. L. Elman, "Finding Structure in Time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, Mar. 1990, doi: 10.1207/s15516709cog1402_1. [Online]. Available: http://doi.wiley.com/10.1207/s15516709cog1402_1. [Accessed: 16-Jan-2022]

[24] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735. [Online]. Available: https://direct.mit.edu/neco/article/9/8/1735-1780/6109. [Accessed: 16-Jan-2022]

[25] A. Vaswani *et al.*, "Attention Is All You Need," *arXiv:1706.03762 [cs]*, Dec. 2017 [Online]. Available: http://arxiv.org/abs/1706.03762. [Accessed: 16-Jan-2022]

[26] "Scikit-Learn documentation." [Online]. Available: https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction

[27] S. Raschka, "Naive Bayes and Text Classification I - Introduction and Theory," *arXiv:1410.5329 [cs]*, Feb. 2017 [Online]. Available: http://arxiv.org/abs/1410.5329. [Accessed: 16-Jan-2022]

[28] M. Maalouf, "Logistic regression in data analysis: An overview," *International Journal of Data Analysis Techniques and Strategies*, vol. 3, no. 3, p. 281, 2011, doi: 10.1504/IJDATS.2011.041335. [Online]. Available: http://www.inderscience.com/link.php?id=41335. [Accessed: 16-Jan-2022]

[29] N. Narayan, "Twitter Bot Detection using Machine Learning Algorithms," in *2021 Fourth International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, 2021, pp. 1–4, doi: 10.1109/ICECCT52121.2021.9616841.

[30] D. Srivastava and L. Bhambhu, "Data classification using support vector machine," *Journal of Theoretical and Applied Information Technology*, vol. 12, pp. 1–7, Feb. 2010.

[31] "Wandb. How to Fine-Tune BERT for Text Classification." [Online]. Available: https://wandb.ai/akshayuppal12/Finetune-BERT-Text-Classification/reports/How-to-Fine-Tune-BERT-for-Text-Classification--Vmlldzo4OTk4MzY#saving-the-models-and-model-versioning