Replicating Scalable Social Bot Detection Through Data Selection

Cathy Guang, Katrina Li, Emanuel Linton, and Matthew Smith-Erb
Department of Computer Science, Carleton College
CS 400
Prof. Anna Rafferty
November 21, 2022

Abstract

Social bots on Twitter are accounts controlled by software that can manipulate opinions in important events such as elections. In order to detect these bots, we want to achieve both scalability and generalizability, to enable faster real-time detection algorithms and to make the classification more robust to new bots that are different from previous ones. We replicate the paper *Scalable and Generalizable Social Bot Detection through Data Selection* (Yang, Varol, Hui, Menczer 2020) and test the robustness of its results. The paper proposed an approach for classification through training random forest models on just user profile information and only on a subset of the available training data. Our results align with Yang, et al. (2020) in that training on subsets of datasets with user metadata improves classifier performance. We differ from their models as we achieved better model performance than their top models using a fewer number of datasets for training. These differences might be due to datasets differences, feature calculation discrepancies, API call rate limit, and our relatively newer sampling time for account features of certain datasets. Relationships across all training and testing datasets are investigated to explain top models' selections of training subsets, and feature importance in the training datasets are analyzed.

Introduction

When you read a tweet, do you know that it might be sent by automated software rather than a human? A portion of social media accounts are in fact bots, controlled by software and created with all kinds of intentions. They manipulate opinions in important events, including recent elections, and also spread extreme ideology or unreliable information (Deb et al. 2019). Detecting social bots has two major challenges: access to streaming data with limited computing resources, which makes scaling up hard for methods that inspect rich information on account's action; and detection of new bots that are different from old ones which are designed to evade detection systems. In the paper *Scalable and Generalizable Social Bot Detection through Data Selection* (Yang, Varol, Hui, Menczer 2020), a new bot detection framework is proposed that can scale up to process the Twitter data stream in real time and generalize well to new accounts outside the training data. The paper addresses the two challenges by training models on just user profile information, which requires less computational power to be accessed in bulk, and also by analyzing the feature space of different datasets and thus compiling subsets of them to build a more robust classifier. The authors evaluate their models using cross-validation accuracy on training data, generalization to unseen data, and consistency with the more feature-rich classifier Botometer (Varol et al. 2017) on unlabeled data.

We replicate the original paper by following the same training and evaluation process. We achieve similar results to the paper, but better performing top models than theirs using the same ranking method. Our training datasets differ slightly from the datasets used by Yang, et al. (2020), which might cause the differences in performance. Other factors that lead to different results can be our feature calculation methods, the limited API call rate for Botometer and thus differences in the Spearman's r evaluation metric, and our relatively newer sampling time for account features of certain datasets. Our results align with theirs in the sense that we both have improved performance by selecting subsets of datasets for training, rather than using all available data.

Bot detection model

Model Goals

We want our bot classifier to achieve the same generalizability and scalability of the models presented by Yang et al. (2020). We train many instances of a model across different subsets of datasets to identify the model which generalized best and performed well against our metrics. These goals aim to improve on previous attempts at bot detectors. For instance, Botometer (Varol et al. 2017) was a larger model that trained on seven specific datasets, each containing over 1,000 features.

Data selection

We had seven training datasets at our disposal from Yang et al. (2020). However, each of these datasets were collected in different ways, and thus represented various types of bot and human accounts. Since each dataset has various levels of generalizability and distribution of features per label, the model trained on all available data won't necessarily perform the best.

We obtain the best bot classifier by training many models and then ranking them based on evaluation metrics. We train 119 distinct models, each identified by the unique combination of datasets that they were trained on. The number 119 comes from the size of the power set of the seven data training sets ($2^7 = 128$) minus all subsets that don't include both humans and bots (nine total subsets). For instance, we never train a model on just the *pronbots* and *political-bots* datasets because there are no human accounts in the data and so our model would never learn to classify unseen data as humans. The hope is that one (or several) of the 119 models is trained on the subset of data which best generalizes to other types of Twitter account data.

Random forest model

The model we use for classification is a random forest. This choice of model prevents overfitting to the training data, which results in better generalizations for predicting unseen data. The random forest classifier also helped us accomplish our goal of scalability, because it performs well on our training data that only had twenty features per observations. The features we train on are embedded in each account's Tweets. As such, capturing a single Tweet from an account is enough to make a new classification. On the other hand, Botometer required 200 Tweets from a single account to make a classification. To be consistent with Yang et al. (2020), we implement our models with the Python library scikit-learn (Pedregosa et al. 2011).

The overall structure of a random forest model is shown in Figure 1. A random forest model consists of a set number of decision trees, in our case 100. A decision tree is built to model a set of sequential and hierarchical decisions that ultimately lead to a final classification. When a random forest is passed in a test instance, each decision tree will output a result of whether the instance is a human or a bot based on the test instance's features, and the random forest model calculates the ultimate probability of the outcome by averaging the output of all of its trees. We use the random forest model because we want to test robustness of the models from the paper, it prevents overfitting of training data by averaging the prediction results from different trees, and the model will give a higher predictive accuracy than a single decision tree.

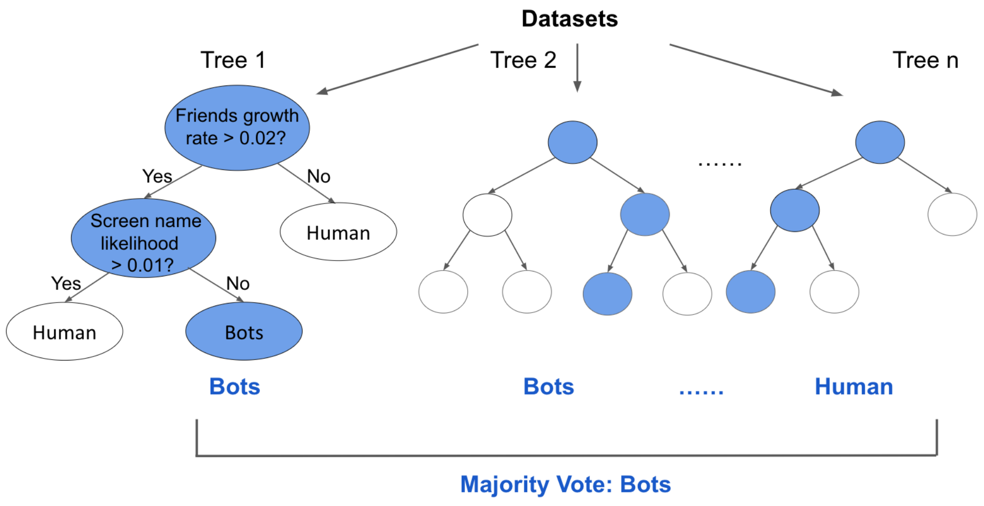


Figure 1. Illustration of a random forest model.

Randomness is introduced to the model in two different methods. First, to train each decision tree, the model selects a random bootstrapping of the data. Next, we randomly sample a square root number of features for training each decision tree. Combined, they introduce variation amongst the decision trees, which helps prevent overfitting to the base set of training data. However, this randomness also causes variance in results between identical training instances. Thus, for each of the 119 models, we train them five times, and average results.

Evaluating classifier effectiveness

We test our models on unseen datasets to evaluate their abilities to make generalized predictions. To evaluate the models on a more diverse sample of accounts, we also test the consistency of their classifications against a more feature-rich classifier Botometer (Varol et al. 2017). We additionally use cross-validation on training data for a full evaluation of model performance.

For each trained model, we have it make predictions on accounts from the four unseen test datasets. From these predictions, we calculate the AUC of each model for the test sets. The AUC score is a value between 0 and 1 which represents the probability that a random positive example is ranked higher than a random negative example. A model whose predictions are 100% correct has an AUC score of 1.0, and a model that is randomly making predictions will have an AUC score of 0.5. The higher the AUC score, the better the model performs on the testing datasets.

For cross-validation, we apply the StratifiedShuffleSplit function from scikit-learn, which helps split the dataset such that each split has approximately the same number of humans and bots. The data is also shuffled each time before the split is done to ensure randomness in each split. For each split, we train an instance of the model on one part of the data, and test the model on the other part of the data. We repeat this process for five different splits, averaging their AUC scores. Testing on a portion of the training datasets further ensures accuracy of the model's predictions.

We use Spearman's Rank correlation coefficient to compare classifications made by our models against Botometer. We created a dataset of 401 random users collected with the Twitter streaming API in September of 2022. We use Botometer as a reference for getting bot scores for the random accounts. The Spearman's R value between -1 and 1 represents the correlation between the relative bot scores assigned by Botometer and our models.

To fully measure the robustness of our models, the same model is made and tested four more times, with all five instances of models' AUC/Spearman's R scores averaged. We combine the scores of all the testing metrics by first finding the relative rank of how well each model performed on each of the six metrics (ranking from 1 to 119). Then, for each model, we find the product of its six relative ranks. The top/best performing model is the one with the lowest product of ranks.

Datasets

Data source

We obtained twelve datasets used in the research by Yang et al. (2020). Seven datasets were used for model training, and five were for model testing. We omitted a training dataset named *caverlee* (Lee et al. 2011) used by Yang et al. (2020) because it was over a decade old, had significantly more observations than other datasets, and Yang et al. (2020) still showed it wasn't present in any of their top models. Yang et al. (2020) chose seven of the datasets for training because they were datasets that an alternate model named Botometer (Varol et al. 2017) was first trained on.

Table 1. All the datasets we used from (Yang et al. 2020) for training and testing.

Datasets	**Usage**	**# Bots**	**# Humans**
varol	training	665	1441
cresci-17	training	6543	3474
pronbots	training	17882	0
celebrity	training	0	5918
vendor-purchased	training	1087	0
botometer-feedback	training	138	380
political-bots	training	62	0
botwiki	testing	698	0
verified	testing	0	1987
midterm-18	testing	42446	8092
gilani-17	testing	1090	1394
cresci-rtbust	testing	353	340

We will give a synopsis of how each dataset was collected, and what domain of Twitter they came from. The dataset *varol* was created by manually labeling accounts which were gathered by sampling Tweets that fell into each decile of bot score given by Botometer (Varol et al. 2017). The intention of sampling from each decile is to have a diverse set of data which represents a range of bot and human behavior. Each bot account in *cresci-17* fell into one of three categories: social spambots, traditional spambots, and fake followers (Cresci et al. 2017). However, we filtered out the fake followers sub-dataset to try to better match the number of observations listed by Yang et al. (2020) (discrepancies between our datasets are recounted in the discussion). There are only bots in *pronbots*, and they were identified by them sharing scam sites (Yang et al. 2019). The observations in *celebrity* are a selection of celebrity Twitter accounts. The dataset *vendor-purchased* contains bots which were obtained by researchers buying fake followers. There are both human and bot accounts in *botometer-feedback*, which were collected by manually labeling classifications flagged for feedback by Botometer users. The dataset *political-bots* contains just 62 bot accounts involved in Massachusetts politics. The accounts in *botwiki* came from the website botwiki.org, and were filtered down to only contain active Twitter bot accounts (Yang et al. 2020). There are only humans in the *verified* dataset, which was created by using Twitter's streaming API and filtering out non-verified accounts. Yang et al. (2020) also manually labeled the data in *midterm-18*, which came from accounts that were active in discourse surrounding the 2018 U.S. midterm elections. The dataset *gilani-17* consists of humans and bots that were sampled with Twitter's streaming API, and were grouped into different numbers of followers (Gilani et al. 2019). There are an almost equal number of humans and accounts in *cresci-rtbust*. It was created by manually labeling Italian accounts that retweeted in June of 2018 (Mazza et al. 2019).

Most of the datasets consisted of a labeled .tsv where one column was a Twitter account ID, and the other column labeled the account as a bot or human. Ten of the datasets also came with a JSON file which contained eight user metadata features (see Table 2) that we used to train the model. We combined the JSON information with the labeled .tsvs into a usable format by having each observation be a row in a CSV file. One of the datasets, *cresci-17* (Cresci et al. 2017), consisted of seven individual CSV files, each already containing the eight metadata features. Thus, we combined all of these into a single file representing the observations of *cresci-17*. Another training set, *varol* (Varol et al. 2017), didn't contain any metadata features, and so we made a Twitter API call for each account to collect this information.

Feature Generation

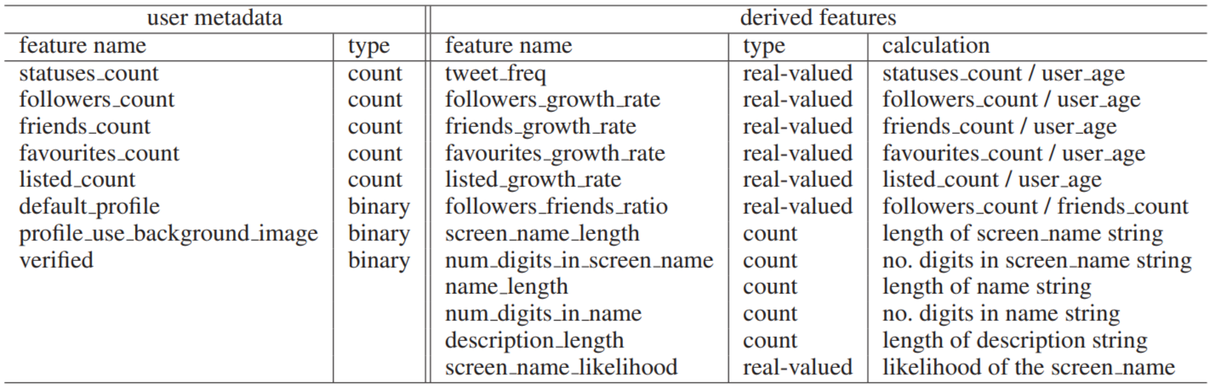
Besides eight raw features, we derive twelve more features based on the metadata features and three more account features: the age of the account, screen name (handle) of the account, and chosen name of the account. We hypothesize that most of the twenty features would matter to the bot detection. Each decision tree in the model will take into consideration four features and have a relatively efficient height. Including more than twenty features in the model would slow down the training and detection process.

One of our derived features was called *screen_name_likelihood*. This feature measures how similar that the observed *screen_name* feature is to all the observed human *screen_name*s. Therefore, in order to calculate this, we did the following steps:

1. Gather all the *screen_name*s of human accounts from all training datasets.
2. For each *screen_name* as a string of characters (letters or digits or the underscore), create all bigrams of two consecutive characters in a list. For example, the string 'data' would break down to ['da','at','ta'].
3. Count the appearance of all bigrams. Note that we are case sensitive.
4. In the meantime, count the appearance of all single characters.
5. Apply add-one smoothing. That is, for each possible bigram, add one to the count. This way, a bigram that never appears in any human *screen_name* (like 'xy') will be counted once. Change the count of single characters correspondingly to match/reflect the counts of bigrams.
6. Normalization: calculate the probability of appearance of each single character by dividing the count of the character by the sum of count of all characters. Similarly, calculate the probability of appearance of each bigram by dividing the count of the bigram by the sum of count of all bigrams.
7. Calculate the conditional probability of one character *c'* after another *c* by dividing the probability of *cc'* by the probability of *c*.
8. Lastly, calculate the likelihood of a *screen_name*, which represents how likely it is for an account with a specific *screen_name* to become a human account. This likelihood should be affected by the bigram. That is, take into account how likely it is for each character in the *screen_name* to appear after its previous. To do this, we multiply the conditional probabilities of all its bigrams and then apply the (*length of screen_name*-1) th root to increase the likelihood of longer *screen_name*s in comparison to shorter ones (Feng et al. 2021).

After the cleaning process, we have thirteen CSV files, seven training sets and five testing sets, each containing labeled observations with twenty features. See table 2 for the full list of features and table 1 in the appendix for an example row with derived features.

Table 2. The list of all raw and derived features from datasets.



user metadata		derived features		
feature name	type	feature name	type	calculation
statuses_count	count	tweet_freq	real-valued	statuses_count / user_age
followers_count	count	followers_growth_rate	real-valued	followers_count / user_age
friends_count	count	friends_growth_rate	real-valued	friends_count / user_age
favourites_count	count	favourites_growth_rate	real-valued	favourites_count / user_age
listed_count	count	listed_growth_rate	real-valued	listed_count / user_age
default_profile	binary	followers_friends_ratio	real-valued	followers_count / friends_count
profile_use_background_image	binary	screen_name_length	count	length of screen_name string
verified	binary	num_digits_in_screen_name	count	no. digits in screen_name string
		name_length	count	length of name string
		num_digits_in_name	count	no. digits in name string
		description_length	count	length of description string
		screen_name_likelihood	real-valued	likelihood of the screen_name

Note. From "Scalable and Generalizable Social Bot Detection through Data Selection," by Yang, K.-C., Varol, O., Hui, P.-M., & Menczer, F., 2020, *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(01), 1096-1103. https://doi.org/10.1609/aaai.v34i01.5460

To evaluate our models, we found the correlation in how they classified unseen accounts to how a previously trained model named Botometer (Varol et al. 2017) classified them. We used the Python package Tweepy (Harmon et al.) to first sample several hundred Twitter accounts that were making live Tweets at the time of data collection. Next, we made requests to Botometer's API to find its overall bot score assigned to each account using the screen names of accounts. We used Botometer to score a total of 401 unseen accounts. We had to further derive the features in Table 2 for each sampled account so that we could assign bot scores to these accounts in order to classify them using our models as well.

Results

Top models

Table 3 below shows the AUC scores calculated by the paper after their training and testing. We then show a comparative table that has our results compared to results by (Yang, et al. 2020) in Table 4. From our training, the paper's purported top three models are ranked low in our testing with ranks 90, 91, and 84. We also used fewer subsets in our top models than theirs – while they always included *varol, cresci_17, celebrity,* and *botometer* for training, we use *cresci_17* and *botometer* in only three of our top five models training. We also don't use *political* in any of the top five models, while Yang, et al. (2020) uses it in their top model training.

Table 3. AUC scores (from Yang et al. (2020)) on unseen datasets, five-fold cross validation, and correlation with Botometer for selected candidate models. M246 is trained on all data, shown for comparison. Their bolded values indicate metrics significantly better than the baseline.

Metric	Botometer	M196	M125	M191	M246
botwiki-verified	0.92	**0.99**	**0.99**	**0.99**	0.91
midterm-18	0.96	**0.99**	**0.99**	**0.98**	0.83
gilani-17	0.67	**0.68**	**0.69**	**0.68**	0.64
cresci-rtbust	0.71	0.60	0.59	0.58	0.57
5-fold cross-validation	0.97	**0.98**	**0.98**	**0.98**	0.95
Spearman's r	1.00	0.60	0.60	0.62	0.60

Note. From "Scalable and Generalizable Social Bot Detection through Data Selection," by Yang, K.-C., Varol, O., Hui, P.-M., & Menczer, F., 2020, *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(01), 1096-1103. https://doi.org/10.1609/aaai.v34i01.5460

Table 4. AUC scores of our top models and their top selections, tested on the same metric.

	Our Top Models					Paper's Top Models			
Training Datasets	**M0101000**	**M0101010**	**M1010010**	**M0100000**	**M1010110**	**M196**	**M125**	**M191**	**M246**
varol			✓		✓	✓	✓	✓	✓
cresci_17	✓	✓		✓		✓	✓	✓	✓
pronbots			✓		✓				✓
celebrity	✓	✓				✓	✓	✓	✓
vendor					✓			✓	✓
botometer		✓	✓		✓	✓	✓	✓	✓
political						✓			✓
Testing Datasets & Metrics									
botwiki-verified	0.99	0.99	0.98	0.96	0.98	0.98	0.98	0.98	0.98
midterm-18	0.98	0.98	0.96	0.98	0.97	0.95	0.96	0.96	0.97
gilani-17	0.65	0.68	0.76	0.66	0.73	0.69	0.68	0.68	0.67
cresci-rtbust	0.76	0.68	0.64	0.76	0.64	0.62	0.65	0.65	0.64
5-fold_cross-validation	0.998	0.997	0.988	0.997	0.987	0.991	0.990	0.988	0.994
spearman_r	0.67	0.63	0.69	0.66	0.71	0.49	0.48	0.50	0.55
Rank based on our training	1	2	3	4	5	90	91	84	61

Datasets analysis

We then perform further investigation on how datasets relate to each other. We want to see whether human and bot classes are consistent across different datasets, which is visualized through Figure 2 by training on one dataset and testing on another. Figure 2 maps the cross-dataset AUC score using random forest classifiers. In the heatmap analysis (Figure 2), when the AUC score is high, it indicates that the datasets are more consistent with each other. When AUC is below 0.5, it suggests that the training and test datasets have contradictory labels. To enable training on single-labeled datasets, we combined datasets that have only bots or humans with the same proportion of the other class from the *midterm* dataset.

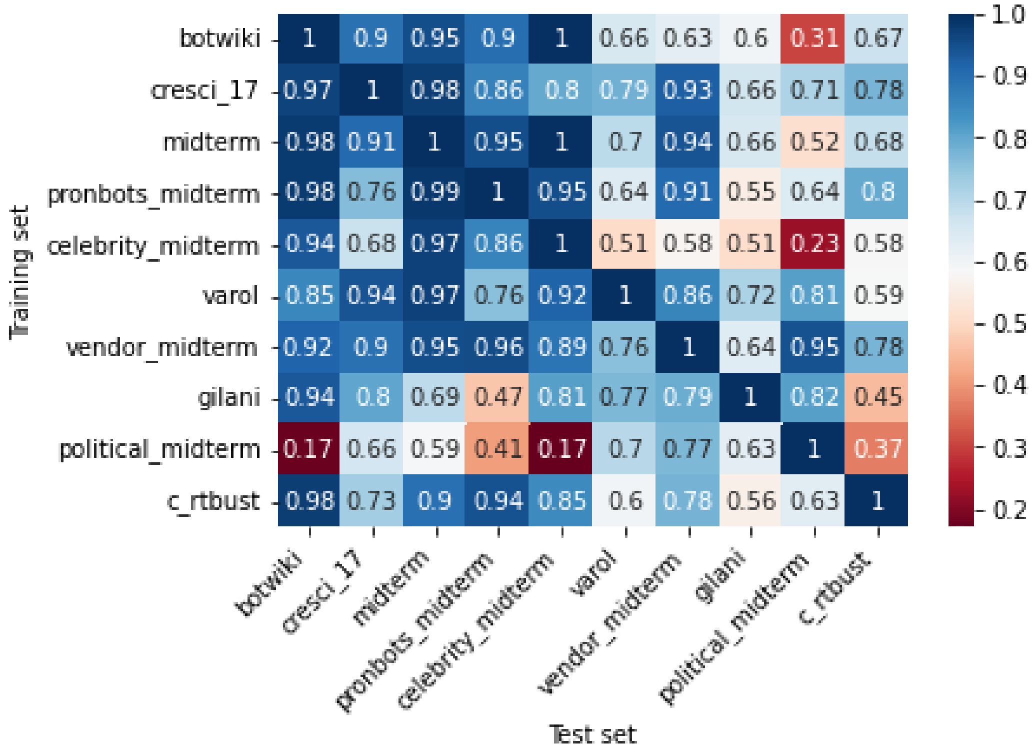
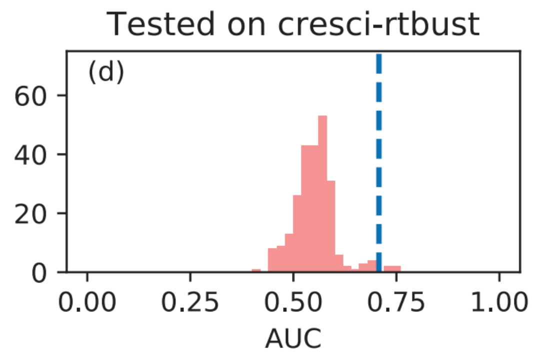


Figure 2. Heatmap for cross-dataset AUC using random forest classifiers. Training on one dataset and testing on another.

Based on this heatmap showing cross-datasets AUC, we can see that training on *political* generates bad models when tested on most other datasets. The relatively small dataset size contributes to the bad performance, but the AUC scores are generally low when we do testing on *political* as well. The low scores for *political* indicates that we shouldn't use it for training as it contradicts with most datasets. This aligns with our top model's datasets selection, which all omit *political*.

Of the three top models originally presented by Yang et al. (2020), all were trained on *varol*, *cresci_17*, *celebrity*, and *botometer-feedback*. However, out of our top 25 models, none were trained on both *varol* and *cresci_17* at the same time. Likewise, *varol* and *celebrity* were never trained on the same top 25 models. Interestingly, the conditional probability that given one of our 25 models was trained on *botometer-feedback*, it was also trained on *varol,* was only 0.2. See Figure 7 in appendix for the full list of conditional probabilities. These probabilities demonstrate that the composition of the training datasets of not just our top 5 models, but even our top 25 models, differ drastically from the top models in Yang et al. (2020).

As we discuss later in greater detail, some of the discrepancies between our models may be due to systematic differences between our datasets and their datasets. For instance, our models have a higher average AUC scores when tested on *cresci-rtbust* compared to all of the models of Yang et al. (2020).



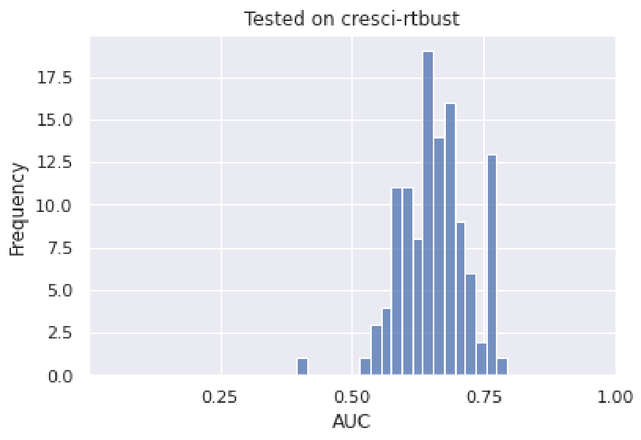


Figure 3. Left: The AUC scores of the 247 models trained by Yang et al. (2020) and tested on *cresci-rtbust* (a set of Italian retweets). The dotted blue line is Botometer's score on *cresci-rtbust*. Right: The AUC scores of our 119 models tested on *cresci-rtbust*.

Note. From "Scalable and Generalizable Social Bot Detection through Data Selection," by Yang, K.-C., Varol, O., Hui, P.-M., & Menczer, F., 2020, *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(01), 1096-1103. https://doi.org/10.1609/aaai.v34i01.5460

Hyperparameters analysis

The choice of hyperparameters can influence the performance of models. Yang et al. (2020) only gave the number of decision trees present in their models, and didn't disclose any other hyperparameters they used. Our goal here is to analyze how hyperparameters specific to random forest models impact the generalizability and accuracy of our models.

In our analysis, we test four hyperparameters: criterion, max_depth, class_weight, and warm_start. The results of this analysis are obtained by averaging the AUC scores of each metric, of which there are 119, then averaging the average scores of each metric, of which there are six. This is how the scalar performance of all models in the model set are determined.

To determine the top 30 models in the model set rather than all 119, the models are ranked from best to worst and only the top 30 scores are averaged the same way, averaging the AUC scores of each metric, then averaging the average score of each metric, except with just the top 30 models.

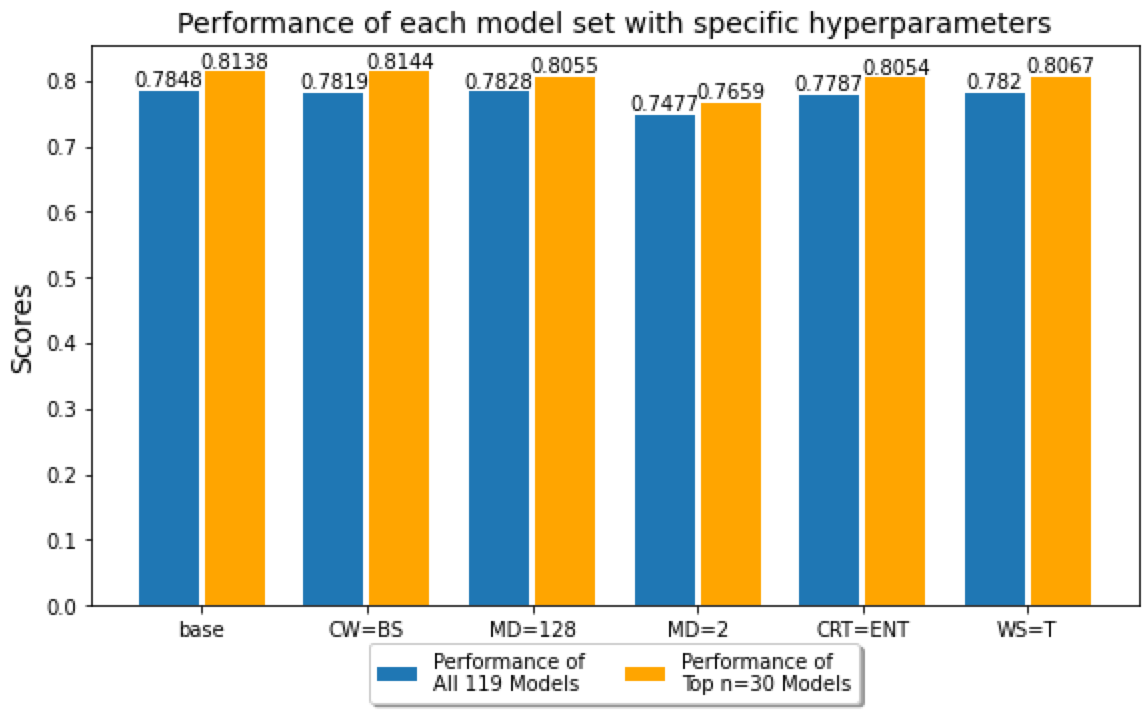


Figure 4. The average AUC scores for every model set with specified hyperparameters, across all 119 models and the top 30 models.

- cw=bs -> class_weight=”balanced_subsample”
- md -> max_depth
- crt=ent -> criterion = “entropy”
- ws=t -> warm_start = True

Based on our current analysis, the hyperparameter/setting with the best positive influence on the top 30 models is the class_weight=“balanced_subsample,” which increases the total score by 0.0006 points over the base model in the same metric, but does worse in the overall performance, with a 0.0029 reduction in score. This means the effect the hyperparameter class_weight=“balanced_subsample” has on all the models is negligible.

SHAP analysis

We also replicate the SHAP analysis that the original paper performed. With its fullname as SHapley Additive exPlanations, SHAP analysis calculates Shapley values with a method from coalitional game theory that assumes each feature value of an instance is a “player” in a game where the output is the payout. SHAP measures how each feature matters in their contribution to the output. Here we use treeSHAP, which can be used for the estimation of feature importance for tree-based models (Molnar, 2022).

We perform SHAP analysis on our best model as well as M196, the best model from Yang et al. (2020). See Figure 5 of the appendix for full results. The order of importance of features turns out to be very different with our best model and our M196, as well as our M196 with their M196. This shows that we were not able to fully replicate the original paper. However, our results are sustained: the effects of each feature are relatively consistent across different models. This can be seen by comparing the shapes of each feature in our best model and our M196. Only *tweet_freq* and *description_length* are significantly different. In comparing our M196 with the original paper's M196, we find that shapes of almost all of the features differ and the *friends_count*, *default_profile*, *statuses_count*, and description_length differ significantly. This result further shows that we did not manage to replicate the model and explains why the AUC scores also shifted.

Discussion

Robustness of our results

Although our best models are not trained on the same training datasets as the original paper's best models, and we achieved a different model using the dataset combination that produced their best models, our replication resembles the majority of results from the original paper. One of the important conclusions shared between the original paper and our results shows that the models that produce the best results are not trained on the combination of all datasets. Indeed, our analyses performed on training datasets suggest that some datasets are contradictory to each other (the datasets in the original paper also share this quality). Therefore, including both of them will worsen the model.

Challenges in replication

As shown in our results, we are unable to precisely replicate several parts of the authors' original paper. Though it's difficult to determine the exact impact, we know there are several discrepancies between our datasets, despite them coming from the same source. One big difference is in the training set *cresci-17*. We calculate there to be 3,474 humans and 6,543 bots, which aligns with the paper that was published alongside the dataset (Cresci et al. 2017). However, the purported number of humans in (Yang et al. 2020) was 2,764 and the number of bots was 7,049. No combination of the eight .CSVs in (Cresci et al. 2017) lead to a total of 7,049 bots.

Our version of *varol* is also different in both the number of observations and the feature values themselves. Since we have to make API calls to Twitter several years after the publication of (Yang et al. 2020), some accounts are deactivated or banned. We go from 1,495 humans in their dataset to 1,441 in ours, and 733 bots in theirs to 665 humans in ours. The fact that only 68 of the bot accounts are banned or deactivated may suggest that some of them could have been mislabeled in the manual annotation process of the original dataset. Similarly, the exact feature values of each observation would differ because we gather them at a later date from Yang et al. (2020).

The rate limit of Twitter's API and Botometer's API both present issues in our replication. Although it is only one of twenty features, we derive *screen_name_likelihood* from a different set of data. We train our language bigram model on the 11,213 human accounts in the training data, whereas the researchers were able to train their language model on over 2 million account names. In the construction of the data used to test the correlation between our models and Botometer, Yang et al. (2020) sampled 100,000 Twitter accounts. We are only able to classify 401 accounts from the live sample of Tweets due to Botometer's API rate limits.

Lastly, the researchers did not specify their hyperparameters of their random forest models. The only specifics they wrote was that they used 100 trees in the forest and implemented the models with the Python package scikit-learn (Pedregosa et al. 2011). One hundred trees (or estimators) is the default in scikit-learn, so it is impossible to know if the researchers used any non default hyperparameters at all. The secrecy of the researchers could be linked to them publishing their classifier, now titled "BotometerLite", as a premium API endpoint, costing $50/month (Menczer, 2020). Without access to the authors' exact datasets and their model hyperparameters, we have to work within our limitations by making many assumptions. This made it difficult to confidently replicate specific parts of *Scalable and Generalizable Social Bot Detection through Data Selection*.

Future directions

Apart from the various types of analyses we performed, several further directions arise from this topic. A possible future direction is to figure out a more effective and elastic way to select data from datasets. For example, if we decide to exclude data, we might not need to discard a whole dataset. Perhaps we can divide data entry-wise/instance-wise instead of dataset-wise. In addition, the current data selection method we followed is done by training all of the models and evaluating them to find out the best ones. This is relatively time consuming as it takes an exponential amount of time to train the models with respect to the number of datasets. Is it possible to determine whether to include or exclude some training instances in the process of training, such that all the selected training instances would output a good model?

In our experiment, we do not try any datasets besides what is mentioned in the original paper. We believe that we can better test the robustness of a random forest model and this data selection method if we can introduce new datasets. We also wonder whether it is possible to come up with a better ranking algorithm that is more holistic across all metrics, since multiplying the ranks for each metric only favors the models that do extremely well in a subset of the metrics. In general, the goal is to find a model that performs great in most if not all of the metrics and does not significantly fail in any metric.

Last but not least, bot detection is important, so we generally encourage further research and training of structurally improved models that also possibly takes in more recent training datasets.

Conclusion

In summary, we successfully replicate the procedure of training all models on every possible subset of datasets, applying cross-validation, calculating each of their performances on every test dataset, and performing analyses on the top models. We find that the resulting top models are different from the original papers' models, and further directions of investigations are raised.

References

Cresci, S., Pietro, R.D., Petrocchi, M., Spognardi, A., & Tesconi, M. (2017). The Paradigm-Shift of Social Spambots: Evidence, Theories, and Tools for the Arms Race. *Proceedings of the 26th International Conference on World Wide Web Companion.*

Deb, A., Luceri, L., Badaway, A., & Ferrara, E. (2019, May). Perils and challenges of social media and election manipulation analysis: The 2018 us midterms. In *Companion proceedings of the 2019 world wide web conference* (pp. 237-247).

Feng, S. (n.d.). *TwiBot-20: A comprehensive twitter bot detection benchmark*. Retrieved November 16, 2022, from https://arxiv-export-lb.library.cornell.edu/pdf/2106.13088

Gilani, Z., Kochmar, E., & Crowcroft, J. (2017, July). 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* (pp. 489-496).

Harmon, Roesslein, J., & other contributors. Tweepy [Computer software]. https://doi.org/10.5281/zenodo.7259945

Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome (2008). The Elements of Statistical Learning (2nd ed.). Springer. ISBN 0-387-95284-5

Lee, K., Eoff, B., & Caverlee, J. (2021). Seven Months with the Devils: A Long-Term Study of Content Polluters on Twitter. *Proceedings of the International AAAI Conference on Web and Social Media,* 5(1), 185-192. https://doi.org/10.1609/icwsm.v5i1.14106

Mazza, M., Cresci, S., Avvenuti, M., Quattrociocchi, W., & Tesconi, M. (2019, June). Rtbust: Exploiting temporal patterns for botnet detection on twitter. In *Proceedings of the 10th ACM conference on web science* (pp. 183-192).

Menczer, F. (2020, September 1). *Botometer v4*. CNetS. Retrieved November 18, 2022, from https://cnets.indiana.edu/blog/2020/09/01/botometer-v4/

Molnar, C. (2022, November 12). *Interpretable machine learning*. 9.6 SHAP (SHapley Additive exPlanations). Retrieved November 16, 2022, from https://christophm.github.io/interpretable-ml-book/shap.html

Pedregosa, F., Varoquaux, Ga"el, Gramfort, A., Michel, V., Thirion, B., Grisel, O., … others. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(Oct), 2825–2830.

Varol, O.; Ferrara, E.; Davis, C. A.; Menczer, F.; and Flammini, A. 2017. Online human-bot interactions: Detection, estimation, and characterization. In *Proc. Intl. AAAI Conf. on Web and Soc. Media (ICWSM)*.

Yang, K. C., Varol, O., Davis, C. A., Ferrara, E., Flammini, A., & Menczer, F. (2019). Arming the public with artificial intelligence to counter social bots. *Human Behavior and Emerging Technologies*, 1 (1), 48–61.

Yang, K.-C., Varol, O., Hui, P.-M., & Menczer, F. (2020). Scalable and Generalizable Social Bot Detection through Data Selection. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(01), 1096-1103. https://doi.org/10.1609/aaai.v34i01.5460

Appendix

Figure 1. Our calculated AUC scores and Spearman's r value for the test metrics on each model.

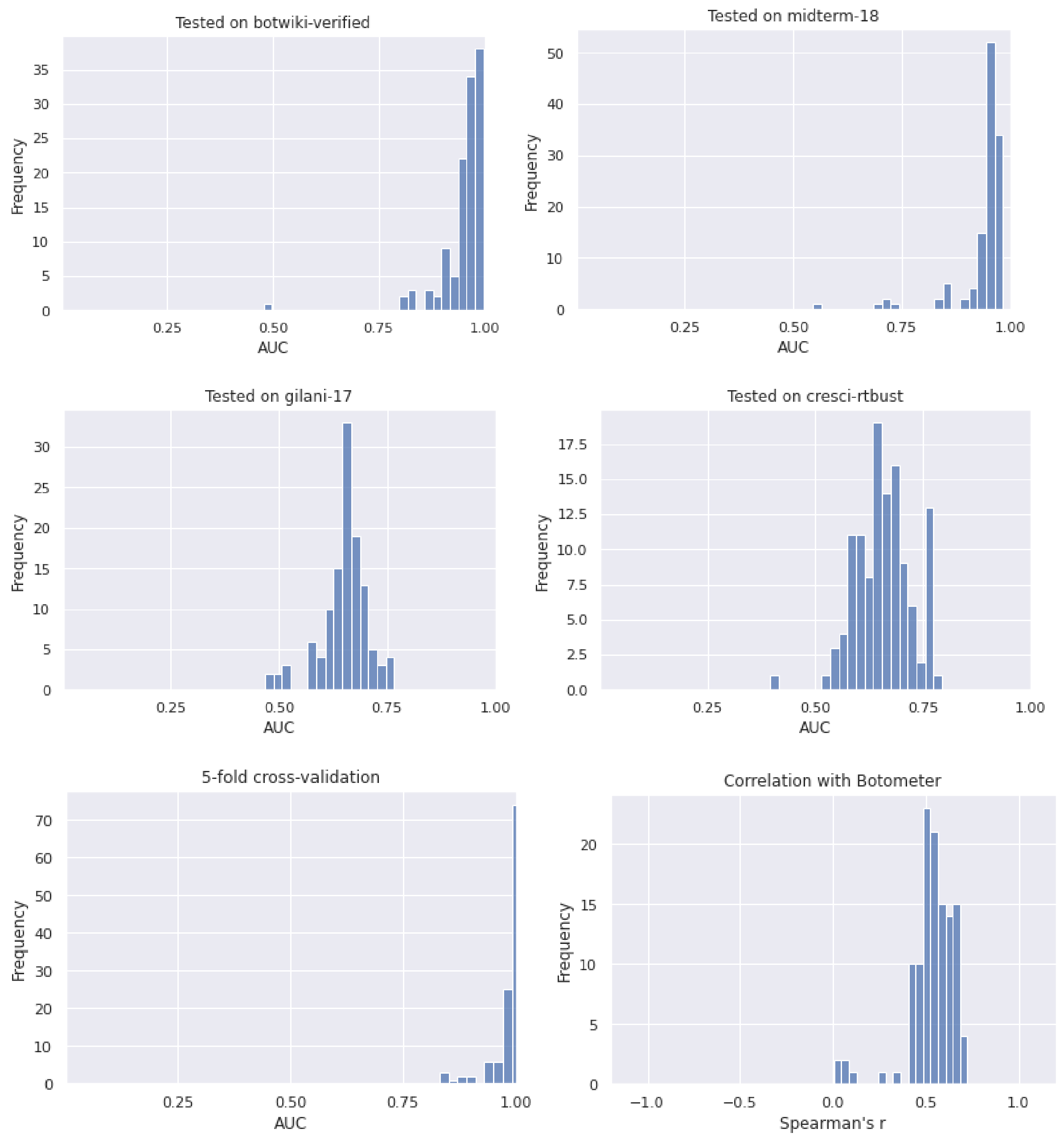
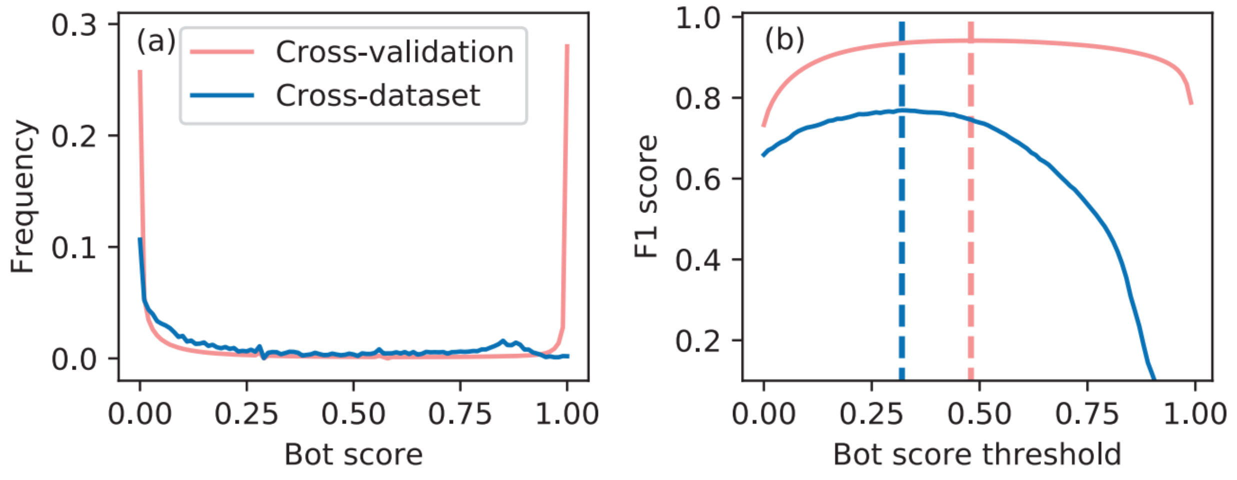


Figure 2. Top: Yang et al. (2020) figures showing the frequency of bot scores and F1 score for their top model on cross-validation data and cross-domain (testing) datasets. The threshold (vertical dotted line) denotes the bot threshold which maximizes the F1 score. Bottom: We recreated the figures using the model trained on the same data as their top model. (M196/M1101011). Note that we have a probability density plot instead of frequency, so the y axis is different, but their relative shapes are still comparable.



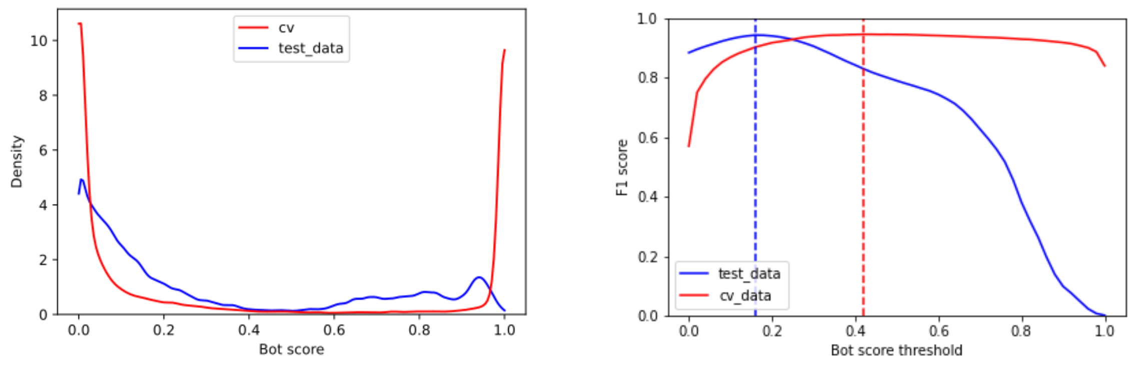


Figure 3. We calculated the bot scores, F1 scores, and thresholds for our top model (M0101000).

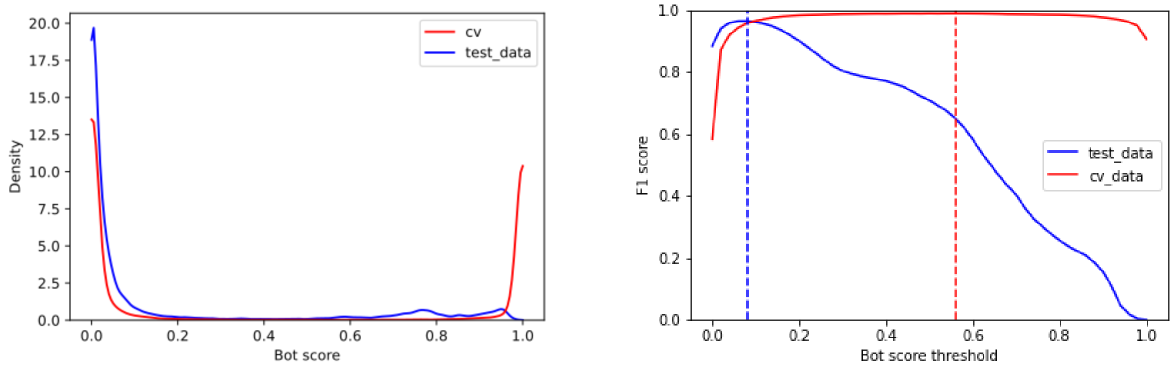


Figure 4. SHAP analysis scatterplots. Each point is a test instance. The more left it is, the more human-like the model predicts it is. Vice versa. The redder it is, the higher the feature value it is. Vice versa.

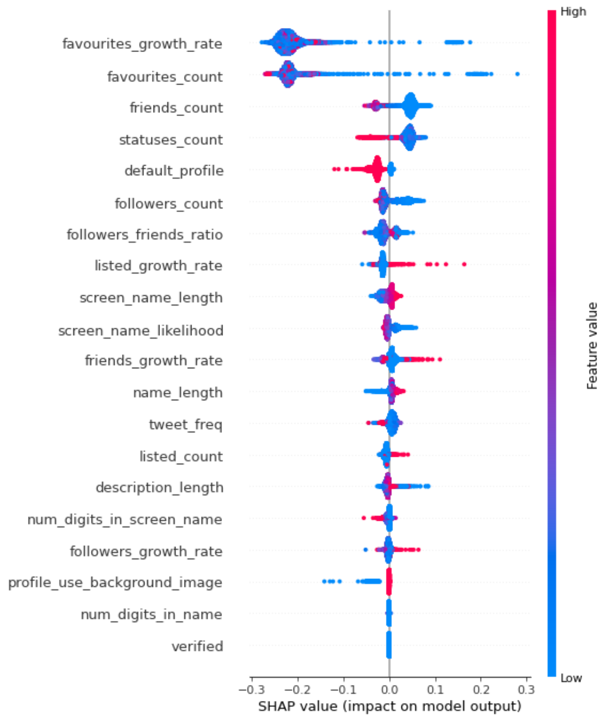


Figure 4.1.1. trained on *cresci-17*, tested on *pronbots*.

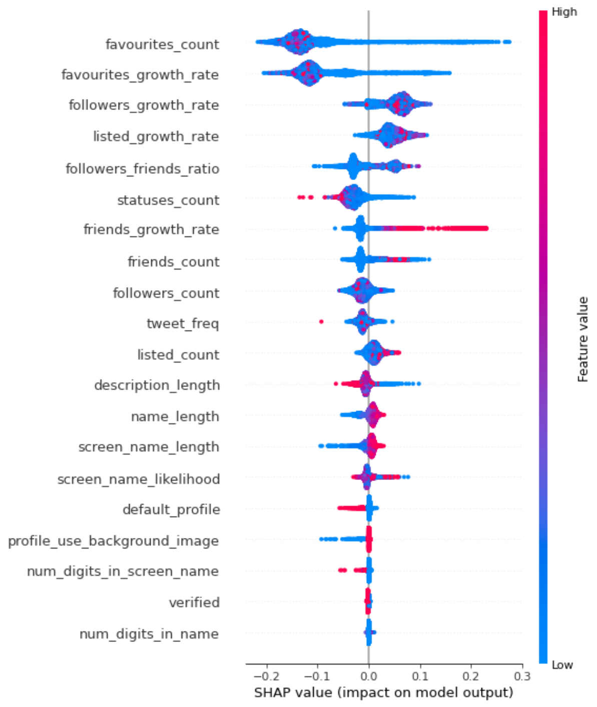


Figure 4.1.2. trained on *cresci-17*, tested on *celebrity*.

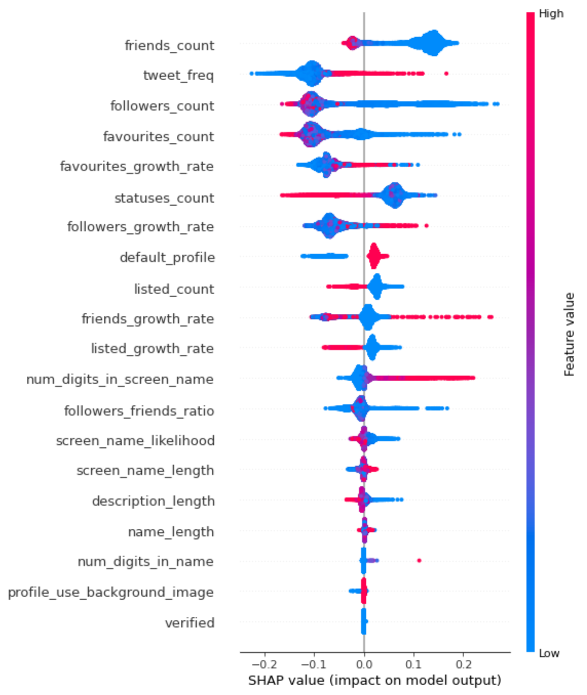


Figure 4.2.1. trained on *midterm-18*, tested on *pronbots*.

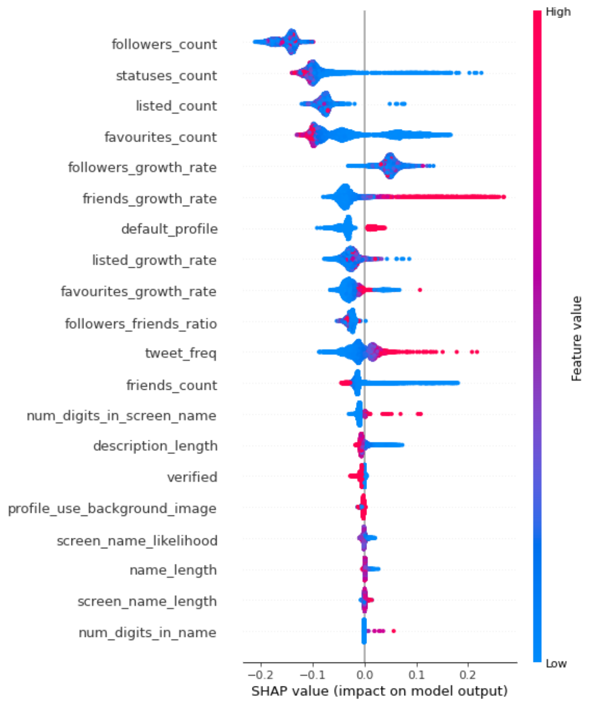


Figure 4.2.2. trained on *midterm-18*, tested on *celebrity*

Figure 5. SHAP analysis on M196 on all 4 test dataset

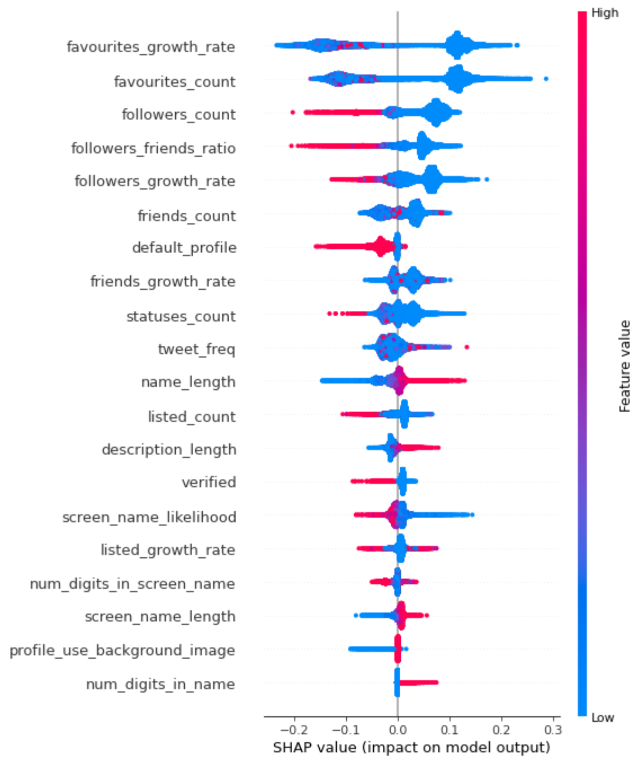


Figure 6. SHAP analysis on M0101000 (our best model) on all 4 test dataset

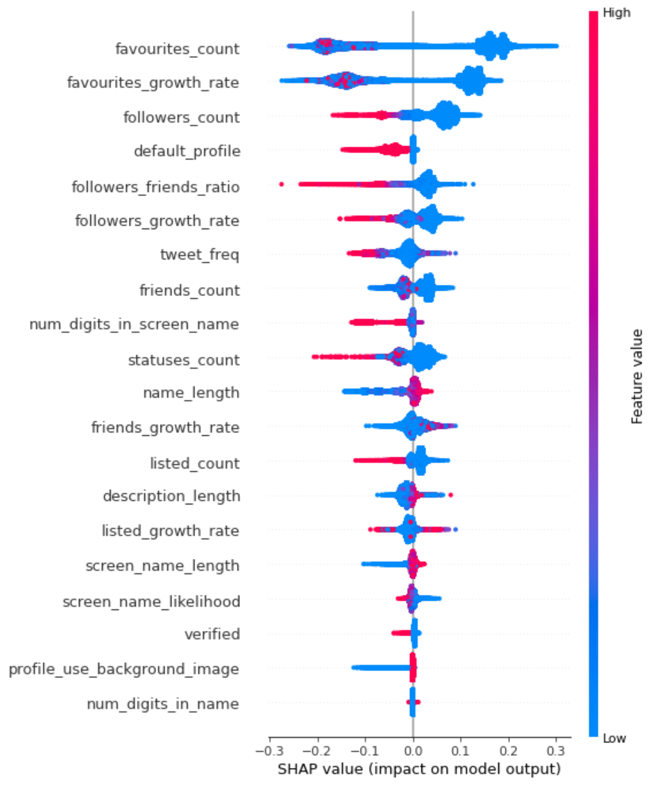


Figure 7. The conditional probabilities that when given a certain data set (row) used to train a top 25 model, another data set (column) was also used to train the same model.

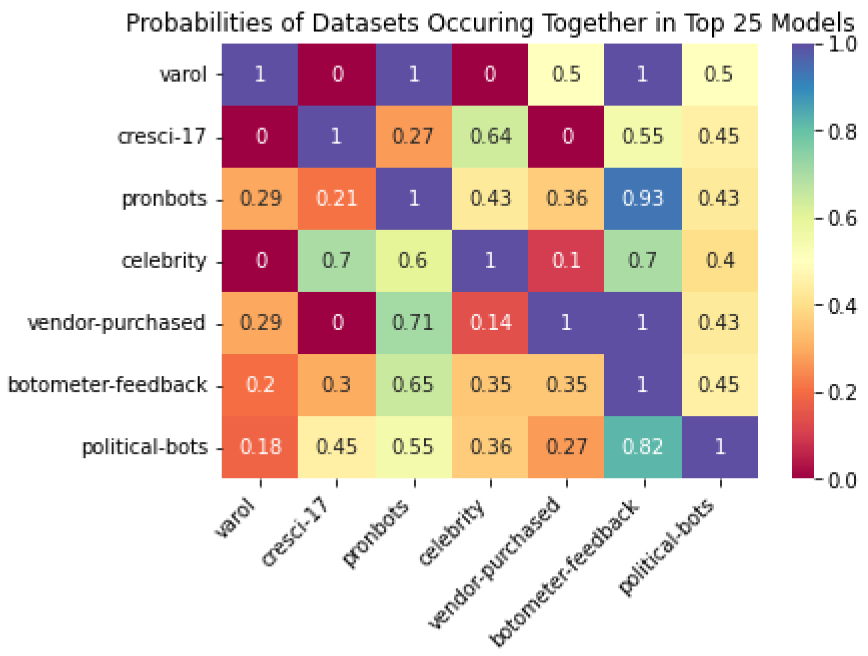


Table 1. Two example rows of a dataset used for training with all twenty features and label is_bot.

statuses_count	followers_count	friends_count	favourites_count	listed_count	default_profile	profile_use_background_image	verified	tweet_freq	followers_growth_rate	friends_growth_rate
39411	651238	665	13561	1761	FALSE	FALSE	TRUE	0.4665833527	7.709949239	0.007872876343
41187	265102	943	11365	783	FALSE	FALSE	TRUE	0.5356097169	3.447476319	0.01226309183

...

favourites_growth_rate	listed_growth_rate	followers_friends_ratio	screen_name_length	num_digits_in_screen_name	name_length	num_digits_in_name	description_length	screen_name_likelihood	is_bot
0.1605474828	0.02084832367	979.3052632	11	0	11	0	117	0.002322266614	0
0.1477943145	0.01018239756	281.126193	13	0	19	0	40	0.005185556779	0