I Spot a Bot: Building a binary classifier to detect bots on Twitter

Jessica H. Wetstone, *wetstone@stanford.edu* and Sahil R. Nayyar, *srnayyar@stanford.edu*

I. Introduction

It has been estimated that up to 50% of the activity on Twitter comes from bots [1]: algorithmically-automated accounts created to advertize products, distribute spam, or sway public opinion. It is perhaps this last intent that is most alarming; studies have found that up to 20% of the Twitter activity related to the 2016 U.S. presidential election came from suspected bot accounts, and there has been evidence of bots used to spread false rumors about French presidential candidate Emmanuel Macron and to escalate a recent conflict in Qatar [1]. Detecting bots is necessary in order to identify bad actors in the "Twitterverse" and protect genuine users from misinformation and malicious intents. This has been an area of research for several years, but current algorithms still lag in performance relative to humans [2].

Given a Twitter user's profile and tweet history, our project was to build a binary classifier that identifies a given user as "bot" or "human." The end-user application for a classifier such as this one would be a web plug-in for the browser that can score a given account in real-time (See page 5 for mock-ups). All of the raw inputs required to classify a public Twitter account via our algorithm are available for download from the Twitter API; in fact, our `check_screenname.py` program is a working prototype that uses the API to classify a given Twitter user handle within seconds. It is our opinion that a product like this is sorely needed for the average Twitter consumer.

II. Related Work

1) Academic: Bot detection on Twitter has been an area of active research for several years. Interestingly, we found much variation in the literature on the definition of "bot", which mirrors the growth in sophistication of these accounts over time. Early research, such as that by Yang et al., focused primarily on the detection of "spambots", which unlike bots that attempt to mimic human behavior, aim primarily to spread malware and unwelcome advertising [3]. Features such as friend-follower ratio and number of urls per tweet are successful in identifying these types of bots. In 2015, Cresci et al. attempted to use classical spam bot detection techniques to identify a specific kind of Twitter bot, the "fake follower", which is a bot employed solely for the purpose of elevating other accounts' numbers of followers. They found similar feature sets to those used for traditional spam bots to also be successful in identifying fake followers [4]. A later study by the same authors investigates the efficacy of those techniques on identifying "social bots", bots who attempt to spoof human users [5]. They found the classical feature sets lacking for that type of bot, and suggest relationship-based techniques may prove the most fruitful for future investigations. Yet another definition of "bot" as used by Varol et al. utilizes an even broader definition, including any account primarily manned by software (including accounts of genuine users whose tweet content is published by Instagram or other apps) [6].

The approach to identifying bots, regardless of type, falls into a few different buckets. [4],[5], and [6] all investigate using a set of features derived from user profile metadata and aggregated statistics based on timeline data (tweet history). Our approach also follows this pattern, since those features are easiest to extract almost immediately from the Twitter API, making them most conducive to inclusion in a real-time end-user application. [6] builds on this feature set by also including sentiment-analysis based features, including happiness and arousal scores. Finally, a promising approach that has been used by Tacchini et al. for Fake News detection on Facebook [7] as well as by Lee et al. for bot detection on Twitter [8] is the use of community-related features, which take advantage of the fact that bots are often created at the same time. To increase the verisimilitude of these accounts, these bots will also follow each other and like each others' posts. The main drawback of this approach is that gathering follower information on Twitter is time-consuming, as the information is not immediately available in the API.

2) End User Applications: There are two products currently on the Chrome App Store marketed to identify bots on Twitter: "Botson", an app powered by the Botometer API from Indiana University (https://botometer.iuni.iu.edu/) with 247 users and "Botcheck.me" with 3,670 users (as of December 14, 2017). Both provide bot scores for users and will either automatically (Botson) or at the push of a button (Botcheck.me) evaluate a particular tweet. Their relatively low user uptake indicates that they have yet to become popular in the mainstream.

III. Dataset and Features

A. Data Sources

To develop our classification model, we made use of a variety of publicly-available datasets consisting of pre-labeled examples of bots and genuine users (Tables I and II), all of which are accessible via Indiana University's Bot Repository project [9].

The Training distribution, (Table I), which we used to train and cross-validate our models (Train and Train-Dev datasets), stems from two papers published by Cresci, et. al. in 2015 [4] and 2017 [5]. The table details the data collection methodology for these samples, which were collected over the period 2009 -

TABLE I. TRAINING DISTRIBUTION [4], [5]

Dataset	Users	%	Tweets	Year	Data collection methodology
genuineAccounts	3,474	31%	8,377,522	2011	Random sample of twitter accounts that were contacted and asked a question in natural language. Responders were considered genuine
fakeProject	469	4%	563,693	2012	"The Fake Project: Users were asked to follow a particular account if they werent fake – they were subsequently verified by a CAPTCHA
#elizioni2013	1,481	13%	2,068,037	2013	Two sociologists reviewed accounts that used the hashtag #elizioni2013 in their tweets between Jan 9 - Feb 28, 2013. All public figures and suspected bots were discarded
Human Total	**5,424**	**48%**	**11,009,252**		
romeElection2014	991	9%	1,610,176	2014	Group of bots discovered that were employed by a mayoral candidate during an election in Rome
#TALNTS	3,457	30%	428,542	2014	Group of bots discovered that were employed to promote #TALNTS, a mobile phone application
Amazon.com	464	4%	1,418,626	2011	Group of bots to promote products on Amazon.com
traditionalSpam	1,000	9%	145,094	2009	Researchers crawled twitter to find bots that tweeted malicious urls
Bot Total	**5,912**	**52%**	**3,602,438**		
Grand Total	**11,336**	**100%**	**14,611,690**		

TABLE II. TEST DISTRIBUTION [6]

Dataset	Users	%	Tweets	Year	Data collection methodology
Human	1,510	67%	1,437,889	2017	Accounts were randomly sampled by monitoring a twitter stream for several months in 2016; Four volunteers were then asked to classify the tweets. Accounts with clear labels were kept in the sample. The raw user and tweet data we downloaded from Twitter in December 2017 via the API.
Bot	735	33%	667,477	2017	
Total	**2,245**	**100%**	**2,105,366**		

2014. The prevalence of positive examples in this distribution is 52%.

The Test distribution (Table II) is meant to more closely approximate the current Twitter environment. We used this distribution for model cross-validation (Dev dataset) as well as for calculating our final results (Test dataset). This distribution is more representative of the true Twitter environment as the annotations were completed more recently (2016) and via a random sampling method by Varol, et. al [6]. We also downloaded fresh tweets and user profile information for each of the accounts. The prevalence of positive examples in this distribution is 33%.

B. *Preprocessing and Feature Selection*

Given the users' profile and tweet information from the datasets described in section 3.A, we generated twelve features, either immediately available from the Bot repository dataset/Twitter API, or from statistics that we derived (Table III). Many of the features that we used were directly available, such as (1) verification status, (2) followers count, (3) favorites count and (4) friends count. Friend-to-follower ratio (5) is a derived statistic that has been used in the literature for many years [3].

We calculated some of our other features by aggregating over users' tweets, specifically (6) number of mentions per tweet, (7) number of hashtags per tweet, (8) number of urls per tweet, (9) retweet count per tweet, and (10) favorite count per tweet [5]. We augmented this feature set with some novel statistics, including (11) unique tweet places count, calculated by counting the number of unique locations from which each user tweeted (though for many examples, the location data was unavailable), as well as (12) variance in tweet rate, which looked at variations in users' average number of tweets per hour.

All of our features were standardized by centering the mean at zero and scaling to unit variance, to prepare them for the model-fitting processes.

C. *Partitioning*

After downloading the raw data from the Twitter API and obtaining the desired features for each user, we partitioned the resulting data into four datasets: (1) Train, (2) Train-dev, (3) Dev, and (4) Test. To create the Train and Train-dev sets, we randomly split the examples from the Training distribution via a 80:20 ratio. We then created the Dev and Test datasets by splitting the Test distribution examples via a 50:50 ratio. The Train dataset was used to train the models; Train-dev and Dev were used for model cross-validation; and the Test dataset was used to report final results.

IV. MODELS

For our training set of 7,152 examples $i = 1, \dots, m$, each with a 12-dimensional feature vector $x^{(i)} \in \mathbb{R}^n$, we encoded the binary class label as $y^{(i)} \in \{0, 1\}$, where 0 denotes a genuine user and 1 denotes a bot. Given our training set $\{(x^{(i)}, y^{(i)})\}_{i=1}^m$ our goal was to learn a function $h : \mathbb{R}^n \to \{0, 1\}$ that accurately predicts the class label y of a given Twitter user with features x. We did this by performing (1) logistic regression, (2) gradient-boosted classification, and classification by a (3) multi-layer perceptron (MLP) neural network. All models were created and fitted using algorithms implemented in the "sci-kit-learn" Python library.

A. *Logistic Regression*

Although the target variable y can only take discrete values 0 and 1, we can approximate the relationship between y and x as a smooth function over the range $[0, 1]$. In logistic regression, the logistic (or sigmoidal) function is chosen to model this relationship:

$$h_{w,b}(x) = \frac{1}{1 + e^{-(w^T x + b)}} \tag{1}$$

Here, $w \in \mathbb{R}^n$ is the vector of weights, and $b \in \mathbb{R}$ is a bias (intercept) term. The actual estimate of the target variable, $\hat{y}$ is then set to whichever of the values $\{0, 1\}$ that $h_{w,b}(x)$

TABLE III. FEATURES

	Feature	Description
(1)	Verification status	True if user has an account that has been authenticated by Twitter (only applies to users of public interest)
(2)	Followers count	Number of followers the user has
(3)	Favorites count	Number of tweets the user has liked in the account's lifetime
(4)	Friends count	Number of users this account is following
(5)	Friend-to-follower ratio	Friends count / Followers count
(6)	Number of mentions per tweet	Count of user mentions in tweet sample "@screenname" / Number of tweets in sample
(7)	Number of hashtags per tweet	Count of hashtags in tweet sample "#topic" / Number of tweets in sample
(8)	Number of urls per tweet	Count of urls in tweet sample "http://www.url/" / Number of tweets in sample
(9)	Retweet count per tweet	Count of retweets in the sample (rebroadcasts of this user's tweets by other users)/ Number of tweets in the sample
(10)	Favorite count per tweet	Count of favorites in the sample (Likes of this user's tweets)/Number of tweets in the sample
(11)	Unique tweet places count	Count of unique tweet places tagged in the sample
(12)	Variance in tweet rate	Variance of tweet rate (# of tweets / hour) in hours the user was active

is closest to, e.g., $\hat{y} = 1\{h_{w,b}(x) \geq 0.5\}$. Geometrically speaking, this entire process is equivalent to dividing $\mathbb{R}^n$ into two subspaces: one for the $y = 1$ domain, and one for the $y = 0$ domain, both separated by a $(n-1)$-dimensional hyperplane. We solved for all parameters by minimizing the log-loss objective with L2-regularization to prevent over-fitting (this requires re-encoding the targets from $\{0, 1\}$ to $\{-1, 1\}$):

$$\min_{w,b} \quad \frac{1}{2}||w||_2^2 + C\sum_{i=1}^{m} \log\left(1 + e^{-y^{(i)}(w^T x^{(i)} + b)}\right) \tag{2}$$

We minimized this objective through performing coordinate descent on the associated dual objective, relying upon the LIBLINEAR C++ implementation interfaced through sci-kit-learn.[10] Through cross-validation with the train-dev set, we tuned the regularization constant to $C = 2.02$ for optimal performance.

B. Gradient-Boosted Classifier

While logistic regression divides the feature space via a linear decision boundary, this may not be good enough for more intricate distributions of positive and negative examples. If we use a decision tree as our model, however, then we can form more intricate decision boundaries that "zig-zag" across the feature space, thereby giving us much more predictive power than in the former case.

Gradient-boosted classification is a technique that uses a process called gradient tree boosting to recursively fit a decision tree model $h_t(x)$ to a training set $\{(x^{(i)}, y^{(i)})\}_{i=1}^{M}$ over a series of predetermined steps $t = 1, \ldots, T$. Let K_t be the tree's total number of "leaves" at step t. Then for all steps t, the tree partitions the input space into K_t adjoint regions $\{R_{kt}\}_{k=1}^{K_t}$: this partition is what forms the "zig-zag". We can represent the corresponding model, $h_t(x)$, in the following form:

$$h_t(x) = \sum_{k=1}^{K_t} b_{kt} 1\{x \in R_{kt}\}, \tag{3}$$

where b_{kt} are weighing coefficients.

We used Friedman's "TreeBoost" algorithm [11] to fit the decision tree to our training data, again using the $y \in \{-1, 1\}$ encoding. To give a brief description of the TreeBoost algorithm: it begins with an initial guess h_0, Then for each step the algorithm finds the split that optimizes the least-squares improvement criterion. The tree structure $\{R_{kt}\}_{k=1}^{K_t}$ is then updated with this new partition, by minimizing the negative binomial log-likelihood objective $L(y, \hat{y}) = \log(1 + e^{-2y\hat{y}})$ for the current step:

$$\gamma_{kt} = \underset{\gamma}{\text{argmin}} \sum_{x^{(i)} \in R_{kt}} L(y^{(i)}, h_{t-1}(x^{(i)}) + \gamma), \tag{4}$$

$$h_t(x) = h_{t-1}(x) + \sum_{k=1}^{K_t} \gamma_{kt} 1\{x \in R_{kt}\} \tag{5}$$

We fit a gradient-boosted classifier model to our data by repeating the above algorithm for $T = 100$ boosting stages.

C. Multi-Layer Perceptron Neural Network

Lastly, we fitted a multi-layer perceptron neural network with two hidden layers to our data, with the first and second hidden layers having 3 and 4 activator nodes, respectively. Given weight and bias parameters $\{W^{[l]}, b^{[l]}\}_{l=1}^{3}$ where:

$$W^{[1]} \in \mathbb{R}^{3\times n}, \quad W^{[2]} \in \mathbb{R}^{4\times 3}, \quad W^{[3]} \in \mathbb{R}^{1\times 4} \tag{6}$$
$$b^{[1]} \in \mathbb{R}^{3}, \quad b^{[2]} \in \mathbb{R}^{4}, \quad b^{[3]} \in \mathbb{R}, \tag{7}$$

the following forward-propagation sequence of equations represents our network's classification output $\hat{y}$, given a set of features x:

$$\hat{y} = 1\{W^{[3]} h^{[2]} + b^{[2]} > 0\} \tag{8}$$
$$h^{[2]} = \max\{0, W^{[2]} h^{[1]} + b^{[2]}\} \tag{9}$$
$$h^{[1]} = \max\{0, W^{[1]} x + b^{[1]}\} \tag{10}$$

To fit the weight and bias parameters to our data, we used the scipy implementation of the limited-memory BFGS algorithm [12], as interfaced through the sci-kit-learn package, to minimize the L2-regularized cross-entropy cost:

$$\min_{W^{[l]}, b^{[l]}} \quad \frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log y^{(i)} + \alpha \sum_{l=1}^{3} ||W^{[l]}||_2^2 + ||b^{[l]}||_2^2 \tag{11}$$

By cross-validation with the train-dev set, we found that setting $\alpha = 0.01$ gave optimal performance for our MLP classifier.

V. Results and Discussion

A. Evaluation Metrics

In our results, we consider various standard metrics to evaluate our models:

- **Accuracy:** Number of correct predictions ÷ all predictions
- **Precision:** Number of correctly predicted positive instances ÷ all positive predictions
- **Recall:** Number of correctly predicted positive instances ÷ all ground truth positive cases
- **F_β score:** A weighted harmonic mean of precision and recall (β specifies the weighting)

$$F_\beta = (1+\beta^2)\frac{precision * recall}{(\beta^2 * precision) + recall}$$

For our application, the severity of type-I error is greater than that of type-II; It is more offensive for a genuine user to be incorrectly labeled as a "bot" than the reverse. Therefore, we have chosen to use an $F_{\beta=0.50}$ score that more heavily weights precision as our primary evaluation metric.

B. Comparison of model performance

TABLE IV. Model Accuracy (threshold = 0.5)

	Datasets			
Models	Train	Train-Dev	Dev	Test
Linear Regr.	95.4%	96.0%	70.9%	72.5%
Grad. Boosted	99.9%	98.4%	74.8%	75.1%
Neural Net	97.6%	97.3%	78.0%	77.7%

TABLE V. $F_{\beta=0.5}$ scores (threshold = 0.5)

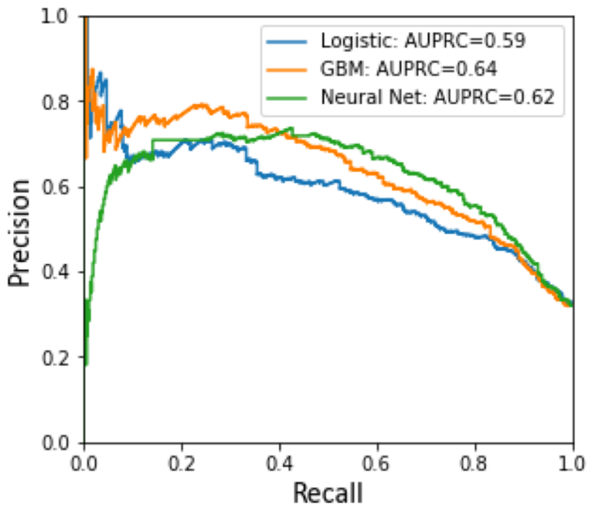
	Datasets			
Models	Train	Train-Dev	Dev	Test
Linear Regr.	0.960	0.962	0.527	0.547
Grad. Boosted	1.000	0.989	0.621	0.602
Neural Net	0.982	0.977	0.683	0.661

The neural network and gradient-boosted models had similar $F_{\beta=0.5}$ scores, with the neural network performing slightly better at a threshold of 0.5 (Table I.), but the gradient-boosted model having a larger AUPRC. Given the latter's larger AUPRC and higher accuracy for its most confident positive predictions (Figure 1), we would select the gradient-boosted classifier as our best model.

C. Distribution mismatch

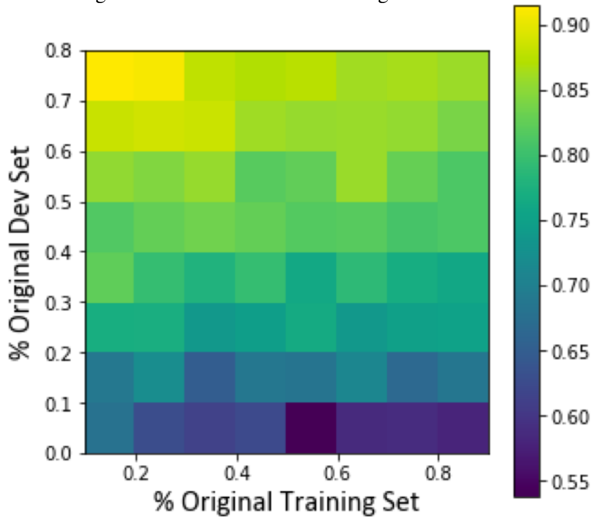
We found that all of our models generalized well on our train-dev dataset, which indicates that our hyper-parameter tuning was successful. In other words, the failure of the models to generalize well to our dev and test datasets was not a result of over-fitting. The most likely explanation for the discrepancy in performance is a distribution mismatch between the Training distribution (Train / Train-Dev datasets) and the Test distribution (Dev / Test datasets). We tested this hypothesis by gradually adding larger partitions from the dev dataset into

Fig. 1. Precision-Recall curves



our train dataset, training the gradient-boosted classifier on that augmented dataset, and then calculating the $F_{\beta=0.5}$ score on the remaining dev data (Figure 2). As expected, model performance improved both as more dev data was added (along the + y axis), and as the proportion of dev vs. training data increased (along the - x axis).

Fig. 2. Testing the distribution mismatch using the GBM model



Given the discrepancies between the two distributions, our suggested next step would be to augment our training dataset with additional examples to make it more representative of the real-world Twitter environment. We will address this and other enhancements in the "Future Work" section.

D. Ground truth reliability

Another potential cause of the distribution mismatch is the questionable reliability of "ground truth" labels in the Test distribution. The identity of users in the Training distribution is fairly reliable due to the data collection methodology for those datasets (Table I). By contrast, ground truth for the Test distribution was generated by agreement from a panel of four human volunteers, who reviewed the user profiles and timelines of each of the accounts. The labels were not verified further.

In order to test the quality of the labels, we asked our own group of 3 volunteers to label 25 randomly sampled accounts from our Dev dataset as "bot" or "human" based purely on viewing the user's profile and timeline. Results from this exercise are in Table VI. The average agreement of the volunteers with the "ground truth" labels was 79%, with individual scores of (72%, 80%, and 84%). This performance supports our suspicion that the labels in the Test distribution are somewhat dubious.

We also compared our model's predictions with the labels from our human volunteers (Column "SpotABot" in Table III). Interestingly, the model's incorrect labels correlated highly with the human volunteers, which suggests that our model did learn a predictive set of features. In the cases where all volunteers and our model were mistaken, it is likely that either the ground truth labels were incorrect in these instances, or that those Twitter bots are very sophisticated.

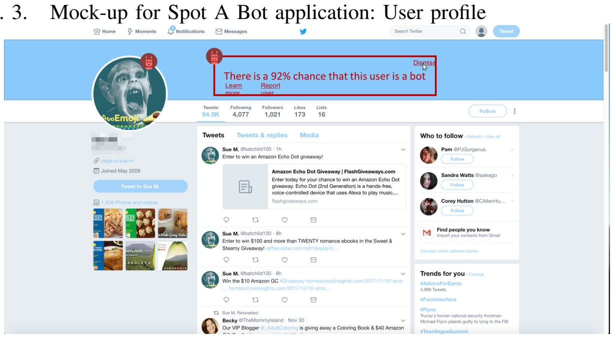
TABLE VI. VOLUNTEER LABELS VS. GROUND TRUTH

Screen name	Volunteer 1	Volunteer 2	Volunteer 3	SpotABot
Geekymz	Human	Human	Human	Human
chadmendes	Human	Human	Human	Human
idoasiroha	Bot	Human	Human	Human
ernie282849	Bot	Human	Human	Human
SouthbayRental	Bot	Bot	Bot	Bot
Chauw13	Human	Human	Human	Human
MadamMorticia	Human	Human	Human	Human
lokerPKU	Bot	Bot	Bot	Bot
socialwire878	Bot	Bot	Bot	Bot
gabbymarshall9	Human	Human	Human	Human
RyanMonge	Human	Human	Human	Human
MariaS1923	Bot	Bot	Human	Human
ElisabethD13	Human	Human	Human	Human
missleovirgo	Human	Human	Human	Human
HawkJam12	Human	Human	Bot	Human
LauraRc17	Bot	Human	Human	Human
katvalvntine	Human	Human	Human	Human
Nancy1370Nancy	Human	Human	Human	Human
CoolFM_1079	Bot	Bot	Bot	Bot
JerryNengwenani	Human	Human	Human	Human
livvvey	Human	Bot	Human	Human
ffsthirlwall	Bot	Bot	Bot	Bot
zheadioquino06	Bot	Human	Human	Human
VOSTamericas	Bot	Human	Bot	Human
rsm3259	Bot	Bot	Human	Bot
Number Correct	**18**	**20**	**21**	**22**
% Correct	**72%**	**80%**	**84%**	**88%**

VI. END-USER APPLICATION

Our vision for the end-user application for this project is a web browser plug-in that acts as an overlay to Twitter, automatically classifying both user profiles and tweets as belonging to either humans or bots. Instead of overwhelming the user with buttons and alerts, our app would only notify the user if the model had a high confidence that the account in question was a bot. Figure 3 shows a mock-up of the UI for the application and Figure 4 is a logical diagram of the current `check_screenname.py` implementation.

Fig. 3. Mock-up for Spot A Bot application: User profile



There is a 92% chance that this user is a bot

page.is/sue-m
Joined May 2009
Tweet to Sue M.
1,438 Photos and videos

Sue M. @batchild100 · 1h
Enter to win an Amazon Echo Dot giveaway!
Amazon Echo Dot Giveaway | FlashGiveaways.com
Enter today for your chance to win an Amazon Echo Dot giveaway. Echo Dot (2nd Generation) is a hands-free, voice-controlled device that uses Alexa to play music,...
flashgiveaways.com

Sue M. @batchild100 · 8h
Enter to win $100 and more than TWENTY romance ebooks in the Sweet & Steamy Giveaway! rafflecopter.com/rafl/display/c...

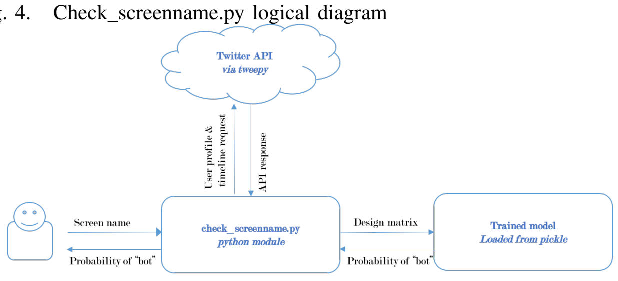
Sue M. @batchild100 · 8h
Win the $10 Amazon GC #Giveaway homeschoolinsights.com/2017/11/10-ama ... homeschoolinsights.com/2017/12/10-ama...

Sue M. Retweeted
Becky @TheMommyIsland · Nov 30
Our VIP Blogger @_AdultColoring is giving away a Coloring Book & $40 Amazon

Pam @PJGurganus Follow
Sandra Watts @saleago Follow
Corey Hutton @CAllenHu... Follow
Find people you know
Import your contacts from Gmail
Connect other address books

#AdviceForSanta
4,998 Tweets
#FaceInterface
#Flynn
Trump's former national security frontman Michael Flynn pleads guilty to lying to the FBI
#TeenVogueSummit

Fig. 4. Check_screenname.py logical diagram



VII. CONCLUSION AND FUTURE WORK

In conclusion, after comparing the performance of three different models (logistic regression, neural network, and gradient-boosted) on the problem of classifying a given Twitter user as "bot" or "human", we found that the gradient-boosted classifier had the best AUPRC and accuracy on highly-confident predictions. Though all three models performed well on the Training and Train-Dev datasets, the maximum F-score attained on the Test dataset with a threshold at 0.5 was 0.661. Our hypothesis that this was due to a distribution mismatch was verified by experimenting with using samples from the Dev dataset to train our model. In addition, we investigated the veracity of the "ground truth" labeling in the Test distribution by running our own human tests.

Looking ahead, we would like to collect our own dataset for further experimentation. Gathering more training examples could improve our distribution mismatch by making the Training distribution more like the real-world; It would also give us more control over how the "bot"/"human" labels were generated. In addition, we could create an English-only tweet corpus, which would enable the use of text-based features (E.g. tweet sentiment as in [5], topic extraction, words used). We would also be able to include friendship and follower relationships between users in the dataset, to exploit bot community features as explored in [7]. Finally, in the future we would like to turn our application prototype into a production-quality web plugin that could provide real-time classification of accounts to Twitter users.

APPENDIX A
TEAM MEMBER CONTRIBUTIONS

Jessica: Wrote python code to combine datasets and generate/standardize design matrices from raw data inputs. Wrote the code to download tweets from the Twitter API for the test distribution, as well as apply the final model to user-inputted screen names. Administrated the human tests for comparison. Collaborated with Sahil on the hyper-parameter tuning and evaluation metric code. Collaborated with Sahil on the milestone, poster and final report.

Sahil: Wrote python code to run the scikit-learn models and generate feature visualizations. Wrote the code to repartition the datasets for testing the distribution mismatch. Collaborated with Jessica on the hyper-parameter tuning and evaluation metric code. Collaborated with Jessica on the milestone, poster and final report.

ACKNOWLEDGMENT

The authors would like to thank the CS229 teaching assistant staff for their guidance and direction during class office hours, as well as professors Ng and Boneh for their instruction this quarter.

REFERENCES

[1] R. Gorwa, "Twitter has a serious bot problem, and wikipedia might have the solution," *Quartz Media*, October 23 2017. [Online]. Available: https://qz.com/1108092/twitter-has-a-serious-bot-problem-and-wikipedia-might-have-the-solution/

[2] K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu, "Fake news detection on social media: A data mining perspective," *CoRR*, vol. abs/1708.01967, 2017. [Online]. Available: http://arxiv.org/abs/1708.01967

[3] C. Yang, R. C. Harkreader, and G. Gu, *Die Free or Live Hard? Empirical Evaluation and New Design for Fighting Evolving Twitter Spammers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 318–337. [Online]. Available: https://doi.org/10.1007/978-3-642-23644-01_7

[4] S. Cresci, R. D. Pietro, M. Petrocchi, A. Spognardi, and M. Tesconi, "Fame for sale: efficient detection of fake twitter followers," *CoRR*, vol. abs/1509.04098, 2015. [Online]. Available: http://arxiv.org/abs/1509.04098

[5] ——, "The paradigm-shift of social spambots: Evidence, theories, and tools for the arms race," *CoRR*, vol. abs/1701.03017, 2017. [Online]. Available: http://arxiv.org/abs/1701.03017

[6] O. Varol, E. Ferrara, C. A. Davis, F. Menczer, and A. Flammini, "Online human-bot interactions: Detection, estimation, and characterization," *CoRR*, vol. abs/1703.03107, 2017. [Online]. Available: http://arxiv.org/abs/1703.03107

[7] E. Tacchini, G. Ballarin, M. L. D. Vedova, S. Moret, and L. de Alfaro, "Some like it hoax: Automated fake news detection in social networks," *CoRR*, vol. abs/1704.07506, 2017. [Online]. Available: http://arxiv.org/abs/1704.07506

[8] K. Lee, B. D. Eoff, and J. Caverlee, "Seven months with the devils: a long-term study of content polluters on twitter," in *In AAAI Intl Conference on Weblogs and Social Media (ICWSM*, 2011.

[9] O. Varol. (2017) Bot repository datasets. [Online]. Available: https://botometer.iuni.iu.edu/bot-repository/datasets.html

[10] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *J. Mach. Learn. Res.*, vol. 9, pp. 1871–1874, Jun. 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1390681.1442794

[11] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001. [Online]. Available: http://www.jstor.org/stable/2699986

[12] Y. Xiao, Z. Wei, and Z. Wang, "A limited memory bfgs-type method for large-scale unconstrained optimization," *Computers Mathematics with Applications*, vol. 56, no. 4, pp. 1001 – 1009, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0898122108001028