Sunjoo Lee
Advanced Machine Learning for Public Policy

Machine Learning Framework for Twitter User Classification

1. Purpose

The purpose of the project is to build a general-purpose machine learning framework using NLP methods to classify Twitter users.[1] This framework can then be applied to any classification task of Twitter users where large amounts of labeled data is available. For this project, I will be building the framework and testing it on the prediction of the political affiliation of Twitter users. In future use of the framework, it can be utilized on other data to predict propaganda tweeters, jihadist groups, etc.

The Github repo for all code can be found here, and the presentation can be accessed here.

2. Related Work

The project will largely be modeled on the work of Pennacchiotti and Popescu (Pennacchiotti & Popescu, 2011), where the authors extracted components of tweets to create features, and then applied machine learning algorithms to predict user characteristics (political affiliation, ethnicity, Starbuck fans). They engineered four general categories of features: 1) user profile characteristics (length of user name, different capitalization forms, use of avatar picture, etc.), 2) tweeting behavior (number of tweets posted, fraction of tweets that are retweets, average number of hashtags and URLs per tweet, etc.), 3) social network characteristics (number of friends in the network, etc.), and 4) linguistic content (features from the text of the tweet). It was found in the study, unsurprisingly, that the text of the tweet was most useful in user classification, and that the other features did not add much to the models. Therefore, I will only focus on the text-based features in this project.

Note that this framework will only work in text-heavy applications; it will probably not be useful in discriminating bots from real Twitter users. Bots are usually detected through the user's profile features (random numbers and letters in usernames, etc.) and tweeting behavior (constant retweets with no or little added content, etc.), rather than the tweet's text. In fact, the R library *botornot*, created by Mike Kearney to detect bots, relies mostly on simple features extracted from the user account to achieve > 90% accuracy.[2] Andy Patel's implementation does more text processing, and is able to detect bots that have content (usually trying the sell things).[3] In general, however, the majority of bots are automated to just retweet tweets, and this framework is inappropriate for detecting such bots (at the very least because reliable data is not available, partly because Twitter has started to detect and delete bot accounts).

3. Data

The framework is supposed to work on any text-heavy (as opposed to network-heavy) Twitter data, with the caveat that labeled data is crucial to modeling/learning. Labeled Twitter datasets are very hard to find due to Twitter's policy against posting data (only the tweet ID can be shared, and the use of the Twitter API is encouraged), and researchers often devote months to data collection (usually through known user accounts or hashtags) or work with Twitter directly.

Therefore, to test the validity of the framework, I will be working with one of the datasets that are publicly available – tweets related to the 2016 election – to predict users' political affiliation. More specifically,

---

[1] The original focus on emojis and sentiments/emotions was shifted towards a more general focus on the text of the tweet. It was determined after a literature review of emoji usage in tweets that a meaningful project could not be done given the constraints (time, labeled data, etc.).

[2] Original code: https://github.com/r0zetta/pronbot_search

[3] Original code: https://github.com/r0zetta/pronbot_search/tree/master/results

the data includes tweets from early July to early November of the Democratic candidates' timeline (timkaine, SenSanders, BernieSanders, MartinOMalley, and HillaryClinton), the Democratic Convention (found through a search of the terms), the Democratic Party Timelines (SenateDems, HouseDemocrats, TheDemocrats), and the Republican Party equivalents (SenateGOP, HouseGOP, GOP, realDonaldTrump, GovPenceIN, tedcruz, etc.).[4] This amounted to approximately 15 million tweets, from which I sampled around 50,000 for training and testing. The tweet IDs were revived to full tweets using the Hydrator app.[5]

4. Feature Generation

The Twitter data was processed using twitter-text-python (ttp), which extracts usernames (for retweets, replies, and mentions), hashtags, and urls from the body of the text.[6] The usernames and urls were deleted from the body of the text for better feature generation from the text, while the hashtags were saved for its own feature generation. The texts and hashtags were then aggregated by user id, which led to about 28,000 unique users in the sample.

As mentioned in Section 3, only the text of the tweet was used for feature generation. The training-testing data were split 70%-30%, and all features were modeled on the training data and the same features were later created in the testing data. There are three general types of text features in the machine learning framework.

A. Proto-words and hashtags

The first type of features are proto-typical words and hashtags. The idea is that, instead of using the entire set of words as features, only the "typical" words for each label are used. Here, given *n* classes, each class c_i is represented by a set of users S_i. Each word *w* used by at least one of the users in S_i is assigned a probability score for the class as follows in (1).[7] For each of the top *k* proto-typical words for each class that become features (*wp*), the user *u*'s value is calculated as follows in (2).[8] The user is also assigned a value for an aggregate feature for each class.[9] The same was applied to the hashtags.

$$proto(w, c_i) = \frac{|w, S_i|}{\sum_{j=1}^{n} |w, S_j|} \quad (1) \qquad f\_proto\_user(u) = \frac{|u, wp|}{\sum_{w \in W_u} |u, w|} \quad (2)$$

I created a data set with a small number of features, 'proto_small', which included 100 proto-words (minimum word count of 5) and 50 proto-hashtags (minimum word count of 3) per label. I then created a larger dataset, 'proto_large', which included 300 proto-words and 100 proto-hashtags per label (same minimums). Some of the top 300 prototypical words and 100 prototypical hashtags per label included those in Table 1. It is clear that Democrats and Republicans tend to use different words, many of which we tend to associate with each party (though some unusual ones as well).

DEM_WORDS	REP_WORDS	DEM_HASHTAGS	REP_HASHTAGS
'grit',	'tuba',	'demsinphilly',	'michelleobama',
'cuomo',	'leopard',	'demconvention',	'rncincl',
'ads',	'bankruptcy',	'demexit',	'election2016',

[4] Original data: https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/PDI7IN
[5] Download here: https://github.com/DocNow/hydrator
[6] Original code: https://github.com/edburnett/twitter-text-python
[7] The number of times the word w is used by all users in the class divided by the number of times the total number of times the word w is used.
[8] The number of times the word w is used by the user u divided by the set of all words used by u.
[9] The number of times a proto-typical word is used by the user u divided by the set of all words used by u.

'atheist',	'hatefest',	'bernie',	'uslatino',
'trustworthy',	'ein',	'dnc',	'infowars',
'kareem',	'presidencia',	'dncinphl',	'foxnews',
'attn',	'djt',	'clintoncash',	'markburns',
'israeli',	'blah',	'dem',	'outnumbered',
'hacked',	'corrected',	'de',	'rudy',
'incarceration',	'approved',	'strongertogether',	'tcot',
'till',	'jailing',	'feelthebern',	'nevertrumporhillary',
'goldman',	'mccaul',	'philadelphia',	'flotus',
'conviction',	'standards',	'berniesanders',	'tedcruz',
'persistent',	'description',	'clinton',	'beckywiththeborrowedspeech',
'contested',	'copying',	'dnc2016',	'mylittlepony',
'yorker',	'royale',	'bernieorbust',	'obamacare',
'implicated',	'guiliani',	'vote',	'ccot',
'teenager',	'marla',	'russia',	'gop4gary',
'warned',	'suck',	'gopwomen',	'fail',
'soldier',	'estranged',	'dems',	'msnbc',
'pobrecito',	'darkness',	'hillarysamerica',	'gopdebate',
'pin',	'monitoring',	'rollcallvote',	'nevertrumppence',
'fillers',	'nigel',	'jillnothill',	'scotus',
'karla',	'blowing',	'unity',	'gopc',
'hacking',	'outbreak',	'blm',	'rncatcle',
'masses',	'tower',	'sheswithus',	'carly2016',
'ruin',	'antigay',		
'indie',	'vomiting',		
'demi',	'dumpster',		
'jeers',	'football',		

Table 1: Prototypical words and hashtags, by label

B. Topic Modeling

The second type of features are the topics of the tweets. The original authors conducted two types of topic modeling, one generic (from a sample of millions of random tweets) and one domain-specific (modeled only on the labeled domain-specific tweets). I implemented only on the latter because they were found to be more useful and because of the lack of access to large amounts of random Twitter data.

The texts of the tweets were tokenized using nltk, then topics were modeled on the training data using gensim's implementation of the Latent Dirichlet Allocation model proposed by David Blei (Blei, 2012). I used 50 topics in the 'topic_small' dataframe and 100 topics in the 'topic_large' dataframe. Each user's score on each topic was the proportion of the text devoted to the topic.

C. Sentiment Analysis

The third type of features is related to the sentiments expressed in the text. The tools for sentiment analysis conducted by the original authors was not usable, but much research has been done on the topic (Tang, Qin, & Liu, Deep learning for sentiment analysis: successful approaches and future challenges, 2015). I identified TS-Lex, a large-scale Twitter-specific sentiment lexicon that was trained on millions of tweets (Tang, Wei, Zhou, Qin, & Liu, 2014), as a suitable lexicon for sentiment analysis.

I first parsed the text of each tweet into subject-verb-object triples using the Stanford parser. Here, if the subject or object was more than one word long, the last word was used since it is most likely to be the noun in the phrase. This was a computationally expensive task that required parallel processing using 16 vCPUs on Amazon Web Services. For each subject and object term in each label, I found the average sentiment towards the term by averaging the sentiment (using TS-Lex) of the m number of window words preceding and succeeding the term. The top k most positive subjects and objects (each) and the bottom

k most negative subjects and objects (each) per label were featurized. Words appearing as positive (or negative) in both labels were excluded from the feature set. In the small data frame, 'sent_small', k equaled 20, and in the large data frame, 'sent_large', k equaled 50 (window of 4 for both).

Each user then received a score that was the average sentiment of the 4 words preceding and succeeding each feature term used. Each user also received an aggregate sentiment score that was the average of the sentiment towards all the terms in each of the label-sentiment groups: Democrat-Negative, Democrat-Positive, Republican-Negative, Republican-Positive.

Some of the top sentiment-charged subjects/object words per label are listed in Table 2. It is clear that Democrats and Republicans feel strongly about different things, many of which we tend to associate with each party. Some terms that Republicans feel most negative about include 'hillaryclinton', 'journalists', 'refugees', 'immigrants', but also 'gingrich', 'mother', and 'businesses'. Some terms that Democrats feel negative about include 'rnc', 'power', 'rules', but also 'professionals', and 'email'.

DEM NEGATIVE	DEM POSITIVE	REP NEGATIVE	REP POSITIVE
'year',	'you…',	'place',	'natives',
'power',	'topics',	'ass',	'!',
'seats',	'us—it',	'mother',	'part',
'experience',	'fun',	'show',	'event',
'rules',	'kick-ass',	'floor',	'fightback',
'tweet',	'yorker',	'hillaryclinton',	'republican',
'rnc',	'friend',	'immigrants',	'conscience',
'speeches',	'flotus',	'course',	'ally',
'flags',	'message',	'businesses',	'election',
't',	'choice',	'h…',	'honor',
'violence',	'family',	'journalists',	'wife',
'press',	'life',	'rally',	'leader',
'professionals',	'…',	'refugees',	'side',
'themselves',	'border',	'chat',	'tonight',
'email',	'http…',	'surgery',	'american',
'morning',	'con',	'lights',	'nomination',
'crisis',	'coverage',	'racism',	'level',
'role',	'moment',	'gingrich',	'safe',
'teenager',	'poll',	'woman',	'demeanor',
'speaker',	'bloomberg',	'policies',	'mongols',

Table 2. Most positive and negative subjects/objects, per label

D. All Features

A final data set with all types of features was created. This was done in conjunction with performance on classification (more details in the next section). The final data set had 982 features, including 300 proto words and 50 proto hashtags (from each label), 50 topics, and 20 sentiment words (from each label, positive and negative each, subject and object each). The final feature generation was also a computation-intensive task that required AWS computing.

5. Classification

I tested machine learning classifiers in stages, with the assumption that classifiers that perform well in the basic form (default classifier settings on basic features) will also perform well in the sophisticated form (tuned classifier parameters on many features), and vise-versa. This may not always be the case, but is probably effective and more efficient than iterating over every combination of parameters and features, especially considering limited the time and computation resources.

A. Initial Classifier Testing

I tested a set of classifiers with default settings on a basic set of features (100 proto-words, 50 proto-hashtags per label, 20 topics). The original authors used gradient boosted decision trees to achieve about 89% accuracy on the full set of features (including profile, behavior, network features) and 77% accuracy on only the linguistic features. I therefore chose to focus heavily on ensemble classifiers.

As seen in Table 3, all the ensemble classifiers performed decently well and trained very quickly. This is probably because they are less likely to overfit on the data, and each weak classifier can be fit quickly. Logistic Regression and the Decision Tree also performed reasonably well, while the Naïve Bayes and SVM did not. Poor performance on the Naïve Bayes classifier indicates that the features are not independent (which is reasonable, given the data). Performance on the SVM may be improved with parameter tuning, but would be very time-intensive. The Decision Tree's performance was enhanced by the ensemble methods by 3-6 percentage points. Therefore, the ensemble classifiers and Logistic Regression were moved on to the parameter tuning stage, and the SVM, Naïve Bayes, and Decision Tree were dropped from testing.

Classifier	**Classification Score**[10]	**Run Time (in minutes)**
Random Forest	0.8782	0.08
Gradient Boosting	0.8566	0.69
AdaBoost, DT	0.8477	0.27
Bagging, DT	0.8481	0.02
Naive Bayes	0.5234	0.01
Logistic Reg	0.8460	2.78
SVM	0.5236	92.96
Decision Tree	0.8187	0.02

Table 3. Score vs. training time, by classifier

B. Parameter Tuning

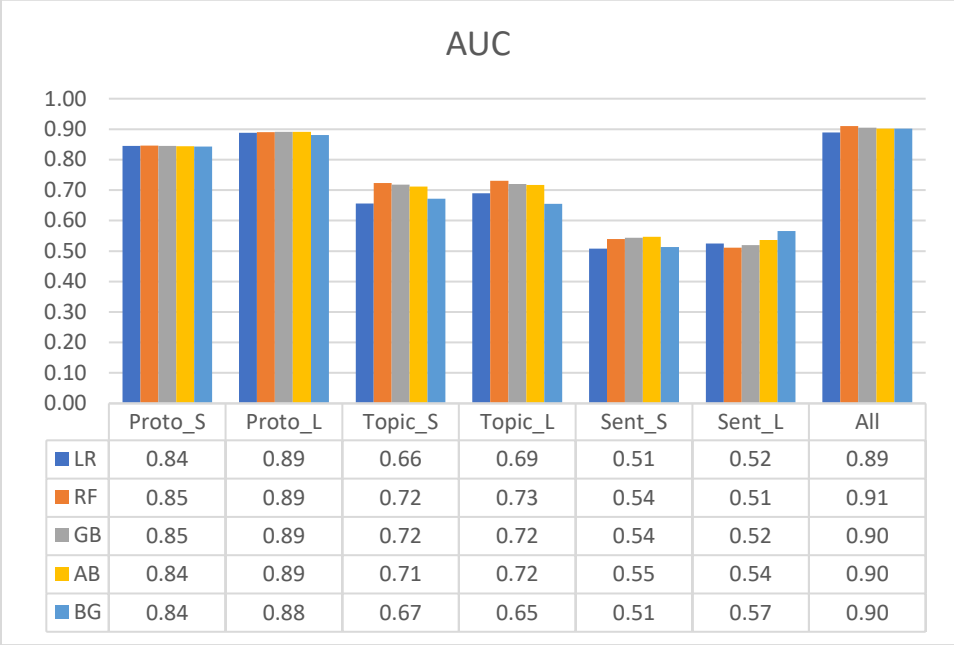
For the Logistic Regression classifier, I tried different combinations of penalties (L1, L2) and C-values. However, different levels of regularization had little effect on the performance (I settled on L1 penalty and 1e5 C-value, which performed marginally better). For the ensemble classifiers, I tried different numbers of estimators (50 ~ 500) and different maximum depths (5~20). For all the ensemble classifiers, having more trees/weak classifiers beyond 50~100 did not affect the performance very much. However, increasing the maximum depth of each weak classifier did enhance performance, suggesting that each weak classifier should have some complexity for the ensemble to perform well. However, increasing the maximum depth increased the time to fit the model. Thus, for the ensemble classifiers, I set the number

[10] This value is from `.score` attribute in sklearn classifiers. It refers to different scores in different classifiers, but is roughly comparable.

of estimators to 100, and the maximum depth to 10. I repeated these tests on different datasets (proto-words small and large, topics small and large) to largely similar findings and an added measure of verification.

C. Best Performing Classifier

I found all classifiers to perform uniformly well with "strong" features (proto-words and hashtags), and different classifiers to show somewhat varying performance with "weak" features (topics, sentiments), as shown in Figure 1. I chose the Random Forest as the "best" performing classifier, since it performed the best with weak features. Thus, I will only focus on the results of the Random Forest classifier in the Analysis section.



	Proto_S	Proto_L	Topic_S	Topic_L	Sent_S	Sent_L	All
LR	0.84	0.89	0.66	0.69	0.51	0.52	0.89
RF	0.85	0.89	0.72	0.73	0.54	0.51	0.91
GB	0.85	0.89	0.72	0.72	0.54	0.52	0.90
AB	0.84	0.89	0.71	0.72	0.55	0.54	0.90
BG	0.84	0.88	0.67	0.65	0.51	0.57	0.90

Figure 1. AUC of different classifiers, per feature-group type

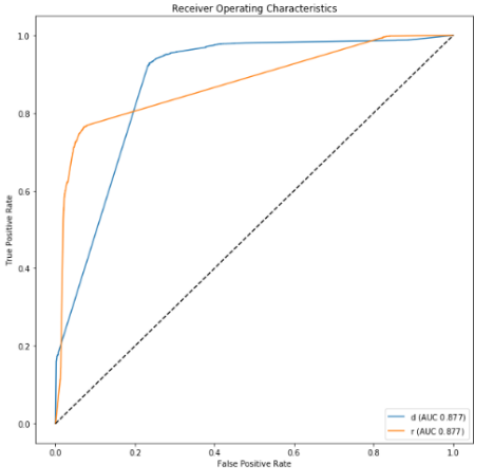
6. Analysis

I assessed the models using Receiver Operator Characteristics - Area Under the Curve. This is because getting both the classes right is important for the project, and AUC plots the false positive rate against the true positive rate. Particularly in the case of binary classes, this seems to be a comprehensive way of measuring performance.

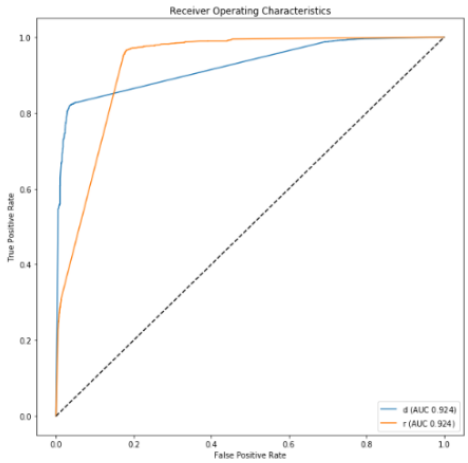
A. Feature Groups

As mentioned above, I created seven data frames in total, two each (small and large number of features) for each feature group separately, and one with all three types of feature groups. The purpose was to test which feature groups contributed to performance.

The Random Forest classifier performed very well on just the prototypical word and hashtag features. Further, performance was enhanced slightly by increasing the number of feature words and hashtags, as seen in Figure 2. Together, this suggest that proto-words and hashtags are "strong" features that carry a lot of weight in the classification.



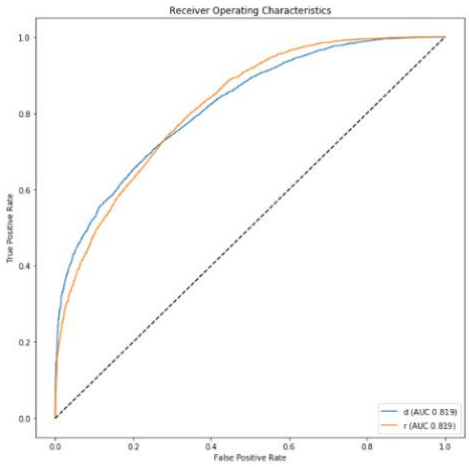
	Words	Hashtags	AUC
Small	100	50	0.85



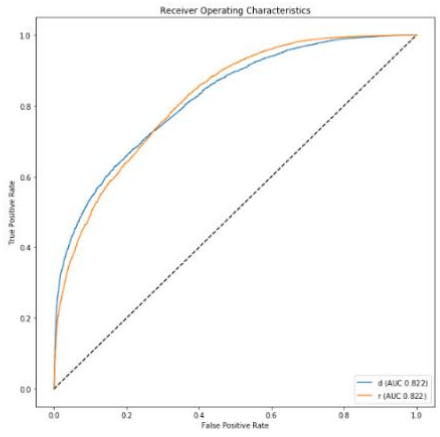
	Words	Hashtags	AUC
Large	300	100	0.89

Figure 2. AUC for small and large number of proto-words and hashtags

This was not the case for the models run on just the topic features. Figure 3 shows that topics were a "weaker" predictor of labels than prototypical words and hashtags. Further, there was no improvement in performance when the number of topics was increased, indicating that the granularity of topics beyond a certain point probably does not add to the model.



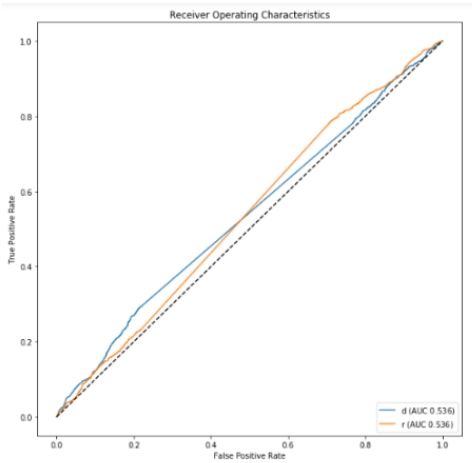
	Topics	AUC
Small	50	0.73



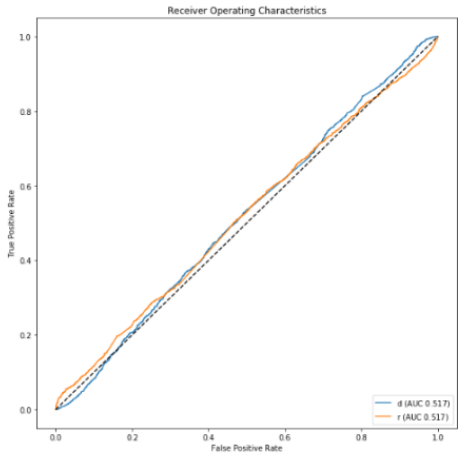
	Topics	AUC
Large	100	0.73

Figure 3. AUC for small and large number topics

The same can be said for the models run on just the sentiment features. Figure 4 shows that sentiments were very "weak" predictors of labels, performing only slightly better than random guessing. Further, performance actually decreased slightly when the number of sentiments was increased, though this may be due to noise or randomness in the classifier. The sentiment features used (Table 2) were in accordance with our common-sense notion of the two labels groups, so it is unclear why the models performed poorly.



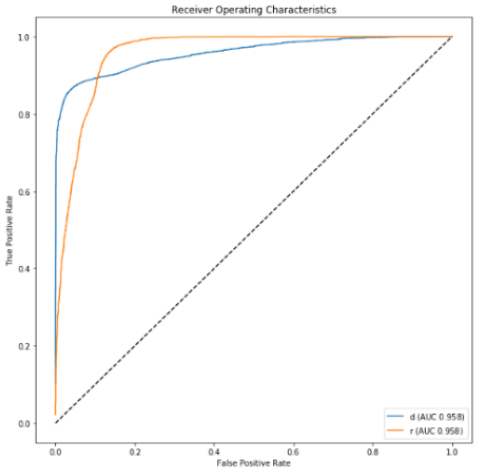
	Sentiments	AUC
Small	20	0.54



	Sentiments	AUC
Large	50	0.52

Figure 4. AUC for small and large number sentiment features

The final data frame included features from all three feature groups. I included the large number of words and hashtags, since they performed better than the small, and the small number of topics and sentiments, since the larger number of features made no difference. In the final data frame, the AUC improved from the large proto features (.89) by 3 percentage points (.91), as shown in Figure 5. The prototypical words and hashtags are probably still carrying a lot of weight in the classification (explored more later). However, the false positive rate between the two labels became more evenly balanced with the addition of the two other types of features, as seen in Figure 6. Thus, despite only the small increase in performance, it is still worthwhile to include all features in the model. The final analysis will be conducted on this dataframe.



	Words	Hashtags	Topics	Sentiments	AUC
Number	300	100	50	20	0.91

Figure 5. AUC for all feature groups

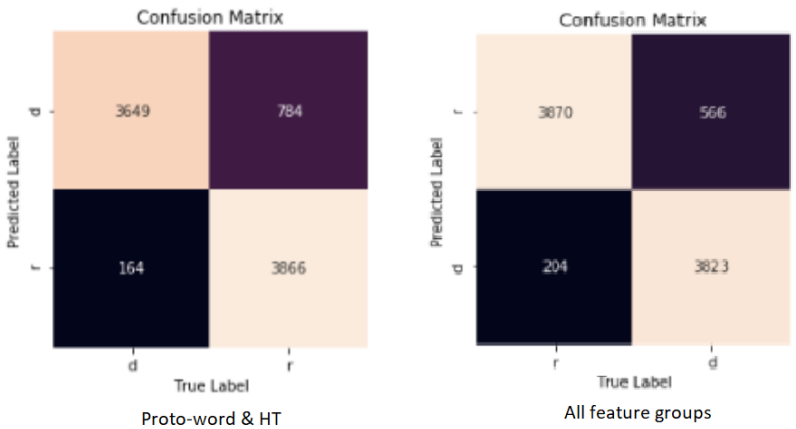
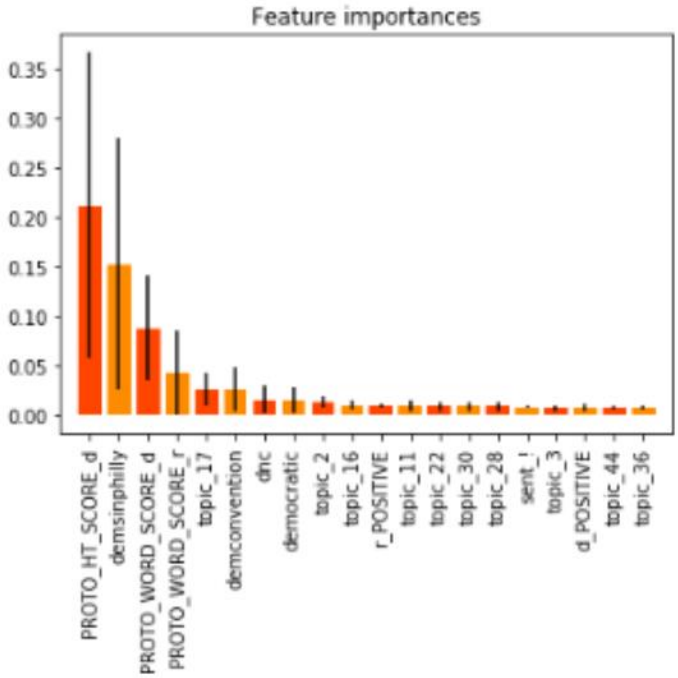


Figure 6. Confusion matrix, proto-words v. all features

B. Important Features

As it can be seen in Figure 7, the most important features were the prototypical words and hashtags. In particular, the aggregate hashtag score for Democrats (how likely a user uses any prototypical Democratic hashtags) was the most discriminatory feature. In addition, other aggregate prototypical words and hashtags and even particular words were identified as important features. This indicates that Democrats and Republicans use very different, distinctive vocabulary and hashtags.

Two topics were in the top ten ranking features, though it is difficult to construe a coherent topic from the words they generated. The aggregate positive sentiment score for Democrats and Republicans (average sentiment for words that Democrats/Republicans feel positive about) were among the top 20 features. This seems to indicate that there are subtle differences in the topics that engage and move Democrats and Republicans, which play an auxiliary role in classification.




```
Feature ranking:
1. feature PROTO_HT_SCORE_d (0.211392)
2. feature demsinphilly (0.152232)
3. feature PROTO_WORD_SCORE_d (0.086908)
4. feature PROTO_WORD_SCORE_r (0.042086)
5. feature topic_17 (0.025700)
6. feature demconvention (0.025660)
7. feature dnc (0.015222)
8. feature democratic (0.014119)
9. feature topic_2 (0.012908)
10. feature topic_16 (0.009833)
11. feature r_POSITIVE (0.009521)
12. feature topic_11 (0.008754)
13. feature topic_22 (0.008548)
14. feature topic_30 (0.008221)
15. feature topic_28 (0.008152)
16. feature sent_! (0.007921)
17. feature topic_3 (0.007058)
18. feature d_POSITIVE (0.006928)
19. feature topic_44 (0.006883)
20. feature topic_36 (0.006876)
```

Figure 7 and Table 4. Top ranking features

C. Resource Allocation

It was found that feature generation was a time and computation-expensive task, much more so than classification. However, as verified above in multiple ways, certain features (particularly the prototypical words and hashtags) were found to be much more important than others. In fact, given strong features, the specific classifier and parameter used did not matter very much in performance. Therefore, in this case and possibly in other similar tasks, extra resources are better spent on feature engineering and combination variations, rather than on parameter tuning for many classifiers. Possible features include proto-typical bigrams or n-grams, separate topics for each label, and automated features generated by neural networks.

7. Conclusion

The machine learning framework for Twitter user classification performed decently well on classifying Democrats and Republicans. It was found that Democrats and Republicans largely use different hashtags and words, which was very important in the performance of the classification task. Most interestingly, this seems to confirm the social reality of segregated political echo chambers, which has become more evident after the election. Further, each party is sentimentally charged by different subjects, but this seems to have a much more subtle effect on classification. Ensemble classifiers using a succession of weak decision trees were very efficient; performance was very high in comparison to the training time. Given more resources, it would be worthwhile to spend time and computation power on more sophisticated and fine-tuned feature engineering.

References

Pennacchiotti , M., & Popescu, A.-M. (2011). A Machine Learning Approach to Twitter User Classification. *Proceedings of the Fifth International AAAI Conference on Weblogs and Social Media.* Retrieved from https://webpages.uncc.edu/anraja/courses/SMS/SMSBib/2886-14198-1-PB.pdf

Blei, D. M. (2012). Probabilistic Topic Models. *Communications of the ACM*, 77-84.

Tang, D., Qin, B., & Liu, T. (2015). Deep learning for sentiment analysis: successful approaches and future challenges. *WIREs Data Mining and Knowledge Discovery* , 292 - 303.

Tang, D., Wei, F., Zhou, M., Qin, B., & Liu, T. (2014). Building Large-Scale Twitter-Specific Sentiment Lexicon : A Representation Learning Approach. *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics*, (pp. 172–182).