Machine Learning

Project Report

Yingqi Huang yh1990

Xiaoyi Zhang xz1618

1. INTRODUCTION

Our project comes from a problem of twitter: User vs. Bots, a user recognition problem aimed to distinguish users from Twitterbots. A Twitterbot is a bot program used to produce automated posts on the Twitter microblogging service, or to automatically follow Twitter users. Twitterbots come in various forms. These automatic tweets are often seen as fun or silly. For example, many serve as spam, enticing clicks on promotional links. Others post replies or automatically "retweet" in response to tweets that include a certain word or phrase. However, some of the Twitterbots are harmful. For example, some of the twitterbots are deployed to slander the president candidate during the presidential election. So, it's meaningful to distinguish the users from Twitterbots. In this project, we try to build a model to classify bots to solve this problem, and give prediction of whether an account is a *user* (labeled as 0) or a *bot* (labeled as 1).

2. DATA

We find all the data set from Twitter API Console of apigee, which provides twitter API management and predictive analytics. The dataset collected by us contains 1056 bots account and 1176 nonbots account, for each data, it has 20 attributes and the last attribute bot to define whether it is a user or a bot. The attribute includes Id, Id_Str, screen_name, location, description, url, followers_count, friends_count, listedcount, created_at, favorite_count, verified, status_count, lang, status, default_profile, default_profile_image, has_extended_profile and name.

3. DATA PROCESSING

We first load the dataset bots and nonbots which are pulled from twitter. Then we merge the bots and nonbots files and stored it as the feature matrix m. Features are also known as predictors, inputs or attributes. We also merge the bots and nonbots labels and store it as response vector n. Responses are also known as target, label, or output. In order to build a model, the features and labels must be numeric, and every observation must have the same features in the same order. So we make all the features and labels numeric.

Now we begin to analyze all the 19 attributes to determine which attributes can be helpful in our experiment. Firsly, we replace all the NaN as 0. Then, the attributes like id, id_str are not null, are totally different from each other even cannot be classified into limited classes and they are useless in our analysis. So, we set all of them as 1.

Unfortunately, the rest of the attributes are still cannot be directly used in our experiment, we have to do something to change the form of these so that we can use them. We notice that the attributes which have keywords 'count', they may have up to thousands possible values. By observing the dataset, we find a better way to define the classes. For example, the attribute followers_count, we can find that when the number of accounts is less than 16, the possibility of the account is bot is much larger than the accounts which have more than 16 followers. So we define the the value that less than or equal to 16 as '0', the

value more than 16 as ‘1’. Similarly, for attribute followers_count, we set the value less than or equal to 35 as 0, otherwise, 1. We do the same thing to attribute like listed_count, favorite_count and statues_count. Also, we deal with the attribute 'created_at' as year-2000 because it's likely that the account is a bot if it is created before 2000. Moreover, for attribute like 'screen_name' and 'description', it's more likely to be a robot if there is a 'bot' in it. So, we set the value of these attributes as 1 if they contain keyword ‘bot’ in them.

The attributes like location, and url, they are consist of words, the only difference of them is null or not null. If the value is null, we will define it as 0. If the value is not null, we will define it as 1. Because in python, the py.to_numeric cannot turn text into digit directly, we use this function to turn all the attribute into numeric value. The test will be converted into NaN and then we replace all the NaN as 1. For the left attributes with only two possible values ‘True’ and ‘False’, we do not have to change the form.

4. AlGORITHM

4.1 BernoulliNB

BernoulliNB is actually a naïve Bayes classifier for multivariate Bernoulli models

Like MultinomialNB, this classifier is suitable for discrete data. The difference is that while MultinomialNB works with occurrence counts, BernoulliNB is designed for binary/boolean features.

In the multivariate Bernoulli event model, features are independent booleans (binary variables) describing inputs. Like the multinomial model, this model is popular for document classification tasks, where binary term occurrence features are used rather than term frequencies. If x_i is a boolean expressing the occurrence or absence of the i'th term from the vocabulary, then the likelihood of a document given a class C_k is given by

$$p(\mathbf{x} \mid C_k) = \prod_{i=1}^{n} p_{ki}^{x_i} (1 - p_{ki})^{(1-x_i)}$$

where p_{ki} is the probability of class C_k generating the term x_i. This event model is especially popular for classifying short texts. It has the benefit of explicitly modelling the absence of terms.

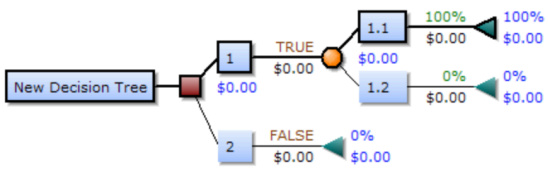
4.2 Decision Tree

A decision tree is a flowchart-like structure in which each internal node represents a "test" on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules.

In decision analysis, a decision tree and the closely related influence diagram are used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated.

A decision tree consists of three types of nodes:

1. Decision nodes – typically represented by squares
2. Chance nodes – typically represented by circles
3. End nodes – typically represented by triangles



It has many specific decision-tree algorithm, including ID3, C4.5, CART, CHAID, MARS etc.

The basic idea of D3 and C4.5 use information gain $I_E(i)$ to weigh the gain.

$$I_E(i) = -\sum_{j=1}^{m} f(i,j) \log_2 f(i,j)$$

In CART , we use Gini coefficient I_G to measure the gain.

4.3 GradientBoosting

Gradient boosting is a machine learning technique for regression and classification problems, which produce a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

Example:

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$ a differentiable loss function $L(y, F(x))$ number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$

2. For m = 1 to M:

1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)},$$

for i = 1 … m

2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i . r_{im})\}_{i=1}^{m}$.

3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$$

4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

3. Output $F_M(x)$

4.4 RandomForest

Random forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set

Algorithm:

Preliminaries**:** decision tree learning above

In particular, trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets

Tree bagging:

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set X = x_1, ..., x_n with responses Y = y_1, ..., y_n, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For b = 1, ..., B:

1. Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
2. Train a classification or regression tree f_b on X_b, Y_b.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x')$$

or by taking the majority vote in the case of classification trees.

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias.

4.5 Logistic regression

Logistic regression is a regression model where the output can take only two values, "0" and "1", which represent outcomes such as pass/fail, win/lose, alive/dead or healthy/sick.

Logistic function: $f(z) = 1/(1 + e^{-z})$

$$P(y = 1|x) = \frac{1}{1 + e^{-z}}, \qquad P(y = 0|x) = \frac{e^{-z}}{1 + e^{-z}}$$

Consider probabilistic model:

$$z = w_0 + \sum_{j=1}^{k} w_k x_k$$

Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function, which is the cumulative logistic distribution. Thus, it treats the same set of problems as probit regression using similar techniques

5. RESULT

5.1 Training Result

We evaluate the results in the following 6 aspects:

Accuracy:

The accuracy of a measurement system is the degree of closeness of measurements of a quantity to that quantity's true value

Precision: *R=TP/(TP+FP)*

Precision (also called positive predictive value) is the fraction of relevant instances among the retrieved instances

Recall: *R=TP/(TP+FN)*

recall (also known as sensitivity) is the fraction of relevant instances that have been retrieved over the total amount of relevant instances.

F1:

F1 score is a measure of a test's accuracy. It considers both the precision *p* and the recall *r* of the test to compute the score: *p* is the number of correct positive

results divided by the number of all positive results, and *r* is the number of correct positive results divided by the number of positive results that should have been returned. The F_1 score is the harmonic average of the precision and recall, where an F_1 score reaches its best value at 1 (perfect precision and recall) and worst at 0.

Auc score:

Area under the curve (AUC) is the area under the curve (mathematically known as the definite integral) in a plot of drug concentration in blood plasma vs. time.

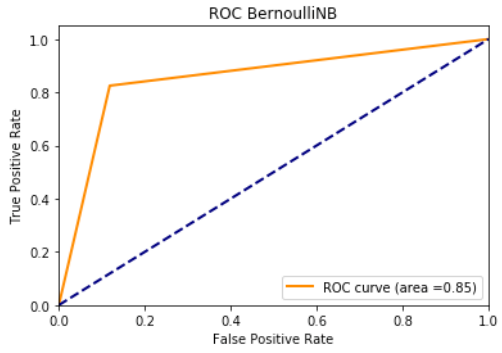
ROC cureve:

ROC curve is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

These are the results of the training data:

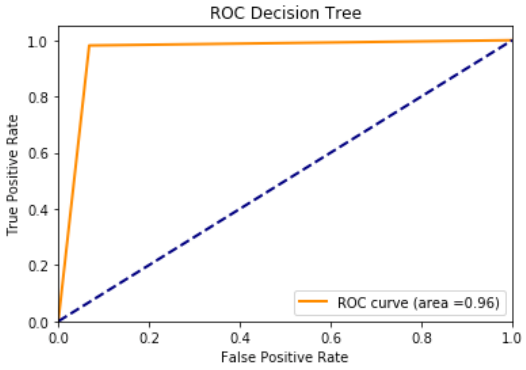
(1) BernoulliNB

```
 BernoulliNB
Accuracy  = 0.852661064426
Precision = 0.878077373974
Recall    = 0.824889867841
F1        = 0.850653038047
Auc score = 0.853151889451
ROC curve of BernoulliNB:
```



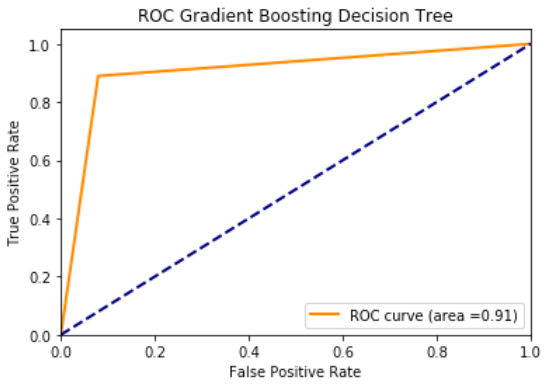
(2) Decision Tree

```
 Decision Tree
Accuracy  = 0.95406162465
Precision = 0.921453692849
Recall    = 0.98127340824
F1        = 0.950423216445
Auc score = 0.956591988673
ROC curve of Decision Tree:
```



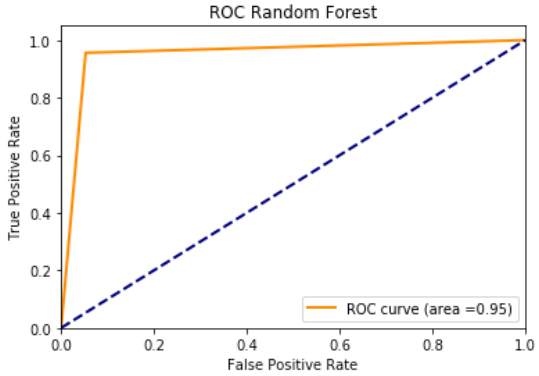
(3) Gradient Boosting Decision Tree

```
 Gradient Boosting Decision Tree
Accuracy  = 0.905322128852
Precision = 0.915592028136
Recall    = 0.889521640091
F1        = 0.902368573079
Auc score = 0.905069530079
ROC curve of Gradient Boosting Decision Tree:
```



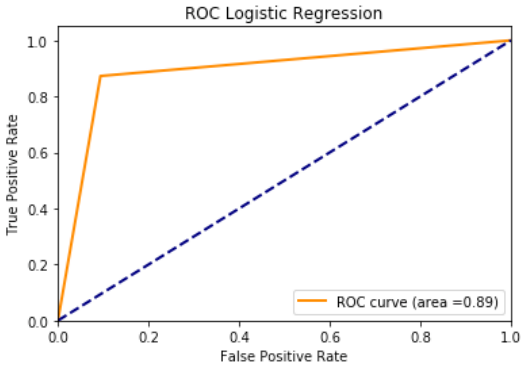
(4) Random Forest

```
 Random Forest
Accuracy  = 0.951260504202
Precision = 0.941383352872
Recall    = 0.955952380952
F1        = 0.948611931483
Auc score = 0.951521164021
ROC curve of Random Forest:
```



(5) Logistic Regression

```
 Logistic Regression
Accuracy  = 0.889635854342
Precision = 0.900351699883
Recall    = 0.872727272727
F1        = 0.886324293133
Auc score = 0.889402310397
ROC curve of Logistic Regression:
```



5.2 Cross Validation

We keep 20% of the data to test the performance of the learned model. Then, we compute the mean of the values of the cross validation in order to compare the algorithms visually.

The following is the result for the test data:

(1) BernoulliNB

```
BNaccuracy  = 0.847090469917
BNPrecision = 0.822672986815
BNRecall    = 0.846666666667
BNF1        = 0.83343869309
BNAuc score = 0.911489285714
```

(2) Decision Tree

```
DTaccuracy  = 0.845065876153
DTPrecision = 0.861806236606
DTRecall    = 0.812619047619
DTF1        = 0.82866793015
DTAuc score = 0.86485952381
```

(3) Gradient Boosting Decision Tree

```
GBDTaccuracy  = 0.863098375055
GBDTPrecision = 0.843190362269
GBDTRecall    = 0.857380952381
GBDTF1        = 0.852303172509
GBDTAuc score = 0.952948809524
```

(4) Random Forest

```
RFaccuracy  = 0.867841458059
RFPrecision = 0.84739420521
RFRecall    = 0.832619047619
RFF1        = 0.848399443866
RFAuc score = 0.945501190476
```

(5) Logistic Regression

```
LRaccuracy  = 0.840421607378
LRPrecision = 0.808677934956
LRRecall    = 0.856904761905
LRF1        = 0.830101437606
LRAuc score = 0.937389285714
```

6. CONCLUSION

As we can see above, Auc score is usually between 0 ~ 1, the closer to 1 the better model it is.

According to the training model,

Auc score Decision Tree > Random Forest > Gradient Boosting Decision Tree > Logistic Regression > BernoulliNB

The best model is Decision tree, but according to the result of testing data, the accuracy and precision of decision tree is not as better as random forest, that is because decision tree will easily cause overfiting, so it can do it pretty well in training data but badly in testing data.

Logistic regression is the worst result, because there are 16 attributes in each sample, it is hard to guarantee these samples are linear. For logistic regression, it can perform really well in linear data instead of non-linear data. So in this method, we simply deal with samples as linear data, which is not very accurate, so the testing accuracy is not very good compared with other algorithm.

For decision tree, we have 16 attributes in each sample, one single decision tree can easily have high depth which causes over fit. But Random forest solve the over fit problem by randomly chose fixed samples and build decision tree for multiple times, then computing the mean. So, that is why random forest perform better than single decision tree.

Source code:
https://github.com/SandyHuang119/ML-project/blob/master/project.ipynb