Twitter Bot Detect

Ye Yuan
yy2068@nyu.edu

Duan Wang
dw2201@nyu.edu

Abstract

Twitter robots are created for many purposes. Too many robots will do a harm to the experience of human users. In this mid-way report, we will do some simple filter of the data set and apply decision tree algorithm and evaluate current result and plane a further improve and schedule other algorithm.

Keyword - *machine-learning, bot, Twitter*

I. Introduction

In this report, we will do some simple filter of the data set and apply decision tree algorithm and evaluate current result and plane a further improve and schedule other algorithm. To apply a decision-tree algorithm, we process the data into a more simple way: feature attribute 'id' by length which can imply id created time; apply logarithm to attribute 'followers_count' value which can decrease the size of tree and so on. Then we apply decision-tree algorithm and evaluate result. (TODO: more)

II. Motivation

Twitter robots are created for many purposes, like spreading rumors, winning a lottery, or advertising products, or just following some specific users. Too many robots will do a harm to the experience of human users. The users may loss competition with someone got many bots followers. The companies need twits by users to make decision, and the robots user may mislead their judgment. So it is important to classify human users with robots.

III. Related Work

-https://www.r-bloggers.com/programming-a-twitter-bot-and-the-rescue-from-procrastination/

-John P. Dickerson,Vadim Kagan,V.S. Subrahmanian "Using Sentiment to Detect Bots on Twitter: Are Humans more Opinionated than Bots?"

-David Mandell Freeman "Using Naive Bayes to Detect Spammy Names in Social Networks"

-https://www.technologyreview.com/s/529461/how-to-spot-a-social-bot-on-twitter/ -July 28, 2014

- Fame for sale: efficient detection of fake Twitter followers, Stefano Crescia,b, Roberto Di Pietrob , Marinella Petrocchia , Angelo Spognardia,1,⁎ , Maurizio Tesconia

IV. Data

The data set came from twitter API, where we can obtain eighteen attributes of a normal account: *id; id_str; screen_name; location; description; url; followers_count; friends_count; listedcount; created_at; favourites_count; verified; statuses_count; lang; status; default_profile; default_profile_image; has_extended_profile; name.* Then, each group of the class manually detect 50 robot data and 50 non-robot data. Combine all groups works, we get 2232 twitter account data finally.

In mid-way case, since we will apply decision-tree algorithm to build model, we deal some tricks on the data to decrease tree complexity and filter some meaning-less data by feature data value into a more meaningful form:

- since attribute 'id_str' is a type convert from 'id', we delete 'id_str' from work data set.
- 'id' is a primary key of data set. Consider it in entropy calculation is meaning-less. Instead we apply log10 as addition created time info.
- 'screen_name' contain some key words

which can bring some important information. We build a simple key words dictionary ["bot", "robot"] and detect whether user's screen name contain one or more of the key words. If has, mark it as 1, if not, mark 0. To improve our score of the algorithm, we may increase size of our key word dictionary in a future version.
- Since we don't have an efficient method to check value in 'location' is a real location on the earth or not, we fail to use above method matching the string. However, some users leave the location info. as blank which give us some tiny signal. Thus, we check if 'location' attribute is nan or not and return 0 or 1.
- 'description' may contain some key words which straightforward reveal a robot account. But lots of account in the data set remain this field as blank, it is also necessary to check if this field is nan or not.
- 'url' field allow users put their own personal page on it. The 'url' in data set is automatically produced by Twitter with some encryption. We can always ping the server " /t.co" successfully. Thus, we can only check whether user has their page upload in this field.
- 'followers_count' and 'friends_count' are important attributes in data set. However, their value have a large range from 0 to more than 10 thousand. We here apply logarithm with base 2 on it and decrease possible child number of these two nodes within 10.
- 'created_at' presented in a date form whose value span from 2013 to 2017. Considering reduce complexity of the tree, we category 'created_at' in one half year interval form year 2013-2017, which return 15 possible children (0 as before 2013).
- Since our computer can't correctly display some value in the field 'name', we drop this attribute. We can neither put this almost unique value into consideration nor to figure out special information hide behind.
- 'status' has a mess of information include other attribute value. The length of the value can somehow measure the account activity rate. We count the number of attribute included in 'status' field as the feature number.
- 'lang', ' default profile" 'default profile image' and 'has extended profile' conform with our aim to improve data set, we leave it unchanged.
With all featured new value ordered and some attributes dropped, we get a new data set ready for next step.

V. ALGORITHM USED

Decision Tree

A decision tree is a hierarchical data structure implementing the divide-and-conquer strategy. It is an efficient nonparametric method, which can be used for classification[1]. And compared with other methods, decision tree is easy to implement. Also it may become the base of other methods, like bagging, boosting or random forests. So we apply it into practice first.

As mentioned in the data part, we do modification on the origin data, like mapping the string attribute to an integer or a bool. This helps to make the tree simpler. Then it’s a common decision tree part: We apply the recursion on make tree. On each node, we first check if it is a leaf. If all the data here are same in result, which means all bots or all non-bots, we mark this node to be a leaf and return. Otherwise, first, we record the percent of the 'bot' in the data in this node. Then, we catch the entropy of each attribute, take the attribute with the least attribute, or the highest information gain, divide the data list into sublists, each sublist are in the attribute, and perform make tree function recursively on each subtree.

One question is that, since we modify but not use the original data, it is possible that: several datas are same in other attributes, but are different in ‘bot’ attribute. For this situation, we modify the make tree part. In each node, it will get not only the datalist, but also a list of attribute, when choose one attribute to divide data, it will remove the attribute from attribute list for building its children. When there is not attribute that can use be to divide, we also mark the node leaf. For this kind of node, we check the percent of 'bot' here. If there are more than or equal to half data are bots, we return bot for this leaf, otherwise we return non-bot.

After training the tree, we begun the classification. On each node, if it is not a leaf, by the attribute in the node and the corresponding value of the attribute in the input, we choose a child from the node and perform search on the child node. There is also one situation that: the value of the attribute from the input cannot find the corresponding child from the node. In this situation, we check the percent of 'bot' in this node. If it is more than 0.5(which means more than half of the data here are 'bot' when training the tree), we treat the input data as 'bot' and return bot, otherwise we return non-bot.

By the build tree and classification method, we got the decision tree.

VI. RESULT

Decision Tree:

First we randomize the order of the data(since it is all ‘bot’ row continued with all ‘non-bot’ row), then we use 10-Fold Cross-Validation to get the performance of the data. We calculate accuracy, precision, recall and f1score here.

Here is one result:

accuracy: 0.82846732863549
precision: 0.8273923317148277
recall: 0.8036031017719789
f1score: 0.8145687083986358

Because we randomized the order of data in the beginning, it returns different answer when we calculate it. So we perform 5 randomize in total, each time perform a 10-Fold Cross-Validation, and return the average.

accuracy: 0.8347425528507364
precision: 0.8308946302345429
recall: 0.8178888758569414
f1score: 0.8236313847394834

VII. CODE

Brief code:

```
import pandas as pd
import numpy as np
import math
import dateutil.parser as dparser
#data pre process
def checkNone(data_class, attrstr):
    #return None:1     not-None:0
def checkTF(data_class, attrstr):
    #return True:0     False:1
def checkNumber(data_class, attrstr, effi):
    #return n=log with base-effi
def checkEnglish(data_class, attrstr):
    #return isEnglish:1     other:0
def checkDate(data_class, attrstr):
    #date index:
    #    year<=2012: 0:3
    #    year==2013: 4:7
    #    year==2014: 8:11
    #    year==2015: 12:15
    #    year==2016: 16:19
    #    year==2017: 20:23
botDict = ["bot",
           "Bot",
           "robot",
           "Robot"]
def checkStringBot(data_class, attrstr):
    #return has sub string in botDict:1
else:0
```

```
#Make Decision Tree
def entroy(df,name,resultname,indexlist):
  #calculate entropy of one attribute
def
makesubtree(df,namelist,resultname,indexlist)
:
  #calculate the percent of bots on this node
  ….
  mynode.percent = pos / (pos+neg)
    if (pos*neg==0):
    mynode.leaf = True
    mynode.judge = t
    return mynode;
  min=32767
  #calculate the entropy and choose attribute
to use
  for name in namelist:
    val =
entroy(df,name,resultname,indexlist)
    if val<min:
      min = val
      ans = name
  #mark it leaf if there is no attribtue to
choose
  if (min==32767):
    mynode.leaf = True
    ….
    return mynode
  #add children and make subtrees
  ….
  for key in ma.keys():
    mynode.addchild(key,makesubtree(df,na
melist,resultname,ma[key]))
  ….
  return mynode;

def judge(node,ma):
  if node.leaf:
    return node.judge;
```

```
  else:
    try:
      return
judge(node.list[ma[node.attribute]],ma);
    except KeyError:
      # we can’t find the value in the node, so we
just return the majority
      return node.percent > 0.5;
```

Poject branch: *https://github.com/Dwan9/TwitterBot_CS6923.git*

VIII. EVALUATION

IX. CONCLUTION

reference

[1] Introduction to Machine Learning 2e
-Ethem Alpaydin