How to detect Twitter bots

Meng MENG
Tandon School of Engineering
New York University
Brooklyn, NY
mm8134@nyu.edu

Runbo GAO
Tandon School of Engineering
New York University
Brooklyn, NY
rg3099@nyu.edu

Xi CHEN
Tandon School of Engineering
New York University
Brooklyn, NY
Xc761@nyu.edu

***Abstract*—Twitter bots might have a negative effect on user experience and even cause risks and dangers. So it is important to distinguish them and take actions to prevent their latent harm. In this paper, we focus on solving this problem by using three methods in machine learning. By designing and comparing the three of the methods, we expect to find a good way to tell the bots from Twitter users.**

Keywords—machine learning, bot

I. INTRODUCTION

In this paper, we aim to design a high accuracy classifier to detect bots account from Twitter users accounts database. Prior to the work, we acquired Twitter account datasets through Twitter APIs and divide them into two datasets as training sets and testing sets. Then we presented three machine learning methods for the twitter account datasets. Those are K Nearest Neighbors, Support Vector Machine and Naïve Bayes methods. Implementation results are evaluated and compared. The result using our classifier has been tested feasible. It's practical to apply such detection tool on Twitter accounts.

II. MOTIVATION

Nowadays, Twitter has become one of the most popular social networking applications all over the world. Along with more and more registered human twitter users, the robots twitter accounts increased fast too. It is necessary for us to distinguish bots users and human users. First, the latent risks of the bots users are uncertain. The bots user accounts may cause huge abundance. The harmful bots could be designed with the goals of persuading, smearing, or deceiving. Second, besides as the connection and communication tool, Twitter becomes a significant data source to do the analysis for many kinds of industries. To gain more accuracy and applicable result of the data analysis, it's very important to extract cleaner data from the data set. It depends on human experience to detect bots users by manually. However, it is not feasible to manually detect millions of unlabeled accounts; in addition, it is not always reliable to classify an account based on intuition and experience. As a result, it seems essential to design methods to distinguish bots from Twitter accounts with high accuracy.

III. RELATED WORK

1. Ferrara, et. Al. The Rise of Social Bots.
2. Lee, et al. Who Will Retweet This? Automatically Identifying and Engaging Strangers on Twitter to Spread Information.
3. Shellman, Erin. Bot or not.
4. Bo Pang and Lillian Lee. Thumbs up? Sentiment Classification using Machine Learning Techniques

IV. DATA

Twitter user data are extracted from Twitter database using Twitter API. The original data format is JSON and they are converted to CSV format for the experiment. Within the dataset, we divide it into two parts: 50 for training set and 50 for testing set. There are 20 fields in the dataset, including 19 features and 1 label. As for label, 1 stands for bot and 0 for non-bot. We have 20 attributes for the data:

1	id	11	Favourites_count
2	Id_str	12	Verified
3	Screen_name	13	Statuses_count
4	Location	14	Lang
5	Description	15	Status
6	Url	16	Default_profile
7	Followers_count	17	Default_profile_image
8	Friends_count	18	Has_extended_profile
9	Listed_count	19	name
10	Created_at	20	Bot

V. ALGORITHM(S) USED

For this project, it is a classification problem but not a regression problem because our purpose is to detect an account is a bot or not. And we selected K-NN model for its usability and efficiency. The SVM model has a high accuracy and fits our goal. And Random Forest model wins for its stable result.

1. K Nearest Neighbors

In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

- In k-NN classification, the output is a class membership. An object is classified by a majority vote of its

neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

- In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

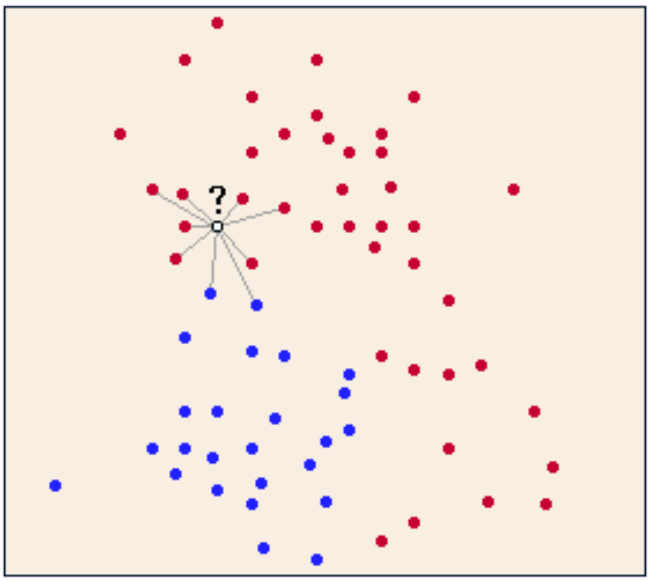

Fig1. k-NN

k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification.

The neighbors are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

A shortcoming of the k-NN algorithm is that it is sensitive to the local structure of the data.[1]

In our experiment, k is set to 10.

2. Support Vector Machine

In machine learning, support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

In the two-category case, the basic idea behind the training procedure is to find a hyperplane, represented by vector $\vec{\omega}$, that not only separates the document vectors in one class from those in the other, but for which the separation, or margin, is as large as possible. This search corresponds to a constrained optimization problem; letting $c_i \in \{1, -1\}$ (corresponding to positive and negative) be the correct class of document d_i, the solution can be written as

$$\vec{\omega} := \sum_i \alpha_i c_i \vec{d_i}, \alpha_i \geq 0,$$

where the α_i's are obtained by solving a dual optimization problem. Those $\vec{d_i}$ such that αj is greater than zero are called support vectors, since they are the only document vectors contributing to $\vec{\omega}$. Classification of test instances consists simply of determining which side of $\vec{\omega}$'s hyperplane they fall on.[2]

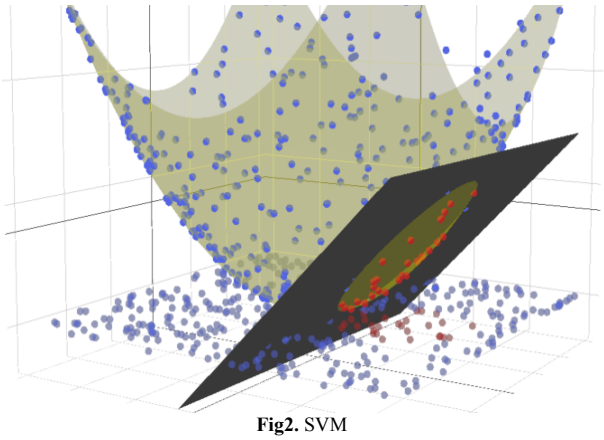

Fig2. SVM

In our experiment, we choose linear support vector classification, with l2 norm in the penalization and standard SVM loss function.

3. Random forests

Decision tree learning uses a decision tree as a predictive model observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). It is one of the predictive modelling approaches used in statistics, data mining and machine learning. Tree models where the target variable can take a finite set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.

In decision tree learning, ID3 (Iterative Dichotomiser 3) is an algorithm used to generate a decision tree from a dataset.

The ID3 algorithm begins with the original set as the root node. On each iteration of the algorithm, it iterates through every unused attribute of the set and calculates the entropy (or information gain) of that attribute. It then selects the attribute which has the smallest entropy (or largest information gain) value. The set is then split by the selected attribute (e.g. age is less than 50, age is between 50 and 100, age is greater than 100) to produce subsets of the data. The algorithm continues to recurse on each subset, considering only attributes never selected before. Recursion on a subset may stop in one of these cases:

- every element in the subset belongs to the same class (+ or -), then the node is turned into a leaf and labelled with the class of the examples
- there are no more attributes to be selected, but the examples still do not belong to the same class (some are + and

some are -), then the node is turned into a leaf and labelled with the most common class of the examples in the subset

- there are no examples in the subset, this happens when no example in the parent set was found to be matching a specific value of the selected attribute, for example if there was no example with age >= 100. Then a leaf is created, and labelled with the most common class of the examples in the parent set.

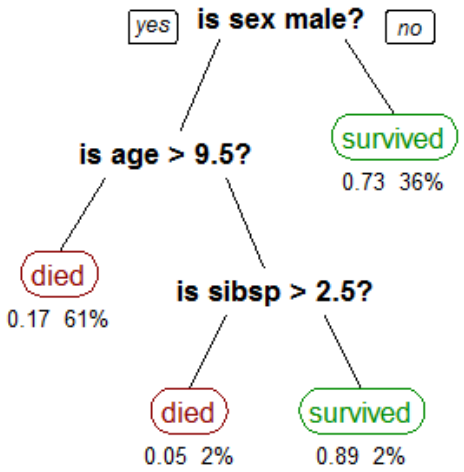

Fig3. Decision Tree

Throughout the algorithm, the decision tree is constructed with each non-terminal node representing the selected attribute on which the data was split, and terminal nodes representing the class label of the final subset of this branch.[3]

Random forests or random decision forest 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 over-fitting to their training set.[4]

In our experiment, we use 10 estimators, with maximum depth of 7, and minimum amount of leaves of 3.

VI. RESULT

Data ETL is the most significant step for this project. The raw data has problems such as type mixing, value missing and format inconsistence.

At first as a test, we only extracted all the features with type of numerical or Boolean from the raw dataset. The accuracy of the result using the three models were all under 0.7.

To improve the accuracy, we added the features in string type. Through the observation of the raw dataset, we found that the texts in column “name”, “screen_name”, “description”, “location” contains the information indicates that it is a bot account. Such texts include ‘bot’ or ‘robot’ and corresponded label is bot. To clean such feature, we added several new columns and assigned the value as 1 if it contains substring and 0 if not. Also, we created a new column “contain_bot” which value is the sum of the new columns which have been assigned 1, which indicates the possibility that this account is a bot.

	kNN	SVM	Random forest
Accuracy	0.8999129	0.8398348	0.8931131
Precision	0.9372001	0.8335125	0.9232856
Recall	0.8452042	0.8271363	0.8452655
F1-score	0.8883459	0.8299765	0.8821495
AUC	0.8972046	0.8390780	0.8906720

After the first step of data cleaning, the accuracy of the results reached more than 0.8 which improved a lot.

And the accuracy might be affected if some features are redundant for the classification. We implemented different combinations of all the extracted features. And it turned out that the result is the best when chose following features: loc*_bot, name_bot, des_bot, screen_bot, verified, contain_bot, followers_count, friends_count, statuses_count.*

Then we implemented the three models on the training set with cross validation. Specifically, we applied 10-fold cross validation.

And the results are as following:

VII. CODE

We use github to do the version control, and the link is as below: *https://github.com/MechineLearingProject/Project*

The codes are depended on IPython with several useful packages, include pandas, numpy, sklearn and matplotlib.

The pandas, numpy, sklearn packages provide built-in machine learning functions. These functions are used to clean data, filter data and analyze data evaluate the results and so on.

The matplotlib package provide methods like in matlab to help visualize the resulst. It provides intuitive charts which helps us analyze the relationship between training methods.

Some core code:

1. *Extract import methods from datasets, drop useless attributes like id.*

```
import pandas as pd
import numpy as np
from sklearn import preprocessing

def tf(x):
    if x == "TRUE" :
        return 1
    elif x == "FALSE" :
        return 0
    else :
        return 0.5
```

```
def isnull(x):
#     if x!="None" and x!="none" and x!="Null" and
        x!="null" and len(x)>2 and x!=np.nan and
        x!='NaN':
    if x=="None" or x=="none" or x=="Null" or
        x=="null"  or x==np.nan or pd.isnull(x):
        return 1;
    else:
        return 0;



# #import data
bots_data = pd.read_csv('bots_data.csv')
nonbots_data = pd.read_csv('nonbots_data.csv')
# concatenate bots and nonbots
# df = pd.concat([bots_data, nonbots_data])
df = pd.read_csv('training_data_2_csv_UTF.csv')
        df.head(1
```

	followers_count	friends_count	listedcount	favourites_count	verified	statuses_count	default_profile	defau
0	1129	7	2	0	False	23557	False	False
1	0	22	0	0	False	1	True	True
2	3	0	3	0	False	1050	True	False
3	505	13	49	0	False	5109	False	False
4	15	0	11	0	False	31365	True	False

2. *Convert data type: convert string or boolean data type to numeric.*

```
df.insert(4,'des_bot',0)
df.loc[pd.notnull(df['description']) &
        df['description'].str.contains("bot|Bot|BOT"),
        'des_bot'] = 1
df.insert(5,'name_bot',0)
df.loc[pd.notnull(df['name']) &
        df['name'].str.contains("bot|Bot|BOT"),
        'name_bot'] = 1
df.insert(6,'loc_bot',0)
df.loc[pd.notnull(df['location']) &
        df['name'].str.contains("bot|Bot|BOT"),
        'loc_bot'] = 1
df.insert(7,'loc_isnull',0)
df['loc_isnull'] = np.vectorize(isnull)(df['location'])
df.insert(8,'des_isnull',0)
df['des_isnull'] = np.vectorize(isnull)(df['description'])
df.insert(9,'url_isnull',0)
df['url_isnull'] = np.vectorize(isnull)(df['url'])
df.insert(10,'status_isnull',0)
df['status_isnull'] = np.vectorize(isnull)(df['status'])
def f(x, y, z, r):
    return x+y+z+r
df.insert(11,'screen_bot',0)
df.loc[pd.notnull(df['screen_name']) &
        df['screen_name'].str.contains("bot|Bot|BOT"),
        'screen_bot'] = 1
        df.insert(12,'des_by',0)
df.loc[pd.notnull(df['description']) &
        df['description'].str.contains("by|By"), 'des_by']
        = 1
df.insert(13,'des_tweet',0)
df.loc[pd.notnull(df['description']) &
        df['description'].str.contains("tweet|Tweet"),
        'des_tweet'] = 1
df.insert(14,'des_er',0)
df.loc[pd.notnull(df['description']) &
        df['description'].str.contains("er,|er "), 'des_er']
        = 1
df.insert(15,'des_st',0)
df.loc[pd.notnull(df['description']) &
        df['description'].str.contains("st,|st "), 'des_st'] =
        1
df.insert(16,'contain_bot',0)
df['contain_bot'] = np.vectorize(f)(df['des_bot'],
        df['name_bot'], df['loc_bot'],df['screen_bot'])
df.insert(17,'des_hourly',0)
df.loc[pd.notnull(df['description']) &
        df['description'].str.contains("hourly|Hourly"),
        'screen_bot'] = 1
df.insert(18,'des_randomly',0)
df.loc[pd.notnull(df['description']) &
        df['description'].str.contains("randomly|Randoml
        y"), 'screen_bot'] = 1
```

	0	1	2	3	4	5	6	7	8
0	1.172116e-05	0.000007	0.000009	0.0	0.0	3.432282e-03	0.0	0.0	1.0
1	0.000000e+00	0.000022	0.000000	0.0	0.0	1.457012e-07	1.0	1.0	1.0
2	3.114567e-08	0.000000	0.000013	0.0	0.0	1.529862e-04	1.0	0.0	1.0
3	5.242855e-06	0.000013	0.000220	0.0	0.0	7.443873e-04	0.0	0.0	1.0
4	1.557284e-07	0.000000	0.000049	0.0	0.0	4.569917e-03	1.0	0.0	1.0

3. *Normalize data: it's a necessary method to use KNN.*

```
 #convert boolean features to integer
 df['verified'] =
 np.vectorize(tf)(df['verified']).astype(int)
 df['default_profile'] =
 np.vectorize(tf)(df['default_profile']).astype(int)
 df['default_profile_image'] =
 np.vectorize(tf)(df['default_profile_image']).astype(int
 )
 df['has_extended_profile'] =
 np.vectorize(tf)(df['has_extended_profile']).astype(int)


 df['followers_count'] =
 df['followers_count'].map(lambda x: np.log10(x+1))
 df['friends_count'] = df['friends_count'].map(lambda
 x: np.log10(x+1))
 df['listedcount'] = df['listedcount'].map(lambda x:
 np.log10(x+1))
```

```
df['favourites_count'] =
df['favourites_count'].map(lambda x: np.log10(x+1))

df['statuses_count'] =
df['statuses_count'].map(lambda x: np.log10(x+1))

# feature projection

df =
df[['contain_bot','followers_count','friends_count','list
edcount','favourites_count','statuses_count','default_pr
ofile','default_profile_image','has_extended_profile','b
ot']]
```

	followers_count	friends_count	listedcount	favourites_count	verified	statuses_count	default_profile	default_pro
0	1129	7	2	0	0	23557	0	0
1	0	22	0	0	0	1	1	1
2	3	0	3	0	0	1050	1	0
3	505	13	49	0	0	5109	0	0
4	15	0	11	0	0	31365	1	0

4. *Using KNN and SVM. Including using cross-validation method.*

```
#data normalization(to [0:1])
x = df.values
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
df = pd.DataFrame(x_scaled)


    from sklearn.model_selection import KFold
    from sklearn.model_selection import
StratifiedKFold
    from sklearn.neighbors import
KNeighborsClassifier
    from sklearn.ensemble import
RandomForestClassifier
    from sklearn import svm
    from sklearn.decomposition import PCA
    from sklearn.metrics import accuracy_score
    from sklearn.metrics import
precision_recall_fscore_support
    from sklearn.metrics import f1_score
    from sklearn.metrics import roc_auc_score



    kf = KFold(n_splits=10, shuffle=True)
    X = df[df.columns[0:-1]]
    # pca = PCA(n_components=7)
    # pca.fit(X)
    # X = pca.transform(X)
    y = df[df.columns[-1]]
    # for neighbor_number in range(3,20):
    accuracy_knn = 0
    accuracy_svm = 0
    accuracy_rt = 0
    precision_knn = 0
    precision_svm = 0
    precision_rt = 0
    recall_knn = 0
    recall_svm = 0
    recall_rt = 0
    f1score_knn = 0
    f1score_svm = 0
    f1score_rt = 0
    auc_knn = 0
    auc_svm = 0
    auc_rt = 0
    for train_index, test_index in kf.split(X,y):
        X_train, X_test = X.ix[train_index,:],
X.ix[test_index,:]
    #     X_train, X_test = X[train_index,:],
X[test_index,:]
        y_train, y_test = y[train_index],
y[test_index]


        knn_classifier =
KNeighborsClassifier(n_neighbors=10)
        knn_classifier.fit(X_train, y_train)
        pred_knn = knn_classifier.predict(X_test)


        svm_classifier =
svm.LinearSVC(penalty='l2',
loss='squared_hinge', dual=True, tol=1e-4)
    #     svm_classifier = svm.NuSVC()
        svm_classifier.fit(X_train, y_train)
        pred_svm = svm_classifier.predict(X_test)


        rt_classifier =
RandomForestClassifier(n_estimators=10,
max_depth = 7, min_samples_leaf = 3)
        rt_classifier = rt_classifier.fit(X_train,
y_train)
        pred_rt = rt_classifier.predict(X_test)


        accuracy_knn = accuracy_knn +
np.mean(pred_knn == y_test)
        accuracy_svm = accuracy_svm +
np.mean(pred_svm == y_test)
        accuracy_rt = accuracy_rt +
np.mean(pred_rt == y_test)


        precision, recall, fscore, support =
precision_recall_fscore_support(y_test,
pred_knn, average = 'binary', pos_label = 1)
        precision_knn = precision_knn + precision
        recall_knn = recall_knn + recall


        precision, recall, fscore, support =
precision_recall_fscore_support(y_test,
pred_svm, average = 'binary', pos_label = 1)
```

```
        precision_svm = precision_svm + precision
        recall_svm = recall_svm + recall


        precision, recall, fscore, support =
precision_recall_fscore_support(y_test, pred_rt,
average = 'binary', pos_label = 1)
        precision_rt = precision_rt + precision
        recall_rt = recall_rt + recall


        f1score_knn = f1score_knn +
f1_score(y_test, pred_knn)
        auc_knn = auc_knn + roc_auc_score(y_test,
pred_knn)
        f1score_svm = f1score_svm +
f1_score(y_test, pred_svm)
        auc_svm = auc_svm +
roc_auc_score(y_test, pred_svm)
        f1score_rt = f1score_rt + f1_score(y_test,
pred_rt)
        auc_rt = auc_rt + roc_auc_score(y_test,
pred_rt)
```

5. *Visualize the result*

```
import matplotlib.pyplot as plt
plt.figure();
        d = {'knn' : pd.Series([accuracy_knn / 10,
precision_knn / 10, recall_knn / 10, f1score_knn / 10,
auc_knn / 10], index=['Accuracy', 'Precision', 'Recall',
'F1-score', 'AUC'])
        ,'svm' : pd.Series([accuracy_svm / 10,
precision_svm / 10, recall_svm / 10, f1score_svm / 10,
auc_svm / 10], index=['Accuracy', 'Precision', 'Recall',
'F1-score', 'AUC'])
        ,'random forest' : pd.Series([accuracy_rt /
10, precision_rt / 10, recall_rt / 10, f1score_rt / 10,
auc_rt / 10], index=['Accuracy', 'Precision', 'Recall',
'F1-score', 'AUC'])
        }
    df2 = pd.DataFrame(d, columns=['knn', 'svm',
'random forest'])
    df2.plot.bar()
    plt.show()
```

VIII. Video Link

https://vimeo.com/216612590

IX. Evaluation

As Fig.4 shows, all of the scores of the classifiers using KNN and random forest are relatively higher compared with SVM. The accuracy of using KNN and random forest reaches about 0.9, which shows that the classifier is quite accurate and acceptable.

What's more, we compared all kinds of scores. It is easy to find that precision is higher than accuracy, which reaches about 0.93; recall is lower than accuracy, which only reaches about 0.84. Unbalanced score means classifiers hardly misjudge non-bots to bots; however, many bots are misjudged to non-bot, which might be possibly caused by hidden information of bot or the features are not so typical.

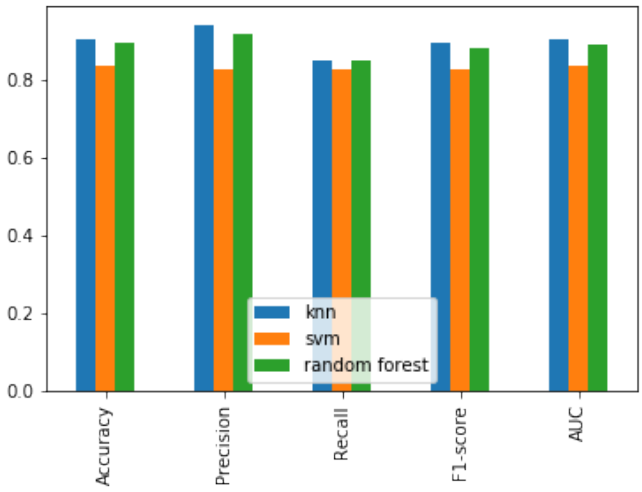

Fig4. Scores

As for random forest, it is different from the other two algorithms. Because all the estimators in random forest are generated randomly, so the results are not stable. However, it avoids over-fitting, which is an advantage in general.

X. Conclusion

We mainly use these two models on the testing data. However, as Kaggle shows after submitting results, random forest turns out to run much better than KNN, whose scores are 0.96 and 0.92 separately. So we use two results of random forest as final submission selection.

After the competition completed, the score of our team turns out to be a good one, which uses random forest tree as the final model. So we come to the conclusion that random forest fits this problem well.

However, as we have mentioned, SVM and KNN didn't work as well as expected. There are some possible reasons. For example, we didn't find a perfect kernel function for SVM. So it is still possible to make an improvement and there are still some work to continue in the future.

XI. Reference

[1] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
[2] https://en.wikipedia.org/wiki/Support_vector_machine
[3] https://en.wikipedia.org/wiki/ID3_algorithm
[4] https://en.wikipedia.org/wiki/Random_forest