

-China Thinks Big

FINAL RESEARCH REPORT: Operation Ravens

Team tech_raven

This report is constructed by (alphabetically ordered):

Austin Liu (刘彦豪) Cyzus Chi (池一舟) Helen Wang (王简) Jack Ding (丁子桓)

Scott Huang (黄星越) Seth Yuan (源岷璁)



Table of Contents

Table of Contents	2
1. Introduction	3
2. Purpose	3
3. Method	3
3.1 Data Collection	4
4. Procedure	4
4.1 Data Manipulation	4
4.2 Feature Selection	8
4.3 Testing Algorithms and Cross Validation	9
4.4 Random Splitting of Testing Data	10
5. Framework	10
5.1 CART	10
5.2 The Evaluation of efficiency	11
6. Result	12
7. Evaluation	13
7.1 Methodological issues	13
7.2 Solutions and improvements	15
8. Reference	16
9. Appendix	17

1. Introduction

Big data, known as those data which are not able to be collected, managed or executed using standard instruments in a specified period. It is a widely spread information property in diverse styles with high increasing rates, that must be controlled using newer executing methods. In the 21st Century, with the rapid development of the internet technology, big data has its potential to be applied to different fields. Can the big data be used to an Email filtering application to help people to filter the junk emails? Our team has always been applying efforts over the past ten weeks, developing a junk email classifier and implementing it to the society. We believe that the nation will benefit from this product, returning a clean environment to the mailboxes.

2. Purpose

The principal purpose of the research is to present a demonstration that the big data can be used to clean up the junk emails in the cyberspace. Apart from this, it is aiming to raise the awareness of the public. Since although there are lots of existing algorithms to deal with the problem, especially junk message filter in the mailbox, there are still lots of junk messages spreading over the internet. Our study is trying to create a combined algorithm which intrigues people to keep updating those methods to prevent those useless and harmful languages.

3. Method

3.1 Data Collection

As our topic is how to apply big data, a significant amount of data is the essential requirement for us to proceed our plan. According to 'Big Data': Big gaps of knowledge in the field of the Internet,' big data usually includes data sets with sizes beyond the ability of commonly used software tools to capture, curate, manage, and process data within a tolerable elapsed time[1]. That is, the data we collected is impossible for people to analyze manually.

A large sample of typical email and spam email is required to train the algorithm for mail filtering. As we are incapable of collecting such amount of sample on our own both due to time reason and because it goes beyond our ability, we decided to obtain data sets that already created by other people. These resources are available on the internet with free accesses, and we directly downloaded them as our raw data[2]. The information on the website containing these data sets was informed by the tutor of our team from CTB.

4. Procedure

4.1 Data Manipulation

There are total over 13,000 of spam emails and over 14,000 of normal email samples. Although it is not a hugely significant amount of data, we considered that as enough for the algorithm to generate reliable features of spam email. Two graphs are constructed on the next page for visually better understanding.



















	1104641208.1952_5.txt	2005/10/27 20:34	文本文档	2 KB
	1104641411.4171_1098.txt	2005/10/27 20:34	文本文档	3 KB
	1104641419.4171_1104.txt	2005/10/27 20:34	文本文档	4 KB
	1104643556.6292_3.txt	2005/10/27 20:34	文本文档	35 KB
	1104643599.6292_30.txt	2005/10/27 20:34	文本文档	6 KB
	1104643608.6292_60.txt	2005/10/27 20:34	文本文档	5 KB
	1104643612.6292_78.txt	2005/10/27 20:34	文本文档	4 KB
	1104643626.6292_198.txt	2005/10/27 20:34	文本文档	3 KB
	1104722194.10543_13.txt	2005/10/27 20:34	文本文档	5 KB
	1104722195.10543_15.txt	2005/10/27 20:34	文本文档	8 KB
	1104722470.11132_35.txt	2005/10/27 20:34	文本文档	6 KB
	1104722470.11132_37.txt	2005/10/27 20:34	文本文档	4 KB
	1104722473.11132_51.txt	2005/10/27 20:34	文本文档	5 KB
	1104722499.11132_100.txt	2005/10/27 20:34	文本文档	4 KB
	1104722499.11132_104.txt	2005/10/27 20:34	文本文档	2 KB
	1104722501.11132_108.txt	2005/10/27 20:34	文本文档	2 KB
	1104722503.11132_114.txt	2005/10/27 20:34	文本文档	3 KB
	1104722504.11132_122.txt	2005/10/27 20:34	文本文档	3 KB

Fig4.1. Data collection

[Your specialist's appointment starts on the 21th](#) +
 ↓
[F!^O_R'i.C^E_T 40 m.gg](#) +
 ↓
[3o P||\\$ 99.00](#) +
[60 P||\\$ 189.95](#) +
[90 PIL\\$ 239.00](#) +
 ↓
[MOre Pain-ReLIIf Here](#) +
 ↓
[Same Day ShippIng](#) +
 ↓
 ↓
 ↓
 ↓
 ↓
[r e'm.0 v.e](#) +
 ↓
 confirm your email +
 ↓
 Ashley Farris +
 Privatedetective +
 RAWAT ENTERPRISES, PUNE - 410507, India +
 Phone: 411-744-6432 +
 Mobile: 122-861-4362 +
 Email: lbjhwzfiuxk@dragoncon.net +

Fig 4.2

emails, the structure is apparent. Only using the string manipulation technique in Python is enough to deal with it, which merely involves deleting the unnecessary information and replacing all the numbers and punctuation.

```
From 1mrn@mailexcite.com Mon Jun 24 17:03:24 2002Return-Path: merchantsworld2001@juno.comDelivery-Date: Mon May 13 04:46:13 2002Received: from m
tml; charset="DEFAULT"<html><body><center><h3><font color="blue"><b>The Need For Safety Is Real In 2002, You Might Only Get One Chance - Be Ready
crime.<p>You hear about it on TV. You read about it in the newspaper.<br>It's no secret that crime is a major problem in the U.S. today.<br>Crim
ty of quality personal<br>security products. Visit our site, choose the personal security<br>products that are right for you. Use them, and joi
.95)<br>__The StunBaton, 300,000 Volts ($79.95)<br>__Pen Knife (One $12.50, Two Or More $9.00)<br>__Wildfire Pepper Spray (One $15.95, Two Or
```

Fig. 4.4 Raw data

```
| TeamI have attached a memo that I would like to send to D Hart setting out AEP s outstanding action
items in respect to the Close Please review and provide any comments e g additional actions for AEP or
revised status for Enron that I may have missed Thanks Brian
```

Fig. 4.5 Completed data

Although the amount of data is not that big, around 30,000 pieces of the file are still impossible to manipulate section by section. Therefore, another file management library was bringing into use-'os' library in python. By looping specific file folder, the absolute address of each file can be read and apply to the extraction algorithm which allows the system to finish all the manipulation without inspection.

```
#Module for Html analysis
import re
from bs4 import BeautifulSoup
import emlEXTRACT as eml
import os
path1 = 'C:/Users/huang/Desktop/CTB/HtmlWash/HtmlWash/GP/'
dirs = os.listdir(path1)
j=1
for i in dirs:
    path2 = path1+i+'/'
    dirs2 = os.listdir(path2)
    for file in dirs2:
        print(os.path.join(path2, file))
        f = open(os.path.join(path2, file), 'r')
```

Fig 4.6.

Through applying the method shown above, all the raw data was changed into the standard ".txt" file which can be used by the learning algorithm. Although the data that have been manipulated make no sense for people to read, they do allow the learning algorithm to generate reliable features.

4.2 Feature Selection

Feature Selection is the next step of processing all those emails. Instead of having words when calculating through machine learning, we first statistically show the most common 300 words appeared in all files. After eliminating all the common words such as 'the' or 'me', all the number figures, and decapitalized all capital letter, we still have to manually eliminate several words that do not make any sense(such as 'sd' or 'dl') to make the whole feature list clean and useful.

After gaining 300 most common word, we treat them as a feature vector label. This label means that we can represent each email, spam or not, as a 300 dimensions feature vector where each aspect shows the frequency of that word in this email. For example, if our standard word list is :['I','love', 'cheese'] and one of the emails said "I love cheese, I just love it", the feature vector for this email will be [2,2,1] because word 'I' appears 2 times, 'love' appears 2 times, and 'cheese' appears 1 time. We then convert all emails into such form of position vector and store them separately according to its label (spam or not).

Upon deliberate considerations of ‘not reinventing the wheel’ as addressed in one of the comments the team received in week 8 suggesting us to make use of existing machine learning libraries instead of re-writing our own, we decided to re-do the project using existing library of ‘sklearn’. It is proven to be successful as we finished our prototype fairly quickly.

We then run a cross-validation result for each algorithm to give us a glimpse on how well the algorithms will do in the case of classifying email. Our test score is shown below:

```
GB Training Score: 0.85
GB Testing Score: 0.84
KNN Training Score: 0.95
KNN Testing Score: 0.93
SVC Training Score: 0.94
SVC Testing Score: 0.94
Decision Tree Training score:0.984883
Decision Tree Testing Score: 0.94
```

Fig. 4.6 Classifying results of different algorithms

As we can see that, for the testing score, SVC and Decision tree are reaching a relatively higher score whereas Gaussian Bayes (Naïve bayes assuming Gaussian distribution, or, in other word, Normal distribution) has a lower score.

4.4 Random Splitting of Testing Data

Then we modified our function of testing the score into testing one particular sample. This sample is used later to classified all email and divide those emails into two different class base for each algorithm. However, testing data also need to be randomly split in two different part each time testing in order to provide accurate result of the scoring.

5. Framework

To optimise prediction results, we came up with an idea of ensembling the use of multiple classifying algorithms, inspired by the inherent logic of Decision Tree. To be specific, we utilized the CART (Classification and Regression Tree) to integrate all classifying algorithms.

5.1 CART

The inherent logic of CART is as follows: It starts on a root node that takes the whole training data set as the input. The root node asks a question and split the data into two subsets according to the answers given by each record. At each subgroup, another node that takes the subset as an input will form and ask another question. This process repeats itself at every node formed hereafter until the data set is separated into multiple pure sets, which means that each set contains only one kind of object. To ensure the best question being asked at each node, the tree will compare how effectively each question split the data.

In our case, a list of all the classifying algorithms we implemented, such as GNB, SVM and K-NN, is created. At each node, instead of asking a question as addressed above, the tree will try to split the data with each of the listed algorithms into two subsets and evaluate the efficiency of classifying respectively. The efficiency is evaluated by a test statistics, the way of reaching which is discussed in details in the next section. After this, the tree will then compare the efficiency test statistics and decide on a best classifying algorithm and store it at this node. Again, the process is repeated at each node until the entire tree is built. The tree is then used to conduct any classification task on any sets of data.

5.2 The Evaluation of efficiency

The standard way of the efficiency of the CART is called Gini Impurity. By definition, Gini impurity is a measurement of how often an element of a set would be incorrectly labeled if it was marked randomly. It merely indicates how mixed up a set is. For example, I pick up a fruit randomly from a bunch of fruit, and a label randomly from a pile of tags of that bunch of fruit. The probability that the tag does not match the fruit is the Gini impurity. A calculation of Gini impurity is as follows:

We denote the possibility of one element to be chosen as P_i , then the probability of a wrong label being picked up is $(1-P_i)$. These two choices are independent because one decision does not affect the other. As a result, the chance of not-matching is a joint probability given by $P_i \times (1-P_i)$. The Gini impurity of the whole set will be the sum of this for every element.

Through this evaluation, the tree can note down the most efficient algorithm possible at each node. Also, it is also a way to determine when the tree will be fully grown - that is when the tree can no longer receive any lower Gini impurity.

6. Result

After deliberately testing through cross-validation of the data several times and took the average, we have calculated our algorithm's accuracy regarding percentage score. With all four algorithms (SVM, KNN, Decision Tree, and Naive Bayes) and let testing sample size be 1000, we get the following score in the table:

K:82.42	S:93.71	N:77.22	D:91.67
KS:94.38	KD:93.93	KN:84.57	SD:93.09
SN:94.87	ND:95.35	KSN:96.06	KSD: 96.73
KND:90.79	SND:94.15	KSND:94.71	Highest: 96.73

K : KNN-algorithm

S: SVM algorithm

N: Naive Bayes

D: Decision Tree

Table 6.1: Test score for tree-algorithms with different combination of algorithms

As the test result suggests, through the trails of this classifier using newly designed-algorithms, we found out that using KNN, SVM, and Decision Tree as a combined algorithm is the most efficient and accurate algorithms as it reaches the overall accuracy of 96.73%. Also, we found out that Naive Bayes algorithms are not suitable for such text classifier as it's average efficiency is 77.22%. We assumed that the reason why Naive Bayes does not do well is that each feature in the feature vector is not strictly independent of each other.

In conclusion, our newly designed-algorithm can improve the accuracy of the specific algorithm of classifying junk email before by 3%. With our new algorithm, a less regular email will be wrongly organized, and more spam email will be eliminated in users' mailboxes.

7. Evaluation

7.1 Methodological issues

7.1.1 Data collection

spam emails and about 14,000 regular emails in total. However, comparing to the email transmit-receive stream of China every day (About 30-40 emails sent and received by an individual)[1], our data is relatively small, which may lead to low ecological validity problems: Does the data we sampled support the general features of spam email? Do the results obtained have high consistency when we apply them to the society? At the same time, hardly can we justify a boundary or criteria to distinguish between standard or junk emails. This has made the research became more difficult.

To get as much data as possible, we have decided using web crawlers to gather the resources we need at the beginning. Using of web crawler might raise an ethical issue: Is the way by which we collect the data ethical? To take the robot protocol as an example, whose purpose is to resist possible web crawling activities. Sometimes we need to break through to gather data from Chrome. This point can be taken a step further: Is it necessary to break this ethical redline while gathering data?

What's more, some of the data we have were from 2005, which was more than ten years ago. Although the general features of spam emails might not have changed a lot, there may be a low ecological validity. Another point which may cause low ecological validity is that we ignored the emails which are video or audio oriented. That means we cannot get features of spam emails in the perspectives of videos and audios.

7.1.2 Pre-existing method

There are lots of pre-existing methods on the internet. Since we are not aiming to invent a new algorithm, we do not reinvent the wheels but using the existing library 'sklearn.' However, to avoid repetition, we combine various algorithms and build up our program ourselves. Our main idea is to use a decision tree as a framework and combine all the result of the other algorithms with weighting of their accuracy.

7.1.3 Data Processing

During data processing, since we do not have enough ability to analyze the use of pictures and videos, so we deliberately clean out the videos and images inside the message for our program used. There are cases that the photographs are the content of the email which makes them spam emails. When we ignore such information, misclassify there will be. Therefore, it may result in invalidity in our result since the images and videos also could influence the decision of whether it is 'spam' (junk message) or 'ham' (standard message).

7.2 Solutions and improvements

7.2.1 Data collection

The project can be improved by using a more massive amount of data to obtain more accurate and general features. In this way, it would have higher possibility to identify the spam message and to protect people from the exposure to those words.

In considering the ethical issue and robots protocol, we gave up using the web crawler to gather data. Instead, we collect existing dataset on the internet as our training and testing set.

7.2.2 Pre-existing methods

To avoid repetition and plagiarism, we built up our classifier by combining various algorithms. The advantage of using a combination of algorithms is that it helps the program to increase its internal reliability. In other words, making the outputs of the algorithms more consistent.

7.2.3 Data processing

We should furthermore identify video and audio features of spam emails, making our outputs contain more ecological validity. For example, we are planning to gather video and audio orientated emails, by firstly using manually distinguish using self-report picture rating (scale from 0: Not spam at all to 5: Definitely spamming). Using questionnaires to develop the self-report to increase the internal reliability and the population validity of the results. Then, implementing the results to train the machine and furthermore establishing a classifier which works well with video and audio spam emails.

8. Reference

- [1]: Snijders, C.; Matzat, U.; Reips, U.-D. (2012). *"Big Data": Big gaps of knowledge in the field of Internet*". International Journal of Internet Science. 7: 1–5.
- [2]: <http://www2.aueb.gr/users/ion/data/enron-spam/>
- [3]: <https://stackoverflow.com/questions/31392361/how-to-read- eml-file-in-python/answered>

9. Appendix

9.1 Classifier method code(in python 3.6)

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn import cross_validation

from sklearn.naive_bayes import GaussianNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import LinearSVC

from sklearn.tree import DecisionTreeClassifier

global testnum

testnum = 1000

import random

class Dataset():

    def __init__(self, data, target):

        self.data = data

        self.target = target
```

```
def test_LinearSVC(X_train, X_test, y_train, y_test,sample):
```

```
    cls = LinearSVC()
```

```
    cls.fit(X_train, y_train)
```

```
    #print('SVC Training Score: %.2f' % cls.score(X_train, y_train))
```

```
    #print('SVC Testing Score: %.2f' % cls.score(X_test, y_test))
```

```
    prediction = cls.predict([sample])
```

```
    #print(prediction)
```

```
    return prediction
```

```
    #return prediction
```

```
def test_GaussianNB(X_train, X_test, y_train, y_test,sample):
```

```
    cls = GaussianNB()
```

```
    cls.fit(X_train,y_train)
```

```
    #print('GB Training Score: %.2f' % cls.score(X_train,y_train))
```

```
    #print('GB Testing Score: %.2f'%cls.score(X_test,y_test))
```

```
    prediction = cls.predict([sample])
```

```
    return prediction
```

```
def KNeighboursClassifier(X_train, X_test, y_train, y_test,sample):
```

```
    cls = KNeighborsClassifier()
```

```
    cls.fit(X_train, y_train)
```

```
    #print('KNN Training Score: %.2f' % cls.score(X_train, y_train))
```

```
    #print('KNN Testing Score: %.2f' % cls.score(X_test, y_test))
```

```
    prediction = cls.predict([sample])
```

```
    return prediction
```

```
def test_DecisionTreeClassifier(X_train, X_test, y_train, y_test,sample):
```

```
    clf = DecisionTreeClassifier()
```

```
    clf.fit(X_train, y_train)
```

```
    #print("Decision Tree Training score:%f" %(clf.score(X_train, y_train)))
```

```
    #print('Decision Tree Testing Score: %.2f' % clf.score(X_test, y_test))
```

```
    prediction = clf.predict([sample])
```

```
    return prediction
```

```
def load_data(dataset):
```

```
    return cross_validation.train_test_split(dataset.data,dataset.target,test_size=0.25,random_state=0)
```

```
def load_vectors():
```

```
    normal = open("normalVector.txt", mode="r")
```

```
    spam = open("spamVector.txt", mode="r")
```

```
    data = []
```

```
    target = []
```

```
    count = 0
```

```
    for line in (normal.read().split("\n")):
```

```
        try:
```

```
            data.append(eval(line))
```

```
            target.append(0)
```

```
            if count == testnum:
```

```
                break
```

```
            else:
```

```
                count+=1
```

```
def load_sample():

    normal = open("normalVector.txt", mode="r")

    spam = open("spamVector.txt", mode="r")

    normalset=[]

    spamset=[]

    count = 0

    for line in (normal.read().split("\n")):

        try:

            normalset.append([eval(line),0])

            if count == testnum:

                break

            else:

                count+=1

        except:

            pass

    count = 0

    for line in (spam.read().split("\n")):

        try:

            spamset.append([eval(line),1])

            if count == testnum:

                break

            else:

                count+=1

        except:

            pass

    return spamset,normalset
```

```
def group(spams, norms):
```

```
group_1 = []
group_2 = []
group_3 = []
group_4 = []
group_5 = []

for i in range(len(spams)):
    spams[i].append(random.randint(1, 6))

    if spams[i][-1] == 1:
        group_1.append(spams[i])
    elif spams[i][-1] == 2:
        group_2.append(spams[i])
    elif spams[i][-1] == 3:
        group_3.append(spams[i])
    elif spams[i][-1] == 4:
        group_4.append(spams[i])
    elif spams[i][-1] == 5:
        group_5.append(spams[i])

for j in range(len(norms)):
    norms[j].append(random.randint(1, 6))

    if norms[j][-1] == 1:
        group_1.append(norms[j])
    elif norms[j][-1] == 2:
        group_2.append(norms[j])
    elif norms[j][-1] == 3:
        group_3.append(norms[j])
    elif norms[j][-1] == 4:
        group_4.append(norms[j])
    elif norms[j][-1] == 5:
        group_5.append(norms[j])
```

```
grouping = [group_1, group_2, group_3, group_4, group_5]
```

```
for i in range(5):
```

```
    random.shuffle(grouping[i])
```

```
return grouping
```

```
def SVM(X_train, X_test, y_train, y_test,testing):
```

```
    spamset=[]
```

```
    normset=[]
```

```
    for sample in testing:
```

```
        result=test_LinearSVC(X_train, X_test, y_train, y_test,sample[0])
```

```
        if result == [0]:
```

```
            normset.append(sample)
```

```
        elif result==[1]:
```

```
            spamset.append(sample)
```

```
    return spamset,normset
```

```
def GNB(X_train, X_test, y_train, y_test,testing):
```

```
    spamset=[]
```

```
    normset=[]
```

```
    for sample in testing:
```

```
        result=test_GaussianNB(X_train, X_test, y_train, y_test,sample[0])
```

```
        if result == [0]:
```

```
            normset.append(sample)
```

```
        elif result==[1]:
```

```
            spamset.append(sample)
```

```
    return spamset,normset
```

```
def DT(X_train, X_test, y_train, y_test,testing):
```

```

spamset=[]

normset=[]

for sample in testing:

    result=test_DecisionTreeClassifier(X_train, X_test, y_train, y_test,sample[0])

    if result == [0]:

        normset.append(sample)

    elif result==[1]:

        spamset.append(sample)

return spamset,normset


def KNC(X_train, X_test, y_train, y_test,testing):

    spamset=[]

    normset=[]

    for sample in testing:

        result=KNeighboursClassifier(X_train, X_test, y_train, y_test,sample[0])

        if result == [0]:

            normset.append(sample)

        elif result==[1]:

            spamset.append(sample)

    return spamset,normset


'''

X_train,X_test,y_train,y_test = load_data(load_vectors())


test_LinearSVC(X_train, X_test, y_train, y_test,sample)

test_GaussianNB(X_train,X_test,y_train,y_test,sample)

KNeighboursClassifier(X_train, X_test, y_train, y_test,sample)

test_DecisionTreeClassifier(X_train, X_test, y_train, y_test,sample)

'''

```

2, Main frame of decision tree in python code (python 3.6)

```
import classifier
```

```
spams,norms=classifier.load_sample()
```

```
grouping = classifier.group(spams, norms)
```

```
#impurities = grouping_training.get_impruties(grouping, spams, norms)
```

```
X_train,X_test,y_train,y_test = classifier.load_data(classifier.load_vectors())
```

```
training_data = grouping[0]
```

```
# list of algorithm at use
```

```
algos = [classifier.SVM,classifier.DT,classifier.GNB, classifier.KNC]
```

```
#print(naive_bayes.)
```

```
class Algorithms:
```

```
# a class to hold algorithms used
```

```
# with utilities included
```

```
def __init__(self, algo_call):
```

```
    self.algo_call = algo_call
```

```
def __repr__(self):
```

```
    return 'algorithms: %s' % (self.algo_call)
```

```
# if a piece of text is spam (true) according to the algo
```

```
def match(self, text):
```

```
    true_set, false_set = self.algo_call(X_train,X_test,y_train,y_test,training_data)#####
```

```
    if text in true_set: return True
```

```
    elif text in false_set: return False
```

```
class Leaf:
```

```
# holds the leaves formed as last
```

```
# class -> how many times it appears in the data set
```

```
# that reaches the leaf
```

```
def __init__(self, data):  
    self.leaf = count_labels(data)
```

```
class Node:
```

```
    # holds the nodes where tree asks question and split data
```

```
def __init__(self, algo, true_branch, false_branch):  
    self.algo = algo  
    self.true_branch = true_branch  
    self.false_branch = false_branch
```

```
    # counts how many records belongs to each labels
```

```
def count_labels(data):
```

```
    counts = {}  
  
    for rec in data:  
        label = rec[1]  
  
        if label not in counts:  
            counts[label] = 0  
        counts[label] += 1  
  
    return counts
```

```
    # function that split a data set into two subsets
```

```
def split(data, algo, X_train, X_test, y_train, y_test):
```

```
    true_set, false_set = algo.algo_call(X_train, X_test, y_train, y_test, data)
```

```
    #####
```

```
    return true_set, false_set
```

calculate the gini impurity of the set at hand

def get_gini_impurity(data):

labels = count_labels(data)

impurity = 1

for lbl **in** labels:

label_num = labels[lbl]

prob_i = label_num / float(len(data))

impurity -= prob_i**2

return impurity

def get_info_gain(true_branch, false_branch, starting_impurity):

true_impurity = get_gini_impurity(true_branch)

false_impurity = get_gini_impurity(false_branch)

true_weight = float(len(true_branch)) / float(len(true_branch) + len(false_branch))

new_impurity = true_weight*true_impurity + (1-true_weight)*false_impurity

info_gain = starting_impurity - new_impurity

return info_gain

def find_best_algo(data):

best_algo = **None**

best_gain = 0

start_gini = get_gini_impurity(data)

for al **in** algos:

algorithm = Algorithms(al)

true_set, false_set = algorithm.algo_call(X_train,X_test,y_train,y_test,data)

if the algo does not divide the data

skip the algo

if len(true_set) == 0 **or** len(false_set) == 0:

```

    continue

    info_gain = get_info_gain(true_set, false_set, start_gini)

    if info_gain >= best_gain:

        best_gain = info_gain

        best_algo = algorithm

# best_algo: an object of class Algorithms

    return best_algo, best_gain


# building tree

def build_tree(data,X_train,X_test,y_train,y_test):

    algo, gain = find_best_algo(data)

    # reach leaf

    if gain == 0:

        return Leaf(data)

    true_set, false_set = split(data, algo,X_train,X_test,y_train,y_test)

    true_branch = build_tree(true_set,X_train,X_test,y_train,y_test,)

    false_branch = build_tree(false_set,X_train,X_test,y_train,y_test,)

    return Node(algo, true_branch, false_branch)


def print_tree(node, spacing = ""):

    if isinstance(node, Leaf):

        print(spacing, 'Predict:', node.leaf)

        return

    print(spacing + str(node.algo))

    print(spacing + '-> True')

    print_tree(node.true_branch, spacing + ' ')

    print(spacing + '-> False')

    print_tree(node.false_branch, spacing + ' ')

```

```

def classify(text, node):

    if isinstance(node, Leaf):

        return node.leaf

    if node.algo.match(text):

        return classify(text, node.true_branch)

    else:

        return classify(text, node.false_branch)

# takes in Leaf object (dict containing results)

# returns printing content

def print_leaves(prediction):

    print('leaf')

    return prediction


# main

my_tree = build_tree(training_data,X_train,X_test,y_train,y_test)

print_tree(my_tree)


test_data = grouping[1]


wrong_count = 0

for row in test_data:

    result = classify(row, my_tree)

```

```
print(result)

predicted=max(result.items(), key=lambda x: x[1])[0]

print(predicted)

actual = row[1]


if actual != predicted:

    wrong_count += 1

score = 100-(wrong_count/len(test_data))*100


print(score)
```