STA242 Project

git@bitbucket.org:sxli2015/sta242-project.git

Mingyu Tian 912434971
Yilun Yang 912425744
Shuxin Li 912525987

1. Introduction to the project

There are many jobs posted on internet every day. There are so much information that we need to read and it takes a lot time for us to choose the interesting job position and fitted position. Though some job employment websites have already classified all job information to let you easily search the fitted job position by search critical word or category, there are still some drawbacks to these websites. The main drawback is that you might lose some jobs position that you are fitted to if you search job based on critical word. Because sometimes the website will only provide the jobs with critical word in their titles. For example, some positions might have titles in insurance, but actually in their job descriptions, there are many works in the descriptions which need people from statistics major.

Therefore, I hope to find a convenient way to read all job information and select the fitted job positions. There are two main steps. The first step is to scrape specified job information posted on a website. The second step is to predict the jobs' categories by models we build.

2. The procedure and idea of the project

Firstly, applying web scraping to collect job information from internet especially scraping all details in the job information such as job qualification, job title and job description to predict job's category.

Secondly, select an appropriate machine learning model and use the job data I scraped as training data to train the model by cross validation.

Thirdly, scraping new job positions updated recently and predicted their categories by the model we have trained. Compare the predicted results with true results.

3. The procedure of web scraping in the project

Web scraping which is called web data extraction as well is a computer software technique of extracting information from websites. Usually, such software programs simulate human exploration of the World Wide Web by either implementing low-level Hypertext Transfer Protocol (HTTP), or embedding a fully-fledged web browser, such as Internet Explorer or Mozilla Firefox.

In this case, I apply python to conduct web scraping. I decide to scape monster.com to collect enough training data. Monster.com is one of the most visited employment websites in the United States and one of the largest in the world.

Therefore, there are many jobs posted on the website every day so that I can get enough data. In addition, monster.com has already organized the job information based on job's category. Thus, we don't need to take time to figure out the job category. It is more convenient to collect training data correctly.

The web scraping steps is below.

Firstly I scraped the URL http://jobs.monster.com/browse to get 59 big job category's URL. Then I scraped all 59 URLs one by one. In each big job category, I scrape all job information page by page. There are 99 page's job information for each job category and each page contains 20 job position.

4. Data Cleaning

Now, I have 59 big job categories which contains 116820 job information. There are 0.013 percent job information scraped which only contains job title because of error occurring during the web scraping. This indicates that web scraping works well. However, some of scraped job information are still unformatted. For example, some garbled word occurred for decoding problem.

We should clean the data in a more neat form. We firstly deleted all the special symbols like html labels, numbers because they are useless for our classification. Next, we only extract the job summary section and qualification section out of the whole text (if the text has these parts). We did this because too many words may slow down the computation very much and we know there are only few parts in a job description that can directly help us determine which category the job should belongs to. We then combined the header with the part we extracted and split them with single blank to get a vector containing a bunch of words.

After dealing with these unformatted data, I generate a big frequency table which contains all word's frequencies. Each row of the frequency table represents a job information. Therefore, I add each job's category as response variable and each different word as predict variable and make a matrix. I can use the matrix to predict job category. Unfortunately, the matrix has relatively far more predict variables than observations. This problem not only takes much more computational time, but also make some models not work such as tree model and many other supervised learning model. We need to get rid of some words which may not be helpful to classify job category. For example "is", "a" and "an" don't contribute to classifying. We also choose to extract the content such as job qualification or job summary to get rid of predict variables. However, sometimes there are 0.013 percent job information scraped which only contains job title because of error occurring during the web scraping.

5. Model Building

After data cleaning, we begin to find the best model to predict job category. Here we use a sample of 2000 observations in order to save time.

5.1 K-Nearest-Neighborhood Method

Given that there are thousands of words of a sample of 2000 observations, methods such as tree, support vector machine and so on can not work well. Therefore, we decided to use k-nearest-neighborhood method to predict.

Here we use k=1, which means that we predict job category by the most similar one. We split jobs into training data and test data and then predict job categories of the test ones by finding the smallest distance of training jobs. If we use knn function in 'class' package, it would be very slow. Thus, we write our own knn function and use it to predict.

However, K-Nearest-Neighborhood worked not well. Only 27.8% of the test jobs could be predicted accurately. And the speed of it is not fast.

For the reason that it's not fast enough and it can not predict well, as well as some models can not work on it, we decide to do dimension reduction by K-Means Clustering Method.

5.2 K-Means Clustering

Now that there are too many variables, we consider clustering on variables to reduce the number of them. The steps are:

Firstly, transpose matrix and then clustering to 100 classes.

Secondly, calculate the mean of the variables in each class and then generate new data frame by means of the 100 classes.

Thirdly, Use two methods: support vector machine and K-Nearest-Neighborhood to predict.

After the first two steps, there are only 100 variables, which is much easier to fit models. By using k-nearest-neighborhood method on 2000 observations, the correct rate is 37.9% which has improved a lot comparing our original result without clustering.

Another method is support vector machine method. Here we use svm function in 'e1071' package. And the correct rate by svm is 43.5%.

However, there are also some drawbacks on K-Means Clustering. First, it still not fast enough. Second, the correct rate here is not high enough to use. Hence, we tried another mothod: Principle Component Analysis to do dimension deduction.

5.3 Principle Component Analysis

Here we use PCA to do dimension reduction. From the result of PCA, we know the first 60 principle components can explain more than 90% variance. We then generate a new data frame containing the PCs and also the job class.

We tried Random Forest, Bagging, SVM to do the classification for this approach. By using random forest method on 2000 observations, the correct rate is 61.3%. By using bagging method in 'ipred' package, the correct rate is 58.7%. Finally, the correct rate for SVM is 53.5%.

6. Model Comparison

After trying K-Nearest Neighborhood, K-Means Clustering and PCA, we decided to use PCA to do dimension reduction for three reasons. First it's the fastest method. Second, it predict much better than the other two methods. Third, we can try more models after dimension reduction by PCA.

We tried Random Forest, Bagging, SVM to do the classification for this approach. By using random forest method on 2000 observations, the correct rate is 61.3%. By using bagging method in 'ipred' package, the correct rate is 58.7%. Finally, the correct rate for SVM is 53.5%.

7. Conclusion

By comparison three different models after PCA, random forest seems to be our best model, we then extend the process to 5000 samples with 3500 training data and 1500 test data. The correct rate increases up to 65%. We may conclude sample size is important to our prediction precision and we are quite sure the precision will be even higher if we use more samples.

8. Further Exploration

Our work here is useful, however, the predication error is still not high enough. If we have opportunity to do further exploration, we may try other ways to improve correct rate. Maybe we can through all data in and then fit model and find method to improve running time. Maybe there are other methods of predication which have higher correct rate and we can try to find them.

Appendix

```
####################  Web Scraping by python  ############################
import gzip
import re
import urllib.request
import bs4
from collections import deque
import os

def download_url(url):
    header = {
        'User-Agent':  'Mozilla/5.0  (Windows  NT  6.1;  rv:37.0)  Gecko/20100101
Firefox/37.0',
        'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
        'Accept-Encoding': 'gzip, deflate',
        'Connection': 'keep-alive'
        }
    req = urllib.request.Request(url, headers = header)
    con = urllib.request.urlopen(req, timeout = 100)
    doc = con.read()
    con.close()
    data = gzip.decompress(doc)
    data = data.decode('ascii', errors = 'ignore')
    return data

# delete blank and \n
def delete_blank(data):
    pat = re.compile('\n')
    data_s = pat.sub('', data)
    return data_s

# find all urls
def extract_urls(data):
    pat = re.compile('href=\"(.+?)\"')
    urls = pat.findall(data)
    return urls

# find all odds urls
def extract_even_urls(data):
    i = 1
    while i<len(data):
        data.pop(i)
```

```
        i = i+1
    return data

queue1 = deque()
queue2 = deque()
queue3 = deque()
queue4 = deque()
queue5 = deque()
queue6 = deque()

# extract the first floor html
url = 'http://jobs.monster.com/browse'
data = download_url(url)
data_n = delete_blank(data)
# extract word in the pattern
pat_11 = re.compile('<h2 class="fnt5">Browse Jobs By Category</h2>(.+?)Browse more<span
class="sr-only">Web Jobs</span></a></div>')
str_url_11 = pat_11.findall(data_n)[0]
# extract url of big categories
urls_1 = extract_urls(str_url_11)
urls_1 = extract_even_urls(urls_1)
pat_12 = re.compile('class=\"fnt4\" title=\"(.+?)\"')
titles_1 = pat_12.findall(str_url_11)


for i in range(0, len(urls_1)):
    try:
        data = download_url(urls_1[i])
        print("extract ------->"+" Big Catergory "+  titles_1[i])
    except:
        print('wrong!!!!!!' + '---->' + titles_1[i])
        queue1.append(urls_1[i])
        continue
    # extract the second floor html
    directory = 'E:/'+ 'STA242PROJECT/'+titles_1[i] +'/'
    if not os.path.exists(directory):
        os.makedirs(directory)
    #
    pg = 99
    for k in range(pg):
        urls_3 = urls_1[i] + '?page=' + str(k + 1)
        try:
            data = download_url(urls_3)
            print("extract----->"+ titles_1[i]+"||"+' page '+ str(k+1))
```

```
except:
    print("wrong!!!"+titles_1[i]+"||"+' page '+ str(k+1))
    queue2.append(urls_3)
    continue
#
linkre = re.compile('href=\"(.+?jobview.+?|.+?job-openings.+?)\"')
webls = linkre.findall(data)
pattern = re.compile('&#39;')
webls = [pattern.sub('\'', x) for x in webls]
linkre1 = re.compile('datetime=\"(.+?)\"')
datetime = linkre1.findall(data)
linkre2 = re.compile('itemprop=\"datePosted\">(.+?)</time>\r')
itemprop = linkre2.findall(data)
linkre3 = re.compile('<span itemprop=\"title\">(.+?)</span></a>')
title_job = linkre3.findall(data)
# extract each job position in a page
for m in range(len(webls)):
    urlnew = webls[m]
    try:
        data = download_url(urlnew)
    except:
        print('wrong1!'+ titles_1[i]+"||"+' page '+ str(k+1)+' Job' + str(m+1))
        queue3.append(urlnew)
        save_path = directory + 'page'+str(k+1)+'_'+str(m+1)+'.txt'
        f = open(save_path, 'a', encoding = 'utf-8')
        f.write(title_job[m]+ '\n' + datetime[m] +' '+ itemprop[m])
        f.close()
        continue
    # if soup doesn't work then just extract job title and record it
    try:
        soup =bs4.BeautifulSoup(data)
        jobname = soup.title.string
    except:
        print('wrong4!'+ titles_1[i]+"||"+' page '+ str(k+1)+' Job' + str(m+1))
        queue6.append(urlnew)
        save_path = directory + 'page'+str(k+1)+'_'+str(m+1)+'.txt'
        f = open(save_path, 'a', encoding = 'utf-8')
        f.write(title_job[m]+ '\n' + datetime[m] +' '+ itemprop[m])
        f.close()
        continue
    # if the code doesn't work then extract job title and record it
    try:
        jobbody = soup.find(id = 'TrackingJobBody')
        jobtext = jobbody.getText()
```

```
                content = jobtext + '\n' + datetime[m] +' '+ itemprop[m]
                save_path = directory + 'page'+str(k+1)+'_'+str(m+1)+'.txt'
                f = open(save_path, 'a', encoding = 'utf-8')
                f.write(jobname +'\n' + content)
                f.close()
                print('extract----->'+ titles_1[i]+"||"+' page '+ str(k+1)+' Job' +
                                                                        str(m+1))
            except:
                try:
                    data_n = delete_blank(data)
                    pat = re.compile('<div class=\"jobview-section\">(.+?)</div><!-
                                                        - ./jobview-section -->')
                    content = pat.findall(data_n)[0]+ '\n' + datetime[m] +' '+
                                                                      itemprop[m]
                    save_path = directory + 'page'+str(k+1)+'_'+str(m+1)+'.txt'
                    f = open(save_path, 'a', encoding = 'utf-8')
                    f.write(jobname +'\n' + content)
                    f.close()
                    print('extract----->'+ titles_1[i]+"||"+' page '+ str(k+1)+' Job'
                                                                         +
str(m+1))
                except:
                    try:
                        save_path = directory + 'page'+str(k+1)+'_'+str(m+1)+'.txt'
                        f = open(save_path, 'a', encoding = 'utf-8')
                        f.write(jobname+ '\n' + datetime[m] +' '+ itemprop[m])
                        f.close()
                        print('wrong2! '+ titles_1[i]+'||'+' page '+ str(k+1)+' Job' +

str(m+1))
                        queue4.append(urlnew)
                    except:
                        save_path = directory + 'page'+str(k+1)+'_'+str(m+1)+'.txt'
                        f = open(save_path, 'a', encoding = 'utf-8')
                        f.write(title_job[m]+ '\n' + datetime[m] +' '+ itemprop[m])
                        f.close()
                        print('wrong3! '+ titles_1[i]+"||"+' page '+ str(k+1)+' Job' +

str(m+1))
                        queue5.append(urlnew)
                        continue
```

```
############################ clean data ################################
setwd("F:/First20")
f = list.files()
g = list.files(f, full.names = TRUE)
rawdata = list()
for(i in 1:length(g)){
  a = readLines(g[i],encoding = "UTF-8")
  a = a[-length(a)]
  index = which(grepl("[[:alpha:]]",a))
  a = a[index]
  b = gsub("<[/[:alpha:]]+>", "", a)
  b = gsub("[[:digit:]]","",b)
  b = tolower(b)
  d = strsplit(b[1], " ")[[1]]
  d = tolower(d[d!=""])
  if(any(grepl("job",d))) d = d[1:(which(grepl("job",d))-1)]

  if(length(b) == 1) rawdata[[i]] = d
  else{
    body = b[2]
    if(grepl('summary:',body)){
      c = strsplit(body, 'summary:')[[1]][2]
      b[2] = strsplit(c, ':')[[1]][1]
    }
    else if (grepl('qualifications:', body)){
      c = strsplit(body, 'qualifications')[[1]][2]
      b[2] = strsplit(c, ':')[[1]][1]
    }
    else b[2] = ''

    e = strsplit(b[2], ' ')[[1]]
    rawdata[[i]] = c(d,e)
    rawdata[[i]] = gsub("[[:punct:]]","",rawdata[[i]])
    rawdata[[i]] = rawdata[[i]][rawdata[[i]]!='']
  }
}
names(rawdata) = g

############################ freq table ############################
#use a sample
load("D:/rawdata.rda")
m = 5000
set.seed(2)
index = sample(length(rawdata), m)
```

```
train = rawdata[index]
trainname = names(train) #names of files
traintable = sapply(train, table)
allwords = lapply(traintable, names)
allword = Reduce(union, allwords) # get all the words
#generate the freq data frame
#give job class first
filename = sapply(trainname, strsplit, "/")
jobclass = sapply(filename, "[", 1)
jobclass = gsub(" Jobs", "", jobclass)
a = matrix(numeric(m*length(allword)), m)
rownames(a) = trainname
colnames(a) = allword
wordfreq = as.data.frame(a)
wordfreq = cbind(jobclass, wordfreq)
for(i in 1:m){
  wordfreq[i,names(traintable[[i]])] = traintable[[i]]
}

############################ KNN ################################
#use knn method
wordscale = scale(wordfreq[,-1])#scale first
n = nrow(wordfreq)
#split in to train and test data
train = wordscale[1:(n/2),]
test = wordscale[(n/2+1):n,]
#get the distance between train and test
distance = as.matrix(dist(wordscale, method =
"canberra"))[-(1:nrow(train)), (1:nrow(train))]
#use myknn function to find the nearst one
index = myknn(distance, 1)
#correct rate
correctrate = mean(wordfreq[(n/2+1):n,1]==wordfreq[1:(n/2),1][index])

#my knn function
myknn = function(distance, k){
  a = matrix( , nrow(distance), k)
  for(i in 1:nrow(distance)){
    a[i,] = order(distance[i,])[1:k]
  }
  return(a)
}

############### clustering and supervised model ###################
```

```
clustering varabiles to reduce number of variables
km.out=kmeans(t(as.matrix(wordfreq[,-1])),100,nstart=20)
a = matrix(numeric(nrow(wordfreq)*100), nrow = nrow(wordfreq))
for(i in 1:100){
  a[,i] = rowMeans(wordfreq[,-1][,which(km.out$cluster==i), drop = FALSE])
}
a = as.data.frame(a)
a$jobclass = wordfreq$jobclass
#split into train and test data
traincl = a[1:(n/2),]
testcl = a[(n/2+1):n,]
#svm method
library(e1071)
svm.fit = svm(jobclass~., data = traincl)
svmpred = predict(svm.fit, testcl, type = "class")
mean(svmpred==testcl$jobclass)

#knn method
library(class)
knn.fit = knn(traincl[,-ncol(traincl)], testcl[,-ncol(testcl)],
traincl$jobclass)
mean(knn.fit==testcl$jobclass)



###################### pca machine learning methods ##################
##PCA dimension reduction
pc = prcomp(wordfreq[,-1])
eigenvalue = (pc$sdev)^2
cumprop = sum(eigenvalue[1:60])/sum(eigenvalue)
cumprop ###60 principle components are enough, explain 90% variance
scrs = pc$x
wordpca = matrix(nrow = 2000, ncol = 60)
wordpca[1:2000,1:60] = scrs[1:2000,1:60]
wordpca = as.data.frame(wordpca)
wordpca$Y = wordfreq$jobclass ##make a new data frame with jobclass as Y
and 60 principle components

###Random Forest
library('randomForest')
ranFst=randomForest(Y~., data = wordpca[1:1000,],  ntree = 100)
error.ranFst = ranFst$err.rate ###prediction error rate given by OOB
estimation
error.ranFst[100,1]
```

```
pred.rf = predict(ranFst,newdata=wordpca[1001:2000,-61])
mean(pred.rf == wordpca[1001:2000,61])

###Bagging
library('ipred')
bagging = bagging(Y~., data = wordpca[1:1000,], nbagg = 100,coob = TRUE)
err.bag = bagging$err
err.bag
pred.bag = predict(bagging, newdata = wordpca[1001:2000,-61])
mean(pred.bag == wordpca[1001:2000,61])

###SVM
library(e1071)
svm.fit = svm(Y~., data = wordpca[1:1000,])
svmpred = predict(svm.fit, wordpca[1001:2000, -61], type = "class")
mean(svmpred == wordpca$Y[1001:2000])
```