GLM on Text Sparse Matrix

Shan Chen
Jan 2016

This file records the thought process of predicting people's review of albums with information such as dimension of photos and words used in photo cation. It will basically contain four parts:

1. EDA

Loading packages and data set

```
library('knitr')
library('ggplot2')
library('gridExtra')
library('sqldf')
```

```
## Loading required package: gsubfn
```

```
## Loading required package: proto
```

```
## Loading required package: RSQLite
```

```
## Loading required package: DBI
```

```
library('dplyr')
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:gridExtra':
##
##     combine
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library('stringi')
library('Matrix')
library('glmnet')
```

```
## Loading required package: foreach
```

```
## Loaded glmnet 2.0-5
```

```
train<-read.csv("C:\\Users\\Shan\\Desktop\\Photo Quality Prediction\\training.csv")
```

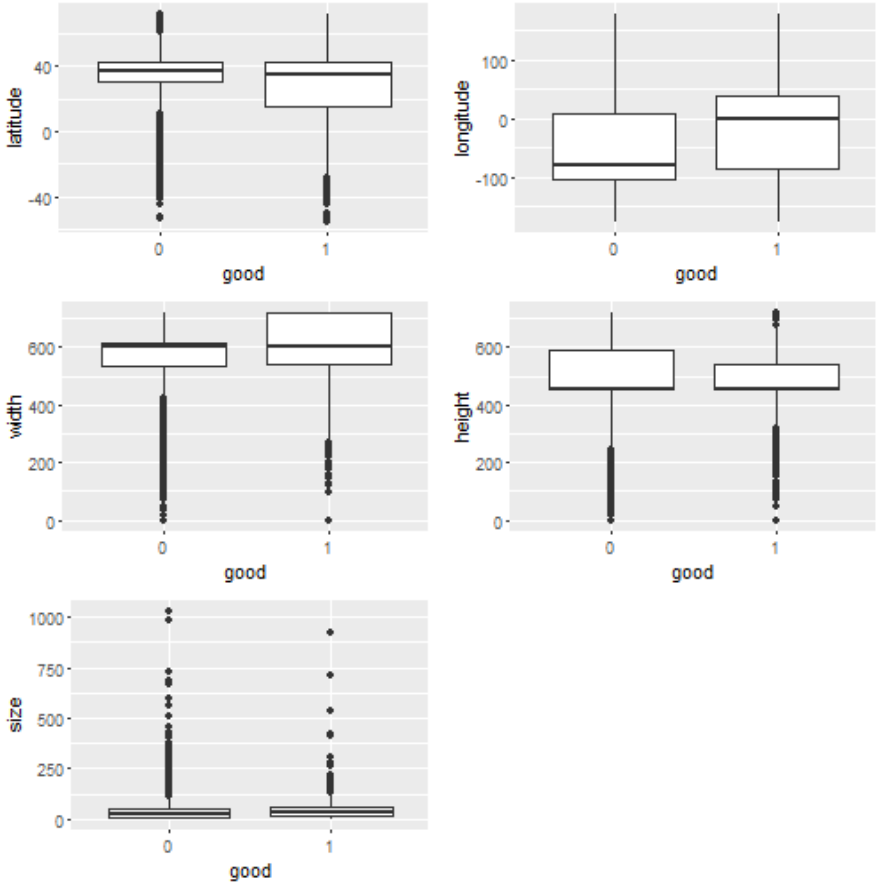
1.1 Non-text Features Visualization

Converting "good" data type from int to factor

```
train$good <- as.factor(train$good)
```

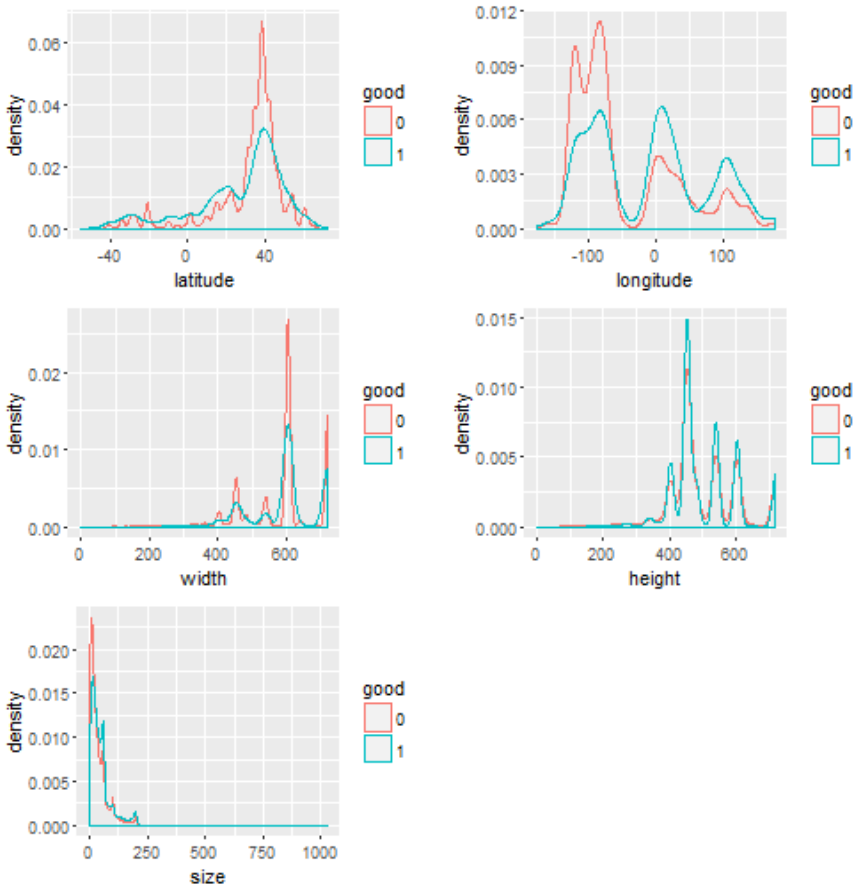
Boxplots:

```
g1=ggplot(train, aes(x=good, y=latitude)) + geom_boxplot()
g2=ggplot(train, aes(x=good, y=longitude)) + geom_boxplot()
g3=ggplot(train, aes(x=good, y=width)) + geom_boxplot()
g4=ggplot(train, aes(x=good, y=height)) + geom_boxplot()
g5=ggplot(train, aes(x=good, y=size)) + geom_boxplot()
a=grid.arrange( g1,g2,g3,g4,g5, nrow=3,ncol=2)
```



Density plots:

```
g1=ggplot(train, aes(x=latitude, color=good)) + geom_density()
g2=ggplot(train, aes(x=longitude, color=good)) + geom_density()
g3=ggplot(train, aes(x=width, color=good)) + geom_density()
g4=ggplot(train, aes(x=height, color=good)) + geom_density()
g5=ggplot(train, aes(x = size,color=good)) + geom_density()
a=grid.arrange( g1,g2,g3,g4,g5, nrow=3,ncol=2)
```



From above plots, it looks like the distribution of non-text variable for both good and not good albums are similar, it may mean that, in terms of latitude variable, for a given latitude area, it is hard to judge if the album would be viewed good or not only through the latitude effect. This is possible due to lack of other vital variables, e.g., longitude, size, text information, or it's simply because there is no obvious linear relationship between latitude and goodness of albums which might make sense if it's the situation that any negative and positive latitude area both have good albums and bad albums (lack of other more specific information).

1.2 Logistic GLM on only Non-text Features

In order to address our guess of lack of other important features, we first add all the non-text features into a glm model (since response variable--good has two values).
Just out of curiousity, I would like to know if glm model would perform well if I only used non-text predictors.

```
#subset only non-text information out of train date set
train<-train[,c(1:6,10)]
fit1 <- glm(good ~latitude+longitude+width+height+size , family = "binomial", data=train)
summary(fit1)
```

```
##
## Call:
## glm(formula = good ~ latitude + longitude + width + height +
##     size, family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4663  -0.7752  -0.6534   1.1837   2.2307
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.8235829  0.0953425 -19.127  < 2e-16 ***
## latitude    -0.0034867  0.0005674  -6.145 8.02e-10 ***
## longitude    0.0048920  0.0001475  33.158  < 2e-16 ***
## width        0.0013556  0.0001130  12.001  < 2e-16 ***
## height       0.0001489  0.0001146   1.299    0.194
## size         0.0038511  0.0002653  14.516  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 46324  on 40261  degrees of freedom
## Residual deviance: 44366  on 40256  degrees of freedom
## AIC: 44378
##
## Number of Fisher Scoring iterations: 4
```

It seems all variables except for height are significant predictors for goodness of albums meaning that glm kind of addressed our feature linearity problem once we add more others features in.

Calculate in-sample error rate

Error rate

```
train.predict <- predict(fit1,type = "response")
predict.good <- ifelse(train.predict >= 0.50, 1,0)
predict.match <- ifelse(predict.good == train$good,1,0)
correct_rate_insample<-sum(predict.match)/nrow(train)
error_rate_insample <- 1-correct_rate_insample
error_rate_insample
```

```
## [1] 0.2635736
```

Error rate is 0.2635736 which is kinda low, then I consider adding the text information and conduct a glmnet model to deal with text variable.

2. 1 gram Vectorization and Sparse Matrix Generation

2.1 Why Need Text Information

```
train<-read.csv("C:\\Users\\Shan\\Desktop\\Photo Quality Prediction\\training.csv")
test<-read.csv("C:\\Users\\Shan\\Desktop\\Photo Quality Prediction\\test.csv")
complete <-bind_rows(train, test)# Union the data together vertically
```

```
## Warning in rbind_all(x, .id): Unequal factor levels: coercing to character

## Warning in rbind_all(x, .id): Unequal factor levels: coercing to character

## Warning in rbind_all(x, .id): Unequal factor levels: coercing to character
```

```
head(complete)
```

```
## Source: local data frame [6 x 10]
##
##      id latitude longitude width height  size                name
##   (int)    (int)     (int) (int)  (int) (int)               (chr)
## 1     1       45        16   604    453    31            454 1659
## 2     2       21       -87   720    534    43            2068 483
## 3     3       38       -97   720    540    71                 802
## 4     4       38      -122   604    453    24
## 5     5      -29        24   720    540    13             1766 20
## 6     6      -21        56   604    453    58 1380 1441 1832 1559
## Variables not shown: description (chr), caption (chr), good (int)
```

Our interests is to find out which words in name, description and caption of an album would tend to give a satisfying quality result and which ones would not, in order to address such a problem, I first extract out all of the words in those three columns to generate quite a lot new word tokens features(1-gram, only one word), and fill in the cell with 1 if the album contains the corresponding word and 0 if not.

2.2 R Functions for Sparse Matrix Generation

I use the below R functions to generate such a **large sparse matrix**.

```
name<-complete$name
description<-complete$description
caption<-complete$caption

#1 gram Vectorization
tokens <- stri_split_fixed(name, ' ')
token_vector <- unlist(tokens)
bagofwords_name <- unique(token_vector)
n_tokens <- sapply(tokens, length)
i <- rep(seq_along(n_tokens), n_tokens)
j <- match(token_vector, bagofwords_name)
sparsematrix_name <- sparseMatrix(i=i, j=j, x=1L)
colnames(sparsematrix_name) <- paste(bagofwords_name,'in Name')

tokens <- stri_split_fixed(description, ' ')
token_vector <- unlist(tokens)
bagofwords_description <- unique(token_vector)
n_tokens <- sapply(tokens, length)
i <- rep(seq_along(n_tokens), n_tokens)
j <- match(token_vector, bagofwords_description)
sparsematrix_description <- sparseMatrix(i=i, j=j, x=1L)
colnames(sparsematrix_description) <- paste(bagofwords_description,'in Description')

tokens <- stri_split_fixed(caption, ' ')
token_vector <- unlist(tokens)
bagofwords_caption <- unique(token_vector)
n_tokens <- sapply(tokens, length)
i <- rep(seq_along(n_tokens), n_tokens)
j <- match(token_vector, bagofwords_caption)
sparsematrix_caption <- sparseMatrix(i=i, j=j, x=1L)
colnames(sparsematrix_caption) <- paste(bagofwords_caption,'in Caption')
#cbind three sparse matrix with other non-text columns
x<-cbind(as.matrix(complete[,c(1:6)]),sparsematrix_name,sparsematrix_description,sparsematrix_caption)
dim(x)#dimension is correct, x is the complete sparse matrix without the good column
```

```
## [1] 52262  6299
```

```
#have a look at a little piece of the matrix
x[1:20,1:20]
```

```
## 20 x 20 sparse Matrix of class "dgCMatrix"
```

```
##    [[ suppressing 20 column names 'id', 'latitude', 'longitude' ... ]]
```

```
##
##  [1,]  1  45   16 604 453  31 1 1 . . . . . . . . . . .
##  [2,]  2  21  -87 720 534  43 . . 1 1 . . . . . . . . .
##  [3,]  3  38  -97 720 540  71 . . . . 1 . . . . . . . .
##  [4,]  4  38 -122 604 453  24 . . . . . 1 . . . . . . .
##  [5,]  5 -29   24 720 540  13 . . . . . . 1 1 . . . . .
##  [6,]  6 -21   56 604 453  58 . . . . . . . . 1 1 1 1 . .
##  [7,]  7  42  -73 540 720   2 . . . . . . . . . . . 1 1
##  [8,]  8  26  -80 604 452  49 . . . . . . . . . . . . .
##  [9,]  9  15  100 604 453  12 . . . . . . . . . . . . .
## [10,] 10  41  -74 640 478 361 . . . . . . . . . . . . 1
## [11,] 11  39  -97 604 453  76 . . . . . . . . . . . . .
## [12,] 12  52    6 453 604  21 . . . . . . . . . . . . .
## [13,] 13  41  -87 299 448  10 . . . . . 1 . . . . . . .
## [14,] 14 -29   24 604 453  80 . . . . . . . . . . . . .
## [15,] 15 -10  -76 604 453  30 . . . . . . . . . . . . .
## [16,] 16  34  -84 403 604  47 . . 1 . . . . . . . . . .
## [17,] 17  39  -99 720 540 128 . . . . . 1 . . . . . . .
## [18,] 18  33 -113 453 604  18 . . . . . . . . . . . . .
## [19,] 19  10  -84 604 453  89 . . . . . . . . . . . . .
## [20,] 20  -5  120 428 253  10 . . . . . . . . . . . . .
```

Now we have the sparse matrix that containing 6299 columns, we are finally ready to conduct a powerful GLMNET algorithm to this large sparse matrix.

3. GLMNET Modeling and Validation

Split x into two parts; one is train the other is test.

```
x_train <- x[1:40262,]
x_test <- x[40263:52262,]
```

3.1 GLMNET Modeling

```
fit2=glmnet(x_train,train$good,family="binomial")
```

Use x_train data to predict train$good and then calculate in-sample error rate.

```
train.predict <- predict(fit2,newx=x_train, s=0.001,type = "response")#s is the value of lambda
predict.good <- ifelse(train.predict >= 0.50, 1,0)
predict.match <- ifelse(predict.good == train$good,1,0)
correct_rate_insample<-sum(predict.match)/nrow(train)
error_rate_insample <- 1-correct_rate_insample
error_rate_insample
```

```
## [1] 0.1717004
```

Error rate is 0.1717004 which is better than only using non-text information as predictors.

3.2 Cross Validation

We see that our model's performance has been improved after we take in text predictors, however I think we still need a bit more work on applying the model to some out-of sample and check the consistence of the model.

```
#cross validation (carve out x_train, the first 1 fouth of the x_train set is treated as the validation set)
valid <- train[1:10000,]
train <- train[10001:40262,]
x_valid <- x_train[1:10000,]
x_train <- x_train[10001:40262,]

fit3=glmnet(x_train,train$good,family="binomial")
```

Then calculate out of sample error rate of validation part of data.

```
valid.predict <- predict(fit3,newx=x_valid, s=0.001,type = "response")#s is the value of lambda
predict.good <- ifelse(valid.predict >= 0.50, 1,0)
predict.match <- ifelse(predict.good == valid$good,1,0)
correct_rate_outofsample<-sum(predict.match)/nrow(valid)
error_rate_outofsample <- 1-correct_rate_outofsample
error_rate_outofsample
```

```
## [1] 0.2256
```

Out of sample error rate is a bit higher than in sample's possibily due to the train sample size becomes smaller but is still lower than our first model (non-text model--fit1)'s error rate.

4. Prediction and Submission

4.1 Prediction

Predict the x_test set using our fist whole x_train glmnet model--fit2

```
test$predict <- predict(fit2,newx=x_test, s=0.001,type = "response")#s is the value of lambda
test$predict.good <- ifelse(test$predict >= 0.50, 1,0)
```

4.2 Submission

Choose only two columns (id and prediction result) as our final entry; give column names and write to a CSV file.

```
submission <- cbind(test$id,test$predict.good)
colnames(submission) <-c("id",'good')
write.csv(submission, file = 'submission.csv', row.names = F)
```

Have a look at the format of our submission.

```
head(submission)
```

```
##         id good
## [1,] 40265    0
## [2,] 40266    0
## [3,] 40267    0
## [4,] 40268    0
## [5,] 40269    1
## [6,] 40270    0
```

Later on I will work on some other feature engineering tools such as PCA and SVD, in order to reduce dimensionality of large sparse matrix to see if such model manipulation could lower a bit the error rate.