Text Mining: Detecting Insults in Social Commentary

—Kaggle 2012

Yuan, Jumao

04/28/2015

Contents

```
> getwd()

[1] "/Users/jumaoyuan/Desktop/All_Proj_7152/Kaggle_Detect_Insult"
```

All documents and materials can be downloaded in my **github** repository https://github.com/jyuan4/Final_Project.

1 Abstract

The problem on Kaggle (https://www.kaggle.com/c/detecting-insults-in-social-commentary) is a single-classifier problem. Our goal is to judge if a comment is a insult or not. Here, we use TF-IDF, shot for term frequency-inverse document frequency (http://en.wikipedia.org/wiki/Tf-idf), is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. Here, we use this technique to convert characters (comments) into sparse matrix, which is so called term-document matrix. In order to optimize this sparse matrix, we need decrease words variables as more as we can considering computation time and prediction accuracy. In this way, we used regular expression (http://en.wikipedia.org/wiki/Regular_expression), (abbreviated regex or regexp and sometimes called a rational expression) is a sequence of characters that forms a search pattern, mainly for use in pattern matching with strings, or string matching, i.e. "find and replace"-like operations. For example, "fuck" and "fuckkkkk" or "fuckk" are the same insulting word, and we need treat them as a single variable in the sparse matrix. After the sparse matrix is done, we applied multiple classification methods, like Logistic (Glmnet in **R**) Regression, Decision Tree for binary classification problem, Naive Bayes, Support Vector Machine (SVM), K-Nearest Neighbors, and some ensemble methods, such as Radom Forest and Gradient Boosting Method (GBM). The evaluation criteria for each model is AUC score or ROC curve. By comparing AUC score and computation time, we finally got the results and the best model.

2 Introduction

Predict whether a comment posted during a public discussion is considered insulting to one of the participants.

The challenge is to detect when a comment from a conversation would be considered insulting to another participant in the conversation. Samples could be drawn from conversation streams like news commenting

sites, magazine comments, message boards, blogs, text messages, etc.

The idea is to create a generalizable single-class classifier which could operate in a near real-time mode, scrubbing the filth of the internet away in one pass.

3 Data Description

All data can be downloaded from `https://www.kaggle.com/c/detecting-insults-in-social-commentary`. The data consists of a label column followed by two attribute fields.

This is a single-class classification problem. The label is either 0 meaning a neutralcomment, or 1 meaning an insulting comment (neutral can be considered as notbelonging to the insult class. Your predictions must be a real number in the range [0,1] where 1 indicates 100% confident prediction that comment is an insult.

The first attribute is the time at which the comment was made. It is sometimes blank, meaning an accurate timestamp is not possible. It is in the form "YYYYMMDDhhmmss" and then the Z character. It is on a 24 hour clock and corresponds to the localtime at which the comment was originally made.

The second attribute is the unicode-escaped text of the content, surrounded by double-quotes. The content is mostly english language comments, with some occasional formatting.

4 Evaluation

Entries will be evaluated using the area under the **receiver operator curve** (AUC). AUC was first used by the American army after the attack on Pearl Harbour, to detect Japanese aircraft from radar signals.

Today, it is a commonly used evaluation method for binary choose problems, which involve classifying an instance as either positive or negative. Its main advantages over other evaluation methods, such as the simpler misclassification error, are:

1. It's insensitive to unbalanced datasets (datasets that have more installeds than not-installeds or vice versa).

2. For other evaluation methods, a user has to choose a cut-off point above which the target variable is part of the positive class (e.g. a logistic regression model returns any real number between 0 and 1 - the modeler might decide that predictions greater than 0.5 mean a positive class prediction while a prediction of less than 0.5 mean a negative class prediction). AUC evaluates entries at all cut-off points, giving better insight into how well the classifier is able to separate the two classes.

Understanding AUC

To understand the calculation of AUC, a few basic concepts must be introduced. For a binary choice prediction, there are four possible outcomes:

- true positive - a positive instance that is correctly classified as positive;
- false positive - a negative instance that is incorrectly classified as positive;
- true negative - a negative instance that is correctly classified as negative;
- false negative - a positive instance that is incorrectly classified as negative);

These possibilities can be neatly displayed in a confusion matrix:

	P	N
P	true positive	false positive
N	false positive	true positive

The true positive rate, or recall, is calculated as the number of true positives divided by the total number of positives. When identifying aircraft from radar signals, it is proportion that are correctly identified.

The false positive rate is calculated as the number of false positives divided by the total number of negatives. When identifying aircraft from radar signals, it is the rate of false alarms.

5 Data Preprocessing

```
> #import data set
> data1 <- read.csv("data/train.csv")
```

```
> #delete date
> data <- data1[,-c(2)]
> write.csv(data, 'data.csv')
> names(data)

[1] "Insult"  "Comment"

> dim(data)

[1] 3947    2

> data[4,2]

[1] "listen if you dont wanna get married to a man or a women DONT DO IT. what would it bother
you if gay people got married stay in your lane do you let them do them. And your god is so nice
but quick to judg if your not like him, thought you wasnt suppose to judge people."

3935 Levels: "_BeautifulFlaws okay I'm putting my throw back shirt &amp;amp; cow girl boots on
seriously I'm coming ." ...

> #check missing data
> length(which(data$Insult==0))+length(which(data$Insult==1))-nrow(data)

[1] 0

> #split data sets
> smp_size <- floor(0.70 * nrow(data))
> ## set the seed to make your partition reproductible
> set.seed(123) #fix training and testing sampling
> train_ind <- sample(seq_len(nrow(data)), size = smp_size)
> training <- data[train_ind, ]
> testing <- data[-train_ind, ]
> com.train <- training$Comment
> com.test <- testing$Comment
```

5.1 Missing values and typos

There are two useful variables and 3947 observations in datasets. The response variable is binary, "Is Insult" or "Is Not Insult".

There is no missing value in the datasets, but we need do token and regular expression analysis.

5.2 TF-IDF

Comments	**Insult**
"Either you are fake or extremely stupid...maybe both..."	yes
"@tonnyb Or they just don't pay attention"	no
"You with the 'racist' screen name\\n\nYou are a PieceOfShit....."	yes
"your such a dickhead..."	yes
`http://www.youtube.com/watch?v=tLYLLPHKRU4`	no
"You are a liar."	no

TF-IDF is used to calculate the importance of the word to each document and the weights are recorded in the new matrix. But in this case, another method in the library (library(tm) in **R**) is used to transform the sum count matrix to a tf-idf representation. Then Logistic Regression are applied on the new matrix to do the prediction.

There may be thousands of words included in the text, which makes the dimension of the feature set is quite high. Then it is worth figuring out if there is any way to reduce complexity of the problem. In the following two sections the author will aim at how to reduce dimensions by removing or merging some features.

5.3 Stemming

Another way to reduce the dimensions is to stem the words. In English, many words derive from the same stem. for instance , the words "use", "uses" "used", "useful", "using" have the same stem "use". Replacing the words with stem will reduce the size of the words in the feature set, and such a process to reduce the words to their stem is called "Stemming". There is a library called "stemDocument" in **R** providing interfaces to solve this problem. The author uses api from nltk.stem to finish the stemming process and

reforms the sum count matrix, then build logistic regression on the new data set. It should be noted that PCA and Stemming can be used at the same time, so there are two models created in this section.

5.4 Text Parsing

Like the comment "You with the 'racist' screen name\\n\nYou are a PieceOfShit 012.....", we need convert upper case to lowercase, delete \n, delete numbers (use regular expression **gsub("[^a-zA-Z])"** in **R**) and adjust space, etc. Pieces of R codes are shown as follows:

```
dd <- Corpus(VectorSource(docs))
dd <- tm_map(dd, stripWhitespace) % Eliminating Extra White Spaces
dd <- tm_map(dd, tolower) % Convert to Lower Case
dd <- tm_map(dd, removePunctuation) % Remove Punctuations
dd <- tm_map(dd, removeWords, stopwords("english")) % Remove stopwords
%dd <- tm_map(dd, stemDocument)
dd <- tm_map(dd, removeNumbers) % Remove numbers
dd <- tm_map(dd, stemDocument,language = 'english') % Do Stemming
```

5.5 TermDocumentMatrix

A document-term matrix or term-document matrix is a mathematical matrix that describes the frequency of terms that occur in a collection of documents. In a document-term matrix, rows correspond to documents in the collection and columns correspond to terms. There are various schemes for determining the value that each entry in the matrix should take. One such scheme is tf-idf. They are useful in the field of natural language processing[5].

For example, for these two sentences,

- D1 = "I like databases"
- D2 = "I hate databases",

	I	like	hate	databases
D1	1	1	0	1
D2	1	0	1	1

In **R**, an object of class TermDocumentMatrix or class DocumentTermMatrix (both inheriting from a simple triplet matrix in package slam) containing a sparse term-document matrix or document-term matrix. The attribute Weighting contains the weighting applied to the matrix.
We use "TermDocumentMatrix" in **R** to constructs or coerces to a term-document matrix or a document-term matrix. The matrix is a parse matrix and will be used for later binary classification analysis with response variable "Insult".

5.6 Feature Extraction

The sparse matrix is huge with high dimensional variables. Lots of words will be generated in the "TermDocumentMatrix". Our main task is to downsize the high dimensional sparse matrix since the stack pointer of memory in **R** is limited (500000).

5.7 Split into training and testing datasets

We randomly split data into 70% test datasets and 30% training datasets and repeate all models with a few iterations.

6 Build Moldes

R code can be accessible on my github repository `https://github.com/jyuan4/Final_Project`.

6.1 Classification Tree

```
library(tree)
train.dat$Insult <- factor(train.dat$Insult)
# Start the clock!
ptm <- proc.time()
tree.fit <- tree(Insult~.,data=train.dat) #takes time
# Stop the clock
proc.time() - ptm
summary(tree.fit)
plot(tree.fit)
text(tree.fit, pretty=0)
```

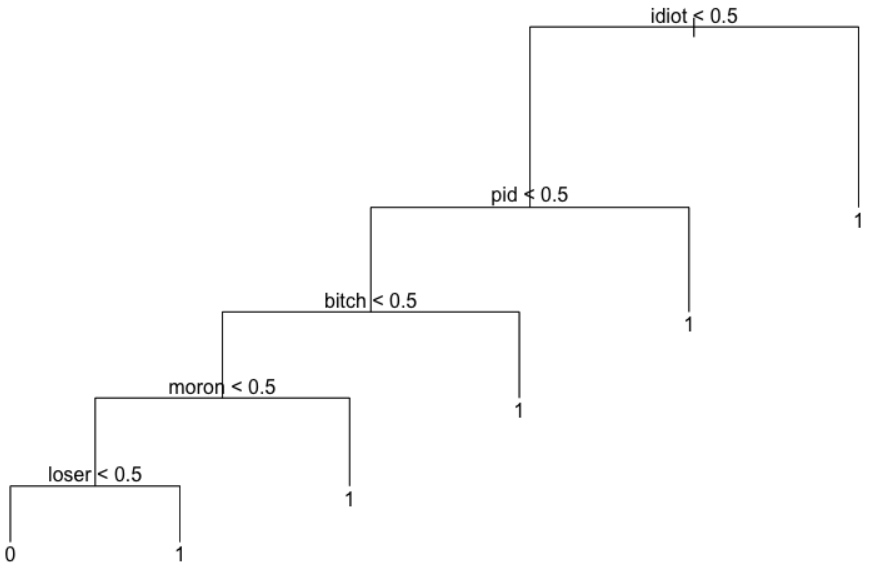

Figure 1: Classification Tree

```
tree.fit


set.seed(2)
testing$Insult <- as.factor(testing$Insult)
tree.pred <- predict(tree.fit, testing, type="class")
tree.table <- table(tree.pred, testing$Insult)


library(pROC)
roc.curve <- roc(as.numeric(tree.pred)-1, as.numeric(testing$Insult))
plot(roc.curve, main = "ROC: Classification Tree", col = "red")
auc.score<-auc(as.numeric(testing$Insult), as.numeric(tree.pred)-1)
auc.score
```

6.2 Random Forest

```
library(randomForest)
set.seed(100)
#500 ntree
# Start the clock!
ptm <- proc.time()
RF <- randomForest(train.dat[,1:ncol(train.dat)-1], factor(train.dat$Insult),
                   sampsize=500, do.trace=TRUE, importance=TRUE, ntree=20, forest=TRUE) #control
# 37.454 seconds for ntree=10
# Stop the clock
proc.time() - ptm
varImpPlot(RF)
rf.pred <- data.frame(Insult.pred=predict(RF,train.dat[,1:ncol(train.dat)-1],type="prob")[,2])

library(pROC)
set.seed(10)
roc.curve <- roc(rf.pred[,1], as.numeric(train.dat$Insult))
plot(roc.curve, main = "ROC: RF", col = "red")
auc.score<-auc(as.numeric(train.dat$Insult), rf.pred[,1])
auc.score
#Area under the curve: 0.8431

rf.pred[rf.pred>=0.5] <- 1
rf.pred[rf.pred<0.5] <- 0
rf.table <- table(pred=rf.pred[,1], train.dat$Insult)
sum(diag(rf.table))/sum(rf.table) #misclassification rate
```

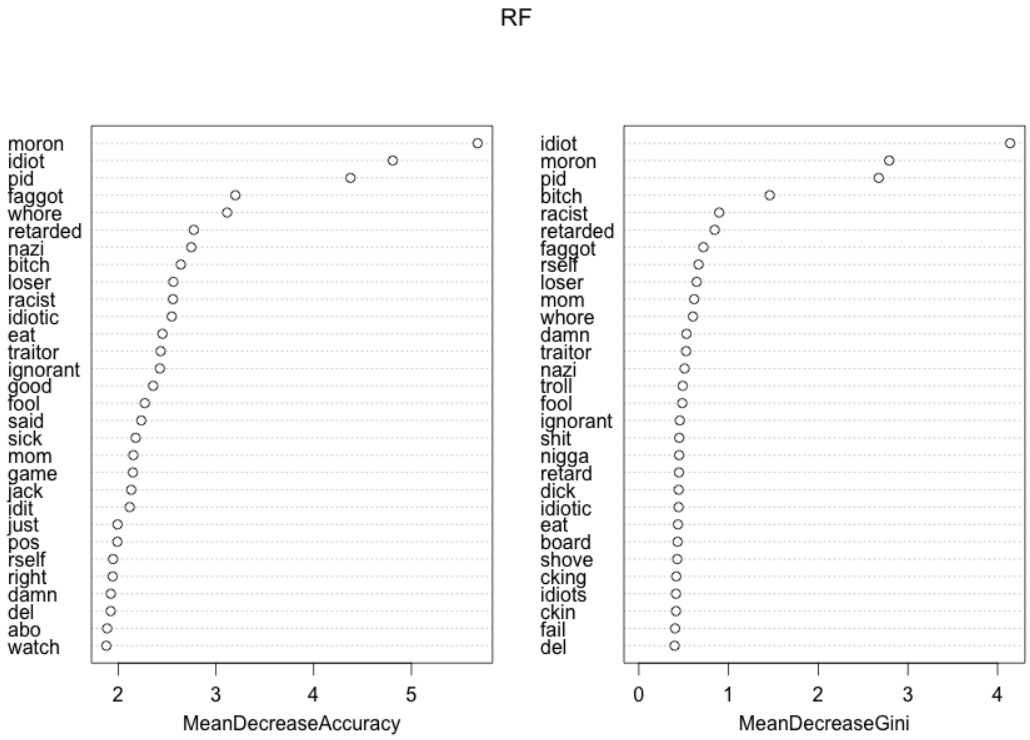

Figure 2: Random Forest

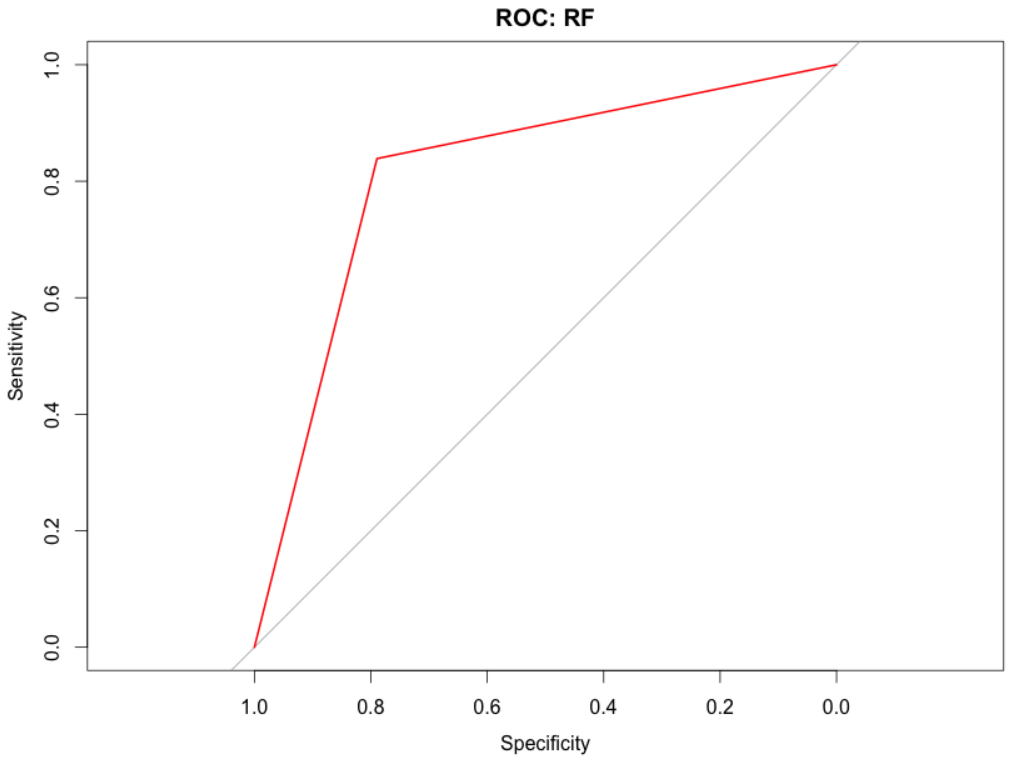

Figure 3: ROC Curve

7 Result

Method	AUC Score	Computation Time (s)
Random Forest	0.814	323.362
Decision Tree	0.789	46.725
Logistic Regression	0.785	—
Naive Bayes	0.798	—
KNN	0.730	—
SVM	0.786	—

8 Conclusion and Future Work

We have tried to investigate how to detect insults, as define in the introduction, that appear in social discussions. To deal with it, we used some knowledge acquired during the quarter in Machine Learning. Indeed, we started with cleaning the data, then finding relevant features and finally trying different classifiers and designing new predictor against over-fitting. Eventually, we used our Super Predictor by combining our predictors via AdaBoost. However, it is still difficult to detect some false negative results such as "this book is fXXXing good" or new words (wordplays) such as "yuck fou". For these cases, we will need to find more elaborated and complex techniques to tackle the issue.

9 References

[1] "An Introduction to Statistical Learning" by James, G., Witten, D., Hastie, T., Tibshirani, R.

[2] `https://www.kaggle.com/c/detecting-insults-in-social-commentary`.

[3] `http://www.chioka.in/kaggle-competition-solutions/`.

[4] `http://webmining.olariu.org/my-first-kaggle-competition-and-how-i-ranked/`.

[5] `http://en.wikipedia.org/wiki/Document-term_matrix`.