Final Prediction Challenge

Saara Ghani

2024-05-09

Project Instructions

While online job searches are a common way to connect jobseekers with employers, the FTC has warned that job scams are prevalent. In this assignment you will use any machine learning method, either from class or one you learned on your own, to predict which help wanted advertisements are scams.

The data are available in helpwanted.RData. This .RData file contains two data frames:

- dJobsTrain - This dataset includes 14,304 job postings, information about each posting, and a 0/1 indicator of whether or not this is a job scam, fraudulent.
- dJobsPred - This dataset includes 3,576 job postings with all the same features as the training dataset, but the fraudulent column is filled with NAs.

Fit your machine learning method to the data and try your best to predict whether or not the job postings in dJobsPred are scams or not.

Submit two items:

1. An R script documenting everything you did to load, clean, and recode the data and the machine learning method that you used.
2. An .RData file containing a data frame called dJobsP with exactly 3,576 rows, only the test data. That data frame must have two columns only, one called job_id and another called p. The column called p should contain your predicted probability that the job posting for the associated job_id is a scam. I will run the following script to evaluate the quality of your model (and you can run the first 6 stopifnot() to make sure your submission is in the correct format).

Note that if you predict every posting to have a probability 0.0497 (the baseline fraud rate in the training data) then the average Bernoulli log-likelihood is -0.198. So your submission should have an average Bernoulli log-likelihood that is larger than that.

Criteria:

- Effort. Did you try to develop an interesting set of posting features? Did you try to extract data from the postings themselves using NLP methods or SVD?
- Approach. Did you correctly use an appropriate machine learning method? Did you use an appropriate method to select tuning parameters?
- Performance. Does your model have good predictive performance? I will sort all submissions based on average Bernoulli log-likelihood. Scoring among the highest on average Bernoulli log-likelihood on the test set (that I am keeping secret) will, of course, make for a more compelling submission. I will give +5 points to the 3 submissions with the highest Bernoulli log-likelihood on the test set. Having a mediocre predictive performance on the test set will not result in any kind of penalty unless the low predictive performance can be attributable to the first two items: effort and approach.

My Solution

```
setwd("/Users/saaraghani/Desktop/Spring24/CRIM4012/HW")
load("data/helpwanted.RData")

# Load the required packages
library(tidyr)
library(dplyr)
library(tm)
```

Set Up

This section will use Natural Language Processing to break down parts of the dJobsTrain data into numerical values that can be used for evaluation.

The following code assesses the four long-text columns: 'company_profile', 'description', 'requirements', and 'benefits' and determine the 10 most common words for rows that have been marked as scams. It then creates a column for each word and checks the text to see if the word occurs, leaving a "1" if the word is present, and a "0" is the word is not present. These 0s and 1s will then be used, along with other numeric data in the data set, to create a model that predicts whether the job posting is a scam. Later, I will use the same words on the dJobsPred (prediction) dataset, to predict which postings are scams and which are real.

```
# Function to get the top 10 most used words
topTenWords <- function(dfCol) {
  corpus <- Corpus(VectorSource(dfCol))

  # Pre-processing
  corpus <- tm_map(corpus, content_transformer(tolower))
  corpus <- tm_map(corpus, removePunctuation)
  corpus <- tm_map(corpus, removeNumbers)
  corpus <- tm_map(corpus, removeWords, stopwords("en"))

  # Tokenization
  matrix <- as.matrix(DocumentTermMatrix(corpus))

  # Show top 10 most frequent words
  head(sort(colSums(matrix), decreasing = TRUE), 10)
}
```

The above function will be used to get the top 10 most used words in each of the 4 aforementioned columns. Then, 40 new columns will be created, 10 columns for each of the 4 features. Each column will have 0/1 values (0 = the word was found, 1 = the word was not found).

```
## Function to create the 4 lists of key words
keyWordGenerator <- function(colName) {
  dJobsTrain |>
    filter(fraudulent == 1) |>
    select(all_of(colName)) |>
    topTenWords() |>
    names()
}
```

```r
# Generate the lists
companyProfileKeyWords <- keyWordGenerator("company_profile")
descriptionKeyWords <- keyWordGenerator("description")
requirementsKeyWords <- keyWordGenerator("requirements")
benefitsKeyWords <- keyWordGenerator("benefits")

# View the lists!
companyProfileKeyWords
```

```
##  [1] "candidates" "services"   "recruiting" "bonus"      "business"
##  [6] "new"        "solutions"  "experience" "aptitude"   "staffing"
```

```r
descriptionKeyWords
```

```
##  [1] "work"       "will"       "amp"        "team"       "business"
##  [6] "position"   "management" "project"    "customer"   "company"
```

```r
requirementsKeyWords
```

```
##  [1] "experience"    "skills"        "work"          "ability"
##  [5] "years"         "knowledge"     "amp"           "must"
##  [9] "communication" "management"
```

```r
benefitsKeyWords
```

```
##  [1] "benefits"    "company"     "training"    "time"        "paid"
##  [6] "work"        "environment" "can"         "opportunity" "working"
```

```r
# Function to create the 10 new columns
keywordColCreator <- function(df, featureColName, keyWordsList) {

  for (i in 1:length(keyWordsList)) {
    df <- df |>
      mutate(!!paste(featureColName,
                     "_KEYWORD_",
                     keyWordsList[i],
                     sep = "")
             := as.numeric(grepl(keyWordsList[i],
                                 tolower(df[[featureColName]]))))
  }
  return(df)
}

# Run the function to create the columns
dJobsTrain <- keywordColCreator(dJobsTrain, "company_profile", companyProfileKeyWords)
dJobsTrain <- keywordColCreator(dJobsTrain, "description", descriptionKeyWords)
dJobsTrain <- keywordColCreator(dJobsTrain, "requirements", requirementsKeyWords)
dJobsTrain <- keywordColCreator(dJobsTrain, "benefits", benefitsKeyWords)
```

There are now 40 additional columns with 0/1 that can be used for predictions.

Update the Prediction Data

The following code updates the dJobsPred data frame with the 40 new columns that were just added to the dJobsTrain data.

```r
dJobsPred <- keywordColCreator(dJobsPred, "company_profile", companyProfileKeyWords)
dJobsPred <- keywordColCreator(dJobsPred, "description", descriptionKeyWords)
dJobsPred <- keywordColCreator(dJobsPred, "requirements", requirementsKeyWords)
dJobsPred <- keywordColCreator(dJobsPred, "benefits", benefitsKeyWords)
```

Prediction

After conducting analysis for both a K-Nearest Nieghbours model and a Decision Tree model, I chose to generate my predictions using the KNN model. This is because the Bernoulli Log Likelihood for the KNN model was higher (-0.1303) than that of the Decision Tree model (-0.1349). However, I have included the code and descriptions for both methods in this write-up.

K-Nearest Neighbours (Chosen Model)

```r
# Load the required packages
library(FNN)
library(dplyr)
library(tidyr)
library(doParallel)

# Prepare the data
X <- dJobsTrain[, c(10:12,19:58)]
y <- dJobsTrain$fraudulent

# Determining the optimal value for K
set.seed(9292)
i <- c(1:nrow(X))
kval <- round(seq(1,100,length=50))
cl <- makeCluster(12)
registerDoParallel(cl)

bernLL <- foreach(j=1:length(kval), .combine = c, .packages = "FNN") %dopar%
  {
    knnPred <- FNN::knn.cv(train = X[i, ],
                           cl = y[i],
                           k = kval[j],
                           prob = TRUE)

    # Extract the predicted probabilities
    p <- attr(knnPred, "prob")
    p <- ifelse(knnPred==1, p, 1-p)
    p <- (p*kval[j] + 1)/(kval[j] + 2)

    # Compute average Bernoulli log-likelihood
    avebernLL <- mean(ifelse(y[i]==1, log(p), log(1-p)))
    return(avebernLL)
  }

stopCluster(cl)
kBest <- kval[which.max(bernLL)]
max(bernLL) # -0.1303
```

```
## [1] -0.1302664
```

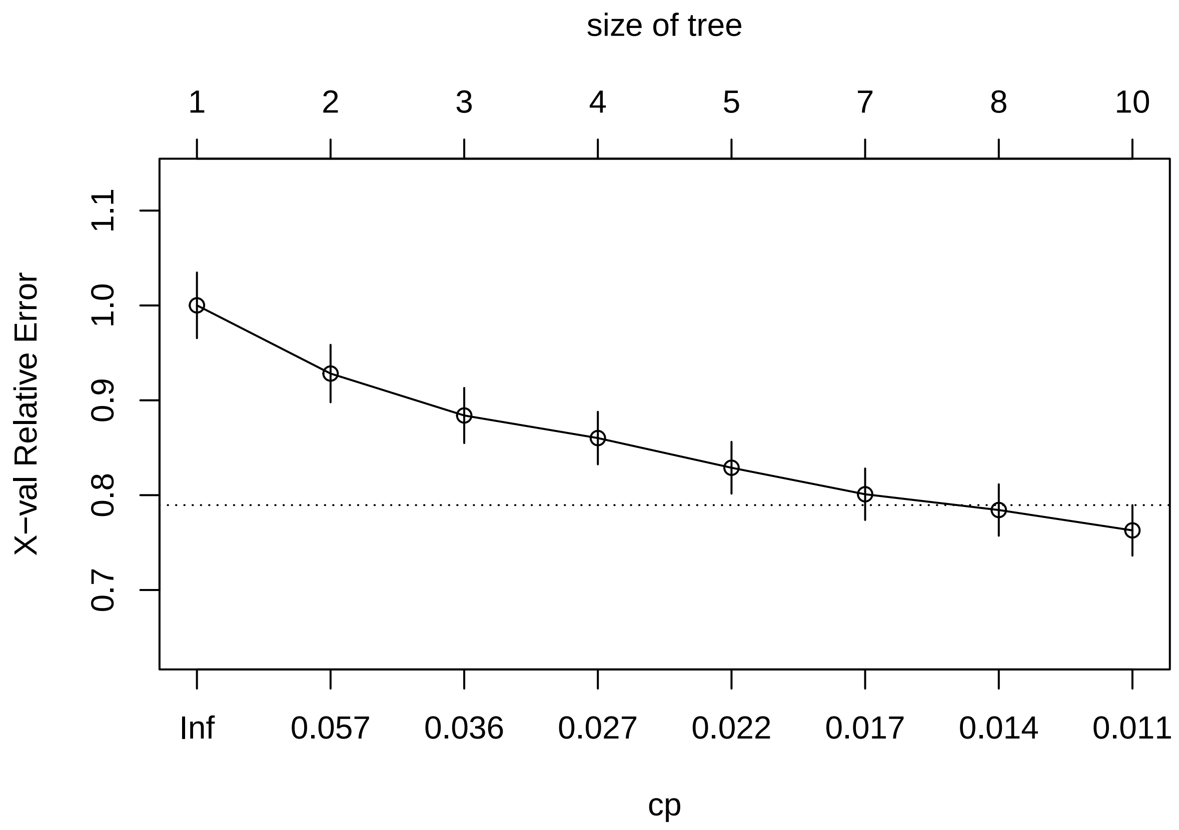
```r
# Use the model to generate predictions
knn1 <- knn(train   = X[i,],
            test    = dJobsPred[, c(10:12,19:58)],
            cl      = y[i],
            k       = kBest,
            prob    = TRUE)

# Calculate probability of fraud job posting
p <- attr(knn1, "prob")
p <- ifelse(knn1==1, p, 1-p)
p <- (p*kBest + 1)/(kBest + 2)

# Add predictions to data frame
dJobsPred$fraudulent <- p
```

Create Submission Material

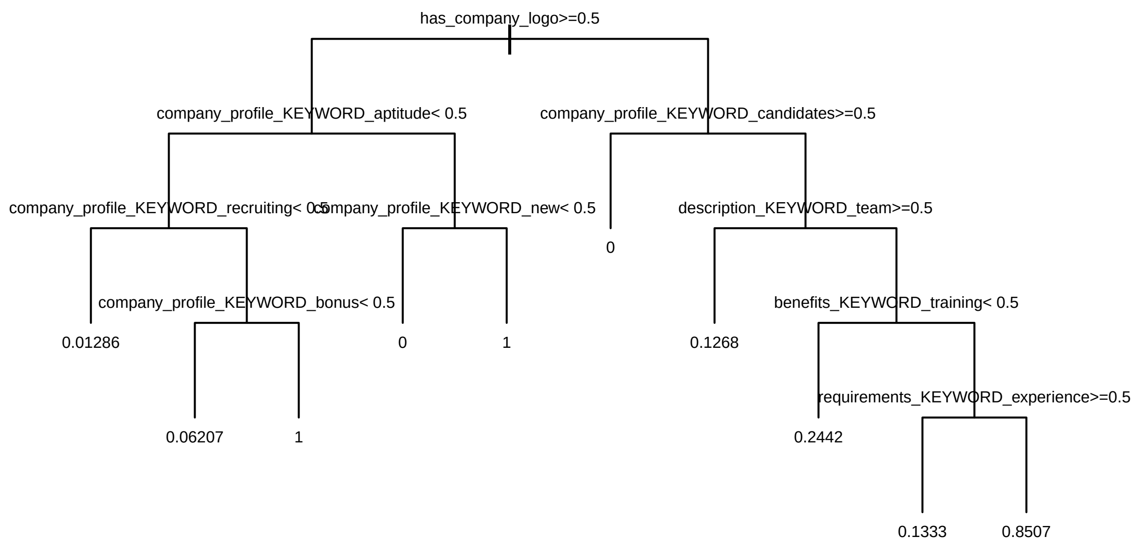
```r
dJobsP <- data.frame("job_id" = dJobsPred$job_id, p = dJobsPred$fraudulent)
save(dJobsP, file = "dJobsP.RData")
```

Decision Trees (Not-Chosen Model)

```r
# Load the required packages
library(rpart)

# Fit a regression tree
tree1 <- rpart(fraudulent ~ telecommuting+has_company_logo+has_questions+
                 company_profile_KEYWORD_candidates+company_profile_KEYWORD_services+
                 company_profile_KEYWORD_recruiting+company_profile_KEYWORD_bonus+
                 company_profile_KEYWORD_business+company_profile_KEYWORD_new+
                 company_profile_KEYWORD_solutions+company_profile_KEYWORD_experience+
                 company_profile_KEYWORD_aptitude+company_profile_KEYWORD_staffing+
                 description_KEYWORD_work+description_KEYWORD_will+
                 description_KEYWORD_amp+description_KEYWORD_team+
                 description_KEYWORD_business+description_KEYWORD_position+
                 description_KEYWORD_management+description_KEYWORD_project+
                 description_KEYWORD_customer+description_KEYWORD_company+
                 requirements_KEYWORD_experience+requirements_KEYWORD_skills+
                 requirements_KEYWORD_work+requirements_KEYWORD_ability+
                 requirements_KEYWORD_years+requirements_KEYWORD_knowledge+
                 requirements_KEYWORD_amp+requirements_KEYWORD_must+
                 requirements_KEYWORD_communication+requirements_KEYWORD_management+
                 benefits_KEYWORD_benefits+benefits_KEYWORD_company+
                 benefits_KEYWORD_training+benefits_KEYWORD_time+benefits_KEYWORD_paid+
                 benefits_KEYWORD_work+benefits_KEYWORD_environment+
                 benefits_KEYWORD_can+benefits_KEYWORD_opportunity+
                 benefits_KEYWORD_working,
               data = dJobsTrain, method = "anova")

# Determine the best sized tree
plotcp(tree1)
```



```
bestCP <- with(tree1, cptable[which.min(cptable[,"xerror"]),"CP"])

# Obtain the best tree based on the minimum cross-validated error
treeBest <- prune(tree1, cp = bestCP)

# Display the pruned tree
par(xpd=NA)
plot(treeBest, uniform=TRUE)
text(treeBest, cex=0.5)
```


has_company_logo>=0.5
company_profile_KEYWORD_aptitude< 0.5
company_profile_KEYWORD_candidates>=0.5
company_profile_KEYWORD_recruiting< 0.5
company_profile_KEYWORD_new< 0.5
description_KEYWORD_team>=0.5
0
company_profile_KEYWORD_bonus< 0.5
benefits_KEYWORD_training< 0.5
0.01286
0
1
0.1268
requirements_KEYWORD_experience>=0.5
0.06207
1
0.2442
0.1333
0.8507

```r
# Compute Bernoulli log-likelihood
predictions <- predict(treeBest, data = dJobsTrain, type = "vector")
mean(ifelse(y == 1, log(predictions), log(1 - predictions))) # -0.1349
```

```
## [1] -0.134876
```

```r
# Calculate probability of fraud job posting
predictions <- predict(treeBest, newdata = dJobsPred, type = "vector")
mean(predictions)
```

```
## [1] 0.04698889
```

Conclusion

The K-Nearest Neighbours model had the best predictive performance, therefore, my dJobsP submission was constructed using that method. However, I also found the Decision Tree method very useful, effective and interesting.