Oscars… WHAT?

12/06/2018

Contents

Stating the Question

TODO: Revise to Question-first mind-set

Objectives

The goal of this project is to develop models to predict whether a movie will win some kind of Oscar award or not, and potentially …..(TODO). In order to achieve this, we will use a dataset that BigMl put together and used to train a deep neural network to get perfect predictions for the 8 categories they targeted for 2018. However, this does not mean their model is perfect, as ….. (TODO: source for where they say even they were surprised?).

Motivation

The motivation is to use these models to identify the most relevant features that make a movie win an Academy Award, and also to have an edge when betting for the next Oscars winners. TODO: more motivations…?

The Data

Gathering the data was very straightforward in this case. After coming up with the idea for the project, a simple Google search of: *oscars machine learning* almost directly handed us the dataset. We quickly found this article, which lead us to BigML. After signing up on their website, we were able to download the dataset for free. However, this would not mean that the data would be ready to be fed to our machine learning algorithms right away… which we easily realized after doing some EDA. BigML combined data from IMDB with entries that specify whether a movie has won some other awards previous to the Oscars.

Exploratory Data Analysis (EDA)

First Contact

Let's load the data and see what features we have and how large is our dataset

```r
oscars <- read_csv("oscars.csv", col_types = cols())
#View(head(oscars)) # Go to the .Rdm file and uncomment this line
# if you want to have a look at the data
dim(oscars)
```

```
## [1] 1183  119
```

```r
#sapply(oscars, typeof) # Same here
```

So we have 1183 rows and 119 features. `year` will be useful to make a train-test split later, but we will drop it after that. `movie` and `movie_id` will also be dropped as they are not relevant for prediction. We will need to encode `certificate`, while `duration` is already in a suitable form, and `genre` will have to be converted to dummy variables. `rate` and `metascore` are also fine. We will also drop `synopsis`, which is very unlikely to matter significantly, and which would also require Natural Language Processing, as skill we lack? (TODO: maybe delete the NLP part). `votes` is also fine, while `gross` will need to be modified to account for inflation over the years. We will also drop `release_date` but will use `release_date.month` (at the end of the features) to create a new column `season`, which is known to be a relevant factor in winning an Oscar. `release_date.year`, `release_date.day-of-month` and `release_date.day-of-week` will also be dropped. `user_reviews`, `critic_reviews` and `popularity` are also numerical metrics that come from IMDB, and thus are already on their suitable form. By visual inspection, `awards_wins` and `awards_nominations` seem to count the number of awards and nominations other than the Oscars, but we will have to make sure of this.

Then we have 16 binary columns saying whether a movie won or was nominated for one of the following Oscars' categories

1. Best Picture

2. Best Director

3. Best Actor

4. Best Actress

5. Best Supporting Actor

6. Best Supporting Actress

7. Best Adapted Screenplay
8. Best Original Screenplay

As we are concerned with predicting whether a movie will win at least one Oscar or not, we will create such column (to be called `Oscars_won_some`) using these and drop them after doing so. Note that we are only using data from 8 categories… (TODO: argue this..?). With regards to the nominated columns… TODO: We didn't discuss this, did we? Maybe we can use the next feature (see below)

Then we have another column called `Oscars_nominated`, which says for how many categories a movie was nominated for, including the ones not mentioned before. `Oscars_nominated_categories` is its analogous but on text form, indicating the specific nominated categories. We will also drop this one, TODO: right?.

The rest of the features all follow the same pattern: first we have `award_name_won`, which says how many categories of such award the movie won, and then we have `award_name_won_categories`, a string specifying these categories. `award_name_nominated` and `award_name_nominated_categories` are the same but for nominations instead of wins. We will only use the numerical features, and drop the categorical ones (TODO: mention the other approach too).

Missing Values

Having looked and experimented with several ways of imputing the missings we have decided to handle imputing on only the set of variables we expect to apply to the model for simplicity. See section [Imputation] below.

TODO: Talk about this…

Quality Checks

Now let's make some quality checks, before diving into preprocessing. Our movies range from

```r
sort(unique(oscars$year))
```

```
##  [1] 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013
## [15] 2014 2015 2016 2017
```

and for each year, we have

```r
table(oscars$year)
```

```
##
## 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014
##   71   67   67   63   71   69   71   68   64   67   69   71   67   73   68
## 2015 2016 2017
##   66   61   30
```

about 70 movies, but for 2017 we only have 30. We are thus not going to use the data from 2017, instead focusing on 2000-2016 data, which appears more complete.

```r
# Lied about not doing yet any preprocessing...
oscars = filter(oscars, year != 2017)
```

One other thing we must make sure of is that we have only 16 winners for each category. TODO: Do this…

```r
# Pattern to get only the relevant columns
pat.oscar_won <- "Oscar_.*_won$"
pat.oscar_nom <- "Oscar_.*_nominated$"

t(oscars %>%
  select(matches(pat.oscar_won)) %>%
  mutate_at(vars(matches(pat.oscar_won)),
            funs(ifelse(. == "No", 0, 1))) %>%
  summarise_all(sum))
```

```
##                                     [,1]
## Oscar_Best_Picture_won                17
## Oscar_Best_Director_won               17
## Oscar_Best_Actor_won                  17
## Oscar_Best_Actress_won                17
## Oscar_Best_Supporting_Actor_won       17
## Oscar_Best_Supporting_Actress_won     17
## Oscar_Best_AdaScreen_won              17
```

```
## Oscar_Best_OriScreen_won            17
```

OK, so for now the data makes sense. However, it is very clear that our data is heavily imbalanced. TODO: Explain how we will deal with this later… examples: balancing the weights, oversampling? or maybe we don't need to.

Let's also make sure that `awards_wins` does not contain any information regarding the Oscars, as this would not be known for us when making future predictions.

```
# Gets the movies that won Oscars for best picture
oscars_won_best_picture <- subset(oscars, Oscar_Best_Picture_won == "Yes")
dim(oscars_won_best_picture)
```

```
## [1]  17 119
```

```
# Gets the first movie of such list
lord_of_the_rings_king <- head(oscars_won_best_picture, 1)
# Gets the number of awards it won, not counting the Oscars
num_awards_won <- 0
# For every column name in the Series
for (i in names(lord_of_the_rings_king)){
  # If "won"" is in the column name at the end
  if(grepl("won$", i) == TRUE){
    # And if "Oscar" is not in the column name
     if(grepl("Oscar", i) == FALSE){
         # Add to num_awards_won the value of such column
         num_awards_won = num_awards_won + lord_of_the_rings_king[c(i)][[1]]
      }
  }
}
# Prints the value of awards_wins
lord_of_the_rings_king[c("awards_wins")]
```

```
## # A tibble: 1 x 1
##   awards_wins
##         <int>
## 1          48
```

```
# Prints the number of awards it won, not counting the Oscars
num_awards_won
```

```
## [1] 48
```

(TODO: maybe do this for more movies, just to be sure). Good! This feature is fair and probably very important.

Data Preprocessing

Let's proceed on an orderly manner. First, let's get the target column, as it will be useful for later preprocessing.

```
# sum number of wins identified per row for cols
tmp.won <- rowSums(
  oscars %>%
    mutate_at(
      vars(matches(pat.oscar_won)),
      funs(ifelse(. == "No", 0, 1))
    ) %>% select(matches(pat.oscar_won))
```

```r
  )

# set win count variable in oscars df
oscars <- oscars %>%
  mutate(Oscars_win_count = tmp.won)

# Validation
oscars_won_best_picture <- filter(oscars, Oscar_Best_Picture_won == 'Yes')
# This movie won 3 oscars, let's see if it checks out
oscars_won_best_picture$Oscars_win_count[1] == 3
```

```
## [1] TRUE
```

```r
# This movie won 2 oscars, let's see if it checks out
oscars_won_best_picture$Oscars_win_count[2] == 2
```

```
## [1] TRUE
```

```r
# Yeap

# Encodes as factor
oscars <- oscars %>% mutate(
  Oscars_won_some = factor(ifelse(Oscars_win_count > 0, "Yes", "No"))
)
```

Now, let's encode `certificate`, which follows this system and this one. That is, `certificate` is an ordinal variable and needs to be encoded as such (preserving the order).

```r
# Prints the unique values of the certificate column
unique(oscars$certificate)
```

```
## [1] "PG-13"     "G"         "R"         "PG"        "Not Rated" "Unrated"
## [7] NA          "TV-MA"
```

```r
# Prints the movies "Unrated" and "Not Rated"
subset(oscars, certificate == "Not Rated")[c("movie")]
```

```
## # A tibble: 14 x 1
##    movie
##    <chr>
##  1 Ken Park
##  2 Evil
##  3 The Secret of Kells
##  4 Il Divo
##  5 Death Proof
##  6 Revanche
##  7 In the Loop
##  8 Chico & Rita
##  9 War Witch
## 10 The Broken Circle Breakdown
## 11 The Great Beauty
## 12 The Invitation
## 13 Tangerines
## 14 Embrace of the Serpent
```

```r
subset(oscars, certificate == "Unrated")[c("movie")]
```

```
## # A tibble: 4 x 1
```

```
##   movie
##   <chr>
## 1 Zus & zo
## 2 Beaufort
## 3 Outside the Law
## 4 Dogtooth
```

Let's get rid of these rows. They would only make us create more columns for `certificate`, as we cannot encode them, and they are not well known movies (aside from The Great Beauty…) and did not win any Oscars.

```r
# Gets the indexes of such rows
not_rated_index = as.numeric(rownames(subset(oscars, certificate == "Not Rated")[c("movie")]))
unrated_index =  as.numeric(rownames(subset(oscars, certificate == "Unrated")[c("movie")]))

# Checks if any of them won an oscar
for(i in 1:nrow(oscars[not_rated_index, ])){
  if(oscars[not_rated_index, ]$Oscars_won_some[i] == "Yes"){
    print("Yes")
  }
}
for(i in 1:nrow(oscars[unrated_index, ])){
  if(oscars[unrated_index, ]$Oscars_won_some[i] == "Yes"){
    print("Yes")
  }
}
```

Assumption checked. Let us drop them.

```r
dim(oscars)
```

```
## [1] 1153  121
```

```r
# https://github.com/tidyverse/dplyr/issues/3196
oscars = oscars %>% filter(certificate != "Not Rated" | is.na(certificate)) %>% filter(certificate != "U
dim(oscars)
```

```
## [1] 1135  121
```

Now let us treat the missing values…

```r
unique(oscars$certificate)
```

```
## [1] "PG-13" "G"     "R"     "PG"    NA      "TV-MA"
```

```r
sum(is.na(oscars$certificate))
```

```
## [1] 10
```

well, let's actually see if we can get the missing values imputed first.

```r
titles_missing_values = oscars[is.na(oscars$certificate), ]$movie
titles_missing_values
```

```
##  [1] "The Twilight Samurai"              "As It Is in Heaven"
##  [3] "Sophie Scholl: The Final Days"     "Katyn"
##  [5] "Ajami"                             "The Milk of Sorrow"
##  [7] "Guy and Madeline on a Park Bench" "Omar"
##  [9] "Theeb"                             "Tanna"
```

If we look up these titles in IMDB, they are not rated. Let's check once again if they won any Oscars:

```r
for(i in titles_missing_values){
  if(subset(oscars, movie == i)$Oscars_won_some == "Yes"){
    print("Yes")}
}
```

The did not, so let's drop them too.

```r
oscars = oscars %>% filter(is.na(certificate) == FALSE)
dim(oscars)
```

```
## [1] 1125  121
```

And finally let's encode it…

```r
#TODO: I'm actually pretty sure this should just be a factor variable
unique(oscars$certificate)
```

```
## [1] "PG-13" "G"     "R"     "PG"    "TV-MA"
```

```r
oscars = oscars %>% mutate(certificate = replace(certificate, certificate == "PG-13", 3))
oscars = oscars %>% mutate(certificate = replace(certificate, certificate == "G", 1))
oscars = oscars %>% mutate(certificate = replace(certificate, certificate == "R", 4))
oscars = oscars %>% mutate(certificate = replace(certificate, certificate == "PG", 2))
oscars = oscars %>% mutate(certificate = replace(certificate, certificate == "TV-MA", 5))
oscars$certificate = as.numeric(oscars$certificate)
unique(oscars$certificate)
```

```
## [1] 3 1 4 2 5
```

Moving on to `genre`, now we do need to create dummy variables

```r
# get all of the values parsed by |'s
genre_unique = unique(oscars$genre)
# get the actual unique values
soup = c()
for(i in 1:length(genre_unique)){
  soup = append(soup,unlist(strsplit(genre_unique[i],"\\|")))
}
new_cols = unique(soup)

# create a dataframe where the column names are the unique genres
gen = data.frame(matrix(nrow=nrow(oscars),ncol=length(new_cols)))
colnames(gen) = new_cols

for(i in 1:ncol(gen)){
  # iterate over columns
  for(j in 1:nrow(gen)){
    # then rows
    # if the string with the column name is in the string for the awards_won column in the original
    # dataset.... give that variable a 1 in the new dataset
    if((grepl(colnames(gen[i]),oscars$genre[j])==TRUE)){
      gen[j,i] = 1
      }
    else{
      gen[j,i] = 0
    }
  }
}
```

```r
# we add the prefix "genre", so that the varaibles are easier to identify.
colnames(gen) = paste("genre",colnames(gen),sep="_")

# get rid of the mispelling of history... somehow some moves were classified as histor and history
# assume music and musical are the same genre.
gen$genre_History = gen$genre_History + gen$genre_Histor
gen$genre_Musical = gen$genre_Musical + gen$genre_Music


# check for duplicates
max(gen$genre_History)
```

```
## [1] 2
```

```r
max(gen$genre_Musical)
```

```
## [1] 2
```

```r
# we have them in both, so remove them.

for(i in 1:length(gen$genre_History)){
  if(gen$genre_History[i]>1){
    gen$genre_History[i] =1
  }
}

for(i in 1:length(gen$genre_Musical)){
  if(gen$genre_Musical[i]>1){
    gen$genre_Musical[i] =1
  }
}

# select the columns that are duplicated
DropCols = c("genre_Histor","genre_Music")
# remove them from the datframe.

gen = gen[,!colnames(gen)%in%DropCols]

oscars = cbind(oscars,gen)

# Changing 0 to "No" and 1 to "Yes" and converting to factor
for(i in names(gen)){
  oscars[c(i)][[1]] = ifelse(oscars[c(i)][[1]] == 1, "Yes", "No")
  oscars[c(i)][[1]] = factor(oscars[c(i)][[1]])
}
```

When we tried to run our model, we got this error ("Prints out this error: Error in `contrasts<-(*tmp*`, value = contr.funs[1 + isOF[nn]]) : contrasts can be applied only to factors with 2 or more levels"), and it was because there is only one movie that falls into the "documentary" genre, and thus only one level for `genre_documentary`

```r
table(oscars$genre_Documentary)
```

```
##
##   No  Yes
## 1124    1
```

The movie in question is

```r
subset(oscars, genre_Documentary == "Yes")$movie
```

```
## [1] "Jim: The James Foley Story"
```

and here are the it received or was nominated for. As it is not that relevant and very annoying (TODO: Find a better way if there is one), let's drop it

```r
oscars <- oscars %>% select(-genre_Documentary)
```

The next feature that needs some manipulation is `gross`. The value of the dollar changed from 2000 to 2016, so we'll need to adjust the values we have to be based on a standard. We'll use the value of the dollar in the year 2000 as a common unit. To convert between 2001 through 2016 dollar amounts, and the 2000 dollar amounts we'll need something to use as a conversion factor. We'll use the Consumer Price Index (CPI) data provided by the United States Bureau of Labor Statistics, to get that conversion factor. Their site (https://data.bls.gov/timeseries/CUUR0000SA0), provides CPI data by month output to an excel sheet. We've opted to include the Annual average, and base our conversion on that. After cleaning the data up in excel, we saved it off in CPI_20181201.csv, which we'll load now.

```r
# Pull in Consumer Price Index Data from
# Burea of Labor Statistics to account for inflation
cpi <- read_csv('CPI_20181201.csv', col_types = cols())
summary(cpi)
```

```
##       Year          Annual
##  Min.   :2000   Min.   :172.2
##  1st Qu.:2004   1st Qu.:190.5
##  Median :2008   Median :214.9
##  Mean   :2008   Mean   :211.1
##  3rd Qu.:2013   3rd Qu.:232.1
##  Max.   :2017   Max.   :245.1
```

Now we just need a function for retrieving the correct CPI for a given year.

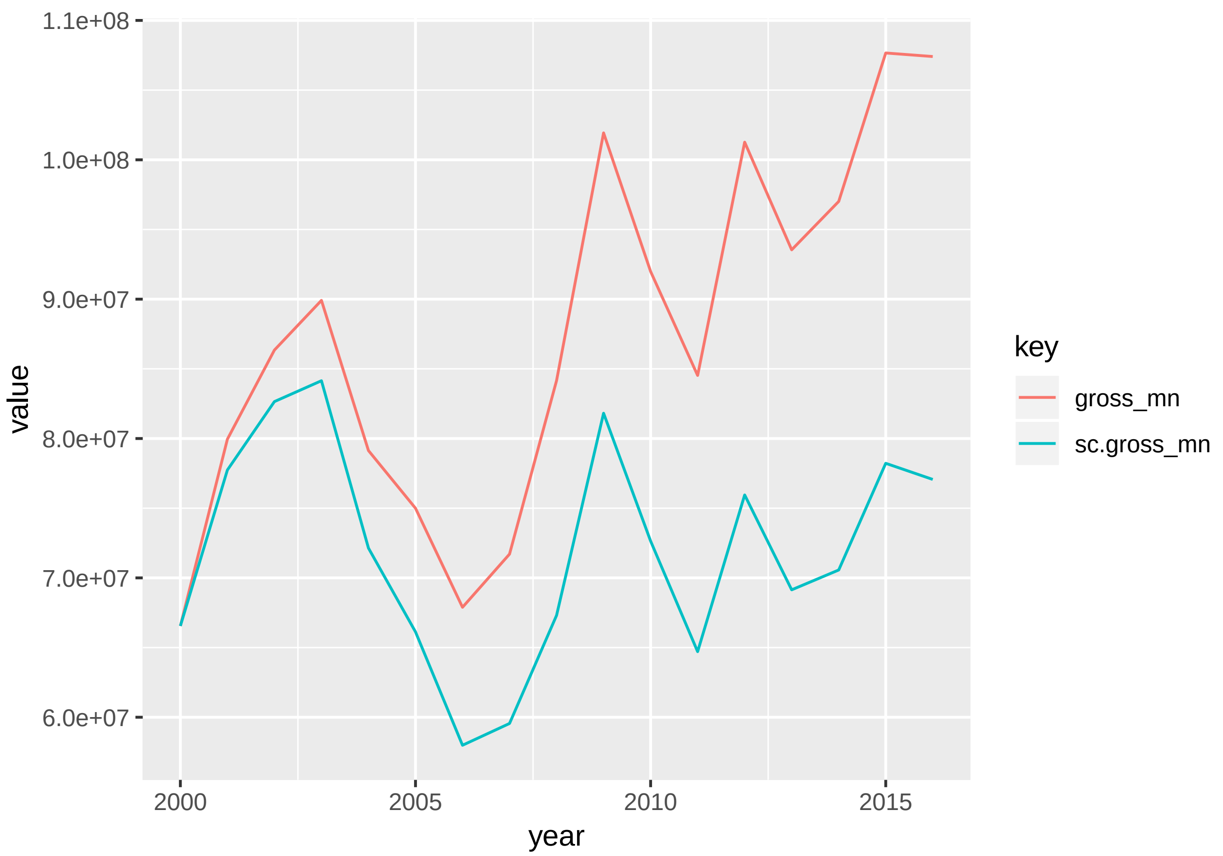
```r
# data ranged from 2000 (1) to 2017 (18)
# so... year mod 2000 + 1 is the indexing scheme.
# NOTE: if cpi csv file is changed the indexing scheme will need updating
cpif <- function(year) {
  idx <- year %% 2000 + 1
  cpi$Annual[idx]
}
```

Now we simply apply the conversion factor derived from the CPI data.

```r
oscars = oscars %>% mutate(
  # Adjust gross field by Consumer Price Index.
  # cpif provides annual average CPI for specified year
  # data provided by the Burea of Labor Statistics website
  # (implementation included in full Rmd document)
  sc.gross = gross * cpif(2000)/cpif(year) # "In 2000 dollars"
) %>%
  # TODO: Should this be here? thinking to move it down
  # mutate oscars won and oscars nominated cols/variables to factor types
  mutate_at(vars(matches(pat.oscar_won)), funs(factor)) %>%
  mutate_at(vars(matches(pat.oscar_nom)), funs(factor))
```

```r
# Validation of currency adjustment
currency_info <- oscars %>% select(
```

```
  gross, sc.gross, year
) %>% group_by(year) %>% summarize(
  gross_mn = mean(gross, na.rm = TRUE),
  sc.gross_mn = mean(sc.gross, na.rm = TRUE)
)

currency_info %>%
  gather(key, value, gross_mn, sc.gross_mn) %>%
  ggplot(aes(x=year, y=value, colour=key)) +
  geom_line()
```



We also want to create a **season** variable, based on the release date information, and using the meteorlogically defined seasons in the northern hemisphere as described here: https://www.timeanddate.com/calendar/aboutseasons.html

```
season <- function(month) {
  retVal <- "Fall"

  if (month <= 2) {
    "Winter" # Winter [December, February]
  } else if (month <= 5) {
    retVal <- "Spring"  # Spring [March, May]
  } else if (month <= 8) {
    retVal <- "Summer" # Summer [June, August]
  } else if (month <= 11) {
    retVal <- "Fall" # Fall   [September, November]
  } else {
```

```
    retVal <- "Winter"
  }

  return (retVal)
}

oscars <- oscars %>%
  rowwise() %>%
  mutate(seasons = season(release_date.month))
oscars$seasons = factor(oscars$seasons)
```

All that is left is dropping the undesired columns, and ensuring the correct data types are being used.

```
drop.cols <- c(
  'movie',
  'movie_id',
  'synopsis',
  'gross', # dropping gross b/c gross.sc contains scaled values
  'Oscars_win_count',
  'release_date',
  'Oscar_nominated_categories',
  'genre',
  'release_date.year',
  'release_date.day-of-month',
  'release_date.day-of-week',
  'release_date.month')
oscars <- oscars %>%
  select(-one_of(drop.cols)) %>%
  select(-matches("categories$")) %>%
  select(-matches("Oscar_.*_won$"))
```

Imputation

Let's figure out to handle any missing values, now that we're removed some of the columns based on the EDA from above.

```
missing_per_column = sapply(oscars, function(x) sum(is.na(x)))
missing_per_column[missing_per_column != 0]
```

```
##      metascore   user_reviews critic_reviews     popularity       sc.gross
##             10              7              7            101             16
```

Using the `tidyimpute` package to impute the missing values identified above based on the mean. After some consideration, this seemed the best approach to avoid having the imputation inject error into our model.

```
oscars <- oscars %>% impute_mean(
  metascore,
  user_reviews,
  critic_reviews,
  popularity,
  sc.gross
)
```

Train and Test Split

and finally let's make sure that we only have the features we want for prediction, and that their types are the right ones (we did not drop `year` yet as we will use it for splitting intro training and testing)

```r
sapply(oscars, typeof)
```

```
##                                 year
##                            "integer"
##                          certificate
##                             "double"
##                             duration
##                            "integer"
##                                 rate
##                             "double"
##                            metascore
##                            "integer"
##                                votes
##                            "integer"
##                         user_reviews
##                            "integer"
##                       critic_reviews
##                            "integer"
##                           popularity
##                            "integer"
##                          awards_wins
##                            "integer"
##                   awards_nominations
##                            "integer"
##          Oscar_Best_Picture_nominated
##                            "integer"
##         Oscar_Best_Director_nominated
##                            "integer"
##            Oscar_Best_Actor_nominated
##                            "integer"
##          Oscar_Best_Actress_nominated
##                            "integer"
##  Oscar_Best_Supporting_Actor_nominated
##                            "integer"
## Oscar_Best_Supporting_Actress_nominated
##                            "integer"
##        Oscar_Best_AdaScreen_nominated
##                            "integer"
##        Oscar_Best_OriScreen_nominated
##                            "integer"
##                      Oscar_nominated
##                            "integer"
##                    Golden_Globes_won
##                            "integer"
##              Golden_Globes_nominated
##                            "integer"
##                            BAFTA_won
##                            "integer"
##                      BAFTA_nominated
##                            "integer"
```

```
##                              Screen_Actors_Guild_won
##                                            "integer"
##                        Screen_Actors_Guild_nominated
##                                            "integer"
##                                   Critics_Choice_won
##                                            "integer"
##                             Critics_Choice_nominated
##                                            "integer"
##                                  Directors_Guild_won
##                                            "integer"
##                            Directors_Guild_nominated
##                                            "integer"
##                                  Producers_Guild_won
##                                            "integer"
##                            Producers_Guild_nominated
##                                            "integer"
##                              Art_Directors_Guild_won
##                                            "integer"
##                        Art_Directors_Guild_nominated
##                                            "integer"
##                                    Writers_Guild_won
##                                            "integer"
##                              Writers_Guild_nominated
##                                            "integer"
##                          Costume_Designers_Guild_won
##                                            "integer"
##                    Costume_Designers_Guild_nominated
##                                            "integer"
##               Online_Film_Television_Association_won
##                                            "integer"
##         Online_Film_Television_Association_nominated
##                                            "integer"
##                      Online_Film_Critics_Society_won
##                                            "integer"
##                Online_Film_Critics_Society_nominated
##                                            "integer"
##                                    People_Choice_won
##                                            "integer"
##                              People_Choice_nominated
##                                            "integer"
##                        London_Critics_Circle_Film_won
##                                            "integer"
##                  London_Critics_Circle_Film_nominated
##                                            "integer"
##                           American_Cinema_Editors_won
##                                            "integer"
##                     American_Cinema_Editors_nominated
##                                            "integer"
##                                   Hollywood_Film_won
##                                            "integer"
##                             Hollywood_Film_nominated
##                                            "integer"
##                   Austin_Film_Critics_Association_won
##                                            "integer"
```

```
##      Austin_Film_Critics_Association_nominated
##                                      "integer"
##                Denver_Film_Critics_Society_won
##                                      "integer"
##          Denver_Film_Critics_Society_nominated
##                                      "integer"
##             Boston_Society_of_Film_Critics_won
##                                      "integer"
##       Boston_Society_of_Film_Critics_nominated
##                                      "integer"
##               New_York_Film_Critics_Circle_won
##                                      "integer"
##         New_York_Film_Critics_Circle_nominated
##                                      "integer"
##       Los_Angeles_Film_Critics_Association_won
##                                      "integer"
## Los_Angeles_Film_Critics_Association_nominated
##                                      "integer"
##                                Oscars_won_some
##                                      "integer"
##                                   genre_Comedy
##                                      "integer"
##                                  genre_Fantasy
##                                      "integer"
##                                  genre_Romance
##                                      "integer"
##                                genre_Animation
##                                      "integer"
##                                genre_Adventure
##                                      "integer"
##                                   genre_Action
##                                      "integer"
##                                   genre_Family
##                                      "integer"
##                                genre_Biography
##                                      "integer"
##                                    genre_Drama
##                                      "integer"
##                                 genre_Thriller
##                                      "integer"
##                                   genre_Horror
##                                      "integer"
##                                   genre_Sci-Fi
##                                      "integer"
##                                    genre_Crime
##                                      "integer"
##                                      genre_War
##                                      "integer"
##                                  genre_History
##                                      "integer"
##                                  genre_Mystery
##                                      "integer"
##                                  genre_Musical
##                                      "integer"
```

```
##                                         genre_Sport
##                                           "integer"
##                                       genre_Western
##                                           "integer"
##                                            sc.gross
##                                            "double"
##                                             seasons
##                                           "integer"
```

TODO: again.. maybe we should drop the `Oscar_..._nominated` if we are only going to predict `Oscars_won_some` and use only `Oscar_nominated` instead.

Everything seems fine. Let's move onto splitting the data. Let's use from 2000 to 2012 (~75%) for training and from 2013 to 2016 for testing. There is no need to stratify as we are not doing a random split, that is, both train and test sets will have the same proportion of class labels, because we are splitting by years.

```r
# Splits by years
oscars_train <- subset(oscars, year %in% c(2000:2012))
oscars_test <- subset(oscars, year %in% c(2013:2016))

# Drops year
oscars <- select(oscars, -year)
oscars_train <- select(oscars_train, -year)
oscars_test <- select(oscars_test, -year)

# Gets the target
y_train = oscars_train$Oscars_won_some
y_test = oscars_test$Oscars_won_some
# Drops the target column
# TODO: Not sure it's necessary to drop target col?
X_train <- select(oscars_train, -Oscars_won_some)
X_test <- select(oscars_test, -Oscars_won_some)
```

Standardization

The last step before we can train our models is standardizing. By default, R omits standardizing factor variables (TODO: Is this true?), so we do not need to be concerned about that. We will follow this method

```r
scaleParam <- preProcess(oscars_train, method=c("center", "scale"))
oscars_train <- predict(scaleParam, oscars_train)
oscars_test <- predict(scaleParam, oscars_test)

scaleParam <- preProcess(X_train, method=c("center", "scale"))
X_train <- predict(scaleParam, X_train)
X_test <- predict(scaleParam, X_test)
```

Well... it seems we are ready to train some Machine Learning models!

Model Building

We decided to tackle the prediction problem using a combination of Logistic Regression and Random Forest Classification because of the categorical nature of the dependent variable.

Logistic Regression

First we'll build a Multiple Logistic Regression model, then we'll attempt to improve upon it using step-wise forward variable selection.

Regular Logistic Regression

Let's just apply a Logistic Regression model based on all the available variables.

```
model.glm.all <- glm(Oscars_won_some~., family = binomial(link = "logit"), data = oscars_train)
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

In viewing the summary of this model we can see that something has clearly gone wrong, in fact R gives us a warning stating that the algorithm did not converge, and that fitted probabilities numerically 0 or 1 occurred. The standard errors are all in the thousands, the z value statistics are all close to zero, and the p-values are close to 1. accross the majority of the the variable types. There are also two independent variables whose coefficients that couldn't be estimated.

After a good deal of discussion, and experimentation we determined that the issue with this model was ultimately us having too few degrees of freedom for the model to successfully train with this many independent variables (hence why some estimates were NAs and p-values where all close to 1)

Having done that, lets experiment with just using a subset of the variables available:

```
# Trains the model
model.glm.subset <-  glm(Oscars_won_some~certificate + duration+rate+metascore+votes+user_reviews+critic
                 family = binomial(link = "logit"), data = oscars_train)
```

The subsetted model shows that the model estimated all the coefficients, and has more reasonable z-value and p-value statistics. The most significant features by far, appear to be `certificate`, `user_reviews`, and `award_wins`.

Let's how well our model predicts the oscar winners in our training set.

```
# Tests the model
model.glm.subset.test <- predict.glm(model.glm.subset,oscars_test,type='response')

cm <- confusionMatrix(factor(ifelse(model.glm.subset.test > 0.5,"Yes","No")), y_test)

cm$table
```

```
##             Reference
## Prediction   No Yes
##        No   234   9
##        Yes    6  12
```

```
cm$overall[1]
```

```
##  Accuracy
## 0.9425287
```

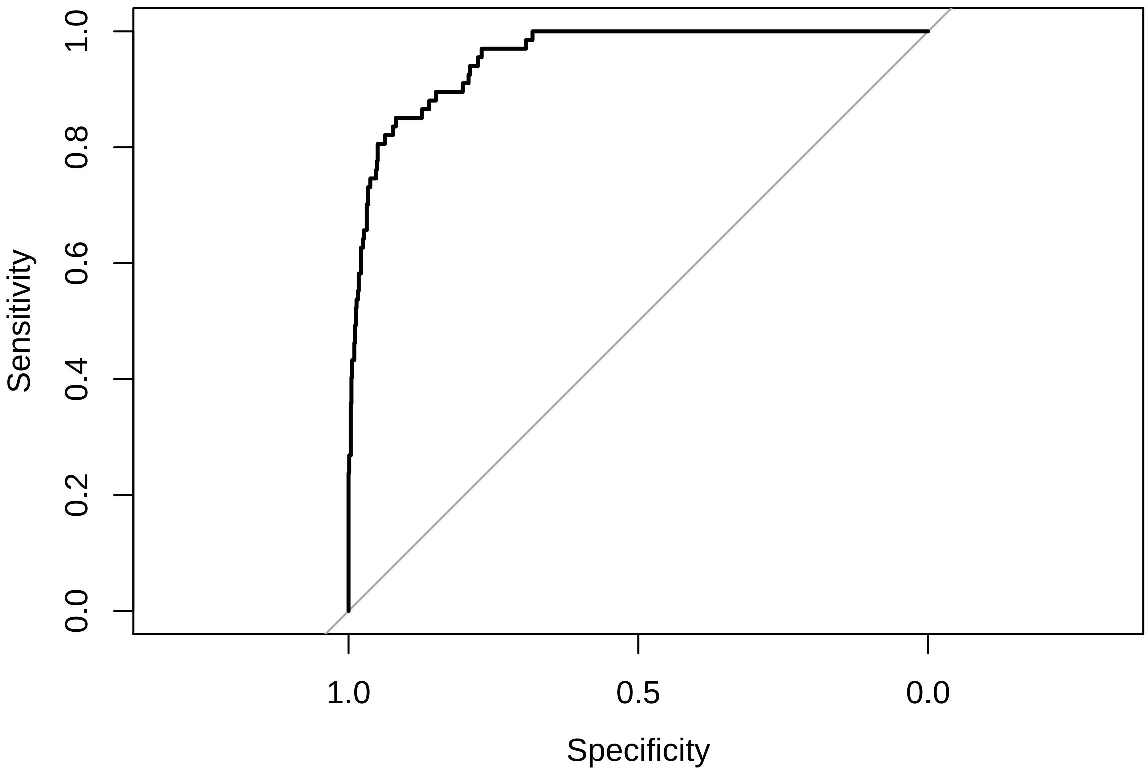
```
cm$overall[5]
```

```
## AccuracyNull
##    0.9195402
```

The models hit rate is ~94%. But we need to consider that most of the movies in our dataset will not win an Oscar (~92%, as shown by AccuracyNull above). Our hit rate is higher than the percentage of movies that did not win Oscars, implying that our model's predictive capability, using a 50% threshold, is an improvement over simply using our target's distribution to predict the winners.

Although we had a high True Negative rate (~97%), our True positive rate was lower (~62%). This doesn't show in our hit rate, because only ~8% of our movies actually won oscars. Let us take a look at the ROC Curves, to see if a different threshold quantity would have a significant impact on the predictive capability of the model.

```
model.glm.subset.prob=plogis(predict.glm(model.glm.subset, type = c("response")))
#head(prob)
model.glm.subset.h <- roc(Oscars_won_some~model.glm.subset.prob, data=oscars_train)
plot(model.glm.subset.h)
```




```
# Area Under the Curve
auc(model.glm.subset.h)
```

```
## Area under the curve: 0.9548
```

The area under the curve is ~95%, which is greater than the no data rate of ~92% that we found before implying that our model discriminates well at different thresholds.

Stepwise Logistic Regression

Now we'll attempt to build a Logistic Regrssion model doing feature selection using stepwise regression with BIC as an evaluation metric.

```
# creating a null model
model.glm.null <-  glm(Oscars_won_some~1,
                 family = binomial(link = "logit"), data = oscars_train)
```

```r
# create a full model
model.glm.all <- glm(Oscars_won_some~.,
                family = binomial(link = "logit"), data = oscars_train)
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
summary(model.glm.final)
```

```
##
## Call:
## glm(formula = Oscars_won_some ~ awards_wins + Screen_Actors_Guild_won +
##     Golden_Globes_won + genre_Drama + Writers_Guild_won + Online_Film_Television_Association_nominate
##     BAFTA_won, family = binomial(link = "logit"), data = oscars_train)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.14165  -0.14737  -0.03099  -0.01815   3.12610
##
## Coefficients:
##                                             Estimate Std. Error z value
## (Intercept)                                  -8.0052     1.4739  -5.431
## awards_wins                                   0.9385     0.4731   1.984
## Screen_Actors_Guild_won                       1.1482     0.2206   5.204
## Golden_Globes_won                             1.2203     0.2364   5.162
## genre_DramaYes                                4.1935     1.3866   3.024
## Writers_Guild_won                             0.8490     0.1738   4.884
## Online_Film_Television_Association_nominated -1.4928     0.3992  -3.739
## BAFTA_won                                     0.8418     0.2623   3.210
##                                             Pr(>|z|)
## (Intercept)                                 5.59e-08 ***
## awards_wins                                 0.047278 *
## Screen_Actors_Guild_won                     1.95e-07 ***
## Golden_Globes_won                           2.45e-07 ***
## genre_DramaYes                              0.002492 **
## Writers_Guild_won                           1.04e-06 ***
## Online_Film_Television_Association_nominated 0.000184 ***
## BAFTA_won                                   0.001329 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 471.29  on 863  degrees of freedom
## Residual deviance: 118.17  on 856  degrees of freedom
## AIC: 134.17
##
## Number of Fisher Scoring iterations: 9
```

The Summary shows Std. Error terms that are much more reasonable, and only includes variables that are significant at least to the 5% level or higher. The forward stepwise variable selection chose 7 out of the 81 variables that were available to it, which allows for a much more interpretable model.

Let us see how well it predicts.

```r
cm <- confusionMatrix(factor(ifelse(model.glm.final.test > 0.5,"Yes","No")),y_test)

cm$table
```

```
##           Reference
## Prediction  No Yes
##        No  236   4
##        Yes   4  17
```

```r
cm$overall[1]
```

```
##  Accuracy
## 0.9693487
```

```r
cm$overall[5]
```

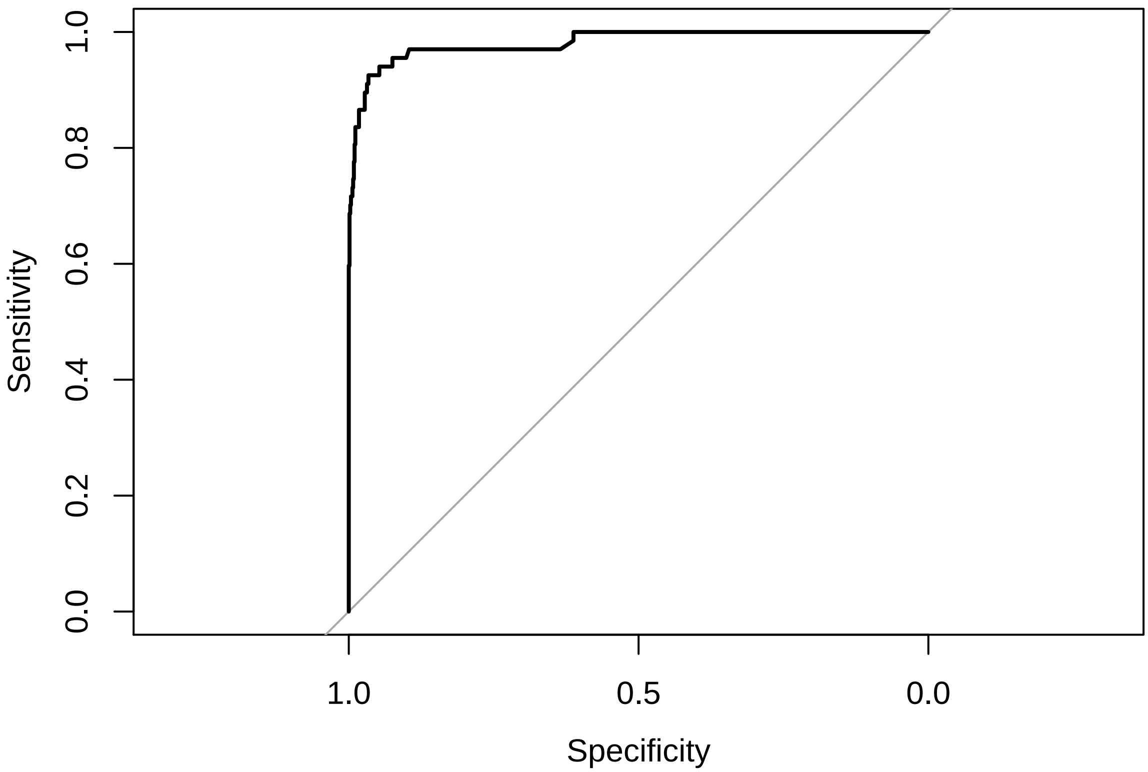
```
## AccuracyNull
##    0.9195402
```

We can see that accuracy (~97%) is appoximately 3% better than the accuracy of our manually subsetted logistic regression model from above. We aso see from the confusion table, that our True Negative rate for this model is (~98%), which is ~1% higher than the subsetted model above. Our True Positive rate in this model is ~81%, showing a dramatic increase of ~19 percentage points over the subsetted model above.

The average accuracy from the cross-validation approach (96.89%) is comparable to the accuracy shown in our model trained only from a single training set (96.93%). The standard deviation of the cross-validation accuracy is also quite low, suggesting that our model is not overfitting our data.

We still want to see how well our model discriminates at different thresholds:

```r
# ROC Curve
prob=plogis(predict.glm(model.glm.final, type = c("response")))
#head(prob)
h <- roc(Oscars_won_some~prob, data=oscars_train)
plot(h)
```




```
# Area Under the Curve
auc(h)
```

```
## Area under the curve: 0.9815
```

The ROC plot shows a definite improvement relative to the previous model, and the AUC is increased by ~3 percentage points (total of ~98%).

We also need to confirm that we aren't seeing a massive amount of multicollinearity:

```
vif(model.glm.final)
```

```
##                                        awards_wins
##                                           5.008870
##                             Screen_Actors_Guild_won
##                                           1.377229
##                                  Golden_Globes_won
##                                           1.530827
##                                        genre_Drama
##                                           1.175590
##                                  Writers_Guild_won
##                                           1.697041
## Online_Film_Television_Association_nominated
##                                           4.987842
##                                          BAFTA_won
##                                           1.862242
```

Due to the large values for `award_wins` and `Online_Film_Television_Association_nominated`, we should probably repeat the stepwise regression process using a subset of the variables that aren't the variables that are aggregated into `award_wins`, while keeping `award_wins` available for use by the stepwise variable selection process.

```r
oscars_train2 <- oscars_train %>% select(-matches(".*nominated$"), -matches(".*won$"))
oscars_test2 <- oscars_test %>% select(-matches(".*nominated$"), -matches(".*won$"))

# creating a null model
model.glm.null2 <-  glm(Oscars_won_some~1,
                family = binomial(link = "logit"), data = oscars_train2)
# create a full model
model.glm.all2 <- glm(Oscars_won_some~.,
                family = binomial(link = "logit"), data = oscars_train2)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
# use step to apply stepwise forward selection with BIC as a evaluation metric
model.glm.final2 <- step(model.glm.null2, scope = formula(model.glm.all2), direction = "forward", k = lo
# NOTE: The output says AIC, but we really are calculating BIC, because we are using the log of the num
```

```r
summary(model.glm.final2)
```

```
##
## Call:
## glm(formula = Oscars_won_some ~ awards_wins + genre_Drama + genre_Biography +
##     certificate, family = binomial(link = "logit"), data = oscars_train2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1981  -0.2238  -0.0890  -0.0425   3.2912
##
## Coefficients:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)          -6.0009     0.8802  -6.817 9.28e-12 ***
## awards_wins           1.5876     0.1667   9.523  < 2e-16 ***
## genre_DramaYes        2.4076     0.8961   2.687 0.007215 **
## genre_BiographyYes    1.6176     0.4346   3.722 0.000197 ***
## certificate           0.7019     0.2824   2.485 0.012944 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 471.29  on 863  degrees of freedom
## Residual deviance: 214.04  on 859  degrees of freedom
## AIC: 224.04
##
## Number of Fisher Scoring iterations: 8
```

Restricting our variable selection to reduce the multicollinearity we saw before, has the step-wise variable selection giving us a final model with only 4 variables. We discovered that all of the 'non-oscar'_nominated and 'non-oscar'_won categories sum to the values in `awards_nominations` and `awards_wins` during the EDA process, therefore we realize that it is likely inappropriate to keep them all in the model while also including `awards_wins` and/or `awards_nominations`. `awards_wins` is still statistically significant in both models.

```r
model.glm.final2.test <- predict.glm(model.glm.final2,oscars_test2,type='response')
cm <- confusionMatrix(factor(ifelse(model.glm.final2.test > 0.5,"Yes","No")),y_test)

cm$table
```

```
##           Reference
## Prediction  No Yes
##        No  235   9
##        Yes   5  12
```

```
cm$overall[1]
```

```
##  Accuracy
## 0.9463602
```

```
cm$overall[5]
```

```
## AccuracyNull
##    0.9195402
```

Our Accuracy is still above the No Information Rate (AccuracyNull above), as with the previous model, but we did see a reduction of the overall prediction accuracy and true positive rates compared to the other step-wise model.

```
vif(model.glm.final2)
```

```
##     awards_wins     genre_Drama genre_Biography     certificate
##        1.149884        1.120068        1.119833        1.153924
```

The VIF statistic is improved relative to the previous model, as none of the variables have VIF scores above 2.

Cross Validation

Let's use 3-fold stratified cross-validation to test the variance of our predictive accuracy. We chose stratified cross-validation to ensure the proportion of class labels is maintained in each of the splits. We chose 3-folds instead of the standard 10-folds, because if we were to use smaller splits we'd risk having too few degrees of freedom to train the model, given the scarcity of the oscar winners.

```
names(model.glm.final2$coefficients)
```

```
## [1] "(Intercept)"        "awards_wins"        "genre_DramaYes"
## [4] "genre_BiographyYes" "certificate"
```

```
# Generates a list of indexes for each stratified split
folds <- KFold(oscars$Oscars_won_some, n = 3, stratified = TRUE, seed = 42)
# Gets a list with the number of folds
list_ = 1:length(folds)
# Gets a list to store the accuracy for each split
scores = c()
# For each fold
for(i in list_){
  # Gets the indexes for getting the training data
  list_train = list_[-i]
  train_index = c(folds[[list_train[1]]], folds[[list_train[2]]])
  # Gets the index for the testing data
  test_index = folds[[i]]
  # Splits between training and testing
  cv.train = oscars[train_index, ]
  cv.test = oscars[test_index, ]
  # Trains the model
  logreg.train <- glm(Oscars_won_some~awards_wins+genre_Drama+genre_Biography+certificate, family = bin
      data = cv.train)
  # Tests the model
```

```r
  glm.final.test <- predict.glm(logreg.train,cv.test,type='response')
  score <- sum(ifelse(glm.final.test > 0.5,"Yes","No")
             ==cv.test$Oscars_won_some)/
  length(cv.test$Oscars_won_some)
  # Appends the score to the list
  scores = c(scores, score)
}
scores
```

```
## [1] 0.9627660 0.9441489 0.9517426
```

```r
mean(scores)
```

```
## [1] 0.9528858
```

```r
sd(scores)
```

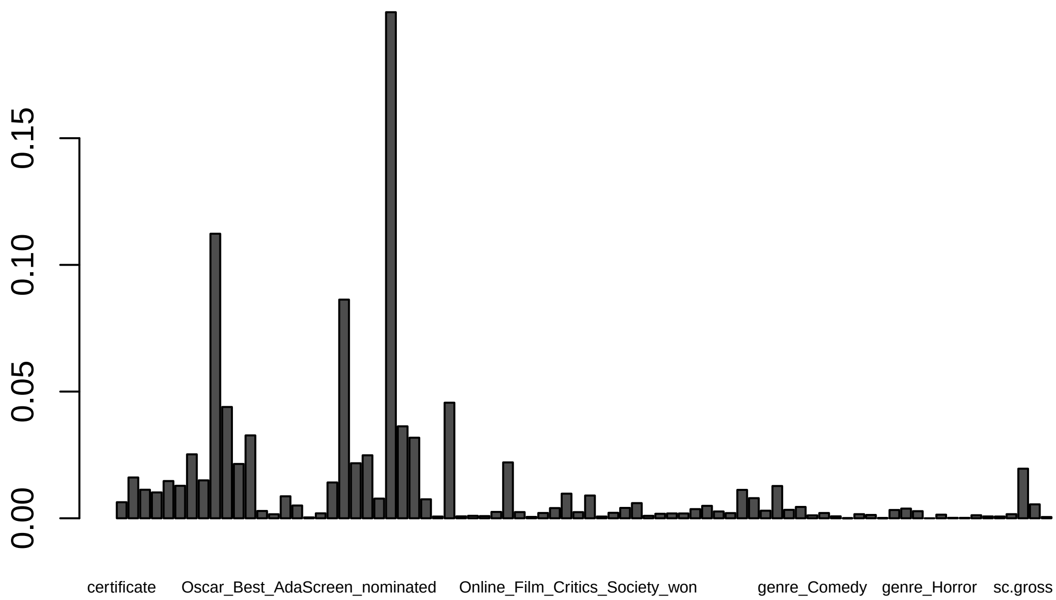
```
## [1] 0.009361013
```

Random Forest

Now we'll attempt to train a random forest model based on all the variables, and tune the hyperparameter mtry (the number of features used at each split).

```r
# apparently randomForest function can't use `-` in variable names.
oscars_train.renamed <- oscars_train %>%
  mutate(genre_Sci_Fi = `genre_Sci-Fi`) %>%
  select(-`genre_Sci-Fi`)
oscars_test.renamed <- oscars_test %>%
  mutate(genre_Sci_Fi = `genre_Sci-Fi`) %>%
  select(-`genre_Sci-Fi`)

# Trains the model (low, med, high mtry numbers)
ranFor.train <- function(mtry) {
  set.seed(42)
  return(randomForest(formula=Oscars_won_some~.,
               importance=TRUE,
               proximity=TRUE,
               mtry=mtry,
               data=oscars_train.renamed))
}

# Tests the model
ranFor.hitrate <- function (model) {
  y_test_pred <- predict(model,oscars_test.renamed, type='response')
  return(sum(y_test_pred==y_test)/length(y_test))
}

# These take a while to run...
#scores = c()
#for (i in seq(1,80)) {
#  scores <- c(scores, ranFor.hitrate(ranFor.train(i)))
#}
#best.mtry <- which(max(scores)==scores)[1]
# after running above, 35, 49, 67, 73 give max performance
```

```r
# to avoid really long document rebuild times... we'll just use 35
best.mtry <- 35

best.ranFor.model <- ranFor.train(best.mtry)

# Computes the importance of each variable
# by accuracy
VI_F1 = importance(best.ranFor.model, type=1)
# by impurity
VI_F2 = importance(best.ranFor.model, type=2)

# https://freakonometrics.hypotheses.org/19835
barplot(t(VI_F2/sum(VI_F2)), cex.names=0.5)
```




```r
ranFor.hitrate(best.ranFor.model)
```

```
## [1] 0.9808429
```

```r
# confusion matrix
y_test_pred <- predict(best.ranFor.model,oscars_test.renamed, type='response')
confusionMatrix(y_test_pred, y_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  No Yes
##        No  238   3
##        Yes   2  18
##
##                Accuracy : 0.9808
##                  95% CI : (0.9559, 0.9938)
##     No Information Rate : 0.9195
##     P-Value [Acc > NIR] : 1.96e-05
##
##                   Kappa : 0.8677
##  Mcnemar's Test P-Value : 1
```

```
##
##             Sensitivity : 0.9917
##             Specificity : 0.8571
##          Pos Pred Value : 0.9876
##          Neg Pred Value : 0.9000
##              Prevalence : 0.9195
##          Detection Rate : 0.9119
##    Detection Prevalence : 0.9234
##       Balanced Accuracy : 0.9244
##
##        'Positive' Class : No
##
```

Random Forest improves our prediction accuracy over the stepwise logistic regression model.

Conclusions