Video-game-classification

Wenfei Yan

4/2/2018

Origin Question:

Can we predict whether the game is rated as "positive", "mixed", or "negative"?

We look in the the user score, with range [0,10]. Since user score is a continuous variable, we tried to classify games with user score in (7.5, 10] as positive, in (5.0. 7.5] as mixed, and [0,5.0] as negative.

```
# clean the data
setwd("/Users/Fay/2017-2018_Winter/STATS 415/415Project")
games = read.csv("./data/Video_Games_Sales_as_at_22_Dec_2016.csv")
games.no.na = na.omit(games)
games.no.na = games.no.na[,-1]
# str(games.no.na)
games.no.na$User_Score = as.character(games.no.na$User_Score)
games.no.na$User_Score = as.numeric(games.no.na$User_Score)
str(games.no.na)
```

```
## 'data.frame':    7017 obs. of  15 variables:
##  $ Platform       : Factor w/ 31 levels "2600","3DO","3DS",..: 26 26 26 5 26 26 5 26 29 26 ...
##  $ Year_of_Release: Factor w/ 40 levels "1980","1981",..: 27 29 30 27 27 30 26 28 31 30 ...
##  $ Genre          : Factor w/ 13 levels "","Action","Adventure",..: 12 8 12 6 5 6 8 12 5 12 ...
##  $ Publisher      : Factor w/ 582 levels "10TACLE Studios",..: 371 371 371 371 371 371 371 371 330 37
##  $ NA_Sales       : num  41.4 15.7 15.6 11.3 14 ...
##  $ EU_Sales       : num  28.96 12.76 10.93 9.14 9.18 ...
##  $ JP_Sales       : num  3.77 3.79 3.28 6.5 2.93 4.7 4.13 3.6 0.24 2.53 ...
##  $ Other_Sales    : num  8.45 3.29 2.95 2.88 2.84 2.24 1.9 2.15 1.69 1.77 ...
##  $ Global_Sales   : num  82.5 35.5 32.8 29.8 28.9 ...
##  $ Critic_Score   : int  76 82 80 89 58 87 91 80 61 80 ...
##  $ Critic_Count   : int  51 73 73 65 41 80 64 63 45 33 ...
##  $ User_Score     : num  8 8.3 8 8.5 6.6 8.4 8.6 7.7 6.3 7.4 ...
##  $ User_Count     : int  322 709 192 431 129 594 464 146 106 52 ...
##  $ Developer      : Factor w/ 1697 levels "","10tacle Studios",..: 1035 1035 1035 1035 1035 1035 1035
##  $ Rating         : Factor w/ 9 levels "","AO","E","E10+",..: 3 3 3 3 3 3 3 3 3 3 ...
```

```
# dim(games.no.na)
# add user.score.level
games.no.na$user.score.level = "negative"
games.no.na$user.score.level[games.no.na$User_Score>5] = "mixed"
games.no.na$user.score.level[games.no.na$User_Score>7.5] = "positive"
sum(games.no.na$User_Score>7.5)
```

```
## [1] 3424
```

```
sum(games.no.na$User_Score>5 & games.no.na$User_Score<=7.5)
```

```
## [1] 2965
```

```
sum(games.no.na$User_Score<=5)
```

```
## [1] 628
```

However, we found that the three classes are unbalanced, which will lead problmes in prediciton. The "negative" class has too few games, which means it might be kind of "ignored" in the classifier. Although there're several techiniques to combact unblaneced classes problem, it is beyond the scope of this course. We simply change our classes to prevent this problem. We change the previous question to ## New Question: Can we predict whether the game is rated as "positive" or not?
A game with user score in (7.5, 10] is considered to be rated as "positive"", and"not positive" otherwise.

```
# delete user.score.level
games.no.na = games.no.na[, -which(names(games.no.na)=="user.score.level")]
# add user.score.positive
games.no.na$user.score.positive = games.no.na$User_Score > 7.5
games.no.na$user.score.positive = as.factor(games.no.na$user.score.positive)
```

```
# split training and test data
positive = which(games.no.na$user.score.positive == TRUE)
not.positive = which(games.no.na$user.score.positive == FALSE)
train.id = c(sample(positive, floor(0.8*length(positive))),
             sample(not.positive, floor(0.8*length(not.positive))))
```

KNN

First I tried KNN. As all the categorical variables have many levels, it's hard to use them in KNN. I just omit all of them.

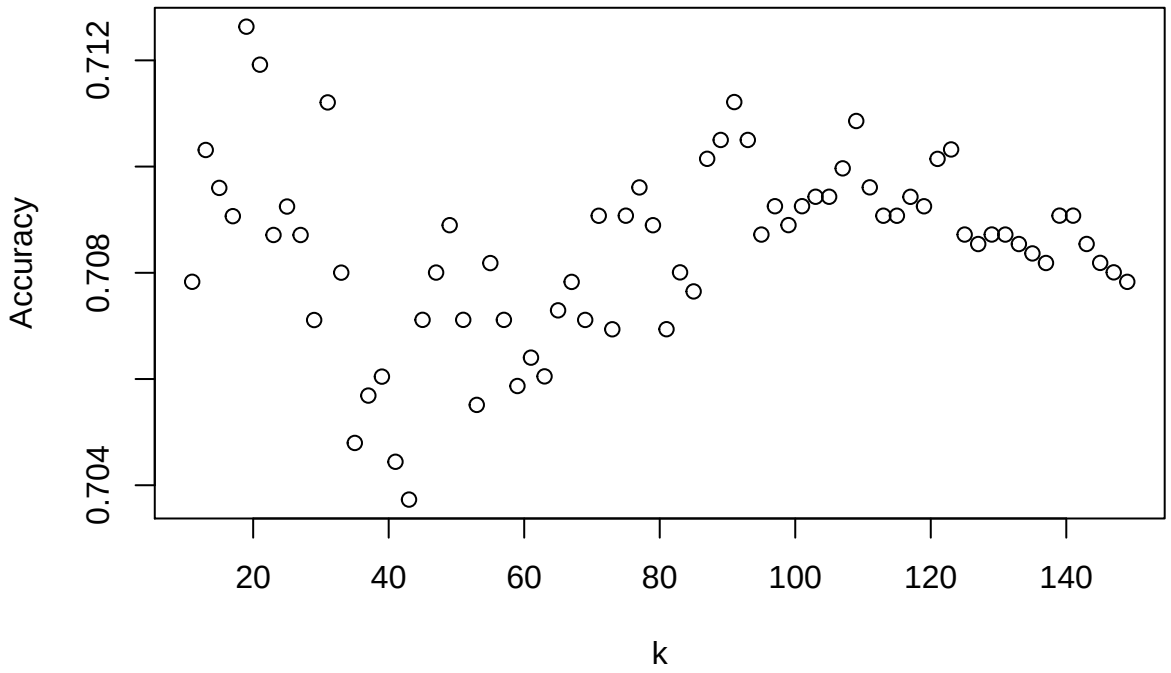
```
# knn
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.4.4
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
data.knn = games.no.na[,c(5,6,7,8,9,10,11,13,16)]
# head(data.knn)
# 10-fold cv
ctrl <- trainControl(method  = "cv", number  = 10)
knn.fit <- train(user.score.positive ~ ., data = data.knn[train.id,], method = "knn",
                 tuneGrid   = expand.grid(k = seq(11,149,2)),
                 trControl = ctrl, preProcess = c("center","scale"))
plot(Accuracy~k, data = knn.fit$results)
```




```
best = knn.fit$result[knn.fit$result$Accuracy == max(knn.fit$results$Accuracy),]
1 - best$Accuracy
```

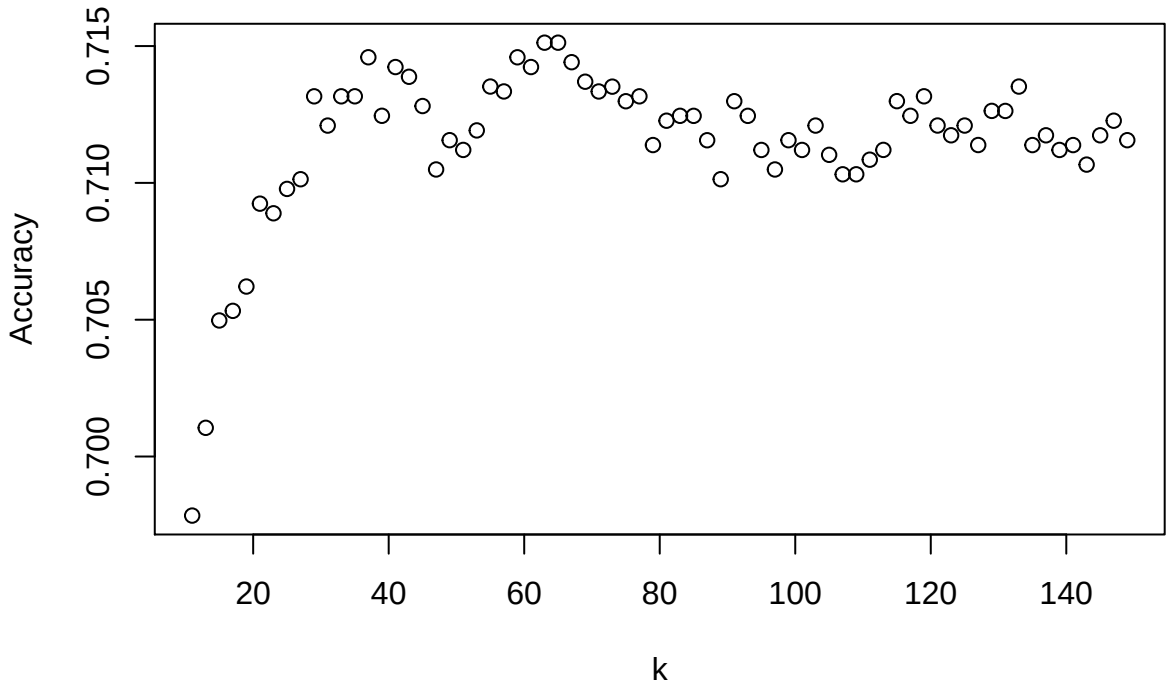
```
## [1] 0.2873658
```

The cv-accuracy is not satysfying. Try dim-reduction (PCA), see if better.

```
X <- model.matrix(user.score.positive ~ ., data = data.knn[train.id,])[, -1]
knn.PCA = prcomp(x = X, center = T, scale = T)
summary(knn.PCA)
```

```
## Importance of components%s:
##                           PC1    PC2    PC3     PC4     PC5     PC6
## Standard deviation     2.0064 1.1326 0.8968 0.81325 0.76674 0.64203
## Proportion of Variance 0.5032 0.1604 0.1005 0.08267 0.07349 0.05153
## Cumulative Proportion  0.5032 0.6636 0.7641 0.84677 0.92026 0.97178
##                           PC7      PC8
## Standard deviation     0.47511 0.003083
## Proportion of Variance 0.02822 0.000000
## Cumulative Proportion  1.00000 1.000000
```

```
data.knn.PCA = as.data.frame(knn.PCA$x[,c(1,2,3,4)])
data.knn.PCA$user.score.positive = data.knn$user.score.positive[train.id]
# head(data.knn.PCA)
ctrl <- trainControl(method  = "cv", number  = 10)
knn.fit.PCA <- train(user.score.positive ~ ., data = data.knn.PCA, method = "knn",
                     tuneGrid   = expand.grid(k = seq(11,149,2)),
                     trControl = ctrl, preProcess = c("center","scale"))
plot(Accuracy~k, data = knn.fit.PCA$results)
```




```
best = knn.fit.PCA$result[knn.fit.PCA$result$Accuracy == max(knn.fit.PCA$results$Accuracy),]
1 - best$Accuracy
```

```
## [1] 0.2848764 0.2848764
```

Unfortunately, there's no apparent improvement. KNN might not be suitable.

Desicion Tree

As the tree funtion in pacakge tree can not deal with factors with more than 32 levels, I delete some of the facotr predictors, who have too many levels. Year_of_Release, Publisher and Developer are deleted.

```
library(tree)
```

```
## Warning: package 'tree' was built under R version 3.4.4
```

```
# names(games.no.na)
data_tree = games.no.na[, -c(2,4,12,14)]
# data_tree = games.no.na[, c(9,10,16)]
# head(data_tree)
tree.fit = tree(user.score.positive ~ ., data = data_tree[train.id,])
tree.fit
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 5613 7778.0 FALSE ( 0.51203 0.48797 )
##    2) Critic_Score < 74.5 3138 3868.0 FALSE ( 0.69344 0.30656 )
##      4) Platform: 3DS,DS,PC,PS3,PS4,PSP,PSV,Wii,WiiU,X360,XOne 2036 2086.0 FALSE ( 0.79126 0.20874 )
##        8) Critic_Score < 61.5 875  544.3 FALSE ( 0.90629 0.09371 ) *
##        9) Critic_Score > 61.5 1161 1409.0 FALSE ( 0.70457 0.29543 ) *
##      5) Platform: GBA,GC,PS,PS2,XB 1102 1527.0 FALSE ( 0.51270 0.48730 )
##       10) Critic_Score < 54.5 258  232.5 FALSE ( 0.83333 0.16667 ) *
##       11) Critic_Score > 54.5 844 1145.0 TRUE ( 0.41469 0.58531 ) *
```

```
##    3) Critic_Score > 74.5 2475 2945.0 TRUE ( 0.28202 0.71798 )
##      6) Platform: DS,PC,PS3,PS4,WiiU,X360,XOne 1282 1727.0 TRUE ( 0.40172 0.59828 ) *
##      7) Platform: 3DS,DC,GBA,GC,PS,PS2,PSP,PSV,Wii,XB 1193 1023.0 TRUE ( 0.15339 0.84661 ) *
```

```
summary(tree.fit)
```

```
##
## Classification tree:
## tree(formula = user.score.positive ~ ., data = data_tree[train.id,
##     ])
## Variables actually used in tree construction:
## [1] "Critic_Score" "Platform"
## Number of terminal nodes:  6
## Residual mean deviance:  1.085 = 6081 / 5607
## Misclassification error rate: 0.2701 = 1516 / 5613
```

```
# set.seed(2109)
# tree.fit.cv = cv.tree(tree.fit, FUN = prune.misclass)
# tree.fit.cv
prune.tree.fit = prune.misclass(tree.fit, best = 6)
summary(prune.tree.fit)
```

```
##
## Classification tree:
## tree(formula = user.score.positive ~ ., data = data_tree[train.id,
##     ])
## Variables actually used in tree construction:
## [1] "Critic_Score" "Platform"
## Number of terminal nodes:  6
## Residual mean deviance:  1.085 = 6081 / 5607
## Misclassification error rate: 0.2701 = 1516 / 5613
```

```
# test error
tree.pred=predict(prune.tree.fit, data_tree[-train.id,], type="class")
table(tree.pred,data_tree$user.score.positive[-train.id])
```

```
##
## tree.pred FALSE TRUE
##     FALSE   473  170
##     TRUE    246  515
```

```
sum(tree.pred!=data_tree$user.score.positive[-train.id])/1404
```

```
## [1] 0.2962963
```

The training and test error are both around 0.278, a little bit better than knn, but still not ideal.

Random Forest

As the randomForest funtion in pacakge randomForest can not deal with factors with more than 53 levels, I delete some of the facotr predictors, who have too many levels. Publisher and Developer are deleted.

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.4.4
```

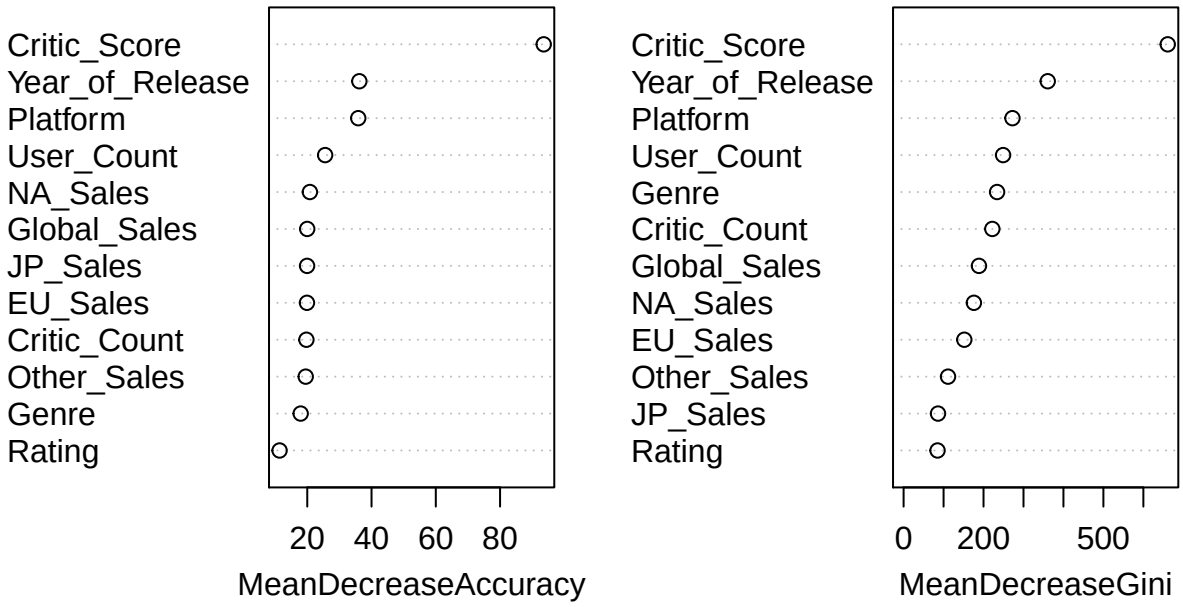
```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
data_rf = games.no.na[, -c(4, 12, 14)]
# n_range = seq(100,1000, by = 10)
# test_error = rep(0, length(n_range))
# for (i in 1:length(n_range)) {
# rf.fit = randomForest(user.score.positive ~ ., data = data_rf[train.id,], ntree = n_range[i])
# rf.fit
# ypred.rf = predict(rf.fit,newdata = data_rf[-train.id,])
# # table(ypred.rf, data_tree$user.score.positive[-train.id])
# test_error[i] = sum(ypred.rf!=data_tree$user.score.positive[-train.id])/1404
# }
# min(test_error)
# n_range[which(test_error == min(test_error))]
# plot(n_range, test_error)
rf.fit = randomForest(user.score.positive ~ ., data = data_rf[train.id,], ntree = 260, importance = TRU
importance(rf.fit)
```

```
##                     FALSE       TRUE MeanDecreaseAccuracy MeanDecreaseGini
## Platform        47.211139 -1.7036262             35.86811        272.40777
## Year_of_Release 50.868302 -0.8553625             36.20773        360.68118
## Genre           21.077707  3.2552605             17.95201        233.96141
## NA_Sales        12.234839 11.9734474             20.82773        175.88798
## EU_Sales        15.212576  9.6686594             19.90804        151.83750
## JP_Sales        19.861526  3.2048506             19.94585         86.08807
## Other_Sales     15.322308  4.8234657             19.51801        111.18790
## Global_Sales    12.669246 11.3704714             19.99019        188.58355
## Critic_Score    69.557598 77.0969544             93.58409        660.99732
## Critic_Count     9.463699 15.4500023             19.73806        221.99120
## User_Count      23.174847  9.5788907             25.55905        248.96273
## Rating           8.084645  7.0906899             11.39212         84.94538
```

```
varImpPlot(rf.fit)
```




```
ypred.rf = predict(rf.fit,newdata = data_rf[-train.id,])
sum(ypred.rf!=data_tree$user.score.positive[-train.id])/1404
```

```
## [1] 0.2450142
```

The best test error is around 0.225. However, this is still not desirable.

Critic score is always the most important predictor.

Ada boost

```
library(gbm)
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:caret':
##
##     cluster
```

```
## Loading required package: splines
```

```
## Loading required package: parallel
```

```
## Loaded gbm 2.1.3
```

```
data_boost = games.no.na[, -c(4, 12, 14)]
data_boost$user.score.positive = as.numeric(data_boost$user.score.positive)-1
boost.fit = gbm(user.score.positive ~ ., data = data_boost[train.id,], distribution = "adaboost", n.tre
boost.test.pred = predict(boost.fit, newdata = data_boost[-train.id,], n.trees = 2700)
```

```
boost.test.pred = sign(boost.test.pred)
boost.test.pred[boost.test.pred==-1] = 0
sum(boost.test.pred != data_boost$user.score.positive[-train.id])/1404
```

```
## [1] 0.2799145
```

The best test error is around 0.255, worse than the random forest result.

SVM

```
library(e1071)
data_svm = games.no.na[,c(5,6,7,8,9,10,11,13,16)]
svm.fit <- svm(user.score.positive ~ ., data = data_svm[train.id,],
               kernel = "linear", cost = 10, scale = TRUE)
svm.test.pred = predict(svm.fit, newdata = data_svm[-train.id,])
sum(svm.test.pred != data_svm$user.score.positive[-train.id])/1404
```

```
## [1] 0.284188
```

```
svm.fit.rad <- svm(user.score.positive ~ ., data = data_svm[train.id,],
               kernel = "radial", cost = 5, scale = TRUE)
svm.test.pred = predict(svm.fit.rad, newdata = data_svm[-train.id,])
sum(svm.test.pred != data_svm$user.score.positive[-train.id])/1404
```

```
## [1] 0.2827635
```

The test error is even higher with svm. . . .