Predictive Models for the player logs dataset

Nitish Rangarajan

November 22, 2017

Models for predicting player unsubscription

After learning about the data, the Deep Learning Network was used with the Keras library in python. Keras is a high level neural networks API that is written in Python and developed with a focus on enabling fast experimentation.

The sequential deep learning model in Keras was used since that is best suited to classify human actions with limited knowledge.

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 3.3.3
```

```
## -- Attaching packages ---------------------------------------------------------------------
--------- tidyverse 1.2.1 --
```

```
## v ggplot2 2.2.1     v purrr   0.2.4
## v tibble  1.3.4     v dplyr   0.7.4
## v tidyr   0.7.2     v stringr 1.2.0
## v readr   1.1.1     v forcats 0.2.0
```

```
## Warning: package 'ggplot2' was built under R version 3.3.3
```

```
## Warning: package 'tibble' was built under R version 3.3.3
```

```
## Warning: package 'tidyr' was built under R version 3.3.3
```

```
## Warning: package 'readr' was built under R version 3.3.3
```

```
## Warning: package 'purrr' was built under R version 3.3.3
```

```
## Warning: package 'dplyr' was built under R version 3.3.3
```

```
## Warning: package 'stringr' was built under R version 3.3.3
```

```
## Warning: package 'forcats' was built under R version 3.3.3
```

```
## -- Conflicts ------------------------------------------------------------------------------
--- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(lattice)
library(keras)
```

```
## Warning: package 'keras' was built under R version 3.3.3
```

Load the dataset and convert the timestamp into MMDDYY format.

```
setwd("C:/Users/nrangara/Downloads/WorldOfWarcraft/output")
data<- read_csv("wowah_data.csv")
```

```
## Parsed with column specification:
## cols(
##   char = col_integer(),
##   level = col_integer(),
##   race = col_character(),
##   charclass = col_character(),
##   zone = col_character(),
##   dummy1 = col_character(),
##   dummy2 = col_integer(),
##   guild = col_integer(),
##   timestamp = col_character()
## )
```

```
data<- data%>% mutate(Date=as.Date(data$timestamp, "%m/%d/%y"))
```

```
## Warning: package 'bindrcpp' was built under R version 3.3.3
```

To predict the unsubscription, a deep learning model has been built that will be trained on the data from 2005 to 2007 to predict the unsubscription for the year 2008-2009. So, the player by total number of entries, number the days online from 2005 to 2007 with the last level, the minimum and the maximum guild were grouped. These are my x variables.

```
threshold<-0
x <- filter(data, Date<"2008-01-01") %>%
  group_by(char) %>%
  summarise(n=n(),
            minGuild=min(guild),
            maxGuild=max(guild),
            minDate = min(Date),
            maxDate = max(Date),
            maxLevel=max(level)) %>%
  arrange(desc(n)) %>%
  mutate(TimeDiff = as.numeric(difftime(maxDate,minDate, units="days")))
```

For the y variable in the network, a binary variable was created that denotes if the user was online during 2008-2009 or not.

```
y <- data %>%
  mutate(n2008=ifelse(Date>=as.Date("2008-01-01"),1L,0L)) %>%
  group_by(char) %>%
  summarise(yCount = sum(n2008)) %>%
  mutate(y=ifelse(yCount>threshold,1,0))
```

We then merge the x and the y variables to form the train and he test dataset.

```
dataset <- merge(x,y,by="char")
```

Remove the date field since we already have the n field that denotes the number of days the user was online. There were totally 91,045 avatar records. The training data that contained 80% of the avatars and test data that had 20% of the avatars was created.

```
dataset[c("minDate","maxDate")]<-list(NULL)

set.seed(101)
sample <- sample.int(n = nrow(dataset), size = floor(.80*nrow(dataset)), replace = F)
train <- dataset[sample, ]
test  <- dataset[-sample, ]
x_train<-train[,2:7]
#y_train<-as.factor(train[,10])
y_train<-train[,8]
x_test<-test[,2:7]
#y_test<-as.factor(test[,10])
y_test<-test[,8]
```

Convert the train and test data frames to matrices for the deep learning network.

```
x_train1<-as.matrix(x_train)
y_train1<-as.matrix(y_train)
x_test1<-as.matrix(x_test)
y_test1<-as.matrix(y_test)
```

Deep Learning model

Reduce the x variable's dimensions to 6 and make the y variable to be catagorical.

```
dim(x_train1) <- c(nrow(x_train), 6)
dim(x_test1) <- c(nrow(x_test), 6)

y_train1 <- to_categorical(y_train, 2)
y_test1 <- to_categorical(y_test, 2)
```

A sequential model was created with 5 hidden layers with the first hidden layer containing 512 units, second hidden layer containing 256 units, third hidden layer containing 128 units, fourth hidden layer containing 64 units and used the "relu" activation function. The last hidden layer contained 2 units because of the size of y and used the "softmax" activation function.

```
model <- keras_model_sequential()
model %>%
  layer_dense(units = 512, activation = "relu", input_shape = c(6)) %>%
  layer_dropout(rate = 0.6) %>%
  layer_dense(units = 256, activation = "relu") %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = "relu") %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 2, activation = "softmax")
```

RMSprop increases the step rates, keep the learning rate constant by exponentially decaying the average of squared gradients.

```
optimizer <- optimizer_rmsprop(lr = 0.01)
```

The model's summary can be seen below. After verifying the model, compile the model.

```
summary(model)
```

```
## ____________________________________________________________________________
## Layer (type)                     Output Shape                  Param #
## ============================================================================
## dense_1 (Dense)                  (None, 512)                   3584
## ____________________________________________________________________________
## dropout_1 (Dropout)              (None, 512)                   0
## ____________________________________________________________________________
## dense_2 (Dense)                  (None, 256)                   131328
## ____________________________________________________________________________
## dropout_2 (Dropout)              (None, 256)                   0
## ____________________________________________________________________________
## dense_3 (Dense)                  (None, 128)                   32896
## ____________________________________________________________________________
## dropout_3 (Dropout)              (None, 128)                   0
## ____________________________________________________________________________
## dense_4 (Dense)                  (None, 64)                    8256
## ____________________________________________________________________________
## dropout_4 (Dropout)              (None, 64)                    0
## ____________________________________________________________________________
## dense_5 (Dense)                  (None, 2)                     130
## ============================================================================
## Total params: 176,194
## Trainable params: 176,194
## Non-trainable params: 0
## ____________________________________________________________________________
```

```
model %>% compile(
  loss = "categorical_crossentropy",
  optimizer = optimizer_rmsprop(),
  metrics = c("accuracy")
)
```

Fit the model that was created with the training data. To fit the model using the training data, 50 epochs were used and a validation split of 0.2 to prevent overfitting

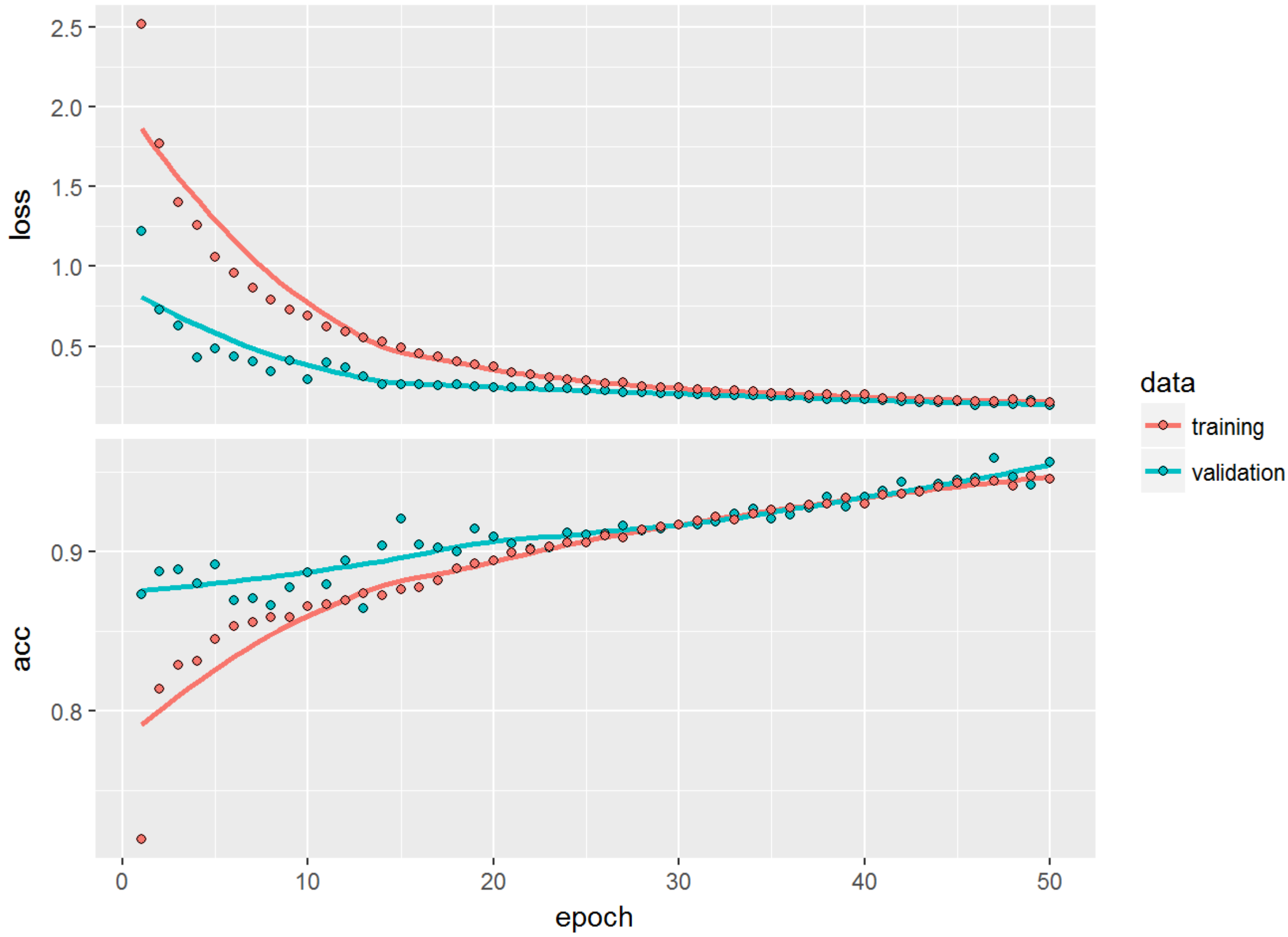
```
history <- model %>% fit(
  x_train1, y_train1,
  epochs = 50,batch_size = nrow(x_test1),
  validation_split = 0.2
)
```

```
##Train on 41940 samples, validate on 10485 samples
##Epoch 1/50
##41940/41940 [==============================] - 2s 53us/step - loss: 0.2706 - acc: 0.9447 - val
_loss: 0.1240 - val_acc: 0.9499
##Epoch 2/50
##41940/41940 [==============================] - 2s 44us/step - loss: 0.1247 - acc: 0.9544 - val
_loss: 0.1080 - val_acc: 0.9554
##Epoch 3/50
##41940/41940 [==============================] - 2s 48us/step - loss: 0.1323 - acc: 0.9542 - val
_loss: 0.1237 - val_acc: 0.9453
##Epoch 4/50
##41940/41940 [==============================] - 2s 44us/step - loss: 0.1186 - acc: 0.9550 - val
_loss: 0.0999 - val_acc: 0.9592
##Epoch 5/50
##41940/41940 [==============================] - 2s 44us/step - loss: 0.1219 - acc: 0.9566 - val
_loss: 0.1173 - val_acc: 0.9467
##Epoch 6/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.1175 - acc: 0.9563 - val
_loss: 0.0970 - val_acc: 0.9639
##Epoch 7/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.1291 - acc: 0.9566 - val
_loss: 0.1239 - val_acc: 0.9451
##Epoch 8/50
##41940/41940 [==============================] - 2s 44us/step - loss: 0.1139 - acc: 0.9553 - val
_loss: 0.0914 - val_acc: 0.9616
##Epoch 9/50
##41940/41940 [==============================] - 2s 44us/step - loss: 0.1120 - acc: 0.9606 - val
_loss: 0.1152 - val_acc: 0.9433
##Epoch 10/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.1143 - acc: 0.9536 - val
_loss: 0.0921 - val_acc: 0.9622
##Epoch 11/50
##41940/41940 [==============================] - 2s 54us/step - loss: 0.0978 - acc: 0.9649 - val
_loss: 0.1168 - val_acc: 0.9505
##Epoch 12/50
##41940/41940 [==============================] - 2s 44us/step - loss: 0.1248 - acc: 0.9599 - val
_loss: 0.0905 - val_acc: 0.9676
##Epoch 13/50
##41940/41940 [==============================] - 2s 44us/step - loss: 0.0939 - acc: 0.9678 - val
_loss: 0.0921 - val_acc: 0.9783
##Epoch 14/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.1144 - acc: 0.9607 - val
_loss: 0.0880 - val_acc: 0.9665
##Epoch 15/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.0885 - acc: 0.9693 - val
_loss: 0.1114 - val_acc: 0.9588
##Epoch 16/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.1345 - acc: 0.9504 - val
_loss: 0.1009 - val_acc: 0.9562
##Epoch 17/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.0932 - acc: 0.9613 - val
_loss: 0.0797 - val_acc: 0.9708
```

```
##Epoch 18/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.0822 - acc: 0.9707 - val
_loss: 0.0791 - val_acc: 0.9823
##Epoch 19/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.1093 - acc: 0.9662 - val
_loss: 0.0844 - val_acc: 0.9720
##Epoch 20/50
##41940/41940 [==============================] - 2s 48us/step - loss: 0.0800 - acc: 0.9725 - val
_loss: 0.0762 - val_acc: 0.9835
##Epoch 21/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.1150 - acc: 0.9677 - val
_loss: 0.0831 - val_acc: 0.9675
##Epoch 22/50
##41940/41940 [==============================] - 2s 44us/step - loss: 0.0758 - acc: 0.9731 - val
_loss: 0.0681 - val_acc: 0.9711
##Epoch 23/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.1339 - acc: 0.9595 - val
_loss: 0.1049 - val_acc: 0.9572
##Epoch 24/50
##41940/41940 [==============================] - 2s 50us/step - loss: 0.0925 - acc: 0.9616 - val
_loss: 0.0822 - val_acc: 0.9642
##Epoch 25/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.0743 - acc: 0.9701 - val
_loss: 0.0610 - val_acc: 0.9791
##Epoch 26/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.0661 - acc: 0.9766 - val
_loss: 0.0861 - val_acc: 0.9722
##Epoch 27/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.1404 - acc: 0.9580 - val
_loss: 0.0906 - val_acc: 0.9616
##Epoch 28/50
##41940/41940 [==============================] - 2s 45us/step - loss: 0.0804 - acc: 0.9665 - val
_loss: 0.0709 - val_acc: 0.9716
##Epoch 29/50
##41940/41940 [==============================] - 2s 45us/step - loss: 0.0664 - acc: 0.9765 - val
_loss: 0.0541 - val_acc: 0.9825
##Epoch 30/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.0632 - acc: 0.9790 - val
_loss: 0.0810 - val_acc: 0.9770
##Epoch 31/50
##41940/41940 [==============================] - 2s 44us/step - loss: 0.1072 - acc: 0.9645 - val
_loss: 0.0768 - val_acc: 0.9689
##Epoch 32/50
##41940/41940 [==============================] - 2s 44us/step - loss: 0.0675 - acc: 0.9733 - val
_loss: 0.0614 - val_acc: 0.9902
##Epoch 33/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.0919 - acc: 0.9727 - val
_loss: 0.0669 - val_acc: 0.9750
##Epoch 34/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.0595 - acc: 0.9793 - val
_loss: 0.0477 - val_acc: 0.9856
##Epoch 35/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.0866 - acc: 0.9723 - val
_loss: 0.0855 - val_acc: 0.9651
```

```
##Epoch 36/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.0749 - acc: 0.9689 - val
_loss: 0.0665 - val_acc: 0.9755
##Epoch 37/50
##41940/41940 [==============================] - 2s 53us/step - loss: 0.0572 - acc: 0.9787 - val
_loss: 0.0476 - val_acc: 0.9866
##Epoch 38/50
##41940/41940 [==============================] - 2s 44us/step - loss: 0.0684 - acc: 0.9792 - val
_loss: 0.0820 - val_acc: 0.9799
##Epoch 39/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.0630 - acc: 0.9798 - val
_loss: 0.0468 - val_acc: 0.9849
##Epoch 40/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.0605 - acc: 0.9806 - val
_loss: 0.0708 - val_acc: 0.9764
##Epoch 41/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.0567 - acc: 0.9803 - val
_loss: 0.0480 - val_acc: 0.9805
##Epoch 42/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.0856 - acc: 0.9763 - val
_loss: 0.0695 - val_acc: 0.9780
##Epoch 43/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.0550 - acc: 0.9813 - val
_loss: 0.0422 - val_acc: 0.9866
##Epoch 44/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.0605 - acc: 0.9814 - val
_loss: 0.0817 - val_acc: 0.9630
##Epoch 45/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.0832 - acc: 0.9695 - val
_loss: 0.0557 - val_acc: 0.9785
##Epoch 46/50
##41940/41940 [==============================] - 2s 48us/step - loss: 0.0503 - acc: 0.9819 - val
_loss: 0.0431 - val_acc: 0.9840
##Epoch 47/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.1104 - acc: 0.9749 - val
_loss: 0.0702 - val_acc: 0.9724
##Epoch 48/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.0588 - acc: 0.9769 - val
_loss: 0.0475 - val_acc: 0.9852
##Epoch 49/50
##41940/41940 [==============================] - 2s 43us/step - loss: 0.0411 - acc: 0.9863 - val
_loss: 0.0392 - val_acc: 0.9835
##Epoch 50/50
##41940/41940 [==============================] - 2s 49us/step - loss: 0.1093 - acc: 0.9710 - val
_loss: 0.0739 - val_acc: 0.9711
```

```
plot(history)
```



The model has an accuracy of 96% with a loss of 0.11 on evaluating it with the test data.

```
model %>% evaluate(x_test1, y_test1,verbose = 0)
```

```
## $loss
## [1] 0.1269096
##
## $acc
## [1] 0.9588006
```

SVM Model

The same training data with the labelled X and Y variables can be fed to the Support Vector Machine as an input and then predict the unsubscription for the test data so that the accuracy of SVM and Deep learning can be compared and the better model can be found.

```
library("e1071")
```

```
## Warning: package 'e1071' was built under R version 3.3.3
```

Create a model and fit the labelled x with labelled y data and summarize the model.

```
svm_model <- svm(y_train~.,data=cbind(x_train,y_train))
summary(svm_model)
```

```
##
## Call:
## svm(formula = y_train ~ ., data = cbind(x_train, y_train))
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.1666667
##     epsilon:  0.1
##
##
## Number of Support Vectors:  14102
```

Now predict the model with the test data. We round the predicted values to 1's and 0's

```
Prediction <- predict(svm_model,x_test)
Prediction<-ifelse(Prediction<0.25,0,1)
```

Accuracy is the number of true negatives and true positives to the total number of observations. Precision is the number of correct observations made from the retrieved observations. Recall is the number of correct observations made from the total correct observations.

```
accuracy <- function(ypred, y){
  tab <- table(ypred, y)
  return(sum(diag(tab))/sum(tab))
}
# function to compute precision
precision <- function(ypred, y){
  tab <- table(ypred, y)
  return((tab[2,2])/(tab[2,1]+tab[2,2]))
}
# function to compute recall
recall <- function(ypred, y){
  tab <- table(ypred, y)
  return(tab[2,2]/(tab[1,2]+tab[2,2]))
}
```

The SVM model had an accuracy of 90% which could be improved by tuning the model using the appropriate range and gamma values.

```
# accuracy measures
accuracy(Prediction, y_test)
```

```
## [1] 0.8998245
```

```
precision(Prediction, y_test)
```

```
## [1] 0.8196532
```

```
recall(Prediction, y_test)
```

```
## [1] 0.5861926
```