ML2-TP2

Team 9 - Erin DiFabio, Chris Farrell, Kris Hooper, Jared Shawver

The Facebook Competition

The competition we decided to pick on Kaggle has to do with Facebook. Before the competition ended, the leaders of this competition would be presented with the opportunity to interview for a software engineer job at Facebook. The goal of the competition is to predict which place a person would like to check into on Facebook. Facebook has a feature where no matter where you are, you can check in on Facebook and it allows your friends to know where you are on that day. Usually this feature is used if a user is either vacationing or going to a popular place. The data given to us was given in 3 excel files, the first being a sample submission, second the test data and lastly the train data. The train data consists of 6 columns whereas the test data consist of 5 columns. The data is a huge dataset consisting of over 104,000 instances. The key features in the data are row ID, x coordinate, y coordinate, accuracy and time. The response variable, found only in the train data, is place ID. This data was artificially created by Facebook consisting of more than 100,000 places. Facebook also ran us no favors by making the data messy to see how we would address these problems. This challenge is an unsupervised machine learning problem.

Reproducibility

Reproducibility is important because it allows methods and results to be verified. Creating a model that is not reproducible is almost pointless because if something is not working properly or leads to complications in the future, how is it possible to figure out what exactly went wrong? Therefore, by checking through the machine learning reproducibility checklist provided, we built a model that is in fact reproducible. Providing clear documentation on how and why we sampled from the original dataset the way we did, showing how we split our train and test sets along with the respective code, and setting a seed, are just a few ways in which we make it possible for our results to be reproduced. Explaining our thought process and justifying our reasoning and any assumptions that we made, allows for another to follow along with a deep understanding of the task at hand and why we chose to model the way we did, including the various techniques that we utilized. It is not enough to simply build a model and find the solution; the methods, techniques, and results of the work must be reproducible in order to be validated.

For our analysis, we chose to use multiple algorithms and compare the outcomes in order to choose our best model. First, we begin by clearing the workspace and loading any necessary pacakages.

We have chosen to set the seed at 6 for reproduceability and we import our train and test dataset. It is important to point out that the train and test datasets have been significantly reduced from the initial 10km by 10km square. In order to even run our models we needed to reduce the dataset down to a 250m by 250m square. All models are utilizing the same data. The Y variable, place_id, is also converted to a factor.

Data Preprocessing

```
original_train <- read.csv(file="train.csv",sep=",",header=T)
original_test <- read.csv(file="test.csv",sep=",",header=T)

# reduce range of data to 1 x 1 kilometer square
original_train.df <- data.frame(original_train)
small_train <- subset.data.frame(original_train.df,original_train.df$x>1)
small_train <- subset.data.frame(small_train,small_train$x<1.5)
small_train <- subset.data.frame(small_train,small_train$y<2.5)
small_train <- subset.data.frame(small_train,small_train$y>2)

original_test.df <- data.frame(original_test)
small_test <- subset.data.frame(original_test.df,original_test.df$x>1)
small_test <- subset.data.frame(small_test,small_test$x<1.5)
small_test <- subset.data.frame(small_test,small_test$y<2.5)
small_test <- subset.data.frame(small_test,small_test$y>2)

# working with time
hours_in_day <- 24
minutes_in_day <- hours_in_day * 60

small_train$hour <- (small_train$time/60) %% hours_in_day
small_train$DayOfWeek <- floor((small_train$time/(minutes_in_day)) %% 7) + 1

small_test$hour <- (small_test$time/60) %% hours_in_day
small_test$DayOfWeek <- floor((small_test$time/(minutes_in_day)) %% 7) + 1

# reorder data
small_train <- small_train[,c(1,2,3,4,5,7,8,6)]

# create validation set
rows <- nrow(small_train)
train.indices <- sample(rows, .8 * rows)
train <- small_train[train.indices,]
val <- small_train[-train.indices,]
test <- small_test

# remove redunant information
drops <- c("row_id","time")
train <- small_train[,!(names(small_train) %in% drops)]
val <- val[,!(names(val) %in% drops)]
test <- small_test[,!(names(test) %in% drops)]

# convert to factors
train$place_id <- as.factor(train$place_id)
val$place_id <- as.factor(val$place_id)
```

```
# export
write.csv(train,"train_small.csv",row.names = F)
write.csv(val,"val_small.csv",row.names = F)
write.csv(test,"test_small.csv",row.names = F)

rm(list=ls())

installIfAbsentAndLoad <- function(neededVector) {
  for(thispackage in neededVector) {
    if( ! require(thispackage, character.only = T) )
    { install.packages(thispackage)}
    require(thispackage, character.only = T)
  }
}

needed <- c('e1071', 'randomForest', 'class')
installIfAbsentAndLoad(needed)

## Loading required package: e1071

## Loading required package: randomForest

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

## Loading required package: class

set.seed(6)

train <- data.frame(read.csv("train_small.csv",header = T))
test <- data.frame(read.csv("test_small.csv",header = T))

train$place_id <- as.factor(train$place_id)
```

Model 1 – Support Vector Machine

SVM requires an incredibly high number of computations to run. Because of this the datasets must be shrunk or else the following error is returned: Error: cannot allocate vector of size xxx GB.

```
rows <- nrow(train)
shrink.indices <- sample(rows, .02 * rows)
train <- train[shrink.indices,]

rows <- nrow(test)
shrink.indices <- sample(rows, .02 * rows)
test <- test[shrink.indices,]

rows <- nrow(train)
```

```r
train.indices <- sample(rows, .8 * rows)
train <- train[train.indices,]
val <- train[-train.indices,]
```

Test SVM – full radial kernel

```r
svm.rad <- svm(place_id~.,data=train,kernel="radial",cost=1,gamma=1,scale=T)

# make predictions
ypred.rad <- predict(svm.rad,val[,1:5])

# determine accuracy
acc.rad <- mean(ifelse(val$place_id==ypred.rad,1,0))
```

A more flexible SVM

```r
svm.rad.flex <- svm(place_id~.,data=train,kernel="radial",cost=1,gamma=5,scal
e=T)

# make predictions
ypred.rad.flex <- predict(svm.rad.flex,val[,1:5])

# determine accuracy
acc.rad.flex <- mean(ifelse(val$place_id==ypred.rad.flex,1,0))
```

An SVM with less training errors

```r
svm.rad.lcost <- svm(place_id~x + y,data=train,kernel="radial",cost=.01,gamma
=5,scale=T)

# make predictions
ypred.rad.lcost <- predict(svm.rad.lcost,val[,1:5])

# determine accuracy
acc.rad.lcost <- mean(ifelse(val$place_id==ypred.rad.lcost,1,0))
```

An SVM with increased training errors

```r
svm.rad.hcost <- svm(place_id~x + y,data=train,kernel="radial",cost=10,gamma=
5,scale=T)

# make predictions
ypred.rad.hcost <- predict(svm.rad.hcost,val[,1:5])

# determine accuracy
acc.rad.hcost <- mean(ifelse(val$place_id==ypred.rad.hcost,1,0))
```

###Best SVM model

```r
results.df <- data.frame("model"=c("Regular Radial",
                                   "More Flexible, Gamma = 5",
                                   "Low Cost, cost = .01",
                                   "High Cost, cost = 10"),
                         "error rate"=c(1-acc.rad,
                                        1-acc.rad.flex,
                                        1-acc.rad.lcost,
                                        1-acc.rad.hcost))

print(results.df)
```

```
##                       model error.rate
## 1            Regular Radial  0.4695652
## 2 More Flexible, Gamma = 5  0.1478261
## 3      Low Cost, cost = .01  0.9608696
## 4     High Cost, cost = 10  0.4304348
```

The best model is the more flexible model with gamma = .5 and error rate of approximately 6%.

Best Model Predictions

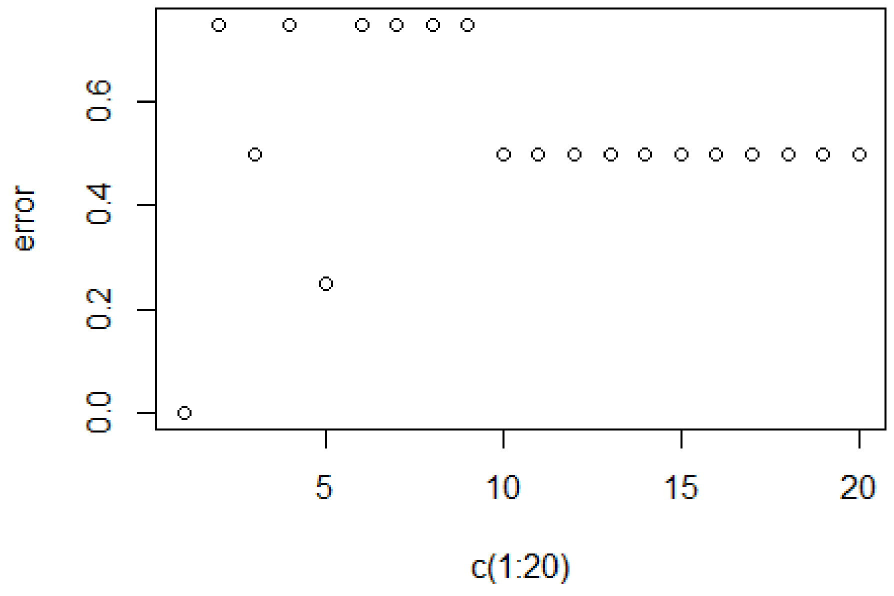
```
ypred.best <- predict(svm.rad.flex,test)
pred.df <- test
pred.df[,6] <- ypred.best

# store the results
names(pred.df)[6] <- "svm_pred_place_id"
```

Model 2 – K-Nearest Neighbors

We begin our KNN model in a similar fashion as the SVM model by resetting the seed to 6 and setting up the same train, test, and validate datasets.

```
set.seed(6)

rows <- nrow(train)
shrink.indices <- sample(rows, .02 * rows)
train <- train[shrink.indices,]

rows <- nrow(train)
train.indices <- sample(rows, .8 * rows)
train <- train[train.indices,]
validate <- train[-train.indices,]

rows <- nrow(test)
shrink.indices <- sample(rows, .02 * rows)
test <- test[shrink.indices,]


train$place_id <- as.factor(train$place_id)
validate$place_id <- as.factor(validate$place_id)

# create train, validate and test set
```

```
train.x <- cbind(train$x, train$y, train$accuracy, train$hour, train$DayOfWee
k)
train.y <- train$place_id
validate.x <- cbind(validate$x, validate$y, validate$accuracy, validate$hour,
validate$DayOfWeek)
validate.y <- validate$place_id
test.x <- cbind(test$x, test$accuracy, test$hour, test$DayOfWeek)

error = c(1:20)
for (k in 1:20) {
  knn.pred <- knn(train.x, validate.x, train.y, k=k)
  error[k] <- mean(knn.pred != validate.y)
}

## Warning in knn(train.x, validate.x, train.y, k = k): k = 19 exceeds number
18 of
## patterns

## Warning in knn(train.x, validate.x, train.y, k = k): k = 20 exceeds number
18 of
## patterns

plot(c(1:20),error)
```



K = 1

```
knn.pred.k1 <- knn(train.x, validate.x, train.y, k=1)
df <- cbind.data.frame(knn.pred.k1,validate.y)
```

```
size <- nrow(df)
hold <- rep(0,size)
for (i in 1:size) {
  if (df[i,1]==df[i,2]) {hold[i]=1} else {hold[i]=0}
} # If the values are the same then store a one else store a zero.
  # The mean of these will be accuracy.

(acc.k1 <- mean(hold))

## [1] 1

(err.k1 <- 1 - acc.k1)

## [1] 0
```

K = 3

```
knn.pred <- knn(train.x, validate.x, train.y, k=3)
df <- cbind.data.frame(knn.pred,validate.y)

size <- nrow(df)
hold <- rep(0,size)
for (i in 1:size) {
  if (df[i,1]==df[i,2]) {hold[i]=1} else {hold[i]=0}
} # If the values are the same then store a one else store a zero.
  # The mean of these will be accuracy.

(acc.k3 <- mean(hold))

## [1] 0.5

(err.k3 <- 1 - acc.k3)

## [1] 0.5
```

K = 5

```
knn.pred <- knn(train.x, validate.x, train.y, k=5)
df <- cbind.data.frame(knn.pred,validate.y)

size <- nrow(df)
hold <- rep(0,size)
for (i in 1:size) {
  if (df[i,1]==df[i,2]) {hold[i]=1} else {hold[i]=0}
} # If the values are the same then store a one else store a zero.
  # The mean of these will be accuracy.

(acc.k5 <- mean(hold))

## [1] 0.5

(err.k5 <- 1 - acc.k5)
```

```
## [1] 0.5
```

K = 10

```
knn.pred <- knn(train.x, validate.x, train.y, k=10)
df <- cbind.data.frame(knn.pred,validate.y)

size <- nrow(df)
hold <- rep(0,size)
for (i in 1:size) {
  if (df[i,1]==df[i,2]) {hold[i]=1} else {hold[i]=0}
} # If the values are the same then store a one else store a zero.
  # The mean of these will be accuracy.

(acc.k10 <- mean(hold))

## [1] 0.25

(err.k10 <- 1 - acc.k10)

## [1] 0.75
```

Best KNN model

```
resultsknn.df <- data.frame("model"=c("K = 1",
                                   "K = 3",
                                   "K = 5",
                                   "K = 10"),
                         "error rate"=c(err.k1,
                                        err.k3,
                                        err.k5,
                                        err.k10))

print(resultsknn.df)

##    model error.rate
## 1  K = 1       0.00
## 2  K = 3       0.50
## 3  K = 5       0.50
## 4 K = 10       0.75

pred.df[,7] <- knn.pred.k1

# store the results
names(pred.df)[7] <- "knn_pred_place_id"
```

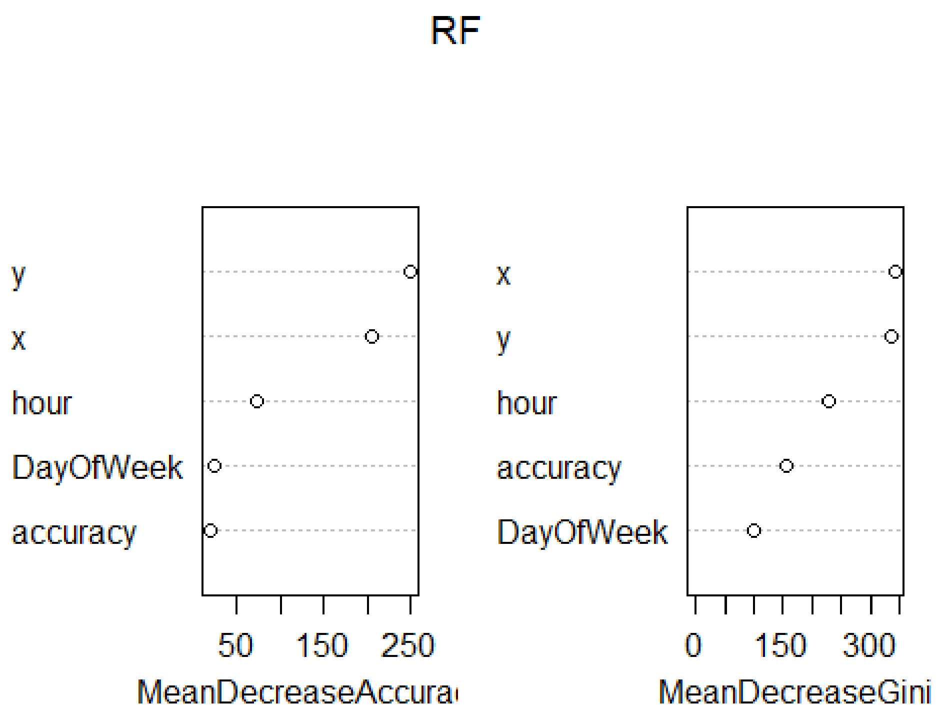
Model 2 – Random Forest

We continue the trend with our Random Forest model by first resetting the random seed to 6 and making sure that the data is in test, train, and validate datasets.

```r
set.seed(6)

train <- data.frame(read.csv("train_small.csv",header = T))
test <- data.frame(read.csv("test_small.csv",header = T))

rows <- nrow(train)
shrink.indices <- sample(rows, .02 * rows)
train <- train[shrink.indices,]

rows <- nrow(test)
shrink.indices <- sample(rows, .02 * rows)
test <- test[shrink.indices,]

rows <- nrow(train)
train.indices <- sample(rows, .8 * rows)
train <- train[train.indices,]
val <- train[-train.indices,]

train$place_id <- as.factor(train$place_id)
```

Random Forest Model

```r
# random forest
RF = randomForest(place_id~., data=train, ntree=500,mtry=4, importance=TRUE,
replace=TRUE)
#RF

# predict on test sed
RF.yhat = predict(RF, newdata=val)

# variable importance
varImpPlot(RF)
```



```
df <- cbind.data.frame(RF.yhat,val$place_id)

size <- nrow(df)
hold <- rep(0,size)
for (i in 1:size) {
  if (df[i,1]==df[i,2]) {hold[i]=1} else {hold[i]=0}
} # If the values are the same then store a one else store a zero.
  # The mean of these will be accuracy.

(acc.rf <- mean(hold))

## [1] 1

(err.rf <- 1 - acc.rf)

## [1] 0
```

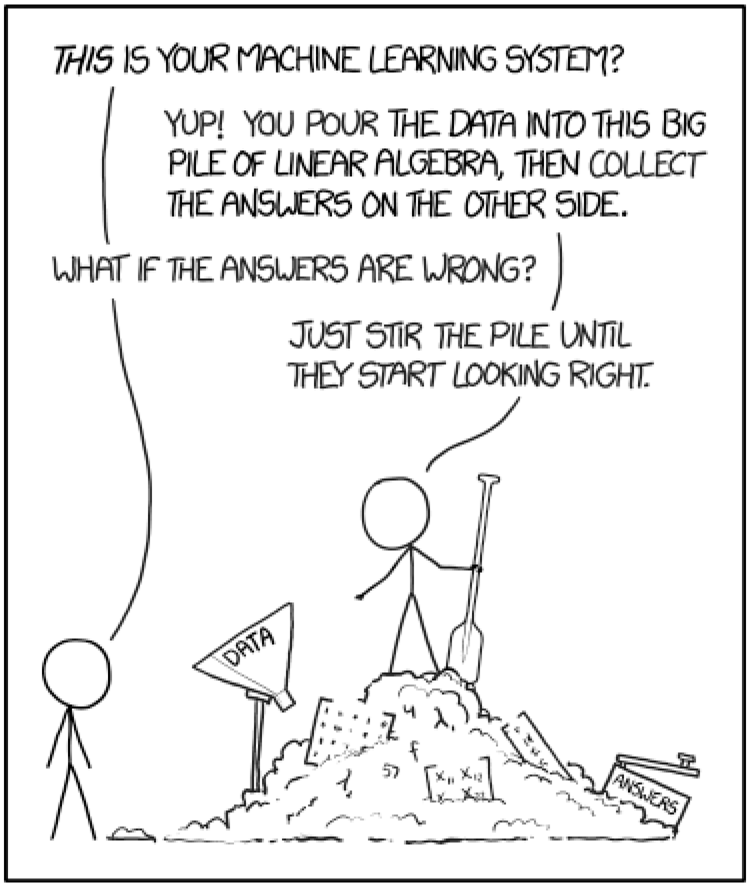
Best RF model

```
resultsrf.df <- data.frame("model"=c("Random Forest"),
                         "error rate"=c(err.rf))

print(resultsrf.df)

##           model error.rate
## 1 Random Forest          0
```

Compilation of our best models compared and executed on the same data

```
RF.yhat = predict(RF, test)

pred.df[,8] <- RF.yhat
# store the results
names(pred.df)[8] <- "rf_pred_place_id"

head(pred.df,10)
```

```
##              x      y accuracy       hour DayOfWeek svm_pred_place_id
## 10191 1.1965 2.1724       35 12.9500000         1        6492221068
## 2908  1.2404 2.4824       77 17.2166667         3        2813387656
## 5630  1.2680 2.1771       33 20.9000000         5        6783074589
## 898   1.4984 2.2694       37 19.8166667         2        6492221068
## 513   1.3096 2.2525       65  2.9000000         4        6492221068
## 21153 1.0285 2.1830       17  0.8166667         3        6492221068
## 13075 1.4525 2.4170       10 20.3500000         3        6492221068
## 12748 1.0026 2.3227      156 10.0833333         7        6492221068
## 11571 1.3642 2.0633       52  1.9333333         3        9451075109
## 9129  1.1007 2.0725      148 19.3833333         3        6492221068
##       knn_pred_place_id rf_pred_place_id
## 10191        4795770188       6304879990
## 2908         1668840545       2813387656
## 5630         2437620236       6783074589
## 898          4795770188       1091437957
## 513          4795770188       1700962451
## 21153        1668840545       6262911124
## 13075        2437620236       8473947908
## 12748        4795770188       7588462064
## 11571        4795770188       9082794995
## 9129         1668840545       9344531911
```


THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS