Homework 1

Pablo Diaz

1/12/2019

Question 2.1

I currently work in analytics at a telephone provider company. Classification is very important because we can offer the best offer to the classified customers based on their phone usage. One interesting model we could make is to predict when a user can move from a non contract plan to a contract plan. Some relevant predictors that we have stored on our databases ARPU (average revenue per user), this is the current amount of money they are paying us, Data MB (the amount of internet data they use), Calls in (the relative amount of time they receive calls), calls out (the relative amount of time they make calls) and Data User (variable to know if a user uses more than 90% of their data plan)

Question 2.2

```
library(kernlab)
# import data, set current directory like this: setwd("~/Documents/R/GeorgiaTech/SVM"
)
data <- read.delim("./credit_card_data-headers.txt")
# make results reproducible
set.seed(42)

# simple linear kernel, C values too small or too big decrease accuracy
model <- ksvm(R1 ~ ., data=data, type="C-svc", kernel="vanilladot", C=50, scaled=TRUE
)
```

```
##  Setting default kernel parameters
```

```
summary(model)
```

```
## Length  Class   Mode
##      1   ksvm     S4
```

```
# calculate a1 to am mutiplying the xmatrix by the coeficient of the linear combinati
on
a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
# calculate a0
a0 <- model@b
# see what the model predicts
pred <- predict(model,data[,1:10])
# see what fraction of the model's predictions match the classification
output <- sum(pred == data[,11]) / nrow(data)
print (output)
```

```
## [1] 0.8639144
```

Test with other Kernel (splinedot)

```
model <- ksvm(R1 ~ ., data=data, type="C-svc", kernel="splinedot", C=200, scaled=TRUE
)
```

```
##  Setting default kernel parameters
```

```
summary(model)
```

```
## Length  Class   Mode
##      1   ksvm     S4
```

```
# calculate a1 to am mutiplying the xmatrix by the coeficient of the linear combinati
on
a <- colSums(model@xmatrix[[1]] * model@coef[[1]])
# calculate a0
a0 <- model@b
# see what the model predicts
pred <- predict(model,data[,1:10])
# see what fraction of the model's predictions match the classification
output <- sum(pred == data[,11]) / nrow(data)
# much better result ? maybe there is overfitting
print (output)
```

```
## [1] 0.9785933
```

Result 2.2

We found out that the best accuracy was `0.8629144` with multipe values of C using a linear kernel, when tested with other kernel it gave a better accuracy but we should consider overfitting.

Question 2.4

```
library(kknn)
#i = 167
#create empty vector
predictions <- c()
for (i in 1:nrow(data)) {

    # remove row i, to avoid finding own nearnest neighbor
    # data[-i,] is all the data except for the ith data point.
    model_kknn=kknn(R1~.,data[-i,],data[i,],k=10, scale = TRUE) # use scaled data

    # prediction
    fit <- model_kknn$fitted.values
    # should make it comparable with R1 column
    predictions <- c(predictions, round(fit)) #save it in array
}
# check accuracy
accuracy = sum(predictions == data[,ncol(data)]) / nrow(data)
print (accuracy)
```

```
## [1] 0.8501529
```

Now convert this to a function to test multiple k values

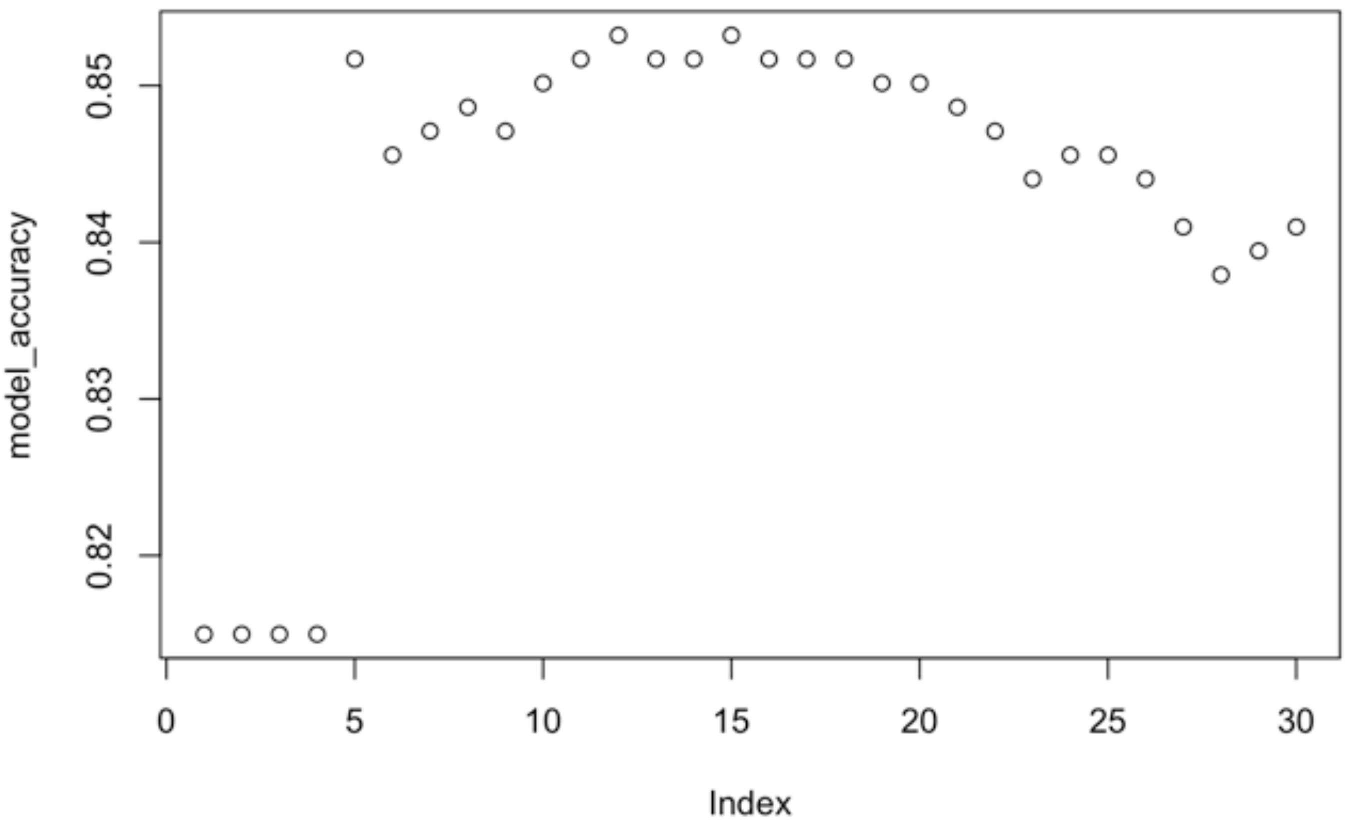
```
findBestK <- function(paramK) {
  predictions <- c()
  for (i in 1:nrow(data)) {

      # remove row i, to avoid finding own nearest neighbor
      # data[-i,] is all the data except for the ith data point.
      # maybe it is a good idea to divide in train an
      model_kknn=kknn(R1~.,data[-i,],data[i,],k=paramK, scale = TRUE) # use scaled da
ta

      # prediction
      fit <- model_kknn$fitted.values
      # should make it comparable with R1 column
      predictions <- c(predictions, round(fit)) #save it in array
  }
  # check accuracy
  accuracy = sum(predictions == data[,ncol(data)]) / nrow(data)
  return (accuracy)
}

model_accuracy <- c()
# test neighbors
# tested with big values but accuracy was around 0.83
# couldn't test with 1000 or beyond
for (i in 1:30) {
  model_accuracy <- c(model_accuracy, findBestK(i))
}
# find best accuracy
max(model_accuracy)
```

```
## [1] 0.853211
```

```
# find best K
which.max(model_accuracy) #returns the position, be careful if range doesn't start wi
th 1
```

```
## [1] 12
```

```
plot(model_accuracy)
```



Result 2.4

So first I tested with some random K’s and found out that values from 1 to 30 where the best, so I chose that range and then used `which.max` to know that the best `K=12`