Project 4

Anton Stråhle & Max Sjödin

28 december 2020

Introduction

This is a project for the course MT7038 which was given during the fall of 2020. In the project we'll examine some basic methods for binary classification such as SVMs, Logistic Regression and LDA.

Data

We picked the `Occupancy Detection` data set as we wanted to work with a binary classification problem as this would allow us to apply most of the methodologies discussed throughout the course. As there are quite a few different binary data sets at UCI we specifically chose our data set as it had a sizable number of instances as well as few, but intuitively explanatory, features. The data set also poses an interesting question with how the features change over time and how this impacts classification.

Attributes

The data `Occupancy Detection` data set includes snapshots of a specific room every minute throughout the course of a few weeks. The aim is to classify the current `Occupancy` of the room using the five features, `Temperature`, `CO2`, `Humidity`, `HumidityRatio` and `Light`, which are observed each minute. The first three features are quite self-explanatory but the two final ones could use some clarification. The `Light` in the room is the light intensity, measured in Lux whilst the `HumidityRatio` is vaguely described as a "derived quantity from temperature and relative humidity, in kgwater-vapor/kg-air".

Exploration

From a quick overview we see that the data set is quite unbalanced.

Table 1: Unoccupied

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
Temperature	1	15810	20.585	0.895	20.500	20.488	0.741	19.000	24.390	5.390	1.325	2.757	0.007
Humidity	2	15810	27.530	5.119	27.150	27.556	5.411	16.745	39.500	22.755	-0.030	-0.760	0.041
Light	3	15810	25.238	81.824	0.000	2.966	0.000	0.000	1546.333	1546.333	4.667	31.278	0.651
CO2	4	15810	604.997	253.027	511.000	545.863	102.299	412.750	2076.500	1663.750	2.442	5.839	2.012
HumidityRatio	5	15810	0.004	0.001	0.004	0.004	0.001	0.003	0.006	0.004	-0.151	-0.767	0.000
Occupancy*	6	15810	1.000	0.000	1.000	1.000	0.000	1.000	1.000	0.000	NaN	NaN	0.000

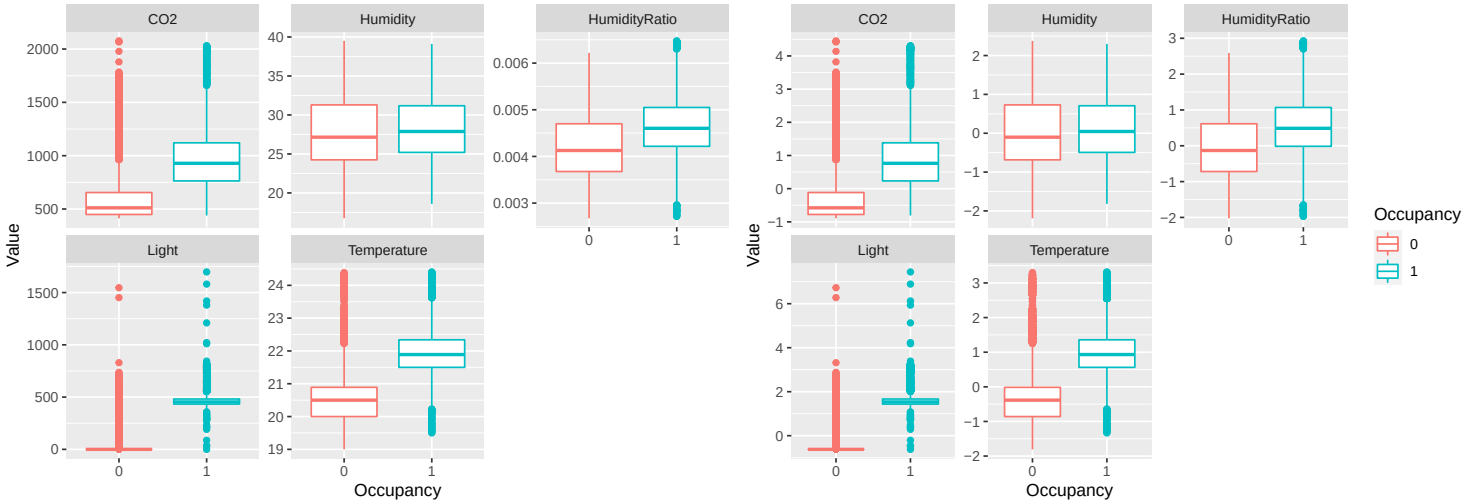
Table 2: Occupied

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
Temperature	1	4750	21.976	0.818	21.890	21.938	0.619	19.500	24.408	4.908	0.414	0.254	0.012
Humidity	2	4750	28.076	4.472	27.882	28.040	4.305	18.600	39.118	20.517	0.156	-0.323	0.065
Light	3	4750	481.967	94.704	454.000	461.597	31.135	0.000	1697.250	1697.250	2.953	17.353	1.374
CO2	4	4750	975.322	317.261	928.583	943.895	267.486	439.000	2028.500	1589.500	0.997	1.154	4.603
HumidityRatio	5	4750	0.005	0.001	0.005	0.005	0.001	0.003	0.006	0.004	-0.059	-0.062	0.000
Occupancy*	6	4750	2.000	0.000	2.000	2.000	0.000	2.000	2.000	0.000	NaN	NaN	0.000

We have about four times more unoccupied than occupied minutes. As the data is observed around the clock it is of course natural that the room is unoccupied during a majority of the day. Due to this quite severe imbalance we initially wanted to make sure that our training, validation and testing sets reflected this inherent property of the data. However, as the data consists of minutley snapshots we have to make sure that each set consists of different time periods. The reason as to why we have to split the date into periods is that subsequent snapshots will very likely have the same feature values as well as occupancy status which would lead to a e.g. a 1-NN predicting with very higher accuracy accuracy (as almost identical data points would be present in both the training, vadliation and testing data), which is not really what we want. Due to the aforementioned issue with identical data points we will not be using cross validation and instead resort to using a specific validation set to evaluate any parameters.

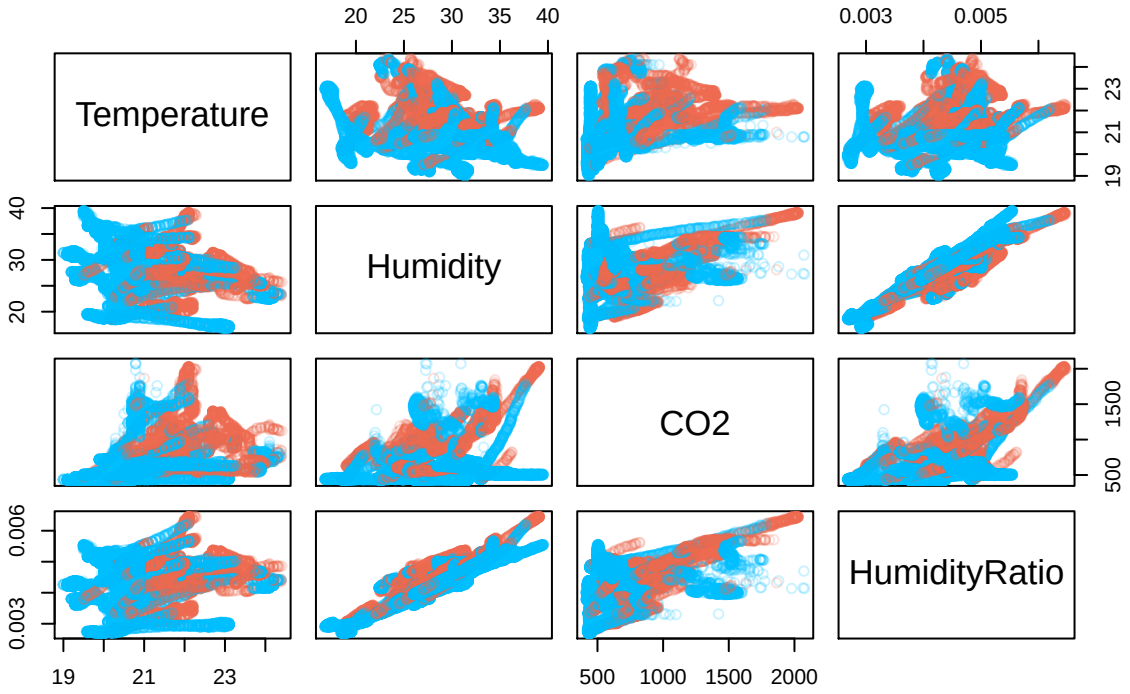
The class imbalance turned out to be a real issues since some classifiers simply classified the room as unoccupied at all times. As such we resorted to upsample both our training and our validation set in order to allow for the classification of both classes (SOURCE ON UPSAMPLING). In the end this proved to be a major success as we not only classified both classes but also improved the general performance of our models drastically.

We also chose to standardize our features to allow for better performance of scale dependent classifiers. As some of the features include some very major deviations as can be seen in the figure below the choice to not standardize would surely impact the performance of these classifiers negatively.



As can be seen in both the figure and the tables above the feature `Light` seems to to quite well in describing the current occupancy of the room. This is quite evident as people rarely gather in a room with the lights turned off, and (hopefully) turn the lights off when they leave. This features solely dominates the others when it comes to classification and is, at least according to us, quite boring. As such we'll choose to excluded it in order to actually be able to perform a somewhat comprehensive analysis that doesn't just include the questions *Was the lights on or off?*.

Figure 2: Correlation between features



As we can see in the figure above the remaining features seem to behave quite nicely, although some could be considered to be somewhat correlated (e.g. `Humidity` and `CO2`).

Methodology

As we're dealing with a binary classification problem there are some different methods that we have decided to test out. We initially tested a variety of classifiers but quickly came to the conclussion that most generalized very poorly in validation and testing scenarios. As such decided to foucs on more stable classifiers that would generalize well and perhaps ignore any slight differences between the different data sets (as previously mentioned the sets are a collection of snapshots from entierly different days so smaller differences are not unlikely). Furthermore we also performed decided to change focus as we assumed that the data could only support a simple decision boundary as the cutoffs (i.e. the minutes after the occupancy status changes) should be very difficult to classify. The feature values take time to change (i.e. the `CO2` does not go to zero the minute that the room becomes unoccupied and neither does it jump to normal levels (for an occupied room) exactly when people enter).

Due to the aforementioned reasons we have resorted to some very simple, but stable, classifiers. Those specifically being SVMs, Logistic Regression and ...

SVM

As support vector machines are good choices for classification it seems appropriate to apply the method to this problem. With a low cost C we should be able to generalize well and thus achieve adequate testing accuracy. It has also been shown in (*Bousquet, Elisseef 2002*) that low cost SVMs are in fact stable classifiers.

Implementation

Using the package `e1071` and the function `svm` we implement a linear, polynomial and a radial SVM and use a coarse-to-fine search for a good value of C using our aforementioned validation set.

We seem to have similar performances for all different kernels as can be seen in the table below.

Table 3: SVM Test Accuracies

Kernel	Cost	TestAccuracy
Linear	0.00013	0.83940

Kernel	Cost	TestAccuracy
Radial	0.00001	0.82702
Polynomial Degree 4	0.00004	0.82964

We also note that the optimal values of C, based on the performance on the validation set, turned out to be very low for all three kernels. This indicates, as we initially thought, that more stable classifiers would generalize well.

We can also examine the confusion matrices for the three models.

Table 4: Linear

	0	1
0	1409	144
1	284	828

Table 5: Radial

	0	1
0	1363	131
1	330	841

Table 6: Polynomial Degree 4

	0	1
0	1375	136
1	318	836

In all three models we seem to have proportionally fewer false negatives than we do false positives. This might be due to how the feature values change at the cutoff points (i.e. when the occupancy status changes). This would perhaps indicate that the change is more rapid when going from occupied to unoccupied than the other way around.

In all it seems that low cost SMVs, regardless of kernel, are a good tool when stability is of the utmost importance (at least when it comes to our data set).

Bibliography

Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction.* 2nd ed. New York: Springer.

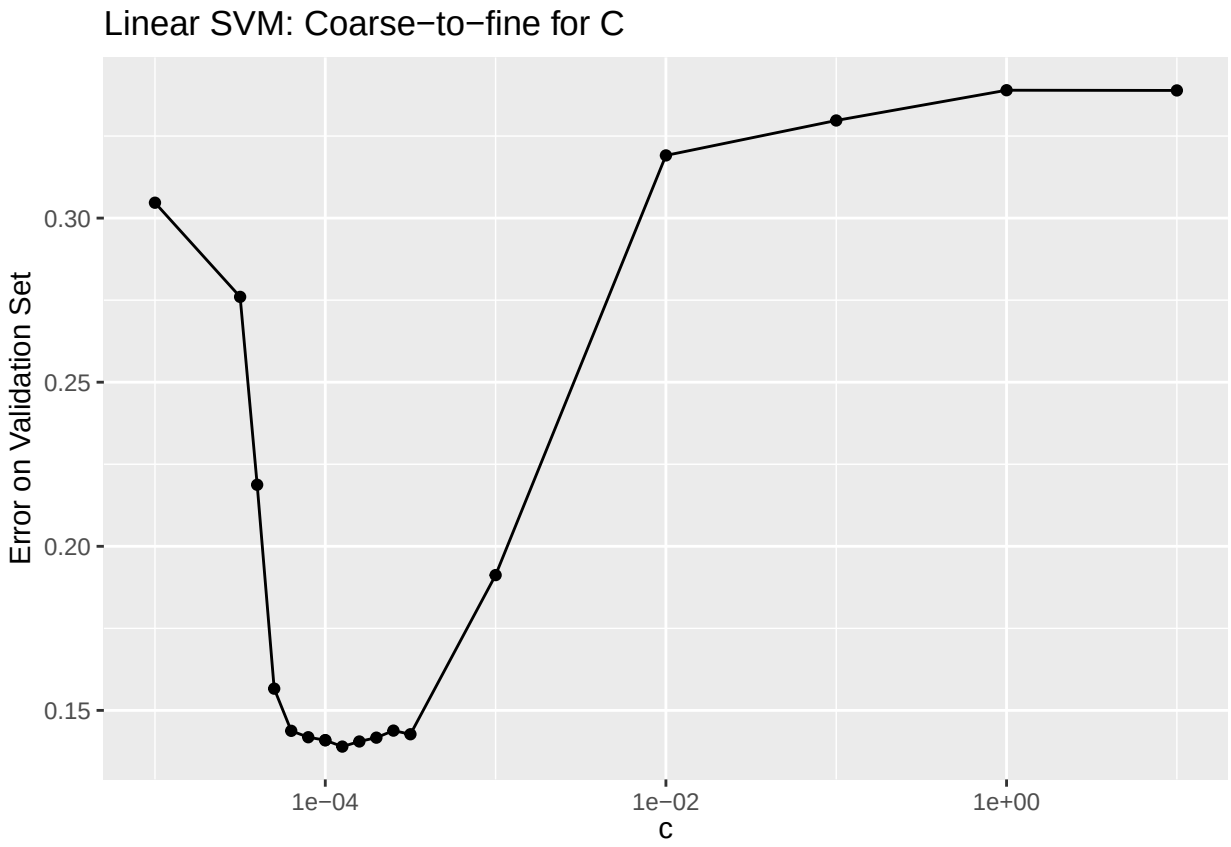
O. Bousquet and A. Elisseeff. *Stability and generalization.* J. Mach. Learn. Res., 2:499–526, 2002.

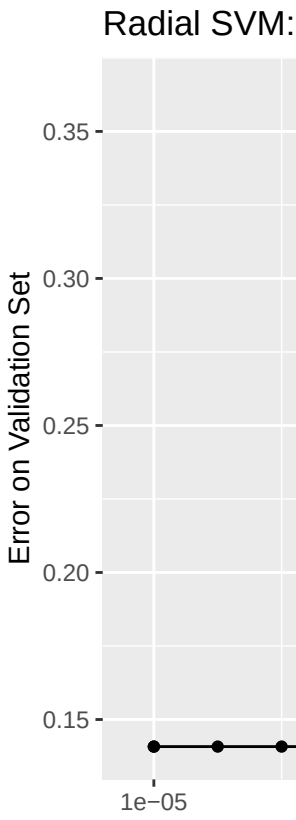
Paul, M. (2017), *Nonlinear Classification* [Presentation]. \ https://cmci.colorado.edu/classes/INFO-4604/fa17/files/slides-9_nonlinear.pdf

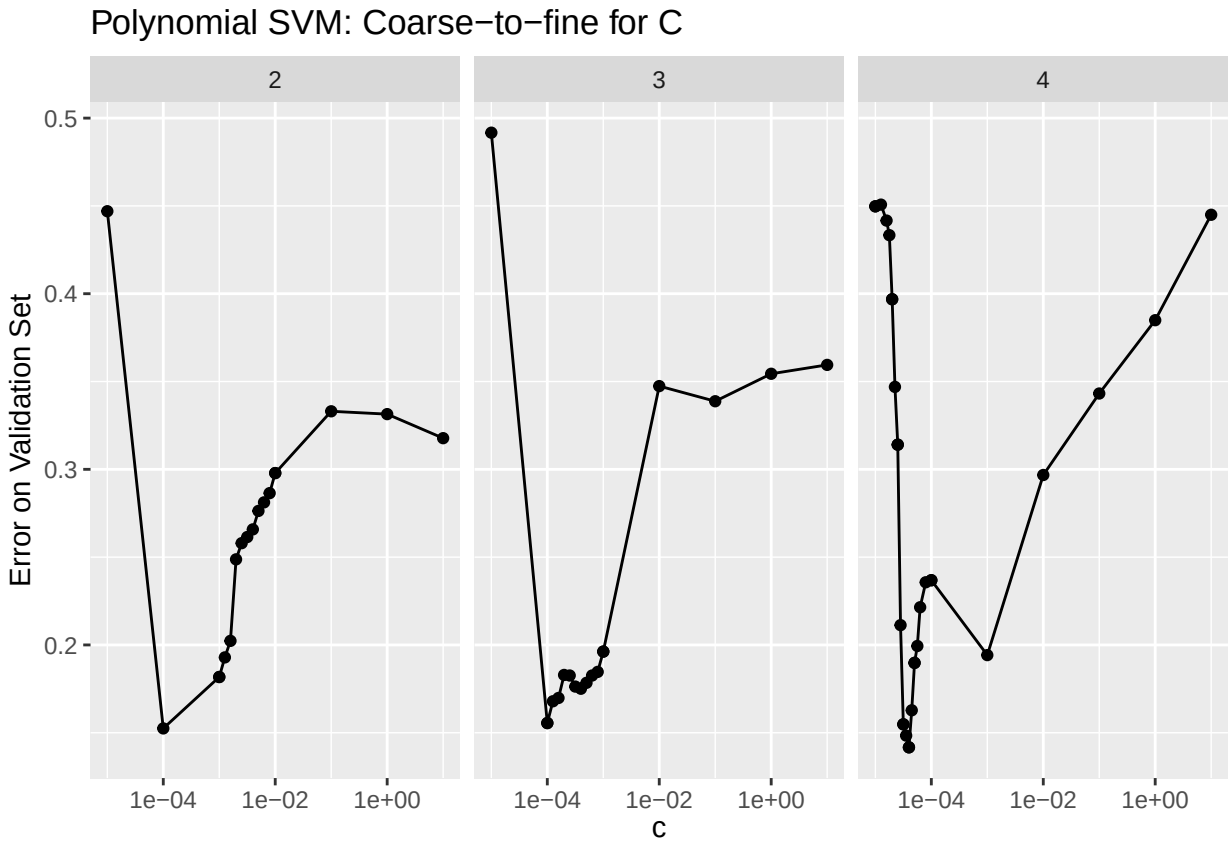
Accurate occupancy detection of an office room from light, temperature, humidity and CO2 measurements using statistical learning models. Luis M. Candanedo, VÃ©ronique Feldheim. Energy and Buildings. Volume 112, 15 January 2016, Pages 28-39

Appendix

Figures







Code

```
library(tidyverse)
library(e1071)
library(ggpubr)
library(readxl)
library(scales)
library(knitr)
library(psych)
library(kknn)
library(grid)
library(gridExtra)
library(caret)

d1 <- read.delim("../data/datatraining.txt", sep = ",")
d2 <- read.delim("../data/datatest.txt", sep = ",")
d3 <- read.delim("../data/datatest2.txt", sep = ",")

occupancyData <- d1 %>%
  rbind(d2) %>%
  rbind(d3) %>%
  select(-date) %>%
  mutate(Occupancy = factor(Occupancy))
sStats <- describeBy(occupancyData, occupancyData$Occupancy)

sStats$`0` %>%
  knitr::kable(caption = "Unoccupied", digits = 3)

sStats$`1` %>%
  knitr::kable(caption = "Occupied", digits = 3)


occupancyData %>%
  gather(key = "Variable", value = "Value", -Occupancy) %>%
  ggplot(aes(x = Occupancy, y = Value, color = Occupancy)) +
  geom_boxplot() +
  facet_wrap(~Variable, scales = "free_y") +
  theme(legend.position = "none") -> p1

as.data.frame(occupancyData) %>%
  mutate_if(is.numeric, scale) %>%
  gather(key = "Variable", value = "Value", -Occupancy) %>%
  ggplot(aes(x = Occupancy, y = Value, color = Occupancy)) +
  geom_boxplot() +
  facet_wrap(~Variable, scales = "free_y") -> p2

grid.arrange(p1, p2, ncol = 2)

pData <- occupancyData %>%
  select(-Light)

cols <- character(nrow(pData))

cols[as.numeric(as.character(pData$Occupancy)) == 0] <- "deepskyblue1"
cols[as.numeric(as.character(pData$Occupancy)) == 1] <- "coral2"

pairs(pData[,-5], col = scales::alpha(cols, 0.2), main = "Figure 2: Correlation between features")

source("svm.R")

#Linear
```

```r
lcDF1 <- valLinearSVM(10^seq(-5, 1, by = 1))
lcDF2 <- valLinearSVM(10^seq(-4.5, -3.5, by = 0.1))

lcDF <- lcDF1 %>%
  rbind(lcDF2)

bestLC <- lcDF %>%
  arrange(error) %>%
  select(c) %>%
  slice(1) %>%
  pull()

ml <- svm(Occupancy ~ ., data = trainingData, kernel = "linear", cost = bestLC)

predL <- predict(ml, testingData)

testErrorLSVM <- mean(predL != testingData$Occupancy)

#Radial

rcDF1 <- valRadialSVM(10^seq(-5, -1, by = 1))
rcDF2 <- valRadialSVM(10^seq(-5, -4, by = 0.25)) #many give us the same validation error, i.e. as long as it is

rcDF <- rcDF1 %>%
  rbind(rcDF2)

bestRC <- rcDF %>%
  arrange(error) %>%
  select(c) %>%
  slice(1) %>%
  pull()

mr <- svm(Occupancy ~ ., data = trainingData, kernel = "radial", cost = bestRC)

predR <- predict(mr, testingData)

testErrorRSVM <- mean(predR != testingData$Occupancy)

#Polynomial

pcDF1 <- valPolynomialSVM(10^seq(-5, 1, by = 1), 2:4, 1)
pcDF2 <- valPolynomialSVM(10^seq(-3, -2, by = 0.1), 2, 1)
pcDF3 <- valPolynomialSVM(10^seq(-4, -3, by = 0.1), 3, 1)
pcDF4 <- valPolynomialSVM(10^seq(-5, -4, by = 0.1), 4, 1)
pcDF5 <- valPolynomialSVM(10^seq(-4.75, -4.25, by = 0.05), 4, 1)

pcDF <- pcDF1 %>%
  rbind(pcDF2) %>%
  rbind(pcDF3) %>%
  rbind(pcDF4) %>%
  rbind(pcDF5)

bestP <- pcDF %>%
  arrange(error) %>%
  select(d, c) %>%
  slice(1)

mp <- svm(Occupancy ~ ., data = trainingData, kernel = "polynomial", cost = bestP$c, degree = bestP$d, coef0 = 1

predP <- predict(mp, testingData)
```

```r
testErrorPSVM <- mean(predP != testingData$Occupancy)


data.frame(Kernel = c("Linear", "Radial", paste("Polynomial Degree", bestP$d)),
           Cost = c(bestLC, bestRC, bestP$c),
           TestAccuracy = c(1-testErrorLSVM, 1-testErrorRSVM, 1-testErrorPSVM)) %>%
  kable(caption = "SVM Test Accuracies", digits = 5)


lCM <- confusionMatrix(predL, testingData$Occupancy)

rCM <- confusionMatrix(predR, testingData$Occupancy)

pCM <- confusionMatrix(predP, testingData$Occupancy)

kable(lCM$table, caption = "Linear") #%>%kable_styling(full_width = FALSE, position = "float_left")

kable(rCM$table, caption = "Radial") #%>%kable_styling(full_width = FALSE, position = "float_left")

kable(pCM$table, caption = paste("Polynomial Degree", bestP$d)) #%>%kable_styling(full_width = FALSE, position =


lcDF %>%
  ggplot(aes(x = c, y = error)) +
  geom_line() +
  geom_point() +
  scale_x_log10() +
  ylab("Error on Validation Set") +
  xlab(expression(c)) +
  ggtitle("Linear SVM: Coarse-to-fine for C")

rcDF %>%
  ggplot(aes(x = c, y = error)) +
  geom_line() +
  geom_point() +
  scale_x_log10() +
  ylab("Error on Validation Set") +
  xlab(expression(c)) +
  ggtitle("Radial SVM: Coarse-to-fine for C")

pcDF %>%
  ggplot(aes(x = c, y = error)) +
  geom_line() +
  geom_point() +
  scale_x_log10() +
  ylab("Error on Validation Set") +
  xlab(expression(c)) +
  ggtitle("Polynomial SVM: Coarse-to-fine for C") +
  facet_wrap(~d)

#svm.R
source("data.R")



valLinearSVM <- function(sequence, train = trainingData, val = validationData){

  bestCost <- 0
  bestValError <- 1
```

```
  for(C in sequence){

    m <- svm(Occupancy ~ ., data = train, kernel = "linear", cost = C)
    pred <- predict(m, val)

    valError <- mean(pred != val$Occupancy)

    #print(paste("Training Error:", round(mean(m$fitted != train$Occupancy), 4), "for C:", C))
    print(paste("Validation Error:", round(valError, 4), "for C:", C))

    if(valError < bestValError){

      bestValError <- valError
      bestCost <- C

    }

  }

  bestCost

}

valRadialSVM <- function(sequence, train = trainingData, val = validationData){

  bestCost <- 0
  bestValError <- 1

  for(C in sequence){

    m <- svm(Occupancy ~ ., data = train, kernel = "radial", cost = C)
    pred <- predict(m, val)

    valError <- mean(pred != val$Occupancy)

    #print(paste("Training Error:", round(mean(m$fitted != train$Occupancy), 4), "for C:", C))
    print(paste("Validation Error:", round(valError, 4), "for C:", C))

    if(valError < bestValError){

      bestValError <- valError
      bestCost <- C

    }

  }

  bestCost

}

valPolynomialSVM <- function(C, deg, coeff, train = trainingData, val = validationData){

  bestCost <- 0
  bestDeg <- 0
  bestValError <- 1

  for(d in deg){

    for(c in C){
```

```
      m <- svm(Occupancy ~ ., data = train, kernel = "polynomial", cost = c, degree = d, coef0 = coeff)
      pred <- predict(m, val)

      valError <- mean(pred != val$Occupancy)

      #print(paste("Training Error:", round(mean(m$fitted != train$Occupancy), 4), "for C:", c))
      print(paste("Validation Error:", round(valError, 4), "for C:", c, "and D:", d))

      if(valError < bestValError){

        bestValError <- valError
        bestCost <- c
        bestDeg <- d

      }

    }

  }

  list("C" = bestCost, "D" = bestDeg)

}
#knn.R
source("data.R")
library(class)

valKNN <- function(K, train = trainingData, val = validationData){

  bestK <- 0
  bestError <- 1

  for(k in K){

    m <- knn(train[, !sapply(train, is.factor)], val[, !sapply(val, is.factor)], cl = train[,sapply(train, is.fa

    valError <- mean(val[, sapply(val, is.factor)] != m)

    print(paste("Validation Error:", round(valError, digits = 3), "for k =", k))

    if(valError < bestError){

      bestK <- k
      bestError <- valError

    }

  }

  bestK

}

#wknn.R
source("data.r")
library(kknn)

valWKNN <- function(K, kernel = "gaussian", train = trainingData, val = validationData){

  bestK <- 0
  bestError <- 1
```

```
  for(k in K){

    m <- kknn(Occupancy ~ ., train = train, test = val, kernel = kernel, k = k)

    valError <- mean(val[, sapply(val, is.factor)] != m$fitted.values)

    print(paste("Validation Error:", round(valError, digits = 3), "for k =", k))

    if(valError < bestError){

      bestK <- k
      bestError <- valError

    }

  }

  bestK

}

#data.R
library(tidyverse)
library(e1071)
library(ggpubr)
library(readxl)
library(scales)
library(rpart)
library(rpart.plot)

#occupancy Dataset

d1 <- read.delim("../data/datatraining.txt", sep = ",")
d2 <- read.delim("../data/datatest.txt", sep = ",")
d3 <- read.delim("../data/datatest2.txt", sep = ",")

occupancyData <- d1 %>%
  rbind(d2) %>%
  rbind(d3) %>%
  select(-date) %>%
  mutate(Occupancy = factor(Occupancy)) %>%
  select(-Light)

#From CrossValidate package (issues with dependencies in the install so I yoinked their source code)
balancedSplit <- function(fac, size){
  trainer <- rep(FALSE, length(fac))
  for(lev in levels(fac)){
    N <- sum(fac==lev)
    wanted <- max(1, trunc(N*size))
    trainer[fac==lev][sample(N, wanted)] <- TRUE
  }
  trainer
}

train <- balancedSplit(occupancyData$Occupancy, size = 0.6)

rawTrainingData <- occupancyData[train,]
remainingData <- occupancyData[!train,]

validation <- balancedSplit(remainingData$Occupancy, 0.5)
```

```r
rawValidationData <- remainingData[validation,]
rawTestingData <- remainingData[!validation,]

standardizeData <- function(data, rawTrain = rawTrainingData){

  attr <- rawTrain[,!sapply(rawTrain, is.factor)]

  mean <- apply(attr, 2, mean)
  sd <- apply(attr, 2, sd)

  data.frame(Occupancy = data[,sapply(rawTrain, is.factor)]) %>%
    cbind(t((t(data[,!sapply(rawTrain, is.factor)]) - mean)/sd))

}

#Standardized separately, should be the same mean and sd for all data
trainingData <- standardizeData(rawTrainingData)
validationData <- standardizeData(rawValidationData)
testingData <- standardizeData(rawTestingData)

#Looks very nice id say

trainingData %>%
  gather(key = "Variable", value = "Value", -Occupancy) %>%
  ggplot(aes(x = Occupancy, y = Value, color = Occupancy)) +
  geom_boxplot() +
  facet_wrap(~Variable, scales = "free_y")
```

```r
#svm.R
source("data.R")



valLinearSVM <- function(sequence, train = trainingData, val = validationData){

  bestCost <- 0
  bestValError <- 1

  for(C in sequence){

    m <- svm(Occupancy ~ ., data = train, kernel = "linear", cost = C)
    pred <- predict(m, val)

    valError <- mean(pred != val$Occupancy)

    #print(paste("Training Error:", round(mean(m$fitted != train$Occupancy), 4), "for C:", C))
    print(paste("Validation Error:", round(valError, 4), "for C:", C))

    if(valError < bestValError){

      bestValError <- valError
      bestCost <- C

    }

  }

  bestCost

}
```

```r
valRadialSVM <- function(sequence, train = trainingData, val = validationData){

  bestCost <- 0
  bestValError <- 1

  for(C in sequence){

    m <- svm(Occupancy ~ ., data = train, kernel = "radial", cost = C)
    pred <- predict(m, val)

    valError <- mean(pred != val$Occupancy)

    #print(paste("Training Error:", round(mean(m$fitted != train$Occupancy), 4), "for C:", C))
    print(paste("Validation Error:", round(valError, 4), "for C:", C))

    if(valError < bestValError){

      bestValError <- valError
      bestCost <- C

    }

  }

  bestCost

}

valPolynomialSVM <- function(C, deg, coeff, train = trainingData, val = validationData){

  bestCost <- 0
  bestDeg <- 0
  bestValError <- 1

  for(d in deg){

    for(c in C){

      m <- svm(Occupancy ~ ., data = train, kernel = "polynomial", cost = c, degree = d, coef0 = coeff)
      pred <- predict(m, val)

      valError <- mean(pred != val$Occupancy)

      #print(paste("Training Error:", round(mean(m$fitted != train$Occupancy), 4), "for C:", c))
      print(paste("Validation Error:", round(valError, 4), "for C:", c, "and D:", d))

      if(valError < bestValError){

        bestValError <- valError
        bestCost <- c
        bestDeg <- d

      }

    }

  }

  list("C" = bestCost, "D" = bestDeg)
```

```
}
```

```
#knn.R
source("data.R")
library(class)

valKNN <- function(K, train = trainingData, val = validationData){

  bestK <- 0
  bestError <- 1

  for(k in K){

    m <- knn(train[, !sapply(train, is.factor)], val[, !sapply(val, is.factor)], cl = train[,sapply(train, is.fa

    valError <- mean(val[, sapply(val, is.factor)] != m)

    print(paste("Validation Error:", round(valError, digits = 3), "for k =", k))

    if(valError < bestError){

      bestK <- k
      bestError <- valError

    }

  }

  bestK

}
```

```
#wknn.R
source("data.r")
library(kknn)

valWKNN <- function(K, kernel = "gaussian", train = trainingData, val = validationData){

  bestK <- 0
  bestError <- 1

  for(k in K){

    m <- kknn(Occupancy ~ ., train = train, test = val, kernel = kernel, k = k)

    valError <- mean(val[, sapply(val, is.factor)] != m$fitted.values)

    print(paste("Validation Error:", round(valError, digits = 3), "for k =", k))

    if(valError < bestError){

      bestK <- k
      bestError <- valError

    }

  }

  bestK

}
```

```r
#data.R
library(tidyverse)
library(e1071)
library(ggpubr)
library(readxl)
library(scales)
library(rpart)
library(rpart.plot)

#occupancy Dataset

d1 <- read.delim("../data/datatraining.txt", sep = ",")
d2 <- read.delim("../data/datatest.txt", sep = ",")
d3 <- read.delim("../data/datatest2.txt", sep = ",")

occupancyData <- d1 %>%
  rbind(d2) %>%
  rbind(d3) %>%
  select(-date) %>%
  mutate(Occupancy = factor(Occupancy)) %>%
  select(-Light)

#From CrossValidate package (issues with dependencies in the install so I yoinked their source code)
balancedSplit <- function(fac, size){
  trainer <- rep(FALSE, length(fac))
  for(lev in levels(fac)){
    N <- sum(fac==lev)
    wanted <- max(1, trunc(N*size))
    trainer[fac==lev][sample(N, wanted)] <- TRUE
  }
  trainer
}

train <- balancedSplit(occupancyData$Occupancy, size = 0.6)

rawTrainingData <- occupancyData[train,]
remainingData <- occupancyData[!train,]

validation <- balancedSplit(remainingData$Occupancy, 0.5)

rawValidationData <- remainingData[validation,]
rawTestingData <- remainingData[!validation,]

standardizeData <- function(data, rawTrain = rawTrainingData){

  attr <- rawTrain[,!sapply(rawTrain, is.factor)]

  mean <- apply(attr, 2, mean)
  sd <- apply(attr, 2, sd)

  data.frame(Occupancy = data[,sapply(rawTrain, is.factor)]) %>%
    cbind(t((t(data[,!sapply(rawTrain, is.factor)]) - mean)/sd))

}

#Standardized separately, should be the same mean and sd for all data
trainingData <- standardizeData(rawTrainingData)
validationData <- standardizeData(rawValidationData)
testingData <- standardizeData(rawTestingData)

#Looks very nice id say
```

```
trainingData %>%
  gather(key = "Variable", value = "Value", -Occupancy) %>%
  ggplot(aes(x = Occupancy, y = Value, color = Occupancy)) +
  geom_boxplot() +
  facet_wrap(~Variable, scales = "free_y")
```