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 on a granular level and then proceed with a more in depth analysis and discussion for those that seem to perform best. The methods that we will initially test are SVM, KNN and Decision Trees.

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 sizeable number of instances as well as few, but intuitivley explanatory, features.

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

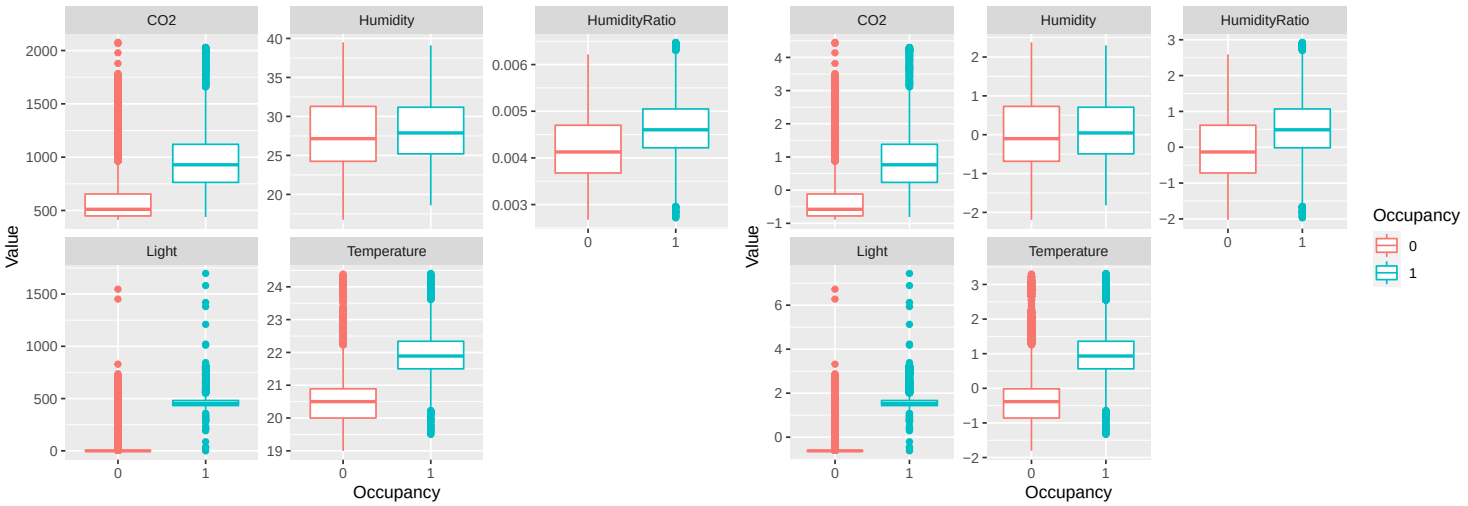
Table 2: Occupied

	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 inbalance we have to make sure that our training, validation and testing sets reflect this inherent property of the data. As such we decided to concatenate the three provided data sets (training, validation and testing) and split these up into balanced sets ourselves as the data providers seems to have just split the complete date by the timestamp which leads to a quite severe inbalance as a weekend is included (i.e. no Occupied observations for two days).

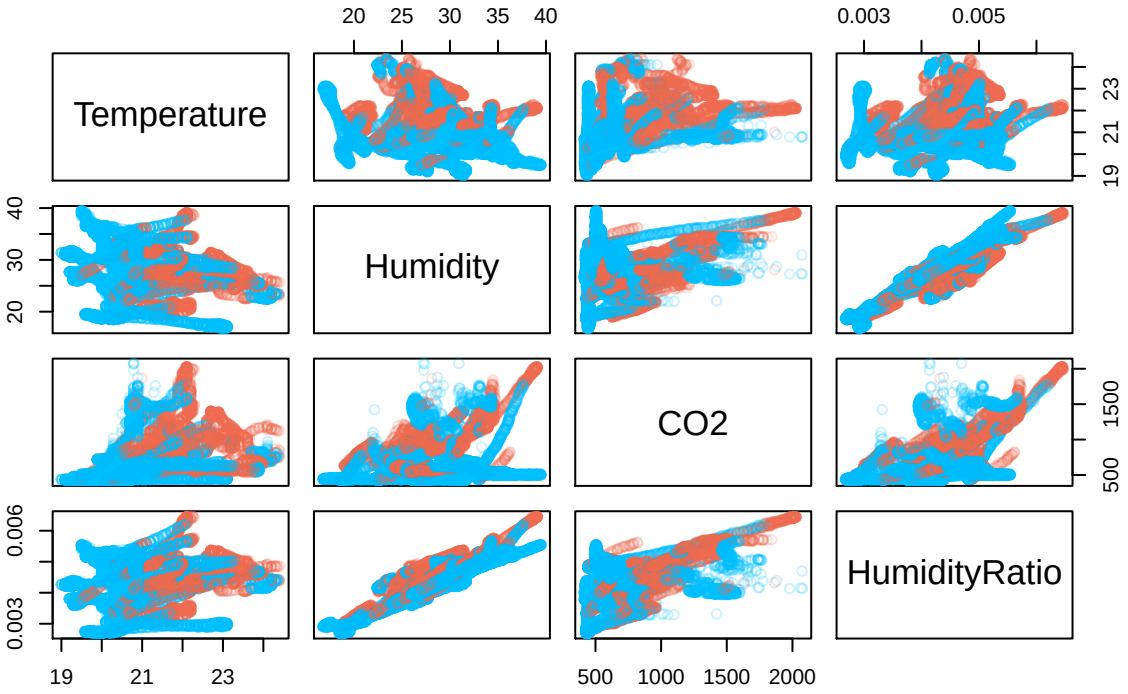
Beyond the poor splitting of the data into training, validation and testing we found not obvious inconsitensies in the data that needed addressing.

We also chose to standardize to allow for better performance of scale dependent classifiers, e.g. kNN or SVM. As some of the features include some very major deviations as can be seen in the figure below this choice to not standardize would surely impact the performance of these classifiers negativley.



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`). There seem to be some quite nice divisions of the occupancy status within the different features which should make adequate classification quite easy.

Methodology

As we're dealing with a binary classification problem there are some different methods that we've decided to test out. The methods which we have chosen are SVM, KNN and Decision Trees. Our idea is to examine all methods quite shallowly and then go a bit deeper for one or two that shows promise.

SVM

As support vector machines are widely used in classification (SOURCE) it seems appropriate to apply the method to this problem. In Figure 2 above we note that we seem to have non-linear data which suggest that either a polynomial or a radial kernel should work well, or at least better than a linear kernel.

Implementation

We examine three different kernels, a linear kernel, a radial kernel as well as a polynomial kernel. We use our aforementioned validation set in order to perform a coarse-to-fine parameter search for the cost C and the degree for the polynomial kernel (in the case of the polynomial kernel we use a grid coarse grid search and then a finer search for the best degree). Using the package `e1071` and the function `svm` we attained the following results for the three scenarios.

For the linear kernel we only resort to a coarse parameter search as it's performance is not going to be able to match that of the polynomial or the radial kernel (and since the validation error is the same it did not change much over the costs examined ($\{10^i\}_{i=-2}^{3}$).

For the polynomial kernel we encountered some computational difficulties (the optimizer did not converge within the iteration limit of the `e1071`) when tuning our paremeters. As such we were only able to efficiently test up until degree 5. We did note an increase in the validation accuracy when increasing both the degree and the cost. This seems to indicate that the data lacks a lot of noise as we seem to favor a low bias, but high variance, classifier which we obtain by using such a high cost.

For the radial kernel we had similar findings as we did for the polynomial kernel. We achieved higher validation accuracies by increasing the cost (this increase did however slow down eventually) which prompted us to once again disregard the finer part of the parameter search in order to simply examine higher costs.

Table 3: SVM Test Accuracies

Kernel	Cost	TestAccuracy
Linear	10.000	0.837
Radial	31622.777	0.915
Polynomial degree 5	316.228	0.922

KNN

As noted previously the data seems to be quite non-linear and as KNN in a good non-linear classifier (SOURCE). As noted when examining the SVMs in the previous section we seem to favor low bias classifiers due to an inherent lack of noise in our data set which should inidcate that we want a smaller k. Furthermore as KNN scales well with data we should hopefully be able to achieve quite good results given the size of our training set (approximatley 12000 observations).

Implementation

Using the package `Class` and the function `knn` we do a parameter search using our validation set for a good k on $(1, ..., 50)$.

The value of k which generates the best validation accuracy turns out to be 1. As the best kNN classifier, which performs extremely well with a test accuracy of 0.9837062 is a 1-NN, this strongly indicates that our data is not very noisy at all (which we also noted when examining the SVMs).

A possible way to enhance the nearest neighbour model is through weighting (SOURCE BOOK). A simple weighting scheme would be to weight the k nearest neighbours by their distance to the new point we want to classify, as such we give further emphesis to neightbours closer to the point which we wish to classify.

In order to implement this weighting scheme we use the `kknn` package and the included function with the same name which uses kernel-difference weighting. The package allows for the usage of many different kernels but the results that they produce do not differ enough to justify an inclusion of all of them. As such we use the so called *Epanechnikov* kernel as it is one which we've encountered before.

When using weighting we actually use 5 neighbours instead of only one, this also leads to a much higher accuracy as well at 0.9844358.

Decions Trees

Using decision trees is a good idea since we have non-linear data and a simple binary classification problem.

Implementation

Constructing a simple decision tree resulted in a test accuracy of 0.9202335. This is can be further optimized by implementing bagging and boosting algorithms.

Bagging procedures draw a predetermined number of bootstrap samples each fitting a model. The models outputted predictions are averaged and gives the resulting bagged estimate for each observation.

Boosting procedures train the classifier by weighting each observation based on its classification, placing more weight on observations incorrectly classified. The next iteration of the procedure focuses more on those previously misclassified observations to better classify the training data. Using the package `ipred` for bagging and the package `adabag` for boosting we attained the following results.

Table 4: Decision Trees

Method	Test.Accuracy
Simple	0.920
Bagging	0.983
Boosting	0.966

In the case of bagging we draw 1000 bootstrap samples and in the case of boosting we iterate over 100 trees. For boosting, iteration over 1000 trees led to long computational time and did not change the result significantly. From Table 4 we can conclude that bagging resulted in higher Test Accuracy than boosting. Although bagging resulted in the highest Test Accuracy we loose the interpretability of the tree based structure, since a bagged tree no longer is a tree (The Elements of Statistical Learning, second edition, s. 286). If interpretability is valued, the boosting procedure is worth considering despite the lower Test Accuracy.

Discussion

The different methods which we've examined throughout this project yield the following results when each of them have been tuned (although to different degrees).

Table 5: Final Test Accuracies

Method	Accuracy
Weighted 5-NN	0.984
Bagging	0.983
Radial SVM	0.915

Bagging decision trees resulted in the highest test accuracy, closely followed by the weighted 5-NN.

Bibliography

Appendix

```
library(tidyverse)
library(e1071)
library(ggpubr)
library(readxl)
library(scales)
library(rpart)
library(rpart.plot)
library(knitr)
library(psych)
library(ipred)
library(adabag)
library(gbm)
library(kknn)
library(grid)
library(gridExtra)

set.seed(2021) #Happy New Year

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` %>%
  kable(caption = "Unoccupied", digits = 3)

sStats$`1` %>%
  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

bestLC <- valLinearSVM(10^seq(-2, 2, by = 0.5))

l <- svm(Occupancy ~ ., data = trainingData, kernel = "linear", C = bestLC)
predL <- predict(l, testingData)
lAcc <- mean(predL == testingData$Occupancy)

#Radial

bestRC <- valRadialSVM(10^seq(-2, 5, by = 0.5))

r <- svm(Occupancy ~ ., data = trainingData, kernel = "radial", C = bestRC)
predR <- predict(r, testingData)
rAcc <- mean(predR == testingData$Occupancy)

#Polynomial

bestP <- valPolynomialSVM(10^seq(-2, 5, by = 0.5), deg = seq(2,5), coeff = 1)

p <- svm(Occupancy ~ ., data = trainingData, kernel = "polynomial", C = bestP$C, degree = bestP$D, coef0 = 1)
predP <- predict(p, testingData)
pAcc <- mean(predP == testingData$Occupancy)


data.frame(Kernel = c("Linear", "Radial", paste("Polynomial degree", bestP$D)),
           Cost = c(bestLC, bestRC, bestP$C),
           TestAccuracy = c(lAcc, rAcc, pAcc)) %>%
  kable(caption = "SVM Test Accuracies", digits = 3)


source("knn.R")

bestK <- valKNN(1:25)

k <- knn(trainingData[, !sapply(trainingData, is.factor)], testingData[, !sapply(testingData, is.factor)],
```

```
        cl = trainingData[,sapply(trainingData, is.factor)], k = bestK)

kAcc <- mean(testingData[, sapply(testingData, is.factor)] == k)

source("wknn.R")

bestWK <- valWKNN(1:50)

wk <- kknn(Occupancy ~ ., train = trainingData, test = testingData, kernel = "epanechnikov", k = bestWK)

wkAcc <- mean(testingData[, sapply(testingData, is.factor)] == wk$fitted.values)

formula <- Occupancy ~ .

#Basic decision tree
tree <- rpart(formula, data = trainingData, method = "class")
treepred <- predict(tree, testingData, type = "class")
testErrorTree <- mean(sqrt((as.numeric(as.character(treepred)) - as.numeric(as.character(testingData$Occupancy))


#Bagging
bagged <- ipred::bagging(formula, data = trainingData, nbagg = 1000, coob = TRUE)
#Out-of-bag error

bagpred <- predict(bagged, newdata = testingData)

testErrorBag <- mean(sqrt((as.numeric(as.character(bagpred)) - as.numeric(as.character(testingData$Occupancy)))^

#Boosting
#Boosting 1000 trees resulted in long computation time
boosttree <- boosting(formula, data = trainingData, mfinal = 100)

boostpred <- predict.boosting(boosttree, newdata = testingData)

testErrorBoost <- boostpred$error

data.frame(Method = c("Simple", "Bagging", "Boosting"),
           'Test Accuracy' = c(1-testErrorTree, 1-testErrorBag, 1-testErrorBoost)) %>%
  kable(caption = "Decision Trees", digits = 3)


data.frame(Method = c("Bagging", paste0("Weighted ", bestWK, "-NN"), "Radial SVM"),
           'Accuracy' = c(1-testErrorBag, wkAcc, rAcc)) %>%
  arrange(desc(Accuracy)) %>%
  kable(caption = "Final Test Accuracies", digits = 3)


#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 = "epanechnikov", 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)

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

```r
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

}

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")
```

```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

}
```

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

valWKNN <- function(K, kernel = "epanechnikov", 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)
```

```r
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) %>%
```

```r
ggplot(aes(x = Occupancy, y = Value, color = Occupancy)) +
geom_boxplot() +
facet_wrap(~Variable, scales = "free_y")
```