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" by the contributors (and authors of *Candanedo et al. 2016*).

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

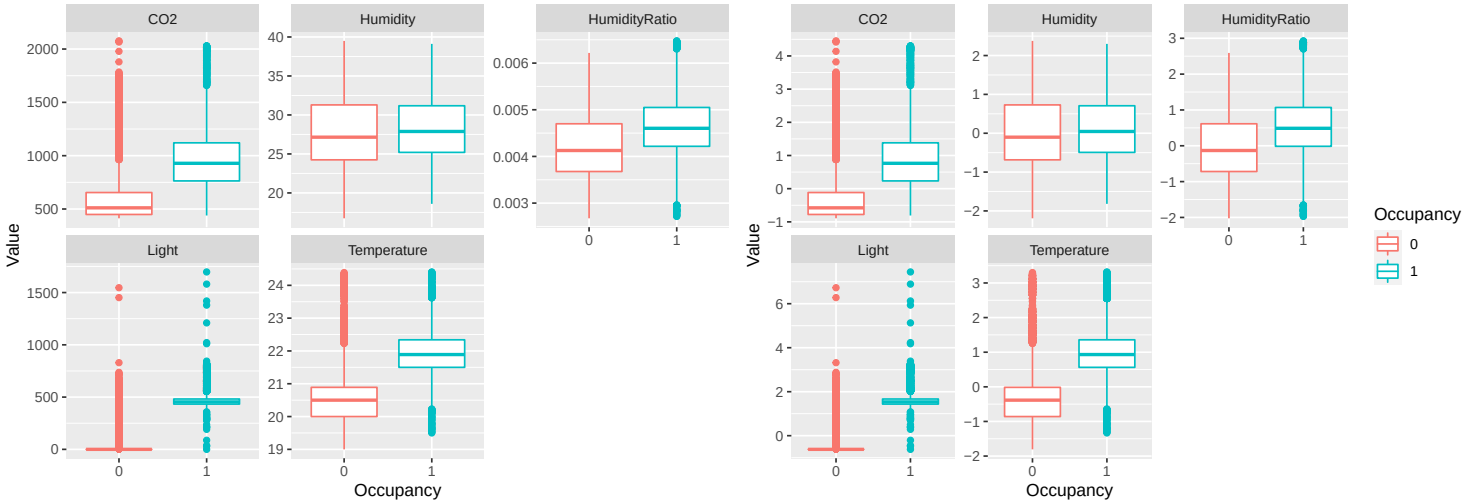
Table 2: Occupied

	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.

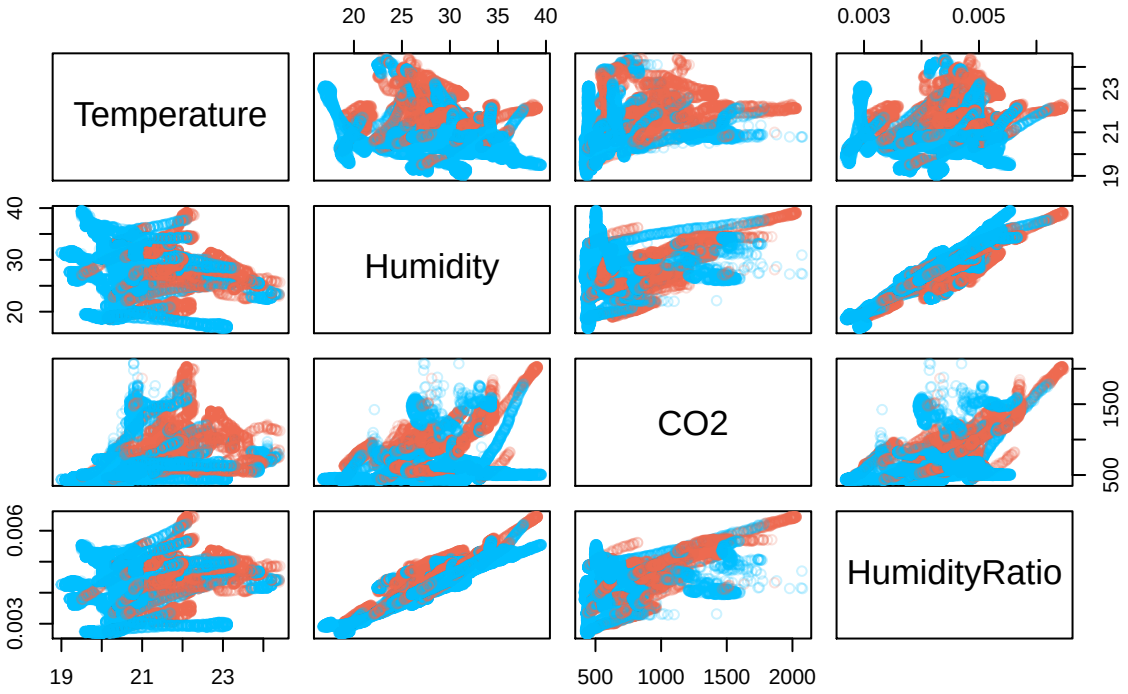
The class imbalance turned out to be a real issues since some classifiers simply classified the room as unoccupied at all times. As such we resorted to upsample both our training and our validation set in order to allow for the classification of both classes (SOURCE ON UPSAMPLING). In the end this proved 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 poorly in validation and testing scenarios. As such decided to foucs on classifiers that we can twek through validation such that they 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 decided to change focus as we assumed that the data could not support a very complex 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) which should make the minutes before and after occupancy difficult to distinguish from actual occupation.

Due to the aforementioned reasons we have resorted to some very simple classifiers that we think will generalize well. Those specifically being SVM and regularized Logistic Regression.

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.

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. The results of this search can be found in Figure X, Y and Z in the Appendix.

We seem to have similar testing 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
Radial	0.00100	0.83490
Polynomial Degree 4	0.00004	0.83752

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. As we initially thought there does seem to be a need to be quite heavily regularized in order to perform well on the validation set.

We can also examine the confusion matrices for the three models in order to examine whether the misclassifications are split evenly between the classes or not.

Table 4: Linear

	0	1
0	1409	144
1	284	828

Table 5: Radial

	0	1
0	1377	124
1	316	848

Table 6: Polynomial Degree 4

	0	1
0	1401	141
1	292	831

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).

Logistic Regression

Logistic regression is a good choice for binary classification problems like this one. We also look at different forms of regularized logistic regression as well as boosting to hopefully achieve higher test accuracies.

Implementation

Using the package `mboost` and the function `glmboost` with the adaboost algorithm in order to boost the logistic regression model. For regularized logistic regression we look at both lasso and ridge regression using the package `glmnet` and the function `glmnet`. Implementing these algorithms gave the following results.

Table 7: Logistic Test Accuracies

Kernel	TestAccuracy
Logit	0.86979
Boosting	0.85854
Lasso	0.85441
Ridge	0.83940

The simple logistic regression with no extra algorithms implemented resulted in slightly higher test accuracy than both boosting and regularized logistic regression. The algorithms implemented failed to improve the test accuracy and in the case of ridge regression even lowered the test accuracy significantly.

Discussion

When upsampling the minority class occupied to resolve the class imbalance in the data sets we saw a drastic increase in testing performance. Without the upsampling we achieved approximatley 72% accuracy for the linear SVM whilst we achieved 84% with the upsampled training and validation data (other methods simply achieved an accuracy of about 65% by classifying everything as unoccupied).

Although we have achieved decent classification accuracy we have to keep in mind that this is a binary classification problem and as such an accuracy of 50% can be obtained by randomly guessing, assuming that the data is balanced. We think that many of the misclassifications are due to the fact that we have cutoff observations (these being the first few minutes at the start of an occupancy and a few minutes after occupancy). These observations are very difficult (if not impossible) to determine the class of as the feature values are extremly similar to the minutes before when the occupancy status was different. The way in which these have been distinguished in the past (that being in the paper *Candanedo et al. 2016* and in *Tütüncü et al. 2018*) is through the inclusion of the feature `Light`. As mentioned this variable was excluded in our case since it essentially explains the occupancy status by entierly by itself (a decision tree that only seperates the classes by `Light` achieved an accuracy of about 98%).

Bibliography

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

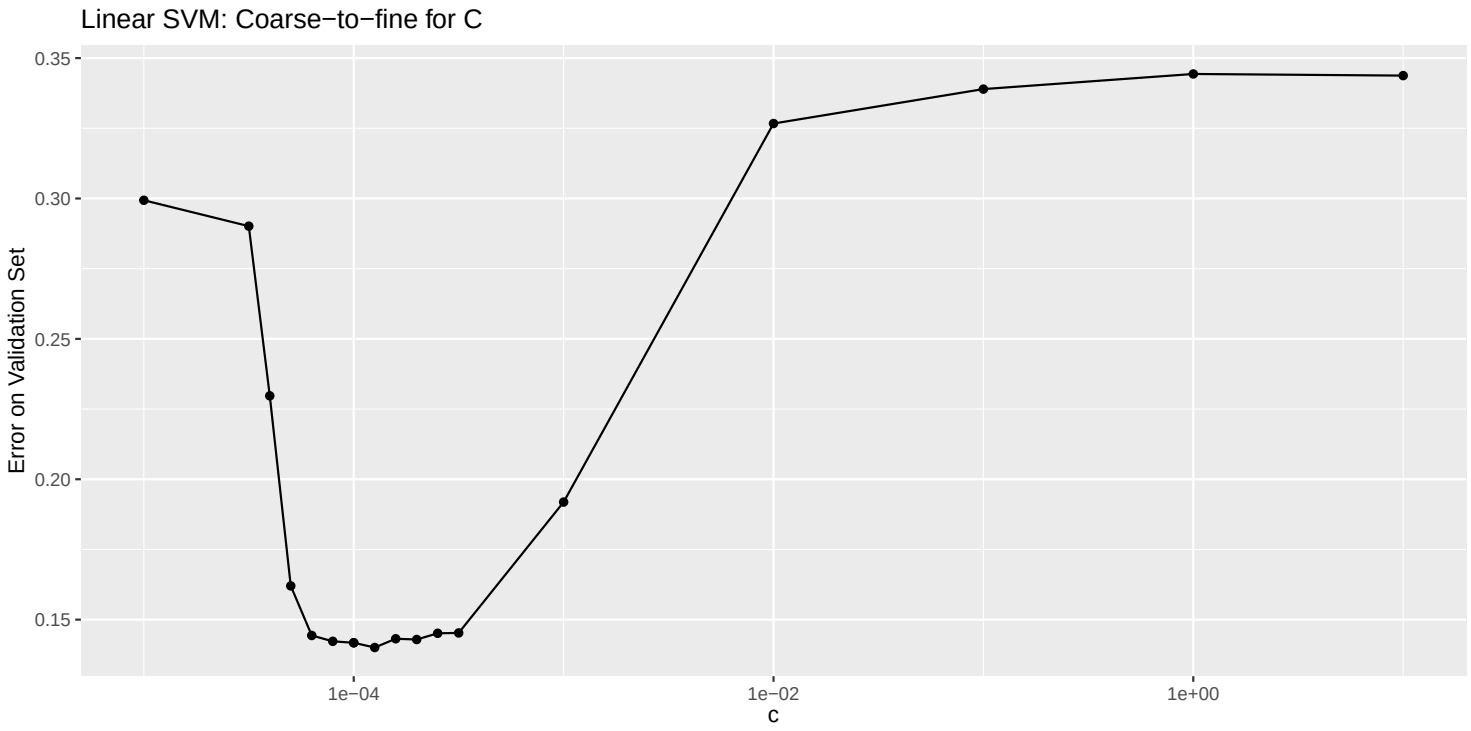
Paul, M. (2017), *Nonlinear Classification* [Presentation]. \ https://cmci.colorado.edu/classes/INFO-4604/fa17/files/slides-9_nonlinear.pdf

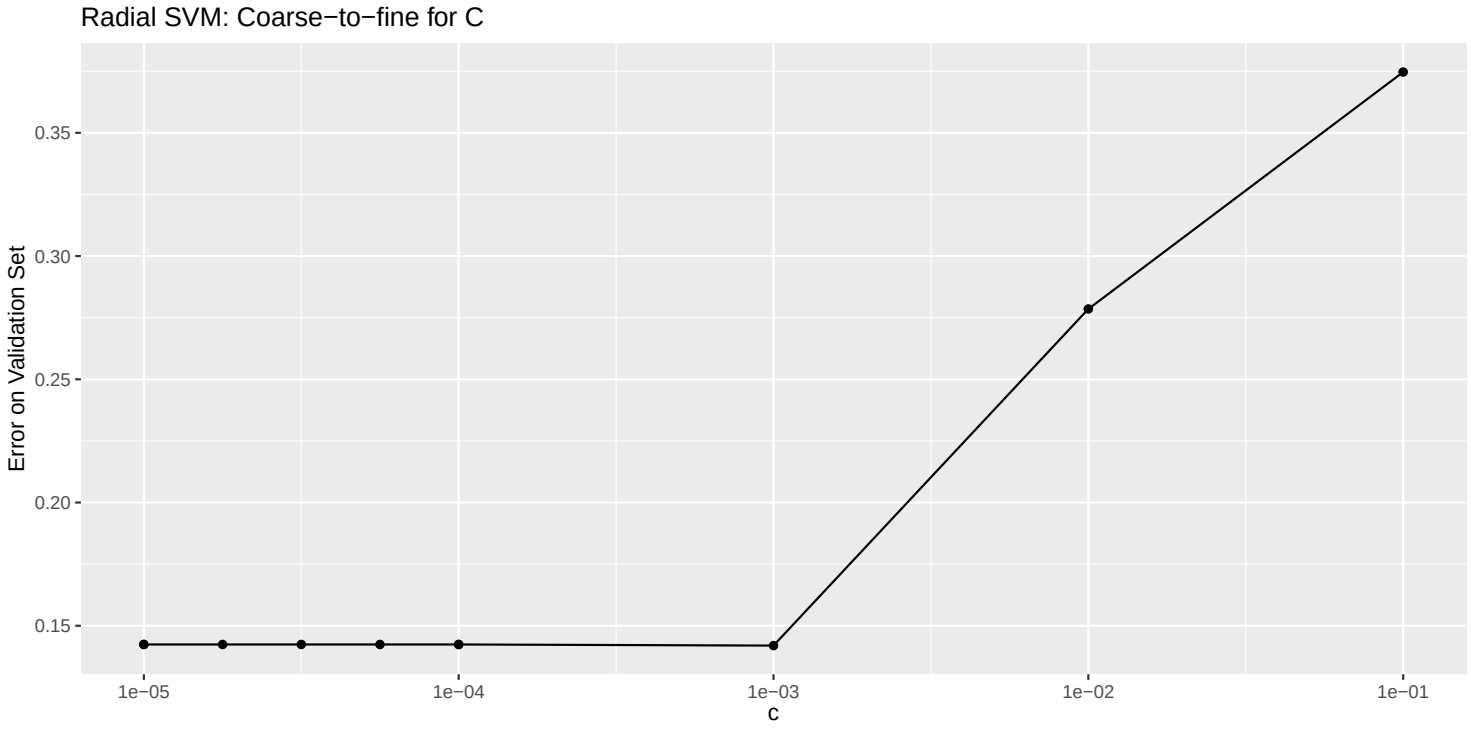
Candanedo, L. M., & Feldheim, V. (2016). *Accurate occupancy detection of an office room from light, temperature, humidity and CO 2 measurements using statistical learning models.* Energy and Buildings, 112, 28–39. doi:10.1016/j.enbuild.2015.11.071

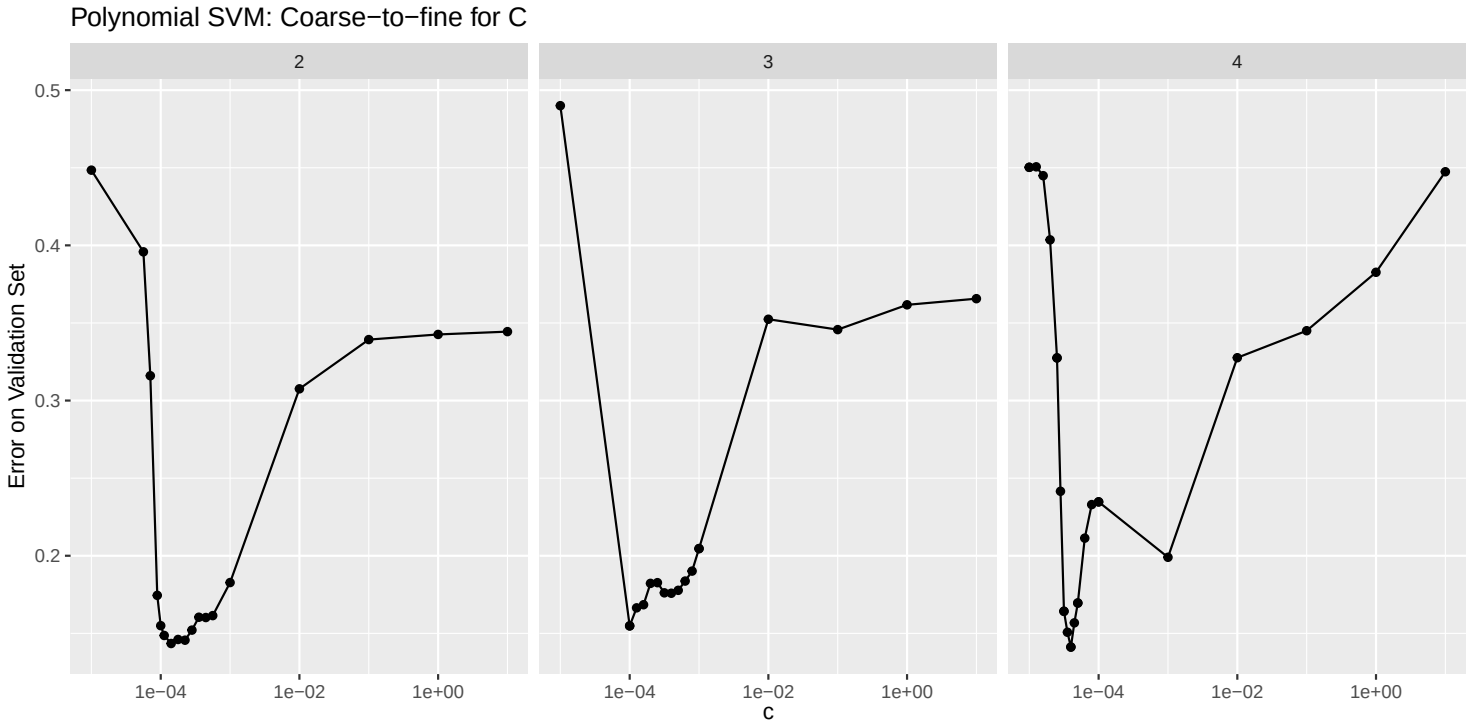
Tütüncü, Kemal & çataltaş, Özcan & Koklu, Murat. (2018). *Occupancy Detection Through Light, Temperature, Humidity and CO2 Sensors Using ANN.*

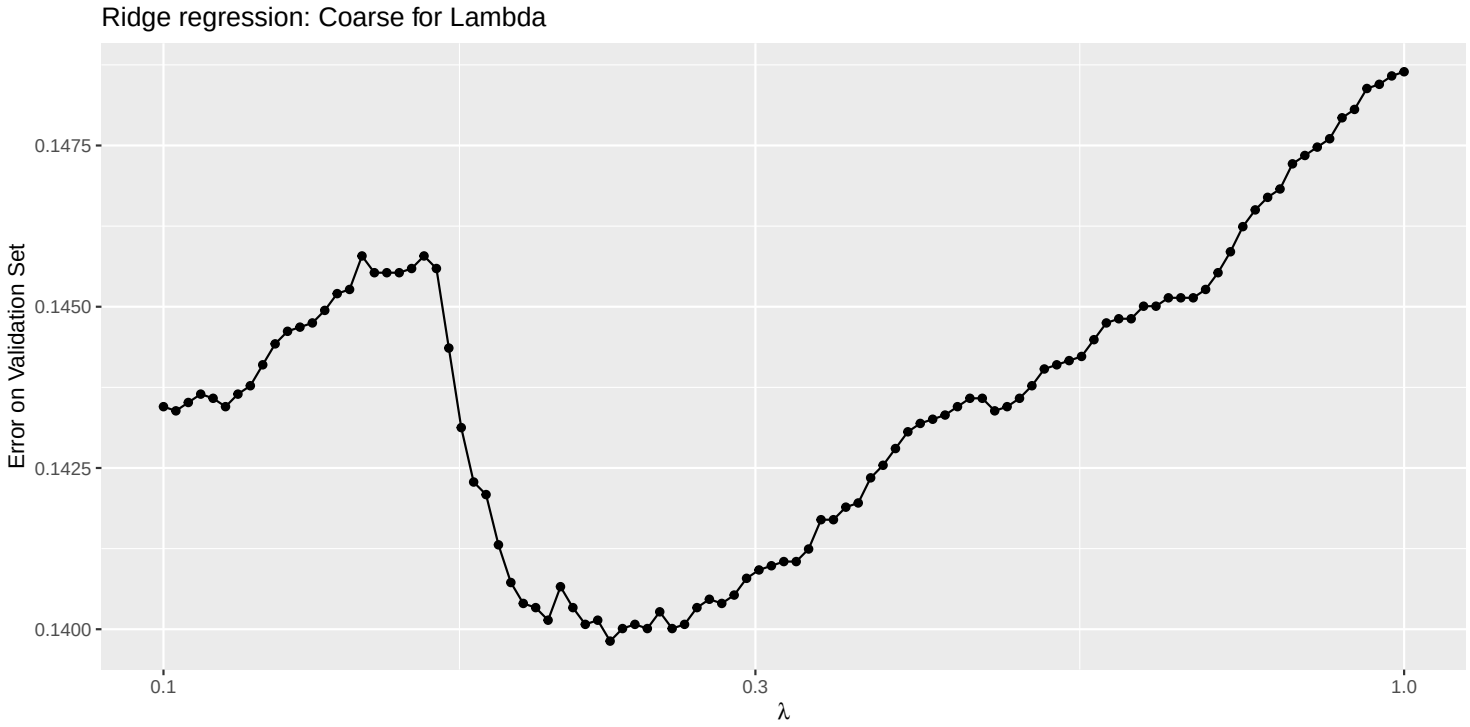
Appendix

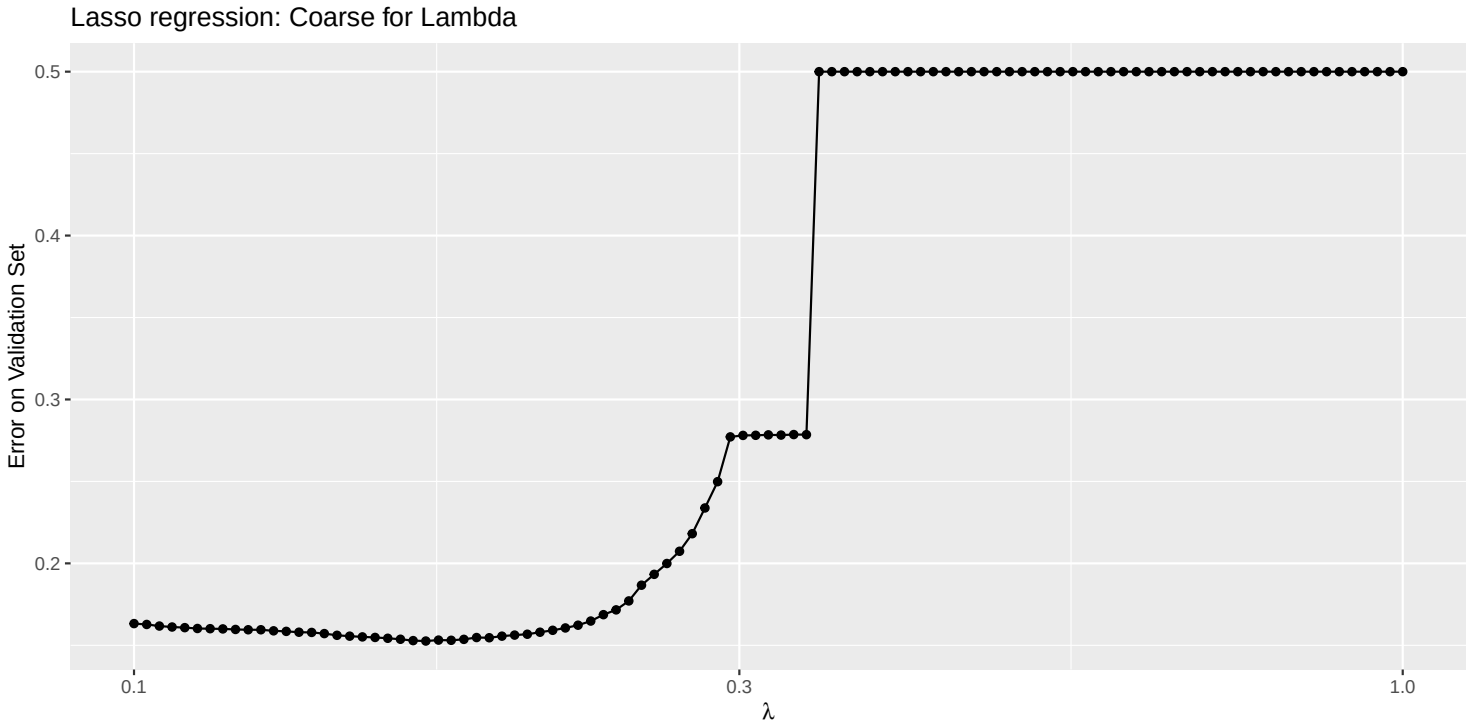
Figures






Polynomial SVM: Coarse−to−fine for C
2
3
4
0.5
0.4
0.3
0.2
Error on Validation Set
1e−04
1e−02
1e+00
C


Ridge regression: Coarse for Lambda
0.1475
0.1450
0.1425
0.1400
Error on Validation Set
0.1
0.3
1.0
λ



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

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)
```

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

#Polynomial

pcDF1 <- valPolynomialSVM(10^seq(-5, 1, by = 1), 2:4, 1)
pcDF2 <- valPolynomialSVM(10^seq(-4.25, -3.25, 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(-4.6, -4.3, 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)

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(rCM$table, caption = "Radial")

kable(pCM$table, caption = paste("Polynomial Degree", bestP$d))

source("boosting.R")

data.frame(Kernel = c("Logit", "Boosting", "Lasso", "Ridge"),
           TestAccuracy = c(1-logit.test.error, 1-boost.test.error, 1-lasso.test.error, 1-ridge.test.error)) %>%
          arrange(-TestAccuracy) %>%
  knitr::kable(caption = "Logistic Test Accuracies", digits = 5)

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)

lRidge %>%
  ggplot(aes(x = l, y = error)) +
  geom_line() +
  geom_point() +
  scale_x_log10() +
  ylab("Error on Validation Set") +
  xlab(expression(lambda)) +
  ggtitle("Ridge regression: Coarse for Lambda")

lLasso %>%
  ggplot(aes(x = l, y = error)) +
  geom_line() +
  geom_point() +
  scale_x_log10() +
  ylab("Error on Validation Set") +
  xlab(expression(lambda)) +
  ggtitle("Lasso regression: Coarse for Lambda")
```

data.r

Includes the initial manipulation of the data.

```
library(tidyverse)
library(e1071)
library(ggpubr)
library(readxl)
library(scales)
library(rpart)
library(rpart.plot)

#occupancy Dataset

rawTrainingData <- read.delim("../data/datatraining.txt", sep = ",") %>%
  select(-date)%>%
  mutate(Occupancy = factor(Occupancy)) %>%
  select(-Light)

rawValidationData <- read.delim("../data/datatest2.txt", sep = ",") %>%
  select(-date) %>%
  mutate(Occupancy = factor(Occupancy)) %>%
  select(-Light)

rawTestingData <- read.delim("../data/datatest.txt", sep = ",") %>%
  select(-date) %>%
  mutate(Occupancy = factor(Occupancy)) %>%
  select(-Light)

upsampling <- function(data){

  ones <- sum(data$Occupancy == 1)
  zeros <- sum(data$Occupancy == 0)

  dif <- zeros - ones

  if(dif > 0){

    toAdd <- data %>%
      filter(Occupancy == 1) %>%
      sample_n(dif, replace = TRUE)

  }else{

    toAdd <- data %>%
      filter(Occupancy == 0) %>%
      sample_n(dif, replace = TRUE)

  }

  rbind(data, toAdd)

}

upsampledTrainingData <- upsampling(rawTrainingData)
upsampledValidationData <- upsampling(rawValidationData)

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

  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(upsampledTrainingData)
validationData <- standardizeData(upsampledValidationData)
testingData <- standardizeData(rawTestingData)
```

svm.r

Includes the functions for validating different svm models.

```
source("data.R")

#SVM

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

  l <- rep(list(c(NA,NA)), length(sequence))

  i = 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))

    l[[i]] <- c(C, valError)

    i = i + 1

  }

  data.frame(matrix(unlist(l), ncol = 2, byrow = T)) %>%
    setNames(c("c", "error"))

}

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

  l <- rep(list(c(NA,NA)), length(sequence))

  i = 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))

    l[[i]] <- c(C, valError)

    i = i + 1

  }

  data.frame(matrix(unlist(l), ncol = 2, byrow = T)) %>%
    setNames(c("c", "error"))

}

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

  l <- rep(list(c(NA,NA,NA)), length(C)*length(deg))

  i = 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))

      l[[i]] <- c(c, d, valError)

      i = i + 1

    }

  }

  data.frame(matrix(unlist(l), ncol = 3, byrow = T)) %>%
  setNames(c("c", "d", "error"))

}
```