Support Vector Machine (SVM):

Parameter Optimization

- The **svm()** function uses several parameters which we have so far ignored
- One parameter is **cost** (default value = 1)
- It determines how strongly classification errors influence the resulting hyperplane
- Changing the value of the parameter can change the resulting classification drastically
- In order to determine which value for the parameter is optimal one can compare the resulting predictions on the test data and calculate for example Matthews correlation coefficient (MCC)
- **A higher MCC means less errors in the prediction**

Support Vector Machines

Example:

Import the data file called **“catsData.csv**” into the variable **mydata**

Process the dataset

- Divide **mydata** in two data frames as **dfTraining** (70%) and **dfTest** (30%)
- Create a vector of costs

>costVec = c(0.3,0.5,0.8,1,5,10,100,500,1000,5000,10000)

- Write a function which calculates the **MCC-value**
- Create a **for-loop** that iterates over the **costVec** and does the following:
  - Create a **svm** for classifying the data based on the sex with **dfTraining** (function in R: **svm**)
  - Create a **confusion matrix** for the predicted and true sex on **dfTest**
  - Calculate the **MCC** values and save it in a vector
- Plot the MCC values for the different cost values and find the optimal cost value

Matthews correlation coefficient (MCC)

$$\mathrm{MCC} = \frac{\mathrm{TP} \times \mathrm{TN} - \mathrm{FP} \times \mathrm{FN}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})(\mathrm{TN}+\mathrm{FP})(\mathrm{TN}+\mathrm{FN})}}$$

Support Vector Machines

Example:

Import the data file called **“catsData.csv**” into the variable **mydata**

```
> mydata= read.table(“catsData.csv”, header=TRUE, sep=“,”)
```

Process the dataset

- Divide **mydata** in two data frames as **dfTraining** (70%) and **dfTest** (30%)

```
>countTraining = round(nrow(mydata)*0.7)
>randomNumbers= sample(1:nrow(mydata), size = countTraining, replace = F) #randomSamples
>dfTraining =mydata[randomNumbers, ]
>dfTest =  mydata[-randomNumbers, ]
```

- Create a vector of costs

```
>costVec = c(0.3,0.5,0.8,1,5,10,100,500,1000,5000,10000)
```

- Write a function which calculates the MCC-values

Support Vector Machines

Example:

Process the dataset

- Create a function for calculating the MCC

Matthews correlation coefficient (MCC)

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

```
>calcMCC = function(truePos,trueNeg,falsePos,falseNeg){
mcc_ part1 = (truePos * trueNeg) - (falsePos*falseNeg) # numerator
mcc_ part2 =sqrt((truePos+falsePos) * (truePos+falseNeg) * (trueNeg+falsePos) * (trueNeg+falseNeg)) #
denominator
mcc= mcc_part1 / mcc_part2
return(mcc)
}
```

- Create a for-loop that iterates over the **costVec** and does the following:
  - Create a **svm** for classifying the data based on the sex with **dfTraining** (function in R: **svm**)
  - Create a **confusion matrix** for the predicted and true sex on **dfTest**
  - Calculate the MCC-values and save it in a vector

Example:

Process the dataset

- Create a for-loop that iterates over the **costVec** and does the following:
  - Create a **svm** for classifying the data based on the sex with **dfTraining** (function in R: **svm**)
  - Create a **confusion matrix** for the predicted and true sex on **dfTest**
  - Calculate the MCC and save it in a vector

```
>mccVec = c()
>for(i in 1:length(costVec)){
    mySVM = svm(Sex~., dfTraining, cost = costVec[i])
    myPred = predict(mySVM,dfTest)  # make prediction based on training
    confTable = table(myPred,dfTest$Sex)
    truePos = confTable[1,1]
    trueNeg = confTable[2,2]
    falsePos = confTable[1,2]
    falseNeg = confTable[2,1]
    mccVec[i] = calcMcc(truePos,trueNeg,falsePos,falseNeg)
}
```

Support Vector Machines

Example:

Process the dataset

- Plot the MCC values for the different cost values and find the optimal cost value

```
>plot(x= costVec, y = mccVec)
#Or with ggplot2
> ggplot(data = data.frame(Cost = costVec, MCC = mccVec), aes(x = Cost, y = MCC)) + geom_point()
```

Support Vector Machine (SVM):

Parameter Optimization

- The previous approach can be quite complex if you want to optimize multiple parameters
- R has the function **tune()** which automatically searches for the best parameter combination given a list of possible values for each parameter to be optimized

>tunedParam = tune(method, train.x, train.y, ranges)

- The optimal parameters can be accessed via

>optimalParam = tunedParam$best.parameters$ParamName

Support Vector Machines

Example:

Previous example with tune()

```
>costVec = c(0.3,0.5,0.8,1,5,10,100,500,1000,5000,10000)
>tunedParam = tune(svm, train.x = dfTraining[,2:3], train.y = dfTraining$Sex, ranges = list(cost = costVec))
>tunedParam$best.parameters$cost
```

- The resulting parameter might be different from the one we found
- This is because **tune()** uses a different way to measure the performance of a parameter