Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering
IDP Institut für Datenanalyse
und Prozessdesign

HS 2016

Statistisches Data Mining (StDM)

Woche 9

Aufgabe 1 Churn bei einem Telephonanbieter

In dieser Aufgabe wird das Churn-Verhalten (Wechseln zu einem anderen Anbieter) von Telekommunikationskunden untersucht. Der Wettbewerb unter den Telekommunikationsanbietern ist sehr gross. Viele Kunden wechseln jedes Jahr den Anbieter. Telekommunikationsunternehmen sind deshalb bestrebt, Kunden durch geeignete Marketing-Programme vom Wechseln abzuhalten. Dazu müssen jedoch die zu einem Wechsel neigenden Kunden identifiziert werden, was z.B. mit einem Klassifikationsverfahren aufgrund der Kundenmerkmale gemacht werden kann.

Ihnen steht nun ein Datensatz zur Verfügung, indem von 3333 Kunden eines US-Telekommunikationsunternehmens bekannt ist, ob sie den Anbieter gewechselt haben oder nicht. Zugleich kennen wir noch folgende Kundenmerkmale:

Variablenname	Kurzbeschreibung	Typ
AccountLength	Vertragsdauer in Monaten	integer
AreaCode	Postleitzahl	*drei* Integerwerte
IntlPlan	Spezialvertrag für internationale Anrufe	binär
VMailPlan	Voice-Mail-Plan	binär
VMailMsg	Anzahl Anrufe	integer
DayMins	Zeitdauer mit Gesprächen am Tag (in Minuten)	stetig
DayCalls	Anzahl Anrufe am Tag	integer
EveMins	Zeitdauer mit Gesprächen am Abend (in Minuten)	stetig
EveCalls	Anzahl Anrufe am Abend	integer
NightMins	Zeitdauer mit Gesprächen in der Nacht (in Minuten)	stetig
NightCalls	Anzahl Anrufe in der Nacht	integer
IntlMins	Zeitdauer mit internationalen Gesprächen (in Minuten)	stetig
IntlCalls	Anzahl internationale Anrufe	integer
CustServCalls	Anzahl Anrufe an den Kundenservice	integer
Churn	Kunde hat den Anbieter gewechselt oder nicht	binär

Im File `Churn2.dat` sind alle Variablen (d.h. Merkmale) klassiert worden d.h. in Bereiche gruppiert worden.

a) Führen Sie die Klassifikation mit der logistischen Regression durch.

```
library(MASS)
Churn2 <- read.table(file.path(baseDir, "Churn2.dat"), sep="", header = TRUE)
Ch2.mn = glm(Churn ~ ., data=Churn2, family = binomial())
```

b) Bestimmen Sie die Konfusions-Matrix und die naive (in sample) Fehlerrate. Was ist besonders unerfreulich bei diesen Werten in der Konfusions-Matrix? (Weshalb?)

```
  # In Sample
  table(true=Churn2$Churn, pred=predict(Ch2.mn, type = 'response') > 0.5)
```

```
##        pred
## true    FALSE TRUE
##   FALSE  2771   79
##   TRUE    347  136
```

Wir haben eine naive Fehlerrate von 12.8 Prozent. Die meisten der Fehlkassifizierungen geschehen bei der Klasse TRUE (347 von 483), die aber gerade für unsere Zwecke die wichtigere ist, und nur 79 von 2850 bei der Klasse FALSE. Deshalb ist das Resultat nicht wirklich brauchbar.

c) Beurteilen Sie das Verfahren mit dem Liftchart (immer noch in sample). Sie können das package `ROCR` verwenden.

```
  require(ROCR)
```

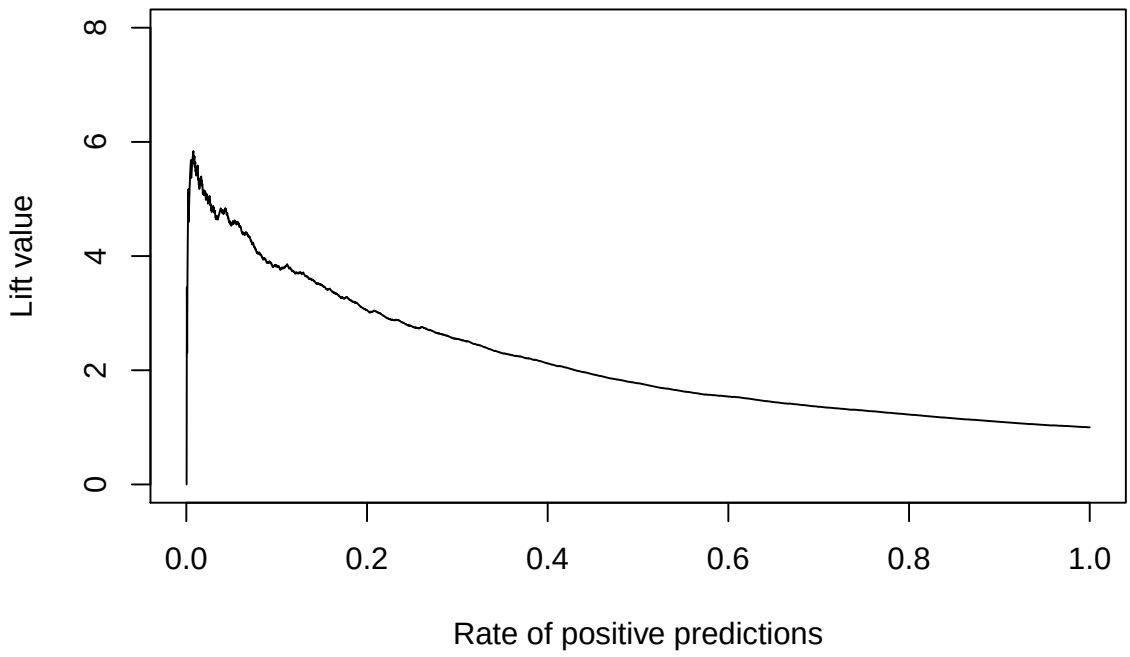
```
## Loading required package: ROCR
```

```
## Loading required package: gplots
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```
  predScore = predict(Ch2.mn, type="response")
  pred <- prediction(predScore, Churn2$Churn)
  perf_log <- performance(pred,measure="lift", x.measure="rpp")
  plot(perf_log, ylim=c(0,8))
```

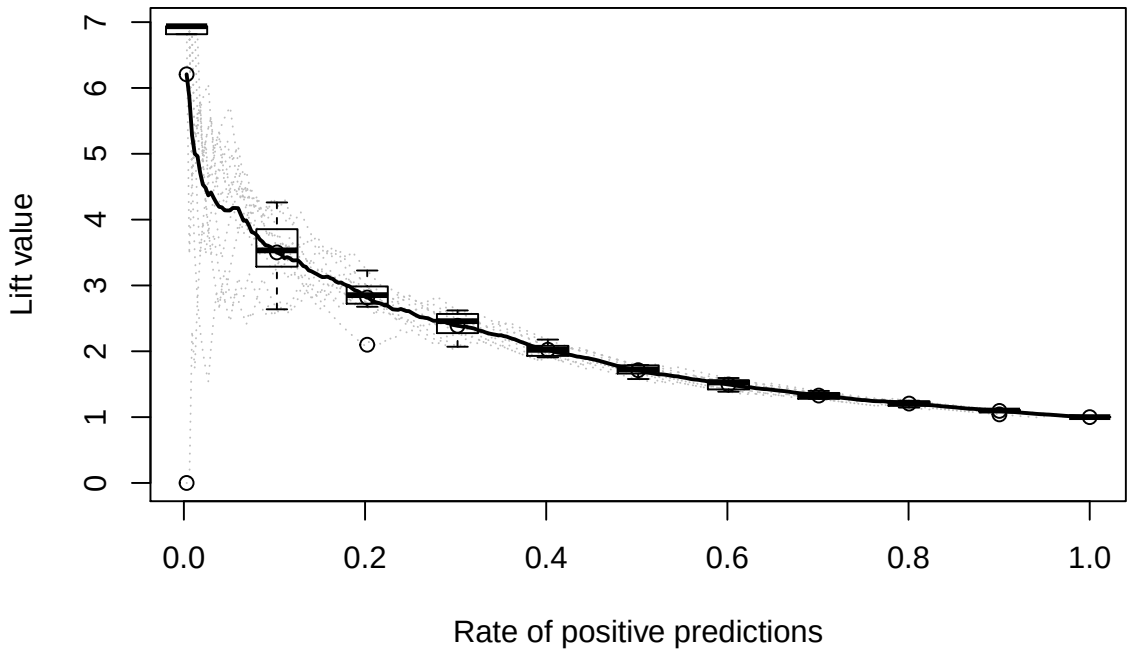


d) Führen Sie eine 10 Fold Crossvalidiation durch und plotten Sie die entsprechenden Lift Charts für die 10 Folds. Tipp: Verwenden Sie den parameter `newdata` in predict, um den Fit auf das Testset einzuschränken.

```r
library(caret)
preds = list()
labels = list()
test_folds = createFolds(Churn2$Churn, 10)
i = 1
for (test in test_folds) {
  fit = glm(Churn ~ ., data=Churn2[-test,], family = binomial())
  preds[[i]] = predict(fit, type="response", newdata=Churn2[test,])
  labels[[i]] = Churn2[test,]$Churn
  i = i + 1
}
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```r
pred <- prediction(preds, labels)
perf <- performance(pred,"lift","rpp")
plot(perf,col="grey",lty=3)
plot(perf,lwd=2,avg="vertical",spread.estimate="boxplot",add=TRUE)
```



Aufgabe 2 Overfitting (simulierter Datenset)

Simulieren Sie Daten wie folgt:

```
set.seed(1)
n<-400;p<-8
set.seed(100)
x<-matrix(rnorm(n*p),ncol=p)
y<-c(FALSE,TRUE)[as.numeric(apply(x^2,1,sum)>(p))+1]
data = data.frame(x=x,y=y)
```

a) Bestimmen Sie die in-sample Accuracy mit der LDA.

```
require(MASS)
fit = lda(y ~ x, data = data)
sum(predict(fit, data)$class == y) / length(y)
```

```
## [1] 0.5875
```

b) Bestimmen Sie die Accuracy mit einer 10 Fold XValidation.

```
require(caret)
test_folds = createFolds(y, 10)
cv = 0
for (test in test_folds) {
  fit = lda(y ~ ., data = data[-test,])
  cv = cv + sum(predict(fit, data[test,])$class == y[test])
}
cv / (length(y))
```

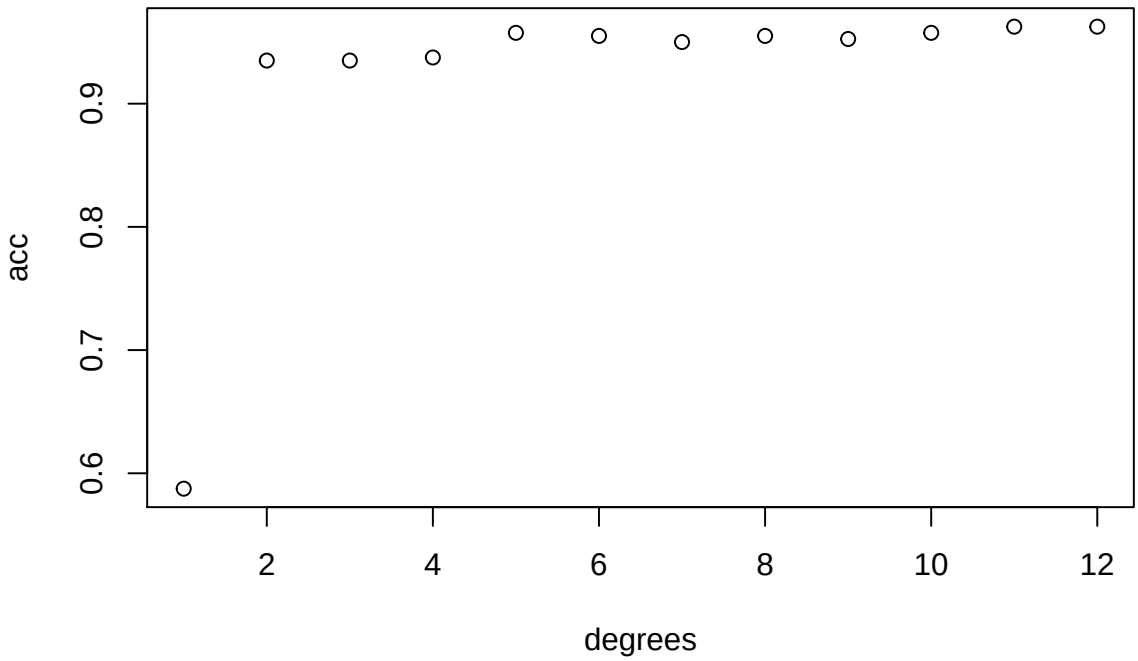
```
## [1] 0.5525
```

c) Fügen Sie nun Potenzen der ursprünglichen Variablen bis zu einem Grad `degree` hinzu. Bestimmen Sie die in-sample Accuracy für die Grade 1 bis 12 und tragen und tragen Sie diese

gegen den Grad auf. Zum Anfügen der Potenzen können Sie folgende Funktion verwenden:

```
addPolynoms = function(data, degree) {
  x_t = data
  x_t$y = NULL
  x_t = as.matrix(x_t)
  x_new = x_t
  if (degree > 1) {
    for (i in 2:degree) {
      x_new = cbind(x_new, x_t^i)
    }
  }
  data_new = data.frame(x=x_new, y=y)
}
```

```
getAccuracy_in_Sample = function(degree = 1) {
  data_new = addPolynoms(data, degree)
  fit = lda(y ~ ., data = data_new)
  sum(predict(fit, data_new)$class == y) / length(y)
}
degrees = seq(1,12)
acc = rep(NA, length(degrees))
i = 1
for (degree in degrees) {
  acc[i] = getAccuracy_in_Sample(degree)
  i = i + 1
}
plot(degrees, acc)
```



d) Wiederholen Sie c) mit einer 10 fachen Kreuzvalidierung. Tragen Sie beide Fehler gegen den Grad auf. Was fällt auf?

```
getAccuracy = function(degree = 1) {
  test_folds = createFolds(y, 10)
```

```
  cv = 0
  data_n = addPolynoms(data, degree)
  for (test in test_folds) {
    fit = lda(y ~ ., data_n[-test,])
    cv = cv + sum(predict(fit, data_n[test,])$class == y[test])
  }
  cv  / (length(y))
}
acc_fold = rep(NA, length(degrees))
i = 1
for (degree in degrees) {
  acc_fold[i] = getAccuracy(degree)
  i = i + 1
}
plot(degrees, 1-acc_fold, col='green', ylim = c(0.0,0.5))
points(degrees, 1-acc,pch="+")
```