6 Klassifikation

Fragen, die dieses Kapitel beantwortet:

- Wie kann man Objekte Gruppen zuordnen?
- Wie kann man Spam-Nachrichten anhand des Texts erkennen?
- Was kann man mit einem Bayes-Klassifikator noch machen?
- Was sind Support Vector Machines?
- Was versteht man unter Likelihood?
- Wie kann man eine allgemeinere Metrik für R^2 formulieren?
- Was versteht man unter einer Residuendevianz?

Bei der Klassifikation werden Objekte aufgrund ihrer Merkmale verschiedenen existierenden Klassen zugeordnet. Nummeriert man die Klassen durch, ist die Lernvariable bei der Klassifikation ein diskreter Wert (im Gegensatz zum kontinuierlichen Wert bei der Regression).

Eine Klassifikation kann durch die logistische Regression als eine Generalisierung des linearen Modells durchgeführt werden. Dies wird in Abschnitt 6.1 behandelt. Auch mit dem Perceptron-Algorithmus (Abschnitt 6.2) und mit Support Vector Machines (Abschnitt 6.3) kann diese Klassifikationsaufgabe erfüllt werden. Eine weitere Alternative für Klassifikationen sind Entscheidungsbaumverfahren, die in Abschnitt 6.4 behandelt werden.

Die Grundlagen zur Klassifizierung nach Bayes wurden bereits 1763 gelegt [Wik21g]. Die Erkennung von Junk-E-Mails mithilfe der Wahrscheinlichkeiten nach Bayes stieß auf große Beachtung und wird in Abschnitt 6.5 gezeigt.

Das kNN-Verfahren, welches die Abstände von Datenpunkten zur Klassifikation nutzt, wird in Abschnitt 6.6 gezeigt.

Das Kapitel schließt ab mit der Diskussion einer generalisierten Metrik, welche auf dem Prinzip der Log-Likelihood basiert.

6.1 Logistische Regression

Bei den Regressionen (Kapitel 5) haben wir das Verhalten einer kontinuierlichen abhängigen Variablen von einer oder mehreren unabhängigen Variablen beschrieben; wir hatten so etwas wie verbindende Eigenschaften gesucht. Im Gegensatz dazu wollen wir in diesem Kapitel eher trennende Eigenschaften suchen, also Trennlinien ziehen.

Betrachten wir Punkte auf einer Ebene (Bild 6.1 links). Im linken unteren Bereich befinden sich Punkte, die zu einer Klasse 0 gehören, die Dreiecke im rechten oberen Bereich gehören zu einer Klasse 1. Wir lassen jetzt die Information über die y-Werte weg und verwenden die y-Achse dazu, die Klassenzugehörigkeit darstellen zu können (Bild 6.1 rechts).

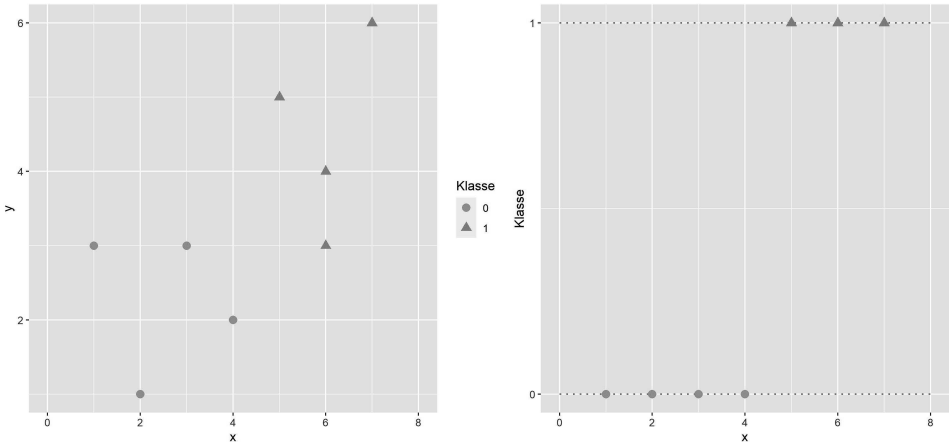

Bild 6.1 Klassifikation

Zur Bestimmung einer Trennlinie könnten wir einem ersten Ansatz einfach versuchen, über eine lineare Regression diese Grenze zu bestimmen und einzuzeichnen; die Grenze könnten wir bei 0.5 festlegen. Eine solche Gerade ist auf der linken Seite in Bild 6.2 gezeigt.

Diese Methode hat aber den Nachteil, dass sehr kleine oder sehr große x-Werte zu Vorhersagewerten führen, die sehr weit unterhalb von 0 oder oberhalb von 1 liegen können. Weiterhin reduzieren sie aufgrund der Hebelwirkung die Steigung der Geraden, wodurch die Trennschärfe reduziert wird.

Deshalb suchen wir eine Abbildungsvorschrift, die für $x \to -\infty$ gegen 0 geht, bei $x = 0$ den Wert 0.5 hat und für $x \to \infty$ gegen 1 geht. Eine Funktion, die das leistet, ist die logistische Funktion aus der Gruppe der Sigmoid-Funktionen (auch Schwanenhalsfunktionen, Fermifunktionen oder S-Funktionen genannt).

Die logistische Funktion wird beschrieben durch eine Gleichung

$$y = f(x) = \frac{e^x}{1+e^x} = \frac{1}{1+e^{-x}}$$

Die Umkehrfunktion der logistischen Funktion nennt man logit-Funktion:

$$x = logit(y) = f^{-1}(y) = ln\left(\frac{y}{1-y}\right) = ln(odds)$$

Die Verwendung der Variablen $odds$ ermöglicht eine kürzere Schreibweise.

Ein beispielhafter Verlauf der logistischen Funktion ist als gestrichelte Linie in Bild 6.2 rechts gezeigt.

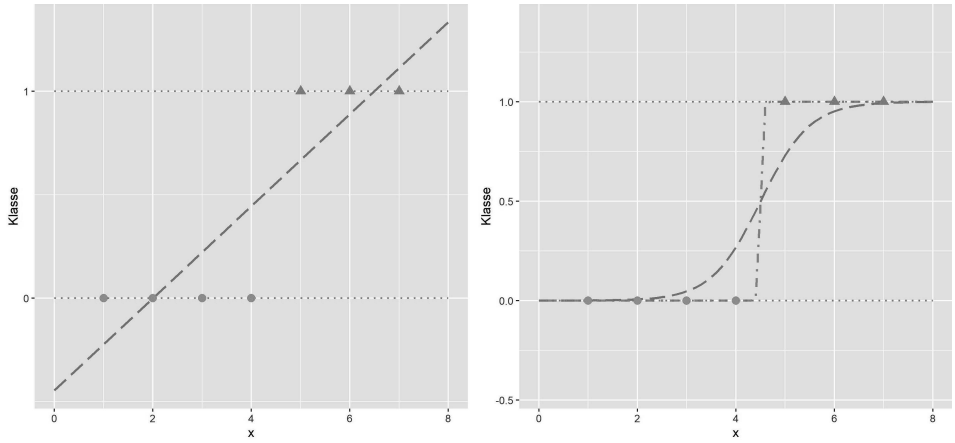

Bild 6.2 Lineare versus logistische Funktion

Bei einer linearen Regression (s. Kapitel 5) nimmt man eine Normalverteilung für die Störgröße an und kann damit Aussagen über die Streuung machen. Ändert man nun die Abbildungsfunktion beispielsweise von einer linearen Funktion auf eine logistische Funktion ab, ist diese Annahme natürlich nicht mehr gültig. Dadurch sind manche Werte (wie z. B. der Vertrauensbereich) nicht mehr in der gewohnten Weise bestimmbar.

Die Einschränkung der Theorie der Regressionen auf normalverteilte Werte wurde durch die Arbeit von [NW72] aufgehoben und auf die gesamte Klasse der Exponentialfamilie erweitert. Neben der Gauß'schen Normalverteilung sind darin auch die Binomial-, Poisson-, Gamma- und inverse Gauß-Verteilung einbezogen. Diese Modelle nennt man generalisierte lineare Modelle (GLM).

In R können Regressionen dieser Klasse über die Funktion `glm()` aufgerufen werden:

```
> logistische_regression <- glm(label ~ x,
                                family = binomial("logit"))
Warnmeldung:
glm.fit: Angepasste Wahrscheinlichkeiten mit numerischem Wert 0 oder 1
    aufgetreten
> summary(logistische_regression)
Call:
glm(formula = label ~ x, family = binomial("logit"))
Deviance Residuals:
       Min          1Q      Median          3Q         Max
-1.364e-05  -2.110e-08   0.000e+00   2.110e-08   1.378e-05
    :          :            :           :            :
```

Wir erhalten eine Warnmeldung, da die logistische Funktion erst für $z \to \pm\infty$ die Werte 0 oder 1 annimmt und man solche Werte in der Praxis kaum vorfindet. Die Vorhersagewerte der Regression kann man erhalten über:

```
> predict_y <- predict(logistische_regression,x,type="response")
```

Der Verlauf der Trennlinie ist als strichpunktierte Linie in Bild 6.2 rechts gezeigt.

Man erhält damit Werte zwischen 0 und 1; liegt ein Wert oberhalb von 0.5, zählt er zur Klasse 1, im anderen Fall zur Klasse 0.

Eine Anwendung der logistischen Regression ist das Problem mit den Dichtungen der Raumfähre „Challenger“ der NASA, welche im Jahr 1986 bei ihrem Start zerbrach. Im Datensatz `shuttle` des Package `SMPracticals` sind die Fehlerzahlen bei bestimmten Temperaturen und Drücken festgehalten (Bild 6.3 links).

Wir bilden aus den Fehlerzahlen die kategoriale Variable „Problem“. Sie ist TRUE, wenn mehr als ein Fehler auftritt. Wir tragen „Problem“ über der Temperatur auf und führen eine logistische Regression durch. Das Ergebnis ist in Bild 6.3 rechts gezeigt.

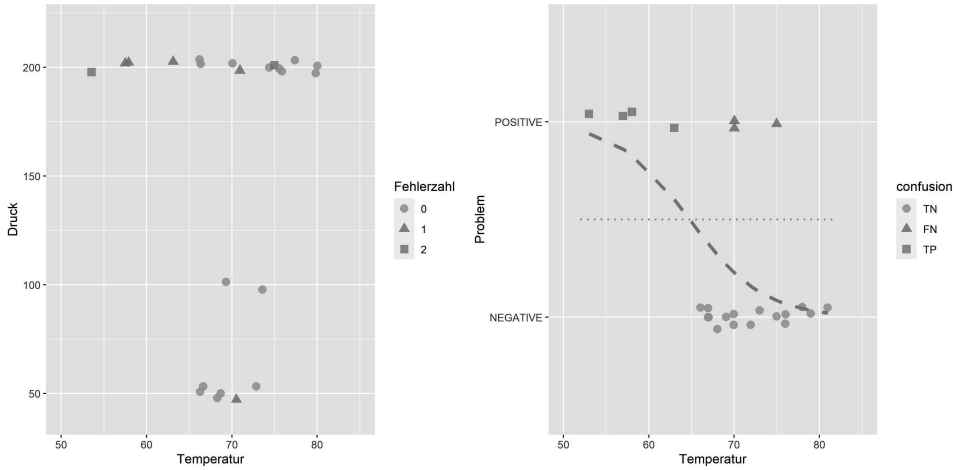

Bild 6.3 Logistische Regression der Dichtungen

Mit abnehmender Temperatur steigt die Wahrscheinlichkeit für ein Problem.

Sachverständige haben festgestellt, dass die mangelhafte Temperaturbeständigkeit der Dichtungsringe für diese Katastrophe mit verantwortlich war [Wik21h].

Aus den vorhergesagten und aktuellen Werten können wir eine Tabelle (Bild 6.4) aufbauen und daran einige Bewertungsmaße betrachten:

```
> ringe$problem_pred
[1] 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
> ringe$problem_actual
[1] 1 1 1 1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0
```

Diese Tabelle vergleicht die Zahl der wirklichen mit den vorhergesagten Werten. In den Spalten trägt man die Häufigkeit der vorhergesagten Werte ein, verglichen mit den wirklichen Häufigkeiten (die aktuellen Klassenzugehörigkeiten) in den Zeilen. Diese Tabellen nennt man Konfusionstabellen (oder auch Konfusionsmatrix, Confusion Matrix oder allgemein Kontingenztabellen):

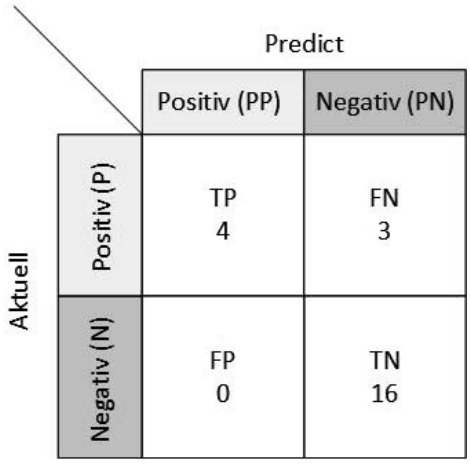

Bild 6.4 Konfusionstabelle

„True positiv“ (TP, auch „hit“ genannt) bedeutet: Es wurde eine positive Vorhersage (predicted positive, PP) getroffen, wobei dies dem aktuellen Wert (positive, P) entspricht. Bei unserem Datensatz trat dies viermal auf.

Entsprechend bedeutet „True negative“ (TN, auch „korrekte Rückweisung“ oder „correct rejection“ genannt) die zutreffende Vorhersage eines negativen Ergebnisses.

„False positive“ (FP, falsch positiv) – hier wurde positiv vorhergesagt, der aktuelle Wert ist negativ. Zusammen mit „False negative“ erhält man so die Summe der Fehlklassifikationen.

Zeilen und Spalten von Konfusionstabellen sind nicht einheitlich geregelt. Sie sind häufig vertauscht.

Anhand dieser Konfusionsmatrix kann man eine Bewertung vornehmen, indem man die Zahl der Korrekt-Klassifikationen zur Gesamtzahl der Klassifikationen ins Verhältnis setzt. Diesen Wert nennt man Korrekt-Klassifikationsrate (auch Treffergenauigkeit oder Accuracy genannt).

Entsprechend ist die Fehlklassifikationsrate das Verhältnis der Fehlklassifikationen zur Gesamtzahl der Klassifikationen.

Unter der Sensitivität (auch Abdeckung, recall, hit rate oder true positive rate (TPR) genannt) versteht man das Verhältnis von true positive zu positive:

$$TPR = \frac{TP}{P} = \frac{TP}{TP+FN} = \frac{4}{7}$$

Dieser Wert ist beeinflussbar durch das Verhältnis von actual positive (P) zu allen Fällen. Dieses Verhältnis nennt man die Prävalenz (PRV):

$$PRV = \frac{P}{P+N} = \frac{7}{23}$$

Ist die Prävalenz sehr groß, hätten wir schon dadurch gute Vorhersagewerte, indem wir immer „positiv“ tippen (dies macht man sich bei Glücksspielen zu eigen).

Die Falsch-Positiv-Rate (false positive rate, fall-out, FPR) ist das Verhältnis der Falsch-positiven Werte zu den negativen:

$$FPR = \frac{FP}{N} = \frac{FP}{FP+TN} = \frac{0}{16}$$

Ein anderes Maß ist die Spezifizität (true negative rate, TNR), das Verhältnis von korrekten Rückweisungen zur Zahl aktueller negativer Werte (N):

$$TNR = \frac{TN}{N} = \frac{TN}{TN+FP} = \frac{16}{16+0}$$

Bei sehr kleinen Prävalenzwerten haben wir schon dadurch gute Vorhersagewerte, indem wir immer N vorhersagen. In diesem Fall ist TPR nur bedingt tauglich für eine Bewertung von Klassifikationen.

Ein anderes Bewertungsmaß ist die Präzision (auch positiver Vorhersagewert, positive predictive value, PPV):

$$PPV = \frac{TP}{TP+FP} = \frac{4}{4} = 1$$

Nach diesem Maßstab wäre unser Klassifikator in unserem Beispiel „präzise“, was der allgemeinen Vorstellung über Präzision sicherlich nicht entspricht. Dieses Phänomen wird als Genauigkeitsparadoxon bezeichnet.

Zur Vermeidung dieses Nachteils kann man das F1-Maß als harmonisches Mittel zwischen Recall und Precision verwenden:

$$F1 = 2 \cdot \frac{1}{\frac{1}{TPR} + \frac{1}{PPV}} = 2 \cdot \frac{TPR \cdot PPV}{TPR + PPV} = \frac{\frac{4}{7} \cdot 1}{\frac{4}{7} + 1} = \frac{4}{11}$$

Der Nachteil dieses Maßes ist die fehlende Berücksichtigung der TN-Fälle.

In Abschnitt 6.7 werden wir uns mit der Frage der Metriken noch weiter auseinandersetzen.

Generalisierte Modelle können – wie im Fall der linearen Regression – auch für die Regression mit mehreren Variablen eingesetzt werden:

```
> multiple_logistische_regression <- lm(label ~ x1 + x2)
```

Auch wenn dieses Verfahren „logistische Regression“ heißt, zählt es zu den Klassifikationsverfahren.

Wir werden die logistische Regression in Abschnitt 9.3 wieder verwenden.

6.2 Der Perceptron-Algorithmus

Eine andere Möglichkeit zur Durchführung von Klassifikationen ist der Perceptron-Algorithmus. Dieser Algorithmus ist ein historischer Ansatz von [ROS58], welcher die Grundlage neuronaler Netze gebildet hat.

Wir betrachten wieder Punkte auf einer Ebene (Bild 6.5). Im linken unteren Bereich befinden sich Punkte, die zu einer Klasse -1 gehören, die Dreiecke im rechten oberen Bereich gehören zu einer Klasse $+1$. Gesucht ist eine Gerade als Trennlinie.

Wir starten mit einer willkürlich festgelegten Geraden durch die Punkte $(2;0)$ und $(0;2)$ und verschieben und rotieren sie solange, bis sie beide Klassen möglichst gut trennt.

Bei einer Beschreibung der Geraden durch eine Gleichung der Form $y = m \cdot x + b = -x + 2$ könnte eine Rotation durch Ändern von m und eine Verschiebung (Translation) durch Ändern von b durchgeführt werden (so haben wir es im Fall der Bestimmung einer Regressionsgeraden in Abschnitt 5.1 praktiziert).

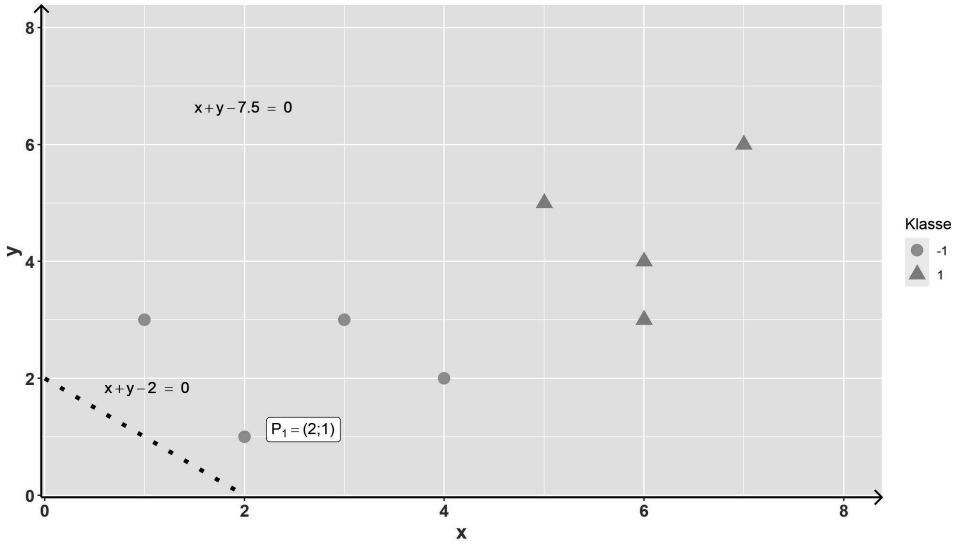

Bild 6.5 Der Perceptron-Algorithmus

Diesmal wollen wir jedoch neben der Translation anstelle eines Dreh- und Angelpunkts zwei Dreh- und Angelpunkte (Pivot-Punkte) verwenden. Dies gelingt durch die Darstellung der Geraden in der Form

$$a \cdot x + b \cdot y + c = 0, \qquad \text{also z. B.} \qquad x + y - 2 = 0$$

Dann haben wir zwei Dreh- und Angelpunkte (Pivotpunkte), nämlich die Schnittpunkte mit der x- bzw. y-Achse, die wir verschieben:

- Erhöht man den Wert von a, verbleibt der Schnittpunkt mit der y-Achse unverändert, während sich der der Schnittpunkt mit der x-Achse nach links verschiebt (und umgekehrt).
- Erhöht man den Wert von b, verbleibt der Schnittpunkt mit der x-Achse unverändert, während sich der Schnittpunkt mit der y-Achse nach unten verschiebt (und umgekehrt).
- Erhöht man den Wert von c, verschiebt man die Gerade (ohne Drehung) nach links unten.

Die gewählte Geradendarstellung kann man auch dazu verwenden, die Lage eines Punktes bezüglich der Geraden festzustellen. Dazu setzt man einfach seine Koordinatenwerte, beispielsweise den Punkt $(2,1)$, in die Geradengleichung ein:

$$\text{Lage} = a \cdot x + b \cdot y + c, \qquad \text{also z. B.} \qquad \text{Lage} = 1 + 2 - 2 = 1 > 0$$

$Lage$ hat einen positiven Wert. Das bedeutet, dass sich der Punkt rechts von der Geraden befindet.

Nun müssen wir noch herausfinden, ob der Punkt auf der richtigen Seite der Geraden liegt. Hierzu verwenden wir seine Klassenzuordnung. Der Punkt $(2,1)$ gehört zur Klasse -1. Das bedeutet, er sollte links der Geraden liegen.

Das heißt, alle Punkte, deren Produkt von $Klasse \cdot Lage$ größer als 0 ist, liegen auf der richtigen Seite. Diese Punkte werden dann nicht weiter berücksichtigt.

Der Punkt P_1 im Ausschnitt von Bild 6.6 liegt auf der falschen Seite. Wir könnten jetzt die Gerade auf einmal um den ganzen Abstand zum Punkt verschieben, damit der Punkt auf der richtigen Seite liegt. Hierzu müssten wir a um die x-Koordinate des Punkts verkleinern und b um die y-Koordinate. Aber wir bewegen die Gerade nur kleines Stück in Richtung Ziel, indem wir die Änderungen von a und b mit einem Bruchteil, der Lernrate λ, multiplizieren. c wird ebenso um diese Lernrate verkleinert. Dieses Prinzip wird als Hebb'sche Lernregel bezeichnet [Heb50].

Die Lernrate wird in der Regel auf einen Wert zwischen 0.001 und 0.1 festgelegt. Hat man eine zu kleine Lernrate gewählt, kann es lange dauern, bis eine Lösung gefunden wurde. Ist die Lernrate zu groß, „schießt man über das Ziel hinaus". Wir wählen in unserem Beispiel für eine bessere Darstellung eine Lernrate von $\lambda = 0.2$ (was einem sehr, sehr großen Wert entspricht).

Mit dieser Lernrate und der Lage des Punktes P_1 erhalten wir dann die neuen Werte

$$\begin{aligned} a_{neu} &= a - \lambda \cdot P_{x1} &&= 1 - 0.2 \cdot 2 &&= 0.6 \\ b_{neu} &= b - \lambda \cdot P_{y1} &&= 1 - 0.2 \cdot 1 &&= 0.8 \\ c_{neu} &= c - \lambda &&= -2 - 0.2 &&= -2.2 \end{aligned}$$

Die neue Geradengleichung lautet also: $0.6 \cdot x + 0.8 \cdot y - 2.2$.

Sie liegt nun rechts von $P_1 = (2;1)$, denn $0.6 \cdot 2 + 0.8 \cdot 1 - 2.2 = 1.2 + 0.8 - 2.2 = -0.2 < 0$.

Der Punkt P_1 wird bei der Optimierung der Lage nicht weiter berücksichtigt, solange er auf der richtigen Seite liegt.

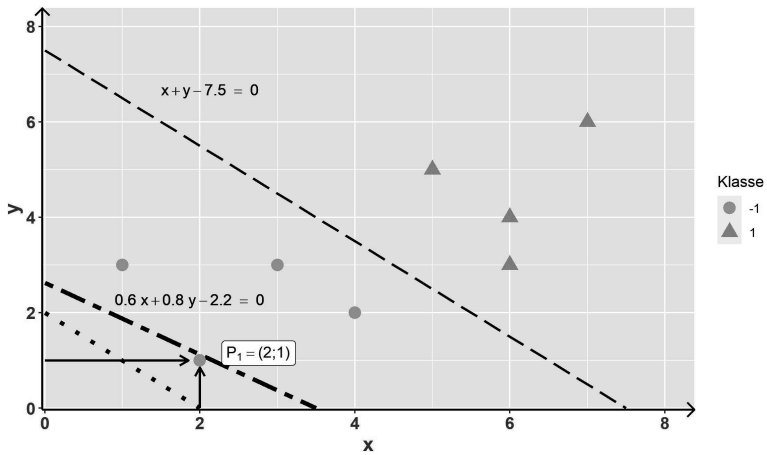

Bild 6.6 Der Perceptron-Algorithmus

In weiteren Iterationsschritten werden zufällig andere Punkte ausgewählt. Liegen sie auf der falschen Seite, wird die Gerade auf die gleiche Weise weiter in Richtung einer Lösung manövriert, bis wir die beiden Bereiche getrennt haben (gestrichelte Linie in Bild 6.6).

6.3 Support Vector Machines

Mithilfe des Perceptron-Algorithmus konnten wir durch ein iteratives Verfahren zwei Bereiche voneinander trennen. Sobald aber beim Perceptron-Algorithmus eine Trennung erfolgreich war, endet der Algorithmus. Bild 6.7 zeigt schematisch die Ergebnisse zweier Durchläufe, welche bei verschiedenen Ausgangswerten gestartet wurden. Offenbar wird durch die gestrichelte Linie eine bessere Trennung als durch die strichpunktierte Linie erzielt. Aber der Perceptron-Algorithmus war schon zufrieden, wenn kein Punkt mehr auf der falschen Seite erkannt wird.

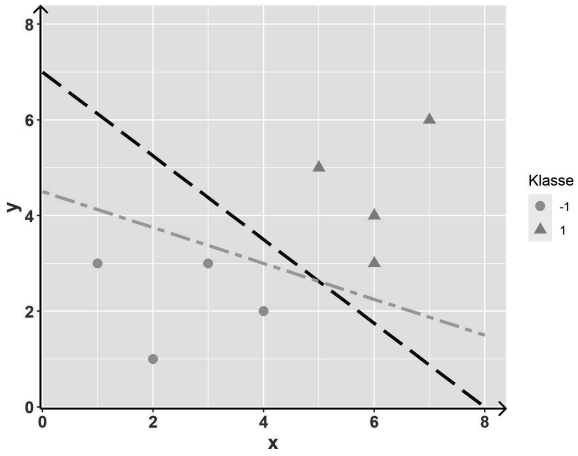

Bild 6.7 Zwei verschiedene Trennlinien mit dem Perceptron-Algorithmus

Support Vector Machines ermöglichen es, bessere Trennlinien zu erzeugen [HCL03].

Man geht aus von der Gleichung einer Startgeraden:

$$a \cdot x + b \cdot y + c = 0, \qquad \text{z. B.} \qquad x + y - c = 0$$

Von dieser Gleichung erzeugt man zwei Parallelen, indem man einen kleineren Wert e auf der rechten Seite einmal addiert und einmal subtrahiert.

$$\begin{array}{rcrcrcr} a \cdot x & + & b \cdot y & + & c & = & -e, \\ a \cdot x & + & b \cdot y & + & c & = & e, \end{array} \qquad \begin{array}{lrcrcrr} \text{z. B.} & x & + & y & -c & = & -0.5 \\ \text{z. B.} & x & + & y & -c & = & 0.5 \end{array}$$

Nun wird der Perceptron-Algorithmus angewandt, sodass zum einen alle runden Punkte links von der strichpunktierten Linie und die Dreiecke alle rechts von der gestrichelten Linie zu liegen kommen. Gleichzeitig wird der Abstand der beiden Parallelen bei jeder Iteration um einen kleinen Betrag vergrößert, indem man die linke Seite der Gleichungen

mit einem Faktor um einen Wert nahe und kleiner als 1 (die Ausdehnungsrate) multipliziert. Zum Schluss hat man eine Gerade, die die Punktepaare durch Parallelen mit größtem möglichen Abstand zueinander trennt (Bild 6.8).

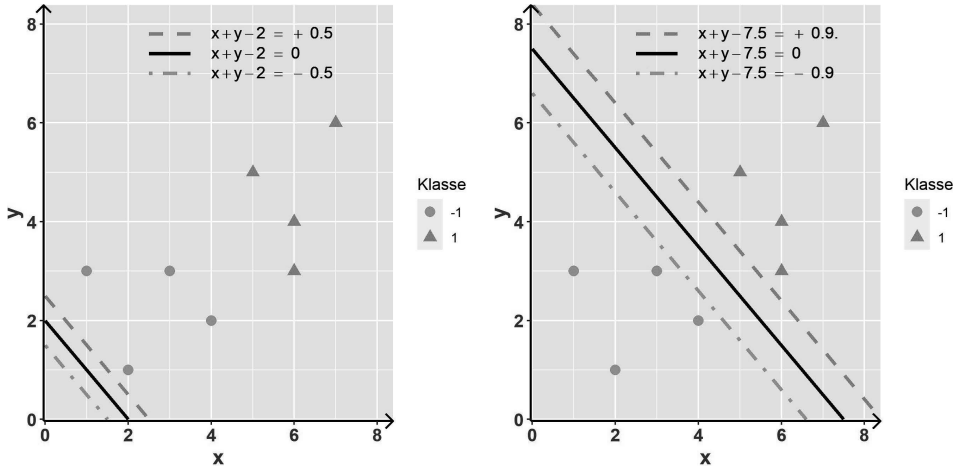

Bild 6.8 Trennung durch den Support-Vector-Algorithmus

Je nach Datenlage ist die korrekte Trennung der Punkte nicht das Optimum. Betrachten wir den Fall, dass der Bereich zwischen den Punkten sehr gering geworden ist (Bild 6.9). Ist nun das linke oder das rechte Ergebnis besser? Wir können dies bei den vorhandenen Informationen nicht entscheiden. Man kann jedoch einen Wert für das Verhältnis zwischen dem mittleren Abstand der falsch klassifizierten Punkte und dem Abstand zwischen den beiden Parallelen als Ziel vorgeben.

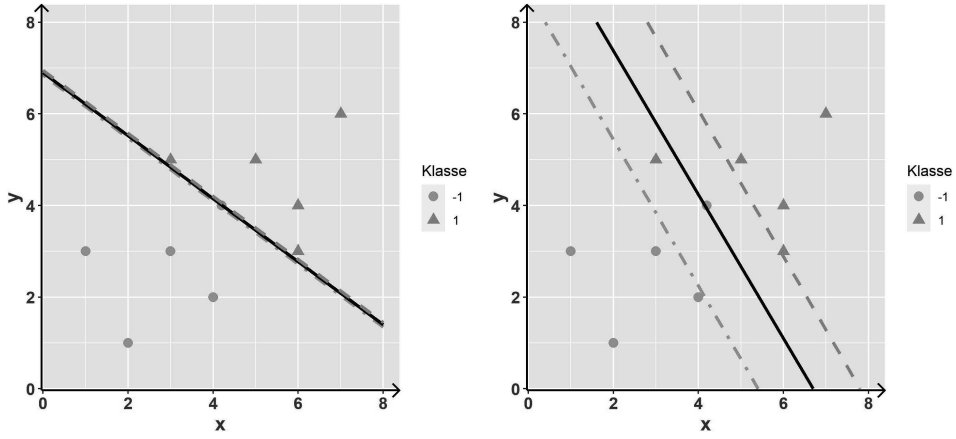

Bild 6.9 Welche Trennlinien sind besser?

6.3.1 Der Iris-Datensatz

Wir wollen mithilfe der Support-Vektor-Maschine versuchen, Trennebenen zwischen die verschiedenen Typen der Schwertlilien-Arten des Iris-Datensatzes zu ziehen. Dass dies nicht einfach werden dürfte, zeigt beispielhaft der Plot, bei dem für die Gruppen Ellipsen zur Abgrenzung eingezeichnet wurden.

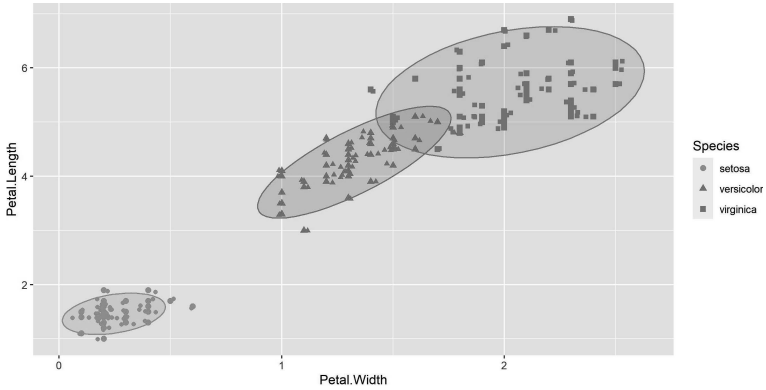

Bild 6.10 Der Iris-Datensatz mit dem ellipse-Geom

Diese Ellipsen werden über das Geom `stat_ellipse` erstellt. Die Bestimmung der Richtungen haben wir bei der Hauptkomponentenanalyse gesehen, die Länge der Halbachsen wird über die 5 %- und 95 %-Quantile für die Petal.width und Petal.length bestimmt:

```
> ggplot(iris, aes(x = Petal.Width,
+                  y = Petal.Length,
+                  color = Species, shape = Species)) +
+   geom_point(size = 2) +
+   geom_jitter() +
+   theme_gray() +
+   stat_ellipse(geom = "polygon", aes(fill = Species),
+                type = "norm",
+                alpha = 0.2,
+                show.legend = FALSE,
+                level = 0.90)
```

Das Package `e1071` enthält verschiedene Funktionen, darunter auch die Unterstützung von Support Vector Machines. Diese wollen wir für diese Klassifikation einsetzen.

Während die bisherigen Verfahren von einem statistischen Modell (z. B. einer Normalverteilung) ausgingen, setzen Support Vector Machines keines dieser Modelle voraus. Deswegen kann man die Güte auch nicht über die sonst üblichen Kennzahlen abschätzen. Stattdessen werden die Daten in Trainings- und Testdaten unterteilt. Wir verwenden dafür pseudo-zufällige Zahlen (mittels `set-seed()`) in einem Verhältnis von 7/3. Hierbei achten wir darauf, dass die Anteile der verschiedenen Iris-Typen ausgewogen sind:

```
> set.seed(9)
> train <- sample(nrow(iris), 0.7 * nrow(iris))
```

```
> iris_train <- iris[train, ]
> iris_validate <- iris[-train, ]
> table(iris_train$Species)
setosa versicolor  virginica
    37         36         32
> table(iris_validate$Species)
setosa versicolor  virginica
    13         14         18
```

Über `svm()` rufen wir die Funktion auf und lassen uns mittels `predict()` die Werte für die Testdaten anzeigen:

```
> fit_svm <- svm(Species ~ ., data = iris_train)
> fit_svm
Call:
svm(formula = Species ~ ., data = iris_train)
Parameters:
SVM-Type:  C-classification
SVM-Kernel:  radial
cost:  1
Number of Support Vectors:  45

> svm_pred <- predict(fit_svm, na.omit(iris_validate))
> svm_perf <- table(na.omit(iris_validate)$Species,
+                   svm_pred, dnn = c("Actual", "Predicted"))
> svm_perf
             Predicted
Actual       setosa versicolor virginica
setosa         13          0         0
versicolor      0         13         1
virginica       0          1        17
```

Bei der Vorhersage für die Testdaten sind lediglich zwei Fehlklassifikationen zu verzeichnen, was einer Fehlklassifikationsrate von 4.4% entspricht.

6.4 Entscheidungsbaumverfahren

6.4.1 Entscheidungsbäume

Entscheidungsbäume (Decision Trees) sind geordnete und gerichtete Bäume. Darin werden Entscheidungsregeln abgelegt. Sie bestehen aus einem Wurzelknoten (oben) und beliebig vielen inneren Knoten (Rechtecke in Bild 6.11). An jedem dieser Knoten kommt man durch eine Entscheidung über eine Kante zu einem Folgeknoten. Ist an einem Knoten keine Entscheidung mehr zu treffen, befindet man sich an einem Blatt (Oval in Bild 6.11), welches das Ergebnis beinhaltet.

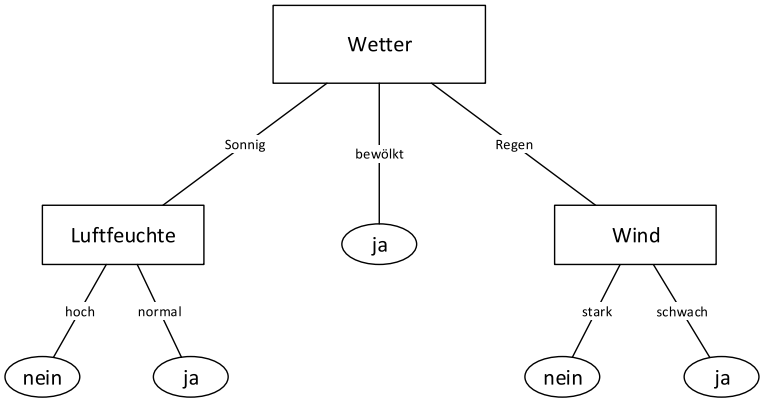

Bild 6.11 Entscheidungsbaum

Will man für einen Datensatz einen Entscheidungsbaum aufbauen, werden für die einzelnen Prädiktorvariablen jeweils getrennte Partitionen erstellt. Der Einfachheit halber teilen wir hier jeweils in zwei Bereiche auf (binärer Split). Diese Stelle kann man beispielsweise finden, indem man vom Mittelwert der Ergebnisvariablen ausgeht und alle Prädiktorvariablen auswählt, deren Ergebnisse in eine erste Hälfte fallen. Derjenige Prädiktor wird für eine Aufteilung herangezogen, bei dem der Fehler nach einer erfolgten Aufteilung minimal wird. So wird rekursiv der Baum aufgebaut. Da dieses Aufspalten wegen des exponentiellen Wachstums sehr schnell zu sehr großen Bäumen führt, schneidet man jedoch die Stellen zurück (Cost complexity pruning).

Entscheidungsbäume können auch für numerische Prädiktorvariablen erzeugt werden.

Die Vorteile von Entscheidungsbäumen sind [JWHT17]:

- Bäume können sehr einfach erklärt werden (einfacher als die lineare Regression).
- Oft hat man den Eindruck, dass Entscheidungsbäume die menschliche Entscheidungsfindung besser widerspiegeln als andere Regressions- und Klassifikationsansätze.
- Bäume können grafisch dargestellt werden und sind auch von einem Laien leicht zu interpretieren, insbesondere wenn sie klein sind.
- Bäume können problemlos qualitative Prädiktoren verarbeiten, ohne dass man Dummy-Variablen erstellen muss.

Diesen Vorteilen stehen die Nachteile gegenüber:

- Bäume haben im Allgemeinen nicht die gleiche Vorhersagegenauigkeit wie andere Regressions- und Klassifikationsansätze.
- Bei großen Strukturen besteht die Gefahr des „Overfitting“.
- Bäume sind oft nicht robust, es entstehen zu viele fragile Regeln: Eine kleine Änderung der Daten kann eine große Änderung des endgültigen geschätzten Baums bewirken.

Die Vorhersagegenauigkeit kann durch Methoden wie Random Forests oder Boosting (Unterabschnitt 6.4.4) verbessert werden.

6.4.2 Der Iris-Datensatz

Im Package `party` können Entscheidungsbäume zur Klassifikation aufgebaut und ausgewertet werden. Wir sehen uns dies wieder am Iris-Datensatz an (mit den gleichen Trainings- und Testdaten wie bei den SVM):

```
> iris_ctree = ctree(Species ~ ., data = iris_train)
> iris_ctree

Model formula:
Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width

Fitted party:
[1] root
|   [2] Petal.Length <= 1.9: setosa (n = 37, err = 0.0%)
|   [3] Petal.Length > 1.9
|   |   [4] Petal.Width <= 1.6
|   |   |   [5] Petal.Length <= 4.8: versicolor (n = 32, err = 0.0%)
|   |   |   [6] Petal.Length > 4.8: virginica (n = 7, err = 42.9%)
|   |   [7] Petal.Width > 1.6: virginica (n = 29, err = 3.4%)

Number of inner nodes:    3
Number of terminal nodes: 4
```

Dieses Verfahren weist nur eine Fehlklassifikation auf:

```
> ctree_pred <- predict(iris_ctree, iris_validate, type = "response")
> ctree_perf <- table(iris_validate$Species, ctree_pred,
+                     dnn = c("Actual", "Predicted"))
> ctree_perf
            Predicted
Actual       setosa versicolor virginica
setosa         13          0         0
versicolor      0         13         1
virginica       0          0        18
```

Mithilfe von Autoplot erhält man eine Übersicht über den erzeugten Baum (Bild 6.12).

```
> autoplot(iris_ctree, main = "Entscheidungsbaum", type = "simple")
```

6.4.3 Bagging

Ein Hauptnachteil von Entscheidungsbäumen ist die große Varianz. Man kann die Varianz von unabhängigen Beobachtungen jedoch dadurch reduzieren, dass man mehrere Beobachtungen macht, denn die Varianz des Mittelwerts von n Beobachtungen mit einer Varianz σ^2 beträgt σ^2/n. Man kann also die Varianz dadurch verkleinern, dass man Mittelwerte von Teilmengen einer Beobachtung bildet.

Man nimmt dazu zufällig ausgewählte Untermengen der Trainingsdaten und baut Entscheidungsbäume auf, die unabhängig voneinander sind. Bei der Vorhersage wird dann der Mittelwert der einzelnen Ergebnisse gebildet. Dies nennt man Bagging.

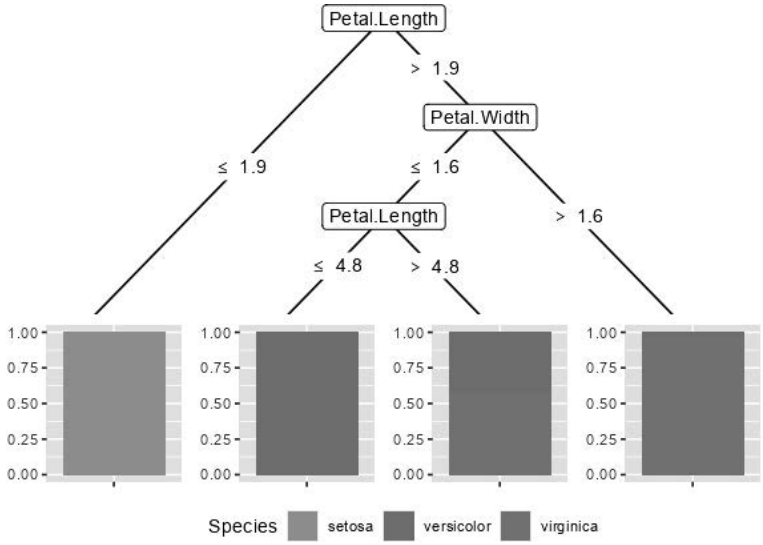

Bild 6.12 Entscheidungsbaum

6.4.4 Random Forests

Entsprechend eines Vorschlags von Breiman [Bre01] werden bei Random Forests – ähnlich wie beim Bagging – Teilmengen zum Aufbau von Bäumen verwendet [LW01].

Während bei Standardentscheidungsbäumen an jedem Knoten die beste Aufteilung unter allen Variablen vorgenommen wird, wird bei Random Forests die beste Aufteilung einer zufällig ausgewählten Teilmenge aller Prädiktorvariablen vorgenommen. Sind jetzt für eine Eingabe alle Bäume ausgewertet, wird das Ergebnis verwendet, welches bei den meisten Bäumen ermittelt wurde.

In *https://www.r-bloggers.com/2017/01/random-forest-classification-of-mushrooms/* wurde diese Methode zur Erkennung essbarer/giftiger Pilze angewandt. Die Korrekt-Klassifikationsrate war erstaunlich gut. Der Autor rät jedoch von der Anwendung dieses Modells ab, weil er keine Aussage über die Zuverlässigkeit der Daten machen konnte.

Wir werden diese Methode an einem anderen Beispiel in Abschnitt 9.4 in Aktion sehen.

6.4.5 Boosted Regression Trees

Boosted Regression Trees kombinieren Bagging und Boosting-Methoden [JWHT17]: Wie bei Bagging werden mehrere Entscheidungsbäume aufgebaut als Wald (forest). Dies erfolgt jedoch im Gegensatz zu Random Forest sequentiell. Hierbei werden die Daten und die Ergebnisse vorhergehender Bäume modifiziert: Man gewichtet die Daten, welche große Fehler verursachen, und verwendet die Gewichte, um bei der Bildung eines nachfolgenden Datensatzes diese Daten mit größerer Wahrscheinlichkeit auszuwählen. Dadurch wird versucht, die Genauigkeit der Bäume sequenziell zu vergrößern.

6.5 Naive Bayes-Klassifikatoren

Bayes-Klassifikatoren bestimmen aufgrund bedingter Wahrscheinlichkeiten ein Ergebnis. Man kann sie unterteilen in multinomiale und Gauß'sche Klassifikatoren.

6.5.1 Multinomiale naive Bayes-Klassifikatoren

Beim Bayes-Klassifikator nutzt man die Eigenschaften bedingter Wahrscheinlichkeiten aus. Die Eigenschaften bedingter Wahrscheinlichkeiten kann man sich sowohl bei kategorialen als auch bei kontinuierlichen Variablen zunutze machen. Bei kategorialen Variablen bedient man sich des Urnenmodells, woraus sich die Binomailverteilung (bzw. bei mehreren Ergebnissen eine multinomiale Verteilung) ergibt. Daher hat dieser Klassifikator seinen Namen.

Zur Behandlung der bedingten Wahrscheinlichkeiten betrachten wir das Vorkommen bestimmter Worte in einer Nachricht. Anhand der Auftrittswahrscheinlichkeiten der Worte soll abgeschätzt werden, ob es sich bei einer Nachricht um eine Spam-Nachricht handelt.

Angenommen, wir haben im Lauf der Zeit 50 spam-Nachrichten (S) und 150 Nicht-spam („ham")-Nachrichten (H) erhalten. In den Nachrichten wurde das Auftreten der beiden Worte „gewonnen" (g) und „Konto" (k) (jeweils in spam und in ham) gezählt (Tabelle 6.1). Dann können wir daraus eine a-priori-Wahrscheinlichkeit von $p(S) = \frac{50}{50+150} = 0{,}25$ für spam und $p(H) = 1 - 0{,}25 = 0{,}75$ für ham abschätzen.

Wir wollen das Auftreten von „gewonnen" (g) und „Konto" (k) heranziehen, um zu entscheiden, ob es sich um eine ham- oder spam-Nachricht handelt.

Tabelle 6.1 Auftreten von Worten in spam und ham

	spam		ham	
Auftreten	n	p(|S)	n	p(|H)
insgesamt	50		150	
g	40		30	
k	25		15	

Die Wahrscheinlichkeit für das Auftreten von g in einer Nachricht beträgt

$$p(g) = \frac{40+30}{50+150} = \frac{7}{20}.$$

Jetzt betrachten wir die Nachrichten unter der Bedingung, dass die Nachricht g zutrifft: Wie groß ist dann die Wahrscheinlichkeit, dass es sich um spam handelt? Dies wird ausgedrückt als $p(S|g)$ (die bedingte Wahrscheinlichkeit für S unter der Bedingung g). Wir können dies einfach aus den Zahlen in der Zeile g ausrechnen:

$$p(S|g) = \frac{\text{Zahl der günstigen Fälle}}{\text{Zahl der möglichen Fälle}} = \frac{40}{40+30} = \frac{4}{7}$$

Entsprechend ist

$$P(H|g) = \frac{30}{40+30} = \frac{3}{7}$$

Vertauschen wir jetzt Bedingung (g) und Variable (S), dann ist die Wahrscheinlichkeit für g unter der Bedingung S

$$p(g|S) = \frac{40}{50} = 0{,}8$$

Bei ham beträgt sie

$$p(g|H) = \frac{30}{150} = 0{,}2$$

Bei diesem Zahlenbeispiel haben wir bei der Bestimmung der Wahrscheinlichkeiten die Bedingungen ausgetauscht. Dies ist die Aussage des Satzes von Bayes:

$$p(A|B) = \frac{p(B|A) \cdot p(A)}{p(B)}.$$

Damit können wir analog $p(g|S)$ auch bestimmen über

$$p(g|S) = \frac{p(S|g) \cdot p(g)}{p(S)} = \frac{\frac{4}{7} \cdot \frac{7}{20}}{0{,}25} = 0{,}8.$$

Entsprechend kann man die Wahrscheinlichkeiten für k abschätzen:

$$p(k|S) = \frac{25}{50} = 0{,}5 \qquad \text{und} \qquad p(k|H) = \tfrac{15}{150} = 0{,}1$$

Nun nehmen wir an, dass die Texte über keinerlei Grammatik (keine Reihenfolge etc.) verfügen und die Worte statistisch unabhängig voneinander auftreten. Wir sprechen dann von einem naiven Klassifikator.

Wie wahrscheinlich ist es, dass g und h in den Nachrichten zusammen vorkommen? Hier gilt

$$p(g \cap k|S) = p(g|S) \cdot p(k|S) \qquad \text{und} \qquad p(g \cap k|H) = p(g|H) \cdot p(k|H)$$

für den Fall unabhängiger Ereignisse (naives Modell).

Die erwartete Anzahl der spam-Nachrichten, die sowohl g als auch k aufweisen, können wir damit einfach über Multiplikation von $p(g \cap k|S)$ mit der Zahl n der spam-Nachrichten erhalten (Tabelle 6.2).

Tabelle 6.2 Wahrscheinlichkeiten von Worten in spam und ham

	spam		ham	
Auftreten	n	p(|S)	n	p(|H)
insgesamt	50		150	
g	40	0,8	30	0,2
k	25	0,5	15	0,1
$g \cap k$	20	0,4	1	0,02

Wenn wir jetzt entscheiden wollen, ob eine Nachricht H oder S ist, müssen wir $p(S|g \cap k) = \frac{20}{20+1} \approx 0{,}95$ in das Verhältnis setzen zu $p(H|g \cap k) = \frac{1}{20+1} \approx 0{,}05$.

Aufgrund dieses Verhältnisses können wir von einer spam-Nachricht ausgehen.

In Formeln und unter Verwendung des Satzes von Bayes können wir angeben:

$$p(S|g \cap k) = \frac{p(g \cap k|S) \cdot p(S)}{p(g \cap k|S) \cdot p(S) + p(g \cap k|H) \cdot p(H)},$$

ausmultipliziert:

$$p(S|g \cap k) = \frac{p(g|S) \cdot p(k|S) \cdot p(S)}{p(gS) \cdot p(|S) \cdot p(S) + p(g|H) \cdot p(k|H) \cdot p(H)}.$$

Diese Formel kann man auf weitere Variablen erweitern. Kommen also bestimmte Worte in einer Nachricht vor (bzw. kommen nicht vor), kann man über den Satz von Bayes die bedingten Wahrscheinlichkeiten für Ham oder Spam ausrechnen und anhand des Verhältnisses über diese Frage entscheiden.

Es kann nun sein, dass die bedingte Wahrscheinlichkeit für eine der Variablen Null ergeben hat. Dann würden wir in beiden Fällen das Produkt 0 erhalten. Um dem vorzubeugen, wird bei allen Variablen ein sehr kleiner Wert hinzuaddiert. Dies verursacht zwar Fehler in der Berechnung, ermöglicht aber verwertbare Ergebnisse.

6.5.2 Das spam-Beispiel

Eine Studie hat bei der Erkennung von spam bei SMS-Nachrichten mithilfe des Bayes-Klassifikators viel Beachtung gefunden [HHSD98]. Hier wurden 5574 SMS-Nachrichten klassifiziert als ham bzw. als spam. Der Datensatz kann von *https://www.kaggle.com/hdza1991/sms-spam* abgerufen werden.

```
> sms_raw <- read.csv("sms_spam.csv",
+                     stringsAsFactors = FALSE)
> sms_raw$type <- factor(sms_raw$type)
> table(sms_raw$type)
 ham spam
4827  747
```

Der erste Schritt besteht darin, die Texte in eine Form zu bringen, dass sie für den Klassifikator verwendbar sind. Hierzu laden wir den gesamten Text zur besseren Verwaltung in den Hauptspeicher:

```
> sms_corpus <- VCorpus(VectorSource(sms_raw$text))
```

An diesem Datensatz werden nun Bereinigungen vorgenommen wie

- Leerzeichen entfernen,
- Zahlen entfernen,
- Stoppwörter (z. B. wie, auch, und ...) entfernen,
- alle Satzzeichen entfernen,
- Bildung von Wortstämmen mit dem Stemmer-Algorithmus [CKLN10].

Danach wird eine Matrix aufgebaut, in der die Dokumentennummern als Zeilen und die einzelnen (bereinigten) Wörter als Spalten abgelegt werden. Die Zahlen in der Matrix geben an, wie oft der Begriff in jedem Dokument vorkommt. All diese Schritte werden von der Funktion `DocumentTermMatrix` vorgenommen:

```
> sms_dtm <- DocumentTermMatrix(sms_corpus, control = list(
+                     tolower = TRUE,
+                     removeNumber = TRUE,
+                     stopwords = TRUE,
+                     removePunctuation = TRUE,
+                     stemming = TRUE,
+                     stripWhitespace=TRUE))
```

Wenn Sie mehr Details über die Textvorbereitung erfahren möchten, bietet Ihnen neben der package-Dokumentation zu `tm` auch *https://confessionsofadatascientist.com/text-analysis-with-r-repro-report.html* sowie *https://github.com/solariz/german_stopwords* weitergehende Informationen.

Nach der Verarbeitung der Texte wird der Datensatz unterteilt in Trainings- und Testdaten sowie die zugehörigen Labels. Über `prop.table()` prüfen wir nach, ob das Verhältnis von ham und spam zwischen den Trainings- und Testdaten ausgewogen ist:

```
> sms_dtm_train <- sms_dtm[1:4169, ] # Training dataset
> sms_dtm_test  <- sms_dtm[4170:5559, ] #Test dataset
> sms_train_labels <- sms_raw[1:4169, ]$type #Training Label
> sms_test_labels  <- sms_raw[4170:5559, ]$type # Test Label
> prop.table(table(sms_train_labels))
sms_train_labels
ham      spam
0.8647158 0.1352842
> prop.table(table(sms_test_labels))
sms_test_labels
ham      spam
0.8697842 0.1302158
```

Unsere Matrix hat 7935 Spalten, von denen etliche nur wenige oder gar keine Einträge aufweisen. Um den Rechenaufwand zu verkleinern, entfernen wir mittels `findFreqTerms()` alle Spalten, in denen ein Wort nicht mindestens fünfmal vorkommt, und reduzieren so die Matrix auf 1234 Spalten:

```
> sms_freq_words <- findFreqTerms(sms_dtm_train, 5)
> sms_dtm_freq_train <- sms_dtm_train[ , sms_freq_words]
> sms_dtm_freq_test <- sms_dtm_test[ , sms_freq_words]


> inspect(sms_dtm_freq_train)
<<DocumentTermMatrix (documents: 4169, terms: 1234)>>
Non-/sparse entries: 25986/5118560
Sparsity           : 99%
Maximal term length: 15
Weighting          : term frequency (tf)
Sample             :
```

```
Terms
Docs   call can come free get just know ltgt now will
1086    0   0    1    0   1    0    0    0   0    9
1580    0   0    0    0   0    0    0   18   0    0
:     :    :     :    :   :    :
```

Bei der Betrachtung des Bayes-Theorems hatten wir für eine Variable nur eine Ja/Nein-Entscheidung für das Auftreten in einer Nachricht betrachtet, nicht aber die Häufigkeit des Auftretens. Deswegen wird jetzt mithilfe einer selbst definierten Funktion `convert_counts()` Felder mit Zahlen größer als Null auf „Yes“ gesetzt, der Rest auf „No“:

```
> convert_counts <- function(x) {
        +   x <- ifelse(x > 0, "Yes", "No")
        + }
> sms_train <- apply(sms_dtm_freq_train, MARGIN = 2, convert_counts)
> sms_test  <- apply(sms_dtm_freq_test, MARGIN = 2, convert_counts)
```

Damit sind wir vorbereitet für den Klassifikator `naiveBayes` und können die Ergebnisse über `CrossTable` betrachten:

```
> sms_classifier <- naiveBayes(sms_train, sms_train_labels)
> sms_test_pred <- predict(sms_classifier, sms_test)
> head(sms_test_pred)
[1] spam ham  ham  ham  ham  ham
Levels: ham spam
> CrossTable(sms_test_pred, sms_test_labels,
+            prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
+            dnn = c('predicted', 'actual'))
Cell Contents
|-------------------------|
|                       N |
|           N / Col Total |
|-------------------------|
Total Observations in Table:  1390

              | actual
   predicted |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |      1204 |        28 |      1232 |
             |     0.996 |     0.155 |           |
-------------|-----------|-----------|-----------|
        spam |         5 |       153 |       158 |
             |     0.004 |     0.845 |           |
-------------|-----------|-----------|-----------|
Column Total |      1209 |       181 |      1390 |
             |     0.870 |     0.130 |           |
-------------|-----------|-----------|-----------|
```

Wir haben mit dieser Methode eine respektable Korrekt-Klassifikationsrate von $\frac{25+5}{1390} \approx 98\%$ erreicht.

Im Zusammenhang mit der Verarbeitung von Texten ist auch das Package `wordcloud` geschrieben worden, mit dem Worte nach Häufigkeit in einer Wolke angeordnet werden (Bild 6.13).

```
> library(wordcloud)
> library(RColorBrewer)
> wordcloud(sms_raw$text, max.words = 50,
+           scale = c(3, 0.5),
+          colors=brewer.pal(6, "Paired"))
```

Bild 6.13 wordcloud

6.5.3 Likelihood

Wir haben in Unterabschnitt 6.5.1 Wahrscheinlichkeiten abgeschätzt über

$$p = \frac{\text{Zahl der günstigen Fälle}}{\text{Zahl der möglichen Fälle}}.$$

Diese Wahrscheinlichkeiten kann man auch über einen anderen Weg erhalten: Hierzu stellen wir uns eine Urne vor, in der eine unbekannte Zahl von roten und weißen Kugeln enthalten sind. Aus dieser Urne ziehen wir 20 Kugeln und erhalten 4 rote und 16 weiße Kugeln. Wenn wir jetzt verschiedene Werte π des Verhältnisses von roten zu weißen Kugeln annehmen, können wir jeweils die Wahrscheinlichkeit bestimmen, von den 20 Stichproben $0, 1, 2 \ldots 20$ rote Kugeln zu ziehen.

In Bild 6.14 sind die Wahrscheinlichkeiten für verschiedene Werte von π gezeigt. Der Fall der 4 roten Kugeln ist im Diagramm rot eingezeichnet. Wie man erwartet, ist das Maximum bei einem Wert von $\pi = 0,2$.

Die Wahrscheinlichkeit für $k = 4$ kann man so bestimmen: Wenn man viermal eine rote Kugel gezogen hat, entspricht das einer Wahrscheinlichkeit von π^4. Das 16-malige Ziehen einer weißen Kugel entspricht einer Wahrscheinlichkeit von $(1-\pi)^{16}$. Jetzt kann man diese gezogenen Kugeln in $\binom{4}{20}$ möglichen Kombinationen anordnen. Dies ergibt eine Wahrscheinlichkeit von

$$L_{20}(\pi) = \binom{4}{20} \cdot \pi^4 \cdot (1-\pi)^{16}$$

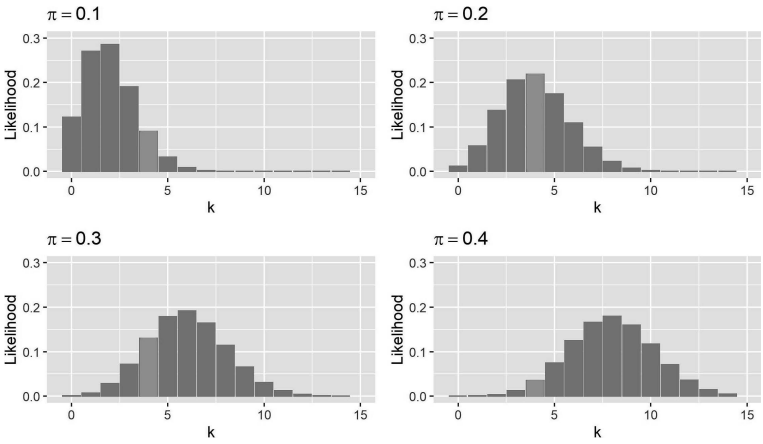

Bild 6.14 Wahrscheinlichkeiten/Likelihoods für verschiedene π

oder allgemeiner

$$L_k(\pi) = \binom{n}{k} \cdot \pi^k \cdot (1-\pi)^{n-k}.$$

Dies ist nichts anderes als die Binomialverteilung. „Normalerweise“ hält man π und n fest und variiert k, woraus sich die Wahrscheinlichkeit ergibt, ein bestimmtes k zu erhalten.

Man kann aber auch k festhalten und π variieren. In diesem Fall wird die Wahrscheinlichkeit Likelihood genannt. Der Name ergibt sich also aus der Frage, welches die abhängige Variable ist.

Im Fall kontinuierlicher Variablen sieht es anders aus. Betrachten wir noch einmal das Beispiel der Normalverteilung (Unterabschnitt A.2.1): Dort ist die Wahrscheinlichkeit, dass sich eine normalverteilte Variable innerhalb des Bereichs $0 \leq z \leq 0.674$ befindet, die Fläche unter der Häufigkeitsdichte (Bild 6.15 links). Die Likelihood für $z = 0.674$ ist bei kontinuierlichen Verteilungen die Wahrscheinlichkeitsdichte (der Y-Wert des Punkts bei $z = 0.674$ in Bild 6.15 rechts).

6.5.4 Gauß’sche naive Bayes-Klassifizierer

Der Gauß’sche naive Bayes-Klassifizierer wendet das Bayes-Theorem auf Variablen an, von denen angenommen wird, dass sie einer Gauß-Verteilung folgen.

Hierzu ein einfaches Beispiel:

Wir betrachten zwei Klassen von Fahrzeugen, Coupés und SUV, von denen wir den Mittelwert und die Standardabweichung der Massen m und Höchstgeschwindigkeiten v kennen (Tabelle 6.3). Es gibt dreimal so viele Coupés wie SUV. Nun soll für ein unbekanntes Fahrzeug der Masse 1.2 to und der Höchstgeschwindigkeit 210 km/h der Typ bestimmt werden. Hierzu schätzen wir aus den Anteilen der Typen die a-priori-Wahrscheinlichkeiten $p = \frac{3}{4}$ bzw. $p = \frac{1}{4}$. Aus der Wahrscheinlichkeitsdichtefunktion der Gauß-Verteilung bestimmen

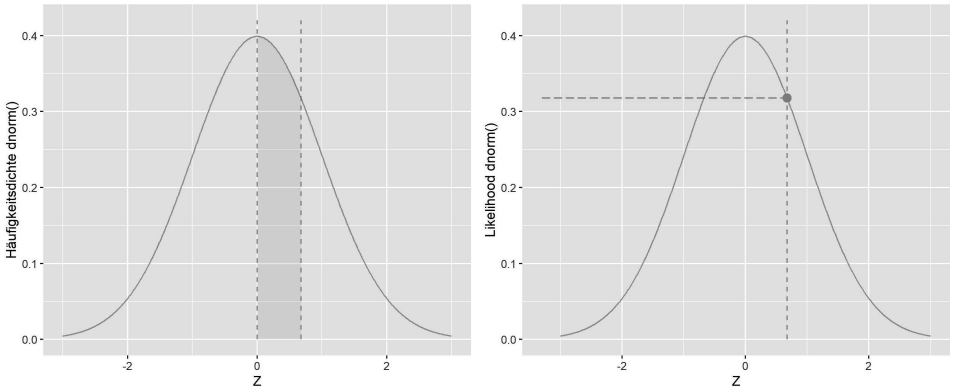

Bild 6.15 Wahrscheinlichkeit und Likelihood

Tabelle 6.3 Beschreibende Größen von Coupés und SUV

Typ	Masse *m*		Höchstgeschwindigkeit *v*	
	μ [to]	σ [to]	μ [km/h]	σ [km/h]
Coupé	1	0.2	220	10
SUV	2	0.4	180	20

wir zunächst mit `dnorm()` die Likelihoods für die Massen und Höchstgeschwindigkeiten für beide Fahrzeugtypen (Bild 6.16).

```
> l_mass_coupe <- dnorm(1.2, mean = 1, sd = 0.2)
> l_mass_coupe
[1] 1.209854
  :     :
> l_speed_suv <- dnorm(210, mean = 180, sd = 20)
> l_speed_suv
[1] 0.00647588
```

Wir nehmen nun an, dass die Likelihoods unabhängig voneinander sind. Dann könnten wir einfach das Produkt der Likelihoods für beide Fahrzeugtypen berechnen. Da aber oft sehr kleine Werte vorkommen und Computer sehr kleine Zahlen als Nullen behandeln, kann dies zu Fehlern führen. Um dieses Problem zu umgehen, bildet man den Logarithmus der Likelihoods (Log-Likelihoods). Damit kann man anstelle der Multiplikation durch Nutzung der Logarithmenregeln einfach die Log-Likelihoods addieren. Da die Likelihoods Werte zwischen 0 und 1 annehmen, liegen Log-Likelihoods zwischen $-\infty$ und 0:

```
> log(3 / 4) + log(l_mass_coupe) + log(l_speed_coupe)
[1] -3.818706
> log(1 / 4) + log(l_mass_suv) + log(l_speed_suv)
[1] -8.428613
```

Man stellt fest, dass die Log-Likelihood in diesem Falle für das Coupé größer ist als die des SUVs. Wir schließen deshalb auf ein Coupé.

Diese Leseprobe haben Sie beim edv-buchversand.de heruntergeladen.
Das Buch können Sie online in unserem Shop bestellen.

Hier zum Shop