1 Begriffe

Siehe extra Tabelle (füge ich vllt irgendwann noch ein)
Bis Overfitting hat Ln(a) die Zusammenfassung geschrieben. (Danke dafür) Meine Version ist hier : Link.

2 Definitionen

2.1 Specification of learning tasks

Realworld $\rightarrow$ Modelworld

1. **Reale Welt:**

- O - Menge an Objekten
- C - Menge an Klassen
- $\gamma : O \rightarrow C$ - idealer Classifier für O

Aufgabe-Klassifizierung:

- bestimme von $o \in O$ die Klasse $\gamma(o) \in C$

2. **Model Welt**

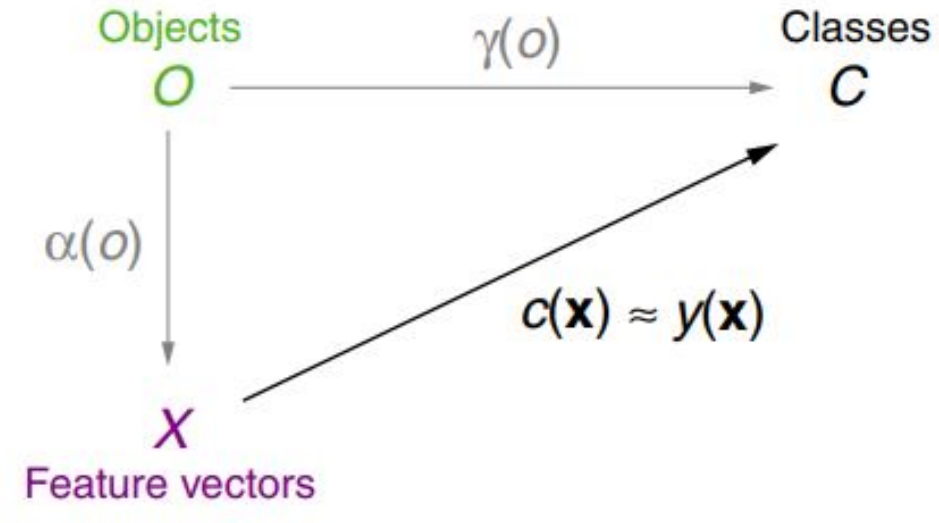
- X - Menge von feature Vektoren
- C - Menge von Klassen
- $c : X \rightarrow C$ - idealer Classifier für X *(c ist unbekannt)*
- $D = \{(\mathbf{x}_1, c(\mathbf{x}_1)), \ \ldots \ , (\mathbf{x}_n, c(\mathbf{x}_n))\}$ - Menge von Beispielen *(bereits klassifiziert)*

Todo: Schätze $c(\mathbf{x})$, welche implizit durch D gegeben sind, durch Model-Funktion $y(\mathbf{x})$

2.2 Machine Learning:

1. Collect real-world examples of the form $(o, \gamma(o)), o \in O$
2. abstract the objects towards feature vectors $\mathbf{x} \in X$, where $\mathbf{x} = \alpha(0)$
3. Formuliere Model-Function: $y : X \rightarrow C, \mathbf{x} \mapsto y(\mathbf{x})$

4. Nutze Statistik, Theorie und Algorithmen aus ML um den fit zwischen $c(\mathbf{x})$ und $y(\mathbf{x})$ zu Maximieren, sodass $y(\mathbf{x}) \approx c(\mathbf{x}) \approx \gamma(o)$



2.3 Supervised learning

Eine Funktion mithilfe von input-output-Daten lernen;
automatisierte Klassifikation mit von Menschen bereits klassifizierten Daten als Grundlage
Beispiel: optical character recognizition

2.4 Unsupervised learning

identifiziert/findet selbstständig Muster und Strukturen in Daten;

- automatisierte Kategorisierung durch Cluster Analysis
- Parameter Optimierung durch Expectation Maximation
- Feature Extrahierung durch Factor Analysis

Beispiel: intrusion detection in a network data stream

2.5 Reinforcement learning

"Learn, adapt, or optimize a behavior strategy in order to maximize the ownbenefit by interpreting feedback that is provided by the environment."
Beispiel: program to play tetris

2.6 Feature Vektor

Ein Feature Vektor ist ein Vektor in dem jede Dimension eine Eigenschaft(Feature) des beschriebenen Objektes enthält.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} \quad \text{Besipiel: } \mathbf{Gitarre} = \begin{bmatrix} Farbe : blau \\ Baujahr : 1997 \\ Material : Holz \\ Elektrisch : ja \end{bmatrix}$$

2.7 Ground Truth

Überprüfung der Klassifizierung eines Lernprozesses auf Richtigkeit für gewolltes Model.
Beispiel: Überprüfung eines Spamfilter nach falsch kategorisierten Mails

2.8 Classification Approaches

2.8.1 generative approches

exploit the distributions underlying the classes → discriminative: classsification rule
essbar nicht essbar

2.8.2 discriminative approaches

learn a boundary between classes → generative: class membership probability
Mietpreis anhand von Lage, Baujahr etc.

3 Machine Learning Basics

3.1 Lineare Regression

3.1.1 One Dimensional Feature Space

Grundformel für lineare Gerade, mit:
$y(x)$ - abhängige Variabel x - unabhängige Variable w_1 - Anstieg der Geraden w_0 - Schnittpunkt der y-Achse

$$y(x) = w_o + w_1 \cdot x \tag{1}$$

Wobei das minimale w_0 und w_1 sich ergeben aus:

$$w_1 = \frac{\sum_{i=1}^{n}(x_i - \overline{x} \cdot (y_i - \overline{y}))}{\sum_{i=1}^{n}(x_i - \overline{x}^2)} \tag{2}$$

$$w_0 = \overline{y} - w_1 \cdot \overline{x} \tag{3}$$

3.1.2 Higher Dimensional Feature Space

$$y(\mathbf{x}) = w_0 + \sum_{j=1}^{p}(w_j \cdot x_j)$$

oder Vektor notation with $x_0 = 1$ und $\mathbf{w} = (w_0, w_1, ..., w_p)^T : y(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

Goodness of Modelfit, Regressionserror Residual sum of Squares(RSS).
Der Residue wird aus der Differenz zwischen (beobachteten) Realwelt-Wert y_i und geschätzten/modelierten Wert $y(\mathbf{x}_i)$ berechnet

$$RSS(\mathbf{w}) = \sum_{i=1}^{n}(y_i - y(\mathbf{x}_i))^2 \tag{4}$$

RSS($\mathbf{w}$) minimieren:

$$\hat{w} \equiv w = (X^T X)^{-1} X^T \mathbf{y}$$

$$\hat{y} = X\hat{\boldsymbol{w}}$$

3.2 LMS: Least Mean Squared

Ziel: Fitting $y(x)$; Anpassung der weights, sodass Klassifizierungsfehler möglichst gering sind.

Input: D - Trainingsdaten $(\mathbf{x}, c(\mathbf{x}))$ mit $\mathbf{x} \in \mathbf{R}^p$ und Zielklasse c($\mathbf{x}$) η Learning rate, kleine positive Konstante

IGD(D, η)

```
1.  initialize_random_weights(w),  t = 0
2.  REPEAT
3.    t = t + 1
4.    FOREACH (x, c(x)) ∈ D DO
5.      δ = c(x) − y(x)     // y(x) =(*) w^T x,  δ ∈ R.
6.      Δw =(*) η · δ · x    // δ · x is the derivative of ℓ2(c(x), y(x)).
7.      w = w + Δw
8.    ENDDO
9.  UNTIL(convergence(c(D), y(D)) OR t > t_max)
10. return(w)
```

Higher-Dimensional Feature Space ML:ll-16

3.3 Concept Learning

Setting:
X - Menge an Feature Vektoren
C - Ist eine Menge mit zwei Klassen *Beipiel:* {*0,1*},{*ja, nein*}
$c : X \to C$ -idealer Klssifizierer für X
$D = \{(\mathbf{x}_1, c(\mathbf{x}_1)), \ldots, (\mathbf{x}_n, c(\mathbf{x}_n))\} \subseteq X \times C$ - Menge mit Beispielen
$h(\mathbf{x})$ hypothesis - besteht aus literalen, wildcards (?) und contradiction ($\perp$)
Todo:
Schätze $c(x)$, was implizit durch D mit feature-Value-Muster
maximally specific hypothesis: $s_0 = \langle \perp, \perp, \perp, \perp \rangle \equiv 0 \Rightarrow \nexists \mathbf{x} \in X : s_0(\mathbf{x})$
maximally general hypothesis: $g_0 = \langle ?, ?, ?, ? \rangle \equiv 1 \Rightarrow \forall \mathbf{x} \in X : g_0(\mathbf{x})$
$h_1(\mathbf{x})$ is more general then $h_2(\mathbf{x})$ ($h_1(\mathbf{x}) \geq_g h_2(\mathbf{x})$) iff:

$$\forall \mathbf{x} \in X : h_2(\mathbf{x}) = 1 \text{implies} h_2(\mathbf{x}) = 1$$

strictly more general ($>_g$): $h_1(\mathbf{x}) \geq_g h_2(\mathbf{x})$ und $h_2(\mathbf{x}) \not\geq_g h_1(\mathbf{x})$ more specific: the same but with $\leq_s$

3.3.1 Find-S

Fang mit $\langle \perp, \perp, \perp, \perp \rangle$ an und gehe dann alle positiven Beispiele durch ersetze $\perp$ durch konkreten Wert und konkrete Werte durch ? bis
$\forall \mathbf{x} \in X : h(\mathbf{x}) = 1$

1. $h(\mathbf{x}) = s_0(\mathbf{x})$ // $h(\mathbf{x})$ is a maximally specific hypothesis in H.
2. **FOREACH** $(\mathbf{x}, c(\mathbf{x})) \in D$ **DO**
   IF $c(\mathbf{x}) = 1$ **THEN** // Use only positive examples.
      IF $h(\mathbf{x}) = 0$ **DO**
         $h =$ *min_generalization*$(h, \mathbf{x})$ // Relax $h(\mathbf{x})$ wrt. $\mathbf{x}$.
      ENDIF
   ENDIF
ENDDO
3. *return*$(h(\mathbf{x}))$

3.3.2 Candidate Elimination Algorithm

1. Initialization: $G = \{g_0\},\ S = \{s_0\}$
2. If $\mathbf{x}$ is a positive example
   - Remove from G any hypothesis that is not consistent with $\mathbf{x}$
   - For each hypothesis s in S that is not consistent with $\mathbf{x}$
     - Remove s from S
     - Add to S all minimal generalizations h of s such that
       1. h is consistent with $\mathbf{x}$ and
       2. some member of G is more general than h
     - Remove from S any hypothesis that is less specific than another hypothesis in S
3. If $\mathbf{x}$ is a negative example
   - Remove from S any hypothesis that is not consistent with $\mathbf{x}$
   - For each hypothesis g in G that is not consistent with $\mathbf{x}$
     - Remove g from G
     - Add to G all minimal specializations h of g such that
       1. h is consistent with $\mathbf{x}$ and
       2. some member of S is more specific than h
     - Remove from G any hypothesis that is less general than another hypothesis in G

```
1. G = {g_0} // G is the set of maximally general hypothesis in H.
   S = {s_0} // S is the set of maximally specific hypothesis in H.
2. FOREACH (x, c(x)) ∈ D DO
     IF c(x) = 1 THEN // x is a positive example.
       FOREACH g ∈ G DO IF g(x) ≠ 1 THEN G = G \ {g} ENDDO
       FOREACH s ∈ S DO
         IF s(x) ≠ 1 THEN
           S = S \ {s}, S⁺ = min_generalizations(s, x)
           FOREACH s ∈ S⁺ DO IF (∃g ∈ G : g ≥_g s) THEN S = S ∪ {s} ENDDO
           FOREACH s ∈ S DO IF (∃s' ∈ S : s' ≠ s ∧ s ≥_g s') THEN S = S \ {s} ENDDO
       ENDDO
     ELSE // x is a negative example.
       FOREACH s ∈ S DO IF s(x) ≠ 0 THEN S = S \ {s} ENDDO
       FOREACH g ∈ G DO
         IF g(x) ≠ 0 THEN
           G = G \ {g}, G⁻ = min_specializations(g, x)
           FOREACH g ∈ G⁻ DO IF (∃s ∈ S : g ≥_g s) THEN G = G ∪ {g} ENDDO
           FOREACH g ∈ G DO IF (∃g' ∈ G : g' ≠ g ∧ g' ≥_g g) THEN G = G \ {g} ENDDO
       ENDDO
     ENDIF
   ENDDO
3. return(G, S)
```

4 Linear Models

4.1 Logistic Regression

$$y(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{1 + e^{-w^T x}}$$

- wie lineare Regression aber logistisch
- (binäre classification in 0/1 anhand von Beispieldaten D)
- nutzen wir weil noise, sonderfälle, Ausreißer und falsch gelabelte Beispiele in den Trainingsdaten weniger Problematisch sich auswirken auf das Ergebnis
- sigmoid logistisch weil sie "S" shaped ist und die spezielle, weil sie sich nur im Intervall [0,1] befindet → cool für Mathe
- $y(\mathbf{x}) = P(c(\mathbf{x}) = 1 \mid \mathbf{x}; \mathbf{w})$: Wahrscheinlichkeit dass $c(\mathbf{x}) = 1$ given $\mathbf{x}$, parameterized by $\mathbf{w}$
- $y(\mathbf{x}) = 0.67 \Rightarrow$ *67 % chance dass Mail = Spam*

$$\text{Classification: Predict} \begin{cases} c(\mathbf{x}) = 1 & \text{if } \sigma(\mathbf{w}^T\mathbf{x}) \geq 0.5 \\ c(\mathbf{x}) = 0 & \text{if } \sigma(\mathbf{w}^T\mathbf{x}) < 0.5 \end{cases}$$

4.1.1 Loss for Logistic Regression

- 0/1 Loss $L_{0/1}(y(\mathbf{x}), c(\mathbf{x})) = I(\text{sign}(y(\mathbf{x}) - 0.5), c(\mathbf{x}))$
- logistic loss $L_\sigma(y(\mathbf{x}), c(\mathbf{x})) = \begin{cases} -log(y(\mathbf{x})) & \text{if } c(\mathbf{x}) = 1 \\ -log(1 - y(\mathbf{x})) & \text{otherwise} \end{cases}$

4.2 Overfitting

- fitting: der Prozess die Parameter einer Modelfunktion y so anzupassen das sie der der Beispieldaten D am besten passen
- overfiiting: "Fitting the data more than is warranted"
- alias besser passen als berechtigt?
- Gründe:
  - zu komplizierte Modefunktion (zu viele Features)
  - zu wenig Daten in D
  - zu viel Datenrauschen

  - D ist zu biased alias nicht repräsentativ

- Folgen:
  - kleiner Error auf D_{tr} anber großer Error auf D_{test} und IRL
  - loss of inductive Bias
  - increase of variance as a result of sensitivity to noise

- Overfitting finden:
  - Visuell untersuchen für Fälle mit Dimensionen < 3 sonst embedding oder projizieren in kleinere Dimensionen
  - Validieren: wenn $Err_{fit} = Err_{val}(y) - Err_{tr}(y)$ zu groß ist

- Overfitting vermeiden:
  - Early stopping through model selection: nach m schritten überprüfen ob sich Err_{fit} noch verkleinert und stoppen wenn er sich vergrößert
  - Qualität (schlechte Beispiele raus) und / oder Quantität (mehr Daten gleichen Rauschen aus) von D verbessern
  - Manually enforcing a higher bias by using a less complex hypothesis space alias Removing Features: In this approach, irrelevant features are removed from the dataset. This enhances the algorithm's ability to generalize
  - Regularization (WUHU!)

4.2.1 Well- and Ill-posed problems

A mathematical problem is called well-posed if

1. a solution exists,
2. the solution is unique,
3. the solution's behavior changes continuously with the initial conditions.

Otherwise, the problem is called ill-posed.

4.3 Regularization

Automatic adjustment of the loss function to penalize model complexity. Let $L(\mathbf{w}$ denote a loss function used to optimize the parameters $\mathbf{w}$ of a model function $y(\mathbf{x})$. Regularization introduces a trade-off between model complexity and inductive bias:

$$\mathcal{L}(\mathbf{w}) = L(\mathbf{w}) + \lambda * R(\mathbf{w})$$

where $\lambda \geq 0$ controls the impact of the regularization term $R(\mathbf{w}) \geq 0$. $\mathcal{L}$ is called "objective function".

4.3.1 Regularized Linear Regression

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda \cdot \overrightarrow{w}^T\overrightarrow{w}$$

Estimate $\mathbf{w}$ by minimizing the residual sum of squares:

$$\hat{w} = \underset{\mathbf{w}\in\mathbf{R^{P+1}}}{\operatorname{argmin}} \mathcal{L}(\mathbf{w})$$

$$\rightsquigarrow RSS(\mathbf{w}) = (\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda\mathbf{w}^T\mathbf{w}$$

Ableitung bilden um RSS($\mathbf{w}$) zu minimieren und man kommt auf:

$$\mathbf{w} = (X^TX + \text{diag}(0, \lambda, ..., \lambda))^{-1}X^T\mathbf{y}$$

$$\text{diag}(0, \lambda, ..., \lambda) = \begin{bmatrix} 0 & 0 & 0 & ... & 0 \\ 0 & \lambda & 0 & ... & 0 \\ 0 & 0 & \lambda & ... & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & ... & \lambda \end{bmatrix}$$

$$\hat{y}(\mathbf{x}_i) = \hat{\mathbf{w}}^T\mathbf{x}_i$$

Um so höher λ um so einfacher ist die Funktion - Regularization achieved!

5 Neural Networks

5.1 Perception Learning

Idee: Lass mal ein Gehirn programmieren!
Typisches Beispiel: Schrifterkennung

$$y(\mathbf{x}) = 1 \Leftrightarrow (\sum_{j=0}^{p} w_j x_j) \geq 0$$

sonst ist y($\mathbf{x}$) = 0

- wenn $w_0 = -\theta$ und $x_0 = 1$(canonical form)

- sonst $y(\mathbf{x}) = 1 \Leftrightarrow (\sum_{j=1}^{p} w_j x_j - \theta) \geq 0$
- $0 = w_0 + w_1 * x_1 + w_2 * x_2$ für 2D Fälle

5.1.1 PT Algorithm

PT(D, η)

1. *initialize_random_weights*$(\mathbf{w})$, $t = 0$
2. **REPEAT**
3. $t = t + 1$
4. $(\mathbf{x}, c(\mathbf{x})) =$ *random_select*(D)
5. $\delta = c(\mathbf{x}) - y(\mathbf{x})$ // $y(\mathbf{x}) \overset{(*)}{=}$ *heaviside*$(\mathbf{w}^T\mathbf{x}) \in \{0,1\}$, $c(\mathbf{x}) \in \{0,1\} \rightsquigarrow \delta \in \{0,1,-1\}$
6. $\Delta\mathbf{w} \overset{(*)}{=} \eta \cdot \delta \cdot \mathbf{x}$
7. $\mathbf{w} = \mathbf{w} + \Delta\mathbf{w}$
8. **UNTIL**(*convergence*$(c(D), y(D))$ **OR** $t > t_{\text{max}}$)
9. *return*$(\mathbf{w})$

- If a separating hyperplane between X_0 and X_1 exists, the PT algorithm will converge. If no such hyperplane exists, convergence cannot be guaranteed.
- A separating hyperplane can be found in polynomial time with linear programming. The PT Algorithm, however, may require an exponential number of iterations.
- Classification problems with noise are problematic

5.2 Gradient Descent

- Finde den kürzesten Weg in ein Min/Max über partielle Ableitungen
- The gradient of a function is the direction of steepest ascent or descent.
- in der VL ist ein Beweis den ich nicht abtippe weil irrelevant

5.2.1 Linear Regression + Squared Loss

$$L_2(\mathbf{w}) = \frac{1}{2} \cdot \sum_{(\mathbf{x}, c(\mathbf{x})) \in D} (c(\mathbf{x}) - y(\mathbf{x}))^2$$

Jetzt müssen wir für jedes w_i aus $\mathbf{w}$ eine partielle Ableitung machen um den weight vector zu updaten ($\mathbf{w} = \mathbf{w} + \Delta\mathbf{w}$)

$$\Delta\mathbf{w} = \frac{\delta}{\delta w_i} L_2(\mathbf{w}) = \eta \cdot \sum_{D} (c(\mathbf{x}) - \mathbf{w}^T\mathbf{x}) \cdot \mathbf{x}$$

η = learning rate - a small positiv constant - legen wir selbst fest

5.2.2 The Batch Gradient Descent (BGD) Algorithm

BGD(D, η)

1. *initialize_random_weights*$(\mathbf{w})$, $t = 0$
2. **REPEAT**
3. $t = t + 1$
4. $\Delta\mathbf{w} = 0$
5. FOREACH $(\mathbf{x}, c(\mathbf{x})) \in D$ DO
6. $\delta = c(\mathbf{x}) - y(\mathbf{x})$ // $y(\mathbf{x}) \stackrel{(\star)}{=} \mathbf{w}^T\mathbf{x}$, $\delta \in \mathrm{R}$.
7. $\Delta\mathbf{w} \stackrel{(\star)}{=} \Delta\mathbf{w} + \eta \cdot \delta \cdot \mathbf{x}$ // $\delta \cdot \mathbf{x}$ is the derivative of $\ell_2(c(\mathbf{x}), y(\mathbf{x}))$.
8. ENDDO
9. $\mathbf{w} = \mathbf{w} + \Delta\mathbf{w}$ // $\Delta\mathbf{w}$ is $-\eta \cdot \nabla L_2(\mathbf{w})$ here.
10. **UNTIL**(*convergence*$(c(D), y(D))$ **OR** $t > t_{\max}$)
11. *return*$(\mathbf{w})$

- wichtig: immer wenn irgendwo $\mathbf{w}^T\mathbf{x}$ steht haben wir $x_0 = 1$ zu $\mathbf{x}$ hinzugefügt
- funktionsweise BGD. wir berechnen über die Ableitung in welche Richtung wir müssen und gehen dann einen Schritt der große η
- die "convergence" schaut ob der Squared Loss noch größer als ein ε ist (das wir auch festlegen)
- BGD ist nicht der schnellste (bestenfalls linear) aber sehr einfach (Newton-Raphson algorithm, BFGS algorithm sind z.B. schneller)
- BGD nimmt den global loss: loss of all examples in D ("batch gradient descent")(Schritt in Richtung die für alle Punkte am besten ist)
- man kann auch den (squared) loss in Bezug auf einzelne Beispiele nehmen (pointwise loss) (dann gehts halt im Zickzack runter) berechnet sich dann $\ell_2(c(\mathbf{x}), y(\mathbf{x})) = \frac{1}{2}(c(\mathbf{x}) - \mathbf{w}^T\mathbf{x})^2$
- bzw. die weight adaptation: $\Delta\mathbf{w} = \eta \cdot (c(\mathbf{x}) - \mathbf{w}^T\mathbf{x}) \cdot \mathbf{x}$
- für BGD_σ wird Zeile 9 zu $\mathbf{w} = \mathbf{w} + \Delta\mathbf{w} + \eta \cdot 2\lambda \cdot \binom{0}{\mathbf{w}}$

5.2.3 The Incremental Gradient Descent IGD Algorithm

IGD(D, η)

1. *initialize_random_weights*$(\mathbf{w})$, $t = 0$
2. **REPEAT**
3. $t = t + 1$
4. FOREACH $(\mathbf{x}, c(\mathbf{x})) \in D$ DO
5. $\delta = c(\mathbf{x}) - y(\mathbf{x})$ // $y(\mathbf{x}) \overset{(*)}{=} \mathbf{w}^T\mathbf{x}$, $\delta \in \mathbf{R}$.
6. $\Delta\mathbf{w} \overset{(*)}{=} \eta \cdot \delta \cdot \mathbf{x}$ // $\delta \cdot \mathbf{x}$ is the derivative of $\ell_2(c(\mathbf{x}), y(\mathbf{x}))$.
7. $\mathbf{w} = \mathbf{w} + \Delta\mathbf{w}$
8. ENDDO
9. **UNTIL**(*convergence*$(c(D), y(D))$ **OR** $t > t_{\text{max}}$)
10. *return*$(\mathbf{w})$

- kleinere Schritte als BGD
- can better avoid getting stuck in a local minimum of the loss function then BGD

5.2.4 Linear Regression + Squared Loss

$$L_{0/1}(\mathbf{w}) = \sum_D \frac{1}{2} \cdot (c(\mathbf{x}) - \text{sign}(\mathbf{w}^T\mathbf{x}))$$

$L_{0/1}8\mathbf{w})$ cannot be expressed as a diffrentiable function alias es kann nicht abgeleitet werden damit ist gradient descent nicht möglich

5.2.5 Logistic Regression + Logistic Loss + Regularization

Wie oben nur mit neuer Formel für $\Delta\mathbf{w}$:

$$\Delta\mathbf{w} = -\eta \cdot \nabla\mathcal{L}_\sigma(\mathbf{w}) = \eta \cdot \sum_D (c(\mathbf{x}) - \sigma(\mathbf{w}^T\mathbf{x})) \cdot \mathbf{x} - \eta \cdot 2\lambda \cdot \begin{pmatrix} 0 \\ \vec{\mathbf{w}} \end{pmatrix}$$

logistic loss Formel:

$$\mathcal{L}_\sigma(\mathbf{w}) = \sum_D -c(\mathbf{x}) \cdot \log(y(\mathbf{x})) - (1 - c(\mathbf{x})) \cdot \log(1 - y(\mathbf{x})) + \lambda \cdot \vec{\mathbf{w}}^T\vec{\mathbf{w}}$$

5.3 Multilayer Perceptron

5.3.1 Linear Separability

2 Klassen sind teilbar wenn ich da eine gerade Linie / Ebene / Hyperplane dazwischen packen kann.... oder:

Two sets of feature vectors, X_0, X_1, sampled from a p-dimensional feature space $\mathbf{X}$, are called linearly separable if p+1 real numbers, $w_0, w_1, ..., w_p$, exist such that the following conditions holds:

1. $\forall \mathbf{x} \in X_0 : \sum_{j=0}^{p} w_j x_j < 0$
2. $\forall \mathbf{x} \in X_1 : \sum_{j=0}^{p} w_j x_j \geq 0$

Problem: viele Probleme sind nicht linear separierbar Lösung: wir zeichnen mehrere Linien! (nehmen multilayer perceptron)

- The first, second, and third layer of the shown multilayer perceptron are called input, hidden,and output layer respectively
- input units perform no computation but only distribute the values to the next layer
- Compared to a single perceptron, the multilayer perceptron poses a significantly more challenging training (= learning) problem, which requires continuous (and non-linear) threshold functions along with sophisticated learning strategies.
- a continuous and non-linear threshold function:

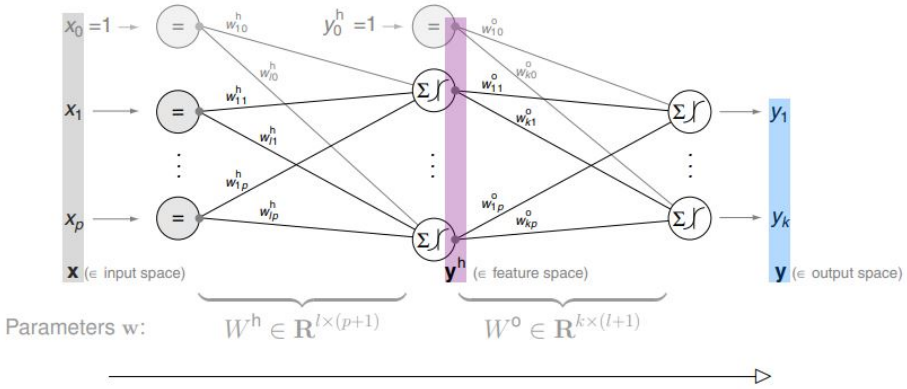
$$\sigma(z) = \frac{1}{1+e^{-z}} \Rightarrow \frac{\delta\sigma(z)}{\delta z} = \sigma(z) \cdot (1 - \sigma(z))$$

- und damit: $y(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$
- eine Alternative zu σ ist:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{e^{2z} - 1}{e^{2z} + 1}$$

- A "multilayer" perceptron with linear threshold functions can be expressed as a single linear function and hence is equivalent to the power of a single perceptron only $\Rightarrow$ Employing a nonlinear is necessary
- Multilayer perceptrons are also called multilayer networks or (artificial) neural network
-

A multilayer perceptron $\mathbf{y}(\mathbf{x})$ with one hidden layer and k-dimensional output layer:



Forward pass computation (aka. forward propagation) :

$$\mathbf{y}(\mathbf{x}) = \sigma\left(W^o\, \mathbf{y}^h(\mathbf{x})\right) = \boldsymbol{\sigma}\left(W^o \begin{pmatrix} 1 \\ \boldsymbol{\sigma}\left(W^h\, \mathbf{x}\right) \end{pmatrix}\right)$$

5.3.2 Forward propagation

The input data is fed in the forward direction through the network. Each hidden layer accepts the input data, processes it as per the activation function and passes to the successive layer

(a) Propagate $\mathbf{x}$ from input to hidden layer: (IGD$_{\text{MLP}}$ algorithm, Line 5)

$$W^h \in \mathbf{R}^{l\times(p+1)} \qquad \mathbf{x} \in \mathbf{R}^{p+1}$$

$$\boldsymbol{\sigma}\left(\begin{bmatrix} w_{10}^h & \dots & w_{1p}^h \\ & \vdots & \\ w_{l0}^h & \dots & w_{lp}^h \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_p \end{bmatrix}\right) = \begin{bmatrix} y_1^h \\ \vdots \\ y_l^h \end{bmatrix}$$

(b) Propagate $\mathbf{y}^h$ from hidden to output layer: (IGD$_{\text{MLP}}$ algorithm, Line 6)

$$W^o \in \mathbf{R}^{k\times(l+1)} \qquad \mathbf{y}^h \in \mathbf{R}^{l+1} \qquad \mathbf{y} \in \mathbf{R}^k$$

$$\boldsymbol{\sigma}\left(\begin{bmatrix} w_{10}^o & \dots & w_{1l}^o \\ & \vdots & \\ w_{k0}^o & \dots & w_{kl}^o \end{bmatrix} \begin{bmatrix} 1 \\ y_1^h \\ \vdots \\ y_p^h \end{bmatrix}\right) = \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix}$$

Forward propagation: Batch Mode

(a) Propagate $\mathbf{x}$ from input to hidden layer: (IGD$_{\text{MLP}}$ algorithm, Line 5)

$$W^{\mathsf{h}} \in \mathbf{R}^{l\times(p+1)} \qquad D \subset \mathbf{R}^{p+1}$$

$$\boldsymbol{\sigma}\left(\begin{bmatrix} w^{\mathsf{h}}_{10} & \dots & w^{\mathsf{h}}_{1p} \\ & \vdots & \\ w^{\mathsf{h}}_{l0} & \dots & w^{\mathsf{h}}_{lp} \end{bmatrix} \begin{bmatrix} 1 & \dots & 1 \\ x_{11} & \dots & x_{1n} \\ & \vdots & \\ x_{p1} & \dots & x_{pn} \end{bmatrix}\right) = \begin{bmatrix} y^{\mathsf{h}}_{11} & \dots & y^{\mathsf{h}}_{1n} \\ & \vdots & \\ y^{\mathsf{h}}_{l1} & \dots & y^{\mathsf{h}}_{ln} \end{bmatrix}$$

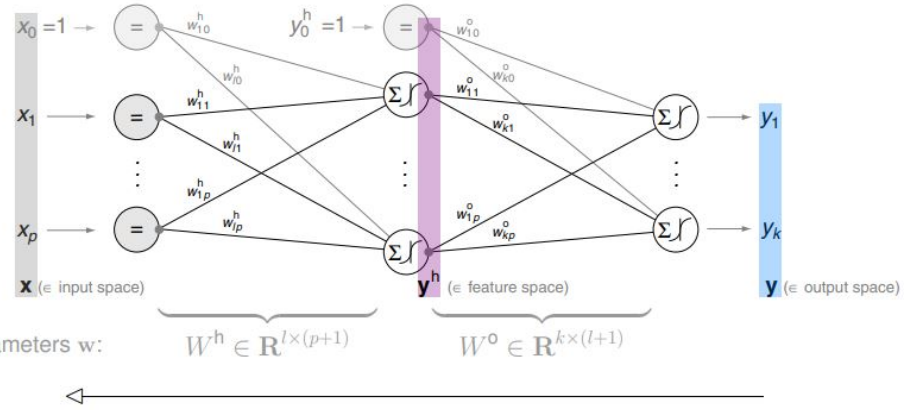
(b) Propagate $\mathbf{y}^{\mathsf{h}}$ from hidden to output layer: (IGD$_{\text{MLP}}$ algorithm, Line 6)

$$W^{\mathsf{o}} \in \mathbf{R}^{k\times(l+1)}$$

$$\boldsymbol{\sigma}\left(\begin{bmatrix} w^{\mathsf{o}}_{10} & \dots & w^{\mathsf{o}}_{1l} \\ & \vdots & \\ w^{\mathsf{o}}_{k0} & \dots & w^{\mathsf{o}}_{kl} \end{bmatrix} \begin{bmatrix} 1 & \dots & 1 \\ y^{\mathsf{h}}_{11} & \dots & y^{\mathsf{h}}_{1n} \\ & \vdots & \\ y^{\mathsf{h}}_{l1} & \dots & y^{\mathsf{h}}_{ln} \end{bmatrix}\right) = \begin{bmatrix} y_{11} & \dots & y_{1n} \\ & \vdots & \\ y_{k1} & \dots & y_{kn} \end{bmatrix}$$

5.3.3 Backwards Propagation

The considered multilayer perceptron $\mathbf{y}(\mathbf{x})$:



Weight update (aka. backward propagation) wrt. the global squared loss:

$$L_2(\mathbf{w}) = \frac{1}{2}\cdot \mathrm{RSS}(\mathbf{w}) = \frac{1}{2}\cdot \sum_{(\mathbf{x},\mathbf{c}(\mathbf{x}))\in D} \sum_{o=1}^{k} (c_o(\mathbf{x}) - y_o)^2$$

- $L_2(\mathbf{w})$ usually contains various local minima

$$\begin{bmatrix} \frac{\delta L_2(\mathbf{w})}{\delta w^o_{10}} & \cdots & \frac{\delta L_2(\mathbf{w})}{\delta w^o_{1l}} \\ & \vdots & \\ \frac{\delta L_2(\mathbf{w})}{\delta w^o_{k0}} & \cdots & \frac{\delta L_2(\mathbf{w})}{\delta w^o_{kl}} \end{bmatrix} \equiv \nabla^o L_2(\mathbf{w})$$

die selbe Formel gibt es nochmal nur mit "h" statt "o"
Update of weight matrix W^o: $W^o = W^o + \delta W^o$ mit:

$$\Delta W^o = -\eta \cdot \nabla^o L_2(\mathbf{w}) = \eta \cdot \sum_D [(W^{oT}\delta^o) \odot y^h(\mathbf{x}) \odot (1 - y^h(\mathbf{x}))]_{1,\ldots,l} \otimes \mathbf{x}$$

weiß irgendwer was diese lange Formel sagen will? Nein? okay.

5.4 IGD for Multilayer Perceptrons

1. *initialize_random_weights*$(W^{\mathsf{h}}, W^{\mathsf{o}})$, $t = 0$
2. **REPEAT**
3. $t = t + 1$
4. FOREACH $(\mathbf{x}, \mathbf{c}(\mathbf{x})) \in D$ DO
5. $\mathbf{y}^{\mathsf{h}} = \begin{pmatrix} 1 \\ \boldsymbol{\sigma}(W^{\mathsf{h}}\mathbf{x}) \end{pmatrix}$ // forward propagation, $\mathbf{x}$ is extended by $x_0 = 1$
6. $\mathbf{y} = \boldsymbol{\sigma}(W^{\mathsf{o}}\mathbf{y}^{\mathsf{h}})$
7. $\boldsymbol{\delta}^{\mathsf{o}} = (\mathbf{c}(\mathbf{x}) - \mathbf{y}) \odot \mathbf{y} \odot (\mathbf{1} - \mathbf{y})$ // backward propagation
8. $\Delta W^{\mathsf{o}} = \eta \cdot (\boldsymbol{\delta}^{\mathsf{o}} \otimes \mathbf{y}^{\mathsf{h}}|_{1,\ldots,l})$
9. $\boldsymbol{\delta}^{\mathsf{h}} = [(W^{oT}\boldsymbol{\delta}^{\mathsf{o}}) \odot \mathbf{y}^{\mathsf{h}} \odot (\mathbf{1} - \mathbf{y}^{\mathsf{h}})]_{1,\ldots,l}$
10. $\Delta W^{\mathsf{h}} = \eta \cdot (\boldsymbol{\delta}^{\mathsf{h}} \otimes \mathbf{x})$
11. $W^{\mathsf{o}} = W^{\mathsf{o}} + \Delta W^{\mathsf{o}}$ // weight update
12. $W^{\mathsf{h}} = W^{\mathsf{h}} + \Delta W^{\mathsf{h}}$
13. ENDDO
14. **UNTIL**(*convergence*$(\mathbf{c}(D), \mathbf{y}(D))$ **OR** $t > t_{\max}$)
15. *return*$(W^{\mathsf{h}}, W^{\mathsf{o}})$

- $\odot$ = Hadamard product
- $\otimes = \mathbf{v}\mathbf{w}^T$ dyadic product

another formel for $\nabla^o L_2(\mathbf{w})$:

$$\frac{\delta}{\delta w^o_{ij}} = -\sum_D (c_i(\mathbf{x}) - y_i(\mathbf{x})) \cdot y_i(\mathbf{x}) \cdot (1 - y_i(\mathbf{x})) \cdot y^h_j(\mathbf{x})$$

6 Decision Trees

6.1 Splitting

Das splitting von X ist die Teilung in mutually exclusive Teilmengen $X_1, ..., X_s$ alias $X = X_1 \cup ... \cup X_s$ mit $X_j \neq \emptyset$ and $X_j \cap X_j \neq \emptyset$, where

$j, j' \in \{1, ..., s\}, j \neq j'$. Eine Teilung $X_1, ..., X_s$ von X induces eine Teilung $D_1, ..., D_s$ von D, bei der $D_j, j = 1, ..., s$, als $\{(\mathbf{x}, c(\mathbf{x})) \in D | x \in X_j\}$ definiert wird.
Oder in Beispielen: Wir teilen unsere Pilze in 2 (oder mehr) Haufen anhand einer Eigenschaft, z.B. Farbe. Statt also einem Haufen beschrifteter Pilze (unser Beispieldatenset D) haben wir 3 Haufen: weiße Pilze, braune Pilze und rote Pilze. (die wiederum weiter aufgeteilt werden bis wir für jeden Haufen klar sagen können: essbar oder nicht essbar)
Die Teilung hängt von der Art der Eigenschaften/ Features ab. Diese können

- numerisch/ quantitative sein: Zahlen und Ratios, z.B. die Temperatur in °C, das Alter in Jahren oder Geld in Währung (Rechnen mit + & - und/oder * & / ergibt Sinn)
- kategorisch/ Qualitative sein: Namen, Farben, IDs, Postleitzahlen, Hausnummern, etc. (nur Vergleichen [=, $\neq$ bzw. manchmal noch $<$ & $>$] ergibt Sinn)

Daraus folgen verschiedene Teilmöglichkeiten:

- m-ary splitting induced by a (nominal) feature A with finite domain (braune, weiße und rote Pilze):

$$A = \{a_1, ...a_m\} : X = \{\mathbf{x} \in X : \mathbf{x}|_a = a_1\} \cup ... \cup \{\mathbf{x} \in X : \mathbf{x}|_a = a_m\}$$

- Binary splitting induced by a (nominal) feature A (Punkte vs keine Punkte):

$$A' \subset A : X = \{\mathbf{x} \in X : \mathbf{x}|_a \in A'\} \cup \{\mathbf{x} \in X : \mathbf{x}|_a \notin A'\}$$

- Binary splitting induced by an ordinal feature A (Temp $<$25 °C und Temp $\geq$25°C):

$$v \in dom(A) : X = \{\mathbf{x} \in X : \mathbf{x}|_a \succeq v\} \cup \{\mathbf{x} \in X : \mathbf{x}|_a \prec v\}$$

- $\mathbf{x}|a_i$ sind die $\mathbf{x}$ die eine eigenschaft a_i erfüllen (z.B. braun)

6.1.1 Possible criteria for splitting of X(t)

1. **Size of D(t)**: $D(t)$ is not split if $|D(t)|$ is below a threshold.
2. **Purity of D(t)**: $D(t)$ is not split if all examples in $D(t)$ are members of the same class
3. **Impurity reduction of D(t)**: $D(t)$ is not split if its impurity reduction, Δ_ι, is below a threshold (wenn es sich nicht lohnt weil es sich nicht signifikant verbessert)

6.2 Decision Trees Definiton

Let X be a set of features and C a set of classes. A decision tree T for X and C is a finite tree with a distinguished root node. A non-leaf node t of T has assigned

1 a set $X(t) \subset X$,

2 a splitting of $X(t)$, and

3 a one-to-one mapping of the subsets of the splitting to its successors.

$X(t) = X$ iff t is root node. A leaf node of T has assigned a class from C.

- decision trees (DT) haben kein Problem mit Rauschen/ Labelnoise (falsch klassifizierte Beispiele) da wenn diese in der Minderheit sind nicht berücksichtigt werden (wenn ich 5 mal bei sonnigem, warmen, windstillen Wetter Sport mache und 1 mal nicht) haben wir ein geteilten Knoten (siehe Bild) und entscheiden und meistens für die Mehrheit (Sport machen).
- manchmal (essbar/ nicht essbar) ergibt jedoch auch eine andere Entscheidung Sinn
- DT können nur klassifizieren nicht Regression

6.2.1 classification of some $\mathbf{x} \in X$ given a decision tree T

1. Find the root node t of T
2. if t is a non-leaf node, find among its successors that node t' whose subset of the splitting of $X(t)$ contains $\mathbf{x}$. Repeat this step with $t = t'$
3. If t is a leaf node, label x with the respective class

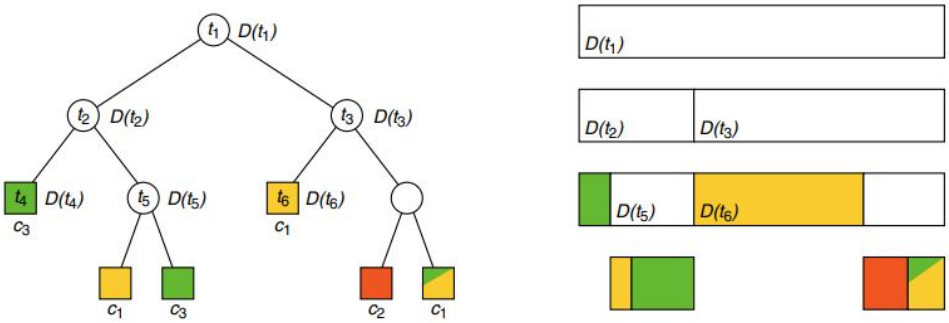
The set of possible decision trees forms the hypothesis space H

- The classification of an $x \in X$ determines a unique path from the root node of T to some leaf node of T
- At each non-leaf node a particular feature of x is evaluated in order to find the next node along with a possible next feature to be analyzed

6.2.2 Notations

Let T be a decision tree for X and C, let D be a set of examples [setting], and let t be a node of T. Then we agree on the following notation:

- $X(t)$ denotes the subset of X that is represented by t.
- $D(t)$ denotes the subset of the example set D that is represented by t, where $D(t) = (\mathbf{x}, c(\mathbf{x})) \in D | \mathbf{x} \in X(t)$. (see the splitting definition)



- The set $X(t)$ is comprised of those members x of X that are filtered by a path from the root node of T to the node t.
- $leaves(T)$ denotes the set of all leaf nodes of T
- ab hier: $t, T : X \to C$ statt $y_t, y_T : X \to C$

6.2.3 DT-construct

DT−construct(D)

1. $t =$ *newNode*$()$
   label$(t) =$ *representativeClass*(D)
2. **IF** *impure*(D)
   THEN *criterion* $=$ *splitCriterion*(D)
   ELSE *return*(t)
3. $\{D_1, \ldots, D_s\} =$ *decompose*$(D,$ *criterion*$)$
4. **FOREACH** D' **IN** $\{D_1, \ldots, D_s\}$ **DO**
   addSuccessor$(t,$ *DT−construct*$(D'))$
   ENDDO
5. *return*(t)

- Since DT-construct assigns to each node of a decision tree T a class, each subtree of T (as well as each pruned version of a subtree of T) represents a valid decision tree on its own
- $representativeClass(D)$: Returns a representative class for the example set D. Note that, due to pruning, each node may become a leaf node.
- $impure(D)$ Evaluates the (im)purity of a set D of examples
- $splitCriterion(D)$ Returns a split criterion for $X(t)$ based on the examples in $D(t)$.
- $decompose(D, criterion)$ Returns a splitting of D according to criterion
- $addSuccessor(t, t')$ Inserts the successor t' for node t

6.2.4 DT-classify

$DT\text{-}classify(\mathbf{x}, t)$

1. **IF** $isLeafNode(t)$
   THEN $return(label(t))$
   ELSE $return(DT\text{-}classify(\mathbf{x}, splitSuccessor(t, \mathbf{x}))$

- $isLeafNode(t)$ Tests whether t is a leaf node
- $splitSuccessor(t, x)$ Returns the (unique) successor t' of t for which $x \in X(t')$ holds

6.3 Evaluation of DT

Um noch Bias zu haben (und somit für nicht gesehene Beispiele abstrahieren zu können) müssen wir den Baum jetzt noch kürzen. (Overfitting vermeiden)

1. Size: Among all decision trees of minimum classification error we choose the one of smallest size
2. Classification Error: Quantifies the rigor according to which a class label is assigned to x in a leaf node of T, based on the examples in D. If all leaf nodes of a decision tree T represent a single example of D, the classification error of T with respect to D is zero. (Wollen wir meistens nicht)

6.3.1 measuring Size

- Leaf node number (wie viele Endknoten)
- Tree height (Längster Weg im Baum von Wurzel bis Blatt)
- External path length (Summe aller Wege von Wurzel bis Blätter = der Platz den es braucht alle in einem Baum gespeicherten Regeln zu speichern)
- Weighted external path length (wie external path lenght aber wir schauen für jeden Weg noch wie viele Beispiele aus D klassifiziert werden) → Ein kurzer Weg der viel klassifiziert + ein sehr langer weg der sehr wenig klassifiziert kann besser sein als 2 Halblange Wege (die beliebig aufteilend klassifizieren)
- Weighted external path length ist meistens bevorzugt

6.3.2 Classification Error

$$label(t) = \underset{c \in C}{\operatorname{argmax}} \frac{|(\mathbf{x}, c(\mathbf{x}) \in D(t) : c(\mathbf{x}) = c)|}{|D(t)|}$$

Missclassification rate/ Error of node classifier t based on that:

$$Err(t, D(t)) = \frac{|\{\mathbf{x}, c(\mathbf{x})) \in D(t) : c(\mathbf{x}) \neq label(t)\}|}{|D(t)|} = 1 - \max_{c \in C} \frac{|\{\mathbf{x}, c(\mathbf{x})) \in D(t) : c(\mathbf{x}) = c\}|}{|D(t)|}$$

Misclassification rate of decision tree classifier T:

$$Err(T, D) = \sum_{t \in leaves(T)} \frac{|D(t)|}{|D|} \cdot Err(t, D(t))$$

argmax gibt das $\mathbf{x}$ zurück für das eine Funktion $f(\mathbf{x})$ maximal ist, max gibt $y(\mathbf{x})$ zurück (Beispiel Wohnungspreise anhand der Lage: $argmax(\mathbf{x})$ = "Innenstadt" und $max(\mathbf{x})$ = "75000 Euro") (und genauso für argmin vs min)
Misclassification cost of node classifier t:

$$Err_{cost}(t, D(t)) = min_{c' \in C} \sum_{c' \in C} \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D(t) : c(\mathbf{x}) = c\}|}{|D(t)|} \cdot cost(c'|c)$$

Misclassification costs of decision tree classifier T:

$$Err_{cost}(T, D) = \sum_{t \in leaves(T)} \frac{|D(t)|}{|D|} \cdot Err_{cost}(t, D(t))$$

Wenn $c' = c$ ist $cost(c'|c) = 0$, verschiedene c können verschiedene Kosten haben (es ist schlimmer einen toxischen Pilz als essbar einzustufen als einen essbaren als toxisch)

6.4 Impurity Functions

An impurity function $\iota : [0; 1]^k \to R$ (mit $k \in N$) is a function defined on the standard $k - 1 - simplex$, denoted Δ^{k-1}, for which the following properties hold:

1. ι becomes minimum at points (1,0,...,0), (0,1,...,0),..., (0,...,0,1)
2. ι is symmetric with regard to its arguments, $p_1, ..., p_k$
3. ι becomes maximum at point $(1/k, ..., 1/k)$

simplex = einfachste Form zwischen Punkten in k- Dim raum (also k = 3 → Dreieck)?

6.4.1 Strict Impurity Function

genauso wie Impurity aber

3. wird zu: $\iota(\lambda p + (1-\lambda)p') > \lambda\iota(p) + (1-\lambda)\iota(p'), 0 < \lambda < 1, p \neq p'$

Wenn ι eine stricte impurity funktion ist gilt:

$$\Delta\iota(D, \{D_1, ..., D_s\}) \geq 0$$

6.4.2 Impurity of an Example Set

Impurity of $D = \iota(D)$:

$$\iota(D) = \iota(\frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_1\}|}{|D|}, ..., \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_k\}|}{|D|})$$

eine Funktion ist maximal impure wenn alle ihr zugeordneten Klassen c gleich wahrscheinlich sind (50% Wahrscheinlichkeit essbar, 50% Wahrscheinlichkeit toxisch).

6.4.3 Impurity Reduction

D aufgeteilt in $D_1, ..., D_s$, durch das splitting von X, dann ist die resulting impurity reduction:

$$\Delta\iota(D, \{D_1, ..., D_s\}) = \iota(D) - \sum_{j=1}^{s} \frac{|D-j|}{|D|} \cdot \iota(D_j)$$

6.4.4 Impurity Functions Based on the misclassification Rate

Definition für 2 classes:

$$\iota_{misclass}(p_1, p_2) = 1 - max\{p_1, p_2\} = \begin{cases} p_1 & \text{if } 0 \leq p_1 \leq 0.5 \\ 1 - p_1 & \text{otherwise} \end{cases}$$

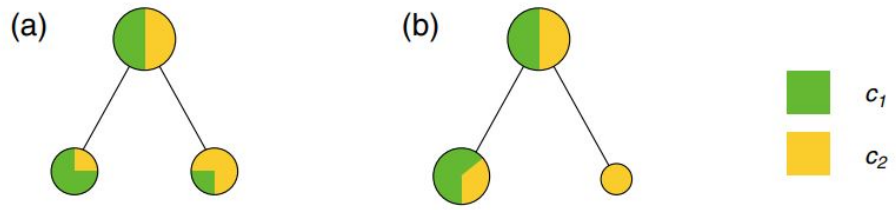
$$\iota_{misclass}(D) = 1 - max\{\frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_1\}|}{|D|}, \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_2\}|}{|D|}\}$$

Definition für k classes:

$$\iota_{misclass}(p_1, ..., p_k) = 1 - \max_{i=1,...,k} p_i$$

$$\iota_{misclass}(D) = 1 - \max_{c \in C} \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c\}|}{|D|}$$

Er hat in dem Video erklärt warum er hier plötzlich p_1 und p_2 benutzt statt c_1 und c_2, hat allerdings nicht so viel Sinn ergeben.



$$\Delta\iota_{misclass} = \iota_{misclass}(D) - \left(\frac{|D_1|}{|D|} \cdot \iota_{misclass}(D_1) + \frac{|D_2|}{|D|} \cdot \iota_{misclass}(D_2)\right)$$

left splitting: $\Delta\iota_{misclass} = \frac{1}{2} - (\frac{1}{2} \cdot \frac{1}{4} + \frac{1}{2} \cdot \frac{1}{4}) = \frac{1}{4}$

right splitting: $\Delta\iota_{misclass} = \frac{1}{2} - (\frac{3}{4} \cdot \frac{1}{3} + \frac{1}{4} \cdot 0) = \frac{1}{4}$

Das Bild erklärt das Rechnen besser als alle Formeln.

6.5 Entropy

Let A denote an event and let $P(A)$ denote the occurrence probability of A. Then the entropy (self-information, information content) of A is defined as $-\log_2(P(A))$. Let A be an experiment with the exclusive outcomes (events) $A_1, ..., A_k$. Then the mean information content of A, denoted as $H(A)$, is called Shannon entropy or entropy of experiment A and is defined as follows:

$$H(A) = -\sum_{i=1}^{k} P(A_i) \cdot \log_2(P(A_i))$$

The smaller the occurrence probability of an event, the larger is its entropy. An event that is certain has zero entropy

6.5.1 Conditional Entropy

Let $\mathcal{A}$ be an experiment with the exclusive outcomes (events) $A_1, ..., A_k$, and let $\mathcal{B}$ be another experiment with the outcomes $B_1, ..., B_s$. Then the conditional entropy of the combined experiment $(\mathcal{A}|\mathcal{B})$ is defined as follows:

$$H(\mathcal{A}|\mathcal{B}) = \sum_{j=1}^{s} P(B_j) \cdot H(\mathcal{A}|\mathcal{B}_j)$$

mit: $H(\mathcal{A}|\mathcal{B}_j) = -\sum_{i=1}^{k} P(A_i|B_j) \cdot \log_2(P(A_i|B_j))$

6.5.2 Information Gain

$$H(\mathcal{A}) - H(\mathcal{A}|\mathcal{B}) = H(\mathcal{A}) - \sum_{j=1}^{s} P(B_j) \cdot H(\mathcal{A}|\mathcal{B}_j)$$

- Information gain is defined as reduction in entropy
- In the context of decision trees, experiment $\mathcal{A}$ corresponds to classifying feature vector $\mathbf{x}$ with regard to the target concept. A possible question, whose answer will inform us about which event $A_i \in \mathcal{A}$ occurred, is the following: "Does $\mathbf{x}$ belong to class c_i?" Likewise, experiment $\mathcal{B}$ corresponds to evaluating feature B of feature vector $\mathbf{x}$. A possible question, whose answer will inform us about which event $B_i \in \mathcal{B}$ occurred, is the following: "Does $\mathbf{x}$ have value b_j for feature B?"
- Since $H(\mathcal{A})$ is constant, the feature that provides the maximum information gain (= the maximally informative feature) is given by the minimization of $H(\mathcal{A}|\mathcal{B})$.

6.5.3 Impurity Functions Based on Entropy

Tippen braucht zu viel Zeit, die Klausur ist morgen, also Screenshots for now:

Definition for two classes [impurity function]:

$$\iota_{entropy}(p_1, p_2) = -(p_1 \cdot \log_2(p_1) + p_2 \cdot \log_2(p_2))$$

$$\iota_{entropy}(D) = -\left(\frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_1\}|}{|D|} \cdot \log_2 \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_1\}|}{|D|} + \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_2\}|}{|D|} \cdot \log_2 \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_2\}|}{|D|}\right)$$

Definition for k classes:

$$\iota_{entropy}(p_1, \ldots, p_k) = -\sum_{i=1}^{k} p_i \cdot \log_2(p_i)$$

$$\iota_{entropy}(D) = -\sum_{i=1}^{k} \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_i\}|}{|D|} \cdot \log_2 \frac{|\{(\mathbf{x}, c(\mathbf{x})) \in D : c(\mathbf{x}) = c_i\}|}{|D|}$$

$\Delta\iota_{entropy}$ corresponds to the information gain $H(\mathcal{A}) - H(\mathcal{A} \mid \mathcal{B})$:

$$\Delta\iota_{entropy} = \underbrace{\iota_{entropy}(D)}_{H(\mathcal{A})} - \underbrace{\sum_{j=1}^{s} \frac{|D_j|}{|D|} \cdot \iota_{entropy}(D_j)}_{H(\mathcal{A}\,|\,\mathcal{B})}$$

- $\iota_{entropy}(D) = \iota_{entropy}(P(A_1), \ldots, P(A_k)) = -\sum_{i=1}^{k} P(A_i) \cdot \log_2(P(A_i)) = H(\mathcal{A})$
- $\frac{|D_j|}{|D|} \cdot \iota_{entropy}(D_j) = P(B_j) \cdot \iota_{entropy}(P(A_1 \mid B_j), \ldots, P(A_k \mid B_j)),\ j = 1, \ldots, s$
- $P(A_i), P(B_j), P(A_i \mid B_j)$ are estimated as relative frequencies based on D.

6.5.4 Impurity Functions Based on the Gini Index

Wurde in der VL nicht besprochen (ist aber in den Folien) ich lass es mal weg. Man findet es hier (ich bin ein Link) ab Folie 33.

6.6 Decision Tree Algorithms

6.6.1 ID3 Algorithm

Characteristics of the ID3 algorithm:

1. Each splitting is based on one nominal feature and considers its complete domain. Splitting für Feature A:
   $A = \{a_1, ...a_m\} : X = \{\mathbf{x} \in X : \mathbf{x}|_a = a_1\} \cup ... \cup \{\mathbf{x} \in X : \mathbf{x}|_a = a_m\}$
2. Splitting criterion is information gain.

ID3(D, Features, Target)

1. Create a node t for the tree.
2. Label t with the most common value of Target in D.
3. If all examples in D are positive, return the single-node tree t, with label "+".
   If all examples in D are negative, return the single-node tree t, with label "–".
4. If Features is empty, return the single-node tree t.

- Otherwise:
  5. Let A* be the feature from Features that best classifies examples in D.
     Assign t the decision feature A*.
  6. For each possible value "a" in A* do:
     - Add a new tree branch below t, corresponding to the test A* = "a".
     - Let D_a be the subset of D that has value "a" for A*.
     - If D_a is empty:
       Then add a leaf node with label of the most common value of Target in D.
       Else add the subtree ID3(D_a, Features \ {A*}, Target).
7. Return t.

ID3$(D, \mathit{Features}, \mathit{Target})$

1. $t = \mathit{createNode}()$
2. $\mathit{label}(t) = \mathit{mostCommonClass}(D, \mathit{Target})$
3. **IF** $\forall \langle \mathbf{x}, c(\mathbf{x}) \rangle \in D : c(\mathbf{x}) = c$ **THEN** $\mathit{return}(t)$ **ENDIF**
4. **IF** $\mathit{Features} = \emptyset$ **THEN** $\mathit{return}(t)$ **ENDIF**
5. $A^* = \text{argmax}_{A \in \mathit{Features}}(\mathit{informationGain}(D, A))$
6. **FOREACH** $a \in A^*$ **DO**

   $D_a = \{(\mathbf{x}, c(\mathbf{x})) \in D : \mathbf{x}|_{A^*} = a\}$

   IF $D_a = \emptyset$ **THEN**

   $t' = \mathit{createNode}()$

   $\mathit{label}(t') = \mathit{mostCommonClass}(D, \mathit{Target})$

   $\mathit{createEdge}(t, a, t')$

   ELSE

   $\mathit{createEdge}(t, a, \mathit{ID3}(D_a, \mathit{Features} \setminus \{A^*\}, \mathit{Target}))$

   ENDIF

   ENDDO
7. $\mathit{return}(t)$

- Step 3 of of the ID3 algorithm checks the purity of D and, given this case, assigns the unique class c, ($c \in dom(Target)$), as label to the respective node.
- The smaller $H(C|feature)$ is, the larger becomes the information gain. Hence, the difference $H(C) - H(C|feature)$ needs not to be computed since $H(C)$ is constant within each recursion step.

6.6.2 ID3 Beispiel

	Color	Size	Points	Edibility
1	red	small	yes	toxic
2	brown	small	no	edible
3	brown	large	yes	edible
4	green	small	no	edible
5	red	large	no	edible

Top-level call of ID3. Analyze a splitting with regard to the feature "color":

$D|_{\text{color}} =$

	toxic	edible
red	1	1
brown	0	2
green	0	1

$\rightarrow \quad |D_{\text{red}}| = 2,\ |D_{\text{brown}}| = 2,\ |D_{\text{green}}| = 1$

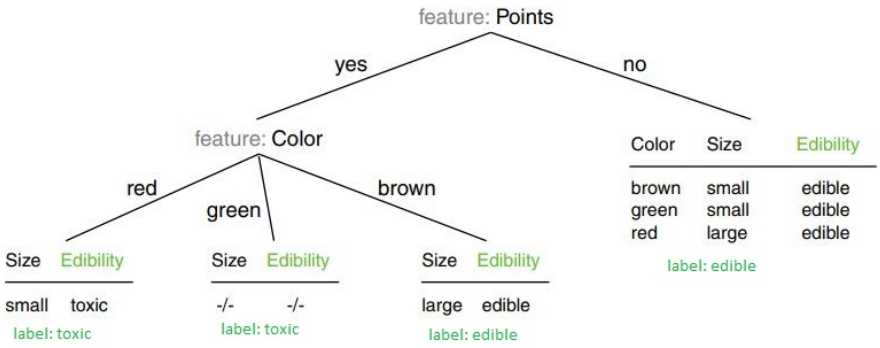
Estimated a-priori probabilities:

$$p_{\text{red}} = \frac{2}{5} = 0.4, \quad p_{\text{brown}} = \frac{2}{5} = 0.4, \quad p_{\text{green}} = \frac{1}{5} = 0.2$$

Conditional entropy values for all features:

$$\begin{aligned} H(C \mid \text{color}) &= -(\,0.4 \cdot (\tfrac{1}{2} \cdot \log_2 \tfrac{1}{2} + \tfrac{1}{2} \cdot \log_2 \tfrac{1}{2}) + \\ &\quad\ 0.4 \cdot (\tfrac{0}{2} \cdot \log_2 \tfrac{0}{2} + \tfrac{2}{2} \cdot \log_2 \tfrac{2}{2}) + \\ &\quad\ 0.2 \cdot (\tfrac{0}{1} \cdot \log_2 \tfrac{0}{1} + \tfrac{1}{1} \cdot \log_2 \tfrac{1}{1})\,) \;=\; 0.4 \end{aligned}$$

$$\begin{aligned} H(C \mid \text{size}) &\approx 0.55 \\ H(C \mid \text{points}) &= 0.4 \end{aligned}$$



6.6.3 ID3 Inductive Bias

Inductive bias is the rigidity in applying the (little bit of) knowledge learned from a training set for the classification of unseen feature vectors. Observations:

- Decision tree search happens in the space of all hypotheses. → The target concept is a member of the hypothesis space.
- To generate a decision tree, the ID3 algorithm needs per branch at most as many decisions as features are given. → no backtracking takes place and the decision tree is a result of local optimization

Where the inductive bias of the ID3 algorithm becomes manifest:

- Small decision trees are preferred
- Highly discriminative features tend to be closer to the root.

Is this justified?

6.6.4 CART Algorithm

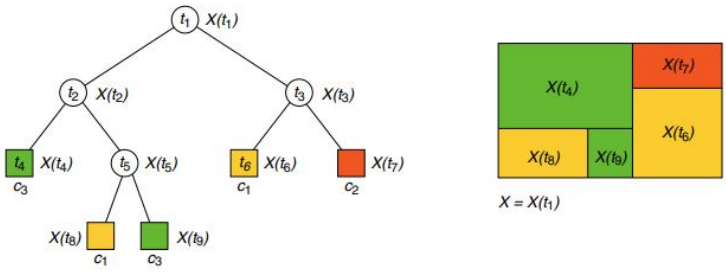
Characteristics of the CART algorithm: Each splitting is binary and considers one feature at a time and Splitting criterion is the information gain or the Gini index.

1. Let A be a feature with domain $\mathbf{A}$. Ensure a finite number of binary splittings for X by applying the following domain splitting rules:
   - If A is nominal, choose $\mathbf{A}' \subset \mathbf{A}$ such that $0 < |\mathbf{A}'| \leq |\mathbf{A} \setminus \mathbf{A}'|$.
   - If A is ordinal, choose $a \in \mathbf{A}$ such that $x_{\min} < a < x_{\max}$, where $x_{\min}$, $x_{\max}$ are the minimum and maximum values of feature A in D.
   - If A is numeric, choose $a \in \mathbf{A}$ such that $a = (x_k + x_l)/2$, where x_k, x_l are consecutive elements in the ordered value list of feature A in D.
2. For node t of a decision tree generate all splittings of the above type.
3. Choose a splitting from the set of splittings that maximizes the impurity reduction $\Delta\iota$:

$$\Delta\iota\big(D(t),\ \{D(t_L), D(t_R)\}\big) \ = \ \iota(t) - \frac{|D_L|}{|D|} \cdot \iota(t_L) - \frac{|D_R|}{|D|} \cdot \iota(t_R),$$

where t_L and t_R denote the left and right successor of t.

Illustration for two numeric features, i.e., the feature space $\mathbf{X}$ corresponds to a two-dimensional plane:



By a sequence of splittings the feature space $\mathbf{X}$ is split into rectangles that are parallel to the two axes.

6.7 Decision Tree Pruning

Most DT- Algos will overfit: Solution we prune!

- Overgrowing is what i call it.
- Accuracy is the percentage of correctly classified examples
- The training error Err(T, Dtr) of a decision tree T is a monotonically decreasing function in the size of T
- man könnte auch eher aufhören beim trainieren um overfitting zu vermeiden. Das werden wir nicht.

6.7.1 Overfitting (alias Overgrowing) Decision Trees

Definition: The hypothesis $h \in H$ is considered to overfit D if an $h' \in H$ with the following property exists:

$$Err(h, D) < Err(h', D) \text{ and } Err^*(h) > Err^*(h')$$

, where $Err^*(h)$ denotes the true missclassification rate of h, while $Err(h, D)$ denotes the error of h on the example set D.
Reasons for overfitting are often rooted in the example set D:

- D is noisy and we "learn noise"
- D is biased and hence not representative
- D is too small and hence pretends unrealistic data properties

6.7.2 stopping instead of pruning

Possible criteria for stopping [splitting criteria] :

1. Size of $D(t)$.
   $D(t)$ is not split if $|D(t)|$ is below a threshold.
2. Purity of $D(t)$.
   $D(t)$ is not split if all examples in $D(t)$ are members of the same class.
3. Impurity reduction of $D(t)$.
   $D(t)$ is not split if the resulting impurity reduction, $\Delta\iota$, is below a threshold.

Problems:

ad 1) A threshold that is too small results in oversized decision trees.

ad 1) A threshold that is too large omits useful splittings.

ad 2) Perfect purity cannot be expected with noisy data.

ad 3) $\Delta\iota$ cannot be extrapolated with regard to the tree height.

6.8 Pruning

The pruning principle:

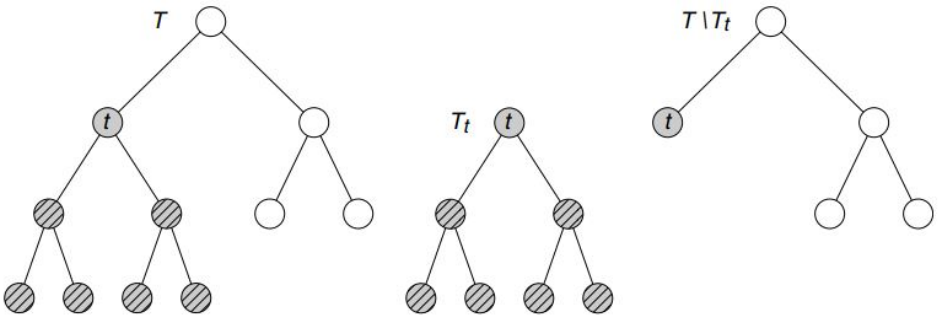
1. Construct a sufficiently large decision tree T_max.
2. Prune T_max, starting from the leaf nodes upwards to the tree root.

Each leaf node t of T_max fulfills one or more of the following conditions:

- $D(t$ is sufficiently small. Typically, $|D(t)| \leq 5$.

- $D(t)$ is pure (only one class)
- $D(t)$ is comprised of examples with identical feature vectors.

Given a decision tree T and an inner (non-root, non-leaf) node t. Then pruning of T with regard to t is the deletion of all successor nodes of t in T. The pruned tree is denoted as $T\backslash T_t$. The node t becomes a leaf node in $T\backslash T_t$



6.8.1 Pruning induced ordering

Let T' and T be two decision trees. Then $T' \preceq T$ denotes the fact that T' is the result of a (possibly repeated) pruning applied to T. The relation $\preceq$ forms a partial ordering on the set of all trees.
Problems when assessing pruning candidates:

- Pruned decision trees may not stand in the $\preceq$ lation.
- Locally optimum pruning decisions may not result in the best candidates
- Seine Monotonie disqualifies $Err(T, D_{tr})$ as an estimator for $Err^*(T)$
- $\rightarrow$ by making one pruning decision we loose access to the other path

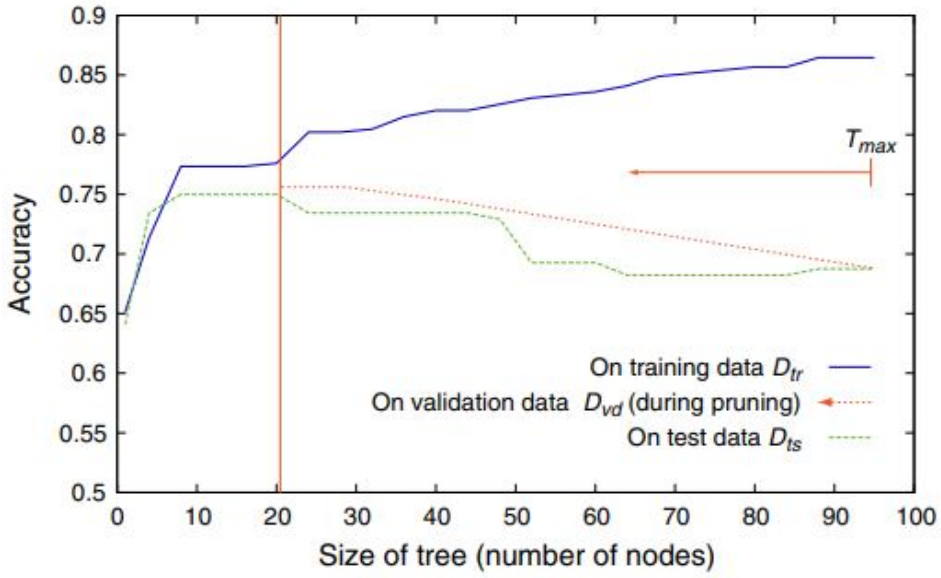
So we control pruning with validation set D_{val}, where $D_{val} \cap D_{tr} = \emptyset$, $D_{val} \cap D_{test} = \emptyset$:

1. $D_{tr} \subset D$ for decision tree construction
2. $D_{val} \subset D$ for overfitting analysis during pruning
3. $D_{test} \subset D$ for decision tree evaluation after pruning

6.8.2 Reduced Error Pruning

1. $T = T_{\max}$
2. Choose an inner node t in T.
3. Perform a tentative pruning of T with regard to t: $T' = T \setminus T_t$.
   Based on $D(t)$ assign class to t. [DT-construct]
4. If $Err(T', D_{vd}) \leq Err(T, D_{vd})$ then accept pruning: $T = T'$.
5. Continue with Step 2 until all inner nodes of T are tested.

Problem: If D is small, its partitioning into three sets for training, validation, and test will discard valuable information for decision tree construction.
Improvement: rule post pruning



7 Statistical Learning

7.1 Probability Basics

$$A \cap B = \emptyset \rightarrow P(A \cup B) = P(A) + P(B)$$

$$p(hypothesis|data) = \frac{p(data|hypothesis) \cdot p(hypothesis)}{p(data)}$$

$P(A|B)$ = "probability of A under condition B."

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \text{ , if } P(B) > 0$$

$$P(B) = \sum_{i=1}^{k} P(A_i) \cdot P(B|A_i)$$

A und B are called statistically independent iff: $P(A \cap B) = P(A) \cdot P(B)$

7.2 Bayes Classification

Theorem 12 (Bayes [1701-1761])

Let $(\Omega, \mathcal{P}(\Omega), P)$ be a probability space, and let $A_1, \ldots, A_k$ be mutually exclusive events with $\Omega = A_1 \cup \ldots \cup A_k$, $P(A_i) > 0$, $i = 1, \ldots, k$. Then for an event $B \in \mathcal{P}(\Omega)$ with $P(B) > 0$ holds:

$$P(A_i \mid B) = \frac{P(A_i) \cdot P(B \mid A_i)}{\sum_{i=1}^{k} P(A_i) \cdot P(B \mid A_i)}$$

$P(A_i)$ is called *prior probability* of A_i.

$P(A_i \mid B)$ is called *posterior probability* of A_i.