ML Konzepte: Teil 2 (Unbearbeitete Version)

Hayrettin Acar

December 14, 2023

1 Binäre Klassifikation

Die **Klassifikation** beschreibt die Abbildung der Eingabedaten auf eine definierte Menge. Bei der **binären** Klassifikation sind die möglichen Klassen gegeben durch die sog. **Positiv-/Negativklasse**, häufig gilt $y \in \{1, 0\}$ mit

$$y = \begin{cases} 0, & \text{Negativklasse} \\ 1, & \text{Positivklasse} \end{cases}$$

1.1 Performance Evaluation

Bevor die eigentlichen Algorithmen zur binären Klassifizierung behandelt werden, soll zunächst dargelegt werden, wie die Erfolgsrate quantifiziert werden kann. Ein erlernter Algorithmus kann mithilfe verschiedener Metriken evaluiert werden. Grundlage hierfür bietet beispielsweise die sog. **Konfusionsmatrix** der vier möglichen Basisfälle

	Positiv	Negativ
Vorhersage: Positiv	True Positive (TP)	False Positive (FP)
Vorhersage: Negativ	False Negative (FN)	True Negative (TN)

Definition (Präzision)**.** *Wir definieren die Präzision als das Verhältnis*

$$\frac{TP}{TP + FP}.$$

Die Präzision ist ein Umkehr-Maß dafür, wie hoch die **Falsch-Alarm** Rate ist. Eine hohe Präzision bedeutet also, dass wenige Negativ-Daten fälschlicher Weise als positiv eingestuft werden.

Definition (Sensitivität (Recall))**.** *Wir definieren die Sensitivität als das Verhältnis*

$$\frac{TP}{TP + FN}.$$

Die Sensitivität ist ein Maß dafür, wie gut das Modell in der Lage ist, die Gesamtanzahl der Positiv-Daten zu erkennen. Das Ziel sollte immer die Maximierung von Präzision und Sensitivität sein. Der sog. F_1 **Score** fasst diese zusammen zu

$$F_1 = 2 \times \frac{\text{Präzision} \times \text{Recall}}{\text{Präzision} + \text{Recall}}$$

Definition (Missrate)**.** *Wir definieren die Missrate als das Verhältnis*

$$\frac{FN}{TP + FN}.$$

Die Missrate ist ein Maß dafür, wie viele Positiv-Daten falsch eingestuft werden.

Definition (Spezifizität)**.** *Wir definieren die Spezifizität als das Verhältnis*

$$\frac{TN}{FP + TN}.$$

Die Spezifizität ist ein Maß dafür, wie gut das Modell in der Lage ist, die Gesamtanzahl der Negativ-Daten zu erkennen.

1.2 Logistische Regression

Die logistische Regression ist ein Verfahren, das häufig in der binären Klassifizierung verwendet wird. Wir wollen annehmen, dass y bernoulli-verteilt ist

$$\begin{aligned} y|x;\theta &\sim Bernoulli(\phi) \\ \phi &= p(y=1|x;\theta), \end{aligned} \tag{1}$$

wobei ϕ die Bernoulli-Wahrscheinlichkeit bezeichnet. Die Hypothese aus der Linearen Regression ist hier aus vielerlei Gründen nicht geeignet, z.B. aufgrund des Wertebereichs der Zielwerte. Sie wird stattdessen mithilfe der **sigmoid**-Funktion modelliert,

$$h_\theta(x) = g(\theta^T \vec{x}) = \frac{1}{1+e^{-\theta^T x.}}. \tag{2}$$

Der Verlauf der sigmoid-Funktion ist gezeigt in (1). Es gilt $\lim_{z\to\infty} = 1, \lim_{z\to-\infty} = 0$.

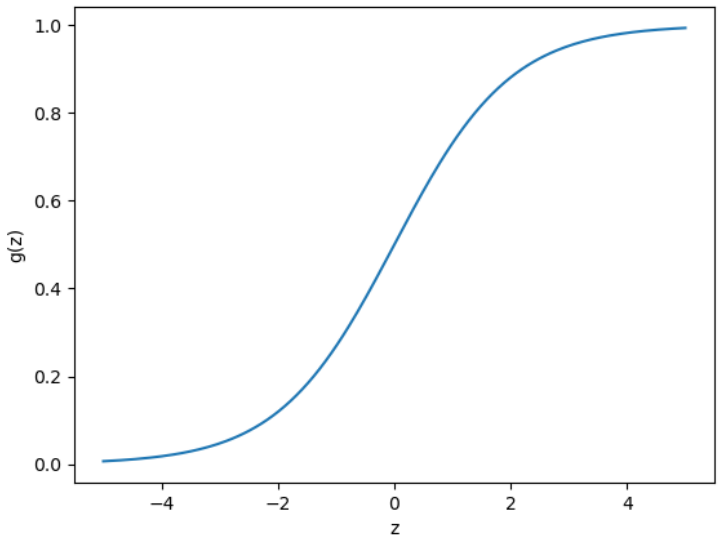


Figure 1: Der Verlauf der sigmoid-Funkton.

Die Raumdefinitionen sind damit

- Eingaberaum (input space) $\mathcal{X} = \mathbb{R}^d$.
- Aktionsraum (action space) $\mathcal{A} = \mathbb{R}$.
- Ergebnisraum (outcome space) $\mathcal{Y} = \{0, 1\}$.

Hier ist auch erstmals ersichtlich, weshalb die Trennung von Aktionsraum und Ergebnisraum sinnvoll ist. Im Falle der logistischen Regression entspricht der vorhergesagte Wert des Aktionsraumes nicht direkt einem definierten Zielwert.

1.2.1 Verlustfunktion

Mit der sigmoid-Funktion als Hypothese wäre die Anwendung der kleinsten Fehlerquadrate (analog zur Linearen Regression) nicht praktikabel, da die Konvexitätseigenschaft der Verlustfunktion nicht mehr gegeben wäre und somit die Optimierung zum globalen Extremum komplizierter würde. Analog zur Linearen Regression kann auch hier eine passende Verlustfunktion mit einem wahrscheinlichkeitsbasierten Ansatz gefunden werden.

Die Wahrscheinlichkeiten für beide Klassen können definiert werden als

$$\begin{aligned} P(y=1|x;\theta) &= h_\theta(x) \\ P(y=0|x;\theta) &= 1 - h_\theta(x). \end{aligned}$$

In kompakter Schreibweise lässt sich die Wahrscheinlichkeitsfunktion beschreiben mit

$$p(y|x;\theta) = (h_\theta(x))^y(1-h_\theta(x))^{1-y}. \tag{3}$$

Mit der Annahme, dass alle Trainingssätze unabhängig sind, kann die likelihood-Funktion beschrieben werden mit

$$\begin{aligned} L(\theta) = P(\vec{y}|x;\theta) &= \prod_{i=1}^{m} P(y^{(i)}|x^{(i)};\theta) \\ &= \prod_{i=1}^{m}(h_\theta(x^{(i)}))^{y^{(i)}}(1-h_\theta(x^{(i)}))^{1-y^{(i)}}. \end{aligned} \tag{4}$$

Die Maximierung des Ausdrucks erfolgt analog zur Linearen Regression durch Minimierung der logarithmischen Funktion,

$$\begin{aligned} l(\theta) &= -\log(L(\theta)) \\ &= \sum_{i=1}^{m}[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]. \end{aligned} \tag{5}$$

Die Kosten für die Differenz zwischen der Vorhersage $h_\theta(x^{(i)})$ und dem Zielwert $y^{(i)}$ werden also mithilfe der negativen logarithmischen Funktion modelliert. Diese Art der Verlustfunkton wird häufig auch (binäre) **Kreuzentropie**-Verlustfunktion genannt. Sie erlaubt den Vergleich zwischen einer vorhergesagten und ihrer *wahren* Wahrscheinlichkeitsverteilung.

1.2.2 Optimierung durch Gradientenverfahren (Gradient Descent)

Die Regel für das Gradientenabstiegsverfahren ist gegeben durch

$$\theta := \theta - \alpha\nabla_\theta l(\theta).$$

Man findet schnell, dass der Gradient gegeben ist mit

$$\frac{\partial}{\partial\theta_j} l(\theta) = (h_\theta(x) - y)x_j. \tag{6}$$

Damit ist das **stochastische** Gradientenverfahren gegeben mit

$$\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j, \forall j \tag{7}$$

und entspricht dabei exakt der bereits gefundenen Regel für die Lineare Regression.

1.2.3 Beispiel 1

Betrachtet sei die Zulassung verschiedener Bewerber, basierend auf dem Ergebnis zweier Prüfergebnisse. In diesem Beispiel werden zwei Eingabemerkmale (Prüfungsergebnis 1/2) betrachtet. Mit Berücksichtigung von $x_0^i = 1$ wird also eine Eingabematrix mit den Dimensionen $(m \times 3)$ verwendet. Insgesamt werden $m = 100$ Trainingsdaten verwendet. Mithilfe der optimierten Parameter von θ lässt sich die Entscheidungsgerade darstellen wie in (2) gezeigt.

Mit den gefundenen Parametern lassen sich darüber hinaus Vorhersagen für neue Daten treffen. So lässt sich beispielhaft die Wahrscheinlichkeit für eine Zulassung für die Ergebnisse PE1: 45 und PE2: 85 mit 77.6 % angeben.

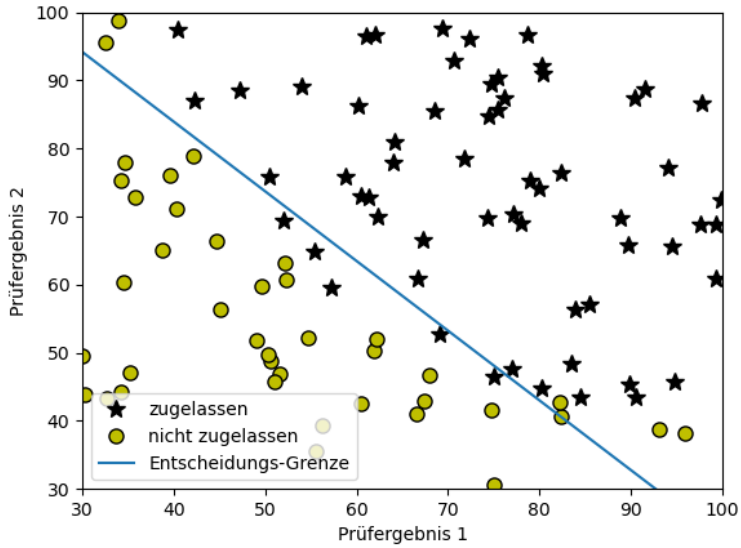


Figure 2: Die Zulassungsergebnisse anhand zweier Prüfungsergebnisse mit Entscheidungsgerade.

1.2.4 Beispiel 2

Betrachtet sei nun die Anzahl funktionierender Mikrochips auf Basis zweier Testwertungen, dargestellt in (3).

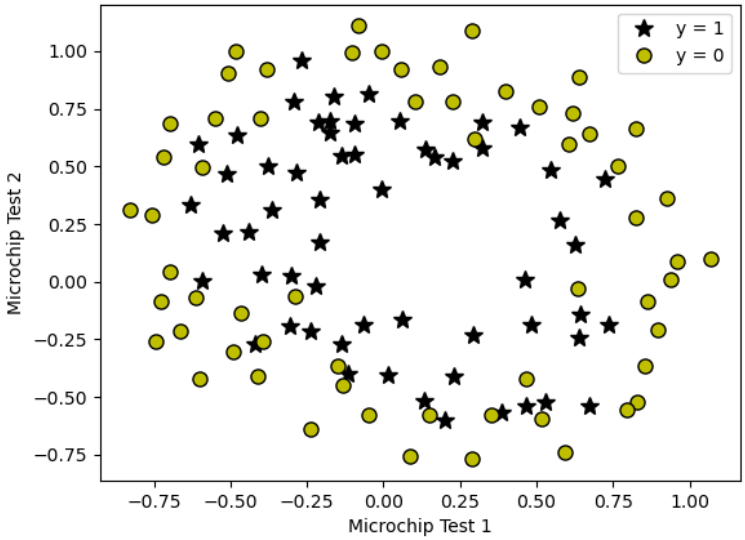


Figure 3: Die Darstellung funktionsfähiger (y=1) Mikrochips in Abhängigkeit zweier Tests.

Ein einfaches lineares Regressionsmodell im ursprünglichen Merkmalsraum ist nicht in der Lage, eine gekrümmte Entscheidungsgrenze zu erfassen. Um dieses Problem zu lösen, können die Merkmale der Trainingsdaten in ein höherdimensionales Problem transformiert werden (Feature-Mapping). Ein Beispiel ist in (4) demonstriert.

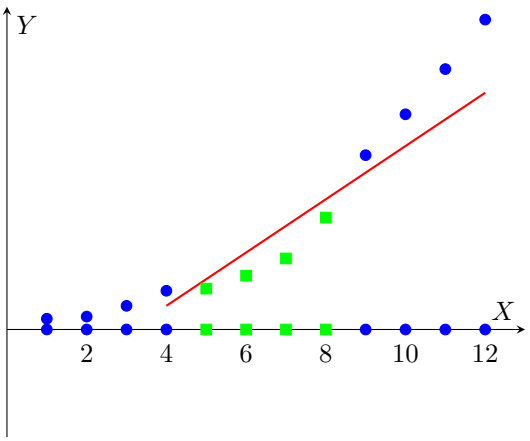


Figure 4: Beispiel für lineare Separation in 2D mithilfe von Feature Mapping.

Es werden also nicht die ursprünglichen Merkmale $x \in \mathbb{R}^d$ betrachtet, sondern eine Funktion $\phi(x) : \mathbb{R}^d \longrightarrow \mathbb{R}^p$. Durch die Anwendung einer Feature-Mapping Funktion, wie das Quadrieren eines Merkmals (x^2), werden neue Merkmal eingeführt, welche es dem Modell ermöglichen, eine gekrümmte Entscheidungsgrenze im transformierten Raum zu erfassen.
Eine **polynomiale** Abbildung zum Grad P wäre beispielsweise gegeben durch

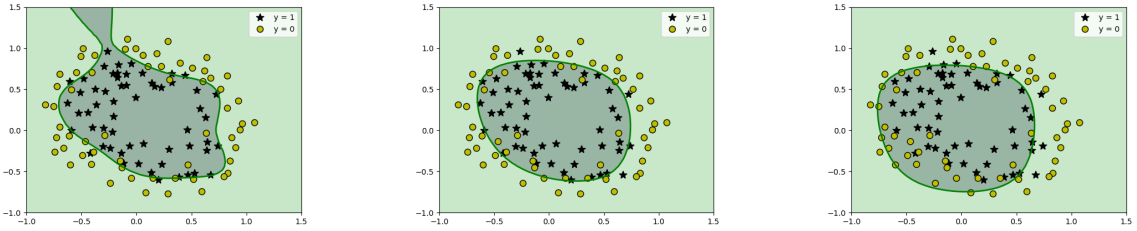
$$\Phi(x) = [1, x_1, x_2, \ldots, x_d, x_1^2, x_1x_2, \ldots, x_d^P, x_1^{P-1}x_2, \ldots, x_1x_d^{P-1}, x_2^2, \ldots, x_d^2]. \tag{8}$$

Da die Neigung zur Überanpassung mit der Feature-Mapping Technik erhöht wird, wird häufig auch Regularisierung verwendet,

$$J(\theta) = \sum_{i=1}^{m}[-y^{(i)} \log(h_\theta(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))] + \frac{\lambda}{2m} \sum_{j=1}^{d} \theta^2. \tag{9}$$

Der Strafterm hilft, die Komplexität des Modells zu kontrollieren, indem die Parameter mit einem quadratisch anwachsenden Kostenanteil belegt werden. Der Parameter λ bestimmt wie stark die Regularisierung berücksichtigt werden soll. Höhere Werte von λ verringern die Parameteroptimierung von θ zugunsten der Regularisierung und führen zu einem **underfitting** Effekt

Betrachtet sei beispielsweise die Abbildung auf ein Polynom 6. Grades, wie in (5) zu sehen ist. Das Beispiel zeigt die Entscheidungsgrenze für zwei unterschiedliche Werte des Regularisierungsparameters λ.



(a) $\lambda = 0$ (b) $\lambda = 1$ (c) $\lambda = 10$

Figure 5: Die Darstellung funktionsfähiger (y=1) Mikrochips in Abhängigkeit zweier Tests mit Entscheidungsgrenze.

Es ist deutlich zu erkennen, dass ohne Regularisierung ($\lambda = 0$) ein overfitting der Trainingsdaten vorliegt. Andererseits führt eine vergleichsweise hohe Regularisierung ($\lambda = 10$) zu underfitting.

2 Binäre Multiklassen-Klassifizierung

Bei der Multi-Klassifikation besteht das Ziel darin, Eingabedatenpunkte einer von mehreren k möglichen Klassen zuzuordnen, $y_k \in \{1, ..., K\}$. Der Algorithmus versucht also, die Eingabedaten in eine von

mehreren vordefinierten Klassen zu kategorisieren. Jeder Datenpunkt kann nur zu einer einzigen Klasse gehören und das Ziel besteht darin, die richtige Klassenbezeichnung für jede Eingabe genau vorherzusagen. In der binärbasierten Klassifizierung werden Algorithmen der binären Klassifizierung verwendet, um Multiklassen-Probleme zu lösen.

In der **one-vs-one** Methode wird jede Klasse paarweise mit jeder anderen Klassen verglichen. Damit ergeben sich $K\frac{K-1}{2}$ verschiedene Klassifizierer. Für eine Vorhersage wird ein neuer Eingabedatensatz mit allen Klassifizierern ausgewertet und die Anzahl der zutreffenenden Klassen sog. **Votes** gezählt. Die Klasse mit der höchsten Anzahl der Votes entspricht dann der Vorhersage.

In der **one-vs-rest** Methode, wird für jede Klasse ein eigener Klassifizierer zur Abgrenzung zum kombinierten Rest der übrigen Daten trainiert. Für eine neue Vorhersage wird ein neuer Eingabedatensatz mit allen Klassifizierern ausgewertet und mit einem festen **Score** $\hat{y}_k$ belegt. Die Klassifizierung mit dem höchsten Score entspricht dann der Vorhersage des Algorithmus,

$$\hat{y}_k = \arg \max_{k \in \{1,\dots,K\}} h_k(x)$$

2.1 OvR-Verfahren: Beispiel

Betrachtet sei die dargestellte Verteilung bezüglich zweier Eingabemerkmale (Feature 1/2), wie nachfolgend in (6) dargestellt.

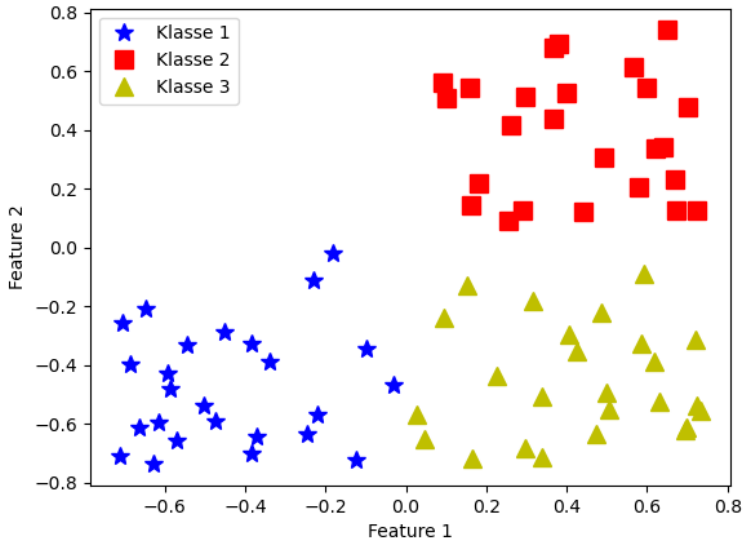
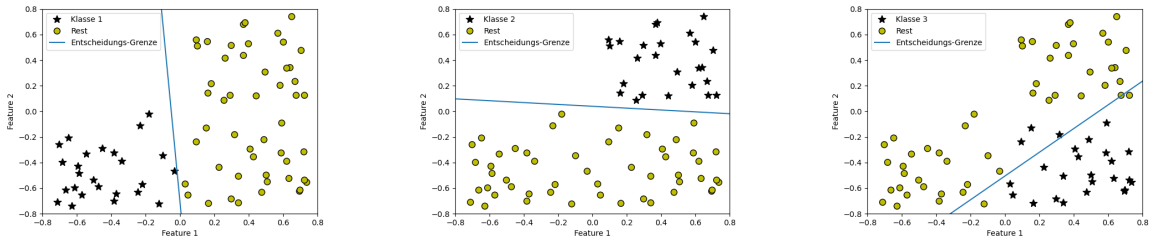


Figure 6: Die Verteilung von 3 verschiedenen Klassen in Abhängigkeit von zwei Eingabemerkmalen.

Mit der (linearen) **OvR**-Methode kann nun für jede Klasse ein Klassifizierer trainiert werden. Zu beachten ist hierbei, dass die Zielvariablen der Trainingsdaten entsprechend der binären Klassifikation auf den Bereich $[0, 1]$ abgebildet werden müssen. Wie in (7) zu sehen ist, kann jede Klasse mithilfe eines des trainierten Klassifizierers von den restlichen Daten abgegrenzt werden.



(a) Klassifikation für Klasse 1. (b) Klassifikation für Klasse 2. (c) Klassifikation für Klasse 3

Figure 7: Die Klassifikation aller drei Klassen mit Entscheidungsgeraden.

Betrachtet sei nun ein Testpunkt mit den Eingabewerten $[-0.2, -0.4]$, wie in (8) dargestellt ist.

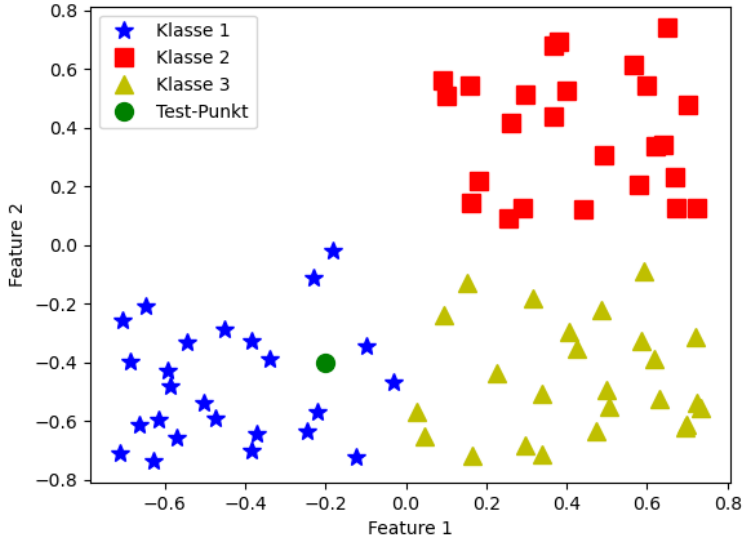


Figure 8: Die Verteilung von 3 verschiedenen Klassen in Abhängigkeit von zwei Eingabemerkmalen mit hervorgehobenem Testpunkt.

Für jede Klasse kann nun ein Score berechnet werden.

$$Score = \begin{cases} 0.977 & , \text{Klasse 1} \\ 0 & , \text{Klasse 2} \\ 0.039 & , \text{Klasse 3} \end{cases}$$

Damit würde der Testpunkt nach dem OvR-Verfahren der Klasse 1 zugeordnet werden (welches bei Betrachtung der Lage sinnvoll erscheint). Auch die relative Nähe des Testpunktes zu Klasse 3 wird im errechneten Score berücksichtigt.

3 Softmax-Regression

Ein natürlicher Ansatz zur Lösung eines Multiklassen-Problems ist die Abbildung des Eingabevektors auf einen Wahrscheinlichkeitsvektor, wobei jede Komponente der Wahrscheinlichkeit einer Klasse entspricht. Die Softmax-Funktion bildet den k-dimensionalen Eingabevektor mit reellen Komponenten auf einen Wahrscheinlichkeitsvektor mit den Komponenten $\phi_1, ..., \phi_K = P(y = 1|x; \theta), ..., P(y = K|x; \theta)$ ab. Dabei wird die Wahrscheinlichkeit der k-ten Klasse mithilfe der d-dimensionalen Parameter-Vektoren $\theta_1, ..., \theta_K \in \mathbb{R}^d$ ermittelt. Der gewichtete Trainingsdatensatz $\theta^T x$ wird häufig auch **Logit** genannt. Für einen einzelnen Trainingsdatensatz gilt dann

$$\phi_k = P(y = k|x; \theta) = \frac{\exp(\theta_k^T x)}{\sum_{k=1}^{K} \exp(\theta_k^T x)}. \tag{10}$$

Der Wahrscheinlichkeitsvektor ist dann gegeben mit

$$\begin{bmatrix} P(y = 1|x; \theta) \\ \vdots \\ P(y = K|x; \theta) \end{bmatrix} = \begin{bmatrix} \frac{\exp(\theta_1^T x)}{\sum_{k=1}^{K} \exp(\theta_k^T x)} \\ \vdots \\ \frac{\exp(\theta_K^T x)}{\sum_{k=1}^{K} \exp(\theta_k^T x)} \end{bmatrix}.$$

Er beschreibt die Wahrscheinlichkeit, dass das i-te Trainingsbeispiel zur k-ten Klasse zugehörig ist.

Mit der Annahme unabhängiger und gleichverteilter Trainingsdaten kann die likelihood-Funktion beschrieben werden mit

$$L(\theta) = \prod_{i=1}^{m} P(\vec{y} = y^{(i)}|x^{(i)}; \theta).$$

In diesem Zusammenhang ist $\vec{y}$ als one-hot encoded Vektor zu verstehen. Es werden also nur die Anteile zur Berechnung der Wahrscheinlichkeit einer jeweiligen Klasse berücksichtigt. Die Maximierung des likelihood-Vektors erfolgt dann wie zuvor, durch die Minimierung des negativen logarithmischen Ausdrucks. Mithilfe der Indikator-Funktion kann die Verlustfunktion geschrieben werden als

$$\begin{aligned} l(\theta) &= -\log(L(\theta)) \\ &= -\sum_{i=1}^{m} \sum_{j=1}^{k} 1\{y^{(i)} = k\} \log(\frac{\exp(\theta_{y^{(i)}}^T x)}{\sum_{k=1}^{K} \exp(\theta_j^T x^{(i)})}). \end{aligned} \tag{11}$$

Iterativ kann nun die Optimierung mithilfe des Gradienten bestimmt werden

$$\nabla_{\theta^{(k)}} J(\theta) = -\sum_{i=1}^{m} [x^{(i)}(1\{y^{(i)} = k\} - P(y^{(i)} = k|x^{(i)}; \theta))] \tag{12}$$

mit

$$P(y^{(i)} = k|x^{(i)}; \theta) = \frac{\exp(\theta_k^T x^{(i)})}{\sum_{k=1}^{K} \exp(\theta_k^T x^{(i)})}.$$

Für das bereits betrachtete **Beispiel** (siehe OvR-Verfahren), lässt sich mithilfe der Softmax- Regression für den Testpunkt $[-0.2, -0.4]$ die folgende Wahrscheinlichkeitsverteilung ermitteln,

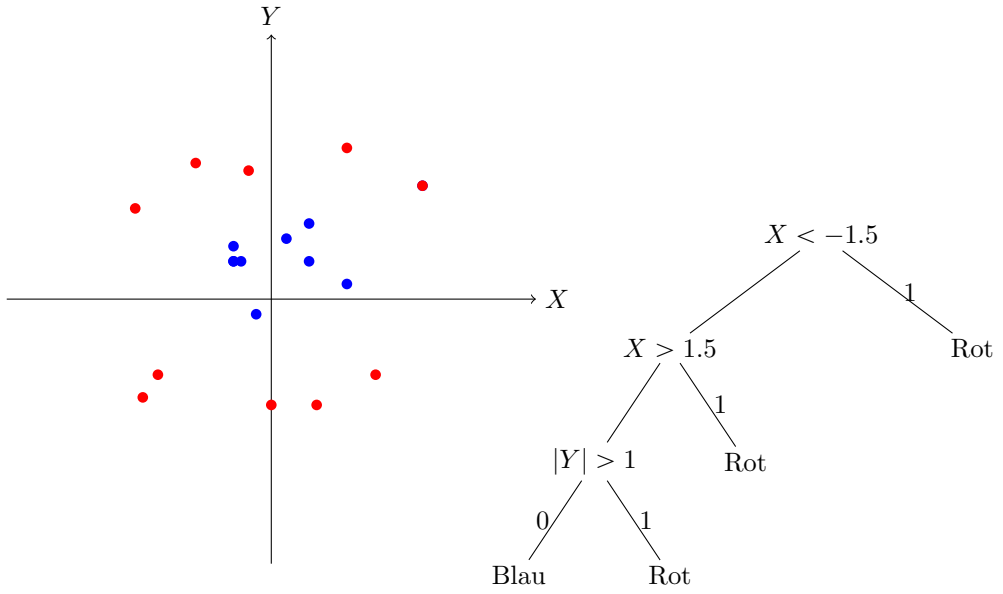
$$P(k) = \begin{cases} 0.509 & \text{, Klasse 1} \\ 0.155 & \text{, Klasse 2} \\ 0.336 & \text{, Klasse 3} \end{cases}$$

4 Entscheidungsbäume (decision trees)

(Binäre) Entscheidungsbäume können sowohl für Regressions- aber auch für Klassifikationsaufgaben verwendet werden. Dabei darf es sich auch um nichtlineare Klassifikationsprobleme handeln, d.h. Verteilungen von Klassen, die anhand ihrer Merkmalscharakteristiken nicht einfach durch eine Gerade oder mithilfe von Ebenen voneinander trennbar sind.

Ein Baum entscheidet immer anhand eines einzelnen Merkmals darüber, wie die Daten aufzuteilen sind, sog. **Entscheidungsknoten**. Die sog. **Blätter** dienen der endgültigen Zuordnung zwischen der Eingabe und dem Ergebnis bzw. der Klasse.

Betrachtet sei beispielsweise ein binäres Klassifizierungsproblem in Abhängigkeit zweier Merkmale X, Y . Ein entsprechender Entscheidungsbaum könnte nun folgendermaßen aussehen.



(a) Klassifikationsproblem mit 2 Merkmalen X,Y.

(b) Beispiel eines binären Entscheidungsbaumes.

Eine erfolgreiche Klassifizierung kann anhand von vielen, unterschiedlichen Entscheidungsbäumen vorgenommen werden. Das Ziel eines entsprechenden Lernalgorithmus ist nun, die Kriterien der Entscheidungsknoten so festzulegen, dass einerseits eine Klassifizierung für möglichst viele Testpunkte erfolgreich läuft. Andererseits soll der Entscheidungsbaum auch so simpel wie möglich sein.

4.1 Regressionsbäume

Die Idee der Entscheidungsbäume ist es, dass der Eingaberaum auf alle verfügbaren Blätter aufgeteilt wird, d.h. eine definierte Menge von Eingaben führt immer zum gleichen Ergebnis. Implizit wird der Eingaberaum dabei auf die verschiedenen (disjunkte) Regionen $R_1, ..., R_M$ aufgeteilt,

$$\mathcal{X} = \bigcup_{i=1}^{M} R_i \text{ , mit } R_i \bigcap_{i \neq j} R_j = \varnothing. \tag{13}$$

Die Anzahl der Regionen ist also direkt abhängig von der Anzahl der Entscheidungsknoten, welche definiert werden. Bei der Regression wird jedem Blatt ein bestimmter Wert c_M zugeordnet. Eine Vorhersage kann dann berechnet werden zu

$$f(x) = \sum_{m=1}^{M} c_M 1(x \in R_m). \tag{14}$$

Die Frage ist hierbei wie die Werte von c_m zu bestimmen sind. Eine Möglichkeit ist dabei, anhand der Trainingsdaten den durchschnittlichen Wert der Ergebnisdaten aller Eingaben zu nehmen, welche

der entsprechende Region zugeordnet wurden,

$$\hat{c}_m = \frac{1}{|R_m|} \sum_{i:x^{(i)} \in R_m} y^{(i)}$$

wobei $|R_m|$ die Gesamtanzahl der Datenpunkte innerhalb einer Region m beschreibt. Eine Region kann anhand eines reellen Schwellwertes immer weiter aufgeteilt werden. Die Aufteilungsregel kann so definiert werden, dass die primäre Region R_P anhand des j-ten Merkmals aufgeteilt wird in

$$\begin{aligned} R_1(j,t) &= \{X | X_j > t, X \in R_P\} \\ R_2(j,t) &= \{X | X_j \leqslant t, X \in R_P\}. \end{aligned} \tag{15}$$

Es wird also immer geprüft, ob der Wert eines Merkmals innerhalb einer Region größer oder kleiner als ein definierter **Entscheidungswert** $t \in \mathbb{R}$ ist. Eine definierte Region kann dann weiter unterteilt werden, z.B. $R_3(j,t) = \{X | X_j > t, X \in R_1\}$ usw. Bevor eine Region aufgeteilt werden kann, stellen sich zunächst zwei Fragen

- Anhand welchen Merkmals soll eine Region aufgeteilt werden?
- Welcher Wert c_m soll der Region zugeordnet werden?

Mit einer Anzahl von m Datensätzen innerhalb einer Region, können sinnvoller Weise nur $m-1$ Entscheidungswerte zwischen den gegebenen Eingabemerkmalen gefunden werden, für die eine unterschiedliche Aufteilung resuliert. Eine übliche Konvention ist hierbei jeden Entscheidungswert exakt zwischen zwei der (sortierten) Eingabewerte zu platzieren,

$$t_j \in \{\frac{1}{2}(x_{j(m)} + x_{j(m+1)}) | m = 1, ..., m-1\}.$$

Der Ansatz zur Bestimmung eines ersten Entscheidungsmerkmals könnte dann wie folgt aussehen. Es werden zunächst alle Merkmale hinsichtlich einer definierten Verlustfunktion für jeden möglichen Entscheidungswert evaluiert. Anschließend wird mit dem Merkmal des geringsten Verlustes begonnen. Diese Prozedur wird dann rekursiv durchgeführt, bis jedes Merkmal verwendet wurde.

Eine Möglichkeit zur Definition der Verlustfunktion ist beispielsweise die Betrachtung der Fehlerquadrate aller Regionen

$$\sum_{i:x^{(i)} \in R_1} (y^{(i)} - \hat{c}_1(j,t))^2 + ... \sum_{i:x^{(i)} \in R_M} (y^{(i)} - \hat{c}_M(j,t))^2 \tag{16}$$

Zu beachten ist allerdings, dass die Gesamtanzahl der Entscheidungsknoten nicht zu komplex ausfallen sollte. Theoretisch kann ein Baum so definiert werden, dass letztlich jeder einzelne Datensatz einer eigenen Region zugeordnet wird, was einem Fall der starken Überanpassung entspricht . Um dies zu verhindern, gibt es verschiedene Regularisierungsansätze (Minimale Blattgröße, Maximale Entscheidungstiefe, Maximale Anzahl von Entscheidungen etc.).

4.2 Klassifikationsbäume

Für eine Unterteilung in $y_k \in \{1, ..., K\}$ Klassen, müssen andere Kriterien gefunden werden. Zunächst sei der Anteil für jede Klasse innerhalb einer Region definiert als

$$\hat{p}_{m,k} = \frac{1}{|R_m|} \sum_{i:x^{(i)} \in R_m} 1(y^{(i)} = k). \tag{17}$$

Die vorhergesagte Klasse ist für diese Region ist dann

$$k(m) = \arg\max_k \hat{p}_{m,k}.$$

Es stellt sich nun die Frage, welches eine geeingte Verlustfunktion für die Klassifikation darstellt. Bei einer guten Klassifikation, beinhaltet jedes Blatt nur die Daten einer bestimmten Klasse, die sog.

Knotenreinheit ist also hoch. Man könnte auf die Idee kommen, die Verlustfunktion anhand der Fehl-Klassifizierung zu definieren,

$$1 - \arg\max_k \hat{p}_{m,k}.$$

Allerdings hilft dies nur bedingt die Knotenreinheit zu maximieren. Eine bessere Alternative ist die Verwendung der allgemeinen **Kreuzentropie** Verlustfunktion

$$-\sum_{k=1}^{K} \hat{p}_{m,k} \log \hat{p}_{m,k}. \tag{18}$$

Die Idee dahinter ist, dass der Fehler für eine vergleichsweise eindeutige Klassenzuordnung, d.h. $\hat{p}_{m,k} \approx 1$, gering ist.