5.4 Beschränkungen linearer Modelle, lineare Separierbarkeit

Lineare Modelle sind sehr intuitiv und leicht zu handhaben (trainieren, optimieren). Sie haben aber auch wichtige Beschränkungen – so wie sich immer Vorteile und Nachteile auswiegen. Eine sehr alte und bekannte Feststellung ist, dass gewisse sehr einfache Boolesche Funktionen nicht mittels linearer Modelle berechnet werden können. Das bekannteste Beispiel hierfür ist das exclusive logische oder, oft mit XOR bezeichnet. XOR ist eine binäre Funktion

$$\{0,1\} \times \{0,1\} \to \{0,1\}$$

definiert durch folgende Tabelle:

XOR	0	1
0	0	1
1	1	0

Diese Funktion ist besonders durch eine Eigenschaft: von

$$\text{XOR}(0,0)$$

ausgehend vergrößert sich das Ergebnis mit einer Vergrößerung in beiden Argumenten einzeln (also monoton steigen). Aber ausgehend von

$$\text{XOR}(0,1) \text{ und } \text{XOR}(1,0)$$

ist es in den beiden 0-Komponononenten *monoton fallend*. Anders gesagt: ob XOR in der ersten/zweiten Komponononente monoton steigend/fallend ist, hängt von der jeweils anderen Komponente ab!

Um zu sehen warum das nicht gehen kann, brauchen wir den Begriff der **linearen Separierbarkeit**. Nimm eine lineare Funktion

$$f : \mathbb{R}^n \to \mathbb{R}.$$

Dann ist die Menge

$$\{x : f(x) \geq \alpha\} \text{ für ein beliebiges } \alpha$$

trennbar durch eine *Hyperebene im Vektorraum* $\mathbb{R}^n$ (also im Fall $n = 1$ ein Punkt, im Fall $n = 2$ eine Gerade, im Fall $n = 3$ eine Ebene etc.).

Wenn man die XOR-Funktion in eine Ebene zeichnet (bzw. die beiden Datenpunkte), dann ist klar dass es keine Linie geben kann, die die Werte ≥ 1 von denen < 1 separiert.

Das ist eine wichtige Beobachtung, die die Beschränkungen linearer Funktionen zeigt. Falls wir aber lineare Funktionen im allgemeineren Sinn nutzen (also Polynome), stimmt die lineare Separierbarkeit bereits nicht mehr; allerdings bleibt die Separierbarkeit durch eine – je nach Grad des Polynoms – konstant begrenzte Menge von Hyperebenen, und das Problem des XOR lässt sich neu stellen. Vergleiche auch: lineare Regression vs. nearest neighbour regression.

Anhand dieses Beispiel kann man auch die Wichtigkeit nichtlinearer Funktionen zeigen: wenn wir unsere Eingabedaten **nicht-linear transformieren**, z.B. mittels der Funktion

$$\phi(x, y) = (x \cdot y, x + y) \tag{43}$$

dann bekommen wir eine Funktion

XOR	0	1	2
0	0	1	-
1	-	-	0

Das Problem hierbei ist: die passende nichtlineare Transformation zu finden setzt im Normalfall voraus, dass wir das Problem um das es sich handelt, bereits gut verstehen, und das wir die gesuchte Generalisierungen bereits in groben Umrissen kennen. Genau das möchten wir beim maschinellen Lernen aber nicht voraussetzen!

5.5 Abschluss unter Komposition

Eine wichtige Eigenschaft linearer Funktionen ist ihr Abschluss unter **Komposition**, dann heißt: wenn wir ein lineares Modell auf ein lineares Modell applizieren, dann resultiert daraus eine lineare Funktion. Wir definieren, wie üblich, $f \circ g$ durch $f \circ g(x) = f(g(x))$. Der Abschluss unter Komposition sagt nun: falls f, g lineare Funktionen sind, dann ist auch $f \circ g$ eine lineare Funktion. Das kann man für den einfachen Fall sehr schön zeigen: sei

$$f(x) = ax + b \tag{44}$$

$$g(x) = cx + d \tag{45}$$

Nun ist

$$\begin{aligned} f \circ g(x) &= a(cx + d) + b \\ &= acx + ad + b \\ = (ac)x + adb \end{aligned}$$

also offensichtlich eine lineare Funktion. Man kann das, nach demselben Muster, auch für komplexere Funktionen zeigen (Polynome, Matrizen auf Vektoren). Die Schlussfolgerung hieraus ist: wir werden nicht mächtiger, wenn wir lineare Modelle hintereinander schalten. Das bedeutet auch: die nichtlinearen Aktivierungsfunktionen in neuronalen Netzen sind entscheiden – wenn die nicht wären, dann wäre das ganze Netz ein lineares Modell. Die Mächtigkeit von neuronalen Netzen basiert also auf der Abfolge

lineares Modell - nichtlineares Modell - ... - lineares Modell - nichtlineares Modell

Wenn wir die Kombination lineares Modell+nichtlineares Funktion nehmen, dann haben wir keinen Abschluss unter Komposition: es gibt also eine echte Hierarchie von Klassen von Funktionen

$$l_1(x) \tag{46}$$

$$n_1 \circ l_1(x) \tag{47}$$

$$l_2 \circ n_1 \circ l_1(x) \tag{48}$$

$$n_2 \circ l_2 \circ n_1 \circ l_1(x) \tag{49}$$

$$... \tag{50}$$

wobei l_i jeweils eine lineare Funktion ist, n_i jeweils eine nichtlineare Funktion. Je mehr solche Funktionen wir hintereinander schalten, desto mächtiger wird die Klasse der Funktionen, die wir auf diese Weise berechnen können. Hierauf basiert die Mächtigkeit neuronaler Netze und das universelle Approximationstheorem.

5.6 Die Logistische Regression und Aktivierungsfunktionen

Logistische Regression ist im engeren Sinne keine Regression, sondern Klassifikation; es ist aber eine Klassifikation, die auf linearer Regression aufbaut. Logistische Regression funktioniert im einfachen Fall, wenn wir 2 Klassen haben, zwischen denen wir auswählen, z.B. A und B. Unser Problem ist also folgendes: wir möchten eine Eingabe in $\mathbb{R}$ nach A oder B klassifizieren. Wir machen das mittels der logistischen Sigmoid-Funktion

$$S(x) = \frac{1}{1+e^{-x}} \tag{51}$$

Einige Beobachtungen: wir haben

- $lim_{x\to-\infty}S(x) = 0$
- $lim_{x\to\infty}S(x) = 1$
- $S(x) \in (0,1)$ f.a. $x \in \mathbb{R}$
- $S(x)$ ist streng monoton steigend in x

```
> f <- function(x){1/(1+exp(-x))}
> plot(f,xlim=c(-10,10),ylim=c(0,1))
```

Allgemeiner nennt man solche Funktionen **Sigmoidfunktionen**. Ein anderes Beispiel für eine solche Funktion ist

$$T(x) = \frac{x}{1+|x|} \tag{52}$$

$$U(x) = \frac{x}{1+x^2} \tag{53}$$

$$tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 1 - \frac{2}{e^{2x}+1} \tag{54}$$

Was allen diesen Funktionen gemeinsam ist, ist dass sie im Bereich um 0 relativ schnell von 0 bzw. -1 Richtung 1 wechseln, sonst für große positive/negative Zahlen langsam Richtung 1/0 konvergieren. Es gibt also nur einen kleinen Bereich, in dem eine nicht extreme Änderung im Argument eine

signifikante Änderung im Wert verursacht; ansonsten gibt es relativ schnell eine Sättigung. solche Prozesse findet man oft in der Natur, z.B. bei Populationen. Solche logistischen Funktionen haben allerdings die Einschränkung dass sie nur $\mathbb{R} \to \mathbb{R}$ definiert sind. Aktivierungsfunktionen spielen in neuronalen Netzen eine große Rolle, und werden normalerweise auf die einzelnen Komponenten von Vektoren angewendet.

Wenn wir nun 2 Klassen haben, dann können wir einfach sagen: wir interpretieren den Wert als eine **Wahrscheinlichkeit**, also:

$$P(X = A|x) = S(f(x)) \tag{55}$$

wobei f eine lineare Funktion ist, die wir mittels linearer Regression induzieren. Wir transformieren also eine Funktion in die reellen Zahlen zu einer Funktion nach $[0, 1]$:

$$f : \mathbb{R} \to \mathbb{R} \Longrightarrow S \circ f : \mathbb{R} \to [0, 1]$$

Das Ergebnis können wir dann als Wahrscheinlichkeit interpretieren. Das gibt uns nun einen Klassifikator:

$$C(x) = \begin{cases} A, \text{ falls } P(X = A|x) > 0.5 \\ B \text{ andernfalls} \end{cases} \tag{56}$$

Das funktioniert natürlich nur, falls wir nur zwei Klassen (hier A, B) zur Verfügung haben. Intuitiv bilden wir hier die Eingabe auf eine Wahrscheinlichkeit ab, und Klassifizieren nach Maximum Likelihood. Natürlich kann auch das sehr leicht generalisiert werden auf beliebige Eingaben $\mathbf{x} \in \mathbb{R}^n$.

5.7 Bedeutung

Die Bedeutung der logistischen Regression ist erstmal folgende: wir suchen eine Klassifikationsfunktion

$$f : \mathbb{R}^n \to \{0, 1\}$$

Wir tun das aber mittels zweier Zwischenschritte:

$$\mathbb{R}^n \underset{f}{\Longrightarrow} \mathbb{R} \underset{S}{\Longrightarrow} \{0, 1\}$$

Die zugrundeliegende Annahme ist hierbei, dass es einen kritischen Bereich gibt, auf dem die Klassifikation von 0 zu 1 springt und umgekehrt ist; aber damit man das sieht, muss man zunächst eine **numerische Transformation** durchführen, nämlich die lineare Regression. Genauer gesagt, die lineare Funktion f kann die numerischen Werte so transformieren, dass gilt:

$$\text{Falls } f(x) > s, \text{ dann } C(x) = 1, \text{ und falls } f(x) \leq s, \text{ dann } C(x) = 0$$

Das bedeutet, wir haben eine *quasi-lineare Abhängigkeit* von unseren Eingabedaten und den Klassen. Das kann man noch genauer fassen, es gibt nämlich eine genaue Beschreibung wenn wir den **Hyperparameter** des **Grades der Funktion** mitberücksichtigen. Nehmen wir an, wir suchen eine Funktion $f : \mathbb{R} \to \{0, 1\}$. Wir wissen, dass

- Eine lineare Funktion hat eine Gerade als Graphen, kann also einen gewissen Wert nur einmal annehmen
- Ein Polynom zweiten Grades hat einen Wendepunkt, kann einen gewissen Wert höchstens zweimal annehmen.
- ...
- Ein Polynom $n$ten Grades hat n Wendepunkte, kann einen gewissen Wert höchstens n-mal annehmen.

Dann heißt das soviel wie:

- Falls wir eine lineare Funktion haben, dann gibt es ein $\alpha \in \mathbb{R}$, so dass falls $x < \alpha$, dann $f(x) = 1$ (bzw. 0), ansonsten $f(x) = 0$ (bzw. 1). Wir spalten also den Eingaberaum $\mathbb{R}$ in zwei Teile.

- Falls wir ein Polynom zweiten Grades haben, dann gibt $\alpha_1, \alpha_2 \in \mathbb{R}$, so dass falls $\alpha_1 < x < \alpha_2$, dann $f(x) = 1$ (bzw. 0), ansonsten $f(x) = 0$ (bzw. 1). Wir spalten also den Eingaberaum $\mathbb{R}$ in drei Teile.
- ...
- Falls wir ein Polynom n-ten Grades haben, haben wir $\alpha_1, ..., \alpha_n$, die $\mathbb{R}$ in $n + 1$ Teile spalten, und wir klassifizieren auf dieser Basis.

5.8 Lernen & Anwendung

Wir betrachten folgendes Beispiel: wir möchten, als abhängige, diskrete Variable vorhersagen, ob ein Student die Prüfung besteht (0 oder 1; abhängig bedeutet nur: wir möchten das vorhersagen). Als Prädiktor nutzen wir die Anzahl der Stunden, die der Student gelernt hat. Unsere Regressionsfunktion soll eine einfache lineare Funktion sein. Unser Datensatz sieht nun wie folgt aus (nennen wir den Datensatz $D1$):

Stunden gelernt	4	5	6	7	8	9	10
Bestanden	0	0	1	0	1	1	1

Eine lineare Regression liefert hier folgendes Ergebnis (danke R):

(57) $f(x) = 0.1786x - 0.6786$

Wir haben also eine positive Abhängigkeit von Lern-Stunden und Klausur-Bestehen (zum Glück); aber dieses Ergebnis ist noch unbefriedigend: wir können das nicht sinnvoll als Wahrscheinlichkeit interpretieren. Wir setzen aber nun diesen Term ein in unsere Aktivierungsfunktion, und defineren:

(58) $$P(\text{bestehen}|n \text{ Stunden lernen}) = \frac{1}{1 + e^{-f(x)}} = \frac{1}{1 + e^{-(0.1786x - 0.6786)}}$$

Das liefert uns nun ordentlich Werte:

Stunden	4	5	6	7	8	9	10
P(bestehen)	0.508	0.553	0.597	0.639	0.679	0.717	0.752

Das ist schon besser, aber nicht wirklich optimal, insbesondere stört die Asymmetrie (bei 4 haben wir bereits eine ziemlich hohe Wahrscheinlichkeit!).

Die logistische Funktion interagiert mit der linearen auf eine Art und Weise, die nicht optimal ist. Wir sollten also stattdessen folgendes suchen:

$$(59) \quad argmin_{a,b\in\mathbb{R}} \sum_{(x,y)\in D} \frac{1}{1+e^{-(ax+b)}} - y$$

Wir nutzen also die Aktivierungsfunktion, um den Unterschied zwischen 0 und 1 sichtbar zu machen, da eine lineare Funktion hierzu nicht in der Lage ist. Um 107 auszurechnen gibt reichen keine elementaren Methoden, man muss also etwas komplexere numerische Optimierung anwenden (der Computer kann das). R macht das mit dem Kommando:

> glm(x ~ y, family = binomial())

Nun definieren wir:

> g <- function(x){1/(1+exp(-(1.251*x-8.113)))}

Das sollte nun stimmen; wir bekommen dementsprechend:

Stunden	4	5	6	7	8	9	10
P(bestehen)	0.0427	0.135	0.353	0.656	0.869	0.959	0.988

Tadaa! Wir bekommen also eine Wahrscheinlichkeit. Was aber interessanter ist, ist folgende Frage: gegeben diese Ergebnisse, wie ist die Wahrscheinlichkeit, dass die Abhängigkeit des Bestehens von der Lernzeit reiner Zufall ist? Das ist eine klassische statistische Frage, und im Zusammenhang mit logistischer Regression gibt es den sog. **Wald-test**:

$$(60) \quad \frac{(\theta_{ML}-\theta)^2}{var(\theta_{ML})}$$

Er gibt die Wahrscheinlichkeit, dass die gegebene Verteilung zufällig ist, also dass es keine Abhängigkeit der beiden Variablen (Lernzeit / Bestehen) gibt. Das ist verwandt mit dem Test des Likelihood-Verhältnisses, was ein wenig einfacher zu verstehen ist. Seien

- ω unsere Beobachtungen, also die Daten in D1;
- H_0 die Nullhypothese, also Unabhängigkeit der Variablen: Lernzeit hat keinen Einfluss auf das Bestehen, Wahrscheinlichkeit von Bestehen oder Nicht-bestehen wird durch Maximum Likelihood geschätzt.

- H_1 die Alternativhypothese, also unsere durch logistische Regression berechnete bedingte Verteilung.

Dann haben wir den Likelihood-Quotienten:

$$R(\omega) := \frac{P(\omega|H_0)}{P(\omega|H_1)} \qquad \text{(Sonderfall für } P(\omega|H_1) = 0) \tag{61}$$

Hierauf kann man nun den Schwellentest S_t anwenden, $t > 0$, und üblicherweise $t = 0.05$. Zur Erinnerung: das ist der Test, der sich für H_0 entscheidet falls $R(\omega) > t$, und für H_1 andernfalls, also:

$$S_t(\omega) = \begin{cases} H_0, \text{ falls } R(\omega) > t \\ H_1 \text{ andernfalls.} \end{cases} \tag{62}$$

Das können wir/Sie nun ausrechnen:

5.9 Ein Vergleich: nearest neighbour Regression

Nearest neighbour regression ist eine denkbar simple Methode: wir haben eine Funktion $f : V \to C$, wobei V ein Vektorraum ist, und C eine (endliche) Menge von Klassen; wir haben eine Menge $D \subseteq V \times C$ von Datenpunkten, mittels derer wir die Funktion f "lernen" sollen. Was wir hierfür erstmal brauchen ist der Begriff der **Norm**:

Eine Norm, definiert auf einem Vektorraum V ist das eine Funktion

$$\| - \| : V \to \mathbb{R}_0^+$$

es werden also beliebige Vektoren auf einen nicht-negativen Wert abgebildet. Zusätzlich muss $\| - \|$ noch folgende Bedingungen erfüllen f.a. $\vec{v} \in V$, $\lambda \in \mathbb{R}$.

1. $\|\vec{v}\| = 0 \Rightarrow \vec{v} = 0_V$ (die 0 des Vektorraums, neutral für Addition)
2. $\|\lambda \cdot \vec{v}\| = |\lambda| \cdot \|\vec{v}\|$, wobei $|\lambda|$ der Betrag ist (respektiert Skalarmultipikation)
3. $\|\vec{v} + \vec{w}\| \leq \|\vec{v}\| + \|\vec{w}\|$ (allgemeine Dreiecksungleichung)

Die intuitivste Norm ist die *euklidische*, die jedem Vektor seine **Länge** zuweist (wenn wir einen Vektor als eine Linie vom Ursprung auf seine Koordinaten (im n-dimensionalen Raum) auffassen. Diese Norm basiert auf einer Verallgemeinerung des Satz des Pythagoras:

$$\|(v_1, ..., v_n)\| = \sqrt{v_1^2 + ... + v_n^2} \tag{63}$$

In dieser geometrischen Interpretation wird Bedingung 3 zur **Dreiecksungleichung**: in jedem rechwinkligen Dreieck ist die Länge der Hypothenuse geringer als die Summe der Länge der Katheten. Es gibt aber noch viele weitere Normen, z.B. die sog. p-Norm, wobei $p \geq 1$ eine reelle Zahl ist:

$$\|(v_1, ..., v_n)\|_p = \left(\sum_{i=1}^{n} |v_i|^p \right)^{\frac{1}{p}} \tag{64}$$

Für $p = 1$ vereinfacht sich das zu

$$\|(v_1, ..., v_n)\|_1 = \sum_{i=1}^{n} |v_i| \tag{65}$$

Das ist die sog. Manhattan-Norm, weil man immer rechtwinklig um die Blocks fahren muss – das gibt im 2-dimensionalen Fall also die kürzeste Strecke in Manhattan an.

Darauf basiert der Begriff der Metrik; jede Norm induziert eine Metrik d mittels

(66) $d(\vec{v}, \vec{w}) = \|\vec{v} - \vec{w}\|$

wobei natürlich

(67) $\vec{v} - \vec{w} = (v_1 - w_1, ..., v_i - w_i)$

Es ist nicht schwer zu sehen dass gilt:

- $d(x, y) \geq 0$ (positiv)
- $d(x, y) = d(y, x)$ (symmetrisch)
- $d(x, y) \leq d(x, z) + d(z, y)$ (Dreiecksungleichung)

Die **euklidische Distanz** ist definiert durch die euklidische Norm, mit

(68) $d_2(\vec{v}, \vec{w}) = \|\vec{v} - \vec{w}\|_2$

Nun machen wir einfach folgendes: gegeben einen beliebigen Vektor $\vec{v}$, eine Norm $\| - \|$ mit Metrik d, definieren wir

(69) $nn_D(\vec{v}) = argmin_{(\vec{w},c)\in D} d(\vec{v}, \vec{w})$

Wir suchen uns also den **nächsten Nachbarn** nach unserer Metrik. Als nächstes machen wir das denkbar naheliegendste: wir machen genau das, was der nächste Nachbar macht (Einfachheit halber fassen wir unseren Datensatz D als partielle Funktion $D : V \to C$ auf)

(70) $NNR_D(\vec{v}) = D(nn_D(\vec{v}))$

In Wort: $NNR_D : V \to C$ ist die nearest neighbour regression über D, und diese (vollständige) Funktion funktioniert, indem sie für jeden Eingabevektor $\vec{w}$ zunächst den (nach Metrik d nächstliegenden Vektor $\vec{v}$ im Datensatz findet (ein endliches Suchproblem), um ihm dann dieselbe Klasse zuzuordnen, die auch $\vec{v}$ zugeordnet wird.

Ein einfaches Beispiel wäre z.B. eine Sprach- oder Bilderkennung: bei Bildern wäre das Pixelbild ein Vektor, in dem jeder Pixel eine Komponente ist, und der Wert ist die Farbe der Pixel. Bei Spracherkennung kann man die verschiedenen Frequenzbereiche F1-F3 auf einen Vektor verteilen, mit dem Wert als Frequenz; die Klassifikation wäre dann die Frage: welcher Gegenstand ist auf dem Bild abgebildet (aus einer endlichen Menge), bzw. welcher laut wird geformt? NNR ist einfach folgende Methode: finde den ähnlichsten Punkt in unserem Datensatz und klassifiziere entsprechend.

Was ganz interessant ist: NNR kann nicht durch eine lineare Funktion beliebiger Ordnung modelliert werden. Das sieht man sehr leicht an einem Beispiel: man nehme einen Datensatz

$$D = \{((0,0),a),((0,1),b),((0,2),a),((0,3),b),...\}$$

NNR wird hier beliebig oft wechseln können zwischen a und b; jedes lineare Modell, wie z.B. bei logistischer Regression, wird nur eine konstante Anzahl von wechseln ermöglichen können. Allerdings gilt das nur *apriori* und ohne Trainingsdaten: denn wenn wir eine NNR haben zusammen mit einem Datensatz D, dann wird es auch nur eine konstante Zahl an wechseln geben, einfach weil die Datenmenge endlich ist!

Wir sehen hier aber, was ein Modell des maschinellen Lernens ausmacht: es ist eine Funktion von (Trainings)Daten zu einer (Klassifikations- oder Regressions-)Funktion.

$$\text{(Abstraktes) ML-Modell} \underset{Daten}{\Longrightarrow} \text{(Konkrete) Funktion} \underset{Eingabe}{\Longrightarrow} \text{Ausgabe}$$

Übrigens gilt auch bei dieser Methode: man sollte die Daten aufspalten in Trainings- und Testdaten, um festzustellen ob die Methodik angemessen ist. Das kann in manchen Fällen der Fall sein, in manchen nicht.

Stunden gelernt	4	5	6	7	8	9	10
Bestanden	0	0	1	0	1	1	1

Wenn wir wiederum diese Tabelle betrachten, dann sehen wir wir $NNR(5.6) = 1$ haben, $NNR(7.4) = 0$. Hier sehen wir das zentrale Merkmal: *NNR generalisiert sehr wenig.* Der Vorteil, dass wir keine weiteren Annahmen in die Daten stecken, ist also auch zugleich ein Nachteil, denn das Modell ist sehr anfällig für Unregelmäßigkeiten in den Daten, da es sie 1 zu 1 übernimmt. Das kann in manchen Fällen gut sein, insbesondere wenn es viel Varianz in den Daten gibt; das lässt sich aber schwer apriori sagen.