AGML, Sommersemester 2020 Vorlesung 1 (4 Seiten)

PAC Learning

Thomas Kesselheim Letzte Aktualisierung: 22. April 2020

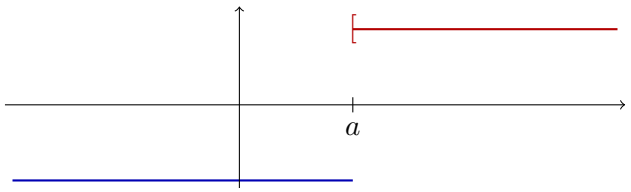
Zum Einstieg betrachten wir *binäre Klassifikation*. Das heißt, wir müssen Datenpunkte klassifizieren nach „positiv“ oder „negativ“. Einige korrekt klassifizierte Punkte sind uns gegeben. Es könnte sich also beispielsweise um E-Mails handeln, bei denen wir automatisch „Spam“ und „Nicht-Spam“ unterscheiden wollen. Diese Beschriftung ist ein *Label*.

1 Schwellenwertfunktionen

Unser erstes Beispiel nimmt stark vereinfachend an, dass jeder Datenpunkt nur ein einziges Merkmal $x \in X := \mathbb{R}$ hat, das eine reelle Zahl ist. Die jeweilige Ausprägung des Merkmals charakterisiert einen Datenpunkt perfekt: Wann immer $x \geq a$ ist, handelt es sich um einen positiv zu klassifizierenden Punkt, ansonsten um einen, der negativ zu klassifizieren ist.

Das heißt, wir können eine Funktion $f\colon X \to \{-1, +1\}$ angeben, die die korrekten Labels beschreibt. Sie lautet

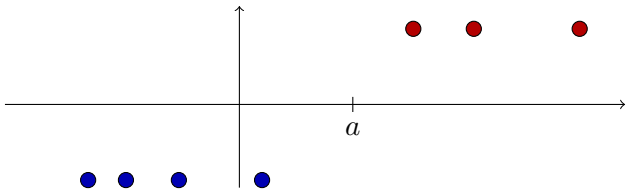
$$f(x) = \begin{cases} +1 & \text{falls } x \geq a \\ -1 & \text{sonst} \end{cases}$$



Diese Funktion nennen wir *Grundwahrheit* (*ground truth*).

Obwohl die Struktur der korrekten Labels sehr einfach ist, ist die Aufgabe nicht trivial. Wir kennen nämlich a nicht. Uns werden lediglich m Datenpunkte mit korrekten Labels gegeben. Die zentrale Frage ist: Wie groß muss m sein, damit wir neue Datenpunkte einigermaßen zuverlässig klassifizieren können?

Ein Beispiel mit $m = 7$ könnte also so aussehen:



2 Hypothesen und Fehler

Konkreter nehmen wir an, dass die Datenpunkte x aus irgendeiner Wahrscheinlichkeitsverteilung $\mathcal{D}$ gezogen werden. Unseren *Hypothesenraum* bezeichnen wir mit $\mathcal{H}$. In diesem Fall ist $\mathcal{H}$ die Menge aller Funktionen der Form $h_{a'}\colon \mathbb{R} \to \{-1, +1\}$ mit

$$h_{a'}(x) = \begin{cases} +1 & \text{falls } x \geq a' \\ -1 & \text{sonst} \end{cases}$$

Unser Ziel ist es, eine Hypothese h mit möglichst kleinem Fehler $\mathrm{err}_{\mathcal{D},f}(h)$ zu finden. Dieser ist wie folgt definiert.

Definition 1.1. *Der* tatsächliche Fehler *(oder* tatsächliches Risiko*)* $\mathrm{err}_{\mathcal{D},f}(h)$ *einer Hypothese* h *hinsichtlich einer Wahrscheinlichkeitsverteilung* $\mathcal{D}$ *über Datenpunkte und Grundwahrheit* f *ist*

$$\mathrm{err}_{\mathcal{D},f}(h) := \mathbf{Pr}_{x\sim\mathcal{D}}\left[h(x) \neq f(x)\right] \ .$$

Beispiel 1.2. *Sei* $\mathcal{D}$ *die uniforme Verteilung auf* $[0,1]$*. Der tatsächliche Fehler einer Hypothese* $h_{a'}$ *ist* $\mathrm{err}_{\mathcal{D},f}(h_{a'}) = |a - a'|$*, wenn* $a, a', \in [0,1]$*. Da* a *jedoch im Allgemeinen nicht bekannt ist, kann dieser jedoch von einem Lernalgorithmus nicht berechnet werden.*

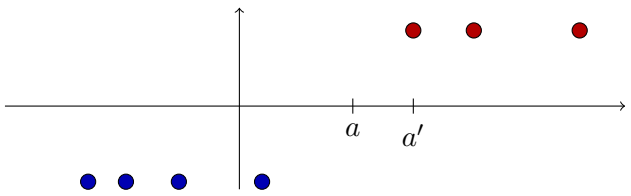
In diesem Beispiel und auch im Folgenden vereinfachen wir uns das Leben durch die Annahme, dass die Grundwahrheit *realisierbar* ist. Das heißt, dass $f \in \mathcal{H}$. Dies ist in unserem Beispiel natürlich erfüllt. In der Realität hingegen sind die Hypothesenklassen meist nicht mächtig genug, um alle Datenpunkt richtig zu klassifizieren.

3 Lernen mit Samples

Wie finden wir also eine Hypothese h, die den tatsächlichen Fehler $\mathrm{err}_{\mathcal{D},f}(h)$ möglichst klein hält? Wir nehmen an, dass uns m Datenpunkte mit korrekten Labels gegeben sind. Seien also $x_1, \ldots, x_m$ Datenpunkten, die unabhängig und identisch verteilt aus $\mathcal{D}$ gezogen sind. Außerdem seien $y_1 = f(x_1), \ldots, y_m = f(x_m)$ die zugehörigen korrekten Labels. Die Menge aller Samples bezeichnen wir mit $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$.

Ein einfacher Lernalgorithmus wählt nun das größte a', sodass $h_{a'}$ die Menge S korrekt klassifiziert. Das heißt, wir setzen a' auf den Wert des kleinsten x_i mit $y_i = 1$, wenn es ein solches gibt. Anderenfalls $a' = \infty$.

In unserem Beispiel sieht das so aus:

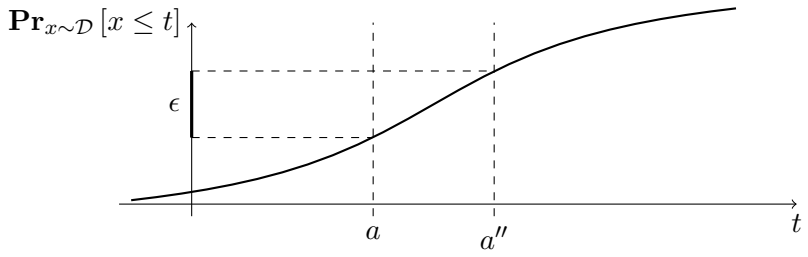


Satz 1.3. *Sei* $h_{a'}$ *die vom einfachen Lernalgorithmus berechnete Hypothese, der als Eingabe ein Sample von* m *Datenpunkten mit korrekten Labels gemäß* f *erhält, die unabhängig und identisch verteilt aus* $\mathcal{D}$ *gezogen werden. Dann gilt für alle* $\epsilon > 0$*, dass* $\mathbf{Pr}\left[\mathrm{err}_{\mathcal{D},f}(h_{a'}) \geq \epsilon\right] \leq \mathrm{e}^{-\epsilon m}$ *.*

Beweis. Weil unser Lernalgorithmus das größte a' wählt, wird auf jeden Fall gelten, dass $a' \geq a$. Falsch klassifiziert werden alle Punkte im Bereich $[a, a')$. Somit gilt nun für jede Verteilung $\mathcal{D}$

$$\mathrm{err}_{\mathcal{D},f}(h_{a'}) = \mathbf{Pr}_{x\sim\mathcal{D}}\left[x \in [a, a')\right] \ .$$

Sei nun a'' die kleinste Zahl, sodass $\mathbf{Pr}_{x\sim\mathcal{D}}\left[x \in [a, a'']\right] \geq \epsilon$. Der Fehler von $h_{a'}$ wird also höchstens ϵ sein, falls $a' \leq a''$. Dies geschieht, wenn es mindestens ein i gibt, sodass $x_i \in [a, a'']$. Sei $\mathcal{E}_i$ das Ereignis, dass $x_i \in [a, a'']$. Es gilt $\mathbf{Pr}\left[\mathcal{E}_i\right] \geq \epsilon$. (Für den Fall einer stetigen Verteilung gilt hier Gleichheit.)



Damit *nicht* $a' \le a''$ gilt, darf keines der Ereignisse $\mathcal{E}_i$ eintreten. Wir interessieren uns also für $\bigcap_i \bar{\mathcal{E}}_i$. Weil $x_1, \dots, x_m$ unabhängige Züge aus $\mathcal{D}$ sind, gilt nun auch

$$\mathbf{Pr}\left[\bigcap_i \bar{\mathcal{E}}_i\right] = \prod_i \mathbf{Pr}\left[\bar{\mathcal{E}}_i\right] \le (1-\epsilon)^m \quad .$$

Wir können nun die Abschätzung $1 - x \le \mathrm{e}^{-x}$ für alle $x \in \mathbb{R}$ verwenden. Somit erhalten wir insgesamt die Behauptung. □

Satz 1.3 sagt uns nun insbesondere, dass wenn wir $m = \frac{1}{\epsilon}\ln\left(\frac{1}{\delta}\right)$ wählen, die Wahrscheinlichkeit, dass der tatsächliche Fehler unserer gefundenen Hypothese größer als ϵ ist, kleiner als δ wird.

4 PAC-Lernbarkeit

Diese Aussage gilt für alle $\epsilon > 0$ und alle $\delta > 0$. Wenn die Anzahl der Samples also nur groß genug ist, werden wir mit großer Wahrscheinlichkeit nur einen sehr kleinen Fehler haben. Die Hypothesenklassen, für die dies gilt, heißen PAC-lernbar.

Definition 1.4. *Eine Hypothesenklasse $\mathcal{H}$ heißt* PAC-lernbar *(im realisierbaren Sinn), wenn es eine Funktion $m_{\mathcal{H}}$ und einen Lernalgorithmus $\mathcal{A}$ gibt, sodass der Algorithmus für alle $\epsilon, \delta > 0$, jede Verteilung $\mathcal{D}$ und alle $f \in \mathcal{H}$, gegeben ein Sample S von Größe mindestens $m_{\mathcal{H}}(\epsilon, \delta)$ von Datenpunkten mit korrekten Labels, eine Hypothese $h_S \in \mathcal{H}$ berechnet, sodass $\mathbf{Pr}\left[\mathrm{err}_{\mathcal{D},f}(h_S) < \epsilon\right] \ge 1 - \delta$.*

Ferner heißt sie effizient PAC-lernbar*, wenn es einen Polynomialzeitalgorithmus $\mathcal{A}$ mit obiger Eigenschaft gibt.*

PAC steht für „probably approximately correct“. „Probably“ bedeutet in diesem Fall, dass die Wahrscheinlichkeit mindestens $1 - \delta$ ist, „approximately correct“ bezieht sich darauf, dass $\mathrm{err}_{\mathcal{D},f}(h_S) < \epsilon$.

Nicht jede Hypothesenklasse ist PAC-lernbar. Zum Beispiel ist die Klasse aller Hypothesen $\mathbb{N} \to \{-1, +1\}$ nicht PAC-lernbar. Dies werden wir im Laufe der Vorlesung beweisen.

5 Weiteres Beispiel: Lernen von Intervallen

Nun betrachten wir als Hypothesenklasse $\mathcal{H}$ die Menge aller Funktionen der Form $h_{a',b'} \colon \mathbb{R} \to \{-1, +1\}$ mit

$$h_{a',b'}(x) = \begin{cases} +1 & \text{falls } x \in [a', b'] \\ -1 & \text{sonst} \end{cases}$$

Wir sind wieder im realisierbaren Fall. Das heißt, es gibt eine Grundwahrheit $f \in \mathcal{H}$, die alle Datenpunkte richtig klassifiziert. Nun also

$$f(x) = \begin{cases} +1 & \text{falls } x \in [a, b] \\ -1 & \text{sonst} \end{cases}$$

Gegeben ist wieder eine Menge $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ von Datenpunkten mit korrekten Labels. Unser Lernalgorithmus wählt das kleinste Intervall $[a', b']$, das S korrekt klassifiziert. Das heißt, wir setzen a' auf den Wert des kleinsten x_i mit $y_i = 1$ und b' auf den Wert des größten x_i mit $y_i = 1$. Den Fall, dass es kein i mit $y_i = 1$ gibt, ignorieren wir.

Satz 1.5. *Für alle $\epsilon > 0$ gilt* $\mathbf{Pr}\left[\mathrm{err}_{\mathcal{D},f}(h_{a',b'}) \geq \epsilon\right] \leq 2\mathrm{e}^{-\frac{\epsilon m}{2}}$.

Beweis. Weil unser Lernalgorithmus das kleinste Intervall wählt, wird auf jeden Fall gelten, dass $a' \geq a$ und $b' \leq b$. Falsch klassifiziert werden alle Punkte im Bereich $[a, a') \cup (b', b]$. Somit gilt nun für jede Verteilung $\mathcal{D}$

$$\mathrm{err}_{\mathcal{D},f}(h_{a',b'}) = \mathbf{Pr}_{x\sim\mathcal{D}}\left[x \in [a, a') \cup (b', b]\right] \ .$$

Sei außerdem ähnlich wie oben a'' die kleinste Zahl, sodass $\mathbf{Pr}_{x\sim\mathcal{D}}[x \in [a, a'']] \geq \frac{\epsilon}{2}$. Analog sei b'' die größte Zahl, sodass $\mathbf{Pr}_{x\sim\mathcal{D}}[x \in [b'', b]] \geq \frac{\epsilon}{2}$. Damit $\mathrm{err}_{\mathcal{D},f}(h_{a',b'}) \leq \epsilon$ ist es nun hinreichend, dass $a' \leq a''$ und $b' \geq b''$. Dies geschieht, wenn es je mindestens ein i gibt, sodass $x_i \in [a, a'']$ bzw. $x_i \in [b'', b]$.

Für jedes i gilt

$$\mathbf{Pr}\left[x_i \in [a, a'']\right] \geq \frac{\epsilon}{2} \quad \text{und} \quad \mathbf{Pr}\left[x_i \in [b'', b]\right] \geq \frac{\epsilon}{2} \ .$$

Weil $x_1, \ldots, x_m$ unabhängige Züge aus $\mathcal{D}$ sind, gilt nun auch

$$\mathbf{Pr}\left[x_1, \ldots, x_m \notin [a, a'']\right] \leq \left(1 - \frac{\epsilon}{2}\right)^m \quad \text{und} \quad \mathbf{Pr}\left[x_1, \ldots, x_m \notin [b'', b]\right] \leq \left(1 - \frac{\epsilon}{2}\right)^m \ .$$

Damit gilt auch

$$\mathbf{Pr}\left[x_1, \ldots, x_m \notin [a, a''] \text{ oder } x_1, \ldots, x_m \notin [b'', b]\right] \leq 2\left(1 - \frac{\epsilon}{2}\right)^m \ ,$$

wobei wir die Abschätzung $\mathbf{Pr}[\mathcal{E} \cup \mathcal{F}] \leq \mathbf{Pr}[\mathcal{E}] + \mathbf{Pr}[\mathcal{F}]$ für zwei Ereignisse $\mathcal{E}$ und $\mathcal{F}$ verwendet haben.

Die Behauptung folgt nun wieder mit der Abschätzung $1 - x \leq \mathrm{e}^{-x}$ für alle $x \in \mathbb{R}$. □

Referenzen

- Foundations of Machine Learning, Kapitel 2.1
- Siehe auch die Vorlesungsskripte von Anna Karlin `https://courses.cs.washington.edu/courses/cse522/17sp/` und Avrim Blum `http://www.cs.cmu.edu/~avrim/ML14/`. Diese enthalten weitere Referenzen.

AGML, Sommersemester 2020 Vorlesung 2 (3 Seiten)

Wachstumsfunktion

Thomas Kesselheim Letzte Aktualisierung: 24. April 2020

1 Wiederholung: PAC-lernbar (Realisierbarer Fall)

Unsere Aufgabe ist es, Datenpunkte aus einer Menge X zu klassifizieren, beispielsweise $X \subseteq \mathbb{R}$. Die Label werden binär sein, das heißt -1 oder 1. Beispielsweise könnte X die Menge aller E-Mails sein und die Labels habe die Bedeutung „nicht Spam“ oder „Spam“. Unser Ziel ist es, dass wir für jeden Datenpunkt x, den wir als Eingabe erhalten, das korrekte Label $y \in \{-1, 1\}$ vorhersagen zu können.

Es gibt eine Klasse von Hypothesen $\mathcal{H}$. Jede hat die Form $h\colon X \to \{-1, 1\}$. Wir nehmen an, dass wir im *realisierbaren Fall* sind. Das heißt, es gibt eine Grundwahrheit $f \in \mathcal{H}$, die eine unserer möglichen Hypothesen ist, und das korrekte Label für $x \in X$ ist immer $f(x)$. Wir möchte nun eine Funktion $h \in \mathcal{H}$ finden, die möglichst ähnlich zum korrekten f ist. Dafür steht uns aber nur eine begrenzte Anzahl Samples mit korrekten Labels zur Verfügung.

Wir erinnern uns an die Definition von PAC-Lernbarkeit.

Definition 2.1. *Eine Hypothesenklasse $\mathcal{H}$ heißt* PAC-lernbar *(im realisierbaren Sinn), wenn es eine Funktion $m_{\mathcal{H}}$ und einen Lernalgorithmus $\mathcal{A}$ gibt, sodass der Algorithmus für alle $\epsilon, \delta > 0$, jede Verteilung $\mathcal{D}$ und alle $f \in \mathcal{H}$, gegeben ein Sample S von Größe mindestens $m_{\mathcal{H}}(\epsilon, \delta)$ von Datenpunkten mit korrekten Labels, eine Hypothese $h_S \in \mathcal{H}$ berechnet, sodass $\mathbf{Pr}\left[\mathrm{err}_{\mathcal{D},f}(h_S) < \epsilon\right] \geq 1 - \delta$.*

Hierbei ist $\mathrm{err}_{\mathcal{D},f}(h) := \mathbf{Pr}_{x\sim\mathcal{D}}\left[h(x) \neq f(x)\right]$ der tatsächliche Fehler von h. Zwei Beispiele dafür haben wir bereits gesehen. Heute wollen wir uns das Thema etwas allgemeiner anschauen.

2 Minimierung des Trainingsfehlers

Wir werden uns allgemeiner Algorithmen anschauen, die den Trainingsfehler minimieren.

Definition 2.2. *Der* Trainingsfehler *(oder* empirisches Risiko*)* $\mathrm{err}_S(h)$ *einer Hypothese h hinsichtlich einer Trainingsmenge S ist*

$$\mathrm{err}_S(h) := \frac{1}{m}|\{h(x_i) \neq y_i\}| \ .$$

Im realisierbaren Fall gilt für die Grundwahrheit f immer $\mathrm{err}_S(f) = 0$ für alle S. Unsere Algorithmen aus der letzten Vorlesung berechneten jedoch auch jeweils Hypothesen h, sodass $\mathrm{err}_S(h) = 0$. Auch diese minimieren also den Trainingsfehler. Unsere Frage heute wird sein, den tatsächlichen Fehler von Hypothesen zu beschränken, die den Trainingsfehler minimieren.

3 Endliche Hypothesenklassen

Wir betrachten zunächst den einfachen Fall, dass die Menge $\mathcal{H}$ endlich ist, wenn auch ansonsten beliebig.

Satz 2.3. *Wenn $m \geq \frac{1}{\epsilon}\ln\left(\frac{|\mathcal{H}|}{\delta}\right)$, dann gilt mit Wahrscheinlichkeit mindestens $1 - \delta$, dass alle $h \in \mathcal{H}$ mit $\mathrm{err}_S(h) = 0$ auch $\mathrm{err}_{\mathcal{D},f}(h) < \epsilon$ erfüllen.*

Beweis. Wir betrachten zunächst ein festes $h \in \mathcal{H}$ mit $\mathrm{err}_{\mathcal{D},f}(h) \geq \epsilon$, das heißt, der tatsächliche Fehler von h ist mindestens ϵ. Nun gilt

$$\begin{aligned}\mathbf{Pr}\left[\mathrm{err}_S(h) = 0\right] &= \mathbf{Pr}\left[h(x_1) = y_1, \ldots, h(x_m) = y_m\right] \\ &= \mathbf{Pr}\left[h(x_1) = y_1\right] \cdot \ldots \cdot \mathbf{Pr}\left[h(x_m) = y_m\right] \leq (1-\epsilon)^m \leq \mathrm{e}^{-\epsilon m} \ .\end{aligned}$$

Das heißt, dass die Wahrscheinlichkeit, dass h keinen Trainingsfehler hat, höchstens $\mathrm{e}^{-\epsilon m}$ ist.

Um die Gesamtwahrscheinlichkeit zu beschränken, dass es irgendeine Hypothese gibt, die zwar keinen Trainingsfehler, aber großen tatsächlichen Fehler hat, benutzen wir die sogenannte Union Bound.

Lemma 2.4 (Union Bound)**.** *Es seien $\mathcal{E}_1, \ldots, \mathcal{E}_n$ (nicht notwendigerweise disjunkte) Ereignisse. Dann gilt*

$$\mathbf{Pr}\left[\bigcup_{i=1}^{n} \mathcal{E}_i\right] \leq \sum_{i=1}^{n} \mathbf{Pr}\left[\mathcal{E}_i\right] \ .$$

Der Beweis der Union Bound folgt durch induktive Anwendung von $\mathbf{Pr}\left[A \cup B\right] = \mathbf{Pr}\left[A\right] + \mathbf{Pr}\left[B\right] - \mathbf{Pr}\left[A \cap B\right] \leq \mathbf{Pr}\left[A\right] + \mathbf{Pr}\left[B\right]$.

Um nun die Union Bound anzuwenden, definieren wir für jede Hypothese $h \in \mathcal{H}$ das Ereignis $\mathcal{E}_h$, dass $\mathrm{err}_S(h) = 0$.

Nun gilt

$$\begin{aligned}\mathbf{Pr}\left[\exists h \in \mathcal{H} : \mathrm{err}_{\mathcal{D},f}(h) \geq \epsilon \text{ und } \mathrm{err}_S(h) = 0\right] &= \mathbf{Pr}\left[\bigcup_{h \in \mathcal{H}, \mathrm{err}_{\mathcal{D},f}(h) \geq \epsilon} \mathcal{E}_h\right] \\ &\leq \sum_{h \in \mathcal{H}, \mathrm{err}_{\mathcal{D},f}(h) \geq \epsilon} \mathbf{Pr}\left[\mathrm{err}_S(h) = 0\right] \\ &\leq |\mathcal{H}| \mathrm{e}^{-\epsilon m} \leq \delta \ . \qquad \square\end{aligned}$$

4 Wachstumsfunktion

Dieses Ergebnis nützt uns natürlich nichts, wenn $\mathcal{H}$ unendlich ist. Wir haben allerdings schon Beispiele gesehen, dass auch unendliche Hypothesenklassen PAC-lernbar sein können, beispielsweise die Schwellenwertfunktionen. Diese haben eine Struktur, die wir ausnutzen können. Dies können wir wie folgt formalisieren.

Definition 2.5. *Gegeben $S \subseteq X$, sei $\mathcal{H}|_S$ die Menge aller Hypothesen $h \in \mathcal{H}$ mit Definitionsbereich eingeschränkt auf S. Das heißt, $\mathcal{H}|_S = \{h|_S \mid h \in \mathcal{H}\}$.*

Die Wachstumsfunktion *von $\mathcal{H}$ ist definiert als $\Pi_{\mathcal{H}}(m) = \max_{S \subseteq X, |S|=m} |\mathcal{H}|_S|$.*

Weil die Abbildungen in $\mathcal{H}|_S$ von S nach $\{-1, +1\}$ abbilden, können es nicht mehr als 2^m verschiedene sein, weil es nicht mehr Abbildungen gibt. Somit muss immer $\Pi_{\mathcal{H}}(m) \leq 2^m$ gelten. Häufig sind die Werte von $\Pi_{\mathcal{H}}$ jedoch viel kleiner.

Beispiel 2.6. *Betrachte $X = \mathbb{R}$ und $\mathcal{H}$ als die Klasse der Schwellenwertfunktionen*

$$h_{a'}(x) = \begin{cases} +1 & \textit{falls } x \geq a' \\ -1 & \textit{sonst} \end{cases}$$

Für $S = \{2, 3, 4\}$ besteht $\mathcal{H}|_S$ aus folgenden vier Funktionen:

$$x \mapsto -1 \quad \textit{für alle } x \qquad\qquad x \mapsto +1 \quad \textit{für alle } x$$

$$x \mapsto \begin{cases} -1 & \textit{für } x = 2 \textit{ oder } x = 3 \\ +1 & \textit{für } x = 4 \end{cases} \qquad x \mapsto \begin{cases} -1 & \textit{für } x = 2 \\ +1 & \textit{für } x = 3 \textit{ oder } x = 4 \end{cases}$$

Es gibt noch vier weitere Funktionen $\{2, 3, 4\} \to \{-1, +1\}$. Diese lassen sich aber nicht über einen Schwellenwert realisieren.

Allgemein gilt $\Pi_{\mathcal{H}}(m) = m + 1$, denn es gibt nur $m + 1$ mögliche „Umschaltpunkte“ von -1 auf $+1$. Das heißt, die Funktion wächst deutlich schwächer als 2^m.

Der folgende Satz zeigt, dass wir in der Aussage von Satz 2.3 im Wesentlichen die Größe von $\mathcal{H}$ durch die Wachstumsfunktion ersetzen können.

Satz 2.7. *Es seien $\epsilon > 0$ und $\delta > 0$ beliebig und*

$$m \geq \max\left\{\frac{8}{\epsilon}, \frac{2}{\epsilon}\log_2\left(\frac{2\Pi_{\mathcal{H}}(2m)}{\delta}\right)\right\} \ . \tag{1}$$

Betrachte ein Sample S von m Datenpunkten mit korrekten Labels gemäß f gezogen unabhängig und identisch verteilt aus $\mathcal{D}$. Es gilt mit Wahrscheinlichkeit mindestens $1 - \delta$, dass alle $h \in \mathcal{H}$ mit $\mathrm{err}_S(h) = 0$ auch $\mathrm{err}_{\mathcal{D},f}(h) < \epsilon$ erfüllen.

Bevor wir mit dem Beweis dieses Satzes beginnen, schauen wir uns zunächst die Aussage etwas genauer an. Sie hat grundsätzlich die Struktur der Aussage, wie wir sie für PAC-Lernbarkeit brauchen. Wenn m Bedingung (1) erfüllt, dann führt beliebiger Lernalgorithmus, der den Trainingsfehler minimiert, zu einem tatsächlichen Fehler von höchstens ϵ mit Wahrscheinlichkeit mindestens $1 - \delta$.

Wann gilt jedoch Bedingung 1 und wann ist sie überhaupt für alle ϵ und δ erfüllbar? Schauen wir uns nur noch $m \geq \frac{8}{\epsilon}$ an, dann brauchen wir noch

$$m \geq \frac{2}{\epsilon}\log_2\left(\frac{2\Pi_{\mathcal{H}}(2m)}{\delta}\right) = \frac{2}{\epsilon}\log_2\left(\Pi_{\mathcal{H}}(2m)\right) + \frac{2}{\epsilon}\log_2\left(\frac{2}{\delta}\right) \quad \Leftrightarrow \quad \frac{m - \log_2\left(\frac{2}{\delta}\right)}{\log_2\left(\Pi_{\mathcal{H}}(2m)\right)} \geq \frac{2}{\epsilon} \ .$$

Wenn $\Pi_{\mathcal{H}}(2m) = 2^{2m}$ (die triviale Schranke), dann ist $\log_2\left(\Pi_{\mathcal{H}}(2m)\right) = 2m$. Die Ungleichung ist also für sinnvolle ϵ (d.h. $\epsilon < 1$) nicht erfüllbar.

Wächst hingegen $\log_2\left(\Pi_{\mathcal{H}}(2m)\right)$ schwächer als m, das heißt, $\log_2\left(\Pi_{\mathcal{H}}(2m)\right) = o(m)$, dann muss m nur ausreichend groß genug gewählt werden, um die Schranke zu erfüllen.

Im Beispiel mit den Schwellenwertfunktionen ist dies der Fall. Es gilt $\Pi_{\mathcal{H}}(2m) = 2m + 1$. Nun gilt also für alle $\delta > 0$, dass

$$\frac{m - \log_2\left(\frac{2}{\delta}\right)}{\log_2\left(\Pi_{\mathcal{H}}(2m)\right)} = \frac{m - \log_2\left(\frac{2}{\delta}\right)}{\log_2(2m + 1)} \to \infty \quad \text{für } m \to \infty \ .$$

Egal, wie ϵ und δ als gewählt sind, für genügend große m ist Bedingung 1 immer erfüllt.

AGML, Sommersemester 2020 Vorlesung 3 (6 Seiten)

Mengensysteme

Anne Driemel Letzte Aktualisierung: 2. Mai 2020

In den letzten Vorlesungen haben wir uns mit PAC-Lernbarkeit unter Annahme fester Hypothesenklassen beschäftigt. Wir haben gesehen, dass die Struktur einer Hypothesenklasse auch etwas über die Lernbarkeit aussagt, sofern die Hypothesenklasse realisierbar ist. In dieser Vorlesung werden wir uns mit der Struktur der Hypothesenklassen aus der Sicht von Mengensystemen befassen und allgemeine Eigenschaften ableiten. Wir nehmen dabei noch stets die Realisierbarkeit der Hypothesenklasse an.

1 Mengensysteme

Definition 3.1 (Mengensystem)**.** *Sei $\mathcal{X}$ eine beliebige Menge und $\mathcal{R}$ eine Menge von Teilmengen von $\mathcal{X}$. Wir nennen $\mathcal{R}$ ein Mengensystem mit Grundmenge $\mathcal{X}$.*

Jede Hypothesenklasse $\mathcal{H}$, definiert durch eine Menge von Funktionen der Form

$$h\colon \mathcal{X} \to \{-1, +1\},$$

kann gleichsam durch ein Mengensystem beschrieben werden. Wir definieren für jede Funktion $h \in \mathcal{H}$ eine Menge

$$r_h = \{\ x \in \mathcal{X}\ \mid\ h(x) = 1\ \}\,,$$

welche also genau der positiven Menge entspricht. Die Menge aller Mengen r_h bildet dann das Mengensystem.

Beispiel 3.2. *Die Menge aller achsenparallelen Rechtecke in der Ebene definiert ein Mengensystem $\mathcal{R}$ mit Grundmenge $\mathcal{X} = \mathbb{R}^2$. Formal ist jedes Element $r \in \mathcal{R}$ definiert durch ein Tupel (a, b, c, d) mit*

$$r_{a,b,c,d} = \{\ (x, y) \in \mathcal{X}\ \mid\ a \le x \le b, c \le y \le d\ \}\,.$$

Eine wichtige kombinatorische Eigenschaft von Mengensystemen ist ihre VC-dimension, benannt nach Vapnik und Chervonenkis.

Definition 3.3 (Abspalten)**.** *Wir sagen eine Menge $A' \subseteq \mathcal{X}$ wird durch ein Mengensystem $\mathcal{R}$ von einer Menge $A \subseteq \mathcal{X}$ abgespalten, wenn A' durch den Schnitt mit einer Menge von $\mathcal{R}$ erzeugt werden kann. Das heißt, es existiert ein $r \in \mathcal{R}$ mit $A' = r \cap A$.*

Definition 3.4 (Aufspalten)**.** *Eine Menge $A \subseteq \mathcal{X}$ wird durch ein Mengensystem aufgespalten, wenn alle Teilmengen von A abgespalten werden können.*

Definition 3.5 (VC-dimension)**.** *Die VC-dimension von $\mathcal{R}$ ist die Anzahl der Elemente in der größten durch $\mathcal{R}$ aufgespaltenen Menge. Falls keine solche Menge existiert, dann ist die VC-dimension unendlich. Wir bezeichnen die VC-dimension mit* $\dim(\mathcal{R})$*. Für den Sonderfall $\mathcal{R} = \emptyset$ definieren wir* $\dim(\emptyset) = 0$*.*

Schauen wir uns die VC-dimension im oben genannten Beispiel genauer an. Die VC-dimension von $\mathcal{R}$ ist mindestens 4, da wir eine 4-elementige Menge A von Punkten in der Ebene angeben können, die von $\mathcal{R}$ aufgespalten wird. Abbildung 1 zeigt eine solche Menge. Gleichzeitig können wir zeigen, dass für jede 5-elementige Menge A' gilt, dass sie *nicht* durch $\mathcal{R}$ aufgespalten wird.

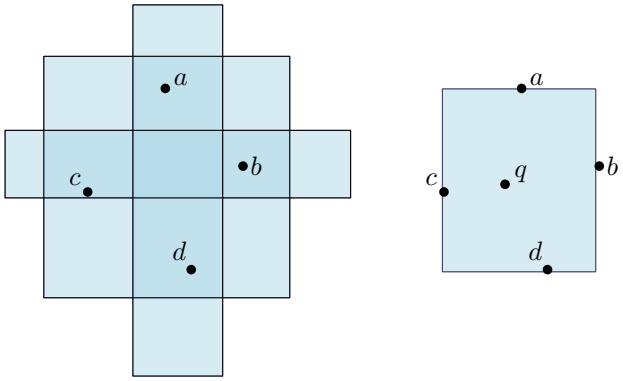


Abbildung 1: (links) Beispielmenge die durch das Mengensystem der achsenparallelen Rechtecke aufgespalten wird. Exemplarisch dargestellt sind auch drei achsenparallele Rechtecke, die verschiedene Teilmengen abspalten. (rechts) Bei fünf Punkten können wir immer einen Punkt q finden, sodass das Komplement nicht durch ein achsenparalleles Rechteck abgespalten werden kann.

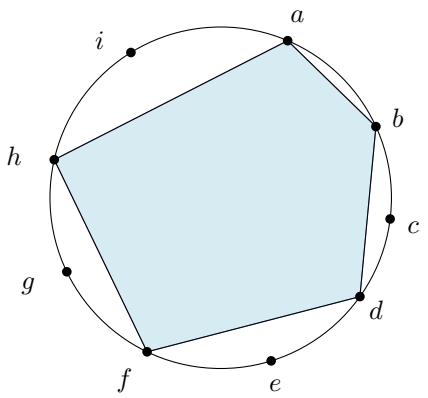


Abbildung 2: Beispielmenge von Punkten die durch das Mengensystem aller konvexer Polygone aufgespalten wird. Exemplarisch dargestellt ein Polygon, das die Teilmenge $\{a, b, d, f, h\}$ abspaltet.

Angenommen alle Koordinaten der Punkte in A sind paarweise verschieden. Da die Menge 5 Punkte enthält, existiert ein Punkt $q \in A$, der weder die x-Koordinate, noch die y-Koordinate in A minimiert oder maximiert. Es folgt, dass q in allen achsenparallelen Rechtecken enthalten ist, welche die Menge $A \setminus \{q\}$ enthalten. Daher existiert keine Menge $r \in \mathcal{R}$, sodass

$$A \setminus \{q\} = A \cap r.$$

Damit ist die VC-dimension des Mengensystems der achsenparallelen Rechtecke genau 4.

Es gibt auch Mengensysteme mit unendlicher VC-dimension. Betrachten wir das Mengensystem aller konvexen Polygone in der Ebene. Konvexe Polygone sind dadurch definiert, dass jeder Innenwinkel höchstens 180° beträgt. Für jede natürliche Zahl n können wir eine n-elementige Menge finden, welche durch dieses Mengensystem aufgespalten wird. Sei A_n eine Menge von n Punkten auf dem Einheitskreis. Jede Teilmenge $A \subseteq A_n$ definiert als Menge von Ecken ein konvexes Polygon P mit der gewünschten Eigenschaft, siehe Abbildung 2. Somit ist die VC-dimension des Mengensystems der konvexen Polygone unendlich.

2 Wachstum von endlichen Mengensystemen

Ein Mengensystem ist endlich, wenn es nur endlich viele Mengen enthält. Wieviele Mengen kann ein Mengensystem mit $\left|\mathcal{X}\right| = m$ enthalten? Allgemein gilt $\left|\mathcal{R}\right| \leq 2^{\left|\mathcal{X}\right|} = 2^m$. Was, wenn die VC-dimension kleiner als m ist?

Beispiel 3.6. *Sei $\mathcal{X} = \{1, 2, \ldots, m\}$ und sei $\mathcal{R}$ das Mengensystem, das alle Teilmengen von maximaler Größe k enthält. Die VC-dimension dieses Mengensystems ist k. Wir können alle generierten Mengen aufzählen und sehen direkt, dass*

$$\left|\mathcal{R}\right| = \sum_{i=0}^{k} \binom{m}{i} \leq \sum_{i=0}^{k} m^k \leq (k+1)m^k.$$

Für ein festes k wächst die Anzahl der Mengen im Beispiel höchstens polynomiell in der Größe des Mengensystems m.

Wir wollen nun eine asymptotische obere Schranke zeigen, die dieses Wachstum im allgemeineren Fall von endlichen Mengensystemen mit endlicher VC-dimension beschreibt. Das folgende Lemma zeigt, dass die VC-dimension das Wachstum in diesem Sinne charakterisiert.

Lemma 3.7. *Es gilt für jedes Mengensystem $\mathcal{R}$ mit m-elementiger Grundmenge $\mathcal{X}$ und VC-dimension* d*, dass*

$$\left|\mathcal{R}\right| \leq \sum_{i=0}^{\mathrm{d}} \binom{m}{i}.$$

Beweis. Wir zeigen den Satz durch Induktion über m mit Induktionsanfang $m = 0$. In diesem Fall kann $\mathcal{R}$ höchstens die leere Menge enthalten, also ist $\left|\mathcal{R}\right| \leq 1$ und $\mathrm{d} \leq 0$. Gleichzeitig gilt per Definition des Binomialkoeffizienten, dass $\binom{0}{0} = 1$. Damit ist die Aussage für den Induktionsanfang erfüllt. Im Induktionsschritt nehmen wir an, dass $m > 0$. Sei $\mathcal{R}$ ein Mengensystem mit Grundmenge $\mathcal{X}$ und VC-dimension d. Nehmen wir an, dass $\mathrm{d} = 0$. In diesem Fall kann man auch zeigen, dass $\left|\mathcal{R}\right| \leq 1$ und die Aussage ist erfüllt. Also nehmen wir an, dass $\mathrm{d} > 0$.

Sei $x \in \mathcal{X}$ fest und betrachte das Mengensystem

$$\mathcal{R}_1 = \{\ r \setminus \{x\}\ \mid\ r \in \mathcal{R}\ \}.$$

Sei die VC-dimension d_1. Beachte, dass $\mathrm{d}_1 \leq \mathrm{d}$ ist, da jede Menge $A \subseteq \mathcal{X} \setminus \{x\}$ die durch $\mathcal{R}_1$ aufgespalten wird, auch durch $\mathcal{R}$ aufgespalten wird.

Nun folgt aus der Induktionsannahme, dass

$$\left|\mathcal{R}_1\right| \leq \sum_{i=0}^{\mathrm{d}_1} \binom{m-1}{i} \leq \sum_{i=0}^{\mathrm{d}} \binom{m-1}{i}.$$

Allerdings könnte es sein, dass zwei verschiedene Mengen in $\mathcal{R}$ durch die Beschränkung auf $\mathcal{X} \setminus \{x\}$ identisch werden und dadurch $\left|\mathcal{R}_1\right|$ strikt kleiner ist als $\left|\mathcal{R}\right|$. Wir definieren ein zweites Mengensystem um genau diese Paare von Mengen zu zählen, wie folgt

$$\mathcal{R}_2 = \{\ r \setminus \{x\}\ \mid\ r \setminus \{x\} \in \mathcal{R} \text{ und } r \cup \{x\} \in \mathcal{R}\ \}.$$

Es folgt nun, dass

$$\left|\mathcal{R}\right| = \left|\mathcal{R}_1\right| + \left|\mathcal{R}_2\right|.$$

Sei $\mathrm{d}_2 = \dim(\mathcal{R}_2)$. Wir behaupten, dass $\mathrm{d}_2 \leq \mathrm{d}-1$. Angenommen, dem wäre nicht so und die VC-dimension wäre mindestens d. Dann existierte eine Menge $A \subseteq \mathcal{X} \setminus \{x\}$ mit $|A| = \mathrm{d}$, sodass A durch $\mathcal{R}_2$ aufgespalten wird. Dann würde auch die Menge $A \cup \{x\}$ durch $\mathcal{R}$ aufgespalten, denn $\mathcal{R}_2$ enthält ja nur solche Paare von Mengen aus $\mathcal{R}$, die bis auf x identisch sind. Das würde aber der Grundannahme widersprechen, dass die VC-dimension von $\mathcal{R}$ gleich d ist.

Somit gilt nach Induktionsannahme, dass

$$|\mathcal{R}_2| \leq \sum_{i=0}^{\mathrm{d}_2} \binom{m-1}{i} \leq \sum_{i=0}^{\mathrm{d}-1} \binom{m-1}{i} = \sum_{j=1}^{\mathrm{d}} \binom{m-1}{j-1}$$

Durch Einsetzen in die obige Gleichung bekommen wir

$$|\mathcal{R}| \leq \sum_{i=0}^{\mathrm{d}} \binom{m-1}{i} + \sum_{j=1}^{\mathrm{d}} \binom{m-1}{j-1} = 1 + \sum_{i=1}^{\mathrm{d}} \binom{m-1}{i} + \binom{m-1}{i-1} = \sum_{i=0}^{\mathrm{d}} \binom{m}{i},$$

wobei die letzte Gleichung aus der rekursiven Darstellung des Binomialkoeffizienten folgt.

□

3 Unendliche Mengensysteme

Wir wollen nun die obige Schranke erweiteren auf unendliche Mengensysteme. Also Mengensysteme, die unendlich viele Mengen enthalten. Insbesondere sind wir interessiert am Wachstum der Anzahl der durch das Mengensystem abgespaltenen Teilmengen. Um das zu formalisieren betrachten wir Untersysteme, die wir wie folgt definieren.

Definition 3.8 (Untersystem)**.** *Sei $\mathcal{R}$ ein Mengensystem mit Grundmenge $\mathcal{X}$. Jede Menge $A \subseteq \mathcal{X}$ bestimmt ein Untersystem von $\mathcal{R}$ wie folgt*

$$\mathcal{R}|_A = \{\ r \cap A \ \mid \ r \in \mathcal{R}\ \}.$$

Das heißt, $\mathcal{R}|_A$ ist ein Mengensystem mit Grundmenge A, welches genau die Teilmengen von A enthält, die von A durch $\mathcal{R}$ abgespalten werden können. Die VC-dimension kann durch die Beschränkung auf ein Untersystem nicht größer werden, also gilt $\dim(\mathcal{R}|_A) \leq \dim(\mathcal{R})$.

Beispiel 3.9. *Wir haben Untersysteme schon kennengelernt, auch wenn wir sie nicht so genannt haben. Insbesondere ist das Mengensystem $\mathcal{R}_1$ aus vorhergehendem Beweis das Untersystem von $\mathcal{R}$ beschränkt auf $\mathcal{X} \setminus \{x\}$, denn,*

$$\mathcal{R}_1 = \{\ r \setminus \{x\} \ \mid \ r \in \mathcal{R}\ \} = \{\ r \cap (\mathcal{X} \setminus \{x\}) \ \mid \ r \in \mathcal{R}\ \} = \mathcal{R}|_{\mathcal{X}\setminus\{x\}}.$$

Satz 3.10 (Wachstumslemma)**.** *Sei $\mathcal{R}$ ein Mengensystem mit Grundmenge $\mathcal{X}$ und VC-dimension* d*. Für jede natürliche Zahl m gilt, dass*

$$\Pi_{\mathcal{R}}(m) = \max_{\substack{A \subseteq \mathcal{X} \\ |A|=m}} \left|\mathcal{R}|_A\right| \leq \left(\frac{\mathrm{e}m}{\mathrm{d}}\right)^{\mathrm{d}}.$$

Wir nennen $\Pi_{\mathcal{R}}$ die Wachstumsfunktion von $\mathcal{R}$.

Beweis. Da die VC-dimension durch die Beschränkung auf ein Untersystem nicht größer werden kann, können wir Lemma 3.7 direkt anwenden und bekommen für jede Menge $A' \subseteq \mathcal{X}$ mit $\left|A'\right| = m$, dass

$$\left|\mathcal{R}|_{A'}\right| \leq \sum_{i=0}^{\mathrm{d}} \binom{m}{i}.$$

Nun machen wir folgende Abschätzung

$$\binom{m}{i} = \frac{m!}{(m-i)! \cdot i!} \leq \frac{m^i}{i!} = \left(\frac{m}{\mathrm{d}}\right)^i \frac{\mathrm{d}^i}{i!} \leq \left(\frac{m}{\mathrm{d}}\right)^{\mathrm{d}} \frac{\mathrm{d}^i}{i!} \ .$$

Zusammen mit der Reihendefinition der Exponentialfunktion $\mathrm{e}^x = \sum_{i=0}^{\infty} \frac{x^i}{i!}$, bekommen wir dann

$$\sum_{i=0}^{\mathrm{d}} \binom{m}{i} \leq \sum_{i=0}^{\mathrm{d}} \left(\frac{m}{\mathrm{d}}\right)^{\mathrm{d}} \frac{\mathrm{d}^i}{i!} = \left(\frac{m}{\mathrm{d}}\right)^{\mathrm{d}} \sum_{i=0}^{\mathrm{d}} \frac{\mathrm{d}^i}{i!} \leq \left(\frac{m}{\mathrm{d}}\right)^{\mathrm{d}} \mathrm{e}^{\mathrm{d}} \ .$$

Da wir die Schranke für jede m-elementige Menge A' zeigen, gilt sie auch für die größte solche Menge. Damit ist der Satz bewiesen. □

4 PAC-Lernbarkeit

Unsere Motivation um Mengensysteme zu studieren war zu Beginn mit der Hoffnung auf bessere Schranken für die PAC-Lernbarkeit von Hypothesenklassen begründet. Sei $\mathcal{H}$ eine Hypothesenklasse und sei $\mathcal{R}$ das entsprechende Mengensystem mit Grundmenge $\mathcal{X}$. Sei $S \subseteq \mathcal{X}$. Schauen wir uns die Definitionen von $\mathcal{R}|_S$ und $\mathcal{H}|_S$ genauer an, sehen wir, dass diese äquivalent im Sinne unserer Abbildung zwischen Hypothesenklassen und Mengensystemen sind. Insbesondere beschreibt die Funktionsmenge $\mathcal{H}|_S$ alle verschiedenen Wege, Labels in $\{-1, +1\}$ für die Menge S zu vergeben mithilfe einer Funktion in $\mathcal{H}$. Im Kontext von Mengensystemen entspricht $\mathcal{R}|_S$ alle verschiedenen Wege, mithilfe einer Menge $r \in \mathcal{R}$ eine Teilmenge von S abzuspalten. Diese Teilmengen entsprechen dann den positiven Teilmengen von S, die sich aus Funktionen in $\mathcal{H}$ ergeben. Somit gilt für die Wachstumsfunktion

$$\Pi_{\mathcal{H}}(m) = \max_{S \subseteq X, |S|=m} \left|\mathcal{H}|_S\right| = \max_{S \subseteq X, |S|=m} \left|\mathcal{R}|_S\right| = \Pi_{\mathcal{R}}(m) \tag{1}$$

Satz 3.11. *Sei $\mathcal{H}$ eine Hypothesenklasse mit VC-dimension* d. *Seien $1 \geq \epsilon > 0$ und $\delta > 0$ beliebig und sei*

$$m \geq \max\left(\frac{4}{\epsilon}\log_2\frac{2}{\delta}, \frac{8\mathrm{d}}{\epsilon}\log_2\frac{16}{\epsilon}\right) \tag{2}$$

Betrachte ein Sample S von m Datenpunkten mit korrekten Labels gemäß f gezogen unabhängig und identisch verteilt aus $\mathcal{D}$. Es gilt mit Wahrscheinlichkeit mindestens $1 - \delta$, dass alle $h \in \mathcal{H}$ mit $\mathrm{err}_S(h) = 0$ auch $\mathrm{err}_{\mathcal{D},f}(h) < \epsilon$ erfüllen.

Beweis. Wir wollen Satz 2.7 aus der letzten Vorlesung anwenden, der besagt, dass die obige Behauptung gilt, sofern die folgende Bedingung für m erfüllt ist.

$$m \geq \max\left\{\frac{8}{\epsilon}, \frac{2}{\epsilon}\log_2\left(\frac{2\Pi_{\mathcal{H}}(2m)}{\delta}\right)\right\} \ . \tag{3}$$

Dafür müssen wir nur zeigen, dass unsere Bedingung (2) die Bedingung (3) impliziert.
Zunächst haben wir, da $\mathrm{d} \geq 1$ und $\epsilon \leq 1$,

$$m \geq \frac{8\mathrm{d}}{\epsilon}\log_2\frac{16}{\epsilon} \implies m \geq \frac{8}{\epsilon}$$

womit der erste Teil von Bedingung (3) gezeigt ist.

Wir behaupten nun, dass aus Bedingung (2) auch folgt, dass

$$m \geq \frac{2}{\epsilon} \log_2 \left(\frac{2}{\delta} \cdot \left(\frac{2\mathrm{e}m}{\mathrm{d}} \right)^{\mathrm{d}} \right). \tag{4}$$

Aus der Gleichheit der Wachstumsfunktionen von Hypothesenklassen und den zugehörigen Mengensystemen in (1) und der Schranke aus dem Wachstumslemma (Satz 3.10) folgt

$$\left(\frac{2\mathrm{e}m}{\mathrm{d}} \right)^{\mathrm{d}} \geq \Pi_{\mathcal{H}}(2m)$$

und somit wäre

$$m \geq \frac{2}{\epsilon} \log_2 \left(\frac{2\Pi_{\mathcal{H}}(2m)}{\delta} \right)$$

Damit wäre auch der zweite Teil der Bedingung 3 gezeigt.

Es bleibt, die Behauptung (2) $\implies$ (4) zu zeigen. Dafür formen wir (4) zunächst wie folgt um.

$$m \geq \frac{2}{\epsilon} \log_2 \frac{2}{\delta} + \frac{2}{\epsilon} \log_2 \left(\left(\frac{2\mathrm{e}m}{\mathrm{d}} \right)^{\mathrm{d}} \right). \tag{5}$$

Zunächst haben wir für die Abschätzung des ersten Terms

$$m \geq \frac{4}{\epsilon} \log_2 \frac{2}{\delta} \implies \frac{m}{2} \geq \frac{2}{\epsilon} \log_2 \frac{2}{\delta} \tag{6}$$

Für die Abschätzung des zweiten Terms zeigen wir

$$\frac{m}{2} \geq \frac{2}{\epsilon} \mathrm{d} \log_2 \left(\frac{2\mathrm{e}m}{\mathrm{d}} \right) \tag{7}$$

Setzen wir zunächst $m = \frac{8\mathrm{d}}{\epsilon} \log_2 \frac{16}{\epsilon}$ auf beiden Seiten ein, sehen wir durch äquivalente Umformung, dass die Ungleichung für $0 < \epsilon \leq 1$ erfüllt ist. Das gilt auch für größere Werte von m. Die Behauptung in (5) folgt nun durch das Addieren der beiden Ungleichungen in (6) und (7). □

Referenzen

- Foundations of Machine Learning, Kapitel 3.3
- Understanding Machine Learning, Kapitel 6.2-6.5 (anderer Beweis!)
- Sariel Har-Peled, Geometric Approximation Algorithms. AMS Mathematical Surveys and Monographs, Band 173. 2011.

AGML, Sommersemester 2020 Vorlesung 4 (5 Seiten)

Wachstumsfunktion und Agnostisches PAC-Lernen

Thomas Kesselheim Letzte Aktualisierung: 5. Mai 2020

1 Erinnerung: Wachstumsfunktion

Wir erinnern uns, dass eine Hypothesenklasse $\mathcal{H}$ eine Menge von Funktionen der Form $h\colon X \to \{-1,+1\}$ ist. Wir haben schon viele Beispiele gesehen, vor allem mit $X = \mathbb{R}$. Diese Hypothesenklassen enthalten jedoch nicht alle Funktionen sondern besitzen deutlich mehr Struktur. Dies wird formalisiert in der Wachstumsfunktion.

Definition 4.1. *Gegeben $S \subseteq X$, sei $\mathcal{H}|_S$ die Menge aller Hypothesen $h \in \mathcal{H}$ mit Definitionsbereich eingeschränkt auf S. Das heißt, $\mathcal{H}|_S = \{h|_S \mid h \in \mathcal{H}\}$.*

Die Wachstumsfunktion *von $\mathcal{H}$ ist definiert als $\Pi_{\mathcal{H}}(m) = \max_{S\subseteq X, |S|=m} |\mathcal{H}|_S|$.*

In der letzten Vorlesung haben wir ein extrem hilfreiches Werkzeug gesehen, um die Wachstumsfunktion zu beschränken: die VC-Dimension. Wir haben bewiesen, dass wenn die VC-Dimension d ist, auch $\Pi_{\mathcal{H}}(m) \leq \left(\frac{em}{\mathrm{d}}\right)^{\mathrm{d}}$ gilt. Das heißt, wenn die VC-Dimension endlich ist, wächst die Wachstumsfunktion nur polynomiell.

2 Subexponentielles Wachstum impliziert PAC-Lernbarkeit

Es steht noch der Beweis des Satzes aus der zweiten Vorlesung aus, dass derartiges subexponentielles Wachstum tatsächlich PAC-Lernbarkeit impliziert. Wir betrachten wieder eine Hypothesenklasse $\mathcal{H}$, eine Grundwahrheit $f \in \mathcal{H}$ und eine beliebige Wahrscheinlichkeitsverteilung $\mathcal{D}$ über X.

Satz 4.2. *Es seien $\epsilon > 0$ und $\delta > 0$ beliebig und*

$$m \geq \max\left\{\frac{8}{\epsilon}, \frac{2}{\epsilon}\log_2\left(\frac{2\Pi_{\mathcal{H}}(2m)}{\delta}\right)\right\} \quad . \tag{1}$$

Betrachte ein Sample S von m Datenpunkten mit korrekten Labels gemäß f gezogen unabhängig und identisch verteilt aus $\mathcal{D}$. Es gilt mit Wahrscheinlichkeit mindestens $1-\delta$, dass alle $h \in \mathcal{H}$ mit $\mathrm{err}_S(h) = 0$ auch $\mathrm{err}_{\mathcal{D},f}(h) < \epsilon$ erfüllen.

Um Satz 4.2 zu zeigen, beweisen wir zunächst zwei Lemmata, die für sich genommen schon interessante Aussagen sind. Erst im Anschluss werden wir sie zum Beweis des Satzes zusammenfügen.

Wir halten zunächst fest, dass es eigentlich gar nicht mal sehr wahrscheinlich ist, dass eine feste Hypothese mit großem tatsächlichen Fehler auch „typischerweise“ einen großen Trainingsfehler hat.

Lemma 4.3. *Sei h eine Hypothese mit $\mathrm{err}_{\mathcal{D},f}(h) \geq \epsilon$ und sei S' eine Menge von m zufällig gezogenen Samples. Falls m Bedingung (1) erfüllt, dann gilt $\mathbf{Pr}\left[\mathrm{err}_{S'}(h) \geq \frac{\epsilon}{2}\right] \geq \frac{1}{2}$.*

Beweis. Wir können uns das Zufallsexperiment vorstellen als m unabhängige Münzwürfe, wobei die Wahrscheinlichkeit für Kopf $p := \mathrm{err}_{\mathcal{D},f}(h) \geq \epsilon$ in jedem Wurf beträgt. Wir behaupten, dass wir mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ mindestens $\frac{\epsilon}{2}m$ mal Kopf sehen.

Sei dazu Z die Anzahl Kopf in den Münzwürfen. Es gelten $\mathbf{E}[Z] = pm$ und $\mathrm{Var}[Z] = p(1-p)m$. Wegen $p \geq \epsilon$ gilt also nach der Tschebyschew-Ungleichung

$$\mathbf{Pr}\left[Z \leq \frac{\epsilon}{2}m\right] \leq \mathbf{Pr}\left[Z \leq \frac{p}{2}m\right] \leq \mathbf{Pr}\left[|Z - \mathbf{E}[Z]| \geq \frac{p}{2}m\right] \leq \frac{\mathrm{Var}[Z]}{\left(\frac{p}{2}m\right)^2} \leq \frac{p(1-p)m}{\left(\frac{p}{2}m\right)^2} = \frac{4(1-p)}{pm} \leq \frac{1}{2} \ ,$$

wobei wir im letzten Schritt $m \geq \frac{8}{\epsilon}$ und deshalb $pm \geq \epsilon m \geq 8$ benutzen. □

Die nächste Aussage ist, dass es, wenn *zwei Sample-Mengen* gezogen werden, eher unwahrscheinlich ist, dass es eine Hypothese gibt, die auf der einen Menge einen großen und auf der anderen Menge keinen Trainingsfehler hat.

Lemma 4.4. *Seien S und S' Mengen von m zufällig gezogenen Samples. Falls m Bedingung (1) erfüllt, dann gilt*

$$\mathbf{Pr}\left[\exists h' \in \mathcal{H} : \mathrm{err}_{S'}(h') \geq \frac{\epsilon}{2} \textit{ und } \mathrm{err}_S(h') = 0\right] \leq \frac{\delta}{2} \ .$$

Beweis. Wir beschreiben einen anderen aber äquivalenten Weg, um S und S' zu bestimmen: Wir ziehen $2m$ mal aus der Verteilung $\mathcal{D}$; sei das Ergebnis T. Jetzt ziehen wir m mal *ohne Zurücklegen* aus T und nennen das Ergebnis S. Schließlich ist S' der Rest aus T also $S' = T \setminus S$.

Betrachte nun eine feste Menge T und festes $h' \in \mathcal{H}$. Sei $h'(x) \neq f(x)$ für genau k Elemente aus T. Die einzige Art und Weise, wie $\mathrm{err}_{S'}(h') \geq \frac{\epsilon}{2}$ eintreten kann, ist dass $k \geq \frac{\epsilon}{2}m$.

Darüber hinaus ist die Wahrscheinlichkeit, dass h' keinen Fehler auf S macht gegeben als

$$\begin{aligned}\mathbf{Pr}\left[\mathrm{err}_S(h') = 0 \;\middle|\; T\right] &= \frac{2m-k}{2m} \cdot \frac{2m-k-1}{2m-1} \cdot \ldots \cdot \frac{m-k+1}{m+1} \\ &= \frac{m(m-1)\ldots(m-k+1)}{(2m)(2m-1)\ldots(2m-k+1)} \leq 2^{-k} \ .\end{aligned}$$

Hierbei gilt die zweite Gleichung, weil sich die alle Faktoren aus dem Zähler und dem Nenner kürzen bis auf die ersten k im Nenner und die letzten k im Zähler.

Das bedeutet, dass für festes h' und festes T

$$\mathbf{Pr}\left[\mathrm{err}_S(h') = 0 \text{ und } \mathrm{err}_{S'}(h') \geq \frac{\epsilon}{2} \;\middle|\; T\right] \leq \left\{ \begin{array}{ll} 0 & \text{falls } k < \frac{\epsilon}{2}m \\ 2^{-k} & \text{sonst} \end{array} \right\} \leq 2^{-\frac{\epsilon}{2}m} \ .$$

An dieser Stelle kommt die Wachstumsfunktion ins Spiel: die Menge T hat nur Größe $2m$. Das bedeutet, weil nur die Funktionswerte von h' auf T wichtig sind, dass es effektiv höchstens $\Pi_{\mathcal{H}}(2m)$ unterschiedliche Wahlen für h gibt. Deshalb gibt uns die Union Bound jetzt

$$\mathbf{Pr}\left[\exists h' \in \mathcal{H} : \mathrm{err}_S(h') = 0 \text{ und } \mathrm{err}_{S'}(h') \geq \frac{\epsilon}{2} \;\middle|\; T\right] \leq \Pi_{\mathcal{H}}(2m) 2^{-\frac{\epsilon}{2}m} \leq \frac{\delta}{2} \ .$$

Diese Schranke gilt für alle bedingten Wahrscheinlichkeiten, egal welche Menge T wir nutzen. Also gilt sie auch für die unbedingte Wahrscheinlichkeit. □

Beweis von Satz 4.2. Wir werden nun die Lemmata zusammenfügen. Sei A das Ereignis, dass es ein $h \in \mathcal{H}$ gibt mit $\mathrm{err}_{\mathcal{D}}(h) \geq \epsilon$ aber $\mathrm{err}_S(h) = 0$. Wir möchten zeigen, dass $\mathbf{Pr}[A] \leq \delta$.

Um Lemma 4.4 anzuwenden, führen wir ein Hilfsereignis B ein. Sei dafür S' eine andere Menge von m Datenpunkten mit zugehörigen Labels, die auch unabhängig aus $\mathcal{D}$ gezogen sind. Sei B das Ereignis, dass es ein $h' \in \mathcal{H}$ gibt mit $\mathrm{err}_{S'}(h') \geq \frac{\epsilon}{2}$ aber $\mathrm{err}_S(h') = 0$. Gemäß Lemma 4.4 gilt $\mathbf{Pr}[B] \leq \frac{\delta}{2}$.

Darüber hinaus behaupten wir, dass $\mathbf{Pr}[B \mid A] \geq \frac{1}{2}$. Dafür sollten wir verstehen, was diese bedingte Wahrscheinlichkeit bedeutet. Ereignis A ist bereits eingetreten. Dieses hängt von der Menge S ab und sagt, dass es ein $h \in \mathcal{H}$ mit $\mathrm{err}_{\mathcal{D}}(h) \geq \epsilon$ aber $\mathrm{err}_S(h) = 0$. Damit Ereignis B eintritt, ist es nun hinreichend, dass $\mathrm{err}_{S'}(h) \geq \frac{\epsilon}{2}$. (Es ist nicht gefordert, dass $h = h'$ ist, deshalb ist dies nur hinreichend aber nicht notwendig.) Nun können wir Lemma 4.3 nutzen. Die Wahrscheinlichkeit, dass für genau dieses h gilt $\mathrm{err}_{S'}(h) \geq \frac{\epsilon}{2}$ ist mindestens $\frac{1}{2}$.

Nun nutzen wir $\mathbf{Pr}[B] \geq \mathbf{Pr}[B \mid A]\,\mathbf{Pr}[A]$, um $\mathbf{Pr}[A] \leq \frac{\mathbf{Pr}[B]}{\mathbf{Pr}[B|A]}$ zu erhalten. Mit $\mathbf{Pr}[B] \leq \frac{\delta}{2}$ und $\mathbf{Pr}[B \mid A] \geq \frac{1}{2}$, folgt also $\mathbf{Pr}[A] \leq \delta$. □

3 Der Nicht-Realisierbare/Agnostische Fall

Bislang haben wir im Kontext von PAC-Learning nur den realisierbaren Fall behandelt. Das bedeutet, es gibt nicht nur eine Grundwahrheit $f\colon X \to \{-1, +1\}$, die die korrekten Labels angibt, sondern auch, dass f in der Hypothesenklasse $\mathcal{H}$ enthalten ist, die wir betrachten. Dies bedeutet insbesondere, dass es immer möglich ist, eine Hypothese zu finden, die keinen Trainingsfehler hat.

In typischen Fragen des Maschinellen Lernens ist diese Annahme jedoch eigentlich nie erfüllt. Die Merkmale beschreiben niemals die Wirklichkeit vollständig. Im Fall von Spam-Klassifikation mögen als Merkmale Worthäufigkeiten, IP-Adressen, Daten im Header und so weiter zur Verfügung stehen. Auf Basis dieser Information ist es aber unmöglich, alle E-Mails immer korrekt zu klassifizieren. Etwas philosophischer kann man sich auch fragen, ob es überhaupt eine klare Trennung zwischen Spam und erwünschten E-Mails gibt. Schließlich gibt es noch einen weiteren Aspekt: Selbst wenn es möglich wäre, eine Hypothesenklassen anzugeben, die eine perfekte Klassifikation ermöglichen würde, möchte man aus Effizienzgründen vielleicht eine weniger komplexe Klasse wählen.

Wie modellieren wir also Lernprobleme jenseits des realisierbaren Falls? Betrachten wir zunächst das linke Beispiel von Abbildung 1. Hier ist $X = [0, 1]^2$ und es gibt in der Tat eine Grundwahrheit $f\colon X \to \{-1, +1\}$, die allerdings relativ komplex ist. Nun könnte $\mathcal{H}$ die Menge aller linearen Klassifikatoren sein, das heißt, die Funktionen, die durch eine Gerade positive und negative Punkte trennen. In einem solchen Fall könnten wir weiterhin den tatsächlichen Fehler $\mathrm{err}_{\mathcal{D},f}(h)$ einer Hypothese h hinsichtlich einer Verteilung über Datenpunkte $\mathcal{D}$ und der Grundwahrheit f definieren als

$$\mathrm{err}_{\mathcal{D},f}(h) := \mathbf{Pr}_{x\sim\mathcal{D}}[h(x) \neq f(x)] \quad .$$

Falls $f \notin \mathcal{H}$ ist, ist es nun aber nicht mehr möglich, dass $\mathrm{err}_{\mathcal{D},f}(h)$ beliebig klein wird.

Das rechte Beispiel ist komplexer. Hier gibt es keine Grundwahrheit. Es könnte beispielsweise sein, dass im Datenpunkte im grauen Bereich mit Wahrscheinlichkeit 50 % positiv und sonst negativ sind. Hierfür schauen wir uns Wahrscheinlichkeitsverteilungen über $X \times \{-1, +1\}$ an. Das heißt, diese Verteilung liefert einen zufälligen Datenpunkt mit Label. Äquivalent könnten wir auch wieder eine Verteilung über unbeschriftete Datenpunkte haben und dann für jeden von diesen eine Wahrscheinlichkeit eines positiven Labels.

Der tatsächlichen Fehler $\mathrm{err}_{\mathcal{D}}(h)$ einer Hypothese h hinsichtlich einer solchen Verteilung $\mathcal{D}$ über Datenpunkt-/Label-Paare ist definiert als

$$\mathrm{err}_{\mathcal{D}}(h) := \mathbf{Pr}_{(x,y)\sim\mathcal{D}}[h(x) \neq y] \quad .$$

In beiden Fällen haben wir keine Hoffnung, eine Hypothese zu finden, sodass der tatsächliche Fehler beliebig klein wird. Stattdessen hoffen wir nun, möglichst nah an die bestmögliche Hypothese zu kommen.

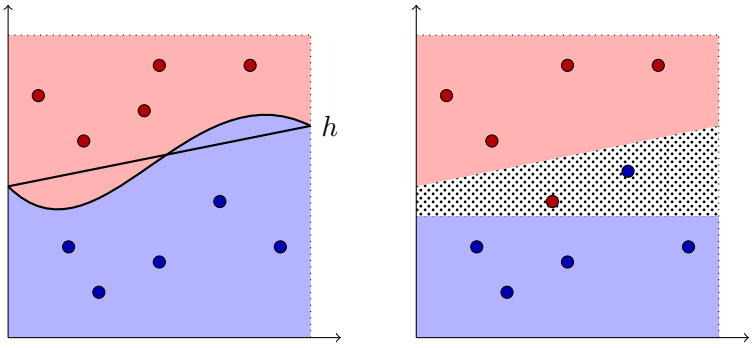


Abbildung 1: Beispiele von nicht-realisierbaren Fällen. Links gibt es keinen Hypothese h in unserer Klasse der linearen Separatoren, die mit der Grundwahrheit f auf allen Punkten übereinstimmt. Rechts sind im grauen Bereich die Labels zufällig; beispielsweise -1 oder $+1$ mit Wahrscheinlichkeit 50%. Es gibt also gar keine Funktion $f\colon X \to \{0,1\}$, die immer das korrekte Label zurückgibt.

Definition 4.5. *Eine Hypothesenklasse $\mathcal{H}$ ist PAC-lernbar (im agnostischen Sinn), wenn es eine Funktion $m_{\mathcal{H}}$ und einen Lernalgorithmus gibt, der für alle $\epsilon, \delta > 0$ und jede Verteilung $\mathcal{D}$ über Datenpunkt-/Label-Paare mithilfe eines zufälligen Samples S der Größe mindestens $m_{\mathcal{H}}(\epsilon, \delta)$ aus $\mathcal{D}$ gezogen, eine Hypothese $h_S \in \mathcal{H}$ berechnet, sodass*

$$\mathbf{Pr}\left[\mathrm{err}_{\mathcal{D}}(h_S) < \min_{h' \in \mathcal{H}} \mathrm{err}_{\mathcal{D}}(h') + \epsilon\right] \geq 1 - \delta \ .$$

Agnostisch bezieht sich hierbei darauf, dass nicht bekannt, aber auch unerheblich ist, ob es eine Grundwahrheit (in $\mathcal{H}$ bzw. allgemein) gibt, oder nicht.

AGML, Sommersemester 2020 Vorlesung 5 (4 Seiten)

Mehr zum Agnostischen Fall und Grenzen der Lernbarkeit

Thomas Kesselheim Letzte Aktualisierung: 8. Mai 2020

In der vergangenen Vorlesung haben wir die Definition von PAC-Lernen mit agnostischem Sinn kennengelernt. Hier gibt es eine Verteilung $\mathcal{D}$ über Datenpunkt-/Label-Paaren, also über der Menge $X \times \{-1, +1\}$. Der tatsächliche Fehler einer Hypothese h is definiert als

$$\mathrm{err}_{\mathcal{D}}(h) := \mathbf{Pr}_{(x,y)\sim\mathcal{D}}\left[h(x) \neq y\right] \ .$$

Es gibt im Allgemeinen keine Grundwahrheit f, die eine mögliche Hypothese ist. In diesem Fall gilt auch $\min_{h'\in\mathcal{H}} \mathrm{err}_{\mathcal{D}}(h') > 0$. Es ist somit nicht möglich, dass der tatsächliche Fehler eines Algorithmus verschwindet, egal wie viele Samples wir ihm bereitstellen. Stattdessen ist das Ziel, möglichst nah an $\min_{h'\in\mathcal{H}} \mathrm{err}_{\mathcal{D}}(h')$ heranzukommen.

1 Minimieren des Trainingsfehlers im Agnostischen Fall

Gegeben eine Trainingsmenge $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ können wir den Trainingsfehler eine Hypothese definieren als

$$\mathrm{err}_S(h) := \frac{1}{m}|\{h(x_i) \neq y_i\}| \ .$$

Wir können uns nun Algorithmen anschauen, die diesen Trainingsfehler minimieren. Während dies im realisierbaren Fall bedeutet, dass kein Fehler auf S gemacht werden darf, ist dies nun nicht immer möglich. Es ist nur das Ziel, möglichst wenige Fehler zu machen.

Für den agnostischen Fall kann man eine sehr ähnliche Aussage wie im realisierbaren Fall herleiten, die die Wachstumsfunktion nutzt.

Satz 5.1. *Seien eine $\mathcal{H}$ beliebige Hypothesenklasse über X und $\mathcal{D}$ eine Verteilung über $X \times \{-1, +1\}$. Seien $\epsilon > 0$, $\delta > 0$ beliebig und*

$$m \geq \frac{32}{\epsilon^2} \ln\left(\frac{4\Pi_{\mathcal{H}}(2m)}{\delta}\right) \ .$$

Betrachte ein Sample S von m Datenpunkten mit Labels gezogen unabhängig und identisch verteilt aus $\mathcal{D}$. Es gilt mit Wahrscheinlichkeit mindestens $1 - \delta$, dass jede Hypothese h, die $\mathrm{err}_S(h)$ minimiert, auch $\mathrm{err}_{\mathcal{D}}(h) \leq \min_{h'\in\mathcal{H}} \mathrm{err}_{\mathcal{D}}(h') + \epsilon$ erfüllt.

Insbesondere folgt aus dieser Schranke auch, dass eine Hypothesenklassen im agnostischen Sinn PAC-lernbar ist, wenn ihr VC-Dimension endlich ist. Der Lernalgorithmus ist in diesem Fall ein beliebiger Algorithmus, der den Trainingsfehler minimiert.

Viele Schritte im Beweis dieses Satzes sind analog zu seinem Pendant im realisierbaren Fall. Um die Unterschiede und zusätzlichen Techniken zu verdeutlichen, betrachten wir nun den Fall einer *endlichen* Hypothesenklasse $\mathcal{H}$. Wir zeigen, dass für

$$m \geq \frac{2}{\epsilon^2} \ln\left(\frac{2|\mathcal{H}|}{\delta}\right) \tag{1}$$

die Aussage von Satz 5.1 erfüllt ist. Hierzu beweisen wir folgende Behauptung.

Behauptung 5.2.

$$\mathbf{Pr}\left[\exists h \in \mathcal{H} : |\mathrm{err}_{\mathcal{D}}(h) - \mathrm{err}_S(h)| \geq \frac{\epsilon}{2}\right] < \delta \ .$$

Diese Aussage hilft uns wie folgt. Angenommen, wir haben eine Menge S, sodass

$$|\mathrm{err}_{\mathcal{D}}(h) - \mathrm{err}_S(h)| < \frac{\epsilon}{2} \qquad \text{für alle } h \in \mathcal{H}. \tag{2}$$

Das heißt, der tatsächliche Fehler und der Trainingsfehler sind nah bei einander für jede mögliche Hypothese. Ist nun h eine Hypothese, die den Trainingsfehler $\mathrm{err}_S(h)$ minimiert; h' eine Hypothese, die den tatsächlichen Fehler $\mathrm{err}_{\mathcal{D}}(h')$ minimiert, dann gilt

$$\mathrm{err}_{\mathcal{D}}(h) < \mathrm{err}_S(h) + \frac{\epsilon}{2} \leq \mathrm{err}_S(h') + \frac{\epsilon}{2} < \mathrm{err}_{\mathcal{D}}(h') + \epsilon \ .$$

Für den Beweis von Behauptung 5.2 zeigen nun wieder zunächst eine Aussage über eine einzelne Hypothese.

Lemma 5.3. *Betrachte eine feste Hypothese $h \in \mathcal{H}$. Sei S eine Menge von m Datenpunkt-/Label-Paaren aus $\mathcal{D}$. Dann gilt für alle $\gamma > 0$*

$$\mathbf{Pr}\left[|\mathrm{err}_{\mathcal{D}}(h) - \mathrm{err}_S(h)| \geq \gamma\right] \leq 2\exp\left(-2m\gamma^2\right) \ .$$

Beweis. Diese Aussage folgt einigermaßen direkt aus der Hoeffding-Ungleichung. Diese lautet wie folgt.

Lemma 5.4 (Hoeffding-Ungleichung)**.** *Seien $Z_1, \ldots, Z_N$ unabhängige Zufallsvariablen, sodass $a_i \leq Z_i \leq b_i$ mit Wahrscheinlichkeit 1. Sei $\bar{Z} = \frac{1}{N}\sum_{i=1}^{N} Z_i$ ihr Durchschnitt. Dann gilt für alle $\gamma \geq 0$*

$$\mathbf{Pr}\left[|\bar{Z} - \mathbf{E}\left[\bar{Z}\right]| \geq \gamma\right] \leq 2\exp\left(-\frac{2N^2\gamma^2}{\sum_{i=1}^{N}(b_i - a_i)^2}\right) \ .$$

Die Ungleichung quantifiziert (und verallgemeinert) das Gesetz der großen Zahlen: Der Durchschnitt vieler Züge aus derselben Verteilung konvergiert gegen den Erwartungswert.

Für unsere Aussage sei $Z_i = 1$, falls $h(x_i) \neq y_i$ und 0 sonst. Dann gilt $\bar{Z} = \mathrm{err}_S(h)$. Außerdem sind $Z_1, \ldots, Z_m$ unabhängig und es gilt $0 \leq Z_i \leq 1$. Also können wir die Hoeffing-Ungleichung mit $a_i = 0$, $b_i = 1$ and $N = m$ anwenden.

Schließlich stellen wir fest, dass $\mathbf{E}\left[Z_i\right] = \mathrm{err}_{\mathcal{D}}(h)$ für alle i und damit auch $\mathbf{E}\left[\bar{Z}\right] = \frac{1}{m}\sum_{i=1}^{m}\mathbf{E}\left[Z_i\right] = \mathrm{err}_{\mathcal{D}}(h)$. Die Aussage des Lemmas ist also genau die Schranke, die aus der Hoeffding-Ungleichung folgt. □

Jetzt ist der Beweis von Behauptung 5.2 auch unkompliziert.

Beweis von Behauptung 5.2. Wir nutzen wieder die Union Bound and wählen $\gamma = \frac{\epsilon}{2}$ in Lemma 5.3. Damit bekommen wir

$$\mathbf{Pr}\left[\exists h \in \mathcal{H} : |\mathrm{err}_{\mathcal{D}}(h) - \mathrm{err}_S(h)| \geq \frac{\epsilon}{2}\right] \leq |\mathcal{H}| \cdot 2\exp\left(-2m\frac{\epsilon^2}{4}\right) \leq \delta \ . \qquad \square$$

2 Unendliche VC-Dimension

Wir haben bereits gesehen, dass jede Hypothesenklassen $\mathcal{H}$ endlicher VC-Dimension PAC-lernbar ist. Aber was ist im Fall von unendlicher VC-Dimension? Beispielsweise die Klasse aller Hypothesen $\mathbb{N} \to \{-1, +1\}$. Oder allgemeiner alle Funktionen $X \to \{-1, +1\}$. Wie wir zeigen werden, sind diese nicht PAC-lernbar.

Satz 5.5. *Jede Hypothesenklasse von unendlicher VC-Dimension ist nicht PAC-lernbar im realisierbaren Sinn.*

Um diesen Satz zu beweisen, müssen wir zeigen, dass Lernalgorithmus $\mathcal{A}$ und Funktion $m_{\mathcal{H}}$ aus der Definition von PAC-Lernbarkeit nicht existieren. Wir werden die folgende Aussage zeigen.

Behauptung 5.6. *Sei $\mathcal{H}$ eine Hypothesenklasse von VC-Dimension mindestens d. Dann gibt es für jeden Lernalgorithmus $\mathcal{A}$ eine Verteilung $\mathcal{D}$ und eine Grundwahrheit f, sodass auf einer Trainingsmenge S der Größe höchstens $\frac{d}{2}$ gilt:* $\operatorname{err}_{\mathcal{D}}(h_S) \geq \frac{1}{8}$ *mit Wahrscheinlichkeit mindestens* $\frac{1}{7}$.

Beweis. Laut Definition spaltet $\mathcal{H}$ eine Menge der Größe d auf. Sei also $T \subseteq X$, $|T| = d$, eine solche Menge. Es gilt nun $|\mathcal{H}|_T| = 2^d$. Definiere $k = 2^d$ und schreibe $\mathcal{H}|_T = \{\ell_1, \ldots, \ell_k\}$, wobei jeweils $\ell_i \colon T \to \{-1, +1\}$ und alle ℓ_i unterschiedlich sind.

Für jedes ℓ_i finden wir ein $f_i \in \mathcal{H}$, sodass $f_i(x) = \ell_i(x)$ für alle $x \in X$. Jede dieser Funktionen f_i könnte die Grundwahrheit sein. Die entscheidende Beobachtung ist, dass wenn uns lediglich ein Sample der Größe $\frac{d}{2}$ gegeben wird, wir für höchstens $\frac{d}{2}$ Punkte in T das korrekte Label wissen. Für die übrigen Punkte können die Label vollkommen beliebig sein.

Betrachte nun einen festen Lernalgorithmus und als Verteilung $\mathcal{D}$ die uniforme Verteilung auf T. Sei $h_{S,i}$ die Hypothese, die der Lernalgorithmus auf Sample S berechnet, wenn die Grundwahrheit f_i ist[1]. Wir möchten nun zeigen, dass

$$\max_i \mathbf{Pr}\left[\operatorname{err}_{\mathcal{D},f_i}(h_{S,i}) \geq \frac{1}{8}\right] \geq \frac{1}{7} \ .$$

Das heißt, dass es eine Grundwahrheit gibt, für die der Algorithmus schlecht ist. Definieren wir nun Zufallsvariablen Z_i (abhängig von S), so dass $Z_i = 1$ falls $\operatorname{err}_{\mathcal{D},f_i}(h_{S,i}) \geq \frac{1}{8}$, anderenfalls $Z_i = 0$.

In dieser Notation wollen wir zeigen, dass

$$\max_i \mathbf{Pr}\left[Z_i = 1\right] \geq \frac{1}{7} \ .$$

Hierfür ist es hinreichend, dass

$$\frac{1}{k}\sum_{i=1}^{k} \mathbf{Pr}\left[Z_i = 1\right] \geq \frac{1}{7} \ .$$

Da $\mathbf{Pr}\left[Z_i = 1\right] = \mathbf{E}\left[Z_i\right]$, ist diese Aussage mittels Linearität des Erwartungswertes äquivalent zu

$$\mathbf{E}\left[\sum_{i=1}^{k} Z_i\right] \geq \frac{k}{7} \ .$$

Betrachten wir ein festes $x \in T$, dann gibt es für jede Hypothese f_i genau eine Hypothese f_{-i}, die überall auf T mit f_i übereinstimmt, nur $f_i(x) \neq f_{-i}(x)$. Falls $x \notin S$, muss folglich gelten $h_{S,i} = h_{S,-i}$. Also muss entweder $h_{S,i}(x) \neq f_i(x)$ oder $h_{S,-i}(x) \neq f_{-i}(x)$ sein. Allgemeiner gesagt bedeutet dies, dass für alle $x \notin S$ gilt, dass $h_{S,i}(x) \neq f_i(x)$ für genau die Hälfte aller i.

Für jede feste Menge S mit $|S| \leq \frac{1}{2}|T|$ können wir also schreiben

$$\frac{1}{k}\sum_{i=1}^{k} \operatorname{err}_{\mathcal{D},f_i}(h_{S,i}) \geq \frac{1}{2}\frac{|T \setminus S|}{|T|} \geq \frac{1}{4} \ .$$

[1]Prinzipiell könnte $h_{S,i}$ auch randomisiert sein. Der Beweis würde genauso gelten. Der Einfachheit halber gehen wir aber davon aus, dass $h_{S,i}$ deterministisch von S und i abhängt.

Wenn wir S durch $\frac{d}{2}$ Züge aus $\mathcal{D}$ bestimmen, ist $|T \setminus S| \geq \frac{1}{2}|T|$.

Andererseits gilt auch

$$\sum_{i=1}^{k} \mathrm{err}_{\mathcal{D},f_i}(h_{S,i}) \leq \sum_{i=1}^{k} Z_i + \frac{1}{8}\left(k - \sum_{i=1}^{k} Z_i\right) = \frac{1}{8}k + \frac{7}{8}\sum_{i=1}^{k} Z_i \ ,$$

denn diejenigen i mit $Z_i = 1$ tragen höchstens 1, die übrigen höchstens $\frac{1}{8}$ zu der Summe bei.

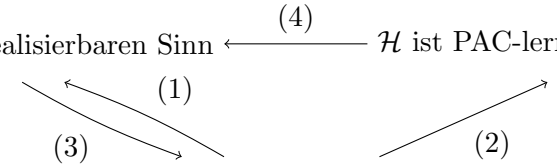
Folglich gilt also für jedes S immer

$$\sum_{i=1}^{k} Z_i \geq \frac{k}{7} \ .$$

Damit gilt die Ungleichung erst recht auch im Erwartungswert über S. □

3 Gesamtbild: PAC-Lernbarkeit

Zusammengenommen haben wir nun folgendes Bild von Implikationen.



Implikation (1) haben wir in den vergangenen Vorlesungen gezeigt. (2) folgt aus Satz 5.1, den wir nicht bewiesen haben. (3) ist die Aussage von Satz 5.5. (4) ist eine Übungsaufgabe. Insgesamt sind also alle drei Begriffe äquivalent.

Dies bedeutet übrigens nur, dass bei Hypothesenklassen mit endlicher VC-Dimension „genügend“ Samples für bei jeder Verteilung $\mathcal{D}$ ausreichen, um die beste Hypothese zu finden. Es bedeutet nicht, dass „genügend“ im realisierbaren und im agnostischen Fall gleich große Zahlen sind. Auch kann es bei Hypothesenklassen mit unendlicher VC-Dimension Verteilungen $\mathcal{D}$ geben, die Lernbarkeit ermöglichen.

AGML, Sommersemester 2020 Vorlesung 6 (6 Seiten)

Lineare Klassifikation

Anne Driemel Letzte Aktualisierung: 14. Mai 2020

Die lineare Klassifikation ist eine der grundlegendsten Methoden im Maschinellen Lernen. Die entsprechende Hypothesenklasse $\mathcal{H}$ ist definiert als die Menge von Funktionen der Form $h_{a,b} : \mathbb{R}^d \to \{-1, +1\}$ mit $a \in \mathbb{R}^d, b \in \mathbb{R}$ und[1]

$$h_{a,b}(x) = \begin{cases} +1 \text{ falls } \langle a, x\rangle \geq b \\ -1 \text{ sonst} \end{cases}$$

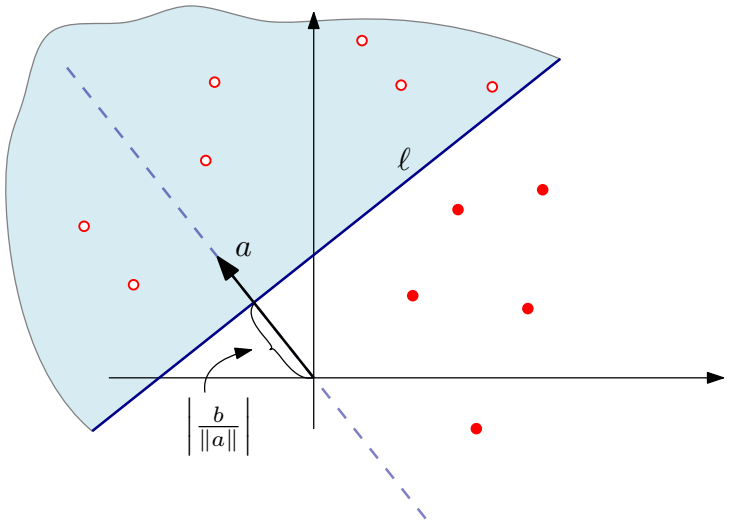
Wir können dies wieder äquivalent als Mengensystem beschreiben mit den Mengen

$$r_{a,b} = \left\{ x \in \mathbb{R}^d \;\middle|\; \langle a, x\rangle \geq b \right\}$$

Die Menge $r_{a,b}$ definiert einen Halbraum von $\mathbb{R}^d$. Ein Halbraum ist eine Menge, die durch eine Hyperebene beschränkt ist. In unserem Fall ist das die folgende Hyperebene

$$\ell = \left\{ x \in \mathbb{R}^d \;\middle|\; \langle a, x\rangle = b \right\}$$

Im $\mathbb{R}^2$ können wir uns das geometrisch vorstellen. Die Hyperebene ℓ ist orthogonal zu der Geraden g durch den Nullpunkt, die den Vektor a enthält und schneidet diese Gerade im Abstand $\left|\frac{b}{\|a\|}\right|$ zum Nullpunkt[2]. Der Halbraum $r_{a,b}$ umfasst alle Punkte zu der Seite von ℓ die durch die Richtung des Vektors a angegeben ist.



[1] Für Vektoren $x = (x_1, \ldots, x_d)$ und $y = (y_1, \ldots, y_d)$ ist das Skalarprodukt definiert als $\langle x, y\rangle = \sum_{i=1}^{d} x_i y_i$

[2] Die Norm eines Vektors $x = (x_1, \ldots, x_d)$ ist hier definiert als $\|x\| = \sqrt{\sum_{i=1}^{d} x_i^2}$

1 VC-dimension von Halbräumen

Wir wollen heute die VC-dimension des Mengensystems aller Halbräume analysieren. Zunächst wollen wir dazu ein paar grundlegende Begriffe einführen.

Definition 6.1 (Affinkombination)**.** *Für beliebige Punkte $p_1, \dots, p_n \in \mathbb{R}^d$ und Parameter $\alpha_1, \dots, \alpha_n \in \mathbb{R}$ mit $\sum_{i=1}^n \alpha_i = 1$ ist $\sum_{i=1}^n \alpha_i p_i$ eine Affinkombination. Die Punkte $p_1, \dots, p_n$ sind affin abhängig wenn es einen Punkt p_i gibt, sodass $p_i = \sum_{\substack{j=1 \\ j \neq i}}^n \alpha_i p_i$. Also genau dann wenn wir einen Punkt der Menge durch eine Affinkombination der anderen Punkte ausdrücken können. Die Menge aller Affinkombinationen einer festen Menge wird als ihre affine Hülle bezeichnet.*

Eine Affinkombination ist also eine Linearkombination mit der zusätzlichen Bedingung, dass die Summe der Koeffizienten 1 ergibt. Mithilfe von Affinkombinationen lassen sich Geraden, Ebenen und Hyperebenen darstellen.

Beispiel 6.2. *Die Menge aller Affinkombinationen zweier Punkte $p_1, p_2 \in \mathbb{R}^2$ ist*

$$\{\ tp_1 + (1-t)p_2 \ \mid \ t \in \mathbb{R}\ \}$$

Das ist die Menge aller Punkte auf der Geraden, welche p_1 und p_2 enthält.

Lemma 6.3. *Jede Menge von $d+2$ Punkten im $\mathbb{R}^d$ ist affin abhängig.*

Beweis. Sei $A = \{p_1, \dots, p_{d+2}\}$. Setze ein beliebiges p_i fest und betrachte für $i \neq j$ die $d+1$ Punkte $q_j = p_j - p_i$. Da $d+1$ Punkte linear abhängig sind, existieren Parameter β_j für $i \neq j$ und ein Punkt q_r sodass

$$q_r = \sum_{\substack{j=1 \\ i \neq j \text{ und } j \neq r}}^{d+2} \beta_j q_j.$$

Somit gilt

$$p_r = q_r + p_i = \left(\sum_{\substack{j=1 \\ i \neq j \text{ und } j \neq r}}^{d+2} \beta_j q_j \right) + p_i = \sum_{\substack{j=1 \\ i \neq j \text{ und } j \neq r}}^{d+2} \beta_j p_j - \left(\sum_{\substack{j=1 \\ i \neq j \text{ und } j \neq r}}^{d+2} \beta_j \right) p_i + p_i$$

Jetzt können wir $\beta_i = -\left(\sum_{\substack{j=1 \\ i \neq j \text{ und } j \neq r}}^{d+2} \beta_j \right) + 1$ definieren und somit haben wir $\sum_{\substack{j=1 \\ j \neq r}}^{d+2} \beta_j = 1$ womit die Bedingung für eine Affinkombination erfüllt ist. □

Definition 6.4 (Konvexkombination)**.** *Für beliebige Punkte $p_1, \dots, p_n \in \mathbb{R}^d$ und Parameter $\alpha_1, \dots, \alpha_n \in \mathbb{R}$ mit $\sum_{i=1}^n \alpha_i = 1$ und $\alpha_i \geq 0$ für alle $1 \leq i \leq n$ ist $\sum_{i=1}^n \alpha_i p_i$ eine Konvexkombination. Die Menge aller Konvexkombinationen einer festen Menge wird als ihre konvexe Hülle bezeichnet.*

Beispiel 6.5. *Die konvexe Hülle von zwei Punkten $p_1, p_2 \in \mathbb{R}^2$ ist die Menge*

$$\{\ tp_1 + (1-t)p_2 \ \mid \ t \in [0,1]\ \}.$$

Das ist die Strecke mit Endpunkten p_1 und p_2.

Beispiel 6.6. *Die konvexe Hülle von drei Punkten $p_1, p_2, p_3 \in \mathbb{R}^2$ ist die Menge*

$$\{\ \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3 \ \mid \ \alpha_1, \alpha_2, \alpha_3 \geq 0 \text{ und } \alpha_1 + \alpha_2 + \alpha_3 = 1\ \}.$$

Das ist das Dreieck mit den Eckpunkten p_1, p_2 und p_3.

1.1 Radon's Lemma

Lemma 6.7 (Radon's Lemma)**.** *Für jede Menge A von $d+2$ Punkten im $\mathbb{R}^d$ existieren Teilmengen $A_1, A_2 \subseteq A$ mit $A_1 \cap A_2 = \emptyset$ und ein Punkt q, sodass q sowohl als Konvexkombination von A_1 dargestellt werden kann, als auch eine Konvexkombination von A_2. Wir bezeichnen q als Radonpunkt der Mengen A_1 und A_2.*

Beweis. Sei $A = \{p_1, \ldots, p_{d+2}\}$. Da wir $d+2$ Punkte haben, sind diese affin abhängig. Das heißt, es gibt ein $p_i \in A$ welches durch eine Affinkombination der anderen Punkte in A dargestellt werden kann. Also existieren Parameter α_j für $1 \leq j \leq n$ mit $i \neq j$, sodass

$$p_i = \sum_{\substack{j=1 \\ i \neq j}}^{d+2} \alpha_j p_j \quad \text{mit} \quad \sum_{\substack{j=1 \\ i \neq j}}^{d+2} \alpha_j = 1$$

Setzen wir nun $\alpha_i = -1$, dann können wir $\alpha_i p_i$ auf beiden Seiten der Gleichung addieren und bekommen

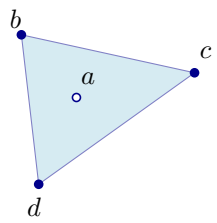
$$0 = \sum_{j=1}^{d+2} \alpha_j p_j \quad \text{mit} \quad \sum_{j=1}^{d+2} \alpha_j = 0.$$

Wir definieren nun zwei Indexmengen $I_1 = \{\ i \mid \alpha_i > 0\ \}$ und $I_2 = \{\ i \mid \alpha_i < 0\ \}$. Durch äquivalente Umformung bekommen wir

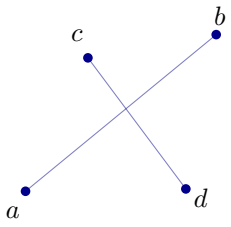
$$-\sum_{i \in I_2} \alpha_i p_i = \sum_{i \in I_1} \alpha_i p_i \quad \text{und} \quad -\sum_{i \in I_2} \alpha_i = \sum_{i \in I_1} \alpha_i$$

Setzen wir nun $\gamma = \sum_{i \in I_1} \alpha_i$, dann definiert $q_1 = \sum_{i \in I_1} \beta_i p_i$ mit $\beta_i = \frac{\alpha_i}{\gamma}$ eine Konvexkombination der Punkte in A mit Index in I_1. Ähnlich definert $q_2 = \sum_{i \in I_2} \beta_i p_i$ mit $\beta_i = -\frac{\alpha_i}{\gamma}$ eine Konvexkombination der Punkte in A mit Index in I_2. Weiter ist $q_1 = q_2$ und $I_1 \cap I_2 = \emptyset$. Damit ist der Satz bewiesen. □

Beispiel 6.8 (Radonpunkt)**.** *Für 4 verschiedene Punkte a, b, c, d in der Ebene gibt es im Prinzip zwei Möglichkeiten wie die Teilmengen in Radon's Lemma zueinander liegen können.*



(a) $A_1 = \{a\}, A_2 = \{b, c, d\}$.
Ein Punkt a ist in der konvexen Hülle der anderen Punkte $\{b, c, d\}$ enthalten. Hier ist a ein Radonpunkt.



(b) $A_1 = \{a, b\}, A_2 = \{c, d\}$.
Zwei Strecken $\overline{ab}$ und $\overline{cd}$ schneiden sich in einem Punkt. Der Schnittpunkt stellt einen Radonpunkt dar.

1.2 Beweis der VC-dimension

Satz 6.9. *Die VC-dimension von Halbräumen in $\mathbb{R}^d$ ist höchstens $d+1$.*

Beweis. Sei $\mathcal{R}$ das Mengensystem von Halbräumen in $\mathbb{R}^d$. Das heißt, jede Menge in $\mathcal{R}$ ist von der Form $r_{a,b} = \left\{ x \in \mathbb{R}^d \;\middle|\; \langle a, x\rangle \geq b \right\}$ mit $a \in \mathbb{R}^d, b \in \mathbb{R}$. Wir zeigen, dass die VC-dimension höchstens $d+1$ sein kann. Angenommen dem wäre nicht so. Wir führen diese Annahme zu einem Widerspruch. Sei $A = \{p_1, \ldots, p_{d+2}\} \subseteq \mathbb{R}^d$ eine Menge von $d+2$ Punkten die durch $\mathcal{R}$ aufgespalten wird. Laut Radon's Lemma gibt es zwei disjunkte Teilmengen $A_1, A_2 \subseteq A$ die einen gemeinsamen Radonpunkt q besitzen. Das heisst, es gibt Konvexkombinationen

$$q = \sum_{i \in I_1} \alpha_i p_i \text{ und } q = \sum_{i \in I_2} \beta_i p_i$$

wobei I_1 und I_2 die Indexmengen von A_1 und A_2 sind.

Sei $r_{a,b}$ der Halbraum, der A_1 von A abspaltet. Dann ist

$$A_1 = r_{a,b} \cap A \quad \text{ und } A_2 \cap r_{a,b} = \emptyset$$

also ist

$$\langle a, p_i\rangle \geq b \text{ für alle } i \in I_1$$

und

$$\langle a, p_i\rangle < b \text{ für alle } i \in I_2$$

Betrachten wir nun $\langle a, q\rangle$, so können wir mit der ersten Konvexkombination unter Nutzung der Linearität des Skalarprodukts herleiten

$$\langle a, q\rangle = \left\langle a, \sum_{i \in I_1} \alpha_i p_i \right\rangle = \sum_{i \in I_1} \alpha_i \langle a, p_i\rangle \geq \sum_{i \in I_1} \alpha_i b = b$$

Die Abschätzung $\alpha_i \langle a, p_i\rangle \leq \alpha_i b$ können wir natürlich nur machen, da alle α_i positiv sind. Die letzte Gleichheit folgt aus der anderen Bedingung an die Konvexkombination, dass die Summe der Koeffizienten gleich 1 ist. Ähnlich können wir mit der zweiten Konvexkombination herleiten

$$\langle a, q\rangle = \left\langle a, \sum_{i \in I_2} \beta_i p_i \right\rangle = \sum_{i \in I_2} \beta_i \langle a, p_i\rangle < \sum_{i \in I_2} \beta_i b = b$$

Damit ergibt sich ein Widerspruch mit $b > \langle a, q\rangle \geq b$. □

Geometrisch kann man sich den Beweis für die obere Schranke wie folgt veranschaulichen. Laut Radon's Lemma gibt es zwei disjunkte Teilmengen $A_1, A_2 \subseteq A$, sodass die beiden konvexen Hüllen von A_1 und A_2 *nicht* zueinander disjunkt sind. Ein Halbraum, der A_1 von A abspaltet ist durch eine Hyperebene beschränkt, die A_1 und A_2 linear separiert. Die Hyperebene würde dann aber auch die beiden konvexen Hüllen linear separieren.

Satz 6.10. *Die VC-dimension von Halbräumen in $\mathbb{R}^d$ ist mindestens $d+1$.*

Beweis. Wir zeigen, dass es eine Punktmenge mit $d+1$ Punkten gibt, die durch $\mathcal{R}$ aufgespalten wird. Dafür konstruieren wir eine Menge $A \subseteq \mathbb{R}^d$ mit $|A| = d+1$, die durch $\mathcal{R}$ aufgespalten wird. Sei e_i der Einheitsvektor, der überall 0 ist und nur an der $i$ten Koordinate eine 1 hat. Wir definieren als Menge A die d Einheitsvektoren und den Nullvektor $e_0 = (0, \ldots, 0)$. Nun können

wir für jede Teilmenge $A' \subseteq A$ zeigen, dass A' abgespalten wird. Wir wählen $a = (a_1, \ldots, a_d) \in \mathbb{R}^d$ und $b \in \mathbb{R}$ wie folgt

$$a_i = \begin{cases} 1 \text{ falls } e_i \in A', \\ -1 \text{ sonst} \end{cases} \qquad \text{und } b = \begin{cases} 0 \text{ falls } e_0 \in A' \\ 1 \text{ sonst} \end{cases}$$

Dann gilt $\langle a, e_0 \rangle = 0$ und für $0 < i \leq d$

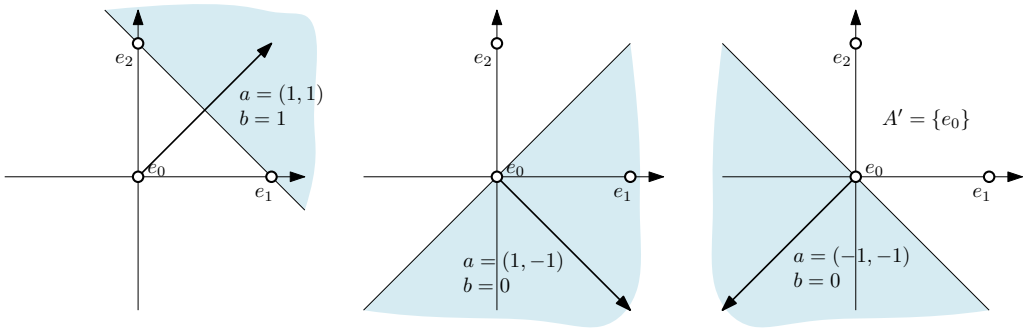
$$\langle a, e_i \rangle = \begin{cases} 1 \text{ falls } e_i \in A' \\ -1 \text{ sonst} \end{cases}$$

Durch eine Fallanalyse kann man nun zeigen, dass für alle $e_i \in A$ gilt, dass

$$\langle a, e_i \rangle \geq b \quad \Leftrightarrow \quad e_i \in A'$$

Somit kann die Menge A' immer durch einen Halbraum abgespalten werden. □

Beispiel 6.11. *Im Beispiel von $d = 2$ können wir exemplarisch Teilmengen von $A = \{e_0, e_1, e_2\}$ aus dem obigen Beweis und die zugehörigen Halbräume visualisieren.*



(a) $A' = \{e_1, e_2\}$ (b) $A' = \{e_0, e_1\}$ (c) $A' = \{e_0\}$

2 Homogene Halbräume

Sei $\mathcal{R}_0$ das Mengensystem aller Halbräume der Form

$$r_w = \left\{ x \in \mathbb{R}^d \;\middle|\; \langle w, x \rangle \geq 0 \right\} \quad \text{mit } w \in \mathbb{R}^d$$

Wir bezeichnen Halbräume dieser Form als homogene Halbräume.

Mit dem Mengensystem $\mathcal{R}_0$ sind auch allgemeine Halbräume im $\mathbb{R}^{d-1}$ darstellbar. Insbesondere können wir eine Funktion $\phi : \mathbb{R}^{d-1} \to \mathbb{R}^d$ definieren als $\phi(x) = (x_1, \ldots, x_{d-1}, 1)$ und dann existiert für jeden Halbraum

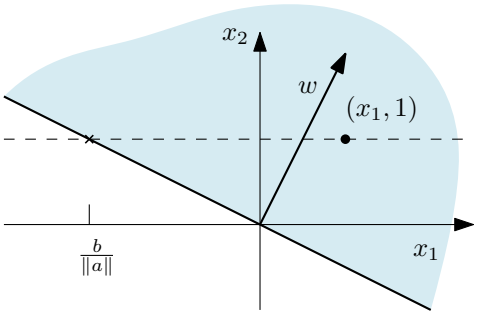
$$r_{a,b} = \left\{ x \in \mathbb{R}^{d-1} \;\middle|\; \langle a, x \rangle \geq b \right\} \quad \text{mit } a \in \mathbb{R}^{d-1}, b \in \mathbb{R}$$

ein Halbraum $r_w \in \mathcal{R}_0$ sodass für alle $x \in \mathbb{R}^{d-1}$ gilt

$$x \in r_{a,b} \quad \Leftrightarrow \quad \phi(x) \in r_w$$

insbesondere können wir $w = (a_1, \ldots, a_{d-1}, -b)$ wählen damit dies erfüllt ist.

Beispiel 6.12. *Für $d = 2$ können wir uns diese Halbräume im konkreten Beispiel wie folgt veranschaulichen. Sei $a = 1$, $b = -2$, dann ist $w = (1, 2)$. Die Gerade, die den Halbraum r_w im $\mathbb{R}^2$ beschränkt schneidet die horizontale Gerade bei $y = 1$ in der x-Koordinate $\frac{b}{\|a\|}$. Die Punkte auf der Horizontalen, die in r_w enthalten sind, entsprechen den Punkten in $\mathbb{R}$, die in $r_{a,b} \subseteq \mathbb{R}$ enthalten sind.*



Referenzen

- Foundations of Machine Learning, Kapitel 3.3. (insbesondere Beispiel 3.12).
- Understanding Machine Learning, Kapitel 9 (insbesondere Kapitel 9.1.3).
- Jiří Matoušek, Lectures on Discrete Geometry, Springer Graduate Texts in Mathematics.

AGML, Sommersemester 2020 Vorlesung 7 (6 Seiten)

Lineare Klassifikation II

Anne Driemel Letzte Aktualisierung: 11. Mai 2020

In der letzten Vorlesung haben wir die VC-dimension von Halbräumen analysiert. Die entsprechende Hypothesenklasse $\mathcal{H}$ ist definiert als die Menge von Funktionen der Form $h_{w,u} : \mathbb{R}^d \to \{-1, +1\}$ mit $w \in \mathbb{R}^d, u \in \mathbb{R}$ und

$$h_{w,u}(x) = \begin{cases} +1 \text{ falls } \langle w, x\rangle \geq u \\ -1 \text{ sonst} \end{cases}$$

Lernalgorithmen, die unter Annahme dieser Hypothesenklasse arbeiten, werden unter dem Begriff der linearen Klassifikation zusammengefasst.

Anhand der VC-dimension können wir feststellen, dass eine Hypothesenklasse PAC-lernbar ist. Ein anderer Aspekt ist die Berechnungskomplexität des Lernproblems. Zur Erinnerung, eine Hypothesenklasse ist *effizient* PAC-lernbar, wenn sie mithilfe eines Polynomialzeitalgorithmus $\mathcal{A}$ PAC-lernbar ist.

Wir widmen uns heute der Berechnungskomplexität der linearen Klassifikation. Sei $S = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$ eine beschriftete Trainingsmenge mit $x^{(i)} = (x_1^{(i)}, \ldots, x_d^{(i)}) \in \mathbb{R}^d$ und $y^{(i)} \in \{-1, +1\}$. Die Aufgabe des Lernalgorithmus ist es, Werte für $w \in \mathbb{R}^d$ und $u \in \mathbb{R}$ zu finden sodass der Trainingsfehler

$$\frac{1}{m} \left| \left\{ \; i \in \{1, \ldots, m\} \;\middle|\; h_{w,u}(x^{(i)}) \neq y^{(i)} \; \right\} \right|$$

minimiert wird.

1 Realisierbarer Fall

Im realisierbaren Fall gehen wir davon aus, dass eine Hypothese mit Trainingsfehler 0 existiert. Das entspricht dem Fall, dass die positive und die negative Menge durch eine Hyperebene separierbar sind. In diesem Fall behaupten wir, dass eine solche Hypothese mithilfe linearer Programmierung gefunden werden kann.

Ein lineares Programm bekommt als Eingabe eine Matrix $A \in \mathbb{R}^{m \times n}$ und Spaltenvektoren $b \in \mathbb{R}^m$ und $c \in \mathbb{R}^n$. Die Aufgabe ist es, einen Spaltenvektor $v \in \mathbb{R}^n$ mit $Av \geq b$ zu finden, der $\langle c, v\rangle$ maximiert. Falls dies nicht möglich ist, dann gibt es zwei Möglichkeiten. Entweder existiert kein $v \in \mathbb{R}^n$ welches $Av \geq b$ erfüllt, oder es existiert kein Maximum für $\langle c, v\rangle$ in der Menge der $v \in \mathbb{R}^d$, die $Av \geq b$ erfüllen. Ein lineares Programm kann in polynomieller Zeit in n, m und der Größe der Koordinaten in A, b, c gelöst werden.

Satz 7.1. *Im realisierbaren Fall können wir in polynomieller Zeit in m, d und der Größe der Koordinaten eine Hypothese $h_{\hat{w},\hat{u}} \in \mathcal{H}$ finden, die S korrekt klassifiziert (d.h. $h_{\hat{w},\hat{u}}(x^i) = y^{(i)}$ für alle i).*

Beweis. Wir können die Bedingung $h_{\hat{w},\hat{u}}(x^{(i)}) = y^{(i)}$ wie folgt ausschreiben. Gesucht sind $\hat{w} \in \mathbb{R}^d$ und $\hat{u} \in \mathbb{R}$, sodass für alle $1 \leq i \leq m$ gilt:

(i) $\left\langle \hat{w}, x^{(i)} \right\rangle \geq \hat{u}$ wenn $y^{(i)} = +1$, und

(ii) $\left\langle \hat{w}, x^{(i)} \right\rangle < \hat{u}$ wenn $y^{(i)} = -1$

Wir wollen nun schrittweise ein lineares Programm herleiten, um Werte für $\hat{w}$ und $\hat{u}$ zu finden, die (i) und (ii) erfüllt. Laut der Annahme im Satz existieren w und u, welche diese Bedingungen für $w = \hat{w}$ und $u = \hat{u}$ erfüllen. Daraus folgt

$$\max_{\substack{1 \leq i \leq m \\ y^{(i)} = -1}} \left\langle w, x^{(i)} \right\rangle < u \leq \min_{\substack{1 \leq i \leq m \\ y^{(i)} = +1}} \left\langle w, x^{(i)} \right\rangle \tag{1}$$

wobei w und u unbekannt sind. Da das Maximum auf der linken Seite über eine endliche Menge gebildet wird, existiert ein $u' \in \mathbb{R}$ mit

$$\max_{\substack{1 \leq i \leq m \\ y^{(i)} = -1}} \left\langle w, x^{(i)} \right\rangle < u' < u \leq \min_{\substack{1 \leq i \leq m \\ y^{(i)} = +1}} \left\langle w, x^{(i)} \right\rangle$$

Also gilt für alle $1 \leq i \leq m$, dass

$$y^{(i)} \left\langle w, x^{(i)} \right\rangle > y^{(i)} u'$$

Weiter können wir die rechte Seite subtrahieren und bekommen

$$y^{(i)} \left\langle w, x^{(i)} \right\rangle - y^{(i)} u' > 0$$

Es folgt, dass ein Wert $\gamma > 0$ existiert, sodass für alle $1 \leq i \leq m$

$$y^{(i)} \left\langle w, x^{(i)} \right\rangle - y^{(i)} u' \geq \gamma$$

Das können wir äquivalent umformen zu

$$\left\langle y^{(i)} x^{(i)}, w'' \right\rangle - y^{(i)} u'' \geq 1 \tag{2}$$

mit $w'' = \frac{w}{\gamma}$ und $u'' = \frac{u'}{\gamma}$.

Wir können nun die Zeilen der Matrix A des linearen Programms definieren als $(d+1)$-dimensionale Zeilenvektoren

$$a_i = (y^{(i)} x_1^{(i)}, y^{(i)} x_2^{(i)}, \ldots, y^{(i)} x_d^{(i)}, -y^{(i)})$$

für $1 \leq i \leq m$. Für b wählen wir den m-dimensionaler Spaltenvektor $(1, \ldots, 1)$ und für c den m-dimensionalen Spaltenvektor $(0, \ldots, 0)$.

Das lineare Programm findet dann ein $v = (v_1, \ldots, v_n)$ mit $Av \geq b$, sodass $\langle c, v \rangle$ maximiert wird. Dabei ist $\langle c, v \rangle = 0$ für alle $v \in \mathbb{R}^n$ und wir interessieren uns eigentlich nur für den ersten Teil der Bedingung.

Laut unserem linearen Programm haben wir dann ein v, das (2) erfüllt mit $v = (w_1'', \ldots, w_d'', u'')$. Durch unsere Herleitung aus w und u wissen wir, dass solch ein v existieren muss. Das heisst, wir können nun $w'' \in \mathbb{R}^n$ und u'' aus den Koordinaten von v ablesen. Wir wählen nun

$$\hat{w} = \frac{w''}{\|w''\|}$$

und

$$\hat{u} = \min_{\substack{1 \leq i \leq m \\ y^{(i)} = +1}} \left\langle \hat{w}, x^{(i)} \right\rangle$$

und geben diese zurück als Lösung. Tatsächlich klassifiziert die Hypothese $h_{\hat{w},\hat{u}}$ alle Punkte in S korrekt, da

$$\hat{w} = \frac{w''}{\|w''\|} = \frac{(\frac{w_1}{\gamma}, \ldots, \frac{w_d}{\gamma})}{\|(\frac{w_1}{\gamma}, \ldots, \frac{w_d}{\gamma})\|} = \frac{\frac{1}{\gamma} w}{\frac{1}{\gamma}\|w\|} = \frac{w}{\|w\|}$$

und weil aus (1) folgt, dass auch

$$\max_{\substack{1 \leq i \leq m \\ y^{(i)} = -1}} \left\langle \frac{w}{\|w\|}, x^{(i)} \right\rangle < \min_{\substack{1 \leq i \leq m \\ y^{(i)} = +1}} \left\langle \frac{w}{\|w\|}, x^{(i)} \right\rangle$$

gilt. □

2 Nicht-Realisierbarer Fall

Im nicht-realisierbaren Fall gehen wir *nicht* davon aus, dass die positive Menge und die negative Menge durch eine Hyperebene separierbar sind. In diesem Fall ist es NP-schwer einen Halbraum zu finden, der den Trainingsfehler minimiert. Wir zeigen dies im speziellen Fall der Hypothesenklasse $\mathcal{H}_0$ von Funktionen der Form $h_w : \mathbb{R}^d \to \{-1, +1\}$ mit $w \in \mathbb{R}^d$ und

$$h_w(x) = \begin{cases} +1 \text{ falls } \langle w, x \rangle \geq 0 \\ -1 \text{ sonst} \end{cases}$$

In der letzten Vorlesung hatten wir gesehen, dass diese Klasse, mithilfe einer Transformation in einen höherdimensionalen Raum, auch allgemeine lineare Klassifikatoren darstellen kann.

Wir zeigen die NP-Schwerheit des Lernproblems unter $\mathcal{H}_0$ mithilfe einer Reduktion von dem folgenden NP-schweren Problem.

Definition 7.2 (MAX-E2-SAT)**.** *Gegeben eine Menge von m Klauseln über n booleschen Variablen $x_1, \ldots, x_n$, wobei jede Klausel genau zwei Literale (negierte oder nicht-negierte Variablen) enthält. Finde eine Wahrheitsbelegung der Variablen, welche die Anzahl der erfüllten Klauseln maximiert.*

Beispiel 7.3. *Sei $\{(x_1 \vee x_2), (\overline{x_1} \vee \overline{x_2}), (\overline{x_2} \vee \overline{x_3}), (\overline{x_1} \vee x_3)\}$ eine Menge von Klauseln. Eine Wahrheitsbelegung, welche die Anzahl der erfüllten Klauseln maximiert, ist $x_1 = 1$, $x_2 = 0$, $x_3 = 1$. Diese Wahrheitsbelegung ist maximal, da alle Klauseln durch sie erfüllt werden.*

Satz 7.4 (Håstad)**.** *Falls $P \neq NP$, dann existiert kein polynomieller Algorithmus für MAX-E2-SAT. (ohne Beweis)*

Wir wollen aus dem Satz von Håstad folgern, dass auch das Lernproblem über $\mathcal{H}_0$ NP-schwer ist. Das heisst, wir wollen den folgenden Satz zeigen.

Satz 7.5. *Falls $P \neq NP$, dann existiert kein polynomieller Algorithmus, der ein $h \in \mathcal{H}_0$ findet welches den Trainingsfehler minimiert.*

Gegeben sei eine Menge $\mathcal{I}$ von m Klauseln über n Variablen $x_1, \ldots, x_n$ als Eingabe für das MAX-E2-SAT Problem. Wir transformieren diese Eingabe in eine Eingabe $\mathcal{I}'$ für das Lernproblem über $\mathcal{H}_0$. Wir definieren für jede Klausel C einen Punkt $\phi(C) \in \mathbb{R}^n$ mithilfe einer Funktion

$$\phi_j(C) = \begin{cases} 1 \text{ falls } x_j \in C \\ -1 \text{ falls } \overline{x_j} \in C \\ 0 \text{ sonst} \end{cases}$$

Sei $\phi(C) = (\phi_1(C), \ldots, \phi_n(C))$. Wir geben diesem Punkt ein positives Label.

Zusätzlich definieren wir für jede Klausel C über Variablen x_i, x_j eine Menge von vier Punkten $\{e_i, e_j, -e_i, -e_j\}$, wobei e_i den Einheitsvektor von $\mathbb{R}^n$ bezeichnet, der überall 0 ist, und nur an der $i$ten Koordinate eine 1 hat. Wir geben diesen Punkten ein negatives Label und fügen sie in zweifacher Ausführung hinzu. Die Klausel C erzeugt also eine beschriftete Menge

$$\Phi(C) = \big\{(\phi(C), +1), (e_i, -1), (e_i, -1), (e_j, -1), (e_j, -1), (-e_i, -1), (-e_i, -1), (-e_j, -1), (-e_j, -1)\big\}$$

Die Eingabe $\mathcal{I}'$ für das Lernproblem besteht nun aus der Vereinigung dieser beschrifteten Punktmengen über alle Klauseln. Beachte, dass in der erzeugten Menge Punkte mehrfach vorkommen.

Definition 7.6. *Sei $h_w \in \mathcal{H}_0$ eine Hypothese mit $w = (w_1, \ldots, w_n)$. Wir definieren eine Funktion $\alpha : \mathbb{R}^n \to \{0,1\}^n$ mit*

$$\alpha_i(w) = \begin{cases} 1 \text{ falls } w_i \geq 0 \\ 0 \text{ sonst} \end{cases}$$

als $\alpha(w) = (\alpha_1(w), \ldots, \alpha_n(w))$. Die Funktion bildet die Hypothese h_w auf eine Wahrheitsbelegung für die Variablen $x_1, \ldots, x_n$ ab, indem wir $x_i = \alpha_i(w)$ setzen.

Sei $h_w \in \mathcal{H}_0$ eine Hypothese, die den Trainingsfehler auf Eingabe $\mathcal{I}'$ minimiert. Wir behaupten, dass $\alpha(w)$ die Anzahl der erfüllten Klauseln in $\mathcal{I}$ maximiert. Um das zu zeigen, müssen wir zunächst ein paar strukturelle Eigenschaften unserer Konstruktion zeigen.

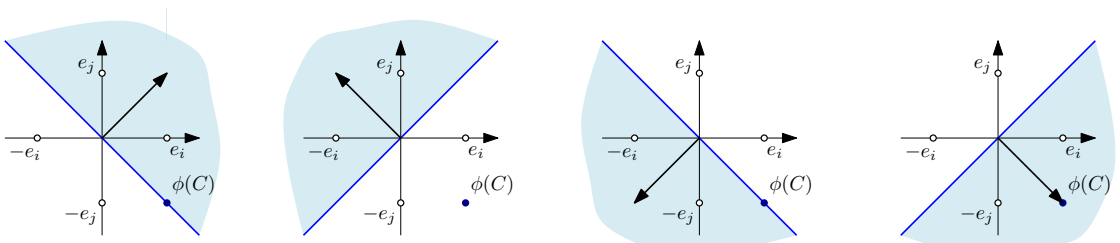
Behauptung 7.7. *Wenn für ein $k \geq 0$ eine Wahrheitsbelegung $a \in \{0,1\}^n$ existiert, die k Klauseln von $\mathcal{I}$ erfüllt, dann existiert ein $h_w \in \mathcal{H}_0$, welches $k + 4m$ Punkte in $\mathcal{I}'$ korrekt klassifiziert.*

Beweis. Dafür setzen wir

$$w_i = \begin{cases} 1 \text{ falls } a_i = 1 \\ -1 \text{ falls } a_i = 0 \end{cases}$$

Dann ist $\langle w, \phi(C) \rangle \geq 0$ genau dann wenn die Wahrheitsbelegung a die Klausel C erfüllt. Das lässt sich leicht durch eine Fallanalyse zeigen, die wir hier nicht ausführen. Ferner werden genau 4 negative Punkte von $\Phi(C)$ korrekt klassifiziert. Damit ist Behauptung 7.7 bewiesen. □

Beispiel 7.8. *Sei $C = (x_i \vee \overline{x_j})$, dann ist $\phi_i(C) = 1$ und $\phi_j(C) = -1$ und alle anderen Koordinaten von $\phi(C)$ sind gleich null. Das heisst, $\phi(C)$ liegt in dem linearen Unterraum, der durch die Einheitsvektoren e_i und e_j aufgespannt wird. Daher können wir uns die vier Hypothesen aus obigem Beweis, die den vier Wahrheitsbelegungen von x_i und x_j entsprechen, wie folgt vorstellen:*



(a) $x_i = 1$, $x_j = 1$
$(x_i \vee \overline{x_j}) = 1$

(b) $x_i = 0$, $x_j = 1$
$(x_i \vee \overline{x_j}) = 0$

(c) $x_i = 0$, $x_j = 0$
$(x_i \vee \overline{x_j}) = 1$

(d) $x_i = 1$, $x_j = 0$
$(x_i \vee \overline{x_j}) = 1$

Der Fall (b) ist die einzige Belegung, wo die Klausel nicht erfüllt ist. Das ist auch der einzige Fall, in dem $\phi(C)$ nicht korrekt klassifiziert wird. Weiter ist die Anzahl der negativen Punkte, die von h_w als negativ klassifiziert werden, immer genau $4m$. Also werden genau $k+4m$ Punkte korrekt klassifiziert. Die anderen Klauseln können auf die gleiche Art analysiert werden.

Behauptung 7.9. *Sei $h_w \in \mathcal{H}_0$ mit $w = (w_1, \ldots, w_n) \in \mathbb{R}^n$ eine Hypothese, die den Trainingsfehler minimiert, dann ist $w_i \neq 0$ für alle $1 \leq i \leq n$.*

Beweis. Sei $w_i = 0$ für eine Hypothese h_w. Sei C eine Klausel über Variablen x_i und x_j. Dann ist $\langle w, e_i \rangle \geq 0$, sowie $\langle w, -e_i \rangle \geq 0$. Gleichzeitig ist entweder $\langle w, e_j \rangle \geq 0$, oder $\langle w, -e_j \rangle \geq 0$. Da diese Punkte in zweifacher Ausführung in $\Phi(C)$ vorkommen, klassifiziert h_w also mindestens 6 Punkte von $\Phi(C)$ falsch, also höchstens 3 Punkte korrekt. Gleichzeitig klassifiziert $h_{w'}$ mit einem beliebigen $w' = (w'_1, \ldots, w'_n)$ mit $w'_j \neq 0$ für alle $1 \leq j \leq n$ mindestens 4 negative Punkte pro Klausel korrekt. Damit ist Behauptung 7.9 bewiesen. □

Behauptung 7.10. *Sei $h_w \in \mathcal{H}_0$ mit $w \in \mathbb{R}^n$ eine Hypothese, die den Trainingsfehler minimiert. Sei $\phi(C)$ ein Punkt, der durch h_w korrekt klassifiziert wird, dann wird die Klausel C durch $\alpha(w)$ erfüllt.*

Beweis. Das kann wieder durch eine Fallanalyse gezeigt werden. Sei C die Klausel $(x_i \vee x_j)$. Dann ist $\phi_i(C) = 1$ und $\phi_j(C) = 1$ und alle anderen Koordinaten sind gleich null. Daher gilt für alle $w \in \mathbb{R}^n$

$$\langle w, \phi(C) \rangle \geq 0 \quad \Leftrightarrow \quad w_i + w_j \geq 0$$

Wir unterscheiden die folgenden Fälle.

(a) $(w_i > 0, w_j > 0) \Rightarrow (x_i = 1, x_j = 1) \Rightarrow C$ ist durch $\alpha(w)$ erfüllt

(b) $(w_i > 0, w_j < 0) \Rightarrow (x_i = 1, x_j = 0) \Rightarrow C$ ist durch $\alpha(w)$ erfüllt

(c) $(w_i < 0, w_j > 0) \Rightarrow (x_i = 0, x_j = 1) \Rightarrow C$ ist durch $\alpha(w)$ erfüllt

(d) $(w_i < 0, w_j < 0) \Rightarrow (w_i + w_j < 0) \Rightarrow \phi(C)$ wird nicht korrekt klassifiziert

Wir können annehmen, dass $w_i \neq 0$ und $w_j \neq 0$, da sonst h_w nicht optimal ist (Behauptung 7.9). Somit ist die obige Fallanalyse für die betrachtete Klausel C vollständig. Die anderen Möglichkeiten für C sind die Klauseln $(x_i \vee \overline{x_j}), (\overline{x_i} \vee x_j), (\overline{x_i} \vee \overline{x_j})$. In diesen Fällen kann die Behauptung analog gezeigt werden, was wir hier nicht ausführen. Damit wäre Behauptung 7.10 bewiesen. □

Beweis von Satz 7.5. Wir können nun alles zusammenführen und unseren Satz beweisen. Laut Behauptung 7.7 existiert für jede Wahrheitsbelegung mit k erfüllten Klauseln von $\mathcal{I}$ eine Hypothese, die $k + 4m$ Punkte in $\mathcal{I}'$ korrekt klassifiziert. Gleichzeitig folgt aus Behauptung 7.9 für jedes h_w, das den Trainingsfehler auf $\mathcal{I}'$ minimiert, dass die Anzahl der negativen Punkte, die durch h_w korrekt klassifiziert werden, gleich $4m$ ist. Wenn h_w also $k + 4m$ Punkte korrekt klassifiziert, dann sind k Punkte davon positiv. Aus Behauptung 7.10 folgt dann, dass h_w eine Wahrheitsbelegung $\alpha(w)$ impliziert, die mindestens k Klauseln von $\mathcal{I}$ erfüllt. Wenn es also eine Wahrheitsbelegung gibt, die k Klauseln in $\mathcal{I}$ erfüllt, dann gibt unsere Reduktion mithilfe eines Lernalgorithmus für $\mathcal{I}'$ eine Wahrheitsbelegung zurück, die mindestens k Klauseln in $\mathcal{I}$ erfüllt. Gäbe es also einen polynomiellen Algorithmus für das Lernproblem, dann gäbe es auch einen polynomiellen Algorithmus für MAX-E2-SAT. Damit folgt Satz 7.5 aus Satz 7.4. □

Referenzen

- Foundations of Machine Learning, Kapitel 5.2.
- Understanding Machine Learning, Kapitel 9.1.1.
- Bernhard Korte und Jens Vygen, Combinatorial Optimization–Theory and Algortihms, Third Edition, Springer.
- Shai Ben-David , Nadav Eiron , Philip M. Long, "On the Difficulty of Approximately Maximizing Agreements", Journal of Computer and System Sciences, 2000.

AGML, Sommersemester 2020 Vorlesung 8 (5 Seiten)

Support Vector Machines und Konvexität

Thomas Kesselheim **Vorschau** Letzte Aktualisierung: 18. Mai 2020

Wie auch in den vergangenen Vorlesungen werden wir uns heute wieder mit linearer Klassifikation beschäftigen. Wir erinnern uns, dass die Hypothesenklasse $\mathcal{H}$ definiert ist als die Menge von Funktionen der Form $h_{\mathbf{w},u}\colon \mathbb{R}^d \to \{-1,+1\}$ für $\mathbf{w} \in \mathbb{R}^d$ und $u \in \mathbb{R}$ und

$$h_{\mathbf{w},u}(\mathbf{x}) = \begin{cases} +1 \text{ falls } \langle \mathbf{w}, \mathbf{x} \rangle \geq u \\ -1 \text{ sonst} \end{cases} .$$

Hierbei beschreibt $\langle \mathbf{w}, \mathbf{x} \rangle$ das Skalarprodukt der Vektoren $\mathbf{w}$ und $\mathbf{x}$. Wir nehmen auch wieder an, dass uns eine Trainingsmenge S von Datenpunkten mit Labels $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$ gegeben ist.

Wie wir in der letzten Vorlesung gesehen haben, können wir in Polynomialzeit eine Hypothese berechnen, die alle Datenpunkte in S korrekt klassifiziert, sofern dies möglich ist. Gleichzeitig gibt es unter der Annahme P $\neq$ NP keinen Polynomialzeitalgorithmus, der die maximale mögliche Anzahl von Punkten korrekt klassifiziert.

Beide Probleme werden wir heute erneut betrachten. Wir werden Probleme formulieren, deren Ziel es ist eine „möglichst gute“ Hypothese zu berechnen, und die gleichzeitig Polynomialzeitalgorithmen zulassen. Die Algorithmen selbst werden wir dann in den kommenden Vorlesungen besprechen.

1 Hard-SVM-Problem

Das Ziel beim Hard-SVM-Problem ist es, eine Hypothese $h_{\mathbf{w},u}$ zu finden, die alle Datenpunkte in S richtig klassifiziert unter der Annahme, dass das möglich ist. In anderen Worten sollen die positiven von den negativen Punkten linear separierbar sein. Zusätzlich sollen die Datenpunkte möglichst deutlich klassifiziert werden. Das bedeutet, dass der Abstand von der Hyperebene, die durch $\mathbf{w}$ und u definiert wird, möglichst groß sein soll. Anders formuliert soll die Hypothese auch noch möglichst lange korrekt bleiben, selbst wenn die Punkte in ihrer Umgebung verschoben werden.

Leiten wir nun zunächst eine Formel für den Abstand von einer Hyperebene her. Zur Erinnerung: Der Abstand zweier Punkte $\mathbf{v}$ und $\mathbf{v}'$ ist definiert als die Norm der Differenz der Vektoren $\|\mathbf{v} - \mathbf{v}'\|$. Wir betrachten im Folgenden nur die euklidische Norm, definiert als $\|\mathbf{v}\| = \sqrt{\langle \mathbf{v}, \mathbf{v} \rangle}$.

Lemma 8.1. *Der Abstand eines Punktes $\mathbf{x}$ von einer Hyperebene definiert durch $(\mathbf{w}, u)$ ist $\frac{1}{\|\mathbf{w}\|} |\langle \mathbf{w}, \mathbf{x} \rangle - u|$.*

Beweis. Wir definieren einen Punkt $\mathbf{v} = \mathbf{x} - c\mathbf{w}$ mit $c = \frac{1}{\|\mathbf{w}\|^2}(\langle \mathbf{w}, \mathbf{x} \rangle - u)$. Nun werden wir nachweisen, dass $\mathbf{v}$ (i) in der Hyperebene liegt, (ii) den besagten Abstand von $\mathbf{x}$ hat und (iii) kein Punkt der Hyperebene näher an $\mathbf{x}$ liegt.

Für (i) setzen wir die Definition $\mathbf{v}$ ein und erhalten

$$\langle \mathbf{w}, \mathbf{v} \rangle = \langle \mathbf{w}, \mathbf{x} - c\mathbf{w} \rangle = \langle \mathbf{w}, \mathbf{x} \rangle - c \langle \mathbf{w}, \mathbf{w} \rangle = \langle \mathbf{w}, \mathbf{x} \rangle - c\|\mathbf{w}\|^2 = u \ .$$

Also erfüllt $\mathbf{v}$ die Hyperebenegleichung.

Für (ii) nutzen wir ebenfalls die Definition von $\mathbf{v}$ und elementare Umformungen. Dies gibt uns

$$\|\mathbf{x}-\mathbf{v}\| = \|c\mathbf{w}\| = |c|\|\mathbf{w}\| = \left|\frac{1}{\|\mathbf{w}\|^2}(\langle \mathbf{w},\mathbf{x}\rangle - u)\right| \|\mathbf{w}\| = \frac{1}{\|\mathbf{w}\|}\,|\langle \mathbf{w},\mathbf{x}\rangle - u| \ .$$

Für (iii) betrachten wir nun irgendeinen anderen Punkt $\mathbf{v}'$ auf der Hyperebene. Das Quadrat dessen Abstands zu $\mathbf{x}$ berechnet sich zu

$$\|\mathbf{x}-\mathbf{v}'\|^2 = \|\mathbf{x}-\mathbf{v}+\mathbf{v}-\mathbf{v}'\|^2 = \|\mathbf{x}-\mathbf{v}\|^2+\|\mathbf{v}-\mathbf{v}'\|^2+2\left\langle \mathbf{x}-\mathbf{v},\mathbf{v}-\mathbf{v}'\right\rangle \geq \|\mathbf{x}-\mathbf{v}\|^2+2\left\langle \mathbf{x}-\mathbf{v},\mathbf{v}-\mathbf{v}'\right\rangle \ .$$

Es bleibt also nur zu zeigen, dass $\langle \mathbf{x}-\mathbf{v},\mathbf{v}-\mathbf{v}'\rangle \geq 0$. Aufgrund der Definition von $\mathbf{v}$ ist $\mathbf{x}-\mathbf{v} = c\mathbf{w}$ also

$$\left\langle \mathbf{x}-\mathbf{v},\mathbf{v}-\mathbf{v}'\right\rangle = \left\langle c\mathbf{w},\mathbf{v}-\mathbf{v}'\right\rangle = c\left(\langle \mathbf{w},\mathbf{v}\rangle - \left\langle \mathbf{w},\mathbf{v}'\right\rangle\right) = c(-u+u) = 0 \ .$$

Hierbei haben wir ausgenutzt, dass sowohl $\mathbf{v}$ als auch $\mathbf{v}'$ auf der Hyperebene liegen. □

Wir wollen nun eine Hyperebene finden, die alle Punkte $(\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_m,y_m)$ korrekt klassifiziert und außerdem unter diesen Hyperebenen den minimalen Abstand zu den Punkten maximiert. Dies können wir nun als ein Optimierungsproblem aufschreiben

$$\begin{aligned} \text{maximiere } & \min_i \frac{1}{\|\mathbf{w}\|}\,|\langle \mathbf{w},\mathbf{x}_i\rangle - u| & \\ \text{unter den Nebenbedingungen } & \langle \mathbf{w},\mathbf{x}_i\rangle - u \geq 0 & \text{falls } y_i = 1 \\ & \langle \mathbf{w},\mathbf{x}_i\rangle - u < 0 & \text{falls } y_i = -1 \end{aligned}$$

Eine optimale Lösung zu dieser Formulierung zu finden ist nicht einfach. Die Nebenbedingungen sind zwar linear, aber die Zielfunktion ist kompliziert. Deshalb schreiben wir das Problem leicht um.

Zunächst einmal stellen wir fest, dass wir mittels der Nebenbedingungen die Betragsstriche in der Zielfunktion eliminieren können. Egal ob $y_i = 1$ oder $y_i = -1$, gilt immer $|\langle \mathbf{w},\mathbf{x}_i\rangle - u| = y_i(\langle \mathbf{w},\mathbf{x}_i\rangle - u)$. So lautet unser Problem nun

$$\begin{aligned} \text{maximiere } & \min_i \frac{1}{\|\mathbf{w}\|} y_i(\langle \mathbf{w},\mathbf{x}_i\rangle - u) & \\ \text{unter den Nebenbedingungen } & \langle \mathbf{w},\mathbf{x}_i\rangle - u \geq 0 & \text{falls } y_i = 1 \\ & \langle \mathbf{w},\mathbf{x}_i\rangle - u < 0 & \text{falls } y_i = -1 \end{aligned}$$

Betrachten wir nun eine optimale Lösung $(\mathbf{w},u)$, stellen wir fest, dass niemals $\langle \mathbf{w},\mathbf{x}_i\rangle - u = 0$ für ein i sein wird, weil wir ansonsten u leicht erhöhen könnten. Dies würde den Zielfunktionswert nur verbessern und die Lösung würde weiter gültig bleiben. Außerdem erfüllt jede Lösung mit positivem Zielfunktionswert automatisch alle Nebenbedingungen. Somit vereinfacht sich das Problem dahingehend $\mathbf{w}$ und u zu finden, sodass

$$\min_i \frac{1}{\|\mathbf{w}\|} y_i(\langle \mathbf{w},\mathbf{x}_i\rangle - u)$$

maximiert wird.

Gegeben eine optimale Lösung $(\mathbf{w}, u)$, sei nun $\gamma = \min_i y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle - u)$. Betrachte $\mathbf{w}' = \frac{1}{\gamma}\mathbf{w}$, $u' = \frac{1}{\gamma}u$. Wir stellen fest, dass

$$\frac{1}{\|\mathbf{w}'\|} y_i(\langle \mathbf{w}', \mathbf{x}_i \rangle - u') = \frac{\gamma}{\|\mathbf{w}\|} y_i \left(\left\langle \frac{\mathbf{w}}{\gamma}, \mathbf{x}_i \right\rangle - \frac{u}{\gamma} \right) = \frac{1}{\|\mathbf{w}\|} y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle - u)$$

für alle i. Also hat $(\mathbf{w}', u')$ denselben Zielfunktionswert wie $(\mathbf{w}, u)$, ist also auch eine optimale Lösung. Wir können also genauso gut auch $(\mathbf{w}', u')$ suchen. Weil bei dieser Lösung $\min_i y_i(\langle \mathbf{w}', \mathbf{x}_i \rangle - u') = 1$, ist dies gleichbedeutend mit

$$\begin{aligned} &\text{minimiere } \|\mathbf{w}'\|^2 \\ &\text{unter den Nebenbedingungen } y_i(\langle \mathbf{w}', \mathbf{x}_i \rangle - u') \geq 1 && \text{für alle } i \end{aligned}$$

Diese Formulierung heißt *Hard-SVM*. In der Tat werden wir Algorithmen kennenlernen, die ein solches Optimierungsproblem lösen können.

2 Soft-SVM-Problem

Die Ergebnisse in Abschnitt 1 setzen voraus, dass die Punkte linear separierbar sind. Das heißt, dass es eine Hypothese gibt, die alle Punkte in der Menge S korrekt klassifiziert. Um den Trainingsfehler zu minimieren, müsste man eine Hypothese finden, die möglichst viele Datenpunkte korrekt klassifiziert. In der Notation, die wir nun eingeführt haben, bedeutet dies, dass die Bedingung $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + u) \leq 1$ für möglichst wenige i nicht erfüllt ist. Wie wir bereits in der letzten Vorlesung gesehen haben, ist dies jedoch NP-schwer.

Der *Soft-SVM*-Ansatz ist daher ein anderer. Wir führen bei jeder Nebenbedingung eine Variable ξ ein, wie weit sie verletzt ist. Das heißt, wir fordern nun noch, dass $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle - u) \leq 1 - \xi_i$. Es ist nun auch das Ziel, den durchschnittlichen Fehler zu minimieren.

Die neue Formulierung lautet somit

$$\begin{aligned} &\text{minimiere } \lambda\|\mathbf{w}\|^2 + \frac{1}{m}\sum_{i=1}^{m} \xi_i \\ &\text{unter den Nebenbedingungen } y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle - u) \geq 1 - \xi_i && \text{für alle } i \\ &\qquad\qquad \xi_i \geq 0 && \text{für alle } i \end{aligned}$$

Hierbei drückt $\lambda \geq 0$ eine Gewichtung aus: $\|\mathbf{w}\|^2$ ist der Term, der ursprünglich ausgedrückt hat, dass der Abstand möglichst groß sein soll; $\frac{1}{m}\sum_{i=1}^{m} \xi_i$ ist der durchschnittliche Fehler, der misst, wie weit Punkte jeweils auf der falschen Seite der Hyperebene sind.

Dieses Problem können wir noch umformulieren. Wir nutzen aus, dass in einer optimalen Lösung immer $\xi_i = \max\{0, 1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle - u)\}$ sein wird. Damit ist es äquivalent,

$$\lambda\|\mathbf{w}\|^2 + \frac{1}{m}\sum_{i=1}^{m} \max\{0, 1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle - u)\}$$

zu minimieren.

3 Konvexe Optimierung

Das Hard-SVM- und das Soft-SVM-Problem lassen sich wie folgt darstellen. Wir möchten eine Funktion $f\colon S \to \mathbb{R}$ minimieren, wobei $S \subseteq \mathbb{R}^n$ die Menge aller zulässigen Lösungen darstellt.

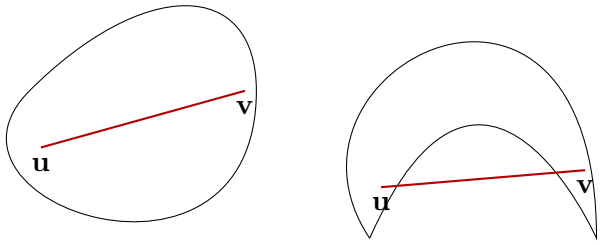


Abbildung 1: Links eine konvexe Menge, rechts eine nicht-konvexe Menge.

In unserem Fall enthält S alle $(d+1)$-dimensionalen zulässigen Vektoren $(\mathbf{w}, u)$. Das heißt, wir fügen unter die d Komponenten von $\mathbf{w}$ mit u einer weitere Komponente an. Im Fall von Soft-SVM gibt es keine weiteren Einschränkungen, also ist $S = \mathbb{R}^n$ mit $n = d+1$. Im Fall von Hard-SVM müssen mittels S die Nebenbedingungen berücksichtigen.

Glücklicherweise sind sowohl die Menge S als auch die Funktion f konvex. Deshalb werden wir die Probleme mithilfe von Algorithmen aus der Konvexen Optimierung lösen können.

Die Menge S ist jeweils *konvex*. Das heißt, dass für zwei Punkte $\mathbf{u}, \mathbf{v} \in S$ alle Punkte auf der Verbindungslinie wieder in S enthalten ist (siehe Abbildung 1). Formal also $\lambda\mathbf{u} + (1-\lambda)\mathbf{v} \in S$ für alle $\lambda \in [0,1]$.

Zusätzlich ist auch die Funktion f konvex. Das bedeutet, dass der Funktionsgraph zwischen zwei Punkten jeweils unterhalb der Verbindungslinie dieser beiden Punkte liegt. Das heißt, für $\mathbf{u}, \mathbf{v} \in S$ gilt $f(\lambda\mathbf{u} + (1-\lambda)\mathbf{v}) \leq \lambda f(\mathbf{u}) + (1-\lambda) f(\mathbf{v})$ für alle $\lambda \in [0,1]$. Ein typisches Beispiel einer konvexen Funktion, das man immer im Kopf haben sollte, ist eine quadratische Funktion in einer Dimension (siehe Abbildung 2 links).

Wenn die Funktionen differenzierbar sind, gibt es viele äquivalente Definitionen von Konvexität. Betrachten wir zunächst den eindimensionalen Fall. Hier muss beispielsweise die zweite Ableitung nicht-negativ sein. Im Kontext von Konvexer Optimierung werden wir jedoch folgende äquivalente Definition nutzen: Die Funktion fällt niemals unterhalb ihre Tangenten. Ausgedrückt in der ersten Ableitung bedeutet dies, dass eine differenzierbare Funktion $f\colon S \to \mathbb{R}$ konvex ist, wenn für alle $u, v \in S$ gilt

$$f(u) \geq f(v) + f'(v)(u - v) \ .$$

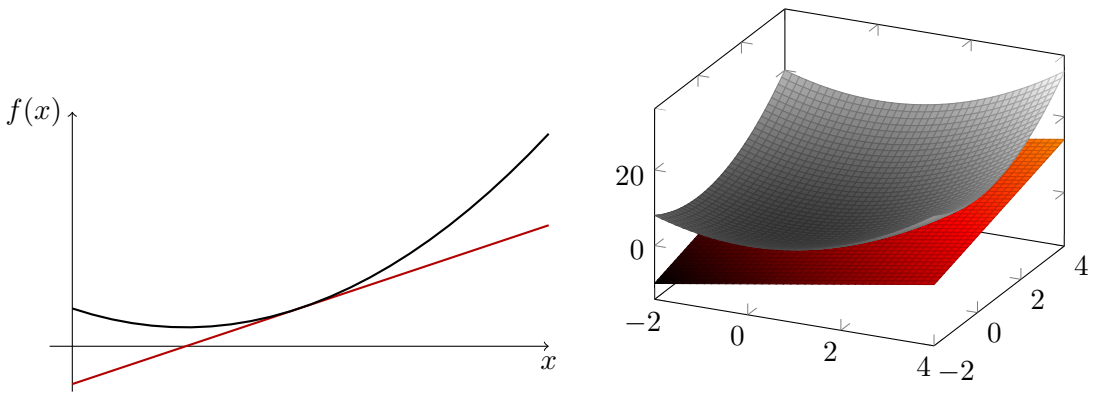


Abbildung 2: Typische konvexe Funktionen in einer bzw. zwei Dimensionen, jeweils mit einer Tangente bzw. Tangentialhyperebene in rot.

All diese Definitionen lassen sich auch ins Mehrdimensionale übertragen. Die Funktion f hat nun einen Gradienten ∇f, der der Vektor aller partiellen Ableitungen ist; $(\nabla f(\mathbf{u}))_i = \frac{\partial f}{\partial u_i}(\mathbf{u})$. Eine differenzierbare Funktion $f\colon S \to \mathbb{R}^n$ ist konvex, wenn sie niemals unter ihre Tangentialhyperebene fällt (siehe Abbildung 2 rechts). Das heißt, dass für alle $\mathbf{u}, \mathbf{v}$

$$f(\mathbf{u}) \geq f(\mathbf{v}) + \langle \nabla f(\mathbf{v}), (\mathbf{u} - \mathbf{v}) \rangle \ . \tag{1}$$

Die Soft-SVM-Zielfunktion ist nicht differenzierbar. Trotzdem erfüllt sie eine ähnliche Bedingung, wie wir sehen werden.

Wir werden nicht nachweisen, dass die Hard- und Soft-SVM-Zielfunktionen konvex sind. Dies folgt aus relativ einfachen Rechnungen. Folgende Abschlusseigenschaften sind dabei hilfreich.

Lemma 8.2. *1. Sind f und g konvex, dann sind auch $f + g$ und $\max\{f, g\}$ konvex.*

2. Ist f konvex und $\alpha \geq 0$, dann ist auch αf konvex.

3. Sind f und g konvex und g zusätzlich monoton steigend, dann ist auch $g \circ f$ konvex.

AGML, Sommersemester 2020 Vorlesung 9 (5 Seiten)

Gradient Descent

Thomas Kesselheim Letzte Aktualisierung: 22. Mai 2020

In der letzten Vorlesung haben wir in Form von Hard- und dem Soft-SVM-Problem bereits zwei konvexe Optimierungsprobleme kennengelernt. Im Maschinellen Lernen gibt es eine Vielzahl weiterer derartiger Probleme. Heute werden wir diskutieren, mit welchen algorithmischen Ansätzen man sie lösen kann.

Allgemein ist ein konvexes Optimierungsproblem wie folgt definiert. Wir müssen eine konvexe Funktion $f\colon S \to \mathbb{R}$ minimieren, wobei $S \subseteq \mathbb{R}^n$ die (konvexe) Menge aller zulässigen Lösungen darstellt. Wir beschränken uns auf den Fall, dass $S = \mathbb{R}^n$. Das heißt, es gibt keine Nebenbedingungen.

Zunächst beschränken wir uns auf differenzierbare Funktionen f. Später werden wir jedoch unsere Ergebnisse verallgemeinern, dass sie auch mit nicht-differenzierbaren Funktionen anwendbar sind.

1 Gradienten

Betrachten wir zunächst eine differenzierbare Funktion f. Folglich hat sie einen Gradienten ∇f, der der Vektor aller partiellen Ableitungen ist; $(\nabla f(\mathbf{u}))_i = \frac{\partial f}{\partial u_i}(\mathbf{u})$. Konvexität von f ist nun äquivalent dazu, dass für alle $\mathbf{u}, \mathbf{v}$

$$f(\mathbf{u}) \geq f(\mathbf{v}) + \langle \nabla f(\mathbf{v}), (\mathbf{u} - \mathbf{v}) \rangle \ . \tag{1}$$

Zum Verständnis dieser Ungleichung ist es hilfreich zu verstehen, dass

$$\mathbf{u} \mapsto f(\mathbf{v}) + \langle \nabla f(\mathbf{v}), (\mathbf{u} - \mathbf{v}) \rangle$$

die lineare Approximation von f durch die Tangentialhyperebene an der Stelle $\mathbf{v}$ ist. Das heißt, eine konvexe Funktion muss jeweils oberhalb der Tangentialhyperebene liegen.

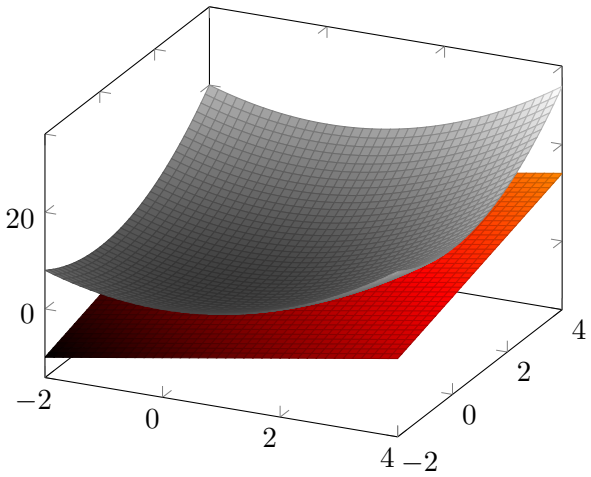


Abbildung 1: $f\colon \mathbb{R}^2 \to \mathbb{R}$ mit $f(x_1, x_2) = x_1^2 + x_2^2$ und die Tangentialebene an $(1, 1)$.

Beispiel 9.1. *In Abbildung 1 ist die Funktion* $f\colon \mathbb{R}^2 \to \mathbb{R}$ *mit* $f(x_1, x_2) = x_1^2 + x_2^2$ *und die Tangentialebene an* f *an* $(1, 1)$ *dargestellt. Der Gradient ist* $\nabla f(x_1, x_2) = \begin{pmatrix} 2x_1 \\ 2x_2 \end{pmatrix}$*, entsprechend ist die Tangentialebene an* $(1, 1)$ *gegeben durch*

$$\mathbf{u} \mapsto 2 + 2(u_1 - 1) + 2(u_2 - 1) \ .$$

Abbildung 2 zeigt eine andere Darstellung einer Funktion $f\colon \mathbb{R}^2 \to \mathbb{R}$ als Relief-Plot. Hier sehen wir Höhenlinien der Funktion eingetragen, also Mengen von Punkten, an denen die Funktion denselben Wert hat. Der Gradient im Punkt $\mathbf{x}$ steht immer senkrecht zur Höhenlinie im Punkt $\mathbf{x}$ der Funktion. Er zeigt in die Richtung des stärksten Anstiegs

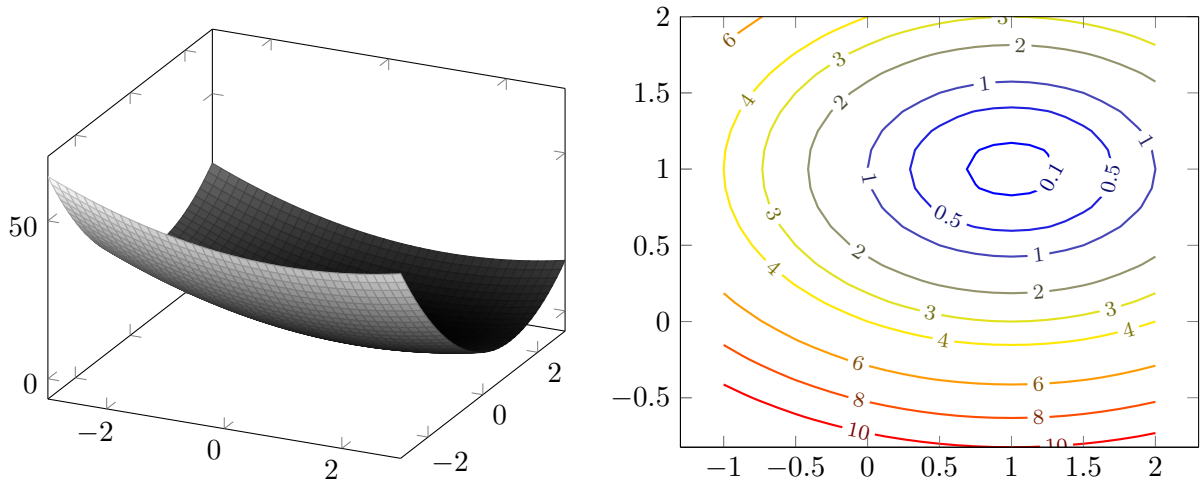


Abbildung 2: Links ein 3D-Plot, rechts ein Relief-Plot mit Höhenlinien der konvexen Funktion $f\colon \mathbb{R}^2 \to \mathbb{R}$ definiert durch $f(x_1, x_2) = (x_1 - 1)^2 + 3(x_2 - 1)^2$.

2 Gradient Descent

Der Algorithmus *Gradient Descent* berechnet eine Folge von Lösungen $\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(T)}$. Wir beginnen mit $\mathbf{w}^{(1)} = 0$. Die Lösung $\mathbf{w}^{(t+1)}$ ergibt sich jeweils aus einer leichten Verbesserung von $\mathbf{w}^{(t)}$.

Betrachten wir hierfür den Gradienten $\mathbf{g}^{(t)} := \nabla f(\mathbf{w}^{(t)})$ von f an der Stelle $\mathbf{w}^{(t)}$. Weil der Gradient in die Richtung des stärksten Anstiegs zeigt, müssen wir uns in die entgegengesetzte Richtung, also $-\mathbf{g}^{(t)}$ bewegen, denn dies ist die Richtung des stärksten Abfalls. Dies führt zur Regel

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{g}^{(t)} \ .$$

Dabei ist η (ausgesprochen: *eta*) ein Parameter des Algorithmus. Wenn wir η zu klein wählen, machen wir keine guten Fortschritte. Wenn wir η zu groß wählen, schießen wir möglicherweise über das Ziel hinaus.

Nach einer festen Anzahl von Iterationen T geben wir die beste gesehene Lösung zurück.[1]

[1] Alternative Formulierungen des Algorithmus geben einen Durchschnitt über alle Lösungen oder die letzte erreichte Lösung zurück.

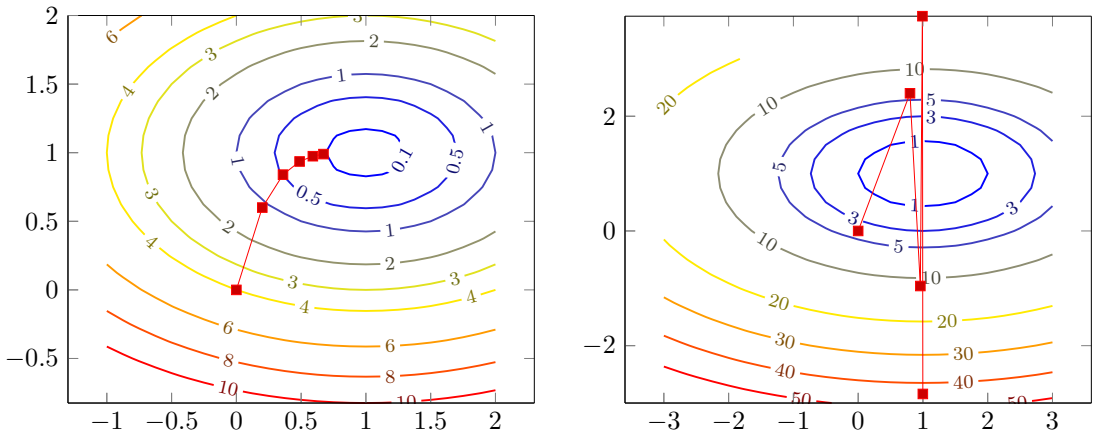


Abbildung 3: Beispiel von Gradient Descent auf $f(x_1, x_2) = (x_1 - 1)^2 + 3(x_2 - 1)^2$, links mit $\eta = 0.3$ mit Konvergenz, rechts mit $\eta = 0.4$ ohne Konvergenz.

3 Analyse von Gradient Descent

Wir können nun zeigen, dass der Algorithmus sich tatsächlich einer optimale Lösung annähert.

Satz 9.2. *Gilt* $\|\mathbf{g}^{(t)}\| \leq \rho$ *für alle* t*, dann gilt für alle* $\mathbf{w}^* \in \mathbb{R}^n$ *mit* $\|\mathbf{w}^*\| \leq B$

$$\min_t f(\mathbf{w}^{(t)}) \leq f(\mathbf{w}^*) + \frac{B^2}{2\eta T} + \frac{\eta \rho^2}{2} \ .$$

Insbesondere gilt für $\eta = \frac{B}{\rho\sqrt{T}}$

$$\min_t f(\mathbf{w}^{(t)}) \leq f(\mathbf{w}^*) + \frac{B\rho}{\sqrt{T}} \ .$$

Insbesondere können wir natürlich $\mathbf{w}^*$ als die optimale Lösung wählen und erhalten damit einen additiven Fehler von höchstens $\frac{B\rho}{\sqrt{T}}$ unter den genannten Bedingungen. Wichtig ist an dieser Stelle, dass der Fehler immer kleiner wird je größer T, also die Anzahl der Iterationen, wird. Die Bedeutungen von B und ρ werden wir später noch diskutieren.

Beweis. Den besten gesehenen Funktionswert können wir abschätzen durch den durchschnittlich gesehenen Funktionswert

$$\min_t \left(f(\mathbf{w}^{(t)}) - f(\mathbf{w}^*) \right) \leq \frac{1}{T} \sum_{t=1}^{T} \left(f(\mathbf{w}^{(t)}) - f(\mathbf{w}^*) \right) \ . \tag{2}$$

Nun folgt der einzige Schritt, in dem wir Konvexität nutzen. Gemäß dieser gilt für alle t

$$f(\mathbf{w}^*) \geq f(\mathbf{w}^{(t)}) + \left\langle \mathbf{g}^{(t)}, \mathbf{w}^* - \mathbf{w}^{(t)} \right\rangle \ . \tag{3}$$

Also gilt

$$f(\mathbf{w}^{(t)}) - f(\mathbf{w}^*) \leq \left\langle \mathbf{g}^{(t)}, \mathbf{w}^{(t)} - \mathbf{w}^* \right\rangle \ .$$

Dieses Skalarprodukt drücken wir nun in einer Summe von Vektornormen aus. Es gilt nämlich für alle $\mathbf{u}, \mathbf{v}$, dass

$$\langle \mathbf{u} + \mathbf{v}, \mathbf{u} + \mathbf{v} \rangle = \|\mathbf{u} + \mathbf{v}\|^2 \ ,$$

aber auch

$$\langle \mathbf{u}+\mathbf{v}, \mathbf{u}+\mathbf{v}\rangle = \langle \mathbf{u},\mathbf{u}\rangle + \langle \mathbf{v},\mathbf{u}\rangle + \langle \mathbf{u},\mathbf{v}\rangle + \langle \mathbf{v},\mathbf{v}\rangle = \|\mathbf{u}\|^2 + \|\mathbf{v}\|^2 + 2\langle \mathbf{u},\mathbf{v}\rangle \ .$$

Zusammengenommen also

$$\langle \mathbf{u},\mathbf{v}\rangle = \frac{1}{2}\left(\|\mathbf{u}+\mathbf{v}\|^2 - \|\mathbf{u}\|^2 - \|\mathbf{v}\|^2\right) \ .$$

Mittels der Gleichung können wir nun schreiben

$$\left\langle \mathbf{w}^{(t)} - \mathbf{w}^*, -\eta\mathbf{g}^{(t)}\right\rangle = \frac{1}{2}\left(\|\mathbf{w}^{(t)} - \mathbf{w}^* - \eta\mathbf{g}^{(t)}\|^2 - \|\mathbf{w}^{(t)} - \mathbf{w}^*\|^2 - \|-\eta\mathbf{g}^{(t)}\|^2\right) \ .$$

Wir können nun $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta\mathbf{g}^{(t)}$ einsetzen. Zusätzlich teilen wir die Gleichung durch $-\eta$. Somit ergibt sich

$$\begin{aligned}\left\langle \mathbf{g}^{(t)}, \mathbf{w}^{(t)} - \mathbf{w}^*\right\rangle &= -\frac{1}{\eta}\left\langle \mathbf{w}^{(t)} - \mathbf{w}^*, -\eta\mathbf{g}^{(t)}\right\rangle \\ &= -\frac{1}{2\eta}\left(\|\mathbf{w}^{(t+1)} - \mathbf{w}^*\|^2 - \|\mathbf{w}^{(t)} - \mathbf{w}^*\|^2 - \|-\eta\mathbf{g}^{(t)}\|^2\right) \\ &= \frac{1}{2\eta}\left(\|\mathbf{w}^{(t)} - \mathbf{w}^*\|^2 - \|\mathbf{w}^{(t+1)} - \mathbf{w}^*\|^2\right) + \frac{\eta}{2}\|\mathbf{g}^{(t)}\|^2 \ .\end{aligned}$$

Als Teleskopsumme ergibt sich damit für (2) zusammen mit (3)

$$\begin{aligned}\sum_{t=1}^{T}\left(f(\mathbf{w}^{(t)}) - f(\mathbf{w}^*)\right) &\le \frac{1}{2\eta}\sum_{t=1}^{T}\left(\|\mathbf{w}^{(t)} - \mathbf{w}^*\|^2 - \|\mathbf{w}^{(t+1)} - \mathbf{w}^*\|^2\right) + \frac{\eta}{2}\sum_{t=1}^{T}\|\mathbf{g}^{(t)}\|^2 \\ &= \frac{1}{2\eta}\left(\|\mathbf{w}^{(1)} - \mathbf{w}^*\|^2 - \|\mathbf{w}^{(T+1)} - \mathbf{w}^*\|^2\right) + \frac{\eta}{2}\sum_{t=1}^{T}\|\mathbf{g}^{(t)}\|^2 \ .\end{aligned}$$

Mit $\mathbf{w}^{(1)} = 0$ und $\|\mathbf{w}^{(T+1)} - \mathbf{w}^*\|^2 \ge 0$ können wir also abschätzen

$$\sum_{t=1}^{T}\left(f(\mathbf{w}^{(t)}) - f(\mathbf{w}^*)\right) \le \frac{1}{2\eta}\|\mathbf{w}^*\|^2 + \frac{\eta}{2}\sum_{t=1}^{T}\|\mathbf{g}^{(t)}\|^2 \ .$$

Insgesamt erhalten wir damit

$$\min_t\left(f(\mathbf{w}^{(t)}) - f(\mathbf{w}^*)\right) \le \frac{B^2}{2\eta T} + \frac{\eta\rho^2}{2} \ .$$

Und mit $\eta = \frac{B}{\rho\sqrt{T}}$ gilt nun $\min_t\left(f(\mathbf{w}^{(t)}) - f(\mathbf{w}^*)\right) \le \frac{B\rho}{\sqrt{T}}$. □

4 Nicht-Differenzierbare Funktionen

Ist eine Funktion nicht differenzierbar, so gibt es nicht an jeder Stelle $\mathbf{v}$ einen Gradienten $\nabla f(\mathbf{v})$. Somit ist auch die Tangentialhyperebene nicht (eindeutig) definiert. In Abbildung 4 ist die Betragsfunktion dargestellt. An der Stelle 0 ist sie nicht differenzierbar. Es gibt nun eine Vielzahl von Tangenten, die wir an dieser Stelle anlegen können. Die Abbildung zeigt zwei Beispiele.

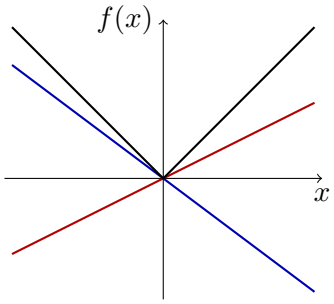


Abbildung 4: Die Betragsfunktion mit zwei möglichen Tangenten an der Stelle 0. Die Funktion liegt oberhalb von allen diesen Tangenten.

Wir erinnern uns, das Konvexität bei differenzierbaren Funktionen f äquivalent dazu ist, dass für alle $\mathbf{u}, \mathbf{v}$

$$f(\mathbf{u}) \geq f(\mathbf{v}) + \langle \nabla f(\mathbf{v}), (\mathbf{u} - \mathbf{v}) \rangle \ .$$

Diese Ungleichung haben wir für die Analyse von Gradient Descent genutzt.

Für allgemeine, nicht notwendigerweise differenzierbare Funktionen gibt es glücklicherweise folgende Verallgemeinerung: Eine Funktion f ist konvex, wenn es für alle $\mathbf{v}$ ein $\mathbf{g}$ gibt, sodass für alle $\mathbf{u}$ gilt

$$f(\mathbf{u}) \geq f(\mathbf{v}) + \langle \mathbf{g}, (\mathbf{u} - \mathbf{v}) \rangle \ . \tag{4}$$

Wenn f in $\mathbf{v}$ differenzierbar ist, dann ist tatsächlich $\mathbf{g} = \nabla f(\mathbf{v})$ die einzige Wahl, die diese Ungleichung erfüllt. Ist f in $\mathbf{v}$ nicht differenzierbar, gibt es möglicherweise mehrere Möglichkeiten, $\mathbf{g}$ zu wählen.

Definition 9.3. *Für eine Funktion $f \colon S \to \mathbb{R}$ und $\mathbf{v} \in S$ nennen wir*

$$\partial f(\mathbf{v}) = \{\mathbf{g} \mid f(\mathbf{u}) \geq f(\mathbf{v}) + \langle \mathbf{g}, (\mathbf{u} - \mathbf{v}) \rangle \text{ für alle } \mathbf{u} \in S\}$$

das Subdifferenzial *von f in $\mathbf{v}$. Die Elemente von $\partial f(\mathbf{v})$ heißen* Subgradienten.

Eine Funktion f ist also genau dann konvex, wenn $\partial f(\mathbf{v}) \neq \emptyset$ für alle $\mathbf{v}$. Dies liegt daran, dass Wahlen für $\mathbf{g}$ in Ungleichung (4) genau den Elementen aus $\partial f(\mathbf{v})$ entsprechen.

5 Subgradient Descent

Der Algorithmus Subgradient Descent funktioniert genauso wie Gradient Descent. Der einzige Unterschied ist die Wahl von $\mathbf{g}^{(t)}$. Galt bisher die Regel, dass $\mathbf{g}^{(t)}$ auf $\nabla f(\mathbf{w}^{(t)})$ gesetzt wurde, ist nun $\mathbf{g}^{(t)} \in \partial f(\mathbf{w}^{(t)})$ beliebig. Das heißt, dass wir anstatt des Gradienten nun einen beliebigen Subgradienten verwenden. Für differenzierbare Funktionen ändert sich damit nichts.

Satz 9.2 und sein Beweis gelten weiterhin. Lediglich in Ungleichung (3) haben wir die Definition von $\mathbf{g}^{(t)}$ genutzt. Diese Ungleichung entspricht jedoch genau der Definition des Subgradienten.

Referenzen

- Understanding Machine Learning, Kapitel 14.1–14.2

AGML, Sommersemester 2020 Vorlesung 10 (6 Seiten)

Stochastic Gradient Descent

Thomas Kesselheim Letzte Aktualisierung: 26. Mai 2020

Wir betrachten heute wie der Gradient-Descent-Algorithmus auf dem Soft-SVM-Problem abläuft. Wir werden in diesem Zusammenhang eine Verallgemeinerung des Algorithmus namens *Stochastic Gradient Descent* kennenlernen, die schnellere Laufzeiten ermöglicht.

1 Soft-SVM: Wiederholung und neue Notation

Wir erinnern uns, dass uns beim Soft-SVM-Problem eine Menge S von Datenpunkten mit Labels $\mathbf{z}_1 = (\mathbf{x}_1, y_1), \dots, \mathbf{z}_m = (\mathbf{x}_m, y_m)$ gegeben ist, wobei $\mathbf{x}_i \in \mathbb{R}^d$ und $y_i \in \{-1, +1\}$ für alle i. Das Ziel ist es nun $\mathbf{w} \in \mathbb{R}^d$ und $u \in \mathbb{R}$ zu finden, so dass

$$\lambda \|\mathbf{w}\|^2 + \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle - u)\}$$

minimiert wird, wobei λ ein Parameter ist. Um die Notation einfach zu halten, fordern wir im Folgenden $u = 0$. Dies ist mehr oder weniger ohne Beschränkung der Allgemeinheit, wenn wir u als die $d+1$-te Komponente von $\mathbf{w}$ interpretieren und an alle $\mathbf{x}_i$ als letzte Komponente 1 anfügen. Zu einem anderen Zeitpunkt werden wir diese Anspekte noch genauer diskutieren.

Führen wir an dieser Stelle etwas Notation ein. Definiere nun

$$\ell^{\text{hinge}}(h_{\mathbf{w}}, \mathbf{z}_i) = \max\{0, 1 - y_i \langle \mathbf{w}, \mathbf{x}_i \rangle\} \ ,$$

das ausdrückt, „wie falsch“ die Hypothese $h_{\mathbf{w}}$ auf dem i-ten Datenpunkt $\mathbf{z}_i = (\mathbf{x}_i, y_i)$ ist. Diese Funktion nennt sich *Hinge Loss*. Der Name bezieht sich darauf, dass der Funktionsgraph aussieht wie ein Türscharnier (siehe Abbildung 1). Der durchschnittliche Loss auf S ist nun

$$L_S^{\text{hinge}}(h_{\mathbf{w}}) = \frac{1}{m} \sum_{i=1}^{m} \ell^{\text{hinge}}(h_{\mathbf{w}}, \mathbf{z}_i) \ .$$

Wir müssen also $\mathbf{w} \in \mathbb{R}^d$ finden, sodass $f(\mathbf{w}) := R(\mathbf{w}) + L_S^{\text{hinge}}(h_{\mathbf{w}})$ minimiert wird, wobei $R(\mathbf{w}) = \lambda \|\mathbf{w}\|^2$. Auf die Bedeutung von $R(\mathbf{w})$ werden wir in einer späteren Vorlesung eingehen.

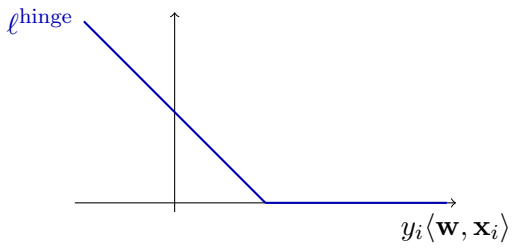


Abbildung 1: Die Hinge-Loss-Funktion.

2 Gradient Descent für Soft-SVM

Diese Funktion f ist konvex. Wir können also Gradient Descent nutzen, um sie zu minimieren. Genauer gesagt müssen wir Subgradient Descent nutzen, denn sie ist nicht überall differenzierbar.

Betrachten wir der Einfachheit halber eine Stelle $\mathbf{w}$, an der sie differenzierbar ist. Der Gradient ist der Vektor aller partiellen Ableitungen. Die partielle Ableitung nach w_j können wir mittels der üblichen Rechenregeln berechnen

$$\frac{\partial}{\partial w_j} f(\mathbf{w}) = \frac{\partial}{\partial w_j} R(\mathbf{w}) + \frac{\partial}{\partial w_j} L_S^{\text{hinge}}(h_{\mathbf{w}}) = \frac{\partial}{\partial w_j} R(\mathbf{w}) + \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial w_j} \ell^{\text{hinge}}(h_{\mathbf{w}}, \mathbf{z}_i) \ . \tag{1}$$

Weiterhin gelten

$$\frac{\partial}{\partial w_j} R(\mathbf{w}) = 2\lambda w_j \quad \text{und} \quad \frac{\partial}{\partial w_j} \ell^{\text{hinge}}(h_{\mathbf{w}}, \mathbf{z}_i) = \begin{cases} -y_i x_{i,j} & \text{falls } 1 - y_i \langle \mathbf{w}, \mathbf{x}_i \rangle > 0 \\ 0 & \text{falls } 1 - y_i \langle \mathbf{w}, \mathbf{x}_i \rangle < 0 \end{cases}$$

Also gilt insgesamt

$$\nabla f(\mathbf{w}) = 2\lambda \mathbf{w} - \frac{1}{m} \sum_{i : 1 - y_i \langle \mathbf{w}, \mathbf{x}_i \rangle > 0} y_i \mathbf{x}_i \ .$$

Wenn wir dies also in die Iterationsvorschrift von Gradient Descent $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla f(\mathbf{w}^{(t)})$ einsetzen, ergibt sich

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \left(2\lambda \mathbf{w}^{(t)} - \frac{1}{m} \sum_{i : 1 - y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle > 0} y_i \mathbf{x}_i \right) = (1 - 2\eta\lambda) \mathbf{w}^{(t)} + \frac{\eta}{m} \sum_{i : 1 - y_i \langle \mathbf{w}, \mathbf{x}_i \rangle > 0} y_i \mathbf{x}_i \ .$$

Der Algorithmus ist so also überraschend einfach. Hinsichtlich der Laufzeit einer einzelnen Iteration stellen wir fest, dass diese durch die Berechnung des Gradienten dominiert wird. Pro Dimension benötigen wir lineare Zeit in der Anzahl Samples m, insgesamt also $\Theta(dm)$. Das Problem hierbei ist, dass m typischerweise sehr groß sein sollte, denn die Stärke des Maschinellen Lernens liegt genau darin, aus der großen Menge an verfügbaren Daten Schlüsse zu ziehen.

3 Stochastic (Sub-) Gradient Descent

Die aufwändige Berechnung des Gradienten können wir wie folgt umgehen. Wie wir in Gleichung (1) sehen, ergibt sich die partielle Ableitung der Funktion f aus dem Durchschnitt der partiellen Ableitungen der Loss-Funktionen der einzelnen Datenpunkte. Diese Durchschnitt ersetzen wir nun durch ein Zufallsexperiment: Wir ziehen einen einzelnen Datenpunkt $\mathbf{z}_i$ und betrachten nur die partielle Ableitung, die sich für diesen einzelnen Punkt ergibt. Im Erwartungswert ergibt sich damit genau die gewünschte partielle Ableitung und damit auch Richtung für Gradient Descent.

Allgemeiner funktioniert der Algorithmus *Stochastic Gradient Descent* für eine beliebige konvexe Funktion f wie folgt. Wir beginnen wieder mit $\mathbf{w}^{(1)} = 0$. In Schritt t bestimmen wir $\mathbf{w}^{(t+1)}$ aus $\mathbf{w}^{(t)}$ wie folgt.

- Ziehe einen Vektor $\mathbf{g}^{(t)}$ aus irgendeiner Wahrscheinlichkeitsverteilung, sodass $\mathbf{E}\left[\mathbf{g}^{(t)} \mid \mathbf{w}^{(t)}\right] \in \partial f(\mathbf{w}^{(t)})$.[1]
- Setze $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{g}^{(t)}$.

[1] Diese Notation bedeutet, dass der *bedingte* Erwartungswert betrachtet wird. Der Vektor $\mathbf{w}^{(t)}$ wird also festgehalten und nun wird ein weiteres Zufallsexperiment durchgeführt, das von $\mathbf{w}^{(t)}$ abhängt.

4 Stochastic Subgradient Descent angewendet auf Soft-SVM

Im Fall von Soft-SVM hatten wir ja für Gradient Descent

$$\mathbf{g}^{(t)} = \nabla R(\mathbf{w}^{(t)}) + \frac{1}{m}\sum_{i=1}^{m} \nabla \ell^{\text{hinge}}(h_{\mathbf{w}^{(t)}}, \mathbf{z}_i)$$

gesetzt. Nun ziehen wir in jedem Schritt t ein I_t unabhängig, identisch verteilt aus $\{1, \dots, m\}$ und setzen

$$\mathbf{g}^{(t)} = \nabla R(\mathbf{w}^{(t)}) + \nabla \ell^{\text{hinge}}(h_{\mathbf{w}^{(t)}}, \mathbf{z}_{I_t}) = 2\lambda \mathbf{w}^{(t)} + \begin{cases} -y_{I_t}\mathbf{x}_{I_t} & \text{falls } 1 - y_{I_t}\langle \mathbf{w}^{(t)}, \mathbf{x}_{I_t}\rangle > 0 \\ 0 & \text{sonst} \end{cases} . \quad (2)$$

Anders formuliert erhalten wir

$$\mathbf{w}^{(t+1)} = \begin{cases} (1-\eta\lambda)\mathbf{w}^{(t)} + \eta y_i \mathbf{x}_i & \text{falls } 1 - y_i\langle \mathbf{w}, \mathbf{x}_i\rangle > 0 \\ (1-\eta\lambda)\mathbf{w}^{(t)} & \text{sonst} \end{cases} .$$

Nun gilt

$$\mathbf{E}\left[\mathbf{g}^{(t)} \;\middle|\; \mathbf{w}^{(t)}\right] = \sum_{i=1}^{m} \mathbf{Pr}\left[I_t = i\right]\left(\nabla R(\mathbf{w}^{(t)}) + \nabla \ell^{\text{hinge}}(h_{\mathbf{w}^{(t)}}, \mathbf{z}_i)\right) = \nabla R(\mathbf{w}^{(t)}) + \frac{1}{m}\sum_{i=1}^{m} \nabla \ell^{\text{hinge}}(h_{\mathbf{w}^{(t)}}, \mathbf{z}_i) .$$

Der bedingte Erwartungswert von $\mathbf{g}^{(t)}$ ist somit also genau der Gradient, den Gradient Descent nutzen würde.

5 Analyse von Stochastic (Sub-) Gradient Descent

Die allgemeine Formulierung von Stochastic (Sub-) Gradient Descent fordert nur $\mathbf{E}\left[\mathbf{g}^{(t)} \;\middle|\; \mathbf{w}^{(t)}\right] \in \partial f(\mathbf{w}^{(t)})$. Eine Möglichkeit wäre es also auch, den Vektor $\mathbf{g}^{(t)}$ deterministisch zu bestimmen als einen Subgradienten von f. Genau dies macht der Algorithmus Gradient Descent bzw. Subgradient Descent. Stochastic (Sub-) Gradient Descent ist also eine Verallgemeinerung. Trotzdem können wir genau dieselbe Garantie herleiten.

Satz 10.1. *Gilt $\|\mathbf{g}^{(t)}\| \leq \rho$ für alle t mit Wahrscheinlichkeit 1, dann gilt für alle $\mathbf{w}^* \in \mathbb{R}^n$ mit $\|\mathbf{w}^*\| \leq B$*

$$\mathbf{E}\left[\min_t f(\mathbf{w}^{(t)})\right] \leq f(\mathbf{w}^*) + \frac{B^2}{2\eta T} + \frac{\eta\rho^2}{2} .$$

Insbesondere gilt für $\eta = \frac{B}{\rho\sqrt{T}}$

$$\mathbf{E}\left[\min_t f(\mathbf{w}^{(t)})\right] \leq f(\mathbf{w}^*) + \frac{B\rho}{\sqrt{T}} .$$

Wir erhalten also im Wesentlichen die gleiche Garantie wie bei Gradient Descent mit dem Unterschied, dass sie nur im Erwartungswert gilt. Das folgende Lemma fasst die wesentliche Änderung im Argument zusammen.

Lemma 10.2. *Bei Stochastic (Sub-) Gradient Descent gilt für alle t*

$$\mathbf{E}\left[f(\mathbf{w}^{(t)}) - f(\mathbf{w}^*)\right] \leq \mathbf{E}\left[\left\langle \mathbf{g}^{(t)}, \mathbf{w}^{(t)} - \mathbf{w}^*\right\rangle\right] .$$

Beweis. Betrachten wir Schritt t und halten wir die Zufallsereignisse, die bis hier geschehen sind fest. Mathematisch formuliert, betrachten wir also den bedingten Wahrscheinlichkeitsraum für ein festes $\mathbf{w}^{(t)}$. Sei nun $\bar{\mathbf{g}} = \mathbf{E}\left[\mathbf{g}^{(t)} \mid \mathbf{w}^{(t)}\right]$. Gemäß unserer Annahme gilt $\bar{\mathbf{g}} \in \partial f(\mathbf{w}^{(t)})$. Das heißt insbesondere

$$f(\mathbf{w}^*) \geq f(\mathbf{w}^{(t)}) + \left\langle \bar{\mathbf{g}}, \mathbf{w}^* - \mathbf{w}^{(t)} \right\rangle$$

und somit

$$f(\mathbf{w}^{(t)}) - f(\mathbf{w}^*) \leq \left\langle \bar{\mathbf{g}}, \mathbf{w}^{(t)} - \mathbf{w}^* \right\rangle \ .$$

Nun ist $\bar{g}_i = \mathbf{E}\left[g_i^{(t)} \;\middle|\; \mathbf{w}^{(t)}\right]$, also gilt wegen Linearität des Erwartungswerts

$$\begin{aligned}
\left\langle \bar{\mathbf{g}}, \mathbf{w}^{(t)} - \mathbf{w}^* \right\rangle &= \sum_{i=1}^{n} \bar{g}_i (\mathbf{w}^{(t)} - \mathbf{w}^*)_i \\
&= \sum_{i=1}^{n} \mathbf{E}\left[g_i^{(t)} \;\middle|\; \mathbf{w}^{(t)}\right] (\mathbf{w}^{(t)} - \mathbf{w}^*)_i \\
&= \mathbf{E}\left[\sum_{i=1}^{n} g_i^{(t)} (\mathbf{w}^{(t)} - \mathbf{w}^*)_i \;\middle|\; \mathbf{w}^{(t)}\right] \\
&= \mathbf{E}\left[\left\langle \mathbf{g}^{(t)}, \mathbf{w}^{(t)} - \mathbf{w}^* \right\rangle \;\middle|\; \mathbf{w}^{(t)}\right] \ .
\end{aligned}$$

Damit gilt für jedes $\mathbf{w}^{(t)}$, egal wie wir es erreicht haben

$$f(\mathbf{w}^{(t)}) - f(\mathbf{w}^*) \leq \mathbf{E}\left[\left\langle \mathbf{g}^{(t)}, \mathbf{w}^{(t)} - \mathbf{w}^* \right\rangle \;\middle|\; \mathbf{w}^{(t)}\right] \ .$$

Um nun die Rechnung unkompliziert formal korrekt zu halten, nehmen wir an, dass $\mathbf{w}^{(t)}$ nur endlich viele Werte $\mathbf{v}_1, \ldots, \mathbf{v}_k$ und $\mathbf{g}^{(t)}$ nur endlich viele Werte $\mathbf{g}_1, \ldots, \mathbf{g}_\ell$ annehmen kann. Dann gilt für den unbedingten Erwartungswert

$$\begin{aligned}
\mathbf{E}\left[f(\mathbf{w}^{(t)}) - f(\mathbf{w}^*)\right] &= \sum_{i=1}^{k} \mathbf{Pr}\left[\mathbf{w}^{(t)} = \mathbf{v}_i\right] (f(\mathbf{v}_i) - f(\mathbf{w}^*)) \\
&\leq \sum_{i=1}^{k} \mathbf{Pr}\left[\mathbf{w}^{(t)} = \mathbf{v}_i\right] \mathbf{E}\left[\left\langle \mathbf{g}^{(t)}, \mathbf{w}^{(t)} - \mathbf{w}^* \right\rangle \;\middle|\; \mathbf{w}^{(t)} = \mathbf{v}_i\right] \\
&= \sum_{i=1}^{k} \mathbf{Pr}\left[\mathbf{w}^{(t)} = \mathbf{v}_j\right] \sum_{j=1}^{\ell} \mathbf{Pr}\left[\mathbf{g}^{(t)} = \mathbf{g}_j \;\middle|\; \mathbf{w}^{(t)} = \mathbf{v}_i\right] \langle \mathbf{g}_j, \mathbf{v}_i - \mathbf{w}^* \rangle \\
&= \sum_{i=1}^{k} \sum_{j=1}^{\ell} \mathbf{Pr}\left[\mathbf{w}^{(t)} = \mathbf{v}_j, \mathbf{g}^{(t)} = \mathbf{g}_j\right] \langle \mathbf{g}_j, \mathbf{v}_i - \mathbf{w}^* \rangle \\
&= \mathbf{E}\left[\left\langle \mathbf{g}^{(t)}, \mathbf{w}^{(t)} - \mathbf{w}^* \right\rangle\right] \ .
\end{aligned}$$

Diese Rechnung gilt auch allgemeiner. Dafür müssten wir allerdings den bedingten Erwartungswert formaler definieren, was über die Inhalte der Vorlesung hinausgeht. □

Nun können wir den Algorithmus im Wesentlichen wie Gradient Descent analysieren. Wir müssen lediglich des öfteren Gebrauch davon machen, dass der Erwartungswert linear ist.

Beweis von Satz 10.1. In der Analyse von Gradient Descent haben wir gezeigt, dass für all $\mathbf{u}, \mathbf{v}$ gilt

$$\langle \mathbf{u}, \mathbf{v} \rangle = \frac{1}{2} \left(\|\mathbf{u} + \mathbf{v}\|^2 - \|\mathbf{u}\|^2 - \|\mathbf{v}\|^2 \right) \ .$$

Diese Gleichung haben wir wie folgt genutzt, um $\left\langle \mathbf{g}^{(t)}, \mathbf{w}^{(t)} - \mathbf{w}^* \right\rangle$ umzuschreiben. Dabei ist es unerheblich, wie $\mathbf{g}^{(t)}$ definiert ist. Wir nutzen lediglich $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \mathbf{g}^{(t)}$.

$$\begin{aligned}
\left\langle \mathbf{g}^{(t)}, \mathbf{w}^{(t)} - \mathbf{w}^* \right\rangle &= -\frac{1}{\eta} \left\langle \mathbf{w}^{(t)} - \mathbf{w}^*, -\eta \mathbf{g}^{(t)} \right\rangle \\
&= -\frac{1}{2\eta} \left(\|\mathbf{w}^{(t+1)} - \mathbf{w}^*\|^2 - \|\mathbf{w}^{(t)} - \mathbf{w}^*\|^2 - \|-\eta \mathbf{g}^{(t)}\|^2 \right) \\
&= \frac{1}{2\eta} \left(\|\mathbf{w}^{(t)} - \mathbf{w}^*\|^2 - \|\mathbf{w}^{(t+1)} - \mathbf{w}^*\|^2 \right) + \frac{\eta}{2} \|\mathbf{g}^{(t)}\|^2 \ .
\end{aligned}$$

Ebenfalls erhalten wir über die Teleskopsumme und $\mathbf{w}^{(1)} = 0$ und $\|\mathbf{w}^{(T+1)} - \mathbf{w}^*\|^2 \geq 0$ wieder

$$\begin{aligned}
\sum_{t=1}^{T} \left\langle \mathbf{g}^{(t)}, \mathbf{w}^{(t)} - \mathbf{w}^* \right\rangle &= \frac{1}{2\eta} \sum_{t=1}^{T} \left(\|\mathbf{w}^{(t)} - \mathbf{w}^*\|^2 - \|\mathbf{w}^{(t+1)} - \mathbf{w}^*\|^2 \right) + \frac{\eta}{2} \|\mathbf{g}^{(t)}\|^2 \\
&\leq \frac{1}{2\eta} \|\mathbf{w}^*\|^2 + \frac{\eta}{2} \sum_{t=1}^{T} \|\mathbf{g}^{(t)}\|^2 \ .
\end{aligned}$$

Nun können wir diese Gleichung mit Lemma 10.2 kombinieren. Aufgrund der Linearität des Erwartungswertes erhalten wir

$$\begin{aligned}
\mathbf{E} \left[\sum_{t=1}^{T} \left(f(\mathbf{w}^{(t)}) - f(\mathbf{w}^*) \right) \right] &= \sum_{t=1}^{T} \mathbf{E} \left[\left(f(\mathbf{w}^{(t)}) - f(\mathbf{w}^*) \right) \right] \\
&\leq \sum_{t=1}^{T} \mathbf{E} \left[\left\langle \mathbf{g}^{(t)}, \mathbf{w}^{(t)} - \mathbf{w}^* \right\rangle \right] \\
&= \mathbf{E} \left[\sum_{t=1}^{T} \left\langle \mathbf{g}^{(t)}, \mathbf{w}^{(t)} - \mathbf{w}^* \right\rangle \right] \\
&\leq \mathbf{E} \left[\frac{1}{2\eta} \|\mathbf{w}^*\|^2 + \frac{\eta}{2} \sum_{t=1}^{T} \|\mathbf{g}^{(t)}\|^2 \right] \\
&= \frac{1}{2\eta} \|\mathbf{w}^*\|^2 + \frac{\eta}{2} \sum_{t=1}^{T} \mathbf{E} \left[\|\mathbf{g}^{(t)}\|^2 \right] \ .
\end{aligned}$$

Weil $\|\mathbf{w}^*\|^2 \leq B^2$ und $\mathbf{E} \left[\|\mathbf{g}^{(t)}\|^2 \right] \leq \rho^2$ gemäß Annahme, folgt der Satz. □

6 Norm des Subgradienten

Die Garantie in Satz 10.1 hängt von ρ ab, wobei wir fordern, dass $\|\mathbf{g}^{(t)}\| \leq \rho$ für alle t mit Wahrscheinlichkeit 1. Wie können wir diese Werte im Fall von Soft-SVM beschränken?

Betrachten wir Gleichung (2), können wir $\mathbf{g}^{(t)}$ schreiben als $\mathbf{g}^{(t)} = 2\lambda \mathbf{w}^{(t)} + \mathbf{v}^{(t)}$, wobei

$$\mathbf{v}^{(t)} = \begin{cases} -y_{I_t} \mathbf{x}_{I_t} & \text{falls } 1 - y_{I_t} \langle \mathbf{w}, \mathbf{x}_{I_t} \rangle > 0 \\ 0 & \text{sonst} \end{cases} \ .$$

Wir können also mittels der Dreiecksungleichung abschätzen

$$\|\mathbf{g}^{(t)}\| \leq 2\lambda\|\mathbf{w}^{(t)}\| + \|\mathbf{v}^{(t)}\| \leq 2\lambda\|\mathbf{w}^{(t)}\| + \max_i \|\mathbf{x}_i\| \ .$$

Entscheidend ist also, wie groß $\|\mathbf{w}^{(t)}\|$ werden kann. Dies ergibt sich aus dem bisherigen Verlauf des Algorithmus. Hierfür können wir $\mathbf{g}^{(t-1)}, \ldots, \mathbf{g}^{(1)}$ einsetzen und erhalten

$$\mathbf{w}^{(t)} = \mathbf{w}^{(t-1)} - \eta\left(2\lambda\mathbf{w}^{(t-1)} + \mathbf{v}^{(t-1)}\right) = (1-2\eta\lambda)\,\mathbf{w}^{(t-1)} - \eta\mathbf{v}^{(t-1)} = \ldots = \sum_{i=1}^{t-1}(1-2\eta\lambda)^{t-1-i}\,\eta\mathbf{v}^{(i)} \ .$$

Nun erhalten wir mittels Dreiecksungleichung und geometrischer Summenformel

$$\|\mathbf{w}^{(t)}\| \leq \sum_{i=1}^{t-1}(1-2\eta\lambda)^{t-1-i}\,\eta\|\mathbf{v}^{(i)}\| \leq \sum_{i=0}^{\infty}(1-2\eta\lambda)^{i}\,\eta\max_i\|\mathbf{x}_i\| = \frac{1}{2\eta\lambda}\eta\max_i\|\mathbf{x}_i\| = \frac{1}{2\lambda}\max_i\|\mathbf{x}_i\| \ .$$

In die obige Schranke auf $\|\mathbf{g}^{(t)}\|$ eingesetzt, bekommen wir also

$$\|\mathbf{g}^{(t)}\| \leq 2\max_i\|\mathbf{x}_i\| \ .$$

Referenzen

- Understanding Machine Learning, Kapitel 14.3 und 14.5

AGML, Sommersemester 2020 Vorlesung 11 (4 Seiten)

Kernel-Funktionen

Thomas Kesselheim Letzte Aktualisierung: 29. Mai 2020

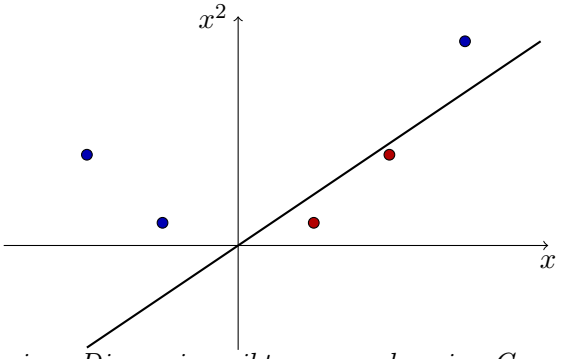
In vielen Fällen kann man mittels linearer Klassifikation keine genügend guten Vorhersagen treffen. Wir werden uns heute komplexere Klassifikatoren anschauen. Die zugrundeliegenden Optimierungsprobleme können wir allerdings auf lineare Klassifikation zurückführen.

Beispiel 11.1. *Uns seien folgende Trainingsdaten gegeben:*

x_i	y_i
-2	-1
-1	-1
1	$+1$
2	$+1$
3	-1

Hier ist lineare Klassifikation, also die Wahl einer Schwellenwertfunktion, offensichtlich keine sonderlich gute Idee. Es ist relativ offensichtlich, dass eigentlich ein Intervall gesucht wird. Interessant ist, dass ein Algorithmus dieses Intervall auch mittels linearer Klassikation finden kann, wenn wir als Merkmale $(x_i, x_i^2) \in \mathbb{R}^2$ ansehen.

(x_i, x_i^2)	y_i
$(-2, 4)$	-1
$(-1, 1)$	-1
$(1, 1)$	$+1$
$(2, 4)$	$+1$
$(3, 9)$	-1



Durch Hinzunahme einer Dimension gibt es nun also eine Gerade, die die Punkte separiert.

1 Einbettungen und Feature Space

Anstatt lineare Klassifikation über dem Merkmalsraum X betrachten wir diese nun über einem *Feature Space* F; zunächst ist $F = \mathbb{R}^n$, wobei $n \in \mathbb{N}$ unterschiedliche groß sein kann. Dazu ist uns eine Einbettung $\psi\colon X \to F$ gegeben.

Beispiel 11.2.

- *Im oben Beispiel ist $X = \mathbb{R}$, $F = \mathbb{R}^2$, $\psi(x) = (x, x^2)$.*
- *Eine Einbettung, über die wir schon implizit gesprochen haben, ist die folgende. Ist $X = \mathbb{R}^d$, können wir $F = \mathbb{R}^{d+1}$ und $\psi(\mathbf{x}) = (\mathbf{x}, 1)$ betrachten. Das heißt, wir fügen jedem $\mathbf{x}$-Vektor als letzte Komponente eine 1 an. Jetzt können wir uns auf lineare Klassifikation mittels Hyperebenen beschränken, die durch den Ursprung gehen.*
- *Allgemeiner können wir polynomielle Einbettungen betrachten. Sei dafür $X = \mathbb{R}^d$ und $k \in \mathbb{N}$ fest. Nun definieren wir $\psi(\mathbf{x})$ als den Vektor, dessen Komponenten alle möglichen Formen $\prod_{i=1}^d x_i^{j_i} = x_1^{j_1} \cdot x_2^{j_2} \cdot \ldots \cdot x_d^{j_d}$ mit $0 \leq j_i \leq k$ für alle i hat. Die Dimension von F ist $n = (k+1)^d$, kann also leicht sehr groß werden. Konkret können wir $d = 2$ und $k = 2$ anschauen, dann ist $\psi(x_1, x_2) = (1, x_1, x_1^2, x_2, x_1x_2, x_1^2x_2, x_2^2, x_1x_2^2, x_1^2x_2^2)$.*

- *Es könnte aber auch X die Menge aller E-Mails sein und F könnte ein Vektor irgendwelcher Eigenschaften sein, beispielsweise wie oft das gewisse Wörter vorkommen.*

Der Lernalgorithmus, der eine Einbettung ψ benutzt, könnte also wie folgt aussehen:

1. Berechne die Einbettung der Trainingsdaten. Sei die eingebettete Trainingsmenge $\hat{S}$ entsprechend definiert als $(\psi(\mathbf{x}_1), y_1), \dots, (\psi(\mathbf{x}_m), y_m)$.
2. Finde einen möglichst guten linearen Klassifikator $h_{\mathbf{w}} \colon F \to \{-1, +1\}$, mit Trainingsmenge $\hat{S}$.
3. Gib Hypothese $h \colon X \to \{-1, +1\}$ zurück mit

$$h(\mathbf{x}) = \begin{cases} +1 & \text{falls } \langle \mathbf{w}, \psi(x) \rangle \geq 0 \\ -1 & \text{sonst} \end{cases} .$$

Im zweiten Schritt könnten wir beispielsweise das Hard- oder das Soft-SVM-Problem auf F mit Trainingsmenge $\hat{S}$ lösen.

Je nachdem, wie ψ gewählt wird, also welche Features dem Algorithmus zur Verfügung stehen, werden die Ergebnisse besser oder schlechter. Deren Auswahl hängt von der Anwendung ab. Hier steckt ein bisschen die Kunst des Maschinellen Lernens.

2 Repräsentationssatz

Ob der Algorithmus, der die Einbettung nutzt, eine sinnvolle Laufzeit hat, hängt maßgeblich von der Dimension n des Feature Space ab. Diese kann jedoch sehr hoch sein, wie beispielsweise bei der oben genannten polynomiellen Einbettung. Wir werden nun einen Satz zeigen, mit dessen Hilfe sich die Laufzeit jedoch drastisch reduzieren lässt.

Dafür nehmen wir an, dass wir im zweiten Schritt einen Vektor $\mathbf{w} \in \mathbb{R}^n$ suchen, der eine Funktion $f \colon \mathbb{R}^n \to \mathbb{R}$ minimiert, die die Form

$$f(\mathbf{w}) = f_1(\|\mathbf{w}\|) + f_2(\langle \mathbf{w}, \psi(\mathbf{x}_1) \rangle, \dots, \langle \mathbf{w}, \psi(\mathbf{x}_m) \rangle) \tag{1}$$

hat, wobei $f_1 \colon \mathbb{R} \to \mathbb{R}$ monoton steigend und $f_2 \colon \mathbb{R}^m \to \mathbb{R}$ eine beliebige Funktion ist. Wichtig ist, dass beide Funktionen nur in einer sehr eingeschränkten Art von $\mathbf{w}$ abhängen. Die erste hängt lediglich von der Norm von $\mathbf{w}$ ab, die zweite lediglich von den Skalarprodukten von $\mathbf{w}$ mit $\mathbf{x}_1, \dots, \mathbf{x}_m$.

Alle Arten zur linearen Klassifikation, die wir bislang kennengelernt haben, lassen sich so darstellen.

- Bei Soft-SVM ist dies relativ offensichtlich. Hier könnten wir

$$f_1(a) = \lambda a^2, \qquad f_2(a_1, \dots, a_m) = \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y_i a_i\}$$

  wählen.

- Um Hard-SVM zu erfassen, nutzen wir

$$f_1(a) = a^2, \qquad f_2(a_1, \dots, a_m) = \begin{cases} 0 & \text{falls } y_i a_i \geq 1 \text{ für alle } i \\ \infty & \text{sonst} \end{cases} .$$

  Die Funktion f_2 bringt also in diesem Fall die Nebenbedingungen zum Ausdruck.

- Auch die Zielfunktion, die Anzahl falsch klassifizierter Punkte lässt sich in dieser Form schreiben. Hier ist $f_1(a) = 0$ für alle a und $f_2(a_1, \dots, a_m) = |\{i \mid y_i a_i \leq 0\}|$.

Satz 11.3. *Für jede Auswahl von Datenpunkten $\mathbf{x}_1, \dots, \mathbf{x}_m \in X$, Einbettungsfunktion $\psi\colon X \to F$, und jede Funktion f der Form wie in Gleichung (1) gibt es $\alpha_1, \dots, \alpha_m$, sodass der Vektor $\mathbf{w}' = \sum_{i=1}^m \alpha_i \psi(\mathbf{x}_i)$ die Funktion f minimiert.*

Das heißt, dass es um f zu minimieren ausreicht, nur die Linearkombinationen von $\psi(\mathbf{x}_1), \dots \psi(\mathbf{x}_m)$ zu betrachten.

Beweis von Satz 11.3. Sei $\mathbf{w}^* \in F$ eine optimale Lösung des Optimierungsproblems. Die Vektoren $\psi(\mathbf{x}_1), \dots, \psi(\mathbf{x}_m)$ erzeugen eine Unterraum U von F von Dimension höchstens m. Wir betrachten nun eine Orthonormalbasis $\mathbf{b}_1, \dots, \mathbf{b}_k$ dieses Unterraums U. (Diese könnte man beispielsweise mit dem Gram-Schmidtschen Orthogonalisierungsverfahren bestimmen.) Das heißt $\langle \mathbf{b}_j, \mathbf{b}_j \rangle = 1$ und $\langle \mathbf{b}_j, \mathbf{b}_{j'} \rangle = 0$ für $j \neq j'$. Außerdem lässt sich jedes $\psi(\mathbf{x}_i)$ als Linearkombination von $\mathbf{b}_1, \dots, \mathbf{b}_k$ darstellen. Weil es sich um eine Orthonomalbasis handelt, ist dies besonders einfach. Es gilt

$$\psi(\mathbf{x}_i) = \sum_{j=1}^{k} \langle \psi(\mathbf{x}_i), \mathbf{b}_j \rangle \mathbf{b}_j \ .$$

Nun betrachten wir die Projektion von $\mathbf{w}^*$ auf U. Diese berechnet sich in ähnlicher Weise als

$$\mathbf{w}' = \sum_{j=1}^{k} \langle \mathbf{w}^*, \mathbf{b}_j \rangle \mathbf{b}_j \ .$$

Es gilt $\mathbf{w}' \in U$, denn U umfasst ja genau alle Linearkombinationen von $\mathbf{b}_1, \dots, \mathbf{b}_k$. Wir können $\mathbf{w}'$ aber auch als Linearkombination von $\psi(\mathbf{x}_1), \dots, \psi(\mathbf{x}_m)$ schreiben, denn auch diese Vektoren erzeugen U. Das heißt, es gibt $\alpha_1, \dots, \alpha_m \in \mathbb{R}$ mit

$$\mathbf{w}' = \sum_{i=1}^{m} \alpha_i \psi(\mathbf{x}_i) \ .$$

Wir behaupten nun, dass $f(\mathbf{w}') \leq f(\mathbf{w}^*)$. Betrachten wir zunächst das Skalarprodukt von $\mathbf{w}'$ mit einem beliebigen $\mathbf{b}_{j'}$. Es gilt

$$\langle \mathbf{w}', \mathbf{b}_{j'} \rangle = \left\langle \sum_{j=1}^{k} \langle \mathbf{w}^*, \mathbf{b}_j \rangle \mathbf{b}_j, \mathbf{b}_{j'} \right\rangle = \sum_{j=1}^{k} \langle \mathbf{w}^*, \mathbf{b}_j \rangle \cdot \langle \mathbf{b}_j, \mathbf{b}_{j'} \rangle = \langle \mathbf{w}^*, \mathbf{b}_{j'} \rangle \ .$$

Somit gilt also auch

$$\langle \mathbf{w}', \psi(\mathbf{x}_i) \rangle = \left\langle \mathbf{w}', \sum_{j=1}^{k} \langle \psi(\mathbf{x}_i), \mathbf{b}_j \rangle \mathbf{b}_j \right\rangle = \sum_{j=1}^{k} \langle \psi(\mathbf{x}_i), \mathbf{b}_j \rangle \cdot \langle \mathbf{w}', \mathbf{b}_j \rangle = \sum_{j=1}^{k} \langle \psi(\mathbf{x}_i), \mathbf{b}_j \rangle \cdot \langle \mathbf{w}^*, \mathbf{b}_j \rangle = \langle \mathbf{w}^*, \psi(\mathbf{x}_i) \rangle \ .$$

Das heißt, dass $f_2(\langle \mathbf{w}', \psi(\mathbf{x}_1) \rangle, \dots, \langle \mathbf{w}', \psi(\mathbf{x}_m) \rangle) = f_2(\langle \mathbf{w}^*, \psi(\mathbf{x}_1) \rangle, \dots, \langle \mathbf{w}^*, \psi(\mathbf{x}_m) \rangle)$.

Eine analoge Rechnung liefert uns $\langle \mathbf{w}', \mathbf{w}' \rangle = \langle \mathbf{w}^*, \mathbf{w}' \rangle$. Definieren wir uns also $\mathbf{c} = \mathbf{w}^* - \mathbf{w}'$, stellen wir fest, dass $\langle \mathbf{w}', \mathbf{c} \rangle = \langle \mathbf{w}', \mathbf{w}^* \rangle - \langle \mathbf{w}', \mathbf{w}' \rangle = 0$. Somit gilt auch, dass

$$\|\mathbf{w}^*\|^2 = \langle \mathbf{w}' + \mathbf{c}, \mathbf{w}' + \mathbf{c} \rangle = \langle \mathbf{w}', \mathbf{w}' \rangle + \langle \mathbf{c}, \mathbf{c} \rangle = \|\mathbf{w}'\|^2 + \|\mathbf{c}\|^2 \ .$$

Dies bedeutet also auch, dass $\|\mathbf{w}'\| \leq \|\mathbf{w}^*\|$ und damit $f_1(\|\mathbf{w}'\|) \leq f_1(\|\mathbf{w}^*\|)$ aufgrund der Monotonie.

Insgesamt gilt also $f(\mathbf{w}') \leq f(\mathbf{w}^*)$. □

Aufgrund von Satz 11.3 können wir uns also darauf beschränken $\boldsymbol{\alpha} \in \mathbb{R}^m$ zu finden anstatt $\mathbf{w} \in \mathbb{R}^n$. Dies ist von enormem Nutzen, wenn $n \gg m$.

3 Effiziente Berechnung

Wie finden wir also einen Vektor $\boldsymbol{\alpha} \in \mathbb{R}^m$, so dass $f\left(\sum_{i=1}^m \alpha_i \psi(\mathbf{x}_i)\right)$ minimiert wird? Weiterhin hat f die Form aus Gleichung (1). Das heißt, f hängt nur von der Norm und den Skalarprodukten ab. Diese können wir auch direkt durch $\boldsymbol{\alpha}$ ausdrücken. Gilt nämlich $\mathbf{w} = \sum_{i=1}^m \alpha_i \psi(\mathbf{x}_i)$, dann auch

$$\langle \mathbf{w}, \psi(\mathbf{x}_j) \rangle = \left\langle \sum_{i=1}^m \alpha_i \psi(\mathbf{x}_i), \psi(\mathbf{x}_j) \right\rangle = \sum_{i=1}^m \alpha_i \langle \psi(\mathbf{x}_i), \psi(\mathbf{x}_j) \rangle$$

und

$$\|\mathbf{w}\| = \sqrt{\langle \mathbf{w}, \mathbf{w} \rangle} = \sqrt{\left\langle \sum_{i=1}^m \alpha_i \psi(\mathbf{x}_i), \sum_{j=1}^m \alpha_j \psi(\mathbf{x}_j) \right\rangle} = \sqrt{\sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j \langle \psi(\mathbf{x}_i), \psi(\mathbf{x}_j) \rangle} \ .$$

Definieren wir uns also eine neue Funktion $K \colon X \times X \to \mathbb{R}$ über $K(\mathbf{x}_i, \mathbf{x}_j) = \langle \psi(\mathbf{x}_i), \psi(\mathbf{x}_j) \rangle$, dann lassen sich diese Ausdrücke schreiben als

$$\langle \mathbf{w}, \psi(\mathbf{x}_j) \rangle = \sum_{i=1}^m \alpha_i K(\mathbf{x}_i, \mathbf{x}_j)$$

und

$$\|\mathbf{w}\| = \sqrt{\sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)} \ .$$

Somit gilt also

$$f\left(\sum_{i=1}^m \alpha_i \psi(\mathbf{x}_i)\right) = f_1\left(\sqrt{\sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)}\right) + f_2\left(\sum_{i=1}^m \alpha_i K(\mathbf{x}_i, \mathbf{x}_1), \ldots, \sum_{i=1}^m \alpha_i K(\mathbf{x}_i, \mathbf{x}_m)\right) \ .$$

Insgesamt müssten wir also, um f zu berechnen und auch zu minimieren, lediglich $K(\mathbf{x}_i, \mathbf{x}_j)$ für alle Paare i und j ausrechnen. Die einzelnen Werte von $\psi(\mathbf{x}_i)$ sind nicht gar nicht erforderlich. Das heißt, wir rechnen nicht einmal m^2 anstatt $m \cdot n$ Werte aus. Für große n kann dies ein enormer Vorteil sein.

Beispiel 11.4. *Betrachten wir wieder die polynomielle Einbettung des $X = \mathbb{R}^d$. Relativ einfaches Nachrechnen ergibt, dass $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \langle \mathbf{x}_i, \mathbf{x}_j \rangle)^k$. Das heißt, diese Werte lassen sich relativ leicht ausrechnen. Eine Bestimmung der m Vektoren $\psi(\mathbf{x}_1), \ldots, \psi(\mathbf{x}_m)$ mit je $(k+1)^d$ Einträgen ist nicht erforderlich.*

4 Kernels

Wie wir gesehen haben, ist es also nur nötig, die Funktion $K \colon X \times X \to \mathbb{R}$ auszurechnen. Eine solche Funktion nennt sich *Kernel*. Sie ersetzt gewissermaßen das Skalarprodukt auf X.

In der Tat ist es nicht einmal erforderlich, dass der Feature Space F eine endliche Dimension hat, denn die Funktion $\psi \colon X \to F$ muss nicht explizit ausgewertet werden. Der Raum F muss lediglich ein reeller Vektorraum sein, auf dem ein Skalarprodukt definiert ist, ein sogenannter *Hilbertraum*.

Referenzen

- Understanding Machine Learning, Kapitel 16

AGML, Sommersemester 2020 Vorlesung 12 (5 Seiten)

Overfitting

Thomas Kesselheim **Vorschau** Letzte Aktualisierung: 9. Juni 2020

In den letzten Vorlesungen haben wir vor allem diskutiert, wie wir eine Hypothese finden, die die Trainingsdaten möglichst gut beschreibt. Der Sinn einer solchen Hypothese ist es, Vorhersagen bei neuen Datenpunkten zu machen. Konkret also: Uns sind viele E-Mails gegeben, die jeweils als „Spam“ oder „kein Spam“ markiert sind. Auf dieser Basis wollen wir neu ankommende E-Mails möglichst gut klassifizieren.

1 Beispiel: Klassifikation

In Abbildung 1 sind Punkte in $\mathbb{R}$ mit binären Labels -1 und $+1$ gegeben, dargestellt als blaue und rote Punkte. Es wäre nun möglich, mittels durch Auswahl einiger Intervalle eine Hypothese zu finden, die keinen Trainingsfehler hat. Erstellt wurden die Daten jedoch wie folgt. Zunächst wurde $x_i \sim \text{Uniform}[0,1]$ gezogen wurde. Anschließend wurden die Labels bestimmt über

$$y_i = \begin{cases} +1 & \text{falls } x_i + \nu_i \geq \frac{1}{2} \\ -1 & \text{sonst} \end{cases},$$

wobei $\nu_i \sim \text{Uniform}[-0.3, 0.3]$ ein Rauschen mit Erwartungswert 0 ist. Das Rauschen lässt sich nicht vorhersagen. Entsprechend sollte einfach nur

$$h(x) = \begin{cases} +1 & \text{falls } x \geq \frac{1}{2} \\ -1 & \text{sonst} \end{cases}$$

als Hypothese verwendet werden. Diese hat zwar einen Trainingsfehler, ist aber die bestmögliche Vorhersage für neue Punkte.

Abbildung 1: Datenpunkte mit Rauschen.

2 Beispiel: Regression

Ein ähnliches Problem tritt auch bei *Regression* auf. Hier sind nun die Labels nicht mehr -1 oder $+1$ sondern beliebige reelle Zahlen.

Abbildung 2 zeigt ein Beispiel von acht Paaren von Datenpunkten mit ihren Labels (x_i, y_i), wobei $x_i \in [0,1]$ und $y_i \in \mathbb{R}$. Es wäre nun sehr verführerisch, eine Funktion h zu wählen, die die Werte in allen gegebenen Punkten genau trifft. Beispielsweise ein Polynom von Grad sieben. In diesem Fall ist es gegeben durch

$$h(x) = 5940.33x^7 - 20262.6x^6 + 27659.7x^5 - 19294.7x^4 + 7302.01x^3 - 1476.7x^2 + 148.067x - 5.53035 \ .$$

Dies entspricht dem roten Funktionsgraph in der Abbildung.

In diesem Fall wurden die Daten wie folgt generiert: Zunächst wurde $x_i \sim \text{Uniform}[0,1]$ gezogen. Anschließend wurde das Label für x_i bestimmt als $y_i = x_i + \nu_i$, wobei $\nu_i \sim \text{Normal}(0, 0.0025)$. Das heißt, ν_i ist ein zufälliges Rauschen aus einer Normalverteilung mit Erwartungswert 0 und Varianz 0.0025.

Auch in diesem Fall können wir das Rauschen nicht vorhersagen. Deshalb ist die beste Hypothese h in diesem Fall gegeben durch $h(x) = x$, eingetragen als die blaue Gerade.

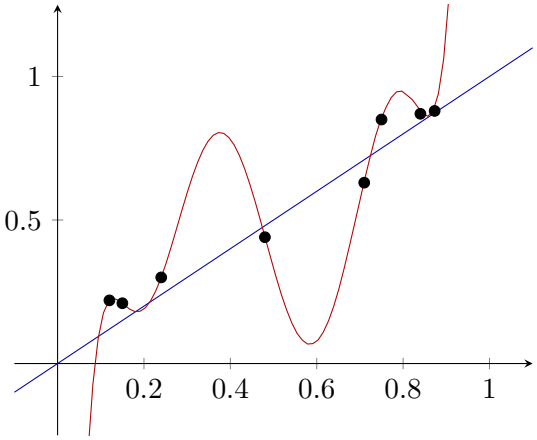


Abbildung 2: Die rote Kurve ist ein Polynom vom Grad sieben, das genau durch die acht gegebenen Punkte geht. Die blaue Gerade minimiert den tatsächlichen Fehler.

3 Problemstellung

Wir nehmen an, dass wir Datenpunkte aus einer Menge X erhalten und Labels für derartige Datenpunkte vorhersagen sollen. Diese Menge möglicher Labels heißt nun Y. Ein Fall ist binäre Klassifikation, also $Y = \{-1, +1\}$. Ein anderer Fall ist Regression mit $Y = \mathbb{R}$.

Wie im agnostischen Fall des PAC-Learning nehmen wir an, dass es eine Wahrscheinlichkeitsverteilung $\mathcal{D}$ über Paare $z = (x, y) \in X \times Y$ gibt, sodass y das korrekte Label ist für x. Uns ist eine Trainingsmenge $S = \{z_1, \ldots, z_m\}$, $z_i = (x_i, y_i) \in X \times Y$ aus m Samples gegeben, die aus $\mathcal{D}$ gezogen ist. Auf Basis von S berechnen wir eine Hypothese $h_S \colon X \to Y$, die ein Label $h_S(x)$ für jeden Punkt x vorhersagt.

Wir haben bereits Begriffe wie den Trainingsfehler und den tatsächlichen Fehler kennengelernt. Diese werden wir nun erweitern.

4 Loss-Funktionen und Fehlerbegriffe

Allgemein schreiben wir $\ell(h, z)$ für den Loss von Hypothese h auf $z = (x, y)$. Im Fall von binärer Klassifizierung ist die einfachste Wahl für ℓ der *0/1 Loss*, definiert durch

$$\ell^{0-1}(h, z) = \begin{cases} 0 & \text{falls } h(x) = y \\ 1 & \text{sonst} \end{cases} .$$

Wir haben bereits den *tatsächlichen Fehler* kennengelernt. Diesen verallgemeinern wir zum erwarteten Loss einer Hypothese h auf einem Datenpunkt-/Label-Paar gezogen aus $\mathcal{D}$, das heißt

$$L_{\mathcal{D}}(h) = \mathbf{E}_{z \sim \mathcal{D}} \left[\ell(h, z) \right] .$$

Auch können wir den Trainingsfehler verallgemeinern. Dieser ist für eine Menge S von m Datenpunkt-/Label-Paaren definiert als

$$L_S(h) = \frac{1}{m} \sum_{i=1}^{m} \ell(h, z_i) .$$

Anstatt des 0/1 Loss können wir auch andere Funktionen einsetzen. Wir haben bereits den *Hinge Loss* im Kontext von Support Vector Machines kennengelernt. Diesen hatten wir nur für lineare Klassifikatoren $h_{\mathbf{w}}$ definiert als

$$\ell^{\text{hinge}}(h_{\mathbf{w}}, \mathbf{z}) = \max\{0, 1 - y\langle \mathbf{w}, \mathbf{x}\rangle\} \ .$$

Der Vorteil des Hinge Loss ist, dass die Funktion stetig und konvex ist. Es gilt $\ell^{0-1}(h_{\mathbf{w}}, \mathbf{z}) \leq \ell^{\text{hinge}}(h_{\mathbf{w}}, \mathbf{z})$ für alle $\mathbf{w}$ und $\mathbf{z}$. Das heißt, Fehler werden im Vergleich zum 0/1 Loss im Normalfall überschätzt. Der Soft-SVM-Ansatz lässt sich auch so interpretieren, dass die Funktion ℓ^{0-1} durch eine stetige, konvexe Funktion ersetzt wird, die leichtere Optimierung ermöglicht.

Bei Regression versucht man im Normalfall, Fehlerquadrate zu minimieren. Die Loss-Funktion ist in diesem Fall

$$\ell^{\text{squared}}(h, z) = (h(x) - y)^2 \ .$$

5 Verallgemeinerungsfehler und Overfitting

Unser Ziel ist es, eine Hypothese zu finden, deren tatsächlicher Fehler $L_{\mathcal{D}}(h)$ möglichst klein ist. Dafür betrachten wir einen Lernalgorithmus, der eine Hypothese h_S auf Basis einer Trainingsmenge S berechnet. Diese Menge S besteht aus m Paaren $z_i = (x_i, y_i)$, die jeweils aus der Verteilung $\mathcal{D}$ gezogen werden.

Den tatsächlichen Fehler $L_{\mathcal{D}}(h_S)$ der berechneten Hypothese können wir uns nun wie folgt vorstellen: Einerseits ist h_S womöglich auf S schon nicht perfekt. Dies beschreibt der Trainingsfehler $L_S(h_S)$. Andererseits repräsentiert das Sample S die Verteilung $\mathcal{D}$ möglicherweise nicht perfekt. Deshalb bezeichnen wir nun

$$L_{\mathcal{D}}(h_S) - L_S(h_S)$$

als den *Verallgemeinerungsfehler*.

Als *Overfitting* versteht man nun das Phänomen, dass bei gewissen Lernalgorithmen der Trainingsfehler klein wird, der Verallgemeinerungsfehler aber groß. Insbesondere problematisch ist es, wenn größere Trainingsmengen über einen größeren Verallgemeinerungsfehler zu einem größeren tatsächlichen Fehler führen.

6 Stabilität von Lernalgorithmen

Wir wollen nun den *erwarteten* Verallgemeinerungsfehler eines Lernalgorithmus besser verstehen. Das heißt, uns interessiert

$$\mathbf{E}\left[L_{\mathcal{D}}(h_S) - L_S(h_S)\right] \ , \tag{1}$$

wobei der Erwartungswert über die Menge S geht. Dies wollen wir umschreiben.

Sei nun I eine Zufallsvariable, die unabhängig gleichverteilt aus $\{1, \ldots, m\}$ gezogen wir. Der erwartete Trainingsfehler ist nun

$$\mathbf{E}\left[L_S(h_S)\right] = \mathbf{E}\left[\frac{1}{m}\sum_{i=1}^{m} \ell(h_S, z_i)\right] = \mathbf{E}\left[\ell(h_S, z_I)\right] \ .$$

Der erwartete tatsächliche Fehler ist der erwartete Loss auf einem frisch gezogenen Datenpunkt-/Label-Paar z', das wiederum aus $\mathcal{D}$ gezogen wird

$$\mathbf{E}\left[L_{\mathcal{D}}(h_S)\right] = \mathbf{E}\left[\ell(h_S, z')\right] \ .$$

Auch dies können wir anders schreiben. Gegeben Samples $z_1, \dots, z_m$ und z', sei S^i die Menge $z_1, \dots, z_{i-1}, z', z_{i+1}, \dots, z_m$. Das heißt, wir ersetzen z_i durch z'. Da z_i und z' beide aus $\mathcal{D}$ gezogen werden, sind sie identisch verteilt und wir können ihre Rollen vertauschen. Deshalb gilt für alle i

$$\mathbf{E}\left[\ell(h_S, z')\right] = \mathbf{E}\left[\ell(h_{S^i}, z_i)\right] \ .$$

Weil diese Gleichung für alle i gilt, können wir auch die Zufallsvariable I von oben wieder verwenden. Damit gilt insgesamt

$$\mathbf{E}\left[L_{\mathcal{D}}(h_S)\right] = \mathbf{E}\left[\ell(h_{S^I}, z_I)\right] \ .$$

Und so kann der erwartete Verallgemeinerungsfehler aus (1) mittels Linearität des Erwartungswerts auch umgeschrieben werden zu

$$\begin{aligned}\mathbf{E}\left[L_{\mathcal{D}}(h_S) - L_S(h_S)\right] &= \mathbf{E}\left[L_{\mathcal{D}}(h_S)\right] - \mathbf{E}\left[L_S(h_S)\right] = \mathbf{E}\left[\ell(h_{S^I}, z_I)\right] - \mathbf{E}\left[\ell(h_S, z_I)\right] \\ &= \mathbf{E}\left[\ell(h_{S^I}, z_I) - \ell(h_S, z_I)\right] \ .\end{aligned}$$

Das heißt, der erwartete Verallgemeinerungsfehler kann nur groß sein, wenn es irgendwelche S^i und S gibt, die zu sehr unterschiedlichen Hypothesen führen. Dabei sollte man bedenken, dass S^i und S sich nur in einem einzigen Punkt unterscheiden. Ein Algorithmus, bei dem dies niemals geschieht, nennen wir stabil.

Definition 12.1. *Sei $\delta\colon \mathbb{N} \to \mathbb{R}$. Ein Lernalgorithmus ist* universell δ-austauschstabil*, wenn für alle $m \in \mathbb{N}$, alle Mengen S von m Datenpunkt-/Label-Paaren, alle $i \in \{1, \dots, m\}$ und alle weiteren Datenpunkt-/Label-Paare z' gilt*

$$\ell(h_{S^i}, z_i) - \ell(h_S, z_i) \leq \delta(m) \ .$$

Wir nennen ihn universell austauschstabil*, falls er universell δ-austauschstabil ist für eine Funktion δ mit $\delta(m) \to 0$ für $m \to \infty$.*

Wir sehen nun, dass wenn unser Lernalgorithmus universell δ-austauschstabil ist, dass

$$\mathbf{E}\left[L_{\mathcal{D}}(h_S) - L_S(h_S)\right] = \mathbf{E}\left[\ell(h_{S^I}, z_I) - \ell(h_S, z_I)\right] \leq \delta(m) \ .$$

Insbesondere, wenn $\delta(m) \to 0$ für $m \to \infty$, dann gibt es kein Overfitting.

Der große Vorteil davon, über Stabilität zu sprechen ist, dass es sich ausschließlich um eine Eigenschaft des Lernalgorithmus handelt. Wir müssen also keine Aussage über Wahrscheinlichkeitsverteilungen oder statistische Eigenschaften diskutieren, sondern lediglich Algorithmen entwickeln, deren Ausgabe sich nicht entscheidend ändert, wenn ein Datenpunkt ausgetauscht wird.

7 Beispiel

In unserem Einstiegsbeispiel haben wir anschaulich gesehen, dass es merkwürdige Effekte haben kann, Regression mittels einer Interpolation durch Polynome zu machen. Schon mit einem sehr einfach Beispiel können wir sehen, dass der Algorithmus, der den Trainingsfehler minimiert, nicht universell austauschstabil ist.

Der Merkmalsraum ist $X = \mathbb{R}$. Für unser Beispiel brauchen wir nur Polynome vom Grad 1, also Geraden bzw. Hypothesen der Form $h_{a,b}(x) = a \cdot x + b$ für $a, b \in \mathbb{R}$.

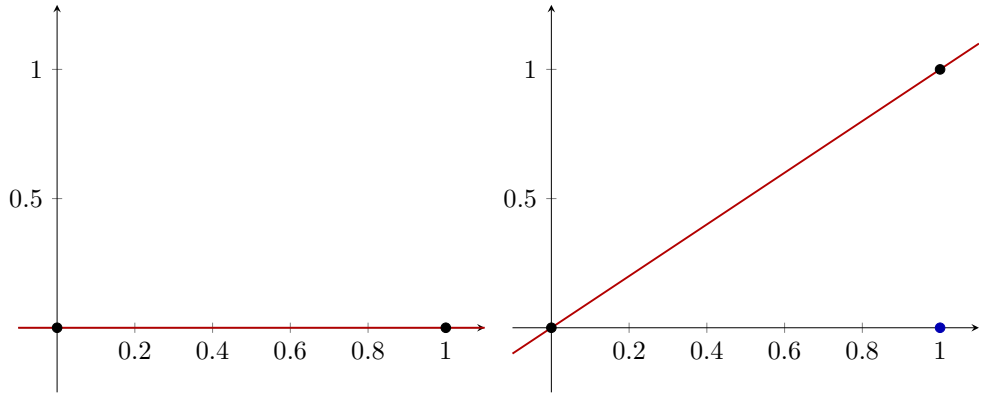


Abbildung 3: Die rote Gerade führt jeweils durch die schwarzen Punkte. Im rechten Bild ist ein Punkt ersetzt. Auf dem bisherigen Punkt (in blau) gibt es nun einen großen Fehler. Wie viele Punkte sich in $(0, 0)$ befinden, ist irrelevant.

Betrachten wir zunächst den Lernalgorithmus, der den Trainingsfehler minimiert. Das heißt $h_S = h_{a,b}$, wobei a und b so gewählt sind, dass $L_S^{\text{squared}}(h_{a,b}) = \frac{1}{m}\sum_{i=1}^{m}(ax_i + b - y_i)^2$ minimal ist. Dieser Algorithmus ist nicht universell austauschstabil. Seien dafür $(x_1, y_1) = (1, 0)$, $(x_2, y_2) = \ldots = (x_m, y_m) = (0, 0)$. Die Hypothese h, die den Trainingsfehler minimiert ist $h_{0,0}$. Betrachten wir $i = 1$, $(x', y') = (1, 1)$. Auf S^i wird der Trainingsfehler von $h_{1,0}$ minimiert (siehe Abbildung 3). Es gilt somit $\ell(h_{S^i}, z_i) - \ell(h_S, z_i) = 1$ und somit $\delta(m) \geq 1$. Auch bei Polynomen von höherem Grad tritt derselbe Effekt auf, denn weiterhin minimieren die Geraden den Trainingsfehler.

In der nächsten Vorlesung werden wir zeigen, dass dies nicht auftritt, wenn wir *Regularisierung* verwenden. In diesem konkreten Fall würden wir statt $L_S^{\text{squared}}(h_{a,b})$ nun $\lambda(a^2 + b^2) + L_S^{\text{squared}}(h_{a,b})$ minimieren, wobei λ ein Parameter ist. Der anschauliche Grund ist, dass der Einfluss eines Punktes (x_i, y_i) auf den Loss, also $\frac{1}{m}(ax_i + b - y_i)^2$, klein wird im Vergleich zu $\lambda(a^2 + b^2)$, sobald m groß wird. Wir erkaufen uns dies über einen höheren Trainingsfehler. Deshalb sollte λ nicht zu groß gewählt sein.

Referenzen

- Blog-Post von Moritz Hardt: `https://www.offconvex.org/2016/03/14/stability/`
- Understanding Machine Learning, Kapitel 13.2
- Foundations of Machine Learning, Kapitel 14.1–14.2 (etwas andere Aussage)

AGML, Sommersemester 2020 Vorlesung 13 (5 Seiten)

Regularisierung

Thomas Kesselheim Letzte Aktualisierung: 12. Juni 2020

In der letzten Vorlesung haben wir das Phänomen des Overfitting kennengelernt. Zu Erinnerung: Wir nehmen an, dass ein Lernalgorithmus eine Trainingsmenge von m Datenpunkt-/Label-Paare aus $X \times Y$ erhält und mithilfe von diesem Sample eine Hypothese $h_S \colon X \to Y$ finden soll, die Labels für Datenpunkte vorhersagen sollen. Beim Overfitting tritt es auf, dass die Hypothese „zu gut“ auf den Trainingsdaten ist und sich daher zu schlecht verallgemeinert. Eine gute Faustregel ist, dass man „einfachere“ Hypothesen verwenden sollte, um Overfitting zu vermeiden. Hierzu werden wir heute ein formales Argument führen.

Wir haben bereits die Definition eines stabilen Lernalgorithmus eingeführt.

Definition 13.1. *Sei $\delta \colon \mathbb{N} \to \mathbb{R}$. Ein Lernalgorithmus ist* universell δ-austauschstabil*, wenn für alle $m \in \mathbb{N}$, alle Mengen S von m Datenpunkt-/Label-Paaren, alle $i \in \{1, \dots, m\}$ und alle weiteren Datenpunkt-/Label-Paare z' gilt*

$$\ell(h_{S^i}, z_i) - \ell(h_S, z_i) \leq \delta(m) \ .$$

Hierbei ist $\ell(h, z)$ der *Loss* von Hypothese h auf $z \in X \times Y$. Dieser drückt aus, „wie falsch“ die Hypothese h auf z ist. Unsere Erkenntnis hinsichtlich Overfitting lässt sich knapp zusammenfassen als:

> Ein universell δ-austauschstabiler Lernalgorithmus mit $\delta(m) \to 0$ für $m \to \infty$ vermeidet Overfitting.

Heute werden mit *Regularisierung* einen grundsätzlichen Ansatz kennenlernen, der zu Stabilität führt. Anstatt eine Hypothese h_S zu wählen, sodass $L_S(h_S)$ minimiert wird, sollten „extreme“ Hypothesen vermieden werden.

1 Annahmen

Wir betrachten heute keine beliebigen Hypothesenklassen mehr, sondern treffen ein paar Annahmen. Zunächst einmal nehmen wir an, dass die Hypothesen in unsere Klasse $\mathcal{H}$ durch Vektoren $\mathbf{w} \in \mathbb{R}^n$ parametrisiert sind. Das heißt,

$$\mathcal{H} = \{h_{\mathbf{w}} \colon X \to Y \mid \mathbf{w} \in M\} \ ,$$

wobei $M \subseteq \mathbb{R}^n$ eine konvexe Menge ist. Ein typisches Beispiel sind lineare Klassifikatoren (hier ist $Y = \{-1, +1\}$)

$$h_{\mathbf{w}}(\mathbf{x}) = \begin{cases} +1 & \text{falls } \langle \mathbf{w}, \mathbf{x} \rangle \geq 0 \\ -1 & \text{sonst} \end{cases} \ .$$

Wie wir gesehen haben, können mittels Einbettungen in einen Feature Space auch andere Hypothesen so dargestellt werden.

Analog kann man lineare Regression darstellen (nun ist $Y = \mathbb{R}$) über

$$h_{\mathbf{w}}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle \ .$$

Für unsere Ergebnisse wird vollkommen unerheblich sein, wie die Hypothese $h_{\mathbf{w}}$ genau definiert ist. Wir nehmen lediglich an, dass die Loss-Funktionen konvex sind. Das heißt, dass $\mathbf{w} \mapsto \ell(h_{\mathbf{w}}, z)$ konvex ist für alle z.

Darüber hinaus nehmen wir an, dass sie ρ-Lipschitz sind. Das heißt, dass für alle $\mathbf{w}, \mathbf{w}' \in M$ und alle z

$$\ell(h_{\mathbf{w}}, z) - \ell(h_{\mathbf{w}'}, z) \leq \rho \|\mathbf{w} - \mathbf{w}'\| \ .$$

Beispiel 13.2. *Der 0/1 Loss ist nicht konvex. Entsprechend sind unsere heutigen Ergebnisse nicht anwendbar.*

Der Hinge Loss auf $z = (x, y)$ ist definiert als

$$\ell^{\text{hinge}}(h_{\mathbf{w}}, \mathbf{z}) = \max\{0, 1 - y\langle \mathbf{w}, \mathbf{x}\rangle\} \ .$$

Er ist $\|\mathbf{x}\|$-Lipschitz.

Der quadratische Loss (für Regression) ergibt sich zu

$$\ell^{\text{squared}}(h_{\mathbf{w}}, z) = (\langle \mathbf{w}, \mathbf{x}\rangle - y)^2 \ .$$

Er ist ρ-Lipschitz für $\rho = 2\|\mathbf{x}\|^2 \max_{\mathbf{w}\in M}\|\mathbf{w}\|$.

2 Starke Konvexität

Wir werden nun eine genauere Definition von Konvexität einführen, die zum Ausdruck bringt, wieviel deutlicher eine Funktion wächst als eine lineare Funktion. Dafür vergleichen wir sie mit einer quadratischen Funktion.

Definition 13.3. *Sei $\sigma \geq 0$. Eine Funktion $f\colon M \to \mathbb{R}$ heißt σ-stark konvex, wenn für alle $\mathbf{u}, \mathbf{v} \in M$ und alle $\lambda \in [0, 1]$ gilt*[1]

$$f(\lambda\mathbf{u} + (1-\lambda)\mathbf{v}) \leq \lambda f(\mathbf{u}) + (1-\lambda) f(\mathbf{v}) - \frac{\sigma}{2}\lambda(1-\lambda)\|\mathbf{u} - \mathbf{v}\|^2 \ .$$

Eine Funktion ist konvex genau dann, wenn sie 0-stark konvex ist.

Konvexität erfordert, dass die Funktion f jeweils unterhalb der Verbindungslinien auf dem Funktionsgraphen bleibt. Starke Konvexität mit $\sigma > 0$ fordert zusätzlich, dass sie unterhalb einer verbindenden Parabel bleibt. Das heißt, die Funktion muss „durchhängen“ (siehe Abbildung 1).

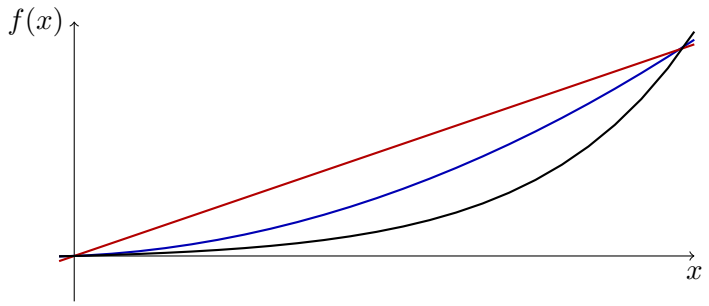


Abbildung 1: Eine stark konvexe Funktion in schwarz mit einer direkten Verbindunglinie zweier Punkt in rot und einer dazwischen liegenden Parabel in blau.

[1] Es mag etwas verwundern, dass der Faktor $\frac{\sigma}{2}$ ist und nicht σ. Auf diese Weise bleibt die Definition äquivalent mit anderen in der Literatur üblichen Formulierungen.

Beispiel 13.4. *Für jedes $\alpha \geq 0$, ist Funktion $f\colon \mathbb{R}^n \to \mathbb{R}$, $f(\mathbf{x}) = \alpha\|\mathbf{x}\|^2$ jeweils 2α-stark konvex.*

Für alle $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ und alle $\lambda \in [0,1]$ gilt

$$\begin{aligned}\|\lambda\mathbf{u} + (1-\lambda)\mathbf{v}\|^2 &= \sum_{i=1}^{n}(\lambda u_i + (1-\lambda)v_i)^2 = \sum_{i=1}^{n}(\lambda u_i)^2 + ((1-\lambda)v_i)^2 + 2\lambda u_i(1-\lambda)v_i \\ &= \lambda^2\|\mathbf{u}\|^2 + (1-\lambda)^2\|\mathbf{v}\|^2 + 2\lambda(1-\lambda)\langle\mathbf{u},\mathbf{v}\rangle \\ &= \lambda\|\mathbf{u}\|^2 - \lambda(1-\lambda)\|\mathbf{u}\|^2 + (1-\lambda)\|\mathbf{v}\|^2 - \lambda(1-\lambda)\|\mathbf{v}\|^2 + 2\lambda(1-\lambda)\langle\mathbf{u},\mathbf{v}\rangle \\ &= \lambda\|\mathbf{u}\|^2 + (1-\lambda)\|\mathbf{v}\|^2 - \lambda(1-\lambda)\|\mathbf{u}-\mathbf{v}\|^2 \ .\end{aligned}$$

Indem wir beide Seiten dieser Gleichung mit α multiplizieren, erhalten wir

$$f(\lambda\mathbf{u} + (1-\lambda)\mathbf{v}) = \lambda f(\mathbf{u}) + (1-\lambda)f(\mathbf{v}) - \frac{2\alpha}{2}\lambda(1-\lambda)\|\mathbf{u}-\mathbf{v}\|^2 \ .$$

Das heißt, die geforderte Ungleichung ist für $\sigma = 2\alpha$ sogar mit Gleichheit erfüllt.

Die Bedeutung von stark konvexen Funktionen zeigt sich im folgenden Lemma. Es sagt aus, dass wir in deutlicher Entfernung vom Minimum auch deutlich größere Funktionswerte beobachten.

Lemma 13.5. *Sei $f\colon M \to \mathbb{R}$ eine σ-stark konvexe Funktion. Sei $\mathbf{w} \in \arg\min_{\mathbf{v}\in M} f(\mathbf{v})$ ein Punkt, der f minimiert. Dann gilt für alle $\mathbf{u} \in M$*

$$f(\mathbf{u}) - f(\mathbf{w}) \geq \frac{\sigma}{2}\|\mathbf{u}-\mathbf{w}\|^2 \ .$$

Beweis. Wir betrachten die Verbindungslinie zwischen $\mathbf{u}$ und $\mathbf{w}$. Für alle $\lambda \in [0,1]$ haben wir gemäß starker Konvexität

$$f(\lambda\mathbf{u} + (1-\lambda)\mathbf{w}) \leq \lambda f(\mathbf{u}) + (1-\lambda)f(\mathbf{w}) - \frac{\sigma}{2}\lambda(1-\lambda)\|\mathbf{u}-\mathbf{w}\|^2 \ .$$

Gleichzeitig wird f durch $\mathbf{w}$ minimiert. Also

$$f(\lambda\mathbf{u} + (1-\lambda)\mathbf{w}) \geq f(\mathbf{w}) \ .$$

Somit gilt für alle $\lambda \in [0,1]$

$$\lambda f(\mathbf{u}) + (1-\lambda)f(\mathbf{w}) - \frac{\sigma}{2}\lambda(1-\lambda)\|\mathbf{u}-\mathbf{w}\|^2 \geq f(\mathbf{w}) \ .$$

Falls $\lambda > 0$ ist, ist dies äquivalent zu

$$f(\mathbf{u}) - f(\mathbf{w}) \geq \frac{\sigma}{2}(1-\lambda)\|\mathbf{u}-\mathbf{w}\|^2 \ .$$

Angenommen, es gilt nun

$$f(\mathbf{u}) - f(\mathbf{w}) < \frac{\sigma}{2}\|\mathbf{u}-\mathbf{w}\|^2 \ ,$$

dann müsste auch

$$f(\mathbf{u}) - f(\mathbf{w}) < c\frac{\sigma}{2}\|\mathbf{u}-\mathbf{w}\|^2$$

für irgendein $c < 1$ gelten. Dann könnten wir $\lambda = 1 - c$ wählen und würden einen Widerspruch erhalten. Also gilt das Lemma. □

Wir halten noch eine einfache Beobachtung fest, die sich durch Nachrechnen zeigen lässt.

Beobachtung 13.6. *Ist $f_1\colon M \to \mathbb{R}$ eine σ-stark konvexe Funktion, $f_2\colon M \to \mathbb{R}$ eine konvexe Funktion, dann ist $f_1 + f_2$ eine σ-stark konvexe Funktion.*

3 Stark konvexe Regularisierung führt zu Stabilität

Wir betrachten nun den Lernalgorithmus, der anstatt $\mathbf{w}$ zu finden, sodass $L_S(h_\mathbf{w})$ minimiert wird, eine *regularisierte* Zielfunktion $f(\mathbf{w}) = R(\mathbf{w}) + L_S(h_\mathbf{w})$ minimiert. Konkret ist in unserem Fall $R(\mathbf{w}) = \alpha\|\mathbf{w}\|^2$. Wie wir oben gesehen haben, ist R nun 2α-stark konvex und somit auch f.

Beispiel 13.7. *Für lineare Klassifikation mittels Hinge Loss ergibt sich genau das Soft-SVM-Problem*[2]*.*

Für Regression nennt sich die Vorgehensweise $\alpha\|\mathbf{w}\|^2 + L_S^{\text{squared}}(h_\mathbf{w})$ *zu minimieren* Ridge Regression.

Wir können nun zeigen, dass jeder Lernalgorithmus, der eine stark-konvexe Regularisierungsfunktion verwendet, stabil ist.

Satz 13.8. *Sind die Loss-Funktionen konvex und ρ-Lipschitz und ist die Regularisierungsfunktion σ-stark konvex, dann ist der Lernalgorithmus universell $\frac{2\rho^2}{m\sigma}$-austauschstabil.*

Es ist wichtig, dass $\delta(m) = \frac{2\rho^2}{m\sigma}$ gegen 0 konvergiert. Gemäß der Ergebnisse aus der letzten Vorlesung heißt das, dass der erwartete Verallgemeinerungsfehler verschwindet, wenn wir genügend Samples verwenden.

Beweis von Satz 13.8. Sei $\mathbf{w}^*$ der Vektor, der die Hypothese beschreibt, die der Lernalgorithmus auf S berechnet. Das heißt, $h_S = h_{\mathbf{w}^*}$. Analog sei $\mathbf{w}^i$ der entsprechende Vektor für die Lösung auf S^i.

Laut Definition minimiert $\mathbf{w}^*$ die Funktion $f(\mathbf{w}) := R(\mathbf{w}) + \frac{1}{m}\sum_{j=1}^m \ell(h_\mathbf{w}, z_j)$. Andererseits minimiert $\mathbf{w}^i$ die Funktion $f^i(\mathbf{w}) := R(\mathbf{w}) + \frac{1}{m}\sum_{j=1, j\neq i}^m \ell(h_\mathbf{w}, z_j) + \ell(h_\mathbf{w}, z')$.

Deshalb erhalten wir jeweils durch Anwendung von Lemma 13.5

$$f(\mathbf{w}^i) - f(\mathbf{w}^*) \geq \frac{\sigma}{2}\|\mathbf{w}^i - \mathbf{w}^*\|^2$$

und

$$f^i(\mathbf{w}^*) - f^i(\mathbf{w}^i) \geq \frac{\sigma}{2}\|\mathbf{w}^* - \mathbf{w}^i\|^2 \ .$$

In Kombination also

$$f(\mathbf{w}^i) - f(\mathbf{w}^*) + f^i(\mathbf{w}^*) - f^i(\mathbf{w}^i) \geq \sigma\|\mathbf{w}^i - \mathbf{w}^*\|^2$$

Wenn wir die Definitionen von f und f^i einsetzen, erhalten wir die äquivalente Ungleichung

$$\frac{1}{m}\ell(h_{\mathbf{w}^i}, z_i) - \frac{1}{m}\ell(h_{\mathbf{w}^i}, z') - \frac{1}{m}\ell(h_{\mathbf{w}^*}, z_i) + \frac{1}{m}\ell(h_{\mathbf{w}^*}, z') \geq \sigma\|\mathbf{w}^i - \mathbf{w}^*\|^2 \ .$$

Durch die Lipschitz-Bedingungen können wir abschätzen

$$\ell(h_{\mathbf{w}^i}, z_i) - \ell(h_{\mathbf{w}^*}, z_i) \leq \rho\|\mathbf{w}^i - \mathbf{w}^*\| \quad \text{und} \quad \ell(h_{\mathbf{w}^i}, z') - \ell(h_{\mathbf{w}^*}, z') \leq \rho\|\mathbf{w}^i - \mathbf{w}^*\| \ .$$

Also

$$2\rho\|\mathbf{w}^i - \mathbf{w}^*\| \geq m\sigma\|\mathbf{w}^i - \mathbf{w}^*\|^2 \ ,$$

[2] Ein technischer Unterschied ist, ob die (nun versteckte) Verschiebung der Hyperebene auch regularisiert wird oder nicht. Wir ignorieren dies.

und somit

$$\|\mathbf{w}^i - \mathbf{w}^*\| \leq \frac{2\rho}{m\sigma} \ .$$

Das heißt, es gilt auch

$$\ell(h_{S^i}, z_i) - \ell(h_S, z_i) = \ell(h_{\mathbf{w}^i}, z_i) - \ell(h_{\mathbf{w}^*}, z_i) \leq \rho\|\mathbf{w}^i - \mathbf{w}^*\| \leq \frac{2\rho^2}{m\sigma} \ .$$

□

4 Fazit

Wie wir gesehen haben, kann Regularisierung also Overfitting vermeiden. Anzumerken ist jedoch, dass die Regularisierung nicht zu stark gewählt werden darf. Anderenfalls wird der Trainingsfehler groß, es tritt also Underfitting ein.

Referenzen

- Understanding Machine Learning, Kapitel 13.3–13.4
- Foundations of Machine Learning, Kapitel 14.3 (weitergehend)

AGML, Sommersemester 2020 — Vorlesung 14 (5 Seiten)

Boosting

Thomas Kesselheim — Letzte Aktualisierung: 16. Juni 2020

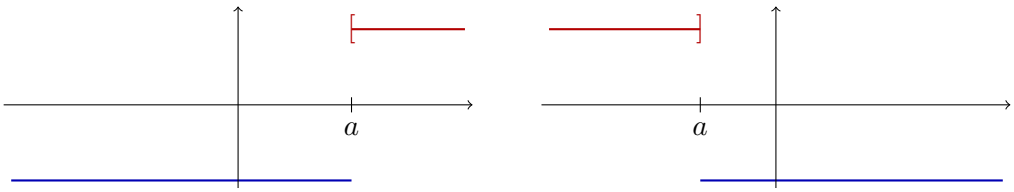
Beim maschinellen Lernen kommt es häufig vor, dass man in der Lage ist, Datenpunkte einigermaßen gut zu klassifizieren aber nicht genau genug. Ein Grund dafür kann sein, dass Hypothesen verwendet werden, die nicht ausdrucksstark genug sind, um Datenpunkte von einander zu unterscheiden. Heute werden wir eine sehr hilfreiche Technik kennenlernen, die die Genauigkeit auf der Trainingsmenge deutlich verbessert, das sogenannte *Boosting*.

1 Ein einführendes Beispiel

Schauen wir uns zunächst ein einfaches Beispiel an. Es mag sich zunächst etwas trivial anfühlen; es bedarf aber hoffentlich nicht zu viel Phantasie, um zu sehen, dass solche Probleme auch in komplexeren Szenarien auftreten.

Betrachten wir den Fall von binärer Klassifikation von Punkten aus den reellen Zahlen $\mathbb{R}$. Zu Beginn haben wir nur Schwellenwertfunktionen zur Verfügung. Dabei handelt es sich um Hypothesen der Form:

$$h(x) = \begin{cases} -1 & \text{falls } x < a \\ 1 & \text{falls } x \geq a \end{cases} \quad \text{oder} \quad h(x) = \begin{cases} -1 & \text{falls } x > a \\ 1 & \text{falls } x \leq a \end{cases} .$$



Wir können solche Hypothesen auch sehr knapp ausdrücken über zwei Parameter $w_1 \in \mathbb{R}$, $w_2 \in \{-1, 1\}$, sodass $h_{w_1,w_2}(x) = w_2 \cdot \text{sign}(x - w_1)$.[1]

Nehmen wir nun weiter an, dass die Grundwahrheit eigentlich ist, dass alle $x \in [a, b]$ positiv sind und alle $x \notin [a, b]$ negativ sind. Auch in diesem sehr einfachen Fall können wir im Allgemeinen keinen Trainingsfehler unter $\frac{1}{3}$ erreichen. Falls beispielsweise $S = \{(-1, -1), (0, +1), (1, -1)\}$ wird immer einer der Punkte inkorrekt klassifiziert werden.

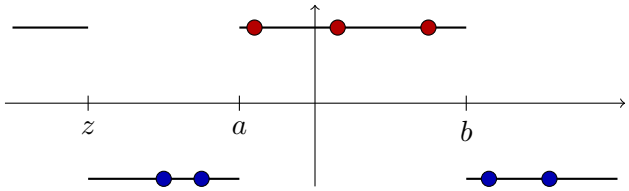
Hingegen wird eine *Linearkombination von Klassifikatoren* gute Ergebnisse liefern. Ist irgendeine Trainingsmenge $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ gegeben, definieren wir h^* mithilfe eines beliebigen $z < \min_i x_i$ über

$$h^*(x) = \text{sign}\left(h_{a,1}(x) + h_{b,-1}(x) + h_{z,-1}(x)\right) .$$

Nun ergibt sich für $x \in [a, b]$, dass $h_{a,1}(x) = h_{b,-1}(x) = 1$, $h_{z,-1}(x) = -1$. Also $h^*(x) = 1$. Für $x \in (z, a) \cup (b, \infty)$ ergibt sich $h^*(x) = -1$. Somit ist für alle i garantiert, dass $h^*(x_i) = y_i$ und damit $\text{err}_S(h^*) = 0$. Trotzdem handelt es sich nicht um die Grundwahrheit: Unterhalb von z ist die Klassifikation falsch.

Unser heutiges Ziel wird sein, diese Idee auf allgemeine Hypothesenklassen zu verallgemeinern.

[1] Wir nehmen an, dass die Signum-Funktion +1 an der Stelle 0 ist.



2 Problemstellung

Uns ist eine Trainingsmenge $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, $x_i \in X$, $y_i \in \{-1, +1\}$ für alle i, gegeben und wir möchten eine Hypothese h berechnen, sodass $\mathrm{err}_S(h) = \frac{1}{m}|\{i \mid h(x_i) \neq y_i\}|$ nah an 0 liegt.

Wir haben allerdings nur einen *schwachen Lernalgorithmus* zur Verfügung. Als Eingabe erhält er beliebige Gewichte $p_1, \ldots, p_m \geq 0$ mit $\sum_i p_i = 1$ (Die Gewichte definieren also eine Wahrscheinlichkeitsverteilung über S). Daraufhin berechnet er eine Hypothese h_p, sodass

$$\mathrm{err}_p(h_p) := \sum_{i:h_p(x_i)\neq y_i} p_i \leq \frac{1}{2} - \gamma$$

für ein festes $\gamma > 0$.[2] Für $p_i = \frac{1}{m}$ ist err_p genau der Trainingsfehler. Das heißt, die Hypothese h_p stellt sicher, dass eine gewichtete Version des Trainingsfehlers klein ist.

Beispiel 14.1. *Betrachten wir wieder das Eingangsbeispiel mit Schwellenwertfunktionen, aber die Grundwahrheit ist definiert über ein Intervall $[a, b]$. Wir werden nun zeigen, dass die Schwellenwertfunktion h, die $\mathrm{err}_p(h)$ minimiert, die obige Garantie mit $\gamma = \frac{1}{6}$ erfüllt. Das heißt, ein Algorithmus, der den gewichteten Trainingsfehler auf Schwellenwertfunktionen minimiert ist ein schwacher Lernalgorithmus, wenn die Grundwahrheit durch ein Intervall definiert ist.*

Um $\gamma = \frac{1}{6}$ zu zeigen, stellen wir fest, dass für jeden Vektor p und alle $a < b$ gilt

$$\sum_{i:x_i<a} p_i \leq \frac{1}{3} \quad \textit{oder} \quad \sum_{i:a\leq x_i\leq b} p_i \leq \frac{1}{3} \quad \textit{oder} \quad \sum_{i:x_i>b} p_i \leq \frac{1}{3} \ .$$

Zwei der drei Arten von Datenpunkten (unter a, zwischen a und b, über b) können wir leicht durch eine Schwellenwertfunktion richtig klassifizieren. Der kleinste Fehler wird also höchstens $\frac{1}{3} = \frac{1}{2} - \frac{1}{6}$ sein.

Wir werden zeigen, dass ein solcher schwacher Lernalgorithmus ausreicht, um einen *starken Lernalgorithmus* zu entwerfen: Gegeben ein $\epsilon > 0$ und eine Trainingsmenge S wird er den schwachen Lernalgorithmus nutzen, um Hypothesen $h_1, \ldots, h_T$ und $\alpha_1, \ldots, \alpha_T$ zu berechnen, sodass h^* mit $h^*(x) = \mathrm{sign}(\alpha_1 h_1(x) + \ldots + \alpha_T h_T(x))$ die Bedingungen $\mathrm{err}_S(h^*) \leq \epsilon$ erfüllt.

3 Idee für einen Algorithmus

Wir nehmen nun an, dass uns ein schwacher Lernalgorithmus gegeben ist, und wollen auf dieser Basis einen starken Lernalgorithmus entwickeln. Die Eingabe für diesen ist eine Trainingsmenge $S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$.

Unser Algorithmus ruft also den schwachen Lernalgorithmus wiederholt auf der Trainingsmenge mit unterschiedlicher Gewichtung auf. Anfänglich setzen wir $p_i^{(1)} = \frac{1}{m}$ für alle i. Das

[2] Das Boosting-Framework kann erweitert werden um die Annahme, dass diese Schranke nur mit gewisser Wahrscheinlichkeit eingehalten wird.

heißt, der schwache Lernalgorithmus minimiert den Trainingsfehler so gut es ihm möglich ist. Wir erhalten eine Hypothese h_1. Wir wissen nun, dass h_1 falsch liegt auf höchstens $(\frac{1}{2} - \gamma)m$ Punkten in S. Anders formuliert ist h_1 auf etwas mehr als der Hälfte der Punkte in S korrekt.

Nun rufen wir den schwachen Lernalgorithmus erneut auf. Die Idee ist nun $p^{(2)}$ so zu setzen, dass $p_i^{(2)} > \frac{1}{m}$, falls $h_1(x_i) \neq y_i$, und $p_i^{(2)} < \frac{1}{m}$, falls $h_1(x_i) = y_i$. Das heißt, in dieser Ausführung soll der schwache Lernalgorithmus sich mehr auf diejenigen Punkte konzentrieren, die bislang falsch klassifiziert wurden.

Wir müssen nun angeben, wie sich die Gewichte $p_i^{(t)}$ ergeben. Wir nutzen dazu Gewichte $w_i^{(t)}$, die sich nicht zwangsläufig zu 1 aufsummieren. Es ist immer $p_i^{(t)} = w_i^{(t)}/W^{(t)}$, wobei $W^{(t)}$ die Summe aller $w_i^{(t)}$ ist.

Das Gewicht $w_i^{(t)}$ drückt aus, wie oft Datenpunkt i von den Hypothesen $h_1, \ldots, h_{t-1}$ falsch klassifiziert worden ist. Je größer es ist, desto mehr dieser Hypothesen lagen falsch. Entsprechend wichtiger ist es, dass der schwache Lernalgorithmus diesen Punkt richtig klassifiziert, wenn er h_t berechnet.

Konkret ist $w_i^{(1)} = 1$ für alle i. In Schritt t werden die Gewichte über die Regel

$$w_i^{(t+1)} = \begin{cases} \mathrm{e}^{-\eta_t} w_i^{(t)} & \text{falls } h_t(x_i) = y_i \\ \mathrm{e}^{\eta_t} w_i^{(t)} & \text{sonst} \end{cases}$$

angepasst. Dabei ist $\eta_t > 0$ für alle Punkte gleich und abhängig vom aktuellen Fehler. Für die spätere Rechnung können wir dies etwas knapper schreiben. Wegen

$$y_i h_t(x_i) = \begin{cases} +1 & \text{falls } h_t(x_i) = y_i \\ -1 & \text{sonst} \end{cases}$$

ergibt sich

$$w_i^{(t+1)} = w_i^{(t)} \mathrm{e}^{-\eta_t y_i h_t(x_i)} \quad .$$

Diese Art, Gewichte mittels multiplikativer Veränderung anzupassen, ist beim Entwurf von Algorithmen relativ verbreitet. Sie begegnet uns auch bei Algorithmen für ganz andere Probleme. Die Zusammenhänge können wir an dieser Stelle leider nicht diskutieren.

4 AdaBoost

Der Algorithmus *AdaBoost* (für Adaptive Boosting) benutzt genau diese Ideen. Er lautet wie folgt.

- Initialisiere $w_i^{(1)} = 1$ für alle i
- In Schritt $t = 1, \ldots, T$
  - Berechne $W^{(t)} = \sum_{i=1}^m w_i^{(t)}$, $p_i^{(t)} = w_i^{(t)}/W^{(t)}$
  - Sei h_t das vom schwachen Lernalgorithmus auf $p^{(t)}$ berechnete Ergebnis
  - Berechne $\epsilon_t = \sum_{i:h_t(x_i)\neq y_i} p_i^{(t)}$ (Fehler von h_t auf $p^{(t)}$)
  - Sei $\eta_t = \frac{1}{2} \ln\left(\frac{1}{\epsilon_t} - 1\right)$
  - Aktualisiere Gewichte $w_i^{(t+1)} = w_i^{(t)} \mathrm{e}^{-\eta_t y_i h_t(x_i)}$
- Gib h^* aus, definiert über $h^*(x) = \mathrm{sign}\left(\sum_{t=1}^T \eta_t h_t(x)\right)$

Satz 14.2. *Der Algorithmus AdaBoost garantiert* $\mathrm{err}_S(h^*) \leq \exp(-2\gamma^2 T)$.

Zur Erinnerung: γ stammt aus der Garantie des schwachen Lernalgorithmus. Interessanterweise braucht der Algorithmus es nicht zu kennen.

Beweis. Sei $g_t(x) = \sum_{t'=1}^{t} \eta_{t'} h_{t'}(x)$. Durch diese Definition sind $w_i^{(t)} = \mathrm{e}^{-y_i g_{t-1}(x_i)}$ and $h^*(x) = \mathrm{sign}(g_T(x))$.

Wir betrachten, wie sich die Summe der Gewichte, $W^{(t)} = \sum_{i=1}^{m} w_i^{(t)}$, über die Zeit verändert. Wir werden zeigen, dass gilt

$$W^{(t+1)} \leq \mathrm{e}^{-2\gamma^2} W^{(t)} \quad \text{für alle } t \in \{1, \ldots, T\} \ . \tag{1}$$

Betrachten wir zunächst, wie diese aus dieser Ungleichung die Aussage des Satzes folgt. Der Algorithmus läuft für T Schritte. Aufgrund von Ungleichung (1) gilt im Anschluss $W^{(T+1)} \leq \mathrm{e}^{-2\gamma^2 T} W^{(1)} = \mathrm{e}^{-2\gamma^2 T} m$.

Darüber hinaus gilt für alle i mit $h^*(x_i) \neq y_i$, dass $y_i g_T(x_i) \leq 0$, denn das Produkt zweier reeller Zahlen mit unterschiedlichem Vorzeichen ist immer nicht-positiv. Das bedeutet, dass für diese i auch $w_i^{(T+1)} = \mathrm{e}^{-y_i g_T(x_i)} \geq 1$. Für alle andere i nutzen wir, dass $w_i^{(T+1)} \geq 0$. So erhalten wir insgesamt $W^{(T+1)} \geq |\{i \mid h^*(x_i) \neq y_i\}|$ und damit

$$\mathrm{err}_S(h^*) \leq \frac{1}{m} W^{(T+1)} \leq \mathrm{e}^{-2\gamma^2 T} \ .$$

Die Aussage des Satzes ist damit bewiesen.

Damit müssen wir also nur noch Ungleichung (1) zeigen. Die Summe der Gewichte nach Schritt t ist

$$W^{(t+1)} = \sum_{i=1}^{m} w_i^{(t+1)} = \sum_{i=1}^{m} w_i^{(t)} \mathrm{e}^{-y_i \eta_t h_t(x_i)} \ .$$

Das heißt, die Änderung ist

$$\frac{W^{(t+1)}}{W^{(t)}} = \sum_{i=1}^{m} \frac{w_i^{(t)}}{W^{(t)}} \mathrm{e}^{-y_i \eta_t h_t(x_i)} = \sum_{i=1}^{m} p_i^{(t)} \mathrm{e}^{-y_i \eta_t h_t(x_i)} = \sum_{i: h_t(x_i) = y_i} p_i^{(t)} \mathrm{e}^{-\eta_t} + \sum_{i: h_t(x_i) \neq y_i} p_i^{(t)} \mathrm{e}^{\eta_t} \ .$$

Gemäß Definition $\sum_{i: h_t(x_i) \neq y_i} p_i^{(t)} = \epsilon_t$ und $\mathrm{e}^{\eta_t} = \sqrt{1/\epsilon_t - 1}$. Also

$$\begin{aligned} \frac{W^{(t+1)}}{W^{(t)}} &= (1-\epsilon_t)\mathrm{e}^{-\eta_t} + \epsilon_t \mathrm{e}^{\eta_t} = (1-\epsilon_t)\frac{1}{\sqrt{1/\epsilon_t - 1}} + \epsilon_t \sqrt{1/\epsilon_t - 1} \\ &= (1-\epsilon_t)\sqrt{\frac{\epsilon_t}{1-\epsilon_t}} + \epsilon_t \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} = 2\sqrt{\epsilon_t(1-\epsilon_t)} \ . \end{aligned}$$

Der schwache Lernalgorithmus garantiert uns, dass $\epsilon_t \leq \frac{1}{2} - \gamma$ und somit

$$\frac{W^{(t+1)}}{W^{(t)}} = 2\sqrt{\epsilon_t(1-\epsilon_t)} \leq 2\sqrt{\left(\frac{1}{2} - \gamma\right)\left(\frac{1}{2} + \gamma\right)} = \sqrt{1 - 4\gamma^2} \leq \sqrt{\mathrm{e}^{-4\gamma^2}} = \mathrm{e}^{-2\gamma^2} \ .$$

Dies zeigt Ungleichung (1) und damit ist die Aussage bewiesen. □

5 Die Schattenseiten

Wir haben nun hergeleitet, dass wir auf jeder Trainingsmenge S einen Trainingsfehler von höchstens $\exp(-2\gamma^2 T)$ erhalten werden, wenn wir T Iteration von AdaBoost verwenden. Es liegt nun nahe, T so groß wie möglich zu wählen, weil dadurch der Fehler kleiner und kleiner wird. Dieser kleinere Fehler kann jedoch durch Overfitting zustande kommen.

Formaler gesprochen: Die VC-Dimension der Klasse von Hypothesen, die AdaBoost in T Iterationen berechnen kann, wächst in T. Betrachten wir dazu $X = \mathbb{R}$ und ein beliebiges $m \in \mathbb{N}$. Wenn T im Verhältnis groß genug ist, dann kann Boosting von Schwellenwertfunktionen m Punkte mit beliebigen Labels versehen. Das heißt, die VC-Dimension ist mindestens m in diesem Fall. Hierzu stellen wir fest, dass $\gamma = \frac{1}{2m}$ gilt, wenn wir das Label eines Punktes richtig setzen und bei mindestens der Hälfte der übrigen Punkte. Mit $T \geq \frac{1}{2\gamma^2} \ln(2m)$ erhalten wir $\mathrm{err}_S(h^*) \leq \frac{1}{2m}$. Also muss $\mathrm{err}_S(h^*) = 0$ sein.

Insgesamt heißt dies, dass man vorsichtig sein muss, wenn man Boosting anwendet: Es ist ein hilfreiches Werkzeug, um besser klassifizieren zu können, aber es gibt den bekannte Abwägung zwischen Trainingsfehler und Overfitting.

Referenzen

- Understanding Machine Learning, Kapitel 10
- Foundations of Machine Learning, Kapitel 7
- Freund, Yoav; Schapire, Robert E (1997). A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences. 55: 119. (Original AdaBoost-Paper)

AGML, Sommersemester 2020 Vorlesung 15 (5 Seiten)

Komposition

Anne Driemel Letzte Aktualisierung: 18. Juni 2020

Wir haben in der letzten Vorlesung das Boosting kennengelernt, welches schwache Lernalgorithmen miteinander kombiniert um einen starken Lernalgorithmus zu erhalten. Beim Boosting ergibt sich eine neue Hypothesenklasse aus den möglichen Linearkombinationen der Hypothesenklassen der benutzten schwachen Lernalgorithmen. Allerdings erzeugt das Boosting auch eine höhere VC-Dimension und somit die Gefahr, dass Overfitting geschieht. Heute werden wir genauer analysieren, wie sich die Komposition mehrerer Hypothesen auf die VC-Dimension der resultieren Hypothesenklasse auswirkt. Wir betrachten neben dem Boosting auch andere Arten der Komposition.

1 Achsenparallele Hyperquader

Wir schauen uns zunächst die Klasse der Schwellenwertfunktionen in $\mathbb{R}^d$ an und zeigen eine obere Schranke für die VC-Dimension. Schwellwertfunktionen können kombiniert werden, um Hyperquader darzustellen. Dies wird uns als einleitendes Beispiel dienen, bevor wir auf komplexere Kompositionen von Hypothesenklassen eingehen.

Sei die Klasse der Schwellenwertfunktionen in $\mathbb{R}^d$ definiert als Menge von Funktionen der Form $h_{i,a,b} : \mathbb{R}^d \to \{+1, -1\}$ mit $1 \leq i \leq d$, $a \in \mathbb{R}$, $b \in \{+1, -1\}$ und

$$h_{i,a,b}(x_1, \ldots, x_d) = \begin{cases} +b & \text{falls } x_i \geq a \\ -b & \text{sonst} \end{cases}$$

Eine Schwellenwertfunktion $h_{i,a,b}$ entspricht der Partitionierung der Grundmenge durch eine achsenparallelen Hyperebene. Wir definieren die Klasse der Hyperquader in $\mathbb{R}^d$ als Menge von Funktionen $h_{\mathbf{a},\mathbf{b}} : \mathbb{R}^d \to \{+1, -1\}$ definiert durch Vektoren $\mathbf{a} = (a_1, \ldots a_d) \in \mathbb{R}^d$ und $\mathbf{b} = (b_1, \ldots b_d) \in \mathbb{R}^d$ mit $a_i < b_i$ für alle $1 \leq i \leq d$ und

$$h_{\mathbf{a},\mathbf{b}}(x_1, \ldots, x_d) = \begin{cases} +1 & \text{falls } \forall i : a_i \leq x_i \leq b_i \\ -1 & \text{sonst} \end{cases}$$

Es ist leicht zu sehen, dass jeder Hyperquader durch eine Komposition von $2d$ Schwellenwertfunktionen darstellbar ist. Wie können wir nun leicht obere Schranken für die VC-Dimension von Hyperquadern zeigen? Wir analysieren zunächst die VC-Dimension der Schwellenwertfunktion.

Lemma 15.1. *Sei $\mathcal{H}$ die Klasse der Schwellenwertfunktionen mit Grundmenge $\mathbb{R}^d$. $\mathcal{H}$ hat VC-Dimension höchstens* $\max(2 \log_2 d, 8)$.

Beweis. Sei $\mathcal{R}$ das zu $\mathcal{H}$ zugehörige Mengensystem und sei $A \subseteq \mathbb{R}^d$ eine Menge, die von $\mathcal{R}$ aufgespalten wird. Zur Erinnering, das heißt dass für jedes $A' \subseteq A$ eine Menge $r \in \mathcal{R}$ existiert, sodass $A' = r \cap A$. Ziel ist es eine obere Schranke für $|A|$ zu zeigen, denn die VC-Dimension ist definiert als die Kardinalität der größten aufgespaltenen Menge. Dafür sei $t = |A|$.

Wir interessieren uns also für die Anzahl der verschiedenen Mengen $r \cap A$ mit $r \in \mathcal{R}$, also die Größe der Menge $\mathcal{R}|_A$. Gleichzeitig wissen wir, dass es genau 2^t verschiedenen Teilmengen von A gibt, die damit dargestellt werden. Es muss also gelten

$$2^t \leq \left|\mathcal{R}|_A\right|$$

Daraus wollen wir eine obere Schranke für t ableiten.

Die wichtige Beobachtung ist nun, dass es höchstens $2dt$ verschiedene nicht-leere Teilmengen von A gibt, die durch eine achsenparallele Hyperebene abgespalten werden können, da A in jeder Dimension höchstens t verschiedene Koordinaten hat. Das heißt

$$\left|\mathcal{R}|_A\right| \leq dt.$$

Also ist $2^t \leq 2dt$. Nun machen wir eine Fallunterscheidung. Angenommen, dass $t \leq d$. Dann ist $2^t \leq 2d^2$. Durch Logarithmieren auf beiden Seiten ergibt sich $t \leq 2\log_2 2d$. Der zweite Fall ist, dass $t > d$. Daraus ergibt sich analog $t < 2\log_2 2t$. Diese Ungleichung kann für $t \in \mathbb{N}$ nur erfüllt werden wenn $t \leq 8$.

Wir haben also hergeleitet, dass

$$t \leq \max(2\log_2 2d, 8)$$

Da dies für beliebige Mengen A gilt, die durch $\mathcal{R}$ aufgespalten werden, folgt die obere Schranke für die VC-Dimension nun direkt. □

2 Komposition

Definition 15.2 (Komposition)**.** *Sei X eine feste Grundmenge und sei C eine Klasse von Funktionen der Form $f : \{+1, -1\}^k \to \{+1, -1\}$. Sei $\mathcal{H}$ eine Hypothesenklassen mit Grundmenge X und sei $\mathcal{R}$ das zugehörige Mengensystem. Sei $\mathcal{H}_C$ die Hypothesenklasse aller Funktionen $g : X \to \{+1, -1\}$ mit*

$$g(x) = f(h_1(x), \ldots, h_k(x)) \quad \text{und} \quad h_1, \ldots, h_k \in \mathcal{H}, f \in C$$

Wir bezeichnen mit $\mathcal{R}_C$ das zugehörige Mengensystem.

Beispiel 15.3. *Im Fall von Boosting, ist die Klasse C die Menge aller Funktionen der Form $f(y_1, \ldots, y_k) = \operatorname{sign}(\sum_{1 \leq i \leq k} \alpha_i y_i)$ mit $\alpha_i \geq 0$. Der Fakt, dass dies einer Komposition nach Definition 15.2 entspricht, ist dabei unabhängig davon, wie die Gewichte α_i gewählt werden.*

Wir betrachten zunächst den Fall, dass die Klasse C nur aus einer festen Funktion besteht, zum Beispiel der Funktion die in dem zugehörigen Mengensystem die Schnittmenge der positiven Mengen erzeugt:

$$f(y_1, \ldots, y_k) = \begin{cases} +1 & \text{falls } \sum_{i=1}^k y_i = k \\ -1 & \text{sonst} \end{cases} \tag{1}$$

Wir bezeichnen die Komposition in dem Fall einer festen Funktion f mit $\mathcal{H}_f$, beziehungsweise das Mengensystem mit $\mathcal{R}_f$.

Beispiel 15.4. *Sei $\mathcal{H}$ die Klasse der Schwellenwertfunktionen und sei f definiert wie in (1) mit $k = 2d$. Dann ist $\mathcal{R}_f$ die Menge aller Hyperquader in $\mathbb{R}^d$. Das heißt, die Menge enthält alle beschränkten Hyperquader und zusätzlich solche, die in mindestens einer Richtung unbeschränkt sind.*

Beispiel 15.5. *Sei $\mathcal{R}$ das Mengensystem aller Halbräume in $\mathbb{R}^2$ und sei f definiert wie in (1) mit $k = 3$. Dann ist $\mathcal{R}_f$ die Menge aller verallgemeinerten Dreiecke in $\mathbb{R}^2$. Das heißt, die Menge enthält alle beschränkten Dreiecke und zusätzlich solche Dreiecke, die in einer Richtung unbeschränkt sind, siehe Abbildung 1.*

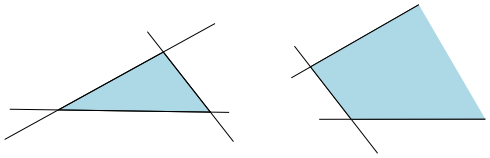


Abbildung 1: Zwei Beispiele von verallgemeinerten Dreiecken.

Wir zeigen nun eine obere Schranke für die VC-Dimension von einfachen Kompositionen, also Kompositionen mit einer festen Funktion f. Dafür zeigen wir erst ein Hilfslemma. Wir notieren mit $\ln x$ den natürlichen Logarithmus zur Basis e.

Lemma 15.6. *Für $x > 0$ und $u \in \mathbb{R}$ gilt $x \leq u \ln x \implies x \leq 2u \ln u$*

Beweis. Wir nutzen, dass für jedes $x > 0$ gilt, dass $\ln x \leq \sqrt{x}$.

$$\begin{aligned} & x \leq u \ln x \\ \implies \quad & x \leq u\sqrt{x} \\ \implies \quad & \ln x \leq \ln u + \tfrac{1}{2} \ln x \\ \implies \quad & \tfrac{1}{2} \ln x \leq \ln u \\ \implies \quad & \ln x \leq 2 \ln u \end{aligned}$$

Die Aussage folgt nun durch einfaches Einsetzen. □

Satz 15.7. *Sei $\mathcal{H}$ eine Hypothesenklasse mit Grundmenge X und VC-Dimension höchstens d mit $3 \leq \mathrm{d} < \infty$. Sei $f : \{+1, -1\}^k \to \{+1, -1\}$ eine feste Funktion mit $k \geq 3$. Die VC-Dimension der Komposition $\mathcal{H}_f$ ist höchstens $4\mathrm{d}k \ln(2\mathrm{d}k)$.*

Beweis. Sei $A \subseteq X$ eine Menge, die von dem zugehörigen Mengensystem $\mathcal{R}_f$ aufgespalten wird. Wir folgen nun derselben Strategie wie in dem Beweis zu Lemma 15.1. Die Herausforderung besteht darin, eine obere Schranke für $\left|\mathcal{R}_f|_A\right|$ zu finden. Zur Erinnerung, diese Menge ist wie folgt definiert.

$$\mathcal{R}_f|_A = \{\ r \cap A \ \mid \ r \in \mathcal{R}_f\ \}$$

Laut Definition des Mengensystems wissen wir, dass für jede Menge $r \in \mathcal{R}_f$ Hypothesen $h_1, \ldots, h_k \in \mathcal{H}$ existieren, sodass

$$r = \{\ x \in X \ \mid \ f(h_1(x), \ldots, h_k(x)) = 1\ \}$$

Also ist

$$r \cap A = \{\ x \in A \ \mid \ f(h_1|_A(x), \ldots, h_k|_A(x)) = 1\ \}$$

Daraus folgt, dass die Anzahl der verschiedenen Mengen $r \cap A$ mit $r \in \mathcal{R}_f$ nur von Funktionen in $\mathcal{H}|_A$ abhängt. Deren Anzahl ist durch die Wachstumsfunktion $\Pi_{\mathcal{H}}(t)$ beschränkt. Insbesondere entsteht eine Menge $r \cap A$ indem wir k Hypothesen aus $\mathcal{H}|_A$ auswählen. Also ist laut dem Wachstumslemma

$$\left|\mathcal{R}_f|_A\right| \leq \left|\mathcal{H}|_A\right|^k \leq (\Pi_{\mathcal{H}}(t))^k \leq \left(\frac{\mathrm{e}t}{\mathrm{d}}\right)^{\mathrm{d}k} \leq t^{\mathrm{d}k} \tag{2}$$

wobei wir nutzen, dass $\mathrm{d} \geq 3$ angenommen wird.

Daraus leiten wir ab, dass $2^t \leq t^{\mathrm{d}k}$ und durch Logarithmieren mit dem natürlichen Logarithmus auf beiden Seiten ergibt sich

$$t \ln 2 \leq (\mathrm{d}k) \ln t$$

Da $\ln 2 > 0.5$ ergibt sich durch Umformen $t \leq 2\mathrm{d}k \ln t$. Nun können wir Lemma 15.6 anwenden und erhalten

$$t \leq 4\mathrm{d}k \ln(2\mathrm{d}k)$$

Da dies für beliebige Mengen A gilt, die durch das Mengensystem aufgespalten werden, ergibt sich die obere Schranke für die VC-Dimension. □

Aus obigen Satz folgt nun für die Mengensysteme in unseren Beispielen, dass die VC-Dimension von Dreiecken durch eine Konstante beschränkt ist und für die Hyperquader in $\mathbb{R}^d$ ergibt sich zusammen mit Lemma 15.1 eine obere Schranke von $O(d \log^2 d)$.

3 VC-Dimension des Boostings

Satz 15.8. *Sei $\mathcal{H}$ eine Hypothesenklasse mit Grundmenge X und VC-Dimension höchstens* d *mit* $3 \leq \mathrm{d} < \infty$. *Sei C die Klasse von Funktionen $f : \{+1, -1\}^k \to \{+1, -1\}$ der Form $f(y_1, \ldots, y_k) = \operatorname{sign}(\sum_{1 \leq i \leq k} \alpha_i y_i)$ mit $\alpha_i \geq 0$ und sei $k \geq 3$. Die VC-Dimension der Komposition $\mathcal{H}_C$ ist höchstens* $4(\mathrm{d}+1)k \ln(2(\mathrm{d}+1)k)$.

Beweis. Wir folgen wieder derselben Strategie wie in dem Beweis zu Lemma 15.1. Der Beweis ist ähnlich zu dem Beweis zu Satz 15.7. Wir müssen allerdings zusätzlich die verschiedenen Funktionen in C beachten.

Sei $A \subseteq X$ eine Menge, die von $\mathcal{R}_C$ aufgespalten wird und sei $t = \big|A\big|$. Wie zuvor wollen wir wieder eine obere Schranke für die Anzahl der verschiedenen Mengen in $\mathcal{R}_C|_A$ finden, und nutzen, dass $2^t \leq \big|\mathcal{R}_C|_A\big|$ gelten muss. Zur Erinnerung,

$$\mathcal{R}_C|_A = \{\ r \cap A \ \mid \ r \in \mathcal{R}_C\ \}$$

Betrachte eine konkrete Teilmenge $A' \subseteq A$. Falls A' abgespalten wird, dann existiert eine Menge $r \in \mathcal{R}_C$ sodass $A' = r \cap A$. Die Menge r ist definiert durch konkrete Hypothesen $h_1, \ldots, h_k \in \mathcal{H}$ und eine konkrete Funktion $f \in C$ mit

$$r = \{\ x \in X \ \mid \ f(h_1(x), \ldots, h_k(x)) = 1\ \}$$

Wie zuvor haben wir

$$r \cap A = \{\ x \in A \ \mid \ f(h_1|_A(x), \ldots, h_k|_A(x)) = 1\ \}$$

Wir wissen aus der vorherigen Analyse im Beweis zu Satz 15.7, dass für ein festes $f \in C$ höchstens $(\Pi_{\mathcal{H}}(t))^k$ verschiedene Mengen erzeugt werden können, weil wir uns auf die Funktionen in $\mathcal{H}|_A$ beschränken können.

Ähnlich wollen wir nun auch die Funktionen $f \in C$ beschränken. Dafür stellen wir zunächst eine andere Frage. Wieviele Mengen können erzeugt werden, wenn wir k Hypothesen aus $\mathcal{H}$ festhalten und $f \in C$ frei wählen können?

Seien $h_1, \ldots, h_k$ fest und betrachte die Menge

$$B = \{\ (h_1(x), \ldots, h_k(x)) \ \mid \ x \in A\ \}$$

Beachte, dass $\big|B\big| = \big|A\big| = t$.

Wir betrachten nun das Mengensystem $\mathcal{R}'$ mit Grundmenge $\{+1,-1\}^k$ in der jede Menge definiert ist durch eine Funktion $f \in C$ mit

$$r_f = \left\{ (y_1,\dots,y_k) \in \{+1,-1\}^k \;\middle|\; f(y_1,\dots,y_k) = 1 \right\}$$

Betrachten wir dieses Mengensystem genauer, dann stellen wir fest, dass es sich um ein Mengensystem von Halbräumen in $\mathbb{R}^k$, beschränkt auf die Grundmenge $\{+1,-1\}^k$, handelt.

Insbesondere ist f definiert durch $\alpha_1,\dots,\alpha_k \in \mathbb{R}$ mit

$$f(y_1,\dots,y_k) = \begin{cases} 1 & \text{falls } \sum_{1\le i\le k} \alpha_i y_i \ge 0 \\ -1 & \text{sonst} \end{cases}$$

Für $\mathbf{w} = (\alpha_1,\dots,\alpha_k)$ und $u = 0$, sowie $\mathbf{y} = (y_1,\dots,y_k)$ ist also

$$\mathbf{y} \in r_f \quad \Leftrightarrow \quad \langle \mathbf{w}, \mathbf{y} \rangle \ge u$$

Das heißt, r_f enthält genau solche $\mathbf{y} \in \{+1,-1\}^k$ die in dem Halbraum liegen, der durch $\mathbf{w}$ und u definiert ist. Da die VC-Dimension von Halbräumen in $\mathbb{R}^k$ gleich k ist, erhalten wir mit dem Wachstumslemma

$$\big|\mathcal{R}'|_B\big| \le \Pi_{\mathcal{R}'}(t) \le \left(\frac{\mathrm{e}t}{k}\right)^k$$

Diese Erkenntnis können wir nun verwenden um eine obere Schranke für die Anzahl der verschiedenen Mengen $r \cap A$ mit $r \in \mathcal{R}_C$ herzuleiten. Indem wir k verschiedene Hypothesen aus $\mathcal{H}$ auswählen, können wir höchstens $(\Pi_{\mathcal{H}}(t))^k$ verschiedene Mengen B erzeugen. Jede solche Menge B entspricht einer Art, den Elementen in A jeweils k Labels aus $\{+1,-1\}$ zuzuweisen. Nun können wir für jede solche Menge B eine Funktion f auswählen. Für eine feste Menge B können wir dadurch höchstens $\Pi_{\mathcal{R}'}(t)$ verschiedene Mengen erzeugen. Also erhalten wir

$$\big|\mathcal{R}_C|_A\big| \le (\Pi_{\mathcal{H}}(t))^k \,\Pi_{\mathcal{R}'}(t) \le \left(\frac{\mathrm{e}t}{\mathrm{d}}\right)^{\mathrm{d}k} \left(\frac{\mathrm{e}t}{k}\right)^k \le t^{(\mathrm{d}+1)k} \tag{3}$$

wobei wir nutzen, dass $k \ge 3 \ge \mathrm{e}$ und $\mathrm{d} \ge 3 \ge \mathrm{e}$. Nun können wir wieder Lemma 15.6 benutzen und erhalten

$$t \le 4(\mathrm{d}+1)k \ln(2(\mathrm{d}+1)k)$$

□

Referenzen

- Understanding Machine Learning, Kapitel 10.3 (VC-Dimension of Boosting)
- Foundations of Machine Learning, Kapitel 7.3 (VC-Dimension of Boosting)

AGML, Sommersemester 2020 Vorlesung 16 (5 Seiten)

Entscheidungsbäume

Anne Driemel Letzte Aktualisierung: 22. Juni 2020

Entscheidungsbäume sind eine beliebte Form um eine Klassifizierung anhand eine Reihe von Tests darzustellen. Damit kann zum Beispiel auf kompakte Weise dargestellt werden, ob eine Person, die eine bestimmte Kombination von Krankheitssymptomen vorweist, einen Arzt aufsuchen sollte oder sich in häusliche Quarantäne begeben sollte.[1] Entscheidungsbäume werden in der Praxis oft per Hand von einem Experten erstellt, zum Beispiel im Rahmen einer Risikoanalyse.

Im Maschinellen Lernen werden Entscheidungsbäume genutzt um komplexe Kompositionen von einfachen Hypothesen darzustellen. Oft wird als Basis die Klasse der Schwellenwertfunktionen genutzt, also Halbräume die durch achsenparallele Hyperebenen beschränkt sind. Denkbar sind aber auch beliebige Halbräume als Basis.

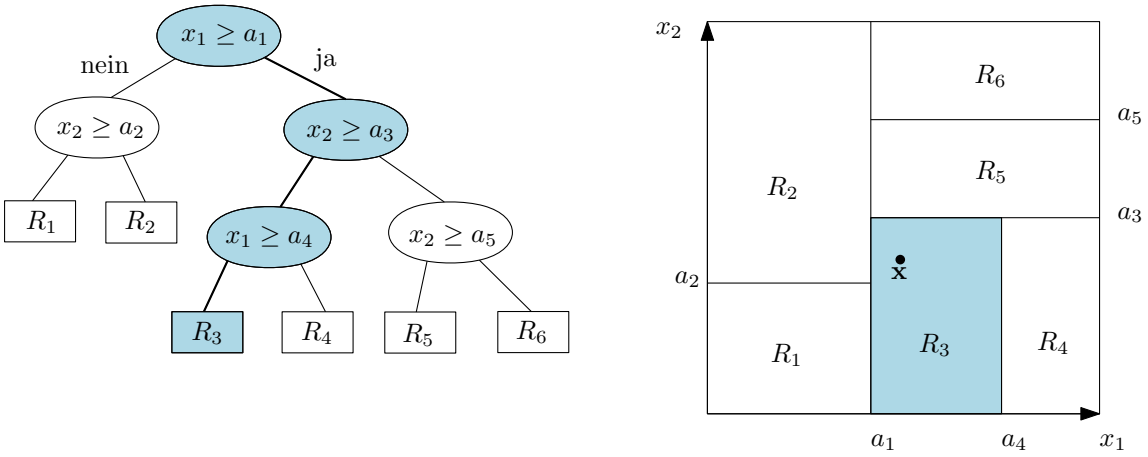


Abbildung 1: Beispiel für die rekursive Partitionierung der Grundmenge $[0,1]^2$ durch einen Entscheidungsbaum mit der Klasse der Schwellenwertfunktion als Basisklasse.

Diese Funktionen werden nun in Form eines binären Baumes rekursiv miteinander kombiniert. Als Ergebnis entsteht eine rekursive Partitionierung der Grundmenge. Jedem Blattknoten ist das Label einer Klasse zugewiesen. Um einen Punkt der Grundmenge zu klassifizieren, folgt man einem Pfad von der Wurzel bis zu dem Blatt, das den Punkt $\mathbf{x}$ enthält und gibt dann das entsprechende Label aus. Siehe Abbildung 1 für ein Beispiel einer Partitionierung für den Fall, dass die Grundmenge $[0,1]^2$ ist.

Der Vorteil einer achsenparallelen Partitionierung gegenüber einer Partitionierung mit Halbräumen, ist, dass jeder Knoten des Entscheidungsbaumes einem Test bezüglich einer festen Komponente des Feature-Vektors darstellt. Zum Beispiel könnte eine bestimmte Komponente darstellen ob und wie stark ein bestimmtes Krankheitssymptom bei einer Person aufgetreten ist. Somit ist die vom Lernalgorithmus berechnete Hypothese von Menschen besser interpretierbar. Daher werden Schwellwertfunktionen auch im Maschinellen Lernen in der Praxis manchmal gegenüber allgemeinen Halbräumen bevorzugt.

[1] Siehe hier für das Corona-Virus
https://www.zeit.de/wissen/gesundheit/2020-03/Coronavirus-Entscheidungshilfe-2020-03-31.pdf

1 Hypothesenklasse

Formal kann man die resultierende Hypothese wie folgt definieren. Sei T ein binärer Baum mit k inneren Knoten und Wurzel w, wobei jedem inneren Knoten v eine Hypothese $h_v \in \mathcal{H}$ und jedem Blattknoten v ein Label $\ell_v \in \{+1, -1\}$ zugewiesen ist. Sei v ein innerer Knoten und sei v_L das linke Kind von v in T und sei v_R das rechte Kind von v in T. Definiere die Funktion $g_v : X \to \{+1, -1\}$ für einen inneren Knoten v mit

$$g_v(\mathbf{x}) = \begin{cases} g_{v_R}(\mathbf{x}) & \text{falls } h_v(\mathbf{x}) = 1 \\ g_{v_L}(\mathbf{x}) & \text{sonst} \end{cases} \tag{1}$$

Falls v ein Blattknoten ist, dann definieren wir $g_v(\mathbf{x}) = \ell_v$.

Die Hypothese g_w, definiert durch den Baum T und den assoziierten Hypothesen an den inneren Knoten sowie den Labelzuweisungen an den Blättern von T, stellt dann einen Entscheidungsbaum mit Basisklasse $\mathcal{H}$ dar. Dies ist eine Komposition, ähnlich wie wir sie in der letzten Vorlesung definiert haben, wobei der Baum T die Kompositionsfunktion definiert. Der einzige Unterschied ist, dass wir zusätzlich zu der Basisklasse auch die Labelzuweisungen an den Blättern haben.

Definition 16.1. *Sei B_k die Menge der gewurzelten binären Bäume mit k inneren Knoten und $k+1$ Blättern, wobei jeder innere Knoten v ein linkes Kind v_L und ein rechtes Kind v_R hat, und genau einen Elternknoten.*

Definition 16.2 (Entscheidungsbaum)**.** *Sei $\mathcal{H}$ eine Hypothesenklassen mit Grundmenge X und sei $\mathcal{R}$ das zugehörige Mengensystem. Sei $\mathcal{H}_{B_k}$ die Hypothesenklasse aller Funktionen $g_w : X \to \{+1, -1\}$ definiert wie in (1) durch*

(i) einen binären Baum $T \in B_k$ mit inneren Knoten $v_1, \ldots, v_k$ und Blättern $b_1, \ldots b_{k+1}$,

(ii) Hypothesen $h_1, \ldots, h_k \in \mathcal{H}$ und Labels $\ell_1, \ldots \ell_{k+1} \in \{+1, -1\}$

Wir legen dabei fest, dass $w = v_1$ die Wurzel des Baumes T ist und dass, für $1 \leq i \leq k$, die Hypothese h_i dem inneren Knoten v_i zugewiesen ist, sowie dass, für $1 \leq i \leq k+1$, das Label ℓ_i dem Blatt b_i zugewiesen ist.

Das folgende Lemma wird uns helfen, eine obere Schranke für die VC-Dimension der Hypothesenklasse der Entscheidungsbäume mit k inneren Knoten zu zeigen.

Lemma 16.3. *Für jede natürliche Zahl $k \geq 1$ ist $\left|B_k\right| \leq k!$*

Beweis. Wir zeigen dies durch Induktion. Sei $k = 1$. In diesem Fall gibt es nur einen Baum in B_k, nämlich die Wurzel selbst mit zwei Blättern. Also ist $\left|B_1\right| = 1 = 1!$ korrekt.

Sei $k > 1$. Wir können jeden Baum $T \in B_k$ aus einem Baum $T' \in B_{k-1}$ erzeugen, indem wir in T' einen Blattknoten entfernen und an derselben Stelle einen Knoten mit zwei neuen Blattknoten als Kindern hinzufügen. Tatsächlich hat der so erzeugte Baum k innere Knoten und $k+1$ Blattknoten (ein Blattknoten wurde entfernt und zwei neue Blattknoten sind hinzugekommen). Die Eigenschaft, dass jeder innere Knoten zwei Kinder hat bleibt dadurch gleichermaßen unberührt.

Für einen festen Baum $T' \in B_{k-1}$ gibt es genau k verschiedenen Möglichkeiten solch einen Baum in B_k zu erzeugen, da T' genau k Blattknoten hat. Also ist

$$\left|B_k\right| \leq k \cdot \left|B_{k-1}\right|$$

Nun können wir die Induktionsannahme für $\left|B_{k-1}\right|$ einsetzen und erhalten

$$\left|B_k\right| \leq k \cdot \left|B_{k-1}\right| \leq k \cdot (k-1)! \leq k!$$

(Es kann passieren, dass der gleiche Baum in B_k durch zwei verschiedene Bäume in B_{k-1} erzeugt wird, aber das stört uns nicht, da wir nur eine obere Schranke zeigen wollen.) □

2 VC-Dimension

Satz 16.4. *Sei $\mathcal{H}$ eine Hypothesenklasse mit Grundmenge X und VC-Dimension* d *mit* $\mathrm{d} < \infty$. *Sei $k \geq 2$ eine natürliche Zahl. Die VC-Dimension von $\mathcal{H}_{B_k}$ ist höchstens* $20\mathrm{d}k\ln(10k)$.

Beweis. Wir nutzen einen ähnlichen Beweis wie in der letzten Vorlesung. Diesmal müssen wir die Anzahl der verschiedenen Kompositionsfunktionen miteinbeziehen, die durch verschiedene Bäume in B_k entstehen können.

Sei $A \subseteq X$ eine Menge, die von $\mathcal{H}_{B_k}$ aufgespalten wird und sei $t = \big|A\big|$. Wir wollen wieder eine obere Schranke für die Anzahl Hypothesen in $\mathcal{H}_{B_k}|_A$ finden, und nutzen, dass $2^t \leq \big|\mathcal{H}_{B_k}|_A\big|$.

Laut Lemma 16.3 gibt es höchstens $k!$ verschiedene Bäume in B_k. Weiter müssen wir k Hypothesen aus $\mathcal{H}|_A$ auswählen. Für die Zuweisung der Labels an die $k+1$ Blätter des Baumes gibt es 2^{k+1} Möglichkeiten. Damit wäre eine Hypothese in $\mathcal{H}_{B_k}|_A$ eindeutig identifiziert.

Wir können also die Anzahl der Hypothesen in $\mathcal{H}_{B_k}|_A$ abschätzen indem wir die Anzahl der verschiedenen Bäume, die Anzahl der verschiedenen Label-Zuweisungen an die Blätter und die Anzahl der Möglichkeiten, k Hypothesen an die inneren Knoten zuzuweisen, miteinander multiplizieren. Es ergibt sich also

$$\big|\mathcal{H}_{B_k}|_A\big| \leq k! \cdot 2^{k+1} \cdot \big|\mathcal{H}|_A\big|^k \leq k! \cdot 2^{k+1} \cdot (\Pi_{\mathcal{H}}(t))^k \leq k! \cdot 2^{k+1} \cdot \left(\frac{\mathrm{e}t}{\mathrm{d}}\right)^{\mathrm{d}k}$$

wobei die letzte Ungleichung wieder aus dem Wachstumslemma folgt. Da alle 2^t verschiedenen Teilmengen von A dargestellt werden, ergibt sich ähnlich wie zuvor, dass

$$2^t \leq k! \cdot 2^{k+1} \cdot \left(\frac{\mathrm{e}t}{\mathrm{d}}\right)^{\mathrm{d}k} \leq k^k \cdot k^{k+1} \cdot \left(\frac{t}{\mathrm{d}}\right)^{2\mathrm{d}k} \leq \left(\frac{t}{\mathrm{d}}\right)^{k+k+1+2\mathrm{d}k} \leq \left(\frac{t}{\mathrm{d}}\right)^{5\mathrm{d}k}$$

wobei wir hier annehmen, dass $t \geq \mathrm{de}$, und dass $t \geq \mathrm{d}k$, sonst ist die Aussage trivial erfüllt. Wir können nun beide Seiten logarithmieren und erhalten

$$t\ln 2 \quad \leq 5\mathrm{d}k\ln\tfrac{t}{\mathrm{d}}$$

Da $0.5 \leq \ln 2$, ist also

$$\frac{t}{\mathrm{d}} \quad \leq 10k\ln\tfrac{t}{\mathrm{d}}$$

In Lemma 15.6 hatten wir gezeigt, dass für jedes $x > 0$ und $u \in \mathbb{R}$ gilt

$$x \leq u\ln x \implies x \leq 2u\ln u$$

Das können wir nun mit $x = \frac{t}{\mathrm{d}}$ und $u = 10\mathrm{d}k$ anwenden und erhalten

$$t \leq 20\mathrm{d}k\ln(10k)$$

Da dies für jede Menge A gilt, die aufgespalten wird, folgt direkt die obere Schranke für die VC-Dimension. □

Satz 16.4 besagt, dass die VC-Dimension von Entscheidungsbäumen nur von der Anzahl der inneren Knoten k und von der VC-Dimension der Basisklasse abhängt. Gleichzeitig kann man für jede Menge $S \subseteq \mathbb{R}$ der Größe m einen Entscheidungsbaum mit $k = m - 1$ inneren Knoten finden, der Trainingsfehler null hat. Das heißt, für $k \to \infty$ ist die VC-Dimension von $\mathcal{H}_{B_k}$ im schlimmsten Fall unbeschränkt.

3 Lernalgorithmen

Da bei Entscheidungsbäumen, ähnlich wie beim Boosting, die VC-Dimension mit der Komplexität der Hypothesenklasse steigt, besteht auch hier die Gefahr des Overfittings. Aus diesem Grund will man in der Praxis die Anzahl der inneren Knoten des Baumes beschränken.

Das ist aber oft nicht effizient möglich. Es ist zum Beispiel NP-schwer für eine gegebene Menge $S \subseteq \mathbb{R}^3 \times \{+1, -1\}$ und einen Parameter $k \in \mathbb{N}$ einen optimalen Entscheidungsbaum mit k inneren Knoten zu finden, wenn als Basisklasse die Klasse der Halbräume angenommen wird. Das gilt selbst in dem vergleichsweise einfachen Fall, dass die Labels aus der Menge $\{+1, -1\}$ kommen und die Dimension der Grundmenge $d = 3$ ist.

Daher werden in der Praxis Entscheidungsbäume meist heuristisch optimiert, indem die Knoten nacheinander hinzugefügt werden, wobei in jedem Schritt die Zielfunktion lokal optimiert wird. Der Algorithmus muss lokal entscheiden, welcher Knoten hinzugefügt wird und nimmt meist den Knoten, dessen assoziierte Trainingsmenge den größten Trainingsfehler hat. Denkbar ist auch, erst einen größeren Baum zu bauen und dann heuristisch Unterbäume zu entfernen. Das Problem dabei ist aber, dass selbst die erste Hypothese im Wurzelknoten die optimale Lösung blockieren kann.

3.1 Greedy-Algorithmus

Wir wollen trotzdem eine einfache Variante dieses Greedy-Algorithmus genauer definieren. Der Algorithmus bekommt als Eingabe einen Parameter k und eine Datenpunkt/Label-Menge $S = ((x_1, y_1), \ldots, (x_m, y_m))$. Jeder Blattknoten v hat eine assoziierte Menge $S_v \subseteq S$, welche nur für die Konstruktion des Baumes verwendet wird.

`decisionTree`**(k,S)**
1. Initialisiere T mit einem Blattknoten v
2. `initLeaf`(v,S)
3. **for** i **in** $1 \ldots k$ **do**
4. Finde Blattknoten v in T mit größtem Klassifizierungsfehler $err_{S_v}(T)$
5. `split`(v, T)
6. **for** v **in** Menge der Blattknoten von T **do**
7. Entferne die Menge S_v von dem Blattknoten v
8. Gebe den Baum T zurück

`split`**(v,T)**
1. Berechne eine Hypothese $h \in \mathcal{H}$, welche $err_{S_v}(h)$ minimiert
2. Assoziiere mit v die Hypothese h
3. Entferne die Menge S_v von v
4. Füge v_L als linkes Kind von v zu T hinzu
5. Sei $S_{v_L} = \{ (x, y) \in S_v \mid h(x) = -1 \}$
6. `initLeaf`(v_L, S_{v_L})
7. Füge v_R als rechtes Kind von v zu T hinzu
8. Sei $S_{v_R} = \{ (x, y) \in S_v \mid h(x) = +1 \}$
9. `initLeaf`(v_L, S_{v_L})

`initLeaf(v,S)`
1. Assoziiere mit v die Menge S
2. Assoziiere mit v das Label welches unter den Punkten in S_v am meisten vertreten ist

3.2 Random-Forest-Algorithmus

Um die Stabilität des Lernalgorithmus zu verbessen, werden Entscheidungsbäume oft auf zufällig gewählten Untermengen der Trainingsmenge heuristisch berechnet und die entstandenen Hypothesen mit zufällig gewählten Gewichten kombiniert. Dieser Lernalgorithmus wird als *Random-Forest-Algorithmus* bezeichnet. Die VC-Dimension kann hier wieder durch die Linearkombination der einzelnen Hypothesen wachsen. Allerdings tritt bei Random Forests das Phänomen des Overfittings in der Praxis fast nie auf.

Referenzen

- Understanding Machine Learning, Kapitel 18 (Decision Trees)
- Foundations of Machine Learning, Kapitel 9.3.3 (Decision Trees)
- Michael T. Goodrich , Vincent Mirelli , Mark Orletsky , Jeffery Salowe, "Decision Tree Construction in Fixed Dimensions: Being Global is Hard but Local Greed is Good" Technical Report TR-95-1, Johns Hopkins University, 1995.

AGML, Sommersemester 2020 Vorlesung 17 (7 Seiten)

Nächste Nachbarn

Anne Driemel Letzte Aktualisierung: 25. Juni 2020

Ein grundlegender Lernalgorithmus im Maschinellen Lernen ist der Nächste-Nachbarn-Algorithmus. Die Idee ist sehr einfach. Um einen Punkt $q \in X$ auf Basis einer Trainingsmenge $S \subseteq X \times \{-1, +1\}$ zu klassifizieren, berechnen wir den Punkt in S, der q am ähnlichsten ist und geben das entsprechende Label zurück. Dafür müssen wir die Ähnlichkeit zunächst definieren. Einfacher ist es meist, den Punkt zu betrachten, der den geringsten Abstand unter einem bestimmten Distanzmaß hat. Wir betrachten hier zunächst den Euklidischen Abstand. Unsere Hypothese ist also die folgende Funktion $h_S : X \to \{+1, -1\}$ definiert durch

$$h_S(x) = y_i \qquad \text{mit} \qquad i = \operatorname*{arg\,min}_{1 \le i \le m} \|x - x_i\|$$

In diesem Kontext bezeichnen wir x_i als den nächsten Nachbarn von x in S.

Diese einfache Variante der Nächste-Nachbarn Hypothese leidet unter dem Problem des Overfittings. Um dem entgegen zu wirken, werden oft die Labels der k nächsten Nachbarn betrachtet, wobei $k \in \mathbb{N}$ ein Parameter ist. Formal können wir die resultierende Hypothese wie folgt definieren. Für ein $x \in X$ sei $\pi_x : \{1, \dots, m\} \to \{1, \dots, m\}$ eine Bijektion der Menge S auf sich selbst, sodass für alle $i, j \in \{1, \dots, m\}$ gilt

$$\pi_x(i) \le \pi_x(j) \qquad \Longrightarrow \qquad \|x - x_i\| \le \|x - x_j\|$$

Das heißt, π_x stellt eine Permutation der Menge S dar, welche einer aufsteigend sortierten Reihenfolge bezüglich des Abstands zu x entspricht. [1]

Sei $\mathcal{N}_k(x)$ die Indexmenge der k nächsten Nachbarn von x in S. Formal,

$$\mathcal{N}_k(x) = \left\{ \; \pi_x^{-1}(i) \;\; \middle| \;\; 1 \le i \le k \; \right\}$$

Die k-NN Hypothese ist die Funktion $h_{S,k} : X \to \{+1, -1\}$ definiert durch

$$h_{S,k}(x) = \operatorname*{arg\,max}_{\ell \in \{+1,-1\}} \left| \{ \; j \in \mathcal{N}_k(x) \;\; | \;\; y_j = \ell \; \} \right|$$

Wir bezeichnen das algorithmische Problem, die k nächsten Nachbarn in einer Menge zu finden als das k-NN Problem.

Obwohl wir immer noch von Hypothesen sprechen, macht es hier keinen Sinn, die VC-Dimension der entsprechenden Hypothesenklasse zu betrachten. Wir würden dann feststellen, dass die VC-Dimension von der Größe von S abhängt und hätten dann keine Möglichkeit mehr, im Rahmen des PAC-Frameworks, die minimale Größe von S anhand der VC-Dimension festzulegen. Nichtsdestotrotz bildet die Klasse der Lernalgorithmen, die auf dem Prinzip der nächsten Nachbarn basiert, eine grundlegende Methode im Maschinellen Lernen.

[1] Beachte, dass π_x dadurch noch nicht eindeutig definiert ist, da es nicht für jedes x eine eindeutige Permutation der nächsten Nachbarn gibt. Wir definieren deshalb ausserdem die folgende Bedingung, welche die Permutation eindeutig macht.

$$\pi_x(i) < \pi_x(j) \text{ und } \|x - x_i\| = \|x - x_j\| \implies i < j$$

1 Voronoi-Diagramme

Für eine feste Menge S lässt sich die Hypothese h_S (bzw. $h_{S,k}$) durch ein sogenanntes Voronoi-Diagram darstellen. Bei der Hypothese $h_{S,k}$ sprechen wir dann von einem Voronoi-Diagramm der k-ten Ordnung.

Definition 17.1. *Sei $S \subseteq \mathbb{R}^d$ mit $|S| = m$. Sei $k \leq m$ eine natürliche Zahl. Die Voronoi-Region einer Menge $A \subseteq \{1, \ldots, m\}$ mit $|A| = k$ ist die Menge*

$$\mathcal{V}_k(A) = \left\{ x \in \mathbb{R}^d \;\middle|\; \mathcal{N}_k(x) = A \right\}$$

Das Voronoi-Diagramm ist die Unterteilung des Raumes $\mathbb{R}^d$ in die Voronoi-Regionen für alle $A \subseteq \{1, \ldots, m\}$ mit $|A| = k$.

Das Voronoi-Diagram ist also die Unterteilung der Grundmenge in genau die Regionen, für die die Ausgabe des k-NN Problems gleich ist. Jede Strukturierung der Trainingsmenge, die einen effiziente Beantwortung der Frage nach den k nächsten Nachbarn von einem Anfragepunkt x erlaubt, beantwortet implizit die Frage, in welcher Voronoi-Region sich x befindet. Wir interessieren uns deshalb für die Struktur des Voronoi-Diagramms und insbesondere die Komplexität des Diagramms. Wir werden feststellen, dass das Voronoi-Diagram für $k = 1$ und $d = 2$ eine überraschend einfache Struktur hat.

1.1 k-NN auf der Geraden

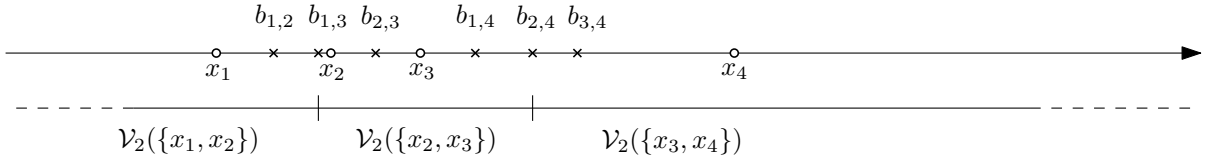
Für $d = 1$ betrachten wir das arithmetische Mittel zwischen zwei Punkten der Trainingsmenge, $b_{i,j} = \frac{x_i + x_j}{2}$. Der Wert $b_{i,j}$ unterteilt die Grundmenge in zwei disjunkte Intervalle

$$I_- = (-\infty, b_{i,j}) \qquad \text{und} \qquad I_+ = (b_{i,j}, \infty)$$

Dabei gilt für ein beliebiges Paar von Punkten $x, x' \in \mathbb{R} \setminus \{b_{i,j}\}$, dass sie genau dann demselben Intervall angehören, wenn sie in der Menge $\{x_i, x_j\}$ denselben nächsten Nachbarn haben.

Allgemeiner, können wir die Werte $b_{i,j}$ der Menge $\binom{S}{2}$ betrachten, also der Menge aller Punktepaare aus S. Diese unterteilen die Grundmenge $\mathbb{R}$ in eine beschränkte Anzahl von Intervallen, sodass in jedem Intervall die Permutation π_x für alle Punkte x in dem Intervall gleich ist. Im Voronoi-Diagram der k-ten Ordnung fassen wir all jene Intervalle zu einer Menge zusammen, bei der die k nächsten Nachbarn, also die Menge $\mathcal{N}_k(x)$, gleich sind.

Beispiel 17.2. *Sei $k = 2$ und seien $x_1, x_2, x_3, x_4 \in \mathbb{R}$ wie folgt*



Für $k = 2$ haben wir in diesem Beispiel die folgenden nicht-leeren Voronoi-Regionen:
$\mathcal{V}_2(\{x_1, x_2\}) = (-\infty, b_{1,3}]$ $\mathcal{V}_2(\{x_2, x_3\}) = (b_{1,3}, b_{2,4}]$ $\mathcal{V}_2(\{x_3, x_4\}) = (b_{2,4}, \infty)$.

Man kann zeigen, dass das Voronoi-Diagramm von m Punkten in $\mathbb{R}$ aus genau $m - k + 1$ nicht-leeren Voronoi-Regionen besteht, die jeweils ein zusammenhängendes Intervall bilden. Es hat also höchstens lineare Komplexität. Für $d = 2$ kann man allerdings Punktmengen finden, für

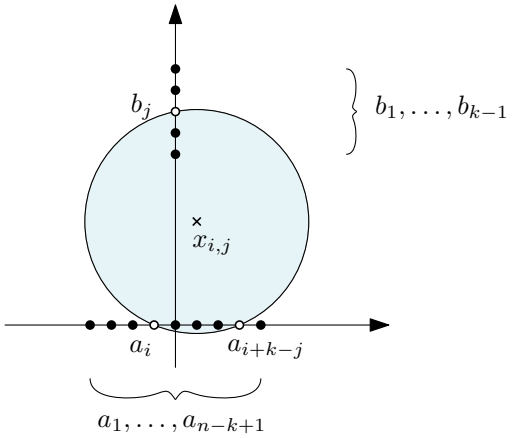


Abbildung 1: Es gibt Punktmengen mit mindestens $(k-1)(m-2k)$ nicht-leeren Voronoi-Regionen in der Ebene.

die das Voronoi-Diagramm der k-ten Ordnung mindestens $(m-2k)(k-1)$ nicht-leere Voronoi-Regionen enthält. Es hat also im schlimmsten Fall mindestens quadratische Komplexität. Im Beispiel in Abbildung 1 gibt es Punkte $a_1, \ldots, a_{m-k+1}$ auf der x-Achse und $k-1$ Punkte auf der y-Achse, die so gewählt sind, dass für jede Koembination von Indizes $(i,j) \in \{1, \ldots, m-2k\} \times \{1, \ldots, k-1\}$ ein Kreis existiert, der genau die Punkte $A_{i,j} = \{b_1, \ldots, b_j\} \cup \{a_i, \ldots, a_{i+k-j}\}$ enthält. Der Mittelpunkt dieses Kreises ist also enthalten in der Voronoi-Region $\mathcal{V}_k(A_{i,j})$. Das bedeutet, dass diese Voronoi-Region nicht leer ist. Also gibt es mindestens $(m-2k)(k-1)$ nicht-leere Voronoi-Regionen.

1.2 1-NN in der Ebene

Für den Fall $k = 1$ hat das Voronoi-Diagram eine überraschend einfache geometrische Struktur. Die Punkte $b_{i,j}$, an denen sich die Permutation der nächsten Nachbarn für $d = 1$ ändert, können wir verallgemeinern zu dem Bisektor, der wie folgt definiert ist.

Definition 17.3. *Der Bisektor $B(p,q)$ zwischen zwei Punkten $p \in \mathbb{R}^d$ und $q \in \mathbb{R}^d$ ist die Menge*

$$B(p,q) = \left\{ x \in \mathbb{R}^d \;\middle|\; \|p-x\| = \|q-x\| \right\}$$

Der Bisektor enthält alle Punkte, für die der Abstand zum Punkt p und der Abstand zum Punkt q genau gleich ist. Für feste p und q ist der Bisektor eine Hyperebene, wie sich leicht überprüfen lässt:

$$\begin{aligned}
& \|p-x\| = \|q-x\| \\
\Leftrightarrow \quad & \|p-x\|^2 = \|q-x\|^2 \\
\Leftrightarrow \quad & \langle p-x, p-x\rangle = \langle q-x, q-x\rangle \\
\Leftrightarrow \quad & \langle p,p\rangle + \langle x,x\rangle - 2\langle p,x\rangle = \langle q,q\rangle + \langle x,x\rangle - 2\langle q,x\rangle \\
\Leftrightarrow \quad & \langle p,p\rangle - 2\langle p,x\rangle = \langle q,q\rangle - 2\langle q,x\rangle \\
\Leftrightarrow \quad & 2\langle q,x\rangle - 2\langle p,x\rangle = \langle q,q\rangle - \langle p,p\rangle \\
\Leftrightarrow \quad & \langle 2(q-p), x\rangle = \langle q,q\rangle - \langle p,p\rangle \\
\Leftrightarrow \quad & \langle w_{p,q}, x\rangle = u_{p,q}
\end{aligned}$$

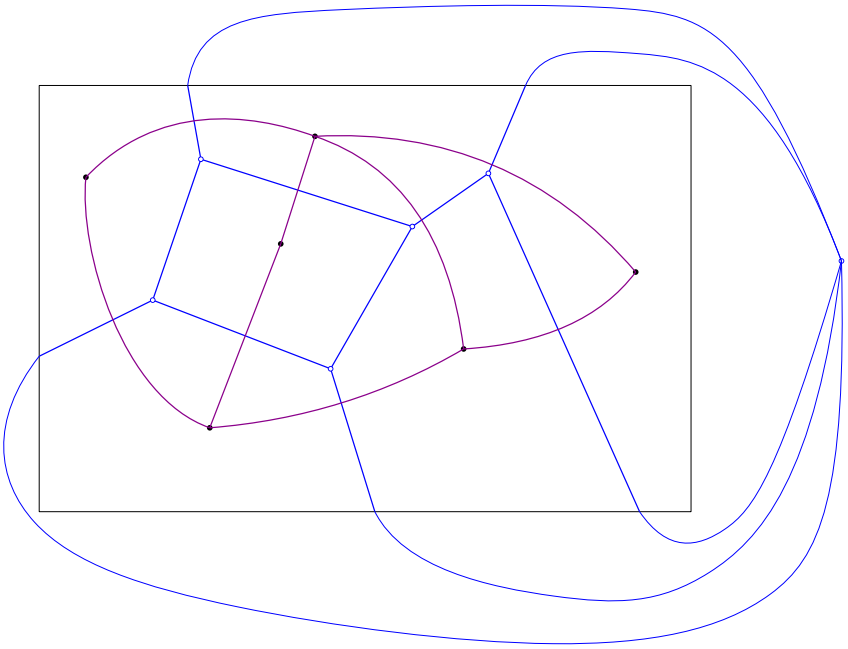


Abbildung 2: Im Kasten sieht man ein Aussschnitt des Voronoi-Diagramms der schwarzen Punkte ($k = 1$). Der blaue Knoten ist der virtuelle Knoten, der alle unbeschränkten Kanten verbindet. Die lila Kanten sind Kanten des dualen Graphen.

mit $w_{p,q} = 2(q - p) \in \mathbb{R}^d$ und $u_{p,q} = \langle q, q\rangle - \langle p, p\rangle \in \mathbb{R}$.

Der Bisektor unterteilt die Grundmenge in zwei offene Halbräume.

$$H_-(p,q) = \left\{ x \in \mathbb{R}^d \;\middle|\; \langle w_{p,q}, x\rangle < u_{p,q} \right\} \text{ und } H_+(p,q) = \left\{ x \in \mathbb{R}^d \;\middle|\; \langle w_{p,q}, x\rangle > u_{p,q} \right\}$$

Dabei gilt für ein beliebiges Paar von Punkten $x, x' \in \mathbb{R}^d \backslash B(p,q)$, dass sie genau dann demselben Halbraum angehören, wenn sie in der Menge $\{p, q\}$ denselben eindeutigen nächsten Nachbarn haben.

Die Voronoi-Region eines Punktes x_i in der Menge $S = \{x_1, \ldots, x_m\}$ ist die Menge der Punkte, für die x_i der eindeutige nächste Nachbar ist.[2]

$$\mathcal{V}_1(x_i) = \bigcap_{\substack{1 \le j \le m \\ i \neq j}} H_-(x_i, x_j)$$

Die Voronoi-Region ist also eine zusammenhängende Menge. Das folgt direkt aus der Konvexität der Halbräume und daraus, dass die Konvexität von Mengen unter endlichen Schnitten abgeschlossen ist.

Die Grenzen der Voronoi-Regionen bestehen aus Teilen der Bisektoren. In der Ebene formen diese zusammen einen Graphen mit Knoten und Kanten. Jeder Punkt auf einer Kante hat dabei den gleichen Abstand zu seinen zwei nächste Nachbarn. Jeder Punkt auf einem Knoten hat den gleichen Abstand zu seinen drei nächsten Nachbarn. Wir können die Anzahl der Knoten und Kanten im Voronoi-Diagramm wie folgt beschränken.

Satz 17.4. *Das Voronoi-Diagramm von m Punkten in $\mathbb{R}^2$ hat $O(m)$ Knoten und Kanten.*

[2]Mathematisch ist das nicht ganz korrekt, da wir die Voronoi-Regionen etwas anders definiert haben. Die Mengen unterscheiden sich aber nur am Rand. Wir sehen darüber um einer einfacheren Definition willen hinweg.

Beweis. Wir nutzen Eulers Formel für planare Graphen. Für einen Graphen G mit v Knoten, e Kanten und f Flächen besagt sie, dass

$$v - e + f = 2$$

Wir wollen diese Formel auf den Graphen der die Voronoi-Regionen begrenzt anwenden. Dafür müssen wir einen virtuellen Knoten hinzufügen, der mit allen unbeschränkten Kanten verbunden ist.[3] Wir wissen, dass $f = m$, da f die Flächen des Graphen mit den Voronoi-Regionen korrespondieren. Sei d_i die Anzahl der Kanten, die inzident zum $i$ten Voronoi-Knoten sind. Wir können die Summe der Knotengrade auf zwei Arten begrenzten,

$$2e = \sum_{i=1}^{v} d_i \geq 3v$$

da jede Voronoi-Kante zu genau 2 Voronoi-Knoten inzident ist, und da jeder Voronoi-Knoten zu mindestens zu 3 Voronoi-Kanten inzident ist. Wir nehmen hier an, dass $m > 2$, sonst ist die Aussage im Satz trivial erfüllt. Daraus folgt $v \leq \frac{2}{3}e$ und daher folgt aus Eulers Formel

$$e = f + v - 2 \leq m + \frac{2}{3}e - 2$$

Dies können wir umformen zu

$$e \leq 3(m - 2)$$

Also ist $e \in O(m)$. Daraus folgt auch, da $v \leq \frac{2}{3}e$, dass $v \in O(m)$. □

1.3 k-NN in der Ebene

Für $k > 1$ können wir und das Voronoi Diagram höherer Ordnung wie folgt vorstellen. Für jede Region $\mathcal{V}_1(x_i)$ im Voronoi-Diagram von S betrachten wir das Voronoi-Diagram von $S \setminus \{x_i\}$ beschränkt auf die Region $\mathcal{V}_1(x_i)$. Das gibt uns die Regionen $\mathcal{V}_2(\{x_i, x_j\}) \cap \mathcal{V}_1(x_i)$ für alle $i \neq j$. Das können wir rekursiv fortführen um weitere Voronoi-Diagramme höherer Ordnung für $k > 2$ zu finden. Allgemein kann man beobachten, dass die Voronoi-Regionen höherer Ordnung immer von Teilen der Bisektoren der Menge $\binom{S}{2}$ begrenzt werden. Insbesondere teilen die Bisektoren die Ebene in Regionen, sodass in jeder Region die Permutation der nächsten Nachbarn gleich ist.

1.4 Voronoi-Diagramme in höheren Dimensionen

In höheren Dimension steigt die Komplexität des Voronoi-Diagramms exponentiell mit der Dimension. Für $d = 3$ kann das Voronoi-Diagram schon quadratische Größe haben. Dafür konstruieren wir zwei windschiefe Geraden g_A und g_B, also zwei Geraden die nicht in derselben Ebene liegen. Sei $A = \{a_1, \ldots, a_n\}$ eine Menge von $n = \lceil \frac{m}{2} \rceil$ Punkten auf g_A und sei $B = \{b_1, \ldots, b'_n\}$ eine Menge von $n' = \lfloor \frac{m}{2} \rfloor$ Punkten auf g_B. Wir nehmen an, dass zwischen zwei Punkten a_i und a_{i+1} kein weiterer Punkt aus A auf g_A liegt, und ähnlich nehmen wir an, dass zwischen zwei Punkten b_i und b_{i+1} kein weiterer Punkt aus B auf g_B liegt. Nun können wir für jedes Tupel $(i, j) \in \{1, \ldots, n-1\} \times \{1, \ldots, n'-1\}$ die Kugel betrachten, die a_i, a_{i+1}, b_j und b_{j+1} auf dem Rand hat. Da die beiden Geraden windschief sind, liegen die vier Punkte nicht in einer Ebene und bestimmen somit eindeutig eine Kugel. Die Kugel enthält keine weiteren Punkte aus $A \cup B$. Daher ist das Zentrum der Kugel ein Knoten im Voronoi-Diagram von $A \cup B$. Daraus folgt, dass das Voronoi-Diagramm mindestens $(n-1)(n'-1) \in \Omega(m^2)$ Knoten hat.

[3] Wir könnten stattdessen auch den dualen Graphen betrachten, welcher auch ein planarer Graph ist. Dieser ist in Abbildung 2 abgebildet. Der virtuelle Knoten entspricht dann der äußeren Fläche.

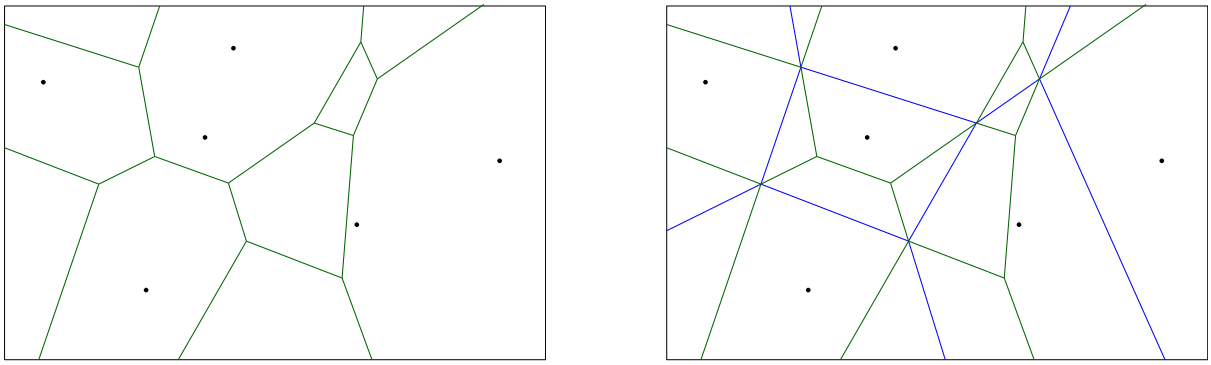


Abbildung 3: Links: Voronoi-Diagramm der zweiten Ordnung für die Punktmenge aus Abbildung 2; Rechts: Voronoi-Diagramme für $k = 1$ und $k = 2$ übereinander gezeichnet.

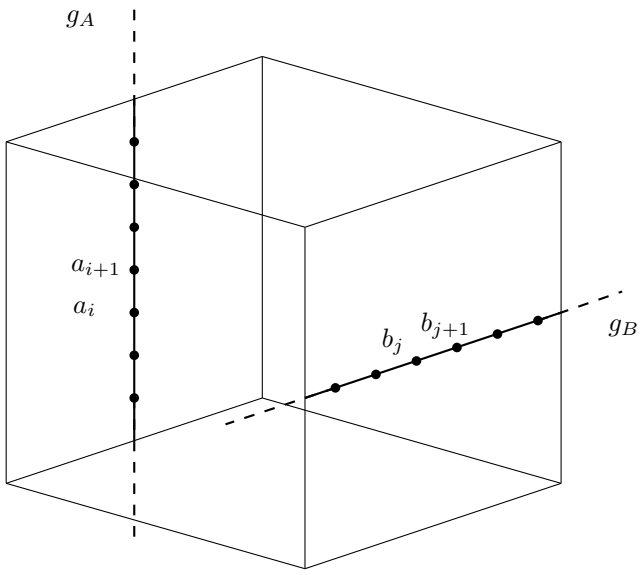


Abbildung 4: Beispiel einer Konstruktion einer Menge von m Punkten in $\mathbb{R}^3$ mit mindestens $\Omega(n^2)$ vielen Voronoi-Knoten.

Allgemein, im $\mathbb{R}^d$ ist die Anzahl der Knoten des Voronoi-Diagramms von m Punkten in $\Theta(m^{\lceil \frac{d}{2} \rceil})$ im schlimmsten Fall. Die Komplexität von Voronoi-Diagrammen höherer Ordnung im $\mathbb{R}^d$ ist nicht genau bekannt. Es ist aber zu vermuten, dass diese noch höher ist, als für $k = 1$.

Aus diesem Grund werden in höheren Dimensionen die k nächsten Nachbarn nicht durch die explizite Berechnung und Vorverarbeitung des Voronoi-Diagramms bestimmt. Alternativ können alle Abstände zu der Menge S explizit berechnet werden, was eine lange Klassifizierungslaufzeit hat. Eine andere Möglichkeit ist es, die nächsten Nachbarn approximativ zu bestimmen. Damit werden wir uns in der nächsten Vorlesung beschäftigen.

Referenzen

- Understanding Machine Learning, Kapitel 19.
- Rolf Klein, Algorithmische Geometrie, Springer, 1996, (Kapitel 5).

AGML, Sommersemester 2020 Vorlesung 18 (6 Seiten)

Approximative Nächste Nachbarn

Anne Driemel Letzte Aktualisierung: 1. Juli 2020

In der letzten Vorlesung ging es um eine Klasse von Lernalgorithmen, die auf dem Prinzip der nächsten Nachbarn basiert. Zur Erinnerung, die grundlegende Variante dieser Lernalgorithms nutzt die folgende Hypothese $h_S : X \to \{+1, -1\}$ definiert für eine Trainingsmenge $S = \{(x_1, y_1), \ldots, (x_m, y_m)\} \in \mathbb{R}^d \times \{-1, +1\}$ durch

$$h_S(x) = y_i \qquad \text{mit} \qquad i = \operatorname*{arg\,min}_{1 \le i \le m} d(x, x_i),$$

wobei $d : X \times X \to \mathbb{R}_{\ge 0}$ eine Abstandsfunktion definiert. Wir bezeichnen x_i als den nächsten Nachbarn von x in S. Für den Euklidischen Abstand hatten wir das algorithmische Problem, den nächsten Nachbarn zu bestimmen, mithilfe der Voronoi-Diagramme analysiert. Das Voronoi-Diagramm beschreibt im Grunde die inverse Funktion der Hypothese h_S. Da die Komplexität eines Voronoi-Diagramms im schlimmsten Fall exponentiell mit der Dimension d wächst bieten sie leider keine effiziente algorithmische Lösung des Problems an. Es bleibt uns scheinbar nur die Möglichkeit, die Abstände zu allen m Elementen der Trainingsmenge zu berechnen, um die Funktion h_S an einem Punkt x zu evaluieren. Dies wird auch als *lineare Suche* bezeichnet, da die Laufzeit in $O(dm)$ ist.

Wir wollen heute eine approximative Variante dieses Problems betrachten. Das Ziel ist es, auf der Menge S eine Datenstruktur zu berechnen, welche eine effizientere Klassifizierung zulässt, also eine Klassifizierungslaufzeit besser als die der linearen Suche, wobei wir trotzdem noch dem Prinzip der nächsten Nachbarn treu bleiben wollen.

1 Lokalitätssensitive Funktionen

Definition 18.1. *Sei $\mathcal{F}$ eine Klasse von Funktionen der Form $h : X \to U$, wobei auf U eine Ordnungsrelation $\le$ definiert ist, und sei $d : X \times X \to \mathbb{R}_{\ge 0}$ eine Abstandsfunktion. $\mathcal{F}$ ist (r, R, α, β)-lokalitätssensitiv bezüglich der Funktion d, wenn für $x, y \in X$*

$$\mathbf{Pr}_{h \in \mathcal{F}}[h(x) = h(y)] > \alpha \quad \textit{falls} \quad d(x, y) < r \tag{1}$$

$$\mathbf{Pr}_{h \in \mathcal{F}}[h(x) = h(y)] < \beta \quad \textit{falls} \quad d(x, y) > R \tag{2}$$

Wir sagen, dass eine Klasse von Funktionen lokalitätssensitiv ist, wenn sie (r, R, α, β)-lokalitätssensitiv ist für ein $\alpha > 0$, ein $\beta < 1$ und $r, R > 0$ mit $r \le R$.

Idealerweise wollen wir, dass α möglichst groß ist, dass β möglichst klein ist und dass R/r möglichst nah bei 1 ist. Die Intuition dahinter ist, dass zwei Punkte, die nah beieinander liegen dann eine hohe Wahrscheinlichkeit haben, durch ein zufälliges h auf denselben Schlüssel abgebildet zu werden, während Punkte, die weit entfernt voneinander entfernt liegen eine niedrige Wahrscheinlichkeit haben, durch ein zufälliges h auf denselben Schlüssel abgebildet zu werden.

Lokalitätssensitive Funktionen erlauben es uns, bekannte Suchstrukturen, wie zum Beispiel Suchbäume, oder Hashing, auf das Nächste-Nachbarn-Problem in höheren Dimensionen anzuwenden. Sei D solch eine Suchstruktur. Wir können dann eine lokalitätssensitive Funktion h zufällig aus der Klasse $\mathcal{F}$ wählen und die Schlüssel $z_i = h(x_i)$ für jeden Punkt $(x_i, y_i) \in S$ aus der Trainingsmenge erzeugen. Die Datensätze (x_i, y_i) speichern wir dann mit dem zugehörigen Schlüssel in der Suchstruktur D. Um den nächsten Nachbarn eines Punktes $y \in X$ in S zu

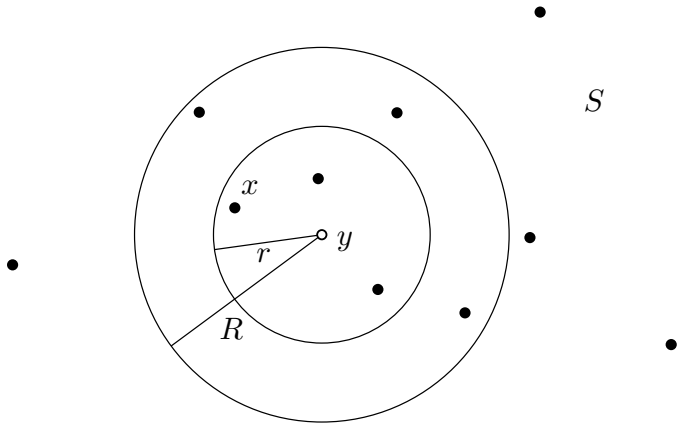


Abbildung 1: Schematische Darstellung der Unterteilung der Punktmenge S anhand der Abstände zu y. Für die Punkte, deren Abstand zwischen r und R liegt, wird in Definition 18.1 kein Aussage gemacht. Sie fallen in den Bereich des Approximationsfehlers.

finden, erzeugen wir den Schlüssel $z = h(y)$ und suchen in D nach z. Dafür ist es wichtig, dass auf der Schlüsselmenge eine Ordnungsrelation existiert, die es erlaubt, die Schlüssel zu sortieren. Beachte, dass die lokalitätssensitiven Funktionen hier nur eine Entscheidungsvariante des Nächste-Nachbarn-Problems lösen, da die Abstandsparameter r und R fest sind. Für den Euklidischen Abstand lässt sich dies durch Skalierung der Punktmenge auf alle anderen Abstandsparameter erweitern.

Insgesamt erinnern lokalitätssensitive Funktionen stark an das Hashing. Sie sollten aber nicht damit verwechselt werden. Beim Hashing geht es darum, ein Universum U auf eine kleine Indexmenge $\{1, \dots, m\}$ abzubilden. Das Ziel ist, einen Datensatz (Teilmenge des Universums) in einem Array der Größe m abzuspeichern und konstante Zugriffszeit auf die Elemente des Datensatzes zu erreichen. Eine sogenannte Hash-Kollision tritt dann auf, wenn zwei verschiedene Elemente im Datensatz auf denselben Schlüssel abgebildet werden. Um mehrere Elemente unter demselben Schlüssel zu speichern, können zusätzliche verkettete Listen verwendet werden. Beim Hashing ist $\mathcal{H}$ also eine Menge von Funktionen $h : U \to \{1, \dots, m\}$. Beim sogenannten uniformen Hashing gilt die folgende Annahme für jede zwei $x, y \in U$: $\mathbf{Pr}_{h \in \mathcal{F}}[h(x) = h(y)] = \frac{1}{m}$. Diese Annahme erlaubt es, die Auswirkungen von Hash-Kollisionen auf die Zugriffszeit zu beschränken. Hash-Kollisionen von nicht-identischen Elementen sollen vermieden werden, da sie die Zugriffszeit verlängern. Bei lokalitätssensitiven Funktionen hingegen sind Hash-Kollisionen sogar erwünscht, sofern sie vorrangig unter den nächsten Nachbarn auftreten. Oft werden beide miteinander kombiniert, indem man erst eine lokalitätssensitive Funktion anwendet und dann auf den so berechneten Schlüssel, eine Hashfunktion anwendet, um die Schlüsselmenge effizient speichern zu können und darin effizient suchen zu können. Wir ignorieren diesen Aspekt hier und beschränken uns auf die Analyse der lokalitätssensitiven Funktionen.

Definition 18.2. *Sei $X = \mathbb{R}$. Sei $\mathcal{F}$ die Klasse von Funktionen $h_\eta : X \to \mathbb{Z}$ mit $h_\eta(x) = \lceil x + \eta \rceil$, und mit $\eta \in [0, 1)$. Betrachte die Wahrscheinlichkeitsverteilung über $\mathcal{F}$, bei der η gleichverteilt im Intervall $[0, 1)$ gewählt wird.*

Lemma 18.3. *Die Klasse $\mathcal{F}$ aus Definition 18.2 ist lokalitätssensitiv bezüglich des Euklidischen Abstandes. Insbesondere gilt für jedes $x, y \in X$*

$$\mathbf{Pr}_{h_\eta \in \mathcal{F}}[h_\eta(x) = h_\eta(y)] = \max(0, 1 - |x - y|)$$

Beweis. Wenn $x = y$, dann ist die Wahrscheinlichkeit dass x und y auf denselben Funktionswert abgebildet werden gleich 1. Für $|x - y| \geq 1$ werden x und y immer auf unterschiedliche Funktionswerte abgebildet, egal welchen Wert η annimmt. Wir betrachten also den Fall $|x - y| < 1$.

Sei ohne Beschränkung der Allgemeinheit $x \leq y$. Die Werte x und y werden genau dann *nicht* auf denselben Funktionswert abgebildet, wenn in dem Intervall zwischen den Werten $(x + \eta)$ und $(y + \eta)$ eine ganze Zahl liegt. Insbesondere gilt

$$h_\eta(x) \neq h_\eta(y) \quad \Leftrightarrow \quad \lceil x + \eta \rceil \in [x + \eta, y + \eta)$$

Betrachten wir die Zufallsvariable τ definiert durch

$$\tau = \lceil x + \eta \rceil - (x + \eta)$$

Dann gilt nach obiger Betrachtung

$$h_\eta(x) \neq h_\eta(y) \quad \Leftrightarrow \quad \tau \in [0, y - x) \tag{3}$$

Welche Verteilung hat also die Zufallsvariable τ?

Eine wichtige Beobachtung ist, dass $\lceil x + \eta \rceil$ mit $\eta \in [0, 1)$ nur zwei verschiedene Werte annehmen kann, nämlich $\lceil x + \eta \rceil \in \{\lceil x \rceil, \lceil x + 1 \rceil\}$.

Wir betrachten eine weitere Zufallsvariable $\tau' = x + \eta$ in den zwei Fällen.

(Fall 1) $\lceil \tau' \rceil = \lceil x \rceil \quad \Rightarrow \quad \tau' \in [x, \lceil x \rceil]$

(Fall 2) $\lceil \tau' \rceil = \lceil x + 1 \rceil \quad \Rightarrow \quad \tau' \in (\lceil x \rceil, x + 1)$

Daraus ergibt sich für $\tau = \lceil \tau' \rceil - \tau'$

(Fall 1) $\tau = \lceil x \rceil - \tau'$

(Fall 2) $\tau = \lceil x + 1 \rceil - \tau'$.

Es ergeben sich die folgenden Intervalle für Werte von τ in den beiden Fällen.

(Fall 1) $\tau \in [\lceil \lceil x \rceil \rceil - \lceil x \rceil, \lceil x \rceil - x] = [0, \lceil x \rceil - x]$

(Fall 2) $\tau \in (\lceil x + 1 \rceil - (x + 1), \lceil x + 1 \rceil - \lceil x \rceil) = (\lceil x \rceil - x, 1)$

Da η in $[0, 1)$ gleichverteilt ist und daher τ' in $[x, x + 1)$ gleichverteilt ist, schließen wir daraus, dass $\tau \in [0, 1)$ gleichverteilt ist. Nun folgt aus (3), dass wenn $|x - y| < 1$ ist,

$$\mathbf{Pr}_{h_\eta \in \mathcal{F}} [h_\eta(x) \neq h_\eta(y)] = |x - y|$$

Daraus folgt

$$\mathbf{Pr}_{h_\eta \in \mathcal{F}} [h_\eta(x) = h_\eta(y)] = \max(0, 1 - |x - y|)$$

Sei $t \in (0, 1)$ ein Parameter. Es folgt nun, dass die Klasse $\mathcal{F}$ aus Definition 18.2 (r, R, α, β)-lokalitätssensitiv ist mit $\alpha = \beta = 1 - t$ und $r = R = t$. □

Wir wollen die Definition auf höhere Dimensionen erweitern. Dafür wählen wir zufällig eine Gerade durch den Ursprung und projizieren die Punkte auf den eindimensionalen Unterraum und wenden die Funktion aus Definition 18.2 auf den Unterraum an. Das geht am einfachsten indem man einen Einheitsvektor zufällig gleichverteilt auf dem Einheitskreis wählt. Der Einheitskreis $\mathbb{S}^1$ ist die Menge der Einheitsvektoren in $\mathbb{R}^2$. Formal, ist $\mathbb{S}^1 = \{x \in \mathbb{R}^2 \mid \|x\| = 1\}$ definiert. Wir können einen Vektor u zufällig gleichverteilt aus $\mathbb{S}^1$ auswählen indem wir einen Winkel ϕ zufällig gleichverteilt im Interval $[0, 2\pi)$ auswählen und $u = (\cos\phi, \sin\phi)$ definieren.

Definition 18.4. *Sei $X = \mathbb{R}^2$. Sei $\mathcal{F}$ die Klasse von Funktionen $h_{u,\eta} : X \to \mathbb{Z}$ mit $h_{u,\eta}(x) = \lceil \langle x, u \rangle + \eta \rceil$, und mit $\eta \in [0, 1)$ und $u \in \mathbb{S}^1$. Betrachte die Wahrscheinlichkeitsverteilung über $\mathcal{F}$, bei der η gleichverteilt im Intervall $[0, 1)$ gewählt wird und $u = (\cos(\phi), \sin(\phi))$, wobei ϕ gleichverteilt im Intervall $[0, 2\pi)$ gewählt wird.*

Lemma 18.5. *Die Klasse $\mathcal{F}$ aus Definition 18.4 ist (r, R, α, β)-lokalitätssensitiv bezüglich des Euklidischen Abstandes mit $r = \frac{1}{2}$,$R = 2$,$\alpha = \frac{1}{2}$,$\beta = \frac{1}{3}$.*

Beweis. Zunächst stellen wir fest, dass

$$| \langle x, u \rangle - \langle y, u \rangle | = | \langle x - y, u \rangle | = \|x - y\| \cdot \|u\| \cdot | \cos\theta | = \|x - y\| \cdot | \cos\theta |,$$

wobei wir mit θ den Winkel zwischen den Vektoren u und $(x - y)$ bezeichnen.

Wir betrachten beide Fälle aus der Definition der lokalitätssensitiven Funktionen. Sei $0 \leq \|x - y\| < \frac{1}{2}$. In diesem Fall, gilt für die Wahrscheinlichkeit, dass x und y auf verschiedene Funktionswerte abgebildet werden

$$\mathbf{Pr}_{h_{u,\eta} \in \mathcal{F}} [h_{u,\eta}(x) \neq h_{u,\eta}(y)] = | \langle x, u \rangle - \langle y, u \rangle | = \|x - y\| \cdot | \cos\theta | < \frac{1}{2}$$

Also ist

$$\mathbf{Pr}_{h_{u,\eta} \in \mathcal{F}} [h_{u,\eta}(x) = h_{u,\eta}(y)] > \frac{1}{2}$$

Im zweiten Fall betrachten wir $\|x - y\| > 2$. Im Ereignis, dass x und y auf denselben Funktionswert abgebildet werden, muss gelten

$$| \langle x, u \rangle - \langle x, u \rangle | < 1$$

Wir setzen ein und formen um und erhalten

$$1 > \|x - y\| | \cos\theta | > 2 | \cos\theta |$$

Daraus folgern wir, dass $| \cos\theta | < \frac{1}{2}$ gelten muss, in dem Ereignis, dass x und y auf denselben Funktionswert abgebildet werden. Der Vektor $(x - y)$ ist fest und unabhängig von der Wahl der lokalitätssensitiven Funktion $h_{\eta,u}$ mit $u = (\cos\phi, \sin\phi)$. Insbesondere muss der Winkel θ gleichverteilt in $[0, \pi)$ sein, da ϕ gleichverteilt in $[0, 2\pi)$ ist. Also ist

$$\mathbf{Pr}_{h_{u,\eta} \in \mathcal{F}} [h_{u,\eta}(x) = h_{u,\eta}(y)] \leq \mathbf{Pr} \left[| \cos\theta | \leq \frac{1}{2} \right] = \mathbf{Pr} \left[\theta \in \left[\frac{\pi}{3}, \frac{2\pi}{3} \right] \right] = \frac{1}{3}$$

□

Die Analyse aus obigem Beweis funktioniert unter der Annahme, dass $X = \mathbb{R}^2$. Tatsächlich kann man aber zeigen, dass eine ähnliche Klasse von Funktionen auch in höheren Dimensionen lokalitätssensitiv bezüglich des Euklidischen Abstandes ist.

2 Verstärkung durch Komposition

In den obigen Funktionsklassen für den Euklidischen Abstand ist die Erfolgswahrscheinlichkeit für zwei Punkte, die nah beieinander liegen, auf denselben Funktionswert abgebildet zu werden noch nicht hoch genug für praktische Anwendungen. Es ist daher sinnvoll, die Wahrscheinlichkeiten zu verstärken indem man eine Komposition von mehreren Funktionen benutzt.

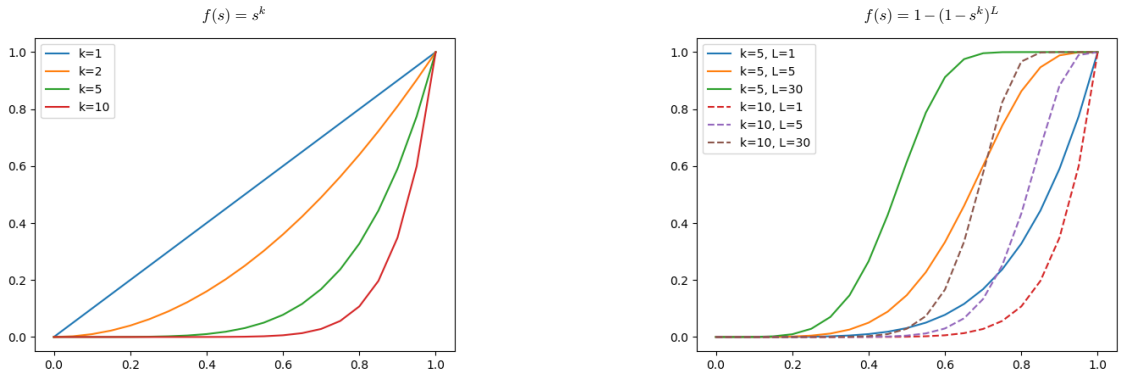


Abbildung 2: Wahrscheinlichkeit für Hashkollisionen für feste $x, y \in X$ bei Komposition, in Abhängigkeit von $s = \mathbf{Pr}_{h \in \mathcal{F}}[h(x) = h(y)]$. Links: k-fache UND-Komposition; Rechts: k-fache UND-Komposition gefolgt von L-facher ODER-Komposition.

Sei k eine natürliche Zahl. Sei $\mathcal{F}$ eine Klasse von (r, R, α, β)-lokalitätssensitiven Funktionen. Eine k-fache UND-Komposition ist eine Funktion $g : X \to U^k$ definiert durch $g(x) = (h_1(x), \ldots, h_k(x))$ mit $h_1, \ldots, h_k \in \mathcal{F}$. Beachte, dass auf U^k eine Ordnungsrelation existiert, sofern auf U eine Ordnungsrelation existiert, zum Beispiel können wir die lexikographische Ordnung annehmen. Dies wird eine UND-Komposition genannt, da $g(x) = g(y)$ voraussetzt, dass $h_i(x) = h_i(x)$ für alle $1 \leq i \leq k$. Wir bezeichnen die resultierende Klasse von Funktionen mit $\mathcal{F}^k$.

Zusätzlich können wir eine ODER-Komposition betrachten. Dies ist eine Komposition der resultierenden Datenstrukturen. Sei L eine natürliche Zahl. Seien $g_1, \ldots, g_L$ zufällig aus $\mathcal{F}^k$ gewählt. Wir berechnen für jede Funktion g_i den Schlüssel $g_i(x)$ für jedes $x \in S$ der Trainingsmenge und fügen $g_i(x)$ in eine separate Datenstruktur D_i ein. Bei einer Anfrage mit einem Element $y \in X$ berechnen wir den Schlüssel $g_i(y)$ und suchen mit diesem Schlüssel in den Datenstrukturen $D_1, \ldots, D_L$. Die Suche ist erfolgreich, wenn wir ein $x \in S$ finden, mit $d(x, y) \leq R$. Angenommen es existiert ein $x \in S$ mit $d(x, y) < r$. Was ist dann die Wahrscheinlichkeit, dass $g_i(x) = g_i(y)$ für mindestens eines der $i \in \{1, \ldots, L\}$?

Lemma 18.6. *Seien $k, L \in \mathbb{N}$. Sei $\mathcal{F}$ eine Klasse von lokalitätssensitiven Funktionen auf einer Grundmenge X. Sei $(g_1, \ldots, g_L)$ eine k-fache UND-Komposition gefolgt von einer L-fachen ODER-Komposition mit $k \cdot L$ Funktionen unabhängig zufällig gewählt aus $\mathcal{F}$. Dann gilt für jedes $x, y \in X$*

$$\mathbf{Pr}\left[\exists i \in \{1, \ldots, L\} : g_i(x) = g_i(y)\right] = 1 - (1 - (\mathbf{Pr}_{h \in \mathcal{F}}[h(x) = h(y)])^k)^L$$

Beweis. Sei $x, y \in X$ fest und sei $s = \mathbf{Pr}_{h \in \mathcal{F}}[h(x) = h(y)]$. Sei $i \in \{1, \ldots, L\}$ fest. Die Wahrscheinlichkeit, dass $g_i(x) = g_i(y)$ ist s^k, da die Funktionswerte von x und y für alle k Funktionen gleich sein müssen. Betrachten wir nun das Ereignis, dass $g_i(x) \neq g_i(y)$ für alle $i \in \{1, \ldots, L\}$. Die Wahrscheinlichkeit dafür ist $(1 - s^k)^L$. Die Wahrscheinlichkeit im Satz ist die Gegenwahrscheinlichkeit dazu. □

Beispiel 18.7. *Betrachten wir die Klasse von Funktionen aus Definition 18.2 für den Euklidischen Abstand in $\mathbb{R}$. Seien $x, y \in \mathbb{R}$ fest und sei $s = \max(0, 1 - |x - y|)$. Laut Lemma 18.3 ist s ist die Wahrscheinlichkeit, dass x und y auf denselben Funktionswert abgebildet werden. Laut Lemma 18.6 ist die Wahrscheinlichkeit, dass $g_i(x) = g_i(y)$ für mindestens eines der*

$i \in \{1, \ldots, L\}$ gleich $1 - (1 - s^k)^L$. Abbildung 2 zeigt Beispiele von Funktionengraphen dieser Funktion für verschiedene Werte von k und L.

Referenzen

- Jeff M. Phillips, Mathematical Foundations of Data Science, Kapitel 4.6, `http://www.cs.utah.edu/~jeffp/M4D/M4D-v0.6.pdf`
- Sariel Har-Peled, Geometric Approximation Algorithms, Springer, Kapitel 15.2, `https://sarielhp.org/book/` (Preprint)

AGML, Sommersemester 2020 Vorlesung 19 (5 Seiten)

Zentrumsbasiertes Clustering

Anne Driemel Letzte Aktualisierung: 2. Juli 2020

Bis jetzt haben wir uns in der Vorlesung mit dem Lernen auf Basis von beschrifteten Trainingsmengen $S \subseteq X \times \{+1, -1\}$ befasst. Dies wird auch als *überwachtes* Lernen bezeichnet, da die Labels der Punkte der Trainingsmenge bekannt sind. Heute werden wir uns mit Algorithmen zum *unüberwachten* Lernen befassen. Einem unüberwachten Lernalgorithmus ist eine Menge $S \subseteq X$, also eine Teilmenge der Grundmenge gegeben, ohne Labels. Das Ziel ist es, die Menge S in Gruppen aufzuteilen, sodass, innerhalb jeder einzelnen Gruppe, die Punkte möglichst ähnlich zueinander sind. Solch eine Aufteilung in Gruppen wird als *Clustering* bezeichnet. Die einzelnen Gruppen der Aufteilung werden *Cluster* genannt. Die Ähnlichkeit können wir wieder mithilfe einer Abstandsfunktion formalisieren.

Definition 19.1. *Sei X eine Menge und sei $d : X \times X \to \mathbb{R}_{\geq 0}$ eine Abstandsfunktion. Wir bezeichnen d als eine Metrik auf X, wenn sie folgende Eigenschaften erfüllt:*

(i) $\forall x, y \in X : d(x, y) = 0 \Leftrightarrow x = y$

(ii) $\forall x, y \in X : d(x, y) = d(y, x)$

(iii) $\forall x, y, z \in X : d(x, z) \leq d(x, y) + d(y, z)$ *(Dreiecksungleichung)*

Die Qualität des Clusterings wird mithilfe einer Zielfunktion definiert. Wir betrachten heute zwei verschiedene Zielfunktionen, die k-Center-Zielfunktion und die k-Means-Zielfunktion. Beide Zielfunktionen sind zentrumsbasiert, sie betrachten die Abstände des Punkte eines Clusters zum Zentrum des Clusters.

1 Gonzales Algorithmus

Sei X eine Grundmenge und sei d eine Metrik auf X. Das k-Center Problem ist, für eine gegebene Menge $S = \{x_1, \ldots, x_m\} \subseteq X$, und einen Parameter $k \in \mathbb{N}$ mit $k \leq m$, eine Menge von Zentren $c_1, \ldots, c_k \in X$ zu berechnen, welche die Zielfunktion

$$\phi_{\text{center}}(c_1, \ldots, c_k) = \max_{1 \leq i \leq m} \min_{1 \leq j \leq k} d(x_i, c_j)$$

minimiert.

Eine Lösung eines zentrumsbasierten Clusteringproblems ist durch eine feste Menge von k Zentren hinreichend definiert. Jeder Eingabepunkt wird seinem nächsten Nachbarn in der Menge der Zentren zugewiesen. Somit ergibt sich das Clustering als die Aufteilung der Eingabemenge, in der jedem Zentrum eine Teilmenge der Eingabemenge zugewiesen ist. Beim k-Center-Clustering bezeichnen wir den Wert der Funktion $\phi_{\text{center}}(C)$ für eine Lösung C als den *Radius* des Clusterings. Sei $C = \{c_1, \ldots, c_k\}$ eine Lösung für Eingabemenge S, und sei $\phi_{\text{center}}(C) = r$ der Radius. Die Eingabemenge ist enthalten in der Vereinigung der metrischen Kugeln

$$\bigcup_{1 \leq j \leq k} \{x \in X \mid d(x, c_j) \leq r\}$$

Abbildung 1 zeigt ein Beispiel für den Euklidischen Abstand.

Gonzales Algorithmus berechnet iterativ eine Menge von Zentren. Im ersten Schritt wird ein beliebiges Element der Eingabemenge als das erste Zentrum ausgewählt. In jedem weiteren Schritt wird ein Eingabepunkt als nächstes Zentrum ausgewählt, bei dem das Maximum in

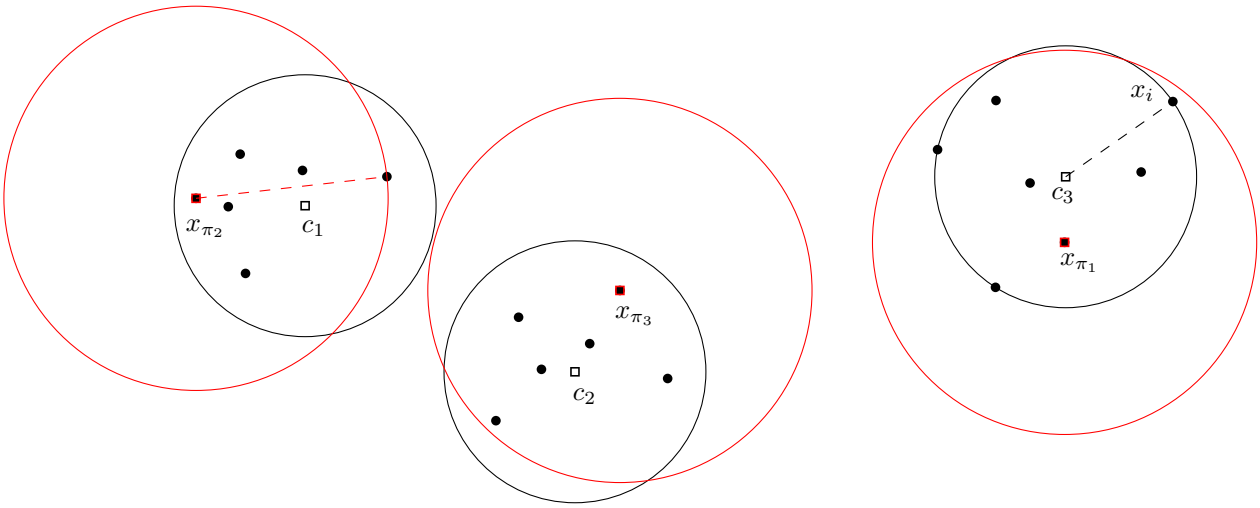


Abbildung 1: Beispiel eines k-Center-Clusterings mit $k = 3$ für eine Menge $S \subseteq \mathbb{R}^2$ mit der Euklidischen Abstandsfunktion. Die gestrichelte Linie zeigt das Paar $c_i \in C$, $x_j \in S$ welches den Radius des Clusterings c_1, c_2, c_3 realisiert. Die Abbildung zeigt ausserdem eine vom Gonzales-Algorithmus berechnete Lösung in rot mit dem entsprechenden Radius als rote gestrichelte Linie.

der Zielfunktion angenommen wird. Die Vereinigung der so gewählten Zentren wird als Lösung zurückgegeben.

```
Gonzales-Algorithmus(S = {x_1, ..., x_m}, k)
  1. π_1 = 1
  2. d_1, ..., d_m = ∞
  3. for i in 1...k do
  4.       for j in 1...m do
  5.             //d_j speichert den Abstand von x_j zu den bisher gewählten Zentren
  6.             d_j = min(d_j, d(x_j, x_{π_i}))
  7.       r_i = max_{1≤j≤m} d_j
  8.       π_{i+1} = arg max_{1≤j≤m} d_j
  9. Return {x_{π_1}, ..., x_{π_k}}
```

Das k-Center-Problem ist NP-schwer, selbst wenn die gewählte Metrik Euklidisch ist und die Punkte in einer Ebene liegen. Die Laufzeit des Algorithmus von Gonzales ist in $O(km)$, was sich leicht überprüfen lässt. Wir nehmen dabei an, dass sich die Abstandsfunktion in konstanter Zeit evaluieren lässt. Der Gonzales-Algorithmus kann also nicht immer eine optimale Lösung berechnen, sofern $P \neq NP$. Abbildung 1 zeigt ein Beispiel für eine Lösung, die vom Algorithmus berechnet wird und nicht optimal ist. Wir werden nun analysieren, wie gut die berechnete Lösung im schlimmsten Fall ist, wir vergleichen sie dabei mit der optimalen Lösung.

Satz 19.2. *Sei $k \in \mathbb{N}$. Sei $C^* = \{c_1, \ldots, c_k\}$ eine optimale Lösung des k-Center-Problems für eine Menge $S = \{x_1, \ldots, x_m\}$ mit $m \geq k$. Seien $\pi_1, \ldots, \pi_{k+1}$, sowie $r_1, \ldots, r_k$ wie vom Gonzales-Algorithmus berechnet. Dann ist*

$$r_k = \phi_{\text{center}}(x_{\pi_1}, \ldots, x_{\pi_k}) \leq 2\phi_{\text{center}}(C^*)$$

Beweis. Zunächst beobachten wir, dass die Variable d_j nach dem $i$ten Durchlauf der äußeren Schleife den kleinsten Abstand von x_j zu den gewählten Zentren $x_{\pi_1}, \ldots, x_{\pi_i}$ speichert. Danach

wird r_i als der maximale Wert der Abstände $d_1, \ldots, d_m$ gesetzt. Das ist genau der Wert der Zielfunktion $\phi_{\text{center}}(x_{\pi_1}, \ldots, x_{\pi_i})$. Damit ist der erste Teil der Behauptung bewiesen.

Daraus folgt auch, dass

$$r_1 \geq r_2 \geq \cdots \geq r_k \tag{1}$$

Sei $k = 1$. Für jeden Punkt x_i gilt laut der Dreiecksungleichung

$$d(x_i, x_1) \leq d(x_i, c_1) + d(c_1, x_1) \leq 2\phi_{\text{center}}(C^*)$$

Also gilt dies auch für den Punkt, der den Abstand zu x_1 maximiert. Das heisst,

$$r_1 = \max_{1 \leq i \leq m} d(x_i, x_1) \leq 2\phi_{\text{center}}(C^*)$$

Also ist der Satz für $k = 1$ bewiesen.

Wir wollen nun das Argument mit der Dreiecksungleichung auf $k \geq 1$ erweitern. Wir betrachten dazu die Menge der gewählten Zentren $x_{\pi_1}, \ldots, x_{\pi_k}$ und zusätzlich den Punkt $x_{\pi_{k+1}}$. Aus dem Schubfachprinzip folgt, dass von diesen $k + 1$ Punkten mindestens zwei Punkte in demselben Cluster im optimalen Clustering C^* zugewiesen sind. Sei c_ℓ das Zentrum dieses Clusters und seien x_{π_i} und x_{π_j} die zwei Punkte in dem Cluster. Jetzt können wir wieder die Dreiecksungleichung auf diese drei Punkte anwenden.

$$d(x_{\pi_i}, x_{\pi_j}) \leq d(x_{\pi_i}, c_\ell) + d(c_\ell, x_{\pi_j}) \leq 2\phi_{\text{center}}(C^*)$$

Sei ohne Beschränkung der Allgemeinheit $i < j$. Beachte, dass in Zeile 7, π_j als der Punkt gewählt wird, der den Radius der Lösung der bisher gewählten Zentren $x_{\pi_1}, \ldots, x_{\pi_{j-1}}$ realisiert. Insbesondere ist

$$r_{j-1} = \min_{1 \leq i \leq j-1} d(x_{\pi_j}, x_{\pi_i}) \leq d(x_{\pi_i}, x_{\pi_j})$$

Zusammen mit (1) ergibt sich, dass

$$r_k \leq r_{j-1} \leq d(x_{\pi_i}, x_{\pi_j}) \leq 2\phi_{\text{center}}(C^*)$$

□

2 Lloyds Algorithmus

Im nächsten Abschnitt betrachten wir die Grundmenge $\mathbb{R}^d$ mit der Euklidischen Metrik.

Definition 19.3. *Das k-Means-Problem ist, gegeben eine Menge $S = \{x_1, \ldots, x_m\} \subseteq \mathbb{R}^d$, und ein Parameter $k \in \mathbb{N}$ mit $k \leq m$, berechne eine Menge $c_1, \ldots, c_k \in \mathbb{R}^d$, welche die Funktion*

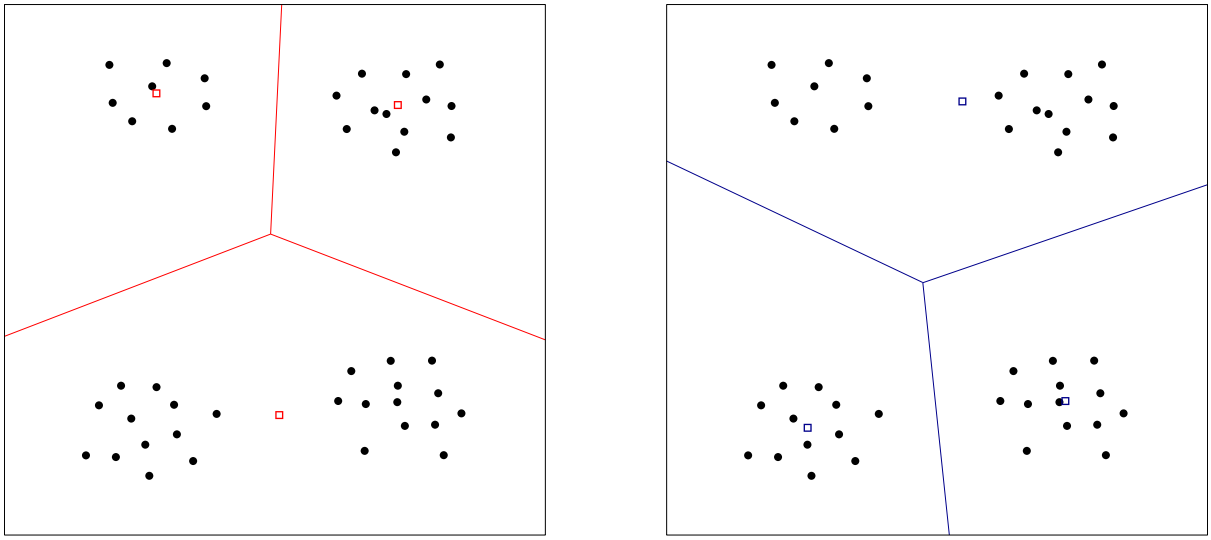
$$\phi_{\text{mean}}(c_1, \ldots, c_k) = \sum_{i=1}^{m} \min_{1 \leq j \leq k} \|x_i - c_j\|^2$$

minimiert.

Lloyds Algorithmus besteht aus zwei Schritten, die immer wieder abwechselnd in einer Schleife ausgeführt werden, bis sich das Clustering nicht mehr ändert (im Pseudocode unten durch die Boolsche Variable b ausgedrückt). Die zwei Schritte können wie folgt beschrieben werden. Der erste Schritt berechnet eine optimale Zuweisung der Eingabepunkte zu einer festen Menge von Zentren. Der zweite Schritt berechnet für jeden Cluster einer festen Clusterzuweisung ein

optimales Zentrum. Die Clusterzuweisung kann in einem m-dimensionalen Array gespeichert werden. Wir speichern an der $i$ten Stelle den Index des Clusters, dem der Punkt x_i zugewiesen ist.

```
Lloyds-Algorithmus(S = {x_1, ..., x_m}, k)
 1. Wähle c_1, ..., c_k zufällig aus der Menge S
 2. Initialisiere S_1 = S und S_2, ..., S_k mit ∅
 3. repeat
 4.     b = false
 5.     // Schritt 1: Berechne optimale Clusterzuweisung zu c_1, ..., c_k
 6.     for i in 1...m do
 7.         j = arg min_{1≤j≤k} d(x_i, c_j)
 8.         if x_i ∉ S_j then
 9.             Wechsle die Clusterzuweisung von x_i zum Cluster S_j
10.             b = true
11.     // Schritt 2: Berechne optimale Clusterzentren für S_1, ..., S_k
12.     for j in 1...k do
13.         c_j = 1/|S_j| Σ_{x_i∈S_j} x_i
14. until b = false
```



Auch das k-Means-Problem ist NP-schwer und Lloyds Algorithmus berechnet nicht immer eine optimale Lösung. Insbesondere kann der Algorithmus in einem lokalen Minimum der Zielfunktion konvergieren. Das obige Bild zeigt zwei verschiedene Lösungen, die auf derselben Punktmenge von Lloyds Algorithmus berechnet wurden. In beiden Fällen ergibt weder die Clusterzuweisung, noch die Zentrenberechnung eine neue Lösung, also terminiert der Algorithmus.

Ob der Algorithmus in einem lokalen Minimum terminiert, hängt stark von der Initialisierung ab. Daher wird der Algorithmus oft mehrmals ausgeführt, wobei die zufällige Initialisierung mit jeder Ausführung neu gewählt wird. Der folgende randomisierte Algorithmus wird zur Initialisierung des Algorithmus von Lloyd benutzt.

`k-Means++-Algorithmus`$(S = \{x_1, \ldots, x_m\}, k)$

1. Wähle π_1 zufällig gleichverteilt in $\{1, \ldots, m\}$
2. $d_1, \ldots, d_m = \infty$
3. **for** i **in** $1 \ldots k$ **do**
4. **for** j **in** $1 \ldots m$ **do**
5. $d_j = \min(d_j, d(x_j, x_{\pi_i}))$
6. Wähle π_{i+1} zufällig aus der Menge $\{1, \ldots, m\}$ mit der folgenden Verteilung:

   Der Index j wird mit Wahrscheinlichkeit $p_j = \frac{d_j}{\sum_{\ell=1}^{m} d_\ell}$ ausgewählt
7. Return $\{x_{\pi_1}, \ldots, x_{\pi_k}\}$

Ähnlich wie beim Gonzales-Algorithmus werden die Zentren iterativ berechnet. In jedem Schritt wird ein Zentrum zufällig ausgewählt, nach einer Verteilung die sich aus Abständen der Eingabepunkte zu den bisher gewählten Zentren berechnet. Ein Punkt hat eine hohe Wahrscheinlichkeit gewählt zu werden, wenn er verhältnismäßig weit weg von dem Zentrum liegt, das ihm am nächsten ist. Beim Gonzales-Algorithmus wurde der Punkt gewählt, der diese Wahrscheinlichkeit maximiert.

Referenzen

- Sariel Har-Peled, Geometric Approximation Algorithms, Springer, Kapitel 4.2, `https://sarielhp.org/book/` (Preprint)
- Jeff M. Phillips, Mathematical Foundations of Data Science, Kapitel 8, `http://www.cs.utah.edu/~jeffp/M4D/M4D-v0.6.pdf`
- Understanding Machine Learning, Kapitel 22.2

AGML, Sommersemester 2020 Vorlesung 20 (6 Seiten)

Hierarchisches Clustering

Anne Driemel Letzte Aktualisierung: 7. Juli 2020

In der letzten Vorlesung haben wir zwei Beispiele des zentrumsbasierten Clusterings kennengelernt. Beim zentrumsbasierten Clustering messen wir die Ähnlichkeit der Element innerhalb eines Clusters mithilfe eines Zentrums, der diesem Cluster zugewiesen ist. Eine weitere Charakteristik des zentrumsbasierten Clusterings ist die Festlegung auf die Anzahl der Cluster mithilfe eines Parameters k. Die Wahl des Parameters hat einen Einfluss auf das berechnete Clustering, was in der Praxis nicht immer gewünscht ist.

Dies wollen wir heute umgehen, indem wir nach hierarchischen Strukturen in der Eingabemenge suchen. Dabei suchen wir nach einer Hierarchie von Partitionierungen (Clusterings) der Eingabemenge, welche ineinander geschachtelt sind. Das heisst, ein Cluster (Teilmenge der Eingabemenge) in einer festen Ebene der Hierarchie sollte in einer höheren Ebene der Hierarchie nicht wieder geteilt werden, sondern dies sollte nur in tieferen Ebenen der Hierarchie passieren.

Beim zentrumsbasierten Clustering wird die Qualität des Clusterings mithilfe einer Zielfunktion beschrieben, welche vom Algorithmus minimiert werden soll. Ähnlich müssen wir nun mathematisch definieren, was die Qualität eines hierarchischen Clusterings ausmacht. Wir definieren dafür drei Eigenschaften, die gegeben sein müssen.

1 Definition

Sei X eine Grundmenge und sei $d(\cdot,\cdot)$ eine Metrik auf X. Ein hierarchisches Clustering einer n-elementigen Menge $S \subseteq X$ ist eine geordnete Menge $\mathcal{C} = \{C_1, \ldots, C_n\}$ mit den folgenden Eigenschaften:

(i) (Jede Menge ist eine Partitionierung von S)
Für alle $1 \leq i \leq n$ hat C_i die Form $\{A_1, \ldots, A_{n-i+1}\}$ mit

  (a) $\bigcup_{1 \leq j \leq n-i+1} A_j = S$ und
  (b) für $j \neq k$ ist $A_j \cap A_k = \emptyset$,

(ii) (Die Mengen sind hierarchisch geschachtelt)
$C_n = \{S\}$ und für alle $1 \leq i < n$ und $A \in C_i$ existiert ein $B \in C_{i+1}$ mit $A \subseteq B$

(iii) (Die Mengen sind optimal)
Für alle $1 \leq i \leq n$ maximiert C_i die folgende Zielfunktion

$$\phi(C_i) = \min_{A,B \in C_i} \min_{a \in A, b \in B} d(a,b)$$

über alle möglichen Partitionierungen von S die aus genau $|C_i|$ Teilmengen von S bestehen.

Die Inklusionsrelationen des hierarchisches Clusterings werden oft als Baum dargestellt. Dafür definieren wir einen Graphen $G = (V, E)$, mit Knotenmenge V und Kantenmenge $E \subseteq V \times V$. Die Knoten des Graphen sind wie folgt definiert. Für jede Teilmenge von S die in einer der Mengen C_i existiert, existiert ein Knoten in V. Der Knoten ist mit dieser Teilmenge eindeutig assoziiert. Eine Kante existiert zwischen zwei Knoten $A \in C_i$ und $B \in C_{i+1}$ wenn $A \subset B$. Aus den oben definierten Eigenschaften kann man ableiten, dass der Graph ein Baum ist. Wir legen die Wurzel des Baumes fest als die Menge S. Abbildung 1 zeigt ein Beispiel. Beachte dass das Clustering $C_1, \ldots, C_n$ nicht eindeutig dargestellt sind, sondern nur die Inklusionsrelationen zwischen den Mengen in $\bigcup_{1 \leq i \leq n} C_i$.

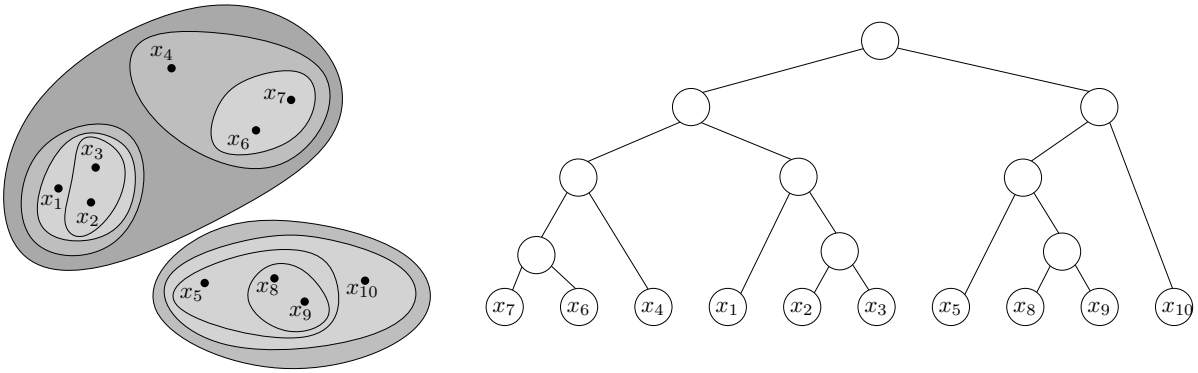


Abbildung 1: Beispiel eines hierarchischen Clusterings einer Menge $\{x_1, \ldots, x_{10}\}$ mit Baumdarstellung der Inklusionsrelationen. Das Clustering $C_1, \ldots, C_n$ lässt sich anhand der Baumdarstellung nicht eindeutig herleiten.

2 Algorithmus

Wir betrachten einen Greedy-Algorithmus, der die Partitionierungen $C_1, \ldots C_n$ iterativ berechnet. Er startet mit der Partitionierung in der jedes Element der Eingabemenge eine einzelne Menge darstellt. In jedem Schritt werden die zwei Mengen vereinigt, die den kleinsten Abstand haben, wobei der Abstand zwischen zwei Mengen A und B definiert ist als der kleinste Abstand, zwischen zwei Elementen $a \in A$ und $b \in B$. Wegen dieser Definition des Abstands wird diese Art von Clustering auch *Single-Link-Clustering* genannt. Wir besprechen zunächst die Korrektheit bezüglich der drei Clustering-Eigenschaften. Danach besprechen wir, wie der Algorithmus effizient implementiert werden kann.

`Single-Link-Clustering`$(S = \{x_1, \ldots, x_n\})$
1. Initialisiere $C_1 = \{\{x_1\}, \ldots, \{x_n\}\}$
2. **for** i **in** $2 \ldots n$ **do**
3. Finde zwei Mengen $A, B \in C_i$ welche $\min_{a \in A, b \in B} d(a, b)$ minimieren.
4. Sei C_{i+1} dieselbe Menge wie C_i, nur dass A und B vereinigt sind
5. Return $C_1, \ldots, C_n$

Satz 20.1. *Die vom Algorithmus berechnete Menge $\mathcal{C} = \{C_1, \ldots, C_n\}$ erfüllt die Clustering-Bedingungen (i),(ii) und (iii).*

Beweis. Die Bedingungen (i) und (ii) lassen sich leicht durch Induktion beweisen. Wir konzentrieren uns uns auf den Beweis der dritten Bedingung. Wir behaupten, dass die folgende Schleifen-Invariante gilt:

Behauptung 20.2. *Sei $A \in C_i$ und seien $a, b \in A$. Dann existieren Elemente $c_1, \ldots c_\ell \in A$ für ein $\ell \in \mathbb{N}$ mit $a = c_1$, $b = c_\ell$ und $d(c_j, c_{j+1}) \leq \phi(C_i)$ für alle $1 \leq j < \ell$. Wir bezeichnen die geordnete Menge $\{c_1, \ldots, c_\ell\}$ als Pfad.*

Wir stellen den Beweis von Behauptung 20.2 hintan und kommen später darauf zurück.

Sei nun C' eine Partitionierung von S in $|C_i|$ Mengen, welche sich von C_i unterscheidet. Es müssen zwei Punkte $a, b \in S$ existieren, die in C' in unterschiedlichen Mengen liegen, aber in C_i in derselben Menge A liegen. Wenn dies nicht der Fall wäre, dann wären alle Mengen von C' Teilmengen von Mengen in C_i. Da beide Partitionierungen dieselbe Anzahl von Mengen enthalten und unterschiedlich sind kann das nicht sein.

Laut Behauptung 20.2 sind a und b in A durch einen Pfad verbunden, der nur kurze Kanten enthält. Wir betrachten den Pfad in C'. Es muss entlang des Pfades eine Kante zwischen zwei Punkten a' und b' geben, die in C' nicht in derselben Menge liegen. Es gilt

$$\phi(C') = \min_{A,B \in C'} \min_{a \in A, b \in B} d(a,b) \leq d(a',b') \leq \phi(C_i)$$

Daraus folgt, dass $\phi(C_i)$ optimal ist. Somit ist die dritte Bedingung unter Annahme von Behauptung 20.2 erfüllt. □

3 Exkurs: Minimale Spannbäume

Für die effiziente Berechnung der hierarchischen Clusterings betrachten wir zunächst minimale Spannbäume.

Definition 20.3 (Minimaler Spannbaum)**.** *Sei $G = (V, E)$ ein Graph mit Kantengewichten $w : E \to \mathbb{R}_{\geq 0}$. Ein Spannbaum von G ist ein zusammenhängender kreisfreier Graph $T = (V, E')$ mit $E' \subseteq E$. Ein minimaler Spannbaum ist ein Spannbaum der die Summe der Kantengewichte $\sum_{e \in E'} w(e)$ minimiert.*

Wir betrachten den Algorithmus von Kruskal zur Berechnung eines minimalen Spannbaumes. Dieser Algorithmus ist seiner Struktur nach dem obigen Algorithmus Single-Link-Clustering sehr ähnlich. Kruskal's Algorithmus berechnet die Kanten des minimalen Spannbaumes iterativ. Er sortiert die Kanten des Graphen aufsteigend nach ihrem Gewicht und bearbeitet sie in dieser Reihenfolge. Für jede Kante testet der Algorithmus ob diese einen Kreis in der bisher gewählten Kantenmenge erzeugen würde. Wenn dem nicht so ist, dann wird die Kante der aktuellen Menge hinzugefügt. Die so berechnete Kantenmenge wird als Ergebnis zurückgegeben.

`Kruskal-Algorithmus`$(G = (V, E), w)$
1. Sortiere die Kantenmenge E nach ihrem Gewicht
2. Seien $e_1, \ldots, e_m$, sodass $i \leq j \Leftrightarrow w(e_i) \leq w(e_j)$
3. Sei $E' = \emptyset$
4. **for** i **in** $1, \ldots, m$ **do**
5. **if** e_i erzeugt in (V, E') keinen Kreis
6. $E' = E' \cup e_i$
7. Return (V, E')

Um den Test, ob die Kante e_i einen Kreis in der Kantenmenge E' einen Kreis erzeugen würde, effizient ausführen zu können, speichert der Algorithmus zusätzlich die Zusammenhangskomponenten des Graphen (V, E'). Sei e_i eine Kante $(a, b) \in E$. Die Kante schliesst in (V, E') genau dann einen Kreis, wenn a und b in derselben Zusammenhangskomponente sind. Für die Zusammenhangskomponenten wird eine Datenstruktur zur Verwaltung von disjunkten Mengen benutzt, welche die folgenden Operationen unterstützt.

- Union(A, B) - Vereinigt die Mengen A und B zu einer neuen Menge und gibt diese zurück.
- Find(x) - Gibt die Menge zurück in der x enthalten ist.

Wir gehen auf diese Datenstruktur nicht weiter ein. Wichtig ist nur, dass Kruskals Algorithmus damit effizient implementiert werden kann. Die Laufzeit des Algorithmus von Kruskal ist in $O(|E| \log |E|)$.

Die folgende grundlegende Eigenschaft von minimalen Spannbäumen hilft dabei, zu zeigen, dass Kruskals Algorithmus einen minimalen Spannbaum berechnet.

Lemma 20.4 (Schnitt-Eigenschaft)**.** *Sei $G = (V, E)$ ein Graph und sei $U \subseteq E$ eine Teilmenge von Kanten eines minimalen Spannbaums $T = (V, E')$ von G. Sei $A \subseteq V$, sodass U keine Kante enthält, die einen Knoten von A mit einem Knoten von $V \setminus A$ verbindet. Sei F die Kantenmenge $\{(a, b) \in E \mid a \in A, b \in V \setminus A\}$. Sei $e = \arg\min_{e \in F} w(e)$. Dann ist $U \cup \{e\}$ Teilmenge eines minimalen Spannbaums von G.*

Beweis. Angenommen, $e \notin E'$. Da T zusammenhängend ist, muss es einen Pfad in T geben, der die Endpunkte von e verbindet. Dieser Pfad muss auch eine Kante in F haben. Sei e' solch eine Kante. Wir können e' durch e ersetzen und erhalten den Graphen $T' = (V, E' \cup \{e\} \setminus \{e'\})$. Wir können zeigen, dass T' auch ein Spannbaum ist, indem wir zeigen, dass T' zusammenhängend ist und $|V| - 1$ Kanten enthält. Die Summe der Kantengewichte von T' ist

$$\sum_{f \in E'} w(f) - w(e') + w(e) \leq \sum_{f \in E'} w(f)$$

□

Satz 20.5. *Kruskals Algorithmus berechnet einen minimalen Spannbaum von G.*

Beweis. Wir führen eine Induktion über die Kanten aus E' in der Reihenfolge, in der sie vom Algorithmus hinzugefügt werden. Für den Induktionsanfang betrachten wir die erste Kante die hinzugefügt wird. Sei $e = (a, b)$ diese Kante. Wir wenden Lemma 20.4 an mit $U = \emptyset$ und $A = \{a\}$. Daraus folgt, dass e in einem minimalen Spannbaum enthalten ist.

Nun folgt der Induktionsschritt. Sei e eine Kante, die von Kruskals Algorithmus in Zeile 6 zu der Kantenmenge E' hinzugefügt wird. Da e keinen Kreis schliesst, verbindet sie zwei Zusammenhangskomponenten in E', seien diese A und B. Da der Algorithmus die Kanten in der aufsteigenden Reihenfolge ihrer Gewichte betrachtet hat e minimales Gewicht unter den Kanten zwischen A und $V \setminus A$. Laut Lemma 20.4 ist $E' \cup \{e\}$ Teil eines minimalen Spannbaums, sofern E' Teilmenge eines minimalen Spannbaums ist. Letzteres folgt aus der Induktionsbehauptung. □

4 Anwendung auf das hierarchische Clustering

Sei $G = (V, E)$ ein vollständiger Graph mit Knotenmenge $V = S$. Das Gewicht einer Kante ist der Abstand zwischen den entsprechenden Elementen in S, also $w(a, b) = d(a, b)$.[1] Betrachte den Kruskal-Algorithmus angewandt auf diesen Graphen und betrachte den Single-Link-Algorithmus angewandt auf S. Bevor wir den folgenden Satz zeigen, müssen wir die Algorithmen noch weiter spezifizieren. Die Anweisung, welche Mengen in Zeile 3 des Single-Link-Clusterings vereinigt werden sollen, ist unter Umständen nicht eindeutig, wenn es Paare von Elementen mit den gleichen Abständen gibt. Gleiches gilt für die Reihenfolge der Kanten in Zeile 2 von Kruskals Algorithmus. Wir nehmen hier einfach an, dass es eine gemeinsame Reihenfolge der Kanten gibt, die von Kruskal verwendet wird und die auch vom Single-Link-Clustering verwendet wird, um zu entscheiden, welcher Abstand am kleinsten ist.

Satz 20.6. *Sei $n = |S|$. Seien $e_1, \ldots e_{n-1}$ die Kanten aus E', in der Reihenfolge, in der sie von Kruskals Algorithmus der Menge E' hinzugefügt werden. Für $1 \leq j \leq n$, sei E_j die Kantenmenge $\{e_i \mid 1 \leq i < j\}$. Seien $C_1, \ldots, C_n$ die Mengen, die durch den Single-Link-Clustering Algorithmus berechnet werden. Dann gilt für alle $1 \leq j \leq n$:*

(i) C_j ist die Menge der Zusammenhangskomponenten des Graphen (V, E_j)

(ii) Für $j < n$ ist $\phi(C_j) = w(e_j)$

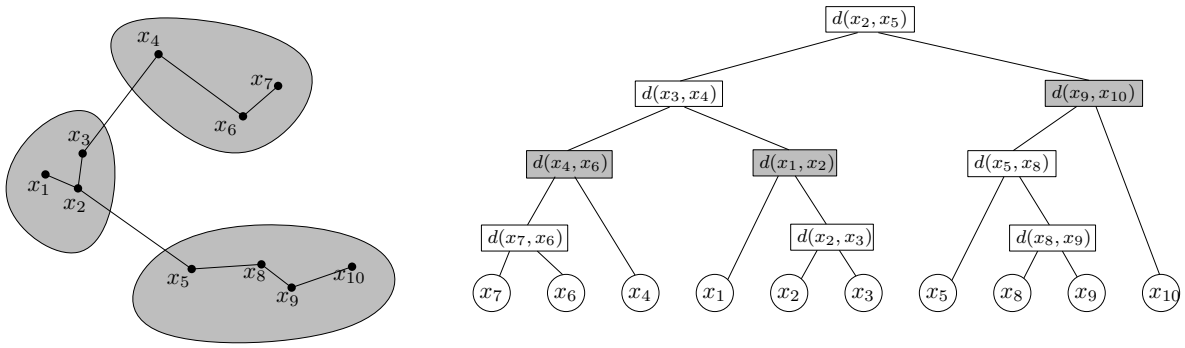


Abbildung 2: Links: Minimaler Spannbaum für die Punktmenge aus Abbildung 1 unter dem Euklidischen Abstand. Rechts: Baumdarstellung der Inklusionsrelationen. Die Abbildung zeigt ausserdem die Menge $C_i = \{\{x_1, x_2, x_3\}, \{x_4, x_6, x_7\}, \{x_5, x_8, x_9, x_{10}\}\}$. In der Baumdarstellung sind die entsprechenden Knoten markiert, deren Unterbäume die Mengen in C_i darstellen. Das Gewicht der Kante (x_3, x_4) bestimmt die Zielfunktion, das heisst $\phi(C_i) = d(x_3, x_4)$.

Beweis. Wir führen eine Induktion über j. Für $j = 1$ ist die Kantenmenge E_j die leere Menge. Die Menge der Zusammenhangskomponenten ist also die Menge der Knoten des Graphen G. Der Single-Link-Clustering Algorithmus definiert die Menge $C_1 = \{\{x_1\}, \ldots, \{x_n\}\}$. Weiter ist $\phi(C_i) = \min_{x_i, x_j \in S} d(x_i, x_j) = w(e_1)$, da die Kante mit kleinstem Gewicht von Kruskal als erstes zu E' hinzugefügt wird. Somit ist der Satz für $j = 1$ erfüllt.

Sei $n > j \geq 1$. Betrachte die Kantenmengen E_j und E_{j+1}. Aus der Induktionsannahme wissen wir, dass die Menge C_j die durch den Single-Link-Algorithmus berechnet wird, gleich den Zusammenhangskomponenten des Graphen (V, E_j) ist. Aus der Konstruktion von E_j und E_{j+1} im Satz ergibt sich $E_{j+1} = E_j \cup \{e_j\}$. Wir wollen zeigen, dass e_j genau die zwei Mengen A und B aus C_j verbindet, welche in C_{j+1} zusammenhängend als die vereinigte Menge $A \cup B$ vorkommen.

Betrachte die Vereinigung der Kanten

$$F_j = \bigcup_{A \in C_j} \{(a, b) \in E \mid a \in A, b \in V \setminus A\}$$

Die Kanten in F_j sind genau die Kanten von G, die in (V, E_j) keinen Kreis erzeugen. Unter den Kanten in F_j betrachtet der Algorithmus die Kante mit minimalem Gewicht zuerst. Da diese keinen Kreis schliesst, wird sie als nächstes hinzugefügt. Also ist,

$$e_j = \arg\min_{e \in F_j} w(e)$$

Da C_j eine Partitionierung von V ist, folgt

$$w(e_j) = \min_{A \in C_j} \min_{\substack{(a,b) \in E \\ a \in A, b \in V \setminus A}} w((a, b)) = \min_{A, B \in C_j} \min_{\substack{(a,b) \in E \\ a \in A, b \in B}} w((a, b))$$

Da jeder kürzeste Weg zwischen zwei Elementen in verschiedenen Zusammenhangskomponenten über den Schnitt gehen muss, folgt ausserdem

$$w(e_j) = \min_{A, B \in C_j} \min_{a \in A, b \in B} d(a, b)$$

[1] Alternativ kann die Abstandsfunktion auch direkt als die Kürzeste-Wege-Metrik in einem Graphen G mit Knotenmenge S gegeben sein. In diesem Fall betrachten wir G direkt.

Daraus folgt, dass die Kante e_j genau die zwei Mengen in C_j verbindet, die den kleinsten Single-Link-Abstand haben. Also ist $\phi(C_j) = w(e_j)$. □

Korollar 20.7. *Kruskals Algorithmus berechnet ein hierarchisches Clustering der Knotenmenge V bezüglich der Abstandsfunktion der kürzesten Wege im Graphen G. Die Laufzeit des Algorithmus ist in $O(|E| \log |E|)$. Die Mengen $C_1, \ldots, C_n$ lassen sich mithilfe der Kantengewichte der Kanten des minimalen Spannbaums eindeutig herleiten. Siehe Abbildung 2.*

Beweis von Behauptung 20.2. Der Beweis ist nun sehr einfach. Jedes Paar von Knoten in einer Menge von C_i ist durch einen Pfad in der entsprechenden Zusammenhangskomponente von (V, E_i) verbunden. Da Kruskals Algorithmus, die Kanten in der aufsteigenden Reihenfolge ihres Gewichts hinzufügt, ist das Gewicht von jeder Kante entlang des Pfads höchstens $w(e_i)$. Gleichzeitig ist $w(e_i) = \phi(C_i)$, wie wir im obigen Beweis gezeigt haben. □

Referenzen

- Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani, Algorithms, Kapitel 5.1 (Minimum Spanning Trees)
- Understanding Machine Learning, Kapitel 22.2

AGML, Sommersemester 2020 Vorlesung 21 (5 Seiten)

Hierarchisches Clustering II

Anne Driemel Letzte Aktualisierung: 10. Juli 2020

In der letzten Vorlesung haben wir uns mit dem hierarchischen Clustering befasst. Wir haben dafür eine Zielfunktion definiert und gezeigt, dass der Single-Link-Clustering Algorithmus ein optimales Ergebnis liefert. Außerdem haben wir einen wichtigen Zusammenhang zwischen minimalen Spannbäumen und dem Single-Link-Clustering gezeigt, der es uns erlaubt, das hierarchische Clustering effizient zu berechnen.

1 Hierarchisches Clustering und k-Center

Was ist aber, wenn wir ein hierarchisches Clustering mit einer anderen Zielfunktion berechnen wollen? Konkret betrachten wir den Fall, dass die dritte Clustering-Eigenschaft wie folgt abgewandelt ist.

Sei X eine Grundmenge und sei $d(\cdot,\cdot)$ eine Metrik auf X. Ein hierarchisches Clustering einer n-elementigen Menge $S \subseteq X$ ist eine geordnete Menge $\mathcal{C} = \{C_1, \ldots, C_n\}$ mit den folgenden Eigenschaften:

(i) (Jede Menge ist eine Partitionierung von S)
Für alle $1 \leq i \leq n$ hat C_i die Form $\{A_1, \ldots, A_{n-i+1}\}$ mit

(a) $\bigcup_{1 \leq j \leq n-i+1} A_j = S$ und
(b) für $j \neq k$ ist $A_j \cap A_k = \emptyset$,

(ii) (Die Mengen sind hierarchisch geschachtelt)
$C_n = \{S\}$ und für alle $1 \leq i < n$ und $A \in C_i$ existiert ein $B \in C_{i+1}$ mit $A \subseteq B$

(iii) (Die Mengen sind optimal)
Für alle $1 \leq i \leq n$ minimiert C_i die folgende Zielfunktion

$$\phi(C_i) = \max_{A \in C_i} \min_{c \in X} \max_{a \in A} d(c, a)$$

über alle möglichen Partitionierungen von S die aus genau $|C_i|$ Teilmengen von S bestehen.

Für ein einzelnes Clustering C_i ist diese Zielfunktion äquivalent zu der im k-Center-Problem, welches wir in der vorletzen Vorlesung kennengelernt haben. Für jede Menge in C_i wird der Radius einer kleinsten umschließenden Kugel gesucht, das Maximum über alle Mengen bestimmt die Zielfunktion. Wir vergleichen also jedes Clustering in der Hierarchie mit dem optimalen k-Center-Clustering. Zusätzlich wollen wir, dass die Mengen hierarchisch geschachtelt sind.

2 Ein Gegenbeispiel

Wir können zunächst feststellen, dass es Mengen S gibt für die keine Lösung existiert, die alle Eigenschaften erfüllt. Wir betrachten dafür das folgende Beispiel.

Beispiel 21.1. *Sei $X = \mathbb{R}$ mit $d(x, y) = |x - y|$, und sei $S = \{0, 4, 6, 10\}$. Ein hierarchisches Clustering von S ist zum Beispiel*

$$C_4 = \{\{0, 4, 6, 10\}\}, C_3 = \{\{0, 4\}, \{6, 10\}\}, C_2 = \{\{0\}, \{4\}, \{6, 10\}\}, C_1 = \{\{0\}, \{4\}, \{6\}, \{10\}\}.$$

Aber hier ist $\phi(C_2) = 2$, während die optimale 3-Center-Lösung den Zielwert 1 hat:

$$\phi(\{0\}, \{4, 6\}, \{10\}) = 1$$

Ein anderes hierarchisches Clustering ist

$$C_4 = \{\{0, 4, 6, 10\}\}, C_3 = \{\{0\}, \{4, 6, 10\}\}, C_2 = \{\{0\}, \{4\}, \{6, 10\}\}, C_1 = \{\{0\}, \{4\}, \{6\}, \{10\}\}.$$

Aber hier ist $\phi(C_3) = 3$, während die optimale 2-Center-Lösung den Zielwert 2 hat.

$$\phi(\{0, 4\}, \{6, 10\}) = 2$$

Wenn wir alle möglichen Mengen $C = \{C_1, \ldots, C_4\}$ welche die Clustering-Bedingungen (i) und (ii) erfüllen analysieren, dann können wir sehen, dass für jede dieser Lösungen entweder $\phi(C_2) \geq 2$ oder $\phi(C_3) \geq 3$ gilt. Also gibt es kein hierarchisches Clustering, welches Bedingung (iii) erfüllt.

3 Der Algorithmus von Dasgupta und Long

In der vorletzten Vorlesung haben wir den Algorithmus von Gonzales kennengelernt. Der Algorithmus berechnet die von ihm gewählten Zentren iterativ. In jedem Schritt ist das berechnete Clustering höchstens um einen Faktor 2 schlechter ist als das optimale Clustering. Allerdings erfüllen die Clusterings nicht unbedingt die Clustering-Bedingung (ii). Wir werden heute den Algorithmus von Dasgupta und Long kennenlernen. Dieser wandelt die Zuweisungen der Punkte zu den von Gonzales-Algorithmus berechneten Clusterzentren so ab, dass hieraus ein hierarchisches Clustering wird. Zunächst müssen wir dazu den Gonzales-Algorithmus leicht abwandeln.

```
Gonzales-Algorithmus(S = {x_1, ..., x_n})
 1. π_1 = 1
 2. d_1, ..., d_n = ∞
 3. for i in 1...n do
 4.       for j in 1...n do
 5.            //d_j speichert den Abstand von x_j zu den bisher gewählten Zentren
 6.            d_j = min(d_j, d(x_j, x_{π_i}))
 7.       r_i = max_{1≤j≤n} d_j
 8.       π_{i+1} = arg max_{1≤j≤n} d_j
 9. Return x_{π_1}, ..., x_{π_n} und r_1, ..., r_n
```

Der einzige Unterschied bis jetzt, ist dass wir den Parameter k auf n gesetzt haben und die Ausgabe erweitert haben. Der Algorithmus gibt eine Permutation der Eingabemenge $x_{\pi_1}, \ldots, x_{\pi_k}$ und die Liste der Radien der berechneten Clusterings zurück. Diese Permutation wird auch *Greedy-Permutation* genannt.

Sei $c_1, \ldots, c_n$ mit $c_i = x_{\pi_i}$ und $r_1, \ldots, r_n$ die Ausgabe des Gonzales Algorithmus für eine Menge S. Wir gruppieren die Punkte nun anhand der Radien in verschiedene *Granularitätsebenen.*

$$L_0 = \{c_1\},\ L_j = \left\{\ c_{i+1}\ \middle|\ r_i \in \left(\frac{r_1}{2^j}, \frac{r_1}{2^{j-1}}\right]\ \right\} \text{ für } j \geq 1.$$

Außerdem definieren wir eine *Elternfunktion* auf der Menge der Punkte. Die Elternfunktion definiert einen gerichteten Graphen G auf der Punktmenge, der eine Baumstruktur hat. Die Wurzel dieses Baumes ist c_1. Sei die Elternfunktion $p : \{2, \ldots, n\} \to \{1, \ldots, n\}$ definiert als

$$p(i) = \operatorname*{arg\,min}_{1 \leq j \leq n} \left\{\ d(c_i, c_j)\ \middle|\ c_j \in \bigcup_{j'=0}^{L(i)-1} L_{j'}\ \right\}$$

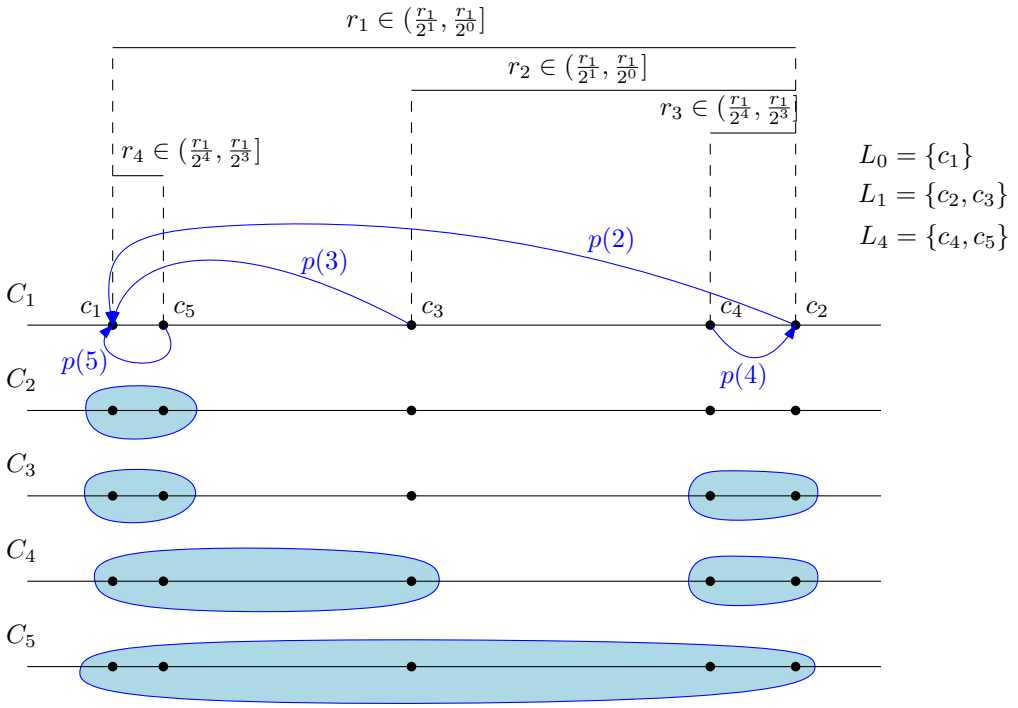


Abbildung 1: Schematische Darstellung eines hierarchischen Clusterings von Dasgupta und Long von der Menge $c_1, \ldots, c_5$ (in der Greedy-Permutation).

wobei $L(i)$ die Granularitätsebene angibt (also den Index der Menge aus $L_0, L_1, \ldots$), in der sich c_i befindet. Für einen Punkt c_i ist $p(i)$ der nächste Nachbar in der Menge der Punkte, die in einer niedrigeren Granularitätsebene liegen. Die Knoten von G sind gegeben durch die Menge $\{c_1, \ldots, c_n\}$ und jeder Knoten c_i, ausser dem Wurzelknoten, hat (genau) eine ausgehende Kante, nämlich die Kante $(c_i, c_{p(i)})$. Abbildung 1 zeigt ein Beispiel dieses Graphen.

Der Algorithmus von Dasgupta und Long berechnet ein hierarchisches Clustering welches unsere Clustering-Bedingungen (i) und (ii) erfüllt, wie folgt. Angefangen mit dem Clustering C_1 werden in jedem Schritt immer genau zwei Cluster vereinigt. Hierbei werden die Punkte in der umgekehrten Reihenfolge betrachtet, in der sie in der Greedy-Permutation auftauchen. Sei c_j ein Element dieser Reihenfolge. Dann vereinigen wir genau diese beiden Cluster, die durch eine Elternkante zwischen c_j und $c_{p(j)}$ verbunden sind. Der Pseudocode des Algorithmus ist wie folgt.

```
Dasgupta-Long-Algorithmus(S = {x_1, ..., x_n})
  1. c_1, ..., c_n, r_1, ..., r_n ← Gonzales-Algorithmus(S)
  2. Berechne Mengen L_0 und L_j mit L_j ≠ ∅
  3. Berechne Werte der Elternfunktion p(i) für alle 1 < i ≤ n
  4. Sei C_1 = {{c_1}, ..., {c_n}}
  5. for i in 1...n do
  6.       Sei j = n − i + 1, sei A ∈ C_i die Menge die c_j enthält
  7.       Sei j' = p(j), sei B ∈ C_i die Menge die c_{j'} enthält
  8.       Sei C_{i+1} dieselbe Menge wie C_i, nur dass A und B vereinigt sind
  9. Return C_1, ..., C_n
```

Eine andere hilfreiche Interpretation des Graphen G in Zusammenhang mit dem hierarchi-

schen Clustering ist die folgende. Das Clustering C_i ergibt sich durch die Zusammenhangskomponenten von G, wenn die Elternkanten entfernt werden, die von den ersten i Punkten in der Greedy-Permutation ausgehen. Statt die Cluster Schritt für Schritt zu vereinigen, könnten wir uns also auch den umgekehrten Prozess vorstellen, in dem Cluster Schritt für Schritt geteilt werden, indem wir Kanten in G entfernen. Um zu überprüfen, ob der Graph G tatsächlich ein Baum ist, stellen wir die folgenden Überlegungen an. Da alle Knoten ausser der Wurzelknoten einen Elternknoten haben, können wir von einem beliebigen Knoten aus, den Elternkanten folgen, bis wir irgendwann am Wurzelknoten ankommen. Also ist der Graph zusammenhängend. Gleichzeitig wissen wir, dass der Graph genau $n-1$ Kanten hat, wobei n die Anzahl der Knoten ist.

4 Analyse der Qualität der berechneten Lösung

Wir wollen nun analysieren, wie gut die Werte der Zielfunktion $\phi(C_1), \ldots, \phi(C_n)$ auf dem berechneten Clustering sind, wobei wir jedes Clustering C_i wieder mit dem optimalen Clustering mit der gleichen Anzahl von Clustern vergleichen.

Lemma 21.2. *Für alle $c_i \in S$ gilt $d(c_i, c_{p(i)}) \leq \frac{r_1}{2^{L(i)-1}}$ wobei $L(i)$ die Granularitätsebene von c_i angibt.*

Beweis. Wir zeigen zuerst eine andere Aussage: Für alle j ist der Abstand zwischen c_i und seinem nächsten Nachbarn in der Menge $L_0 \cup L_1 \cup \cdots \cup L_j$ höchstens $\frac{r_1}{2^j}$. Sei c_k der Punkt mit höchsten Index in L_j. Dann ist

$$L_0 \cup L_1 \cup \cdots \cup L_j = \{c_1, \ldots, c_k\} =: Z$$

Aus der Analyse des Gonzales-Algorithmus folgt, dass jeder Punkt in S Abstand höchstens r_k zu seinem nächsten Nachbarn in Z hat.

Aus der Definition der Granularitätsebenen folgt, dass $c_{k+1} \in L_j$ genau dann wenn $r_k \in (\frac{r_1}{2^j}, \frac{r_1}{2^{j-1}}]$. Da $c_{k+1} \notin L_j$, folgt daraus, dass $r_k \leq \frac{r_1}{2^j}$.

Der Satz folgt nun indem wir $j = L(i) - 1$ wählen. Insbesondere haben wir hergeleitet, dass gilt

$$d(c_i, p(i)) = \min_{1 \leq j' \leq L(i)-1} d(c_i, c_{j'}) \leq \frac{r_1}{2^{L(i)-1}}$$

□

Satz 21.3. *Sei $1 \leq k \leq n$ und sei $i = n - k + 1$. Sei C^* ein optimales k-Center-Clustering einer Menge S mit k Clustern. Für das vom Dasgupta-Long-Algorithmus berechnete Clustering C_i gilt*

$$\phi(C_i) \leq 8\phi(C^*)$$

Beweis. Wir zeigen $\phi(C_k) \leq 4r_k$. Der Satz folgt dann aus Satz 19.2 (Gonzales-Algorithmus). Sei $c_i \in S$ fest. Wir folgen den Elternkanten von c_i bis wir bei einem Punkt in der Menge $Z = \{c_1, \ldots, c_k\}$ ankommen. Die Menge C_i wird vom Algorithmus berechnet. Die Clusterzentren werden aber nicht vom Algorithmus festgelegt. Wir analysieren die Kosten für Clusterzentren Z und wir weisen c_i dem Clusterzentrum $c_{p(i)}$ zu.

Sei die Sequenz der Indizes auf diesem Pfade $i_0, i_2, \ldots, i_\ell$, mit $c_{i_\ell} \in Z$. Das heißt, wir definieren $i_0 = i$, $i_1 = p(i_0)$, $i_2 = p(i_1)$, etc. Die Sequenz $i_0, i_1, \ldots, i_\ell$ ist absteigend, da die Elternkanten nur auf Element in niedrigeren Granularitätsebenen zeigen.

Wir leiten eine obere Schranke für den Abstand zwischen c_i und c_{i_ℓ} her, indem wir die Dreiecksungleichung auf die Kanten der Pfade anwenden.

$$d(c_i, c_{i_\ell}) \leq d(c_{i_0}, c_{i_1}) + d(c_{i_1}, c_{i_2}) + \cdots + d(c_{i_{\ell-1}}, c_{i_\ell})$$

Laut Lemma 21.2 können wir diese Terme beschränken und erhalten

$$d(c_i, c_{i_\ell}) \leq \frac{r_1}{2^{L(i_0)-1}} + \frac{r_1}{2^{L(i_1)-1}} + \cdots \frac{r_1}{2^{L(i_{\ell-1})-1}}$$

Da sich der Index der Granularitätsebene mit jeder Elternkante um mindestens 1 verringert, ist $L(i_0) = L(i)$ und $L(i_j) \leq L(i) - j$. Es folgt

$$d(c_i, c_{i_\ell}) \leq \frac{r_1}{2^{L(i)-1}} + \frac{r_1}{2^{L(i)-2}} + \cdots + \frac{r_1}{2^{L(i)-\ell+1}} + \cdots + \frac{r_1}{2^{L(i_{\ell-1})-1}} \leq \sum_{j=L(i_{\ell-1})-1}^{L(i)-1} \frac{r_1}{2^j}$$

Wir ersetzen $j' = j - (L(i_{\ell-1}) - 1)$ in der Laufvariable der Summe und erhalten

$$d(c_i, c_{i_\ell}) \leq \sum_{j'=0}^{\infty} \frac{r_1}{2^{j'+(L(i_{\ell-1})-1)}} = \frac{r_1}{2^{(L(i_{\ell-1})-1)}} \sum_{j'=0}^{\infty} \frac{1}{2^{j'}} \leq \frac{r_1}{2^{(L(i_{\ell-1})-2)}} \leq 4 \cdot \frac{r_1}{2^{(L(i_{\ell-1}))}}$$

In welcher Granularitätsebene ist also $c_{i_{\ell-1}}$? Wir wissen, dass $c_{i_\ell} \in Z$ und also $i_\ell \in \{1, \ldots, k\}$, da wir den Elternkanten bis zu diesem Punkt gefolgt sind. Also ist $i_{\ell-1} \geq k+1$, da wir sonst schon bei $i_{\ell-1}$ in der Menge Z terminiert wären. Daraus schliessen wir, dass $L(i_{\ell-1}) \geq L(k+1)$, und daher

$$d(c_i, c_{i_\ell}) \leq 4 \cdot \frac{r_1}{2^{(L(k+1))}}$$

Weiter ist nach der Definition der Granularitätsebenen

$$c_i \in L_j \Leftrightarrow r_{i-1} \in \left(\frac{r_1}{2^j}, \frac{r_1}{2^{j-1}}\right]$$

Für $i = k+1$ und $j = L(k+1)$ erhalten wir also $r_k > \frac{r_1}{2^{L(k+1)}}$.

Da wir die Schranke für jedes $x_i \in S$ herleiten können, folgern wir, dass

$$\phi(C_k) \leq \max_{x_i \in S} \min_{c \in Z} d(c_i, c) \leq 4 \cdot \frac{r_1}{2^{(L(k+1))}} \leq 4 r_k$$

□

Referenzen

- Sanjoy Dasgupta, Philip M. Long, Performance guarantees for hierarchical clustering. Journal of Computer and System Sciences, Volume 70, Issue 4, 2005.

AGML, Sommersemester 2020 Vorlesung 22 (7 Seiten)

Dimensionsreduktion

Anne Driemel Letzte Aktualisierung: 14. Juli 2020

In vielen Anwendungen sind die Daten, die wir als Eingabe für unsere Lernalgorithmen bekommen, hochdimensional. In der Bildanalyse, zum Beispiel, ist jeder Datenpunkt eine Kombination von vielen Pixelwerten. Jeder einzelne Pixel ist dabei ein eigenes Merkmal und nimmt somit eine eigene Dimension im Merkmalsraum ein. Gleichzeitig kann man sich leicht vorstellen, dass der exakte Werte jedes einzelnen Pixels nicht unbedingt für die Analyse benötigt wird. In der Dimensionsreduktion geht es darum die Daten in einen geeigneten niedrig-dimensionalen Unterraum zu projizieren, um die Daten vereinfacht darzustellen, wobei die Datenpunkte trotzdem möglichst gut erhalten bleiben sollen.

1 Definition der Zielfunktion

Sei $S = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}^d$ eine Menge von Datenpunkten und sei $k \in \mathbb{N}$ ein Parameter mit $k \leq d$. Wir wollen S mithilfe einer Funktion $f : \mathbb{R}^k \to \mathbb{R}^d$ beschreiben, definiert durch $\mu \in \mathbb{R}^d$ und eine $d \times k$ Matrix V mit

$$f(\lambda) = \mu + V\lambda, \text{ mit } \mu \in \mathbb{R}^d$$

Wir verlangen außerdem von der Matrix V, dass sie orthonormal ist, das heißt

(i) für jeden Spaltenvektor v_i von V gilt, dass $\langle v_i, v_i \rangle = 1$

(ii) für je zwei Spaltenvektoren v_i und v_j von V gilt, dass $\langle v_i, v_j \rangle = 0$

Die Funktion f bildet auf eine k-dimensionale Hyperebene im $\mathbb{R}^d$ ab. Ziel ist es also, die Datenpunkte in S innerhalb einer k-dimensionalen Hyperebene angemessen darzustellen. Die Dimensionsreduktion geschieht hier indem wir jedes x_i über seinen Index dem Vektor λ_i assoziieren. Die Abbildung in den k-dimensionalen Unterraum wird also durch die Wahl der Vektoren $\lambda_1, \ldots, \lambda_n$ bestimmt. Wie gut unsere Repräsentation von S ist, messen wir mithilfe der Summe der quadratischen Abstände. Dies wird in der folgenden Zielfunktion ausgedrückt.

Wir wollen einen Vektor μ, eine Matrix V und Spaltenvektoren $\lambda_1, \ldots, \lambda_n$ finden, welche zusammen die Zielfunktion

$$\phi(\mu, V, \lambda_1, \ldots, \lambda_n) = \sum_{i=1}^{n} \|x_i - f(\lambda_i)\|^2$$

minimieren. Diese Zielfunktion lässt sich noch vereinfachen. Dazu betrachten wir zunächst λ_i und halten dabei V und μ und λ_j mit $i \neq j$ fest. Man kann zeigen, dass ϕ für

$$\lambda_i = V^T(x_i - \mu) \tag{1}$$

minimiert wird. Insbesondere ist $f(\lambda_i)$, für diese Wahl von λ_i, die orthogonale Projektion von x_i auf die Hyperebene, die durch μ und V gegeben ist, und damit der Punkt in der Hyperebene mit dem kleinsten Abstand zu x_i. Im nächsten Schritt halten wir V und die $\lambda_1, \ldots, \lambda_n$ fest und

minimieren ϕ über alle Werte von μ. Hier können wir die partielle Ableitung nach μ wie folgt herleiten. Sei $\gamma_i \in \mathbb{R}^d$ definiert als $\gamma_i = x_i - V\lambda_i$ für jedes $1 \leq i \leq n$.

$$\begin{aligned}
\frac{\partial}{\partial \mu} \sum_{i=1}^{n} \|x_i - f(\lambda_i)\|^2 &= \frac{\partial}{\partial \mu} \sum_{i=1}^{n} \|x_i - \mu - V\lambda_i\|^2 \\
&= \frac{\partial}{\partial \mu} \sum_{i=1}^{n} \|\gamma_i - \mu\|^2 \\
&= \sum_{i=1}^{n} \frac{\partial}{\partial \mu} \langle \gamma_i - \mu, \gamma_i - \mu \rangle \\
&= \sum_{i=1}^{n} \left(\frac{\partial}{\partial \mu_1} (\gamma_{i,1} - \mu_1)^2, \ldots, \frac{\partial}{\partial \mu_d} (\gamma_{i,d} - \mu_d)^2 \right) \\
&= \sum_{i=1}^{n} (-2(\gamma_{i,1} - \mu_1), \ldots, -2(\gamma_{i,d} - \mu_d)) \\
&= \sum_{i=1}^{n} -2(\gamma_i - \mu) \\
&= \sum_{i=1}^{n} -2(x_i - \mu - V\lambda_i)
\end{aligned}$$

Setzen wir dies gleich dem Nullvektor, dann erhalten wir

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i - V \left(\frac{1}{n} \sum_{i=1}^{n} \lambda_i \right)$$

Sei $\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$. Setzen wir nun (1) ein, dann erhalten wir

$$\mu = \overline{x} - V \left(\frac{1}{n} \sum_{i=1}^{n} V^T (x_i - \mu) \right) = \overline{x} - VV^T (\overline{x} - \mu)$$

Das ist äquivalent zu

$$VV^T (\overline{x} - \mu) = \overline{x} - \mu$$

Wir können hier $\mu = \overline{x}$ wählen und diese Gleichung erfüllen, ohne dass die Wahl von V berührt ist.

Damit ergibt sich für unsere Zielfunktion

$$\phi(V) = \sum_{i=1}^{n} \|(x_i - \overline{x}) - VV^T (x_i - \overline{x})\|^2 \tag{2}$$

Wir können dies so interpretieren, dass wir eigentlich eine Funktion f für die zentrierte Menge $S' = \{x'_1, \ldots, x'_n\}$ mit $x'_i = x_i - \overline{x}$ finden wollen. Wir können vereinfachend annehmen, dass die Menge S schon zentriert ist. Dann ist $\overline{x}$ gleich dem Nullvektor und die optimale Hyperebene geht durch den Ursprung. In diesem Fall ist die Funktion f eine lineare Abbildung und bildet auf einen linearen Unterraum ab, die durch die Spaltenvektoren von V aufgespannt wird.

2 Beispiel

Wir wollen uns der Funktion f zunächst weiter anhand eines Beispiels nähern. Abbildung 1 zeigt eine zufällige Auswahl von Bildern einer handgeschriebenen Ziffer Drei, aus dem MNIST Datensatz. Jedes Bild ist durch einen hochdimensionalen Vektor von Pixelwerten gegeben. Ein Bild mit $h \times w$ Pixeln ist demnach ein Vektor im $\mathbb{R}^{h \cdot w}$. Wir wollen diesen Datensatz in der Parametrisierung einer 2-dimensionalen Hyperebene betrachten, welche die Zielfunktion ϕ minimiert. Das linke Bild zeigt den Vektor μ, also das Bild einer gemittelten handgeschriebenen

Ziffer Drei. Das mittlere Bild zeigt eine Darstellung des ersten Spaltenvektors v_1 der Matrix V, das rechte Bild zeigt eine Darstellung des zweiten Spaltenvektors v_2 der Matrix V. Beachte, dass der graue Hintergrund hier ein Artifakt der Darstellung ist. Die Pixelwerte sind in der Darstellung auf Grauwerte zwischen 0 und 1 abgebildet. Die hellen Pixel der Vektoren v_1 und v_2 sollten also als negative Werte interpretiert werden und dunkle Pixel als positive Werte.

Ein Punkt in der k-dimensionalen Hyperebene, die durch μ, v_1 und v_2 bestimmt ist, wird durch einen Parametervektor $\lambda = (t_1, t_2) \in \mathbb{R}^2$ als

$$f(t_1, t_2) = \mu + t_1 v_1 + t_2 v_2$$

dargestellt. Abbildung 2 zeigt das Ergebnis für eine Auswahl an Punkten im Parameterraum.

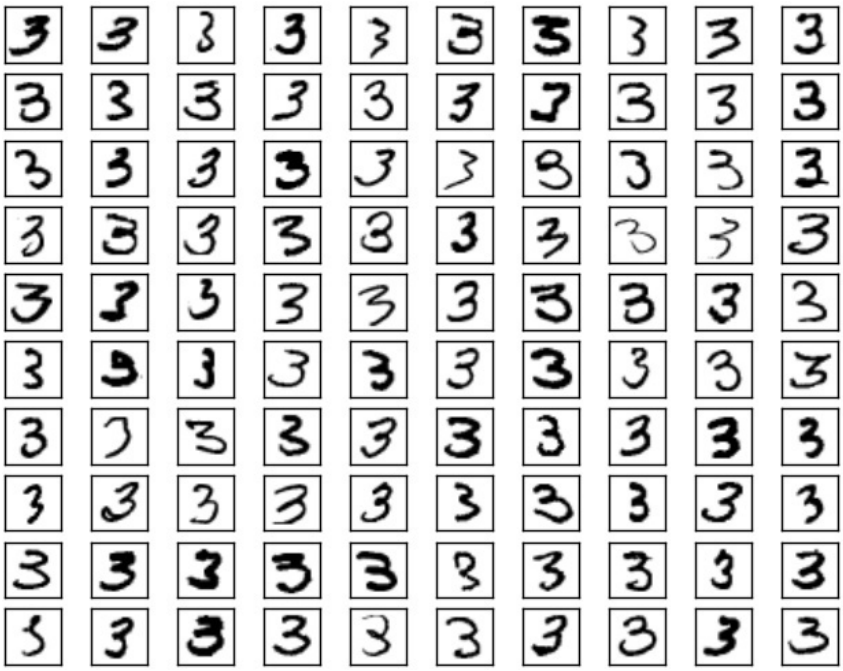


Abbildung 1: Zufällige Auswahl des MNIST-Datensatzes von Bildern von handgeschriebenen Ziffern. Hier ist eine Auswahl getroffen von Beispielen der Ziffer Drei.

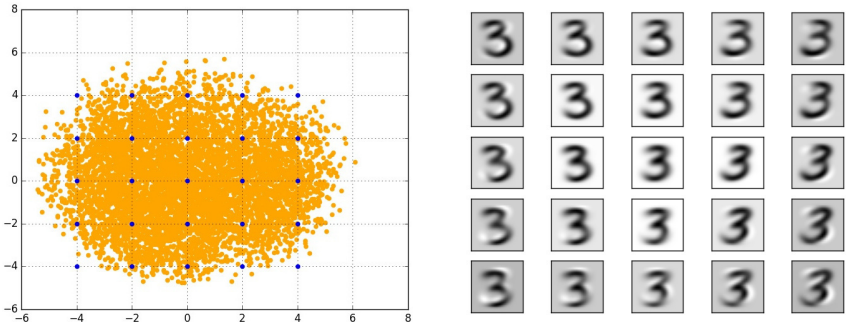


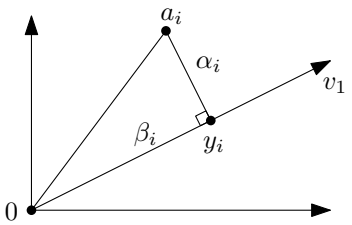
Abbildung 2: Links: Punktmenge (gelb) aus dem MNIST-Datensatz (nur Ziffer Drei) projiziert auf den Unterraum, der durch v_1 und v_2 gespannt wird. Rechts: Darstellung der Rekonstruktion durch die Funktion $f(t_1, t_2) = \mu + t_1 v_1 + t_2 v_2$ für die blauen Gitterpunkte (t_1, t_2) im Bild links.

3 Singulärwertzerlegung

Wir wollen eine Matrix V finden, welche die Zielfunktion ϕ in (2) minimiert. Dazu schreiben wir unsere Menge von Datenpunkten $S = \{x_1, \ldots, x_n\} \subseteq \mathbb{R}^d$ in eine Matrix. Sei A eine $n \times d$ Matrix mit Zeilenvektoren $a_1, \ldots a_n$ mit $a_i = (x_i - \overline{x})$ mit $\overline{x} = \frac{1}{n}\sum_{i=1}^n x_i$ für alle $1 \leq i \leq n$.

Wir betrachten zunächst den Fall $k = 1$. In diesem Fall hat die Matrix V nur einen Spaltenvektor v_1. Dieser Spaltenvektor spannt einen 1-dimensionalen Unterraum, also eine Gerade durch den Ursprung, und wir betrachten die Projektionen der Eingabemenge auf diese Gerade.

Betrachte das Dreieck mit den Eckpunkten a_i, der Projektion $y_i = v_1 \langle v_1, a_i \rangle$, und dem Nullpunkt. Seien $\beta_i = \| v_1 \langle v_1, a_i \rangle \|$ und $\alpha_i = \| a_i - v_1 \langle v_1, a_i \rangle \|$, und $\|a_i\|$ die Seitenlängen dieses Dreiecks.



Es folgt aus dem Satz von Pythagoras, dass

$$\beta_i^2 = \|a_i\|^2 + \alpha_i^2$$

Damit ist $\alpha_i^2 = \|a_i\|^2 - \beta_i^2$. Wir suchen nach einem Vektor v_1 mit $\|v_1\| = 1$, sodass $\phi(v_1) = \sum_{i=1}^n \alpha_i^2$ minimiert wird. Durch Einsetzen der obigen Beobachtung erhalten wir

$$\operatorname*{arg\,min}_{\substack{v_1 \in \mathbb{R}^d \\ \|v_1\|=1}} \sum_{i=1}^n \alpha_i^2 = \operatorname*{arg\,min}_{\substack{v_1 \in \mathbb{R}^d \\ \|v_1\|=1}} \sum_{i=1}^n \|a_i\|^2 - \beta_i^2 = \operatorname*{arg\,max}_{\substack{v_1 \in \mathbb{R}^d \\ \|v_1\|=1}} \sum_{i=1}^n \beta_i^2$$

Um die Summe auf der rechten Seite noch weiter zu vereinfachen, beobachten wir, dass

$$\beta_i = \| v_1 \langle v_1, a_i \rangle \| = | \langle v_1, a_i \rangle |,$$

da $\|v_1\| = 1$ ist. Also ist

$$\sum_{i=1}^n \beta_i^2 = \sum_{i=1}^n | \langle v_1, a_i \rangle |^2 = \|Av_1\|^2$$

Das heißt, ϕ zu minimieren ist äquivalent dazu, $\|Av_1\|$ zu maximieren.

Angenommen, wir könnten v_1 bestimmen. Betrachte den folgenden Greedy-Algorithmus, der weitere Spaltenvektoren $v_2, \ldots, v_k$ der Matrix V unter dieser Annahme bestimmt.

```
Greedy-Algorithmus(n × d Matrix A)
  1. v_1 = arg max_{||v_1||=1} ||Av_1||
  2. σ_1 = ||Av_1||
  3. while σ_i ≠ 0 do
  4.     i = i + 1
  5.     v_i = arg max_{||v_i||=1, v_i ⊥ v_1,...v_{i-1}} ||Av_i||
  6.     σ_i = ||Av_i||
  7. Return v_1, ..., v_i
```

Man kann zeigen, dass der Algorithmus eine sogenannte Singulärwertzerlegung der Matrix A bestimmt. Allgemein besteht die Singulärwertzerlegung einer reellen Matrix A aus drei Matrizen U, D, V, mit

$$A = U \cdot D \cdot V^T$$

und mit den folgenden Eigenschaften der Matrizen

- U ist eine $n \times r$ Matrix mit orthonormalen Spaltenvektoren $u_1, \ldots, u_r$,
- V ist eine $d \times r$ Matrix mit orthonormalen Spaltenvektoren $v_1, \ldots, v_r$,
- D ist eine $r \times r$ Diagonalmatrix mit Einträgen $\sigma_1 \geq \cdots \geq \sigma_r \geq 0$,

wobei r den Rang der Matrix A bezeichnet, das heißt r ist die maximale Anzahl linear unabhängiger Zeilenvektoren von A.

Wir nennen die Spaltenvektoren von V die *rechten Singulärvektoren*, die Spaltenvektoren von U die *linken Singulärvektoren* und die Werte $\sigma_1, \ldots, \sigma_r$ die *Singulärwerte*. Wir können die obige Gleichung schreiben als

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T,$$

Betrachten wir nur die Summe der ersten k Terme, dann erhalten wir eine Matrix

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T$$

Die Zeilenvektoren von A_k entsprechen den Vektoren y_i in dem von V aufgespannten k-dimensionalen Unterraum, welche unsere Datenpunkte a_i approximieren sollen. Dadurch, dass die Singulärwerte ihrer Größe nach geordnet sind, wählen wir mit A_k genau die Terme aus, die am stärksten in die Summe eingehen.

Alternativ können die Vektoren $v_1, \ldots, v_k$ durch eine Eigendekomposition der Matrix $A^T A$ bestimmt werden. Dort würden wir die k Eigenvektoren mit den größten Eigenwerten auswählen. Die Darstellung der Datenpunkte im Unterraum der ersten k Eigenvektoren, bzw. Singulärvektoren, wird auch als Eigenkomponentenanalyse bezeichnet.

4 Potenzmethode

Wie kann man nun den Singulärvektor $\arg\max_{\|v_1\|=1} \|Av_1\|$ bestimmen? Dafür betrachten wir die sogenannte Potenzmethode. Die Methode hat ihren Namen daher, dass sie das Ergebnis bestimmt indem sie eine Matrix immer wieder mit sich selbst multipliziert, um eine hohe Potenz dieser Matrix zu berechnen.

Betrachte die Matrix $B = A^T \cdot A$. Sei $A = \sum_{i=1}^{r} \sigma_i u_i v_i^T$ die Singulärwertzerlegung, wie oben definiert. Dann ist

$$A^T = \sum_{i=1}^{r} \sigma_i (u_i v_i^T)^T = \sum_{i=1}^{r} \sigma_i v_i u_i^T.$$

Also erhalten wir für B

$$\begin{aligned} B &= \left(\sum_{i=1}^{r} \sigma_i v_i u_i^T \right) \left(\sum_{j=1}^{r} \sigma_j u_j v_j^T \right) \\ &= \sum_{i=1}^{r} \sum_{j=1}^{r} \sigma_i \sigma_j (v_i u_i^T)(u_j v_j^T) \\ &= \sum_{i=1}^{r} \sigma_i^2 v_i (u_i^T u_i) v_i^T + \sum_{i=1}^{r} \sum_{\substack{j=1 \\ i \neq j}}^{r} \sigma_i \sigma_j v_i (u_i^T u_j) v_j^T \end{aligned}$$

Da die Vektoren $u_1, \ldots, u_r$ orthonormal sind, gilt $u_i^T u_i = 1$ für $1 \leq i$ und $u_i^T u_j = 0$ für $i \neq j$. Daher folgt

$$B = \sum_{i=1}^{r} \sigma_i^2 v_i v_i^T$$

Betrachte nun die Matrix $B^2 = B \cdot B$.

$$\begin{aligned} B^2 &= \left(\sum_{i=1}^{r} \sigma_i^2 v_i v_i^T \right) \left(\sum_{j=1}^{r} \sigma_j^2 v_j v_j^T \right) \\ &= \sum_{i=1}^{r} \sum_{j=1}^{r} \sigma_i \sigma_j (v_i v_i^T)(v_j v_j^T) \\ &= \sum_{i=1}^{r} \sigma_i^2 v_i (v_i^T v_i) v_i^T + \sum_{i=1}^{r} \sum_{\substack{j=1 \\ i \neq j}}^{r} \sigma_i \sigma_j v_i (v_i^T v_j) v_j^T \end{aligned}$$

Da die Vektoren $v_1, \ldots, v_r$ orthonormal sind, gilt $v_i^T v_i = 1$ für $1 \leq i$ und $v_i^T v_j = 0$ für $i \neq j$. Daher erhalten wir

$$B^2 = \sum_{i=1}^{r} \sigma_i^4 v_i v_i^T$$

Allgemein können wir damit für die $k$te Potenz von B herleiten, dass

$$B^k = \sum_{i=1}^{r} \sigma_i^{2k} v_i v_i^T$$

da der Term $(v_i^T v_i)$ immer gleich 1 ist und bei der Multiplikation stets wegfällt. Wenn $\sigma_1 > \sigma_2$, dann konvergiert B^k für große Werte von k gegen den ersten Term der Summe,

$$B^k \to \sigma_1^{2k} v_1 v_1^T$$

Das heißt, wir können v_1 bestimmen, indem wir einen Spaltenvektor von B^k normieren.

Referenzen

- Foundations of Machine Learning, Kapitel 15.1 und 15.3.1
- Understanding Machine Learning, Kapitel 23.1
- Avrim Blum, John Hopcroft, Ravindran Khannan, Foundations of Data Science, Kapitel 3
- Trevor Hastie, Robert Tibshirani, Jerome Friedman, Elements of Statistical Learning