Algorithmisches Lernen
5. Übungsblatt

Adam Duck

29. Mai 2001

5.1 Aufgabe: In dieser Aufgabe entwickeln wir einen PAC-Algorithmus für $\mathsf{Threshold}_n$:

(a) Beweise, dass jedes Konzept $c \in \mathsf{Threshold}_n$ ausgedrückt werden kann durch eine Threshold-funktion der Form

$$c(x) = 1 \iff \sum_{i=1}^{n} a_i x_i \geq b \text{ und } c(x) = 0 \iff \sum_{i=1}^{n} a_i x_i \leq b - 1$$

wobei $x = (x_1, x_2, \ldots, x_n) \in \{0,1\}^n$ und $a_1, a_2, \ldots, a_n, b \in \mathbb{Z}$

(b) Zeige, dass $\mathsf{Threshold}_n$ einen PAC-Algorithmus besitzt, der polynomiell in n, $\frac{1}{\varepsilon}$ und $\frac{1}{\delta}$ ist.

Hinweis:
Für ein lineares Ungleichungssystem der Form $Ax \leq b$ kann eine Lösung x_0 mit $Ax_0 \leq b$ in polynomieller Zeit bezüglich n, m und L mit Karmarkars Algorithmus bestimmt werden. A ist eine $m \times n$-Matrix, x ein Vektor der Länge n, b ein Vektor der Länge m und L gibt die Länge der Binärkodierung von A und b an. Die obere Schranke $s \leq \frac{1}{\varepsilon}\left(\mathsf{VC}(\mathcal{C}) \ln\left(\frac{1}{\varepsilon}\right) + \ln\left(\frac{1}{\delta}\right)\right)$ für die Beispielanzahl darf ebenfalls benutzt werden. (6 Punkte)
Lösung:

5.2 Aufgabe: Konstruieren sie einen effizienten PAC-Algorithmus für die Konzeptklasse Sym_n aller symmetrischen Funktionen f über n Variablen. Eine Funktion $\{0,1\}^n \to \{0,1\}$ heißt symmetrisch, falls für jede Permutation π gilt:

$$f(x_1, x_2, \ldots, x_n) = f(x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$$

Ein klassifiziertes Beispiel besteht aus einem Eingabevektor, einem Funktionswert und der Klassifizierung, die angibt, ob die zu lernende Funktion für diesen Eingabevektor diesen Funktionswert annimmt oder nicht.

Ihr Algorithmus sollte möglichst wenige Gegenbeispiele anfordern. (6 Punkte)
Lösung:
Zuerstmal sollten wir herausstellen, dass es gar nicht so viele mögliche Eingabefolgen gibt, die wir lernen müssen. Denn wenn bei jeder Permutation derselbe Funktionswert herauskommt, ist für die Funktion nur die Anzahl der Einsen und Nullen wichtig.

Und da $x_1, \ldots, x_n \in \{0,1\}$ gibt es nur 2^n mögliche Kombinationen.

Doch nun zum Algorithmus

- fordere s Beispiele an mit $s = \Theta\left(\frac{n}{\varepsilon} + \frac{1}{\varepsilon} \cdot \ln\left(\frac{1}{\delta}\right)\right)$ S. 51 Skript
- initialisiere $B(1, \ldots, n) = (0, \ldots, 0)$
- wiederhole für jedes Beispiel
  - setze i = Anzahl der Einsen im Eingabevektor
  - falls x ein pos. Beispiel, setze $B[i] = B[i] + 1$
  - falls x ein neg. Beispiel, setze $B[i] = B[i] - 1$
- konstruiere Hypothese $h = x_1 \vee x_2 \vee \ldots \vee x_n$, wobei $x_k = i$ für $B[i] \geq 0$

Wenn wir nun Beispiele zum klassifizieren bekommen, schauen wir nach, ob $h = i$ (wobei i = Anzahl der Einsen) ist.

5.3 Aufgabe: Wir betrachten den folgenden Occam-Algorithmus L für konvexe Polygone in der Ebene:

Gegeben: Eine Menge von m zufällig gewählten Beispielen.

(a) Finde eine Linie, die alle positiven Beispiele auf der einen Seite und möglichst viele negative auf der anderen hat.

(b) Entferne diese negativen Beispiele und wiederhole Schritt 1, solange negative Beispiele vorhanden sind.

(c) Das Ergebnis ist das aus diesen Linien bestehenden konvexe Polygon.

Zeige, dass nur $\mathcal{O}(e \log m)$ Iterationen notwendig sind, wobei e die Eckenzahl des Zielkonzeptes sei.
Hinweis:
Das allgemeine Set-Cover Problem besteht aus einem Mengensystem $S = S_1, S_2, \ldots, S_s$ mit

$$\bigcup_{1 \leq i \leq s} S_i = U$$

Gesucht ist eine möglichst kleines Teilmengensystem $I \subseteq \{1, \ldots, s\}$ mit

$$\bigcup_{i \in I} S_i = U$$

Der Greedy-Algorithmus, der zu jedem Zeitpunkt die Menge zu I hinzufügt, die die meisten noch nicht abgedeckten Elemente aus U überdeckt, liefert eine Lösung I_{greedy}, die der Ungleichung

$$|I_{\mathsf{greedy}}| \leq |I_{\mathsf{opt}}| \cdot \log(|U|) \tag{3.1}$$

genügt. (6 Punkte)
Lösung:
Da unser Problem eine abstrakte Variante hat (Set-Cover), zeigen wir, dass die Ungleichung (3.1) für unseren Fall anwendbar ist.

$\log(|U|) = \log m$ ist trivial, denn bei jeder Iteration bekommen wir mindestens ein Beispiel zu unserer Menge hinzu. Also ist die Anzahl der Iterationen gleich der Anzahl der Elemente in I.

Jetzt muss nur noch gezeigt werden, dass $|I_{\mathsf{opt}}| = e$, also der Anzahl der Ecken ist. Dies ist aber auch klar, denn wir beschreiben mit unserem Greedy-Algorithmus ja die konvexe Hülle der positiven Beispiele (wobei wir den Einschluss von negativen Beispielen mit einem Fehler von ε erlauben). Wenn wir jetzt annehmen, dass jedes positive Beispiel an einem Eckpunkt sitzt – was ja im Worst-case durchaus der Fall ist – haben wir e Elemente in unserer Menge I_{opt}.

Daraus folgt:

$$\begin{aligned} |I_L| &\leq |I_{opt}| \cdot \log |U| \\ &= \mathcal{O}(e \log m) \end{aligned}$$

5.4 Aufgabe: In dieser Aufgabe ... (6 Punkte)
Lösung:
Mein Occam-Algorithmus:

- fordere n Beispiele in Abhängigkeit von ε und δ an
- sortiere die Beispiele (negative wie positive) nach Größe (kleinstes zuerst)
- setze $h = \text{code}(x)$ für das erste positive Beispiel x und speichere Zuordnung
- oder: setze $h = \text{code}(1) \cdot \text{code}(0)$ falls es keins gibt und überspringe den folgenden Schritt
- Wiederhole für jedes Beispiel
  - falls x ein positives Beispiel:
    * wenn das vorherige Beispiel ein positives war, tue nichts
    * falls es ein negatives war, setze $h' = h \cdot \text{code}(x)$ und speichere Zuordnung
  - falls x ein negatives Beispiel:
    * wenn das vorherige Beispiel ein negatives war, tue nichts
    * falls es ein positives war, setze $h' = h \cdot \text{code}(x)$ und speichere Zuordnung
  - setze $h = h'$
- gib h aus.

Nachdem der Algorithmus fertig ist, liegen die Zahlen kodiert vor als $\text{code}(a_1) \cdot \text{code}(b_1) \cdot \text{code}(a_2) \cdot \text{code}(b_2) \cdots \text{code}(a_s) \cdot \text{code}(b_s)$. Dabei haben wir höchstens $2n$ Elemente in unserer Hypothese gespeichert (das ist auch die VC-Dimension dieser Konzeptklasse) und somit Laufzeit $\text{poly}(n)$. Auch das überprüfen der Hypothese ist kein Problem, denn wir haben immer 2 Zahlen gespeichert und durch ihre besondere Kodierung auch deren Länge.