1 Ograniczenia podstawowe

1.1 Entropia zbiorów

Wiele zagadnień uczenia maszynowego wygląda następująco: mamy dany zbiór treningowy par (x_i, y_i) gdzie x_i to zmienne niezależne zaś $y_i = f(x_i)+e_i$ to odpowiedzi które chcemy przewidzieć (tzn. chcemy przewidzieć $f(x_i)$ zaś e_i to błąd). Nasze zadanie to znalezienie funkcji g tak by dla x z z interesującego nas zbioru S funkcja $g(x)$ było dobrym przybliżeniem do $f(x)$. Np. jeśli na zbiorze wartości zadana jest metryka to można żądać by dla pewnego ε i każdego $x \in S$

$$d(f(x), g(x)) < \varepsilon$$

(w zastosowanich wystarcza nierówność nieostra, teoria jest nieco prostsza gdy używamy nierówność ostrą).

Na funkcjach można wprowadzić metrykę wzorem

$$d(f, g) = \sup_{x \in S} d(f(x), g(x))$$

i przy takiej notacji warunek wyżej sprowadza się do

$$d(f, g) < \varepsilon.$$

Metrykę na funkcjach wprowadzoną wyżej zwykle nazywa się metryką jednostajną. Można też rozważać inne metryki na funkcjach np. jeśli na zbiorze S jest zadana miara to dla $p \geq 1$ można rozpatrywać metryki typu L^p:

$$d(f, g) = \left(\int_{x \in S} d(f(x), g(x))^p d\mu(x) \right)^{1/p}$$

Dla ustalenia uwagi dalej będziemy zakładać że S jest podzbiorem $\mathbb{R}^m$ dla pewnego m zaś zbiór wartości jest podzbiorem R.

Zagadnienie znalezienia g jak wyżej to w zasadzie klasyczne zagadnienie aproksymacji czy interpolacji funkcji. Jednak klasyczne metody aproksymacji działają niezbyt dobrze dla dużych m, dlatego do uczenia maszynowego zwykle stosuje się inne metody.

A priori dla pojednyczej funkcji f nie widać ograniczeń dla możliwą jakość aproksymacji. Lecz naturalne jest żądanie by nasza metoda działała nie dla pojedynczego f, ale dla każdego f z pewnego podzbioru K. Tu już widać

ograniczenia: zbiór funkcji które możemy reprezentować w komputerze jest skończony, więc istnieje skończony zbiór $\{g_1, \ldots, g_N\}$ taki że dla każdego $f \in K$ istnieje j takie że

$$d(f, g_j) < \varepsilon$$

Warunek powyżej pojawia się w teorii przestrzeni metrycznych, jeśli jest spełniony to mówimy że zbiór K posiada skończoną ε-sieć. A więc istnienie skończonej ε-sieci jest warunkiem koniecznym do aproksymacji. Należy tu podkreślić że jest to fundamentalne ograniczenie, niezależne zupełnie od metody którą używamy do szukania g. W miarę naturalne jest też żądanie by być może kosztem większego wysiłku dało się znaleźć przybliżenie dla dowolnego $\varepsilon > 0$. W takim przypadku zbiór K musi posiadać skończoną ε-sieć dla dowolnego $\varepsilon > 0$. Taki zbiór nazywamy zbiorem prezwartym. Jeśli dodatkowo K jest zbiorem domkniętym, zaś nasza przestrzeń metryczna jest zupełna to K musi być zbiorem zwartym. Ponadto, jeśli K jest prezwarty zaś zawierająca go przestrzeń jest zupełna to domknięcie K jest zwarte. Teraz widać że jest to istotne ograniczenie: zbiór funkcji ciągłych na odcinku $[0, 1]$ jest przestrzenią metryczną zupełną, ale nie jest zbiorem zwartym (metryka jednostajna którą podaliśmy wyżej to właśnie metryka na przestrzeni funkcji ciągłych).

Ograniczenie się do zbiorów ograniczonych nie pomaga: domknięta kula jednostkowa w przestrzeni funkcji ciągłych nie jest zwarta. Dodajmy że jest to dość ogólne ograniczenie: domknięta kula jednostkowa w przestrzeni wektorowej nad liczbami rzeczywistymi z metryką niezmienniczą na przesunięcia jest zwarta wtedy i tylko wtedy gdy jest przestrzeń jest skończenie wymiarowa. Interesujące przestrzenie funkcji są nieskończenie wymiarowe.

A więc by możliwa była aproksymacja jednostajna zbiór K musi być istotnie mniejszy niż kula jednostkowa. Naturalnym założeniem jest że funkcje z K są regularne. Jeśli założymy że S jest wystarczająco regularnym zbiorem (np. wypukłym) zaś K składa się z funkcji które są wspólnie ograniczone i ich pochodne do pewnego rzędu też są wspólnie ograniczone to wtedy K jest zbiorem zwartym.

Jeśli K jest zbiorem zwartym znika najbardziej podstawowe ograniczenie. Ale nasze ograniczenie moża też sformułować w sposób ilościowy. Jeśli mamy do dyspozycji n bitów pamięci to możemy w tej pamięci reprezentować maksymalnie 2^n funkcji. A więc jeśli każdy element K daje się reprezentować z błędem mniejszym niż ε to K ma ε sieć o mocy nie przekraczającej 2^n. Prowadzi nas to do pojęcia entropii zbioru. Najpierw definujemy $N_\varepsilon(K)$ jako minimalną moc ε-sieci dla K. Następnie definujemy ε-entropię $H_\varepsilon(K)$ jako

$$H_\varepsilon(K) = \log(N_\varepsilon(K)).$$

Logarytm w definicji oznacza że $H_\varepsilon(K)$ jest proporcjonalne do ilości pamięci potrzebnej do reprezentacji elementów K. Naturalne jest pytanie jak duże jest $H_\varepsilon(K)$. Krótka odpowiedź jest taka: $H_\varepsilon(K)$ jest duże, nieco mniejsze niż najbardziej pesymistyczne oszacowanie, ale prawie tak duże. Konkretniej, niech $S = [0,1]^m$ zaś K niech będzie zbiorem funkcji ciągłych f na S takich że dla dowolnego wielowskaźnika α z $|\alpha| \leq k$ zachodzi

$$\sup_{x \in S} |\partial^\alpha f(x)| \leq 1$$

(innymi słowy dla $f \in K$ wszystkie pochodne cząstkowe do rzędu k są ograniczone przez 1). Wtedy dla $\varepsilon < (2(2k+1))^{-k}$

$$H_\varepsilon(K) \geq \log(2)(2(2k+1))^{-m}\varepsilon^{-m/k}$$

Szkic dowodu: Ustalmy $\delta > 0$. Istnieje funkcja $\phi \in C^k$ taka że $\phi(0) = 1$, $|\partial_x^j \phi|(x) \leq ((2k+1)\delta^{-1})^j$ dla $j = 0, \ldots, k$ i $\phi(x) = 0$ dla $x \notin (-\delta/2, \delta/2)$. Niech $a \in \mathbb{Z}^m$ i

$$\psi_a(x) = \varepsilon \Pi_{i=1}^m \phi(x_i - \delta a(i)).$$

Dla $|\alpha| \leq k$ mamy

$$|\partial^\alpha \psi_a(x)| = \varepsilon \Pi_{i=1}^m |\partial^{\alpha(i)} \phi(x_i - a(i))|$$

$$\leq \varepsilon \Pi_{i=1}^m ((2k+1)\delta^{-1})^{\alpha(i)} = \varepsilon((2k+1)\delta^{-1})^{|\alpha|}$$

$$\leq \varepsilon((2k+1)\delta^{-1})^k.$$

A więc $|\partial^\alpha \psi_a(x)| \leq 1$ o ile

$$\varepsilon((2k+1)\delta^{-1})^k \leq 1.$$

Lecz to zachodzi gdy

$$\varepsilon^{1/k}(2k+1) \leq \delta.$$

Niech $\Lambda = \{a \in \mathbb{Z}^m : 0 \leq a(i) \leq \delta^{-1}\}$, niech $c : \Lambda \to \{-1, 1\}$ i

$$u_c(x) = \sum_{a \in \Lambda} c_a \phi_a(x)$$

Łatwo zauważyć że ϕ_a mają nośniki rozłączne, więc

$$|\partial^\alpha u_c(x)| \leq \max_{a \in \Lambda} |c_a| |\partial^\alpha \phi_a(x)| \leq 1$$

czyli $u_c \in K$. Jeśli $c, d \in \Lambda$, $c \neq d$ to

$$\sup_{x \in S} |u_c(x) - u_d(x)| \geq \sup_{x \in \Lambda} \varepsilon |c_a - d_a| = 2\varepsilon$$

Moc Λ to l^m gdzie l to największa liczba całkowita mniejsza lub równa δ^{-1}. A więc K zawiera 2^{l^m} elementów takich że każde dwa są odległe co najmniej o 2ε. Wynika stąd że ε sieć dla K ma co najmniej 2^{l^m} elementów, czyli $H_\varepsilon(K)$ to co najmniej $\log(2)l^m$. Lecz dla $\varepsilon < (2(2k+1))^{-k}$ biorąc

$$\delta = \varepsilon^{1/k}(2k+1)$$

mamy $\delta < 1/2$, czyli $l > (2\delta)^{-1}$ czyli

$$H_\varepsilon(K) \geq \log(2)(2\varepsilon^{1/k}(2k+1))^{-m} = \log(2)(2(2k+1))^{-m}\varepsilon^{-m/k}$$

Zauważmy że dla $\varepsilon < (2(2k+1))^{-k}$ mamy $l \geq 2$ czyli

$$H_\varepsilon(K) \geq \log(2)2^m$$

a więc potrzebujemy co najmniej 2^m bitów do reprezentacji elementów K. Oznacza to że dla większych m reprezentacja elementów K z dużą dokładnością jest praktycznie niemożliwa bo 2^m jest zbyt duże.

Jest też podobne oszacowanie z góry

$$H_\varepsilon(K) \geq b(m,k)\varepsilon^{-m/k}$$

gdzie czynnik $b(m,k)$ zależy wykładniczo od m.

Nasze oszacowanie z dołu jest stosunkowo słabe dla dużych ε, jednakże można oczekiwać że entropia ciągle jest duża.

Co stąd wynika dla uczenia maszynowego: ogólne funkcje wielu zmiennych są beznadziejnie trudne dla obliczeń komputerowch. To że różne metody uczenia maszynowego działają oznacza że funkcje z którymi mamy do czynienia są wyjątkowo łatwe, w szczególności zwykle istotna część problemu może być zredukowana do stosunkowo niskiego wymiaru.

Dodajmy że podobne wyniki zachodzą dla innych norm, więc nasze oszacowanie to nie artyfakt z powodu specjalnego wyboru K.

1.2 Złożoność obliczeniowa

Inną fundamentalną przeszkodą jest to że problemy które chcemy rozwiązywać w wielu wypadkach przy naturalnej formalizacji mają bardzo wysoką złożność obliczeniową.

1.3 Komputery kontra ludzie

Wiele z zadań gdzie stosuje się metody uczenia maszynowego jest lub w przeszłości było rozwiązywane przez ludzi. Duża złożoność o której mówiliśmy w poprzednich podrozdziałach prowadziła to stwierdzeń że komputery

nigdy nie będą w stanie ich rozwiązywać. W takich stwierdzeniach jest fundamentaly błąd. Mianowicie, przypisują one ludziom magiczne możliwości rozwiązywania bardzo trudnych problemów. Jednakże badania mózgu dość mocno sugerują że mózg ludzki nie ma magicznych możliwości. Obecne oszacowania mocy obliczeniowej mózgu są dość niedokładne, ale w miarę rozsądne jest oszacowanie Kurzweila że mózg potrafi wykonać ekwiwalent 10^{15} operacji zmiennopozycyjnych na sekundę. Jest to dużo więcej niż popularne komputery PC, lecz mniej niż współczesne superkomutery. Oszacowania pojemności ludzkiej pamięci są jeszcze mniej dokładne niż oszacowania mocy obliczeniowej, lecz nawet gdyby ludzie mieli większą pamięć od komputerów to fundamentaly problem pozostaje: potrzebna pamięć rośnie wykładniczo z wymiarem.

A więc najrozsądniejszym wytłumaczeniem jest to że problemy które ludzie rozwiązują są znacznie łatwiejsze od ogólnych problemów.

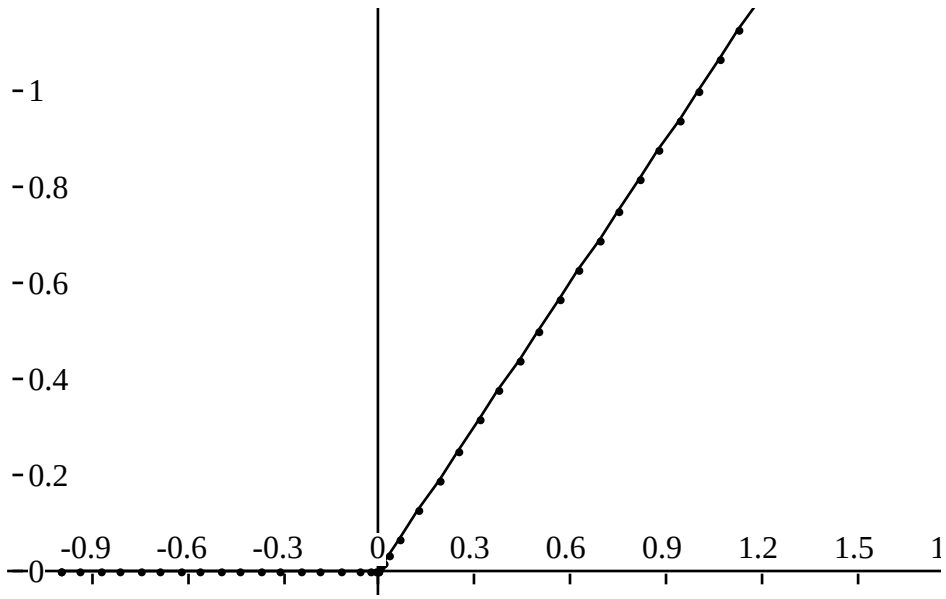
2 Ograniczenia strukturalne

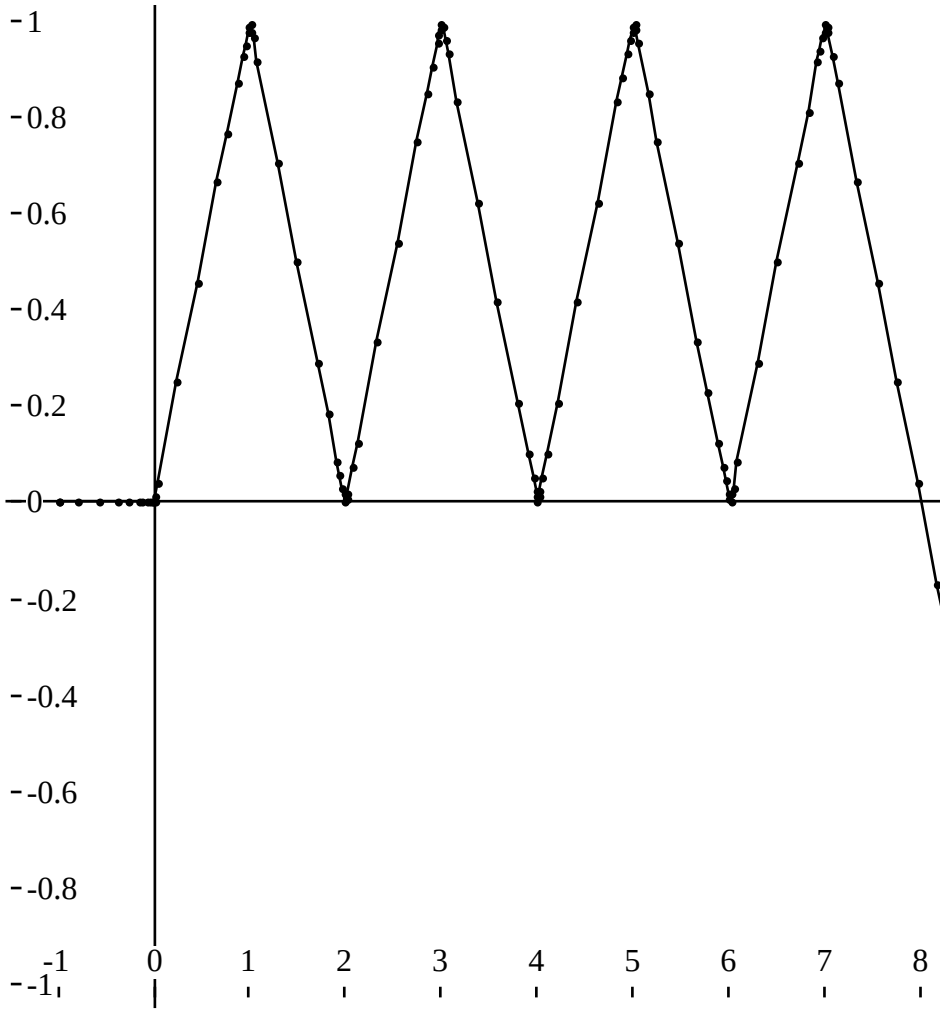
Wiele metod statystycznych jest liniowych lub podobnych do liniowych. Również sieci neuronowe to kombinacja przekształceń liniowych ze stosunkowo prostymi operacjami nieliniowymi. Można podejrzewać że tego typu metody są istotnie słabsze od podejść w pełni uwzględniających nieliniowość. Aby uzasadnić te podejrzenia warto popatrzeć na sieci boolowskie (logiczne). Ogólną sieć logiczną można sobie wyobrazić jako szereg warstw. Każda warstwa na węzły zawiejające podstawe operacje takie jak iloczyn logiczny, suma czy negacja. Dana warstwa bierze sygnały wejściowe z poprzednich warstw, zaś wyniki są używane przez następne warstwy. Oczywiście wejście do piewszej warstwy jest równocześnie wejściem do całej sieci, wyjście z ostatniej warstwy jest wyjściem sieci. W technice jesteśmy zainteresowani szybkością działania sieci. Przy ustalonej szybkości elementów szybkość sieci zależy od ilości warstw, po prostu sygnał z wejścia musi przejść przez wszystkie warstwy. To naturalnie prowadzi do pytania ile warstw jest potrzebne? Prosta klasyczna odpowiedź mówi że wystarczą trzy warstwy: warstwa negacji, warstwa iloczynów i warstwa sum. W praktyce często stosuje się elementy z wbudowną możliwością negacji wejścia, wtedy wystarczą dwie warstwy. Lecz okazało się że stosunkowo proste funkcje logiczne, które wymagają niewielu elementów w realizacji wielowarstwowej wymagają wykładniczo wielu elementów przy realizacji z małą ilością warstw. Przykładem jest n-wejściowa funkcja nieparzystości (XOR): zwraca prawdę gdy nieparzysta liczba wejść jest prawdą, fałsz gdy parzysta licza wejść jest prawdą. Tą funkcję łatwo zrealizować używając $n-1$ dwuwejściowych funkcji nieparzystości. Ustawia-

jąc je w drzewo które jest w przybliżeniu zbalansowane wystarcza $\log_2(n)$ warstw.

Sieci neuronowe różnią się od sieci logicznych, wielowejściową funkcję nieparzystości można zrealizować używając ustaloną liczbę warstw i niezbyt dużą liczbę elementów: w pierwszej warstwie obliczeniowej dajemy pojedynczy neuron który zlicza (sumuje) ilość wejść które są równe 1. Następnie dwie warstwy obliczniowe mogą obliczyć funkcję która jest 1 dla nieparzystych k i zerem dla parzystych k dla k pomiędzy 0 a n. Dokładniej, używając funkcję aktywacji $\phi(x) = x_+$ (zwaną też RELU) budujemy wyrażenie

$$\phi(x) + 2\sum_{i=1}^{n-1}(-1)^i\phi(x-i)$$





To działa bo sieć neuronowa może obliczać wielowejściową sumę, ale można się spodziewać że będą kłopoty z bardziej skomplikowanymi funkcjami.

Wielowarstwowe sieci neuronowe teoretycznie rozwiązują ten problem. Jednakże niedawne wyniki doświadczalne sugerują że również sieci wielowartwowe mogą mieć ograniczenia. Dokładniej, chcemy stworzyć sieć jako efekt uczenia. Nie jest jasne czy obecne metody uczenia są wystarczające w przypadku trudniejszych problemów.

Przykładem są tu złośliwe dane dla sieci: stosując proces podobny do

uczenia sieci udało się uzyskać obrazki które różnią się bardzo mało od jednego wzorca, lecz sieć rozpoznaje je jako inny wzorzec. Innym przykładem są experymenty z użyciem sieci neuronowych do wykonania prostych obliczeń. Udało się nauczyć sieć dodawania prostych liczb (reprezentowanych na wejściu jako ciągi znaków), ale przy bardziej skomplikowanych liczbach czy większej ilości składników sieć robi błędy.

Doświadczenia z rozpoznawaniem stosunkowo prostych kształtów geometrycznych pokazały że sieci preferują liniowe kryteria lub kryteria bliskie linowych nawet wtedy gdy proste ale dość mocno nieliniowe kryterum dałoby zawsze poprawny wynik.