LINEAR MODELS - LINEAR REGRESSION

Linear regression er en linear model som forsøger, at finde en hyperplane i vore data, som fortæller os et tal givet en data vektor. Imodsætning til classification, hvor resultatet er binært, vil linear regression give os en $\mathbb{R}$ værdig. Dette kunne fx være hvor meget skal en bank låne en kunde. Antagelsen man skal gøre, for at bruge linear regression er, at der findes en linear kombination af informationer, som kan approximere, hvad vi gerne vil approximere.

Denne form for læring bruger squared error imellem $h(\mathbf{x})$ og y til at estimere E_{out}:

$$E_{out}(h) = \mathbb{E}\left[h(\mathbf{x} - y)^2\right]$$

Hvor den forventede værdi er taget ift. den forenede sandsynligheds fordeling $P(, y)$. Mået er selfølgelig at opnå en så lille $E_{out}(h)$ som muligt. Vi finde $E_{in}(h)$ ved:

$$E_{in}(h) = \frac{1}{N}\sum_{n=1}^{N}(h(\mathbf{x_n}) - y_n)^2$$

hvor $x_0 = 1$ og $\in \{1\} \times \mathbb{R}^d$ og $\mathbf{w} \in \mathbb{R}^{d+1}$. Når vi har med et *lineært h* at gøre, er det meget brugbart at have en matrix repræsentation af $E_{in}(h)$. Definer $X \in \mathbb{R}^{N\times(d+1)}$, hvor hver row er et input $\mathbf{x}_n$ også har vi:

$$E_{in}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^T\mathbf{x}_n - y_n)^2 = \frac{1}{N}||X\mathbf{w} - \mathbf{y}||^2 = \frac{1}{N}(\mathbf{w}^T X^T X\mathbf{w} - 2\mathbf{w}^T X^T\mathbf{y} + \mathbf{y}^T\mathbf{y})$$

Linear regression algoritmen fås ved at minimere $E_{in}(\mathbf{w})$, og alle mulige $\mathbf{w} \in \mathbb{R}^{d+1}$. Derfor er vi interesseret i følgende optimeserings problem:

$$\mathbf{w}_{lin} = \underset{\mathbf{w}\in\mathbb{R}^{d+1}}{\arg\max} E_{in}(\mathbf{w})$$

Siden $E_{in}(\mathbf{w})$ er differentiabel, kan vi finde dens gradient ved, og løse den for $\nabla E_{in}(\mathbf{w}) = \mathbf{0}$

$$\nabla E_{in}(\mathbf{w}) = \frac{2}{N}(X^T X\mathbf{w} - X^T\mathbf{y})$$

For at løse $\nabla E_{in}(\mathbf{w}) = \mathbf{0}$, finder vi et $\mathbf{w}$ der opfylder

$$X^T X\mathbf{w} = X^T\mathbf{y}$$

Hvis $X^T X$ er invertible, hvilket det er i de fleste tilfælde er, kan vi finde den unikke optilmale løsning for $\mathbf{w}$ ved

$$\mathbf{w} = (X^T X)^{-1}X^T\mathbf{y} = X^\dagger\mathbf{y}$$

som er vores $\mathbf{w}_{lin} = X^\dagger\mathbf{y}$. Denne giver vores hypotese som esitmere $\mathbf{y}$, som $\hat{\mathbf{y}} = X\mathbf{w}_{lin}$, som afviger fra $\mathbf{y}$ grundet in-sample errors.

Fører dette til en god E_out? Det korte svar er ja, hvor det gælder at

$$E_{out}(g) = E_{in}(g) + O\left(\frac{d}{N}\right) \tag{0.1}$$

LIENAR MODELS - PERCEPTRON LEARNING ALGORITHM

Perceptron er en learning algoritme der bruges tli linear clasification af data, igennem iterationer hvor man opdatere vægtninge af forskellige d-dimensionel data, så nogle dimensioner er vigtigere end andre.

Vi bruger algoritmen til, at automatisere classification problemer, eller problemer hvor der skal afgøres noget med binære muligheder.

Vi specificere en hypothese $\mathscr{H}$, og finder vores hypothese $h \in \mathscr{H}$. Denne defineres som

$$h(x) = sign\left(\left(\Sigma_{i=1}^{d} w_i x_i\right) + b\right) = sign(\mathbf{w}^T\mathbf{x}) \in \{0,1\}$$

Igennem $t = 0,1,2,...$ iterationer, opdatere vi de forskellige weights for *misclassified data* ved.

$$\mathbf{w}(t+1) = \mathbf{w}(t) + y(t)\mathbf{x}(t)$$

Værdierne for $\mathbf{w}$ er initialiseret til at være forskellige *tilfældige* værdiger. Vi har at $w_0 = b$ og $\mathbf{w} = [w_0, w_1, ..., w_d]^T \in \mathbb{R}^{d+1}$ og vores space for x er

$$\mathscr{X} = \{1\} \times \mathbb{R}^d = \{[x_0, x_1, ..., x_d]^T | x_0 = 1, x_1, ..., x_d \in \mathbb{R}^\}$$

Efter t itertioner vil PLA stoppe og vi har at $E_{in}(\mathbf{w}_{PLA}) = 0$, *hvis* dataen er linear seperable! Hvis dataen ikke er linear seprable skal vi approximere en løsning, fx igennem pocket algorithm, som for hver gang vi finder en$E_{in}() < E_{in}(\mathbf{w})$, gemmer $\hat{\mathbf{w}}$, og iterere videre, hvor den efter et valgt t iterationer stopper og returnerer den bedst observerede $\mathbf{w}$. Det man giver afkald på med pocket algorithm er ift PLA som opdatere værdier i $\mathbf{w}$ og checker for nogle exempler, de misclassified, bliver pocket algorithm nødt til, at checke for alle, for at kunne berenge $E_{in}(\mathbf{w})$.

Det smarte ved PLA, er at den iterere igennem et uendeligt stort hypothese space, i (beviseligt) endelig tid.

LINEAR MODELS - LOGISTIC REGRESSION

Logistic regression er en linear model som forsøger at fortælle os sandsynligheden for et given indput finder sted. Dette kunne være sandsynligheden for en patient får et hjertestop, givet hans journal. Grunden til denne form for model er, imodsætningen til linear classification som har en *hard threshold* og linear regression som har *no threshold*, så har linear regression et *soft threshold*, mellem $[0-1]$.

Vi definere vores hypothese h i linear regression så ledes

$$h(\mathbf{x}) = \theta(\mathbf{w}^T\mathbf{x})$$

hvor θ er en såkaldt logistisk function, fx $\theta(s) = \dfrac{e^s}{1+e^s}$ eller $\theta(s) = tanh(s) = \dfrac{e^s - e^{-s}}{e^s + e^{-s}}$.

Vores target function vi prøver at ville lære er defineret som $f(\mathbf{x}) = \mathbb{P}[y = +1|\mathbf{x}]$, men det vi kigger på er en mængde data, hvor vi kender udfaldet, som er generet af et *noisy target* $P(y|\mathbf{x})$

$$P(y|\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{for } y = +1 \\ 1 - f(\mathbf{x}) & \text{for } y = -1 \end{cases}$$

Derfor må vi definere en *error measure* som måler hvor tæt hypothese h er på f, ift vores *noisy* ± 1 eksempler. Måle formen for fejl er $e(h(\mathbf{x}), y)$ som er baseret på *likelihood*, altså hvor sandsynligt y er givet $\mathbf{x}$. Vi akn sunstituere $h(\mathbf{x})$ til $\theta(\mathbf{w}^T\mathbf{w})$ og bruger det faktum at $1 - \theta(s) = \theta(-s)$ og får

$$P(y|\mathbf{x}) = \theta(y\mathbf{w}^T\mathbf{w})$$

Vi definere på samme tid vores $E_{in}(\mathbf{w})$ til

$$E_{in}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln\left(1 + e^{-y_n\mathbf{w}^T\mathbf{x}_n}\right)$$

Dette betyder vores *pointwise error measure* er $e(h(\mathbf{x}_n), y_n) = \ln(1 + e^{-y_n\mathbf{w}^T\mathbf{x}_n})$, hvor vi kan se at vores *error measure* er lille når $y_n\mathbf{w}^T\mathbf{x}_n$ er stor og *positiv*, hvilket ville betyde at $sign(\mathbf{w}^T\mathbf{x}_n) = y_n$

For at træne logistisk regression vil vi forsøge at sette $\nabla E_{in}(\mathbf{w}) = \mathbf{0}$, dette er dog et *svært* problem. Derfor løses det med en iterativ algoritme *gradient descent*. En stor fordel for denne tilgang er, grundet vores logistiske regression benytter cross-entropy error, er vores function vi skal descente ned af convex, og vi vil derfor altid gå mod et globalt minimum. Vi øsnker at descente ned, det stejleste vej, for at nå vores mål hurtigts, så vi definere η til at være vores step-size, og $\hat{\mathbf{v}}$ til at være en enhedsvektor. Vi har derfor

$$\begin{aligned} \delta E_{in} &= E_{in}(\mathbf{w}(0) + \eta\hat{\mathbf{v}}) - E_{in}(\mathbf{w}(0)) \\ &= \eta\nabla E_{in}(\mathbf{w}(0))^T\hat{\mathbf{v}} + O(\eta^2) \\ &\geq -\eta||\nabla E_{in}(\mathbf{w}(0))|| \end{aligned}$$

hvor der gælder lighedstegn i sidste linje $\Longleftrightarrow \hat{\mathbf{v}} = \dfrac{\nabla E_{in}(\mathbf{w}(0))}{||\nabla E_{in}(\mathbf{w}(0))||}$. Det er dog ikke en god idé at fasthold η igennem alle iterationer, da det giver god mening at tage mindre skridt, jo tættere vi kommer på det globale minimum, så denne kan justeres hen af vejen. Dette fører os til algoritmen for logistik regression

1. initialiser weigths ved tid $t = 0$ til $\mathbf{w}(0)$
2. **for** $t = 0, 1, 2, ...$**do**
   a) beregn gradienten
   $$\mathbf{g}_t = -\frac{1}{N}\sum_{n=1}^{N} \frac{y_n \mathbf{x}_n}{1 + e^{y_n \mathbf{w}^T(t)\mathbf{x}_n}}$$
   b) sæt bevægelses retning, $\mathbf{v}_t = -\mathbf{g}_t$
   c) updater weigths: $\mathbf{w}(t+1) = \mathbf{w}(t) + \eta \mathbf{v}_t$
   d) iterer til næste step intil tiden stopper
3. Returner final weigth $\mathbf{w}$.

Ved kørsel af algortimen sættet $\mathbf{w}(0)$ til tilfældige tal, for at undgå vi sidder fast på en symmetrisk top, og vi vælger selv tid t for hvornår algoritmen skal stopper. Dette kan være når $||\mathbf{g}_t||$ er mindre end et bestemt kriterium, eller blot efter en arbitrær tid t.

En anden udgave af gradient descent delen af vores algoritme er stochastic gradient descent, hvor der vælges et tilfældigt punkt, som vi opdatere vores $\mathbf{w}$ ud fra, is tedet for at gøre det på hele data mængden, hvilket gør den af algoritmen hurtigere.

LEARNING THEORY - VC DIMENSION

Vi introducere ideen om *growth function*, og til det skal vi bruge dichotomies

DEFINITION 2.1. Lad $\mathbf{x}_1, ..., \mathbf{x}_N \in \mathscr{X}$. Dichotomies generaret af $\mathscr{H}$ for disse points er defineret som :

$$\mathscr{H}(\mathbf{x}_1, ..., \mathbf{x}_N) = \{(h(\mathbf{x}_1), ..., h(\mathbf{x}_N)) | h \in \mathscr{H}\}$$

disse kan ses et par briller hvor igennem vi kigge på hele $\mathscr{H}$ rummet igennem N punkter og kan se forskel på to h hvis de er forskellige ift deres vurdering af $\mathbf{x}_1, ..., \mathbf{x}_N$. En growth function er defineret på antallet af dichotomies

DEFINITION 2.2. Growth function er defineret for en hypothese mængde $\mathscr{H}$ ved

$$m_{\mathscr{H}} = \max_{\mathbf{x}_1, ..., \mathbf{x}_N \in \mathscr{X}} |\mathscr{H}(\mathbf{x}_1, ..., \mathbf{x}_N)|$$

altså er $m_{\mathscr{H}}$ det største antal dichotomies der kan generes af $\mathscr{H}$ for alle N points. Hvis $\mathscr{H}(\mathbf{x}_1, ..., \mathbf{x}_N) = \{0,1\}^N \implies m_{\mathscr{H}}(N) = 2^N$, og vi siger at $\mathscr{H}$ kan *shatter* $\mathbf{x}_1, ..., \mathbf{x}_N$.

DEFINITION 2.3. Hvis intet dataset af størrelse k kan *be shattered* af $\mathscr{H}$, så er k et *breakpoint* for $\mathscr{H}$.

Hvis definition 2.3 gælder, så ser vi at $m_{\mathscr{H}}(k) < 2^k$.

THEOREM 2.4 Hvis $m_{\parallel} < 2^k$ for en værdi k, så

$$m_{\mathscr{H}}(N \leq) \sum_{i=0}^{k-1} \binom{N}{i}$$

for alle N.

hvilket fører os til

DEFINITION 2.5 Vapnik-Chervonenkis dimensionen af en hypothese mængde $\mathscr{H}$, noteret ved $d_{VC}(\mathscr{H})$ eller d_{VC}, er den største værdi N, der gælder at $m_{\mathscr{H}}(N) = 2^N$. Hvis $m_{\mathscr{H}}(N) = 2^N$ for alle N, så er $d_{VC}(\mathscr{H}) = \infty$.

Grunden til at VC dimensioner er vigtige er, både at de angiver hvor mange *degrees of freedom* vi har at arbejde med i vores hypothese mænde men også, at hvis vi ikke kan finde et data set der kan shatter $m_{\mathscr{H}}$ har vi

THEOREM 2.5 (VC GENERATLIZATION BOUND). For er vilkårlig tolerance $\delta > 0$,

$$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{8}{N} \ln \frac{4m_{\mathcal{H}}(2N)}{\delta}}$$

Med sandsynlighed $\geq 1 - \delta$. Vi definere også $\sqrt{\frac{8}{N} \ln \frac{4m_{\mathcal{H}}(2N)}{\delta}} = \omega(N, \mathcal{H}, \delta)$, som værende modellens complexitet, altså som en slags penalty for hvor complex vores model er, og som straffer os med ringere E_{out}.

Vi ser hurtigt at hvis $m_{\mathcal{H}} = \infty$ har vi ingen garanti for at vi kan lære noget givet.

Vi har altså fået givet med VC analyses, at vi skal vores valg af $\mathcal{H}$ skal finde en balance imellem at approximere f på trænings daten, og generalisere over nyt data.

Lad os definere $\bar{g} \approx \frac{1}{K}\sum_{k=1}^{K} g_k(\mathbf{x})$ for alle $\mathbf{x}$. Givet out of sample error på squared error, har vi

$$\begin{aligned}\mathbb{E}_{\mathscr{D}}[E_{out}(g^{(\mathscr{D})})] &= \mathbb{E}_{\mathbf{x}}[\mathbb{E}_{\mathscr{D}}[g^{(\mathscr{D})}(x)^2] - 2\bar{g}(\mathbf{x})f(\mathbf{x}) + f(\mathbf{x})^2] \\ &= \mathbb{E}_{\mathbf{x}}[\underbrace{\mathbb{E}_{\mathscr{D}}[g^{(\mathscr{D})}(\mathbf{x})^2] - \bar{g}(\mathbf{x})^2}_{\mathbb{E}_{\mathscr{D}}[(g^{(\mathscr{D})} - \bar{g}(\mathbf{x}))^2]} + \underbrace{\bar{g}(\mathbf{x})^2 - 2\bar{g}(\mathbf{x})f(\mathbf{x}) + f(\mathbf{x})^2}_{(\bar{g}(\mathbf{x}) - f(\mathbf{x}))^2}]\end{aligned}$$

Det sidste term kalder vi for bias

$$\text{bias}(\mathbf{x}) = (\bar{g}(\mathbf{x}) - f(\mathbf{x}))^2$$