Cornell Machine Learning Notes No. 1

Machine Learning – Algorithms that improves themselves on some task with experience.

– Based on Statistics and Optimization, not Logic.

– Supervised learning, Unsupervised Learning. reinforcement learning.

Terminology.

Setup: $D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2) \cdots (\vec{x}_n, y_n)\} \subseteq \mathcal{X} \times \mathcal{Y}$.

$(\vec{x}_i, y_i) \sim P$ → we never know.

i.i.d →

$\mathcal{X} = \mathbb{R}^d$

→ $f(x_i) \to y_i$. → classification

→ regression.

Remember, the training data and test data must be drawn from the same distribution.

Cornell Machine Learning Notes No. 2.

$D = \{\dots\} \to$ ML Algorithm $\to h \Rightarrow h(\vec{x}_i) \simeq y_i$

$\vec{x}_{new} \to h \to y_{new}$.

$h \in \mathcal{H}$ → hypothesis class. → need to be chosen by human.

then we find the best h in the $\mathcal{H}$ we choosed.

↙

how? → loss functions.

- 0/1 - loss $L(h, D) = \frac{1}{n} \sum_{(\vec{x}_i, y_i)} \delta[h(\vec{x}_i) \neq y_i]$
- squared loss $L(h, D) = \frac{1}{n} \sum (h(\vec{x}_i) - y_i)^2$
- absolute loss $L(h, D) = \frac{1}{n} \sum |h(\vec{x}_i) - y_i|$

Learning process: finding $h \in \mathcal{H}$ with small loss.

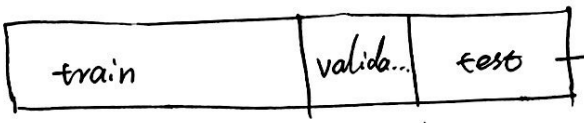
Generalization: $h \in \mathcal{H}$ s.t. $\forall (\vec{x}, y) \sim P$, $h(\vec{x}) \simeq y$.

We want $E[L(h, (\vec{x}, y))]_{(\vec{x}, y) \sim P}$ to be small.

but we don't have acess to P, thus we use a

test set to estimate it. this E.

Cornell Machine Learning Note No. 3.

D = 

→ can only be used to finally test the model, not to choose between models. or you will be overfitting the test set.

From Law of Large Numbers.

$$\uparrow$$

$$l(h, D_{test}) = \frac{1}{n}\sum l(h(\vec{x}_i, y_i)) \xrightarrow{n\to\infty} E[l(h(\vec{x}, y))]$$

How to choose $\mathcal{H}$? Is there a best $\mathcal{H}$?

- We have to make assumpeions on the data, otherwise we can not learn any model of it.
- Different $\mathcal{H}$ makes different assumptions.
- Then we can generate a dataset that violates the assumptions and the model will perform arbitrarily bad on this set.

=> "No ~~Lunch~~ Free Lunch" theorm. there is no one best algorithm that is best at all Datasets.

Note: The algorithms works best when their assumpeion are met.

Cornell Machine Learning Note. No. 4.

K-nearest neighbors. : data points that are close together have the same labels.

For test point x.

$S_x \subseteq D$. $|S_x| = k$. $\forall (x', y') \in D - S_x$. $dist(x, x') \geq \max_{x'' \in S_x} dist(x, x'')$

(x'', y'') in S_x vote $\rightarrow y$.

the key is to use proper distance metric.

$$dist(x, z) = \left(\sum_{r=1}^{d} |[x]_r - [z]_r|^p\right)^{\frac{1}{p}}$$

$p = 1$ -> Manhatton

$p = 2$ -> Euclidean

$p \to \infty$ -> Maximum.

Curse of dimensionality.

In very high dimensional space, almost all the points will be on the edges and be far away from each other. There will not be such thing as "neighbor". One point might be a little close to the test point than another, but that means nothing.

Cornell Machine Learning Notes. No. 5.

k-nearest neighbor suffers a lot from the curse of dimensionality but people are still using it to classify images. why?

The truth is, image data often lies in a ~~sub~~ small subspace of the seemingly high-dimensional space.

In low dimensional case, as n gets larger, k-nearest neighor can be very very accurate, but that will take too long. [$O(n^d)$]

Perceptron → the first machine learning algorithm.

Assumption: the dataset is linearly separable.

↳ almost always hold in high-dimensional space.
seldoms hold in low-dimensional space.

Unlike k-nearest Neighbor, perceptron prefer high-dimensional spaces.

$$\mathcal{H} = \{ x,\ w^T x + b = 0 \}.$$

$y = \text{sign}(w^T x + b)$ testing is fast for all possible points.

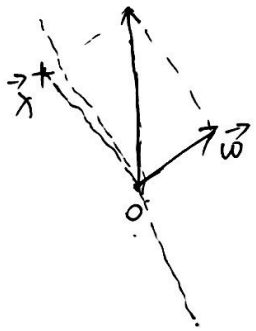
Cornell Machine Learning Notes No. 6.

$Y = \{-1, +1\}$. to learn w, b.

$x_i \to \begin{pmatrix} x_i \\ 1 \end{pmatrix}$, $w \to \begin{pmatrix} w \\ b \end{pmatrix}$ $\left(w'^T x_i' = w^T x_i + b \right)$

$H = \{ x.\ w^T x = 0 \}$. to learn w.

Learning: $\vec{w} = \vec{0}$



While True:

$m = 0$

for every $(x, y) \in D$:

if $y w^T x \le 0$:

$w = w + yx$

$m = m + 1$

if $m = 0$:

break.

END.

For $y = +1$. $w^T x \le 0$

$w = w + yx = w + x$

$w^T x \to (w + x)x = w^T x + x^T x > w^T x$.

..... $w^T x > 0$

For $y = -1$ $w^T x \ge 0$

$w = w - x$

$w^T x \to w^T x - x^T x < w^T x$

$w^T x \le 0$.

Cornell Machine Learning Notes No. 7.

- $y = +1 \Rightarrow w^T x > 0$.
  $y = -1 \Rightarrow w^T x < 0$
  $\Big\}$ $yw^T x > 0$ or, update $w = w + yx$.

- Prove: this algorithm will end in ~~de~~ finite steps..

  $\exists w^*$. s.t. $\forall (x, y) \in D$, $yw^T x > 0$., we can assume $\|w^*\| = 1$ and $\|x_i\| \leq 1$ (since and we can rescale w. and x without changing the sign of $yw^T x$)

  ✓ High level intuion: if we can show that during iterations, $w^T w^*$ becomes ~~more and mo~~ larger and larger, while $w^T w$ doesn't ~~do~~ grow much, we can say that w is getting closer and closer to w^*, and they may eventually define the same hyperplane.

  ✓ when update: $w \to w + yx$ $yw^T x \leq 0$.

$$w^T w^* \to (w + yx) w^* = w^T w^* + \underbrace{yw^{*T} x}_{>0} \geq w^T w^* + \gamma$$

Margin: $\gamma = \min_{(x,y) \in D} |x^T w^*|$

$$w^T w \to (w + yx)^T (w + yx) = w^T w + \underbrace{2yw^T x}_{\leq 0} + \underbrace{x^T x}_{\leq 1} \leq w^T w + 1$$

After M updates: $M\gamma \leq w^T w^* \leq \underbrace{\|w^*\| \|w^*\|}_{=1} \leq \sqrt{w^T w} \leq \sqrt{M}$ $\Rightarrow M \leq \frac{1}{\gamma^2}$

Cornell Machine Learning Notes. No. 8

$M \le \frac{1}{\gamma^2}$, => datasets with large Margin will converge faster.

If we can estimate $p(x,y)$, then we can get $p(y|x)$, then we can make predictions..

How to estimate it? Maximun Likelihood Estimation.

A unique, unfair coin, flip 10 times. got. 4H + 6 T.

MLE: $p(D;\theta) = \text{Constant} \cdot \theta^4 \cdot (1-\theta)^6$

$p(H)$ => $p(H) = \arg\max_\theta p(D;\theta)$

~~$\frac{dp(D;\theta)}{d\theta}$~~ $= \arg\max_\theta \log p(D;\theta)$

$= \arg\max_\theta [4\log\theta + 6\log(1-\theta)]$

$$\frac{2(4\log\theta + 6\log(1-\theta))}{2\theta} = \frac{4}{\theta} + \frac{6}{1-\theta} = 0 \quad \Rightarrow \theta = \frac{4}{4+6}$$

$$\Rightarrow p(H) = \frac{n_H}{n_H + n_T}$$

This seems reasonable, but imagine we have little data and they are all tails, we will get $p(H)_{\text{estimated}} = 0$, which is clearly not right.

How to get around this? Smoothing.

If the dataset is relatively small, we don't trust it very much, instead we pretend that we have got n_H' of Heads and n_T' of tails before. Finally we will get $P(H) = \frac{n_H + n_H'}{n_H + n_T + n_H' + n_T'}$

Typically, n_H', n_T' is set to be one.

This seems reasonable, there is a very nice way to deriving this.
↓
MAP

—

frequency view and bayesian view.

↙ $P(D;\theta)$ → which θ makes the data most possible? θ is a parameter.

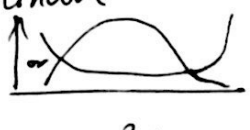
↙ $P(D|\theta)$ → θ is a random variable. a $P(\theta)$ can be assumed. this distribution encodes your prior believe about θ.

—

$P(D|\theta)$ → likelihood. $\geq$ $P(\theta|D)$ → posterior.

$P(\theta)$ → prior. "Given the data, which is the most likely θ?"

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)} = \text{Constant} \cdot \text{prior} \cdot \text{likelihood}$$

gives value from 0~1 → 

—

assume: prior ~ Beta distribution. $P(\theta) = \theta^{\alpha-1}(1-\theta)^{\beta-1}$.

$$P(\theta|D) = \text{Constant} \cdot P(\theta) \cdot P(D|\theta) = \theta^{\alpha-1}(1-\theta)^{\beta-1} \cdot \theta^{n_H} \cdot (1-\theta)^{n_T}$$
$$= \theta^{n_H+\alpha-1} \cdot (1-\theta)^{n_T+\beta-1}$$

Cornell Machine Learning Notes No. 10.

Maximum $P(\theta|D)$ $\Rightarrow$ $P(H) = \theta_{\text{estimate}} = \dfrac{n_H + \alpha - 1}{n_H + n_T + \alpha + \beta - 2}$

$\Rightarrow$ different approaches (smoothing and MAP) get the same result

From a bayesian point of view, by considering θ as a random variable, we can do sth amazing:

almost always unsolvable $\left(\int_\theta P(y|\theta)\,P(\theta|D)\,d\theta\right)$.

$$P(y=\text{HEAD}|D) = \int_\theta P(\text{heads}, \theta|D)\,d\theta = \int_\theta P(\text{heads}, |\theta, D)\cdot P(\theta|D)\,d\theta.$$

This is to say, by seeing θ as a random variable, we can take into account all the possible models and make the final prediction.

To estimate $P(x,y)$: MLE $\theta = \arg\max_\theta P_\theta(D)$

MAP $\theta = \arg\max_\theta P(\theta|D)$

A more complex case of MLE.

$P(x)$. $\forall D \sim P(y,x)$ x -> a single feature

$$P(y|x) = \frac{\sum_{i=1}^{n} 1(x_i = x)\cdot 1(y_i = y)}{\sum_{i=1}^{n} 1(x_i = x)} \left(= \frac{P(Y=y, X=x)}{P(X=x)} \right)$$

If $\vec{X}$ is a d-dimensional vector?

It is unrealistic to see exactly the same $\vec{x}$

Cornell Machine Learning Notes. No. 11

↳ to rescue -> Naive bayes : $P(y=y|X=x) = \frac{P(X=x|Y=y)\cdot P(Y=y)}{P(X=x)}$

$P(X=x)$ -> Normalizer. $P(Y=y)$ -> ratio of y in all ys.

$P(X=x|Y=y)$ assume the features are independent given the label. → this is where the word "Naive" $= \prod_{\alpha=1}^{d} P(X_\alpha = x_\alpha | Y=y)$

It turns out in use, this assumption doesn't come from break things too much

---

$$h(\vec{x}) = \arg\max_y P(y|\vec{x}) = \arg\max_y P(x|y)P(y)$$

$$= \arg\max_y \left[\prod_{\alpha=1}^{d} P(x_\alpha|y)\right] P(y)$$

$$= \arg\max_y \left[\log P(y) + \sum_{\alpha=1}^{d} \log(P(x_\alpha|y))\right]$$

in probabilistics, we never multiply many many probabilities, because that always gives zero., we usually take log and compute the log.

A few cases of Naive Bayes.

- $x_\alpha \in \{v_1, v_2, \dots v_k\}$ ↙ k categories.

$$P(x_\alpha = j | y=c) = [\theta_{jc}]_\alpha \quad \sum_{j=1}^{k} [\theta_{jc}]_\alpha = 1$$

$$[\theta_{jc}]_\alpha = \frac{\sum_{i=1}^{n} I(y=c)\, I(x_\alpha = a_j) + l}{\sum_{i=1}^{n} I(y=c) + lk_\alpha}$$

- $x_\alpha \in \{0, 1, 2, \dots m\}$.

$$m = \sum_{\alpha=1}^{d} x_\alpha$$

$$P(\vec{x}|m, y=c) = \frac{m!}{x_1!\, x_2! \dots x_d!} \prod_{\alpha=1}^{d} \theta_{\alpha c}^{x_\alpha}$$

Cornell Machine Learning Notes. No. 12

Logistic Regression.

$$P(y|\vec{x}) = \frac{1}{1+e^{-y(w^Tx+b)}}, y \in \{+1, -1\}. \qquad P(y|x) = \frac{1}{1+e^{-ywx}}$$

Naive Bayes -> estimate $P(x|y)$, $P(y)$, then $P(y|x)$

Logistic Regression -> estimate $P(y|x)$

MLE: $\underset{w,b}{\operatorname{argmax}} \prod_{i=1}^{n} P(y_i|\vec{x}_i; w,b) = \underset{w,b}{\operatorname{argmax}} \sum_{i=1}^{n} \log P_{w,b}(y_i|\vec{x}_i; w, b)$

$= \underset{w}{\operatorname{argmax}} \sum_{i=1}^{n} \log P_w(y_i|\vec{x}_i, w) = \underset{w}{\operatorname{argmax}} \sum \log \frac{1}{1+e^{-ywx}}$

$= \underset{w}{\operatorname{argmax}} -\sum_{i=1}^{n} (1+e^{-ywx}) = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} (1+e^{-ywx})$

MAP: $\underset{w}{\operatorname{argmax}} P(D|w) \cdot P(w)$ $\quad P(w) \sim N(\vec{0}, \sigma^2)$

$= \underset{w}{\operatorname{argmax}} \left(\prod_{i=1}^{n} \frac{1}{1+e^{-ywx}}\right) \cdot e^{-\frac{w^Tw}{2\lambda}}$

$= \underset{w}{\operatorname{argmax}} \left[\sum_{i=1}^{n} \log(1+e^{-ywx}) - \frac{w^Tw}{2\lambda}\right]$

$= \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} \log(1+e^{-ywx}) + 2w^Tw.$

Cornell Machine Learning notes No. 13

min $L(w)$

$L(\vec{w}+\vec{\delta}) \simeq L(\vec{w}) + g(\vec{w})\vec{\delta} + \frac{1}{2}\vec{\delta}^T H(\vec{w})\vec{\delta} + \dots$

↓ only holds when $\vec{\delta}$ is really small step.

Gradient descent. $\vec{\delta} = -\alpha \vec{g}(w)$

$L(\vec{w}+\vec{\delta}) \simeq L(\vec{w}) - \alpha \vec{g}(w)^T \vec{g}(w)$ → smaller

Newton's Method. → works better when already near the minimum.

$L(\vec{w}+\vec{\delta}) \simeq L(\vec{w}) + g(\vec{w})\vec{\delta} + \frac{1}{2}\vec{\delta}^T H(\vec{w})\vec{\delta} + \dots$

$\frac{\partial L(\vec{w}+\vec{\delta})}{\partial \vec{\delta}} = g(\vec{w}) + H(\vec{w})\vec{\delta} = 0 \Rightarrow \delta = -H^{-1}(\vec{w}) g(\vec{w})$

$L(w) = \sum_{i=1}^{n} \log(1 + e^{-w^T x_i y_i})$

Linear Regression. : $y = w^T x + \varepsilon \quad \varepsilon \sim N(0, \sigma^2)$

$y_i \sim N(w^T x_i, \sigma^2)$

$$p(y_i | \vec{x}_i; w) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(w^T x_i - y_i)^2}{2\sigma^2}}$$

MLE: $\underset{w}{\operatorname{argmax}} \prod_{i=1}^{n} p(y_i | \vec{x}_i; w) = \underset{w}{\operatorname{argmax}} \prod_{i=1}^{n} e^{-\frac{(w^T x_i - y_i)^2}{2\sigma^2}}$

$= \underset{w}{\operatorname{argmax}} \sum_{i=1}^{n} -(w^T x_i - y_i)^2 = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} (w^T x_i - y_i)^2$

Cornell Machine Learning Notes No. 14.

MAP: $P(w|D) = \text{Constant} \cdot P(D|w) \cdot P(w)$

$$\underset{w}{\text{argmax}} \left[\prod_{i=1}^{n} P(y_i|\vec{x}_i, w)\right] \cdot P(w) = \underset{w}{\text{argmax}} \left(\prod_{i=1}^{n} e^{-\frac{(w^T x_i - y_i)^2}{2\sigma^2}}\right) \cdot e^{\frac{w^T w}{2\tau^2}}$$

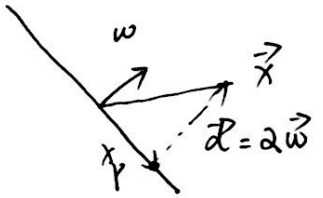
$$= \underset{w}{\text{argmax}} \left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(w^T x_i - y_i)^2 - \frac{1}{2\tau^2} w^T w\right)$$

$$= \underset{w}{\text{arg}\min} \sum_{i=1}^{n}(w^T x_i - y_i)^2 + \lambda w^T w$$

Support Vector Machines - Maximum Margin

Margin: the closest distance between the points and hyperplane.

$$\mathcal{H} = \{w^T x + b = 0\}.$$



$w^T x_p + b = 0 = w^T(\vec{x} - \vec{d}) + b$

$w^T(\vec{x} - 2\vec{w}) + b = 0$

$\Rightarrow 2 = \frac{w^T x + b}{w^T w}$

$\Rightarrow d = \sqrt{d^T d} = \frac{|w^T x + b|}{\sqrt{w^T w}}$

$$\gamma = \min_{(x,y)\in D} \frac{|w^T x + b|}{\sqrt{w^T w}}.$$

Margin (w, b) $\quad \max_{w,b} \gamma(w, b)$

$$\max_{w,b} \gamma(w, b) \quad \text{s.t.} \quad \forall i, \; y_i(w^T x_i + b) \geq 0$$

$$\max_{w,b} \left[\min_{(x,y)\in D} \frac{|w^T x + b|}{\sqrt{w^T w}}\right] \quad \text{s.t.} \quad \forall i \; y_i(w^T x_i + b) \geq 0$$

$$\max_{w,b} \frac{1}{\sqrt{w^T w}} \left[\min_{D} |w^T x + b|\right] \quad \text{s.t.} \quad \forall i. \; y_i(w^T x_i + b) \geq 0.$$

Cornel Machine Learning. Notes. No. 15

Fix $\min_{D} |w^T x + b| = 1$. (can be done by rescaling of w and b)

↳

$\max_{w,b} \frac{1}{\sqrt{w^T w}}$. s.t. .... → $\min_{w,b} w^T w$. s.t. $y_i (w^T x_i + b) \geq 0$

$\min_{D} |w^T x + b| = 1$

~~$\Leftrightarrow \forall i,\ y_i (w^T x_i + b) \geq 0$~~

$\min w^T w$ s.t. $\begin{cases} y_i (w^T x_i + b) \geq 0 \\ \min_{D} |w^T x_0 + b| = 1 \end{cases}$

$\Leftrightarrow$ s.t. $\forall i,\ y_i (w^T x_i + b) \geq 1$ ∵ $\min w^T w$.

$\Leftrightarrow \min_{w,b} w^T w$. s.t. $\forall i,\ y_i (w^T x_i + b) \geq 1$

what if the dataset is not linearly separable?

$\min_{w,b} w^T w$. s.t. $\forall i,\ y_i (w^T x_i + b) \geq 1 - \xi_i$. $\forall i,\ \xi_i \geq 0$

$+ C \sum_i \xi_i$

↳ $\xi_i = \begin{cases} 0 & \text{if } y_i (w^T x_i + b) \geq 1 \\ 1 - y_i (w^T x_i + b) & \text{if } y_i (w^T x_i + b) < 1 \end{cases}$ → hinge loss.

$\min_{w,b} w^T w + C \sum_i \max (1 - y_i (w^T x_i + b), 0)$

↙ regularizaer. ↙ Loss.

— Objective $= \frac{1}{n} \sum_{i=1}^{n} \text{Loss}(w)_i + \lambda \text{Regularizer}(w)$ (apply to many models)

— Loss functions:

Hinge Loss. $\max [1 - h(x_i) y_i, 0]^p$ $p=1$ → svm; $p=2$ → squared Loss svm

Log Loss $\log (1 + e^{-y_i h(x_i)})$ logistic regression.

Cornell Machine Learning Notes. No. 16.

Exp Loss $e^{-y_i h(x_i)}$ Adaboost.

0/1 Loss $\delta(h(x_i) \neq y_i)$

Regression. $y \in \mathbb{R}$

Squared Loss - $(h(x_i) - y_i)^2$ - estimates mean.

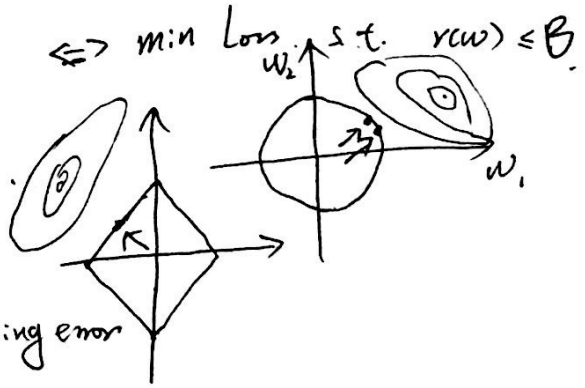
absolute Loss - $|h(x_i) - y_i|$ - estimate median.

Huber Loss - $\begin{cases} \frac{1}{2}(h(x_i) - y_i)^2 & \text{if } |h(x_i) - y_i| < \delta \\ \delta(|h(x_i) - y_i| - \frac{\delta}{2}) & \text{otherwise} \end{cases}$

log-cosh. Loss - ~~$\log(\frac{e^x + e^{-x}}{2})$~~ $\log(\frac{e^{h(x_i) - y_i} + e^{-h(x_i) + y_i}}{2})$.

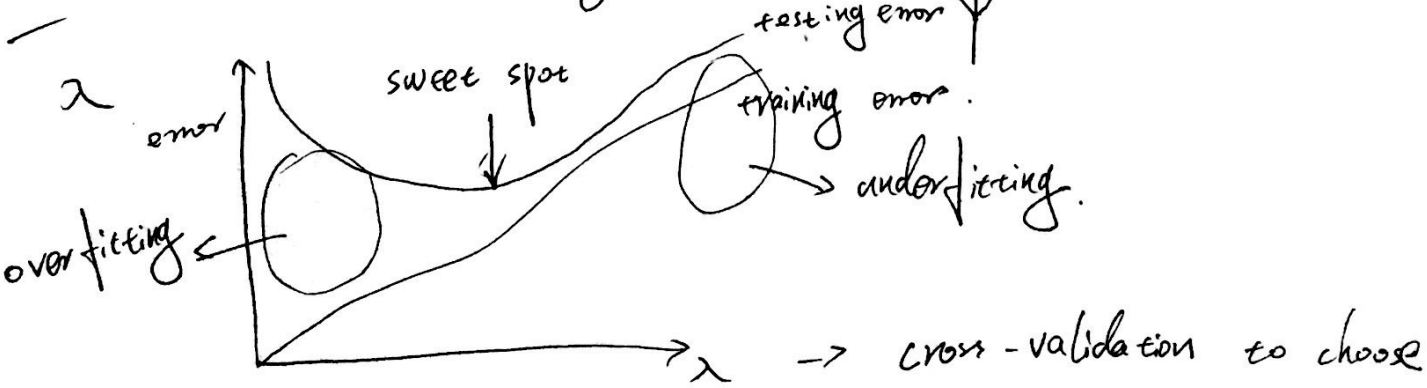
Regularization.

$w^T w$ $\min \text{loss} + \lambda r(w) \Leftrightarrow \min \text{Loss}$ s.t. $r(w) \leq B$.



$r(w) = w^T w$ L_2-regularization.

$r(w) = \|w\|$ L_1-regularization



-> cross-validation to choose

Cornell Machine Learning Notes No. 7

Neural Networks.

~~$h(x) = w^T * \phi(x)$~~ $\phi(x) = \sigma(Ax + b)$ ← to be learned.

$h(x) = \phi^T(x) w + b$ $x \to \phi(x)$ → some non-linear function.

mapping x into some other space and learn that mapping.

$h(x) = w^T \phi(x) + b$ $\phi(x) = \sigma(Ux + c)$

$h(x) = w^T \phi(x) + b$ $\phi(x) = \sigma(U\phi(x) + c)$ $\phi(x) = \sigma(\dots)$

Cornell Machine Learning. Note. No. 18.

Boosting.

$$H(x) = \sum_{t=1}^{I} \alpha_t h_t(x)$$

$$y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \qquad (x_1, x_2, x_3) \to \begin{pmatrix} H(x_1) \\ H(x_2) \\ H(x_3) \end{pmatrix} \qquad \begin{pmatrix} h(x_1) \\ h(x_2) \\ h(x_3) \end{pmatrix}$$

Gradient Descent in the Function Space.

$$H(x) = \sum_{t=1}^{I} \alpha_t h_t(x)$$

$$h = \underset{h \in \mathcal{H}}{\operatorname{argmin}}\ l(H + \alpha h) \approx \underset{h \in \mathcal{H}}{\operatorname{argmin}} \left[l(H) + \left\langle \frac{\partial l}{\partial H}, \alpha h \right\rangle \right]$$

$$= \underset{h \in \mathcal{H}}{\operatorname{argmin}} \left\langle \frac{\partial l}{\partial H}, \alpha h \right\rangle = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \sum_{i=1}^{n} \frac{\partial l}{\partial H(x_i)} \cdot \alpha h(x_i)$$

if $l(H) = \frac{1}{n} \sum_{i=1}^{n} (H(x_i) - y_i)^2$ $\qquad \frac{\partial l}{\partial H(x)} = \begin{pmatrix} H(x_1) - y_1 \\ H(x_2) - y_2 \\ \vdots \\ H(x_n) - y_n \end{pmatrix}$

$$\frac{\partial l}{\partial H(x_i)} = \frac{1}{n} \cdot 2 (H(x_i) - y_i)$$

h in the direction of $y - H(x)$. $\quad = H(x) - y$

Cornell Machine Learning. Note No. 19.

Gradient boosting trees.

$g \in \mathbb{R}$, $h \in$ CART of limited depth, 2 constants.

$$\arg\min \sum_i r_i h(x_i) \qquad r_i := \frac{\partial L}{\partial \hat{y}(x_i)} \qquad t_i = -r_i$$

$$\arg\min_h -\sum_i t_i h(x_i)$$

$\sum t_i^2$ has nothing to do with h.

$$\sum_i h^2(x_i) = C$$

$$\arg\min_h \sum_i h^2(x_i) - 2\sum_i t_i h(x_i) + \sum_i t_i^2$$

$$\arg\min_h \sum_i [h(x_i) - t_i]^2 \rightarrow \text{still squared loss.}$$

$$\hat{y}(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$$