Linear Regression : close to local minimum

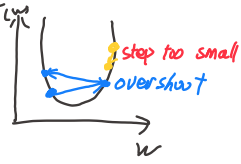
m : number of data.

- cost function : $J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_{w,b}(x^{(i)}) - y^{(i)}\right)^2$
- model : $f(w,b) = wx+b$
- gradient descent :

$$w = w - \alpha\frac{\partial}{\partial w}J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\left(f_{wb}(x^{(i)}) - y^{(i)}\right)x^{(i)}$$

$$b = w - \alpha\frac{\partial}{\partial b}J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\left(f_{w,b}(x^{(i)}) - y^{(i)}\right)$$

α is the learning rate
- too large
- too small



With multiple feature : $f_{\vec{w},b}(\vec{x}) = \vec{w}\vec{x} + b = w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots + b$

Gradient descent (in multiple)

$$\vec{w} = \vec{w} - \alpha\frac{\partial}{\partial \vec{w}}J(\vec{w},b) \qquad \frac{\partial}{\partial w_1}J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\left(f_{w,b}(\vec{x}^{(i)}) - y^{(i)}\right)x_1^{(i)}$$

$$b = b - \alpha\frac{\partial}{\partial b}J(w,b)$$

repeat { update each feature.

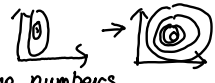
$$w_1 = w_1 - \alpha\frac{\partial}{\partial w_1}J(w,b)$$

$$\vdots$$

$$w_n = w_n - \alpha\frac{\partial}{\partial w_n}J(w,b)$$

}

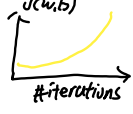
Feature scaling : scale the data have some transform to the numbers



平均归一化 mean normalization $\Rightarrow x_1 = \frac{x_1 - \mu_1}{\max - \min}$

Z-score normalization (σ 标准差) $\Rightarrow x_1 = \frac{x_1 - \mu_1}{\sigma_1}$

Choose learning rate



Feature engineering. transform or combine parameter $x_3 = x_1x_2$

$$f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + \ldots + b \rightarrow f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_2 + w_3x_3 + b$$

Polynomial regression use other function

$f_{\vec{w},b}(x) = w_1x + w_2x^2 + w_3x^3 + b$

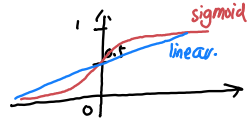
have to use feature scaling

change the function and do the gradient descent.

Classification: Logisic Regression.

(分类)

sigmoid function (logistic)



$g(z) = \frac{1}{1+e^{-z}}$ $0 < g(z) < 1$

$\lim_{x\to+\infty} g(x) = 1$

$\lim_{x\to-\infty} g(x) = 0$

$z = \vec{w}\cdot\vec{x} + b \rightarrow z \rightarrow g(z) = \text{logistic}(z)$ (scale to (0,1)) $\Rightarrow f_{\vec{w},b}(\vec{x}) = g(\vec{w}\cdot\vec{x}+b)$

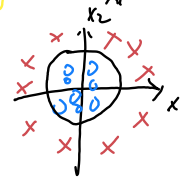
"logistic regression"

result: $f_{\vec{w},b}(\vec{x}) = 0.7$: 70% chance y is 1 (True)

Decision boundary: $f_{\vec{w},b}(\vec{x}) = P(x;\vec{w},b)$ { when $f(\vec{x})_{\vec{w},b} > 0.5$, $\hat{y} = 1$

(When $z = 0$)

Non-linear boundaries



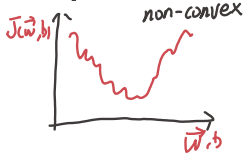
$f_{\vec{w},b}(\vec{x}) = g(z) = g(w_1x_1^2 + w_2x_2^2 + b)$

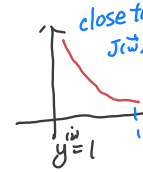
if $z = x_1^2 + x_2^2 - 1 = 0$

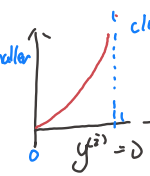
$x_1^2 + x_2^2 = 1$ (boundary)

Cost function: $J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(f_{\vec{w},b}(x^{(i)}) - y^{(i)})^2$ Loss of Linear regression $\Rightarrow L(f_{\vec{w},b}(x^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w},b}(x^{(i)})) & (y^{(i)}=1) \\ -\log(1-f_{\vec{w},b}(x^{(i)})) & (y^{(i)}=0) \end{cases}$ Loss of logistic regression.

(for logistic regression)







Simplified Cost function

new Loss: $L(f_{\vec{w},b}(x^{(i)}), y^{(i)}) = -y^{(i)} \log(f_{\vec{w},b}(x^{(i)})) - (1-y^{(i)}) \log(1 - f_{\vec{w},b}(x^{(i)}))$

y close to 0, this close to 0

y close to 0, this close to 1

Cost function: $J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m} L(f_{\vec{w},b}(x^{(i)}), y^{(i)})$

Use gradient descent (training)

derivative of $J(\vec{w},b)$:

$$\frac{\partial}{\partial w_j} J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}\left[(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})\, x_j^{(i)}\right]$$

$$\frac{\partial}{\partial b} J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})$$

Process: repeat {

$$w_j = w_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(x^{(i)}) - y^{(i)})\, x_j^{(i)}\right]$$

$$b = b - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(x^{(i)}) - y^{(i)})\right]$$

}

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1+e^{-\vec{w}\vec{x}+b}}$$

Problem of Overfitting

use different model to fit the problem

high bias

generalization

when it overfitted

biasis are too small (high variance)

Solve problem:

(1) more data

(2) use fewer features (feature selection)

(3) Regularization (正則化)

setting parameter smaller rather than 0

keep the feature, decrease the impact

Apply Regularization.

make parameter smaller. ($w_j \approx 0$)
(the penalize one)

given that parameters are $w_1, w_2, \ldots w_j, b$

$$J(\vec{w},b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2 \quad (\lambda > 0)$$

λ means Regularization parameter

when λ is bigger. it has more impact of w_j. it will shrink it smaller

Sample 1: (Linear Regression)

model: $f_{\vec{w},b}(\vec{x}) = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$

cost: $J(\vec{w},b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$

Process:

$$w_j = w_j - \alpha\frac{\partial}{\partial w_j}J(\vec{w},b) \rightarrow \frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}w_j$$

$$b = b - \alpha\frac{\partial}{\partial b}J(\vec{w},b) \rightarrow \frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})$$

derivative.

no necessary for regulize b

Sample 2 (Logistic Regression)

model: $f_{\vec{w},b}(\vec{x}) = \frac{1}{1+e^{-(\vec{w}\vec{x}+b)}}$

cost: $J(\vec{w},b) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(f_{\vec{w},b}(\vec{x}^{(i)})) + (1-y^{(i)})\log(1-f_{\vec{w},b}(\vec{x}^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$

loss of this Regression

Process:

$$w_j = w_j - \alpha\frac{\partial}{\partial w_j}J(\vec{w},b) \rightarrow \frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}w_j$$

$$b = b - \alpha\frac{\partial}{\partial b}J(\vec{w},b) \rightarrow \frac{1}{m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})$$

derivative.

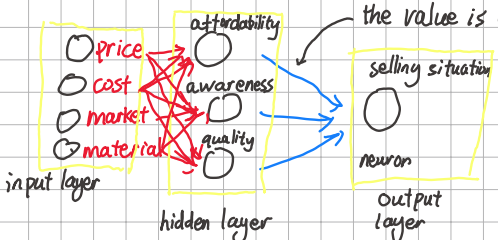
no necessary for regulize b

$f(\vec{w},b)$ is Logistic function

Neuron Network:

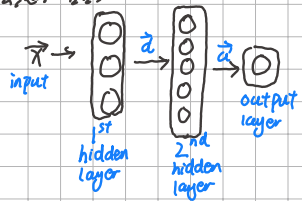
a neuron is given a input and give an output

Demand Prediction:

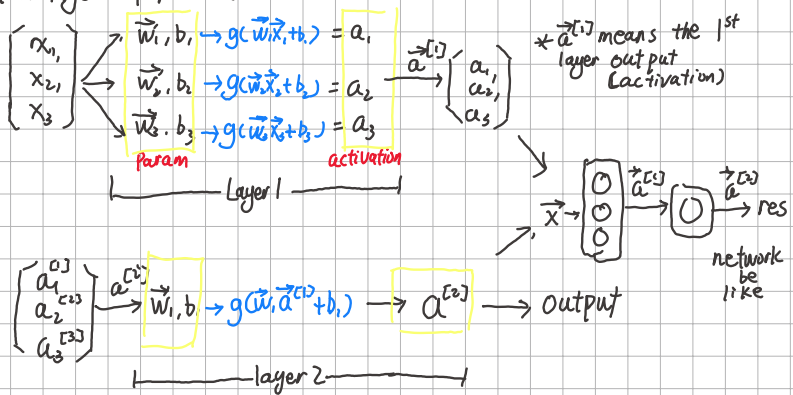


the value is called "activations" (激活值)

Multiple perceptron: multiple layers (多层感知器)



Build a layer of Network:



For the lth layer: $a_j^{[l]} = g(\vec{w}_j^{[l]} \cdot \vec{a}^{[l-1]} + b_j^{[l]})$

parameter of layer l and unit j

Matrix multiply

$\vec{a} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \rightarrow a^T \begin{bmatrix} 1 & 2 \end{bmatrix}$ 转置矩阵 Transpose

column

row

1) $\begin{bmatrix} 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix} = \begin{bmatrix} 1\times3+2\times4 & 1\times5+2\times6 \end{bmatrix}$

$\begin{bmatrix} \leftarrow \vec{a}^T \rightarrow \end{bmatrix} \begin{bmatrix} \uparrow & \uparrow \\ \vec{w}_1 & \vec{w}_2 \\ \downarrow & \downarrow \end{bmatrix} = \begin{bmatrix} \vec{a}^T\vec{w}_1 & \vec{a}^T\vec{w}_2 \end{bmatrix}$

2) $\begin{bmatrix} 1 & 2 \\ -1 & -2 \end{bmatrix} \cdot \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix}$

$= \begin{bmatrix} \leftarrow \vec{a}_1 \rightarrow \\ \leftarrow \vec{a}_2 \rightarrow \end{bmatrix} \cdot \begin{bmatrix} \uparrow & \uparrow \\ \vec{w}_1 & \vec{w}_2 \\ \downarrow & \downarrow \end{bmatrix}$

$= \begin{bmatrix} \vec{a}_1\vec{w}_1 & \vec{a}_1\vec{w}_2 \\ \vec{a}_2\vec{w}_1 & \vec{a}_2\vec{w}_2 \end{bmatrix}$

Training:

① define model $f_{\vec{w},b}(\vec{x})$

② loss & cost: $\underbrace{J(\vec{w},b)}_{\text{cost}} = \frac{1}{m}\sum_{i=1}^{m} \underbrace{L\left(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}\right)}_{\text{Loss function}}$

③ train data to minimize $J(\vec{w},b)$

Other Activation function: $a^{[-]} = g(\vec{w}\cdot\vec{x}+b)$

① Sigmoid

$g(z) = \frac{1}{1+e^{-z}} = P(y=1|\vec{x})$

good to solve binary problem

② ReLU

$g(z) = \max(0, z)$

can 0, 0 or +

③ Soft max (multi-classification) sample: y=1, 2, 3, 4

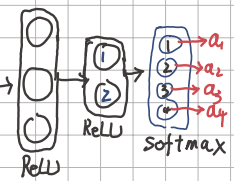
$z_1 = \vec{w}_1\vec{x}+b_1 \Rightarrow \frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}+e^{z_4}} = a_1 = P(y=1|\vec{x})$

$a_1+a_2+a_3+a_4 = 1$

$z_2 = \vec{w}_2\vec{x}+b_2 \Rightarrow \frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}+e^{z_4}} = a_2 = P(y=2|\vec{x})$

$z_3 = \vec{w}_3\vec{x}+b_3 \Rightarrow \frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}+e^{z_4}} = a_3 = P(y=3|\vec{x})$

$z_4 = \vec{w}_4\vec{x}+b_4 \Rightarrow \frac{e^{z_4}}{e^{z_1}+e^{z_2}+e^{z_3}+e^{z_4}} = a_4 = P(y=4|\vec{x})$



↓ to sum up

$$\begin{cases} z_j = \vec{w}_j \vec{x} + b_j, \ j = 1, \dots, N \\ a_j = \dfrac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}} = P(y=j \mid \vec{x}) \end{cases}$$

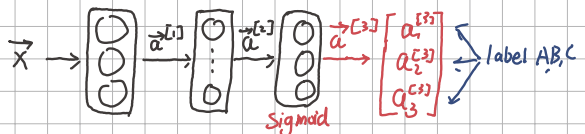
Cost of softmax Regression:

$$\text{Loss: } loss(a_1, \dots, a_N, y) = \begin{cases} -\log a_1 & \text{if } y=1 \\ -\log a_2 & \text{if } y=2 \\ \vdots & \vdots \\ -\log a_N & \text{if } y=N \end{cases}$$

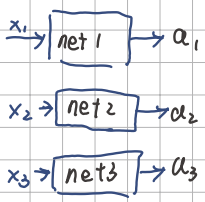
$$\text{Cost: } J(w) = \frac{1}{m} \sum_{n=1}^{m} loss(w)$$

Multiple Classification

1) multi-label



2) multi-Class

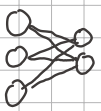


when using multiClass classification, you have to build more network.

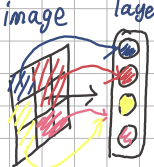
Other learning Algorithm:

Adam automatically adjust learning rate α

Other Layer Type

Dense:  each get all neuron's activation

Convolutional Layer: (卷积层)



each activation focus of different part

benefit: ① less overshooting
② need less data
③ Faster computation.

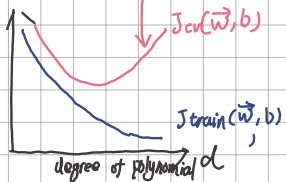
Evaluate model

1) Dataset
- training set 70% → $J_{train}(\vec{w}, b)$
- test set 30% → $J_{test}(\vec{w}, b)$

2) Dataset
- Training set 60%
- cross-validation set 20% → test different model
- test set 20%

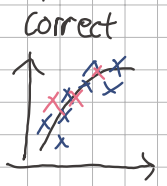
Model selection

1. $f_{\vec{w},b}(\vec{x}) = w_1 x_1 + b$ d=1
2. $f_{\vec{w},b}(\vec{x}) = w_1 x_1 + w_2 x^2 + b$ d=2
⋮
10. $f_{\vec{w},b}(\vec{x}) = w_1 x + w_2 x^2 + \dots + w_{10} x^{10} + b$ d=3



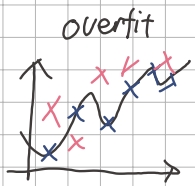
cross-validation: $J_{cv}(w^{<4>}, b^{<4>})$ use to choose which model

test set use to evaluate the final model



$J_{train}(\vec{w},b)$ is low

$J_{cv}(\vec{w},b)$ is low

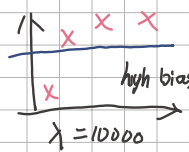


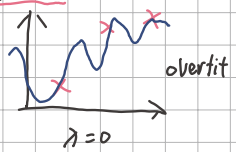
$J_{train}(\vec{w},b)$ is low

$J_{cv}(\vec{w},b)$ is high

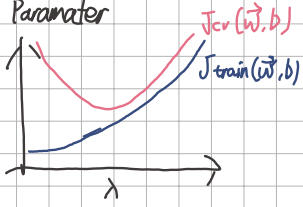
Choose Regularzation · model: $f_{\vec{w},b}(x) = w_1 x + w_2 x^2 + w_3 x^3 + \dots + b$

$$J(\vec{w},b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2 + \boxed{\frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2}$$ Regularization





For Regularization Parameter

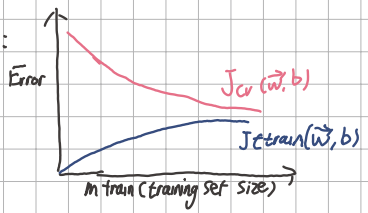


Judge the performance

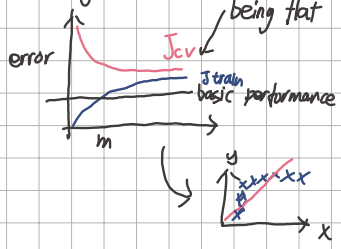
1) Baseline Performance (基准值)
2) Training Error — if ① < ② ⇒ high bias
3) Cross Validation Error — if ① < ③ ⇒ overfit

Learning Curve

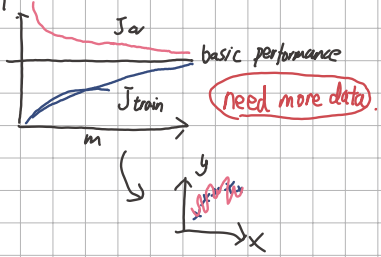
For a model:



For high bias model



For overfit model



Debug an model

1) get more data → high variance
2) smaller features → high variance
3) additional features → high bias

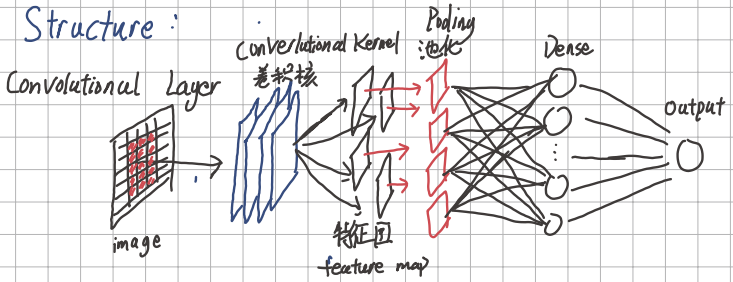
4) add polynomial features (x_1^2, x_2^2, x_1x_2, etc.)
↳ high bias

5) decrease λ → high bias

6) increase λ → high variance

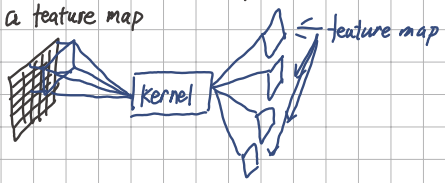
Convolutional Neuron Network
卷积神经网络.

Structure:

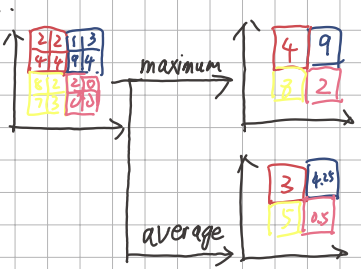


Convolutional kernel:

match each area of the input, compute each part into a feature map



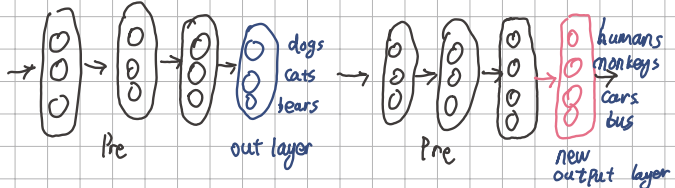
Pooling:



make features easier to caculate

Transfer learning (迁移学习)

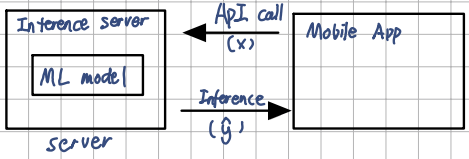
use pre-trained model to train another model
(just change the output layer)



Change layer, and train the new model (further train)

- train new part
- train whole model

Deploy a model



Precision 精度(P) $\frac{\text{True Pos}}{\text{True Pos + False Pos}}$ (True among pos)

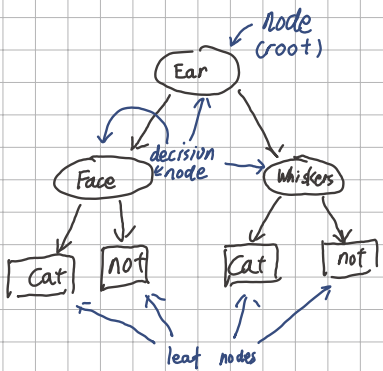
Recall: 召回(R) $\frac{\text{True Positive}}{\text{True Positive + False Negative}}$ (True among True Positive)

F1 Score: $\frac{1}{\frac{1}{2}(\frac{1}{P}+\frac{1}{R})} = 2\frac{RP}{P+R}$

Decision tree (决策树)

1 How to split?

2. How to stop split?
   - node is 100% pure
   - exceeding maximum depth
   - purity score.

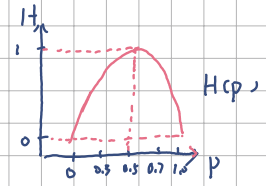


Entropy (H): measure of impurity

p_1: a class of the set

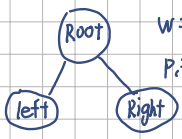
$p_0 = 1 - p_1$

$H(p_1) = -p_1 \log_2(p_1) - p_0 \log_2(p_0)$



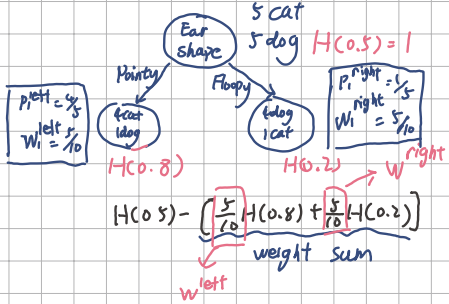
Information gain 信息增益

define: $H(p_1^{root}) - \left(w^{left} H(p_1^{left}) + w^{right} H(p_1^{right})\right)$

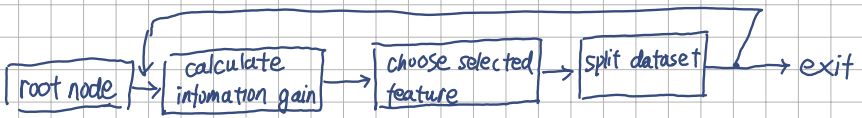


w: weight

p_1: ratio of target



Whole Process:



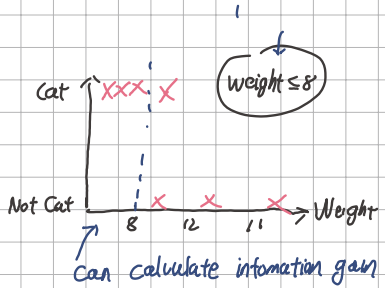
One-hot encoding (Process multiple features)

in k features, 0 means not, 1 means has the feature

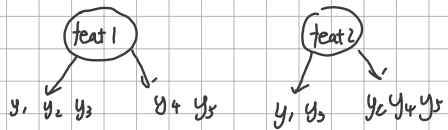
	Pointy ear	Floppy ear	Oval ear
1	1	0	0
2	0	1	0
3	1	0	0
4	0	0	1

Continuous values features

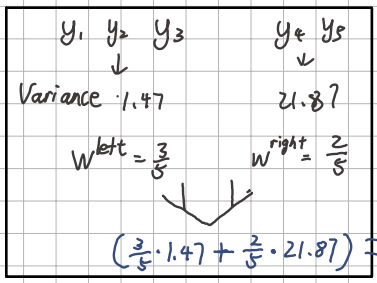
split at a treshold which can get highest information gain



Regression tree



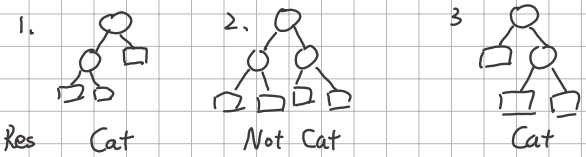
count Variance as purity and get infomation gain (方差)



when choosing the feature, choose the highest weight sum Variance (加权方差) reduction

Tree ensemble.

when you have multiple trees, use both tree to judge and choose the best result



let the trees vote the result, and choose the one who has highest vote.

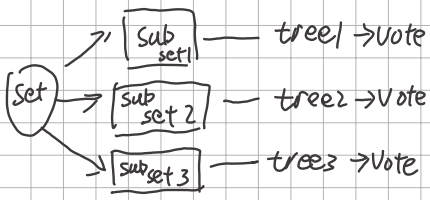
How to come up with those trees?

Sampling with replacement (带放回抽样)

each time get one and put it back, finally will get different random training set. to use that way you can build a new training set.

1	red	red	blue
2	green	blue	yellow
3	yellow	green	red

Random forest:



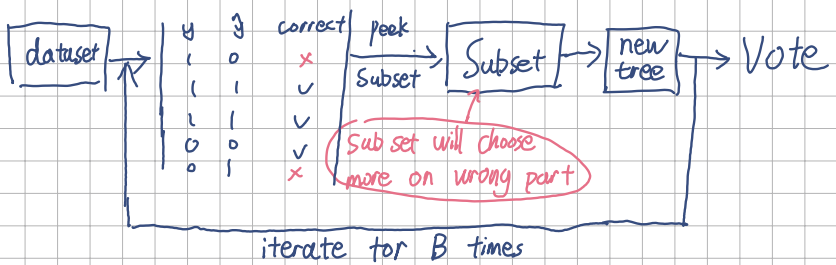
loop
- given a dataset of size m.
- For b=1 to BC100, 64, 128.....)
  - use sampling with replacement. get a new traningset at size of m and get a new tree
  - when predicting, each tree will give a vote

Randomizing the feature choise

from n features, only choose a subset of k<n features to allow algorithm to choose. usually: $k=\sqrt{n}$

XG Boost:

For the loop in Random forest



XG Boost (extreme Gradient Boosting) provide a way to choose subset and fit the dataset

When to use decition trees:

tree:
- structed data (data table)
- fast training

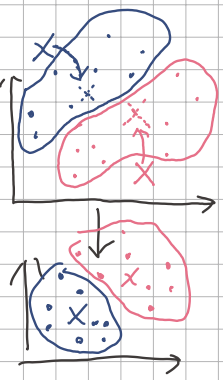
Neuron Networks
- all types data (images, audio)
- mixed data.
- slower
- work with transfer learning

Unsupervised Learning

Clustering (聚类)

k means:

1) choose random cluster centroid (随机心)
2) Assign points depends on the distance to the cluster centroid
3) move the cluster centroid to the average location of the point of itself
4) loop



in program:

Choose initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K$

Repeat {

for i=1 to m

$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$

in math: $\min_k \|x^{(i)} - \mu_k\|^2$ (L2 norm (L2 范数))

for k=1 to K

μ_k := average (mean) of points assigned to cluster k

$\mu_1 = \frac{1}{4}[x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)}]$

}

Cost function of k-means:

$$J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K) = \frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

Distortion (失真)

Randomly choos 100 different initialization and choose the lowest J value

Anomaly Detection (异常检测): find unnormal parts

Gaussian Distribution:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

Training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

x has n features

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Model : $p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$

Process :

1. Choose n features x_i as anomalous examples
2. Fit parameters : $\mu_1 \dots \mu_n, \sigma_1^2, \dots \sigma_n^2$

$$\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)} \qquad \sigma_j = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$$

3. give a new example x, compute $p(x)$:

$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

4: Anomoly if $p(x) < \varepsilon$

Algorithm evaluate

If . 10000 good sample
20 anomalous sample.

Training set: 6000 good samples. (y=0)

Cross Validation: 4000 good samples (y=0)
20 anomalous (y=1)

$$y = \begin{cases} 1 & \text{if } p(x) < \varepsilon \\ 0 & \text{if } p(x) \geq \varepsilon \end{cases}$$

ε threshold

evaluation metrics.

- True pos, false pos, false neg, true ney
- Preception/Recall
- F_1 - score

Recommend System (Predict rating in unlabled)

features.

For users:	Alice(1)	Bob(2)	Carol(3)	x_1 (romance)	x_2 (action)
Movie 1	5	3	4	0.9	0
Movie 2	0	?	?	1.0	0.01
Movie 3	0	0	?	0.1	1.0

Notation:

$r(i,j)$ if user j has rated movie i (0 or 1)

$y^{(i,j)}$ = rating given by user j on movie i

i : movie

j = user

$w^{(j)}, b^{(j)}$ parameters for user j

$x^{(i)} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ feature vector for movie i

Predict rating: $w^{(j)} \cdot x^{(i)} + b^{(j)}$

$m^{(j)}$ = number of movies rated by user j

Cost:

$$\min_{w^{(j)}, b^{(j)}} J(w^{(j)}, b^{(j)}) = \frac{1}{2m^{(j)}} \sum_{i:r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2m^{(j)}} \sum_{k=1}^{n} (w_k^{(j)})^2$$

Total Cost: (fitting the user parameter)

$$(1)\ J(w^{(1)}, \dots, w^{(n_u)}, b^{(1)}, \dots, b^{(n_u)}) = \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (w_k^{(j)})^2$$

Feature Cost (get the proper feature $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$ of movies)

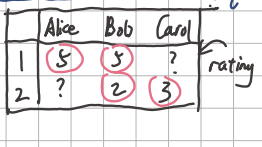
$$(2)\ J(x^{(1)}, x^{(2)}, \dots, x^{(n_m)}) = \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2$$

Collaborative filtering (协同过滤)

Put together:

$$\min_{\substack{w^{(1)}, \dots, w^{(n_u)} \\ b^{(1)}, \dots, b^{(n_u)} \\ x^{(1)}, \dots, x^{(n_m)}}} J(w, b, x) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} \left(w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)}\right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (w_k^{(j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2$$

this formula made w, b, x both as parameters while learning from chart, It's available to count cost from the movie that has rated



Train the model

Linear regression

repeat {

$$w_i^{(j)} = w_i^{(j)} - \alpha \frac{\partial}{\partial w_i^{(j)}} J(w,b,x)$$

$$b^{(j)} = b^{(j)} - \alpha \frac{\partial}{\partial b^{(j)}} J(w,b,x)$$

$$x_k^{(i)} = x_k^{(i)} - \alpha \frac{\partial}{\partial b^{(j)}} J(w,b,x).$$

}

while caculating, w, b, x are real numbers, and $J(w,b,x)$ comes from the dataset (people's rate) to fit.

For binary label

like: did the user clicked? (0/1)

did it attract user? (0/1)

Model:

$$y^{(i,j)}: f_{(w,b,x)}(x) = g(w^{(j)} \cdot x^{(i)} + b^{(j)})$$

Loss:

$$L(f_{(w,b,x)}(x), y^{(i,j)}) = -y^{(i,j)} \log(f_{(w,b,x)}(x)) - (1-y^{(i,j)}) \log(1 - f_{(w,b,x)}(x))$$

二元交叉熵

Loss function:

$$J(w,b,x) = \sum_{(i,j):r(i,j)=1} L(f_{(w,b,x)}(x), y^{(i,j)})$$

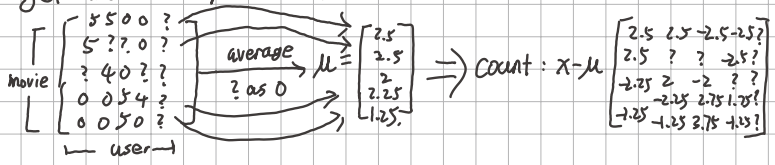
Add new user

	Alice	Bob	Eve
1	5	?	?
2	0	0	?
3	?	3	?

(打分区间 $1 \sim 5$)

use Mean normalization:

get the data from chart



For user j on movie i, we predict: $w^{(j)} \cdot x^{(i)} + b^{(j)} + \mu_i$

Find Related items

feature $\vec{x}^{(i)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

similarity $\| x^{(k)} - x^{(i)} \|^2$

Content-Based filtering

$r(i,j)$ user j rated item i

$y(i,j)$ rating given by user j on item i (if defined)

User features
- Age
- Gender
- Country
- .....

$x_u^{(j)}$ for user j $= \begin{bmatrix} 0.3 \\ 0.42 \\ \cdots \\ 2.7 \end{bmatrix} \Longrightarrow V_u^{(j)}$

Movie feature
- Year
- Genre
- Reviews
- ....

$x_m^{(i)}$ for movie i $= \begin{bmatrix} 0.2 \\ 0.51 \\ 7.3 \\ \cdots \\ 1.1 \end{bmatrix} \Longrightarrow V_m^{(i)}$

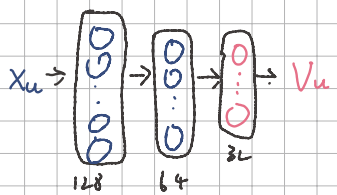
no need to be same dimention

same dimension

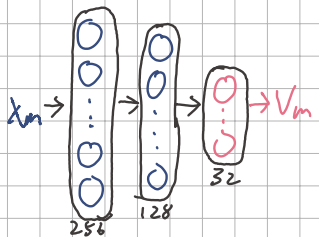
count $V_u^{(j)} \cdot V_m^{(i)}$ as predicted ratings

How to get $V_u^{(j)}$ $V_m^{(i)}$?

$X_u \rightarrow V_u$ User Network



$X_m \rightarrow V_m$ Movie network



Prediction : $\vec{V}_u \cdot \vec{V}_m$

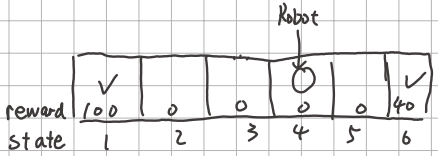
$g(V_u^{(j)} \cdot V_m^{(i)})$ to predict the probability of $y^{(i,j)}$ is 1

Cost function. $J = \sum_{(i,j):r(i,j)=1} (V_u^{(j)} \cdot V_m^{(i)} - y^{(i,j)})^2 + NN$ regularization term.

Reinforcement Learning
强化学习

Return

Assuming a robot reach a goal can have reward and each move have a Discount



the Reward : go left · $0 + (0.9) \cdot 0 + (0.9)^2 \cdot 0 + (0.9)^3 \cdot 100 = 72.9$

Discount factor

The discount Factor represents the discount of each move
The Formula:

$$\text{Return} = R_1 + \gamma R_2 + \gamma^2 R_3 + \dots \quad \text{(until terminal state)}$$

R: reward γ: discount factor

Policy (策略)

The goal of Reinforcement Learning is to find a policy that can make states reflect into actions.

state s —policy π→ action a

State action value function

Q(s, a)

- start from state s
- take action a (once)
- then behave optimally after that
  best behave

Q(s, a) means the reward you get when you do a action and do the best behave after you do that

1) The best possible return from state s is $\max_a Q(s, a)$
2) The best possible action in state s is action a that gives max Q(s, a)

Bellman equation

$R(s)$ = Reward of current state

s: current state
a: current action
s': state you get to after taking action a
a': action that you take in state s.

formula:

$$Q(s,a) = \underbrace{R(s)}_{\text{Reward you get right away}} + \underbrace{\gamma \max_{a'} Q(s',a')}_{\text{Return from behave optimally starting from } s'}$$

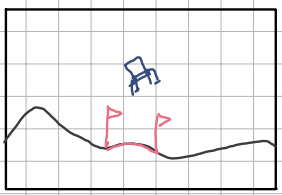
$Q(s,a)$

example: $R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \dots$

$$= R_1 + \gamma \underbrace{[R_2 + \gamma R_3 + \gamma^2 R_4 + \dots]}_{Q(s',a')}$$

It's a recursion function

MOON Lander



a moon lander, can Fire engine (push up) or fire side thruster (push angle), algorithm need to control the lander to the pad

State:

$$s = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \theta \\ l \\ r \end{bmatrix}$$

- x: x axis
- y: y axis
- $\dot{x}$: x speed (change rate)
- $\dot{y}$: y speed
- θ: angle
- l: left leg landed?
- r: right landed?

Reward function

- Getting to land pad 100 ~ 140
- Additional reward for moving toward to pad
- Crashed −100
- Soft landing +100
- Leg grounded +10
- Fire main engine: −0.3
- Fire side thruster: −0.03

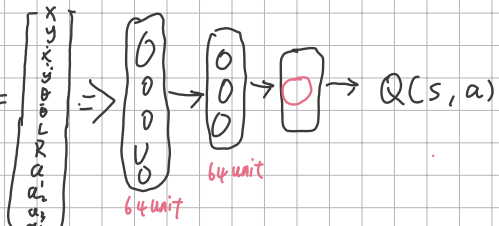
Task: learn a policy π that given s picks action $a = \pi(s)$ so as to maximize the return ($\gamma = 0.985$)

Pick state function

In state S, use NN to compute
Q(s, main) Q(s, left), Q(s, right), Q(s, nothing)
And Ultimately maximize Q(s,a) (one-hot)

Network:

$\vec{x} = \begin{bmatrix} s \\ a \end{bmatrix}$ 12, input



Learn of model

Bellman Equation: $Q(s,a) = R(s) + \gamma \max_{a'} Q(s', a')$

Dataset:

	x	y
$(s^{(1)}, a^{(1)}, R(s^{(1)}), s'^{(1)})$	$x^{(1)} = (s^{(1)}, a^{(1)})$	$y^{(1)}$
$(s^{(2)}, a^{(2)}, R(s^{(2)}), s'^{(2)})$		$y^{(2)}$
$(s^{(3)}, a^{(3)}, R(s^{(3)}), s'^{(3)})$		$y^{(3)}$

$y^{(1)} = R(s^{(1)}) + \gamma \max_{a'} Q(s'^{(1)}, a')$ comes from learning

Process:

Initialize neural Network randomly as guess of Q(s, a)
Repeat{

Get (s, a, R(s), s')
Store recent data tuples ← Replay Buffer

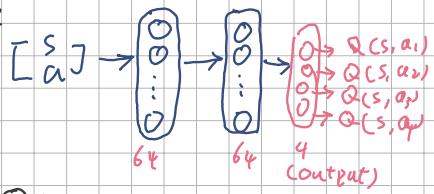
Train:
Create set of example of $x = (s, a)$ $y = R(s) + \gamma \max_{a'} Q(s', a')$ Comes from training
Train Q_{new} such that $Q_{new}(s, a) \approx y$

Set $Q = Q_{new}$

}

Improve Net work

prevent from calculate Q too much times, network can have multiple value



Choose $Q(s, a_i)$

option 1:

pick action a that maximize $Q(s,a)$

Option 2 (ε-greedy)

With 0.95 prob. pick the max $Q(s,a)$ "Exploitation"

With 0.05 prob. pick a randomly "Exploration"

With ε-greedy policy algorithm can discover different way of finding max$Q(s,a)$ better (more discovery)

start ε high 1.0 -> 0.01 Gradually decrease

End.