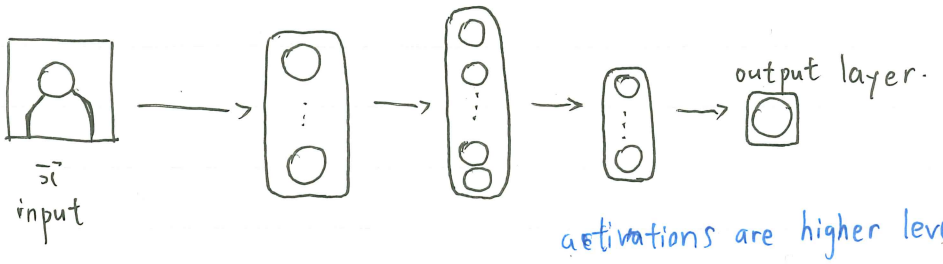


activations are higher level features

e.g. 1st hidden ~~layer~~ neuron might look for a low vetical line

2nd hidden ~~layer~~ neuron might look for an oriented line

---

earliest hidden layers may look for short lines

---

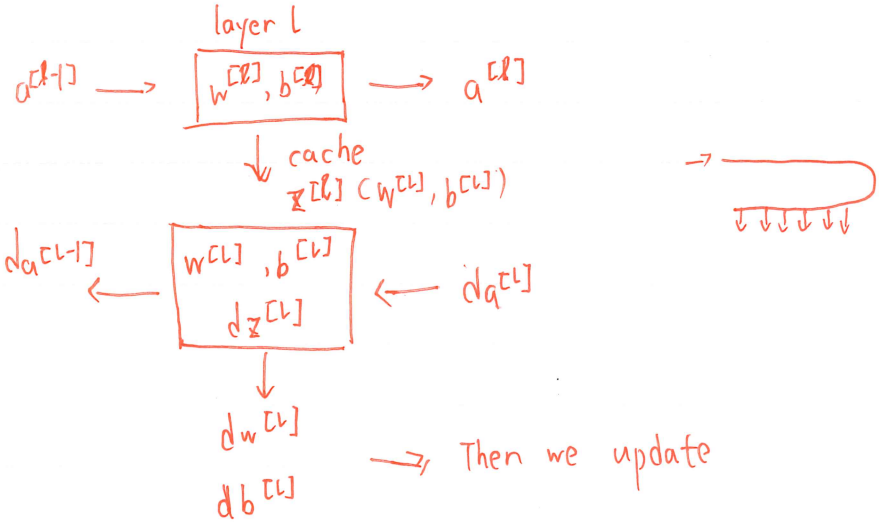
next layer...

(group of short lines) The first neuron may try to detect presence or absence of an eye

The second neuron — corner of a nose/ bottom of a ear

Finally — features of face shapes.

Nobody told it how to detect, all by itself!



No.

Date

Neural network layer.

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

$\vec{x}$ → layer 0

$a_1 = g(\vec{w_1} \cdot \vec{x} + b_1)$

$a_2 = g(\vec{w_2} \cdot \vec{x} + b_2)$

$a_3 = g(\vec{w_3} \cdot \vec{x} + b_3)$

$\vec{a}^{[1]}$ → layer 2. $\vec{w}_1^{[2]}\ b_1^{[2]}$ $a_1^{[2]} = g(\vec{w}_1^{[2]} \cdot \vec{a}^{[1]} + b_1^{[2]})$ → $a^{[2]}$ Final output.

(Then we may apply a threshold: is $a^{[2]} \geq 0.5$? Yes: $\hat{y} = 1$ No: $\hat{y} = 0$)

notation of layer numbering [1], [2]

$\vec{w_1} \rightarrow \vec{w}_1^{[1]} \ldots \vec{w}_1^{[k]}$

$b_1 \rightarrow b_1^{[1]} \ldots b_1^{[k]}$

$\vec{a_1} \rightarrow a_1^{[1]} \ldots a_1^{[k]}$

Activation value of layer l, unit (neuron) j

$a_j^{[l]} = g(\vec{w}_j^{[l]} \cdot \vec{a}^{[l-1]} + b_j^{[l]})$ ← parameters w & b of layer l, unit j

↑ sigmoid (activation function)

↑ output of layer $l-1$

Forward propagation

Handwritten digit recognition.

$\vec{x}$ → $\vec{a}^{[1]}$ → $\vec{a}^{[2]}$ → $\vec{a}^{[3]}$ → probability of being a handwritten "1"

25 units 15 units 1 unit

forward propagation

propagating the activations of neurons.

tanh / Leaky ReLU / swish activation

No.

Date

Choosing $g(z)$ based on the demand of y

Output Layer:

Binary classification → Sigmoid $y=0$ or 1

Regression → Linear activation $y=-/+$

Regression → ReLU $y=0$ or $+$ (can't below 0)

Hidden Layer:

Most common choice — ReLU:

- Faster to compute
- flat at only one place (when $z<0$) make gradient descent faster

- if $g(z)=z$ for all hidden layers, it's equivalent to linear regression because a linear function of a linear function is a linear function
- if g is always sigmoid, it's equivalent to logistic regression.

- The disable feature of the ReLU activation enables models to swith together linear segments to model complex non-linear functions.

No.

Date

Multiclass Classification

Softmax regression (N possible outputs)

$$z_j = \vec{w_j} \cdot \vec{x} + b_j \quad j = 1 \dots N$$

$$a_j = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}} = P(y = j | \vec{x})$$

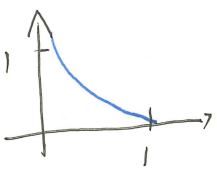
the exponential in the numerator of softmax magifies small differences in value

Note: $a_1 + a_2 + \dots + a_N = 1$

Logistic regression can be seen as a special case of softmax

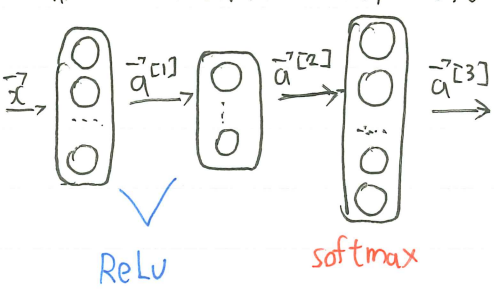
Cost:

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



loss = SparseCategoricalCrossentropy()

Neural Network with Softmax output:



25 units 10 units 10 units

10 classes

$$z_1^{[3]} = \vec{w_1}^{[3]} \cdot \vec{a}^{[2]} + b_1^{[3]}, \quad a_1^{[3]} = \frac{e^{z_1^{[3]}}}{e^{z_1^{[3]}} + \dots + e^{z_{10}^{[3]}}} = P(y=1|\vec{x})$$

$$z_{10}^{[3]} = \vec{w_{10}}^{[3]} \cdot \vec{a}^{[2]} + b_{10}^{[3]}, \quad a_{10}^{[3]} = \frac{e^{z_{10}^{[3]}}}{e^{z_1^{[3]}} + \dots + e^{z_{10}^{[3]}}} = P(y=10|\vec{x})$$

logistic regression: $a_j^{[i]}$ only relates to $z_j^{[i]}$

softmax:

$a_j^{[i]}$ relates to $a_1^{[i]} \dots a_N^{[i]}$

Campus

No.

Date

TensorFlow implementation.

x = np.array([[200, 17]]) → [200 17] 1×2 matrix

x = np.array([[200], [17]]) → $\begin{bmatrix} 200 \\ 17 \end{bmatrix}$ 2×1 matrix.

x = np.array([200, 17]) → 1D vector.

Activation vector:

x = np.array([[200.0, 17.0]]).

layer_1 = Dense(units = 3, activation = 'sigmoid').

a1 = layer_1(x)

tf.Tensor([[0.2 0.7 0.3]], shape = (1, 3), dtype = float32)

similar to matrix 1×3 matrix 32 bits

a1.numpy().

transfer from tensor to numpy

layer_2 = Dense(units = 25, ...

...

model = Sequential([layer_1, layer_2, ...])

model.compile(....)

x = np.array([[0..... 245, ... 17],

[. . . . .]])

y = np.array([1, 0])

model.fit(x, y)

model.predict(x_new)

No.

Date

Neural network implementation in python

```
X = np.array([[200, 17]])
W = np.array([[1, -3, 5], [-2, 4, -6]])
B = np.array([[-1, 1, 2]])

def dense(A_in, W, B):
    Z = np.matmul(A_in, W) + B.    / A_in @ W + B
    A_out = g(Z)                   matrix multiplication
    return A_out
def sequential(x):
    a1 = dense(x, w1, b1).
    ...
    a4 = dense(a3, w4, b4)
    f_x = a4
    return f_x
```

```
import tensorflow as tf.
from tensorflow.keras import Sequential.
model = Sequential(
    [
        tf.keras.Input(shape=(2,)),  from tensorflow.keras.layers import Dense
        Dense(3, activation='sigmoid', name='layer1'),
        ....
    ]
)
model.summary
W1,b1= model.get_layer("layer1").get_weights()  . . .
model.compile(
    loss= tf.keras.losses.BinaryCrossentropy()
    optimizer= tf.keras.optimizers.Adam(learning_rate=0.01),.
)
model.fit(
    Xt, yt,
    epochs=10)
)
w1,b1= . . . . . .
predictions = model.predict(norm_l(X_test))
yhat = (predictions >= 0.5).astype(int)
```

① specify the model

② compile the model
Using specific loss function.

③ train the model

No.

Date

Model Training Steps

① specify how to compute output given input x and parameters w, b

$f_{\vec{w},b}(\vec{x}) = ?$

logic regression

```
z = np.dot(w,x) + b
f_x = 1/(1+ np.exp(-z))
```

neural network

```
model = Sequential([
    Dense( .... )
])
```

② specify loss and cost

$L(f_{\vec{w},b}(\vec{x}), y)$

$loss = -y * np.log(f\_x) - (1-y) * np.log(1-f\_x)$

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

binary cross entropy

```
model.compile(
    loss = BinaryCrossentropy())
```

for numbers — MeanSquaredError()

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

```
model.fit(X,Y, epochs=100)
```

$w = w - alpha * dj\_dw$

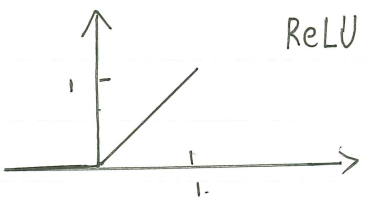
$b = b - alpha * dj\_db$

gradient descent

tensorflow compute pratial derivative.

by using "back propagation"

Activation Functions:

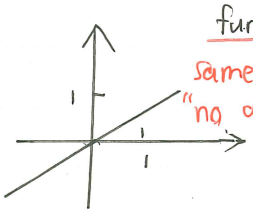


ReLU (Rectified Linear Unit)

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

linear activation function

same with "no activation function"



$g(z) = z$

No.

Date

* Numerical Roundoff Errors

$2.0/10000 \longrightarrow 0.000200\cdots$

$1+\frac{1}{10000}-(1-\frac{1}{10000}) \longrightarrow 0.000199\cdots$

Due to limited memory

Original loss for logistic regression

$loss = -y\log(a) - (1-y)\log(1-a)$

More accurate loss

$loss = -y\log(\frac{1}{1+e^{-z}}) - (1-y)\log(1-\frac{1}{1+e^{-z}})$ (less legible)

Dense(units=1, activation='sigmoid') 'linear'

model.compile(loss=BinaryCrossEntropy(from_logits=True))

f_x = tf.nn.sigmoid(model(X)) z

Then Tensorflow can rearrange terms to aviod very small / big numbers

For softmax:

More accurate:

$$L(\vec{a}, y) = \begin{cases} -\log\frac{e^{z_1}}{e^{z_1}+\cdots+e^{z_{10}}} & \text{if } y=1 \\ \quad\cdots\cdots & \\ -\log\frac{e^{z_{10}}}{e^{z_1}+\cdots+e^{z_{10}}} & \text{if } y=10 \end{cases}$$

model.compile(loss=SparseCategoricalCrossEntropy(from_logits=True))

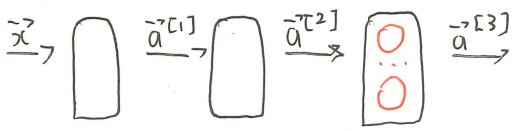
Be aware the result of X would be $z_1 \cdots z_{10}$ instead of $a_1 \cdots a_{10}$.

f_x = tf.nn.softmax(model(X))

No.

Date

Multi label classification different from multiclass



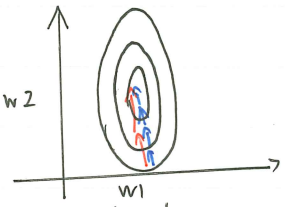
$$\vec{a}^{[3]} = \begin{bmatrix} a_1^{[3]} \\ a_2^{[3]} \\ a_3^{[3]} \end{bmatrix}$$ car / bus / pedestrian

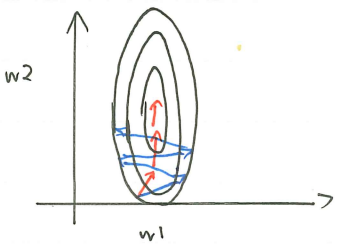
sigmoid

Depending on how Gradient descent is proceeding, the "Adam" algorithm in Tensorflow will adjust learning rate automatically

Adam = Adaptive Moment estimation.

$$w_j = w_j - a_j \frac{\partial}{\partial w_j} J(\vec{w}, b)$$





If w_j (or b) keeps moving in same direction. If w_j (or b) keeps oscillating

increase w_j reduce w_j

model.compile (optimizer = tf.keras.optimizers.Adam(learning_rate = 1e-3)

try to modify it!

Campus

No.

Date

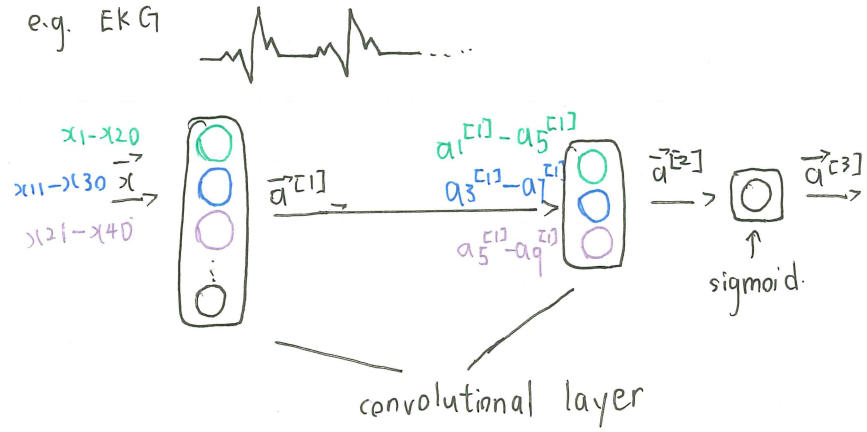
Convolutional Layer.

Each Neuron only looks at part of previous layer's inputs.

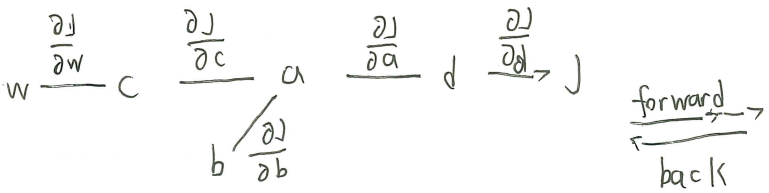
- Faster computation
- less prone to overfitting

Convolutional Neural Network 卷积神经网络

e.g. EKG



Back propagation.



compute. $\frac{\partial J}{\partial a}$ once to compute both $\frac{\partial J}{\partial w}$ and $\frac{\partial J}{\partial b}$.

If model has N nodes and P parameters, we only compute N+P step rather than NxP steps

No.

Date

Steps for backprop:

① : calculate the local derivative of the node $^{(5)}$

② use the chain rule.

Debug a learning algorithm.

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

encounter unacceptably large errors:

- Get more training examples. fix high variance
- Try smaller sets of features fix high variance.
- Try getting additional features fix high bias
- Try adding polynomial features. (x_1^2, x_2^2, x_1x_2 ....) same.
- Try decreasing λ / increasing λ

fix high bias high variance.

Evaluate your model

Divide data set into training set and test set

Fit parameters by minimizing cost function $J(\vec{w}, b)$

compute test error and training error

No.

Date

AI { ANI (narrow), AGI (general)

Model selection

d = 1 → $f_{\vec{w},b}(\vec{x}) = w_1x + b$ → $J_{test}(w^{<1>}, b^{<1>})$

...

d=10 → $f_{\vec{w},b}(\vec{x}) = w_1x + w_2x^2 + \dots + w_{10}x^{10} + b$ → $J_{test}(w^{<10>}, b^{<10>})$

Assume $J_{test}(w^{<5>}, b^{<5>})$ performs the best, we choose d=5

How well does the model perform? Directly report $J_{test}(w^{<5>}, b^{<5>})$?

No. It's likely to be an optimistic estimate.

Training, Cross validation, Test set.

J train   Jcv   J test

6   2   2

→ $J_{cv}(w^{<1>}, b^{<1>})$

→ $J_{cv}(w^{<10>}, b^{<10>})$

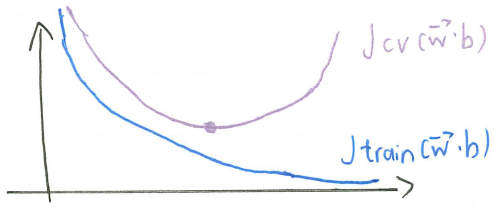
Pick d based on Jcv.

Estimate generalization error using test the set J test

Not diagnostic

KOKUYO

For model with	J_{train}	J_{cv}
High bias (underfit)	High	High
"Just right"	Low	Low
High variance (overfit)	Low	High

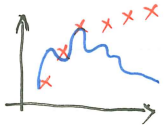


High bias: $J_{train} \approx J_{cv}$ (both high)

High variance: $J_{cv} >> J_{train}$ (J_{train} may be low)

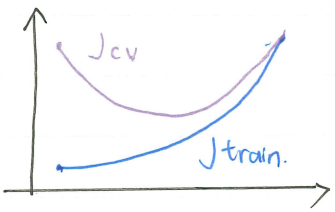
(Sometimes) High variance and high variance:

J_{train} will be high and $J_{cv} >> J_{train}$.

e.g  ∴

Linear regression with regularization.

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



↑ high bias (reason why it looks familiar)

No.

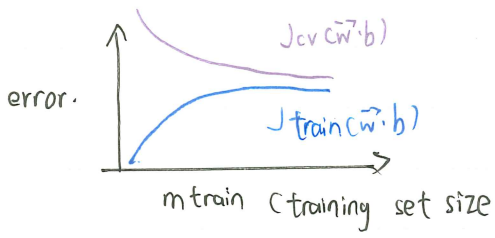
Date

Speech recognition example:

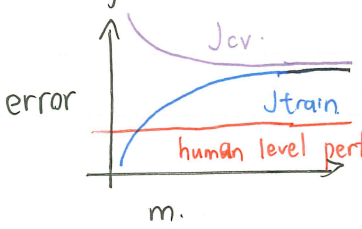
Human level performance:	10.6%		10.6%	10.6%
Training error:	10.8%	↕ 0.2%	15.0%	15.0%
Cross validation error:	14.8%	↕ 4.0%	15.5%	19.7%
		High variance	High Bias	Both High.

- Establish a baseline level of performance.
  - Human level performance
  - Competing model
  - Guess based on experience.

Learning curves

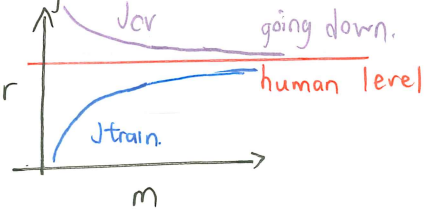


High bias:



If an algorithm has high bias,
It's useless to add data

High variance



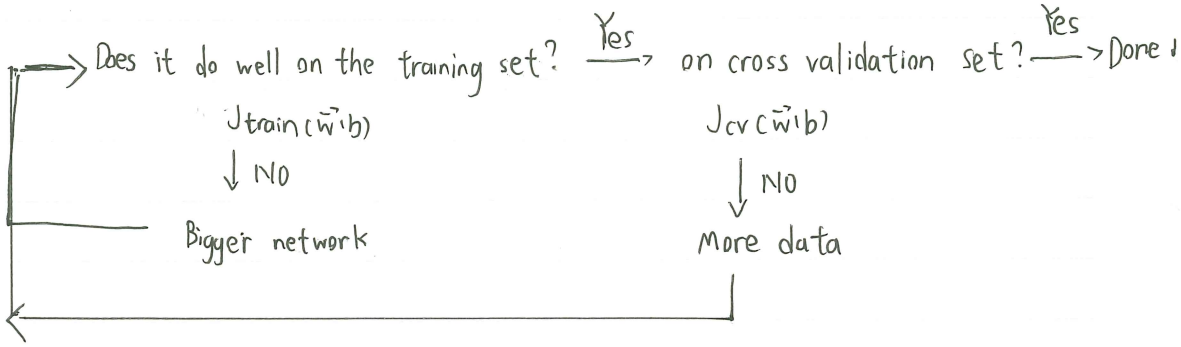
If has high variance,
getting more training data
is likely to help

Can be used to visualize, but computing differen subset of training set is expensive.

No.

Date

Neural networks and bias variance



By carefully choosing regularization, large model can perform well

Caveat: it's slow., but neural network is accurate with low bias

layer = Dense(units = 25, activation = "relu", kernel_regularizer = L2(0.01))

λ

Normally we don't regularize 'b', which may cause difference

Error Analysis

e.g. Manually anaylze misclassfied examples,
Categorize them based on common traits.
If there's too many, fit the model with less examples.
or employ more features
come up with specific solutions.

Add data:

Add more data of types where error analysis has indicated.

No.

Date

Data augmentation (audios and videos)

Modifying an existing training example to create a new training example

e.g. distort / add noise must has meaning

Data synthesis

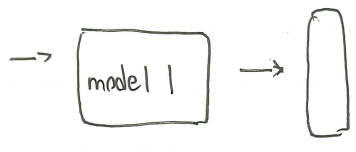
Generating data from scratch

e.g. (mostly visual) using fonts in the computer as photo

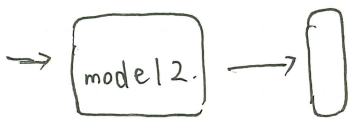
AI = Code + Data.

Conventional model-centric ↑ Data-centric approach

Transfer learning: using data from a different task.



supervised pretraining



fine tuning

Option 1: Only train output layers parameters works better for small sets

Option 2: train all parameters

∴ Use other's model, replace with own output layer./

do a little bit fine tune, further train the network

restriction: input type must be same

No.

Date

Full cycle.

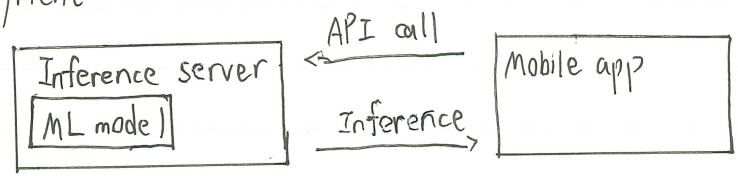
Scope project → Collect data → Train model →

↑ error analysis, iterative improvent

Deploy in production

monitor and maintain system

Deployment



Software engineering may be needed.

MLOps – machine learning operation

practice to make the model reliable, scales well, has good laws, is monitored, can be easily updated

Rare disease classification example — skewed datasets

99.5% accuracy 0.5% error less useful always print 'y=0'

99.0% accuracy 1% error more useful at least make diagnosis

Actual class

Predicted class	1	0
1	True positive	False positive
0	False negative	True negative

(1在变) (0不变)

Precision: $\frac{\text{True positive}}{\text{predicted positive}}$ detect if 1 all the time.

Recall: $\frac{\text{True positive}}{\text{Actual positive}}$ detect if 0 all the time

algorithm with low precision and recall is useless.

Logistic regression : $0 < f_{\vec{w},b}(\vec{x}) < 1$.

Predict 1 if $f_{\vec{w},b}(\vec{x}) \geq 0.5$

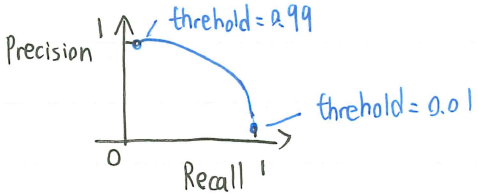
0 if $f_{\vec{w},b}(\vec{x}) < 0.5$ > threshold.

Suppose we want $y=1$ only if very confident

Raise threshold ⟶ higher precision, lower recall

Suppose we want to avoid missing too many cases

Reduce threshold ⟶ lower precision, higher recall



F_1 Score.

$$= \frac{1}{\frac{1}{2}\left(\frac{1}{P}+\frac{1}{R}\right)} = 2\frac{PR}{P+R}$$ (harmonic mean)

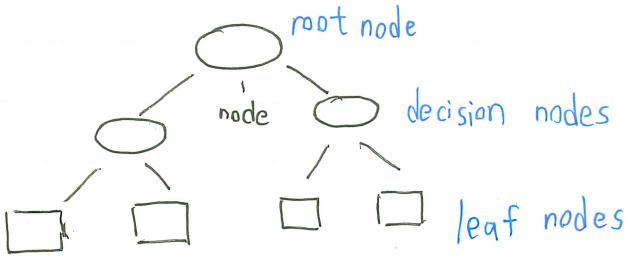
taking an average that emphasize the smaller values more.

Average $\frac{P+R}{2}$ is bad.

No.

Date

Decision tree



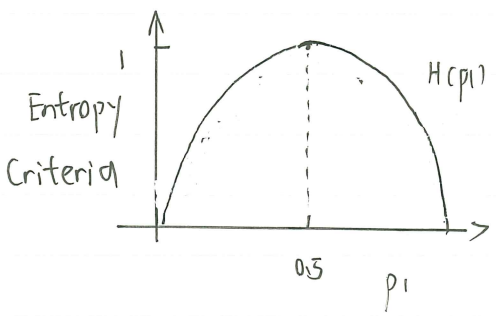
Key decision 1: How to choose what feature to split on at each node?

Maximize purity (minimize impurity (entropy))

Key decision 2: When do you stop splitting?

- When a node is 100% one class
- When splitting a node will result in the tree exceeding a maximum depth
- When improvements in purity score are below a threshold
- When number of examples in a node is below a threshold. (avoid overfitting)

Measure purity - entropy



$p_0 = 1 - p_1$ looks similar?

$$H(p_1) = -p_1 \log_2(p_1) - p_0 \log_2(p_0)$$
$$= -p_1 \log_2(p_1) - (1-p_1)\log_2(1-p_1)$$

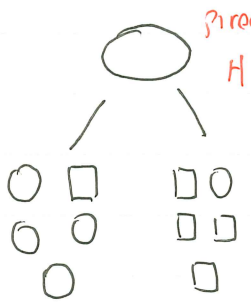
Note $0 \log_2(0) = 0$

Gini criteria is similar

No.

Date

Choosing a split- — Information Gain.



$p_1^{left} = \frac{4}{5}$ $p_1^{right} = \frac{1}{5}$

$w^{left} = \frac{5}{10}$ $w^{right} = \frac{5}{10}$

weight

Information gain=

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

- we measure the reduction (IG) to decide whether it's worth spliting
- we add weight because entropy with smaller samples is better than bigger samples

Decision Tree Learning

- Start at root node
- Pick the feature with highest information gain.
- Split dataset, create left and right branches of the tree.
- Keep repeating splitting process until stopping criteria
  - when a node is 100% one class
  - When splitting a node will result in the tree exceeding max depth
  - Information gain from additional splits is less than threshold.
  - When number of examples in a node is below a threshold.

Cover other subtrees and check every node!

No.

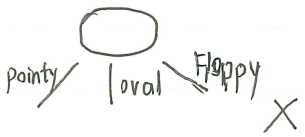
Date

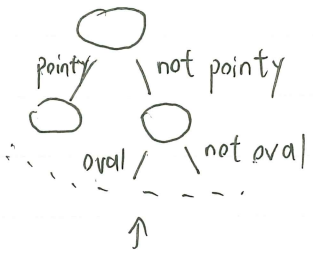
Algorithm that calls itself

Recursive splitting

Build the overall decision tree by build smaller sub-decision trees and then putting them together.

When encounter more than two features.

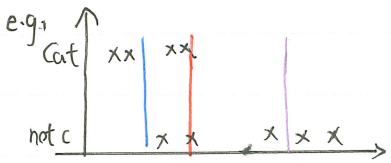




One hot encoding:

If a categorical feature can take on k values, create k binary features (0 or 1)

When encounter continuous features (numerical)



choose midpoints as possible splits

weight ≤ 8 $H(0.5) - (\frac{2}{10}H(\frac{2}{2}) + \frac{8}{10}H(\frac{3}{8})) = 0.24$

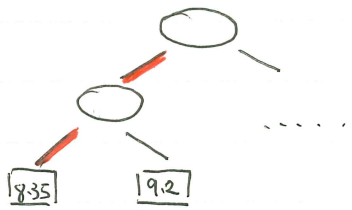
weight ≤ 9 $H(0.5) - (\frac{4}{10}H(\frac{4}{4}) + \frac{6}{10}H(\frac{1}{6})) = 0.61$ ✓

weight ≤ 16 (lbs) $H(0.5) - (\frac{7}{10}H(\frac{5}{7}) + \frac{3}{10}H(\frac{0}{3})) = 0.40$

No.

Date

Regression Trees.



7.2, 8.4, 7.6, 10.2 9.2 (weight of animal)

When given a new example, follow the decision nodes down as usual until it gets to a left node and predict the example's weight based on the average of the weights in leaf node

Choosing a split:

variance at root node 20.51.

variance:	1.47	21.87	27.80	1.37	0.75	23.32
weight	$\frac{5}{10}$	$\frac{5}{10}$	$\frac{7}{10}$	$\frac{3}{10}$	$\frac{4}{10}$	$\frac{6}{10}$

Reduction in variance.

① : $20.51 - (\frac{3}{10} \times 1.47 + \frac{5}{10} \times 21.87) = 8.84$ ✓

② : $20.51 - (\frac{7}{10} \times 27.8 + \frac{3}{10} \times 1.37) = 0.64$

③ : $20.51 - (\frac{4}{10} \times 0.75 + \frac{6}{10} \times 23.32 = 6.22$

Pick the one with highest variance reduction.

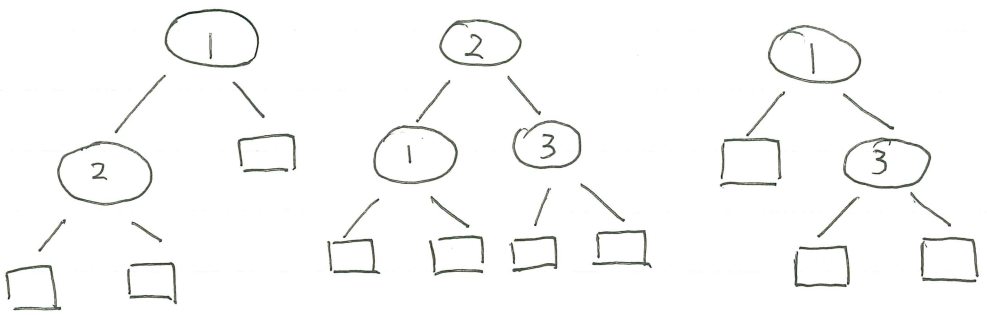
No.

Date

- Trees are highly sensitive to small changes of data.

Tree ensemble

have a lot of decision trees that do the same thing



Prediction : 1 0 1

Final Prediction : 1. vote for the final decision.

Sampling with replacement : tokens can be put back

- Random Forest Algorithm:

Given training set of size m

For $b=1$ to B set B larger never hurt performance, but too large slows computation

Using sampling with replacement to create a new training set of size m.

Train a decision tree on the new dataset

Bagged decision tree

Randomizing feature choice. often $k=\sqrt{n}$

For n features available, pick random subset of $k<n$ and allow the algorithm to only choose from that subset of features

Campus It causes the algorithm to explore a lot of small changes and average it to reduce impact

No.

Date

Boosted trees intuition.

while applying . Random Forest Algorithm

instead of picking from all examples with equal ($\frac{1}{m}$) probability, make it more likely to pick misclassified examples from previously trained trees.

Build Tree ensemble on 1, 2 ... b-1

when in the b^{th}, apply . boosting .

XGBoost (extreme Gradient Boosting)

- open source
- efficient
- good spliting criteria
- built in regularization
- competitive in competitions such as Kaggle
- assign different ways to different training examples.

```
from xgboost import XGBClassifier / XGBRegressor
model = XGBClassifier() / XGBRegressor()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

sklearn. GridSearchCV can refit parameters to best combination

No.

Date

Decision Tree and Tree ensembles

- works well on tabular (structured) data. (can be stored in spread sheet format)
- not recommended for unstructured data (image, audio, text)
- Fast.
- small decision trees may be human interpretable.

Neural networks

- works well on all types of data.
- maybe slower.
- works with transfer learning
- when building a system of multiple models working together, it might be easier to string together multiple neural networks

No.

Date

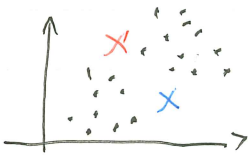
Unsupervised model

Clustering

K-means

1: Assign points to cluster centroids

2: Move cluster centroids by taking average. > Repeat



if not data is assigned to a centroid, delete it.

K-means algorithm.

Randomly initialize K cluster centroid $\mu_1, \mu_2 \dots \mu_K$

Repeat. {

Assign points to cluster centroids.

for i=1 to m.

$c^{(i)}$:= index of cluster centroid closed to $x^{(i)}$

Move cluster centroids.

for k = 1 to K

μ_k := average of points in cluster k.

}

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

K-means always converges

Random initialization:

Choose $K < m$ (if $K > m$, there won't be enough data)

Randomly pick K training examples

Set $\mu_1 \dots \mu_K$ equal to these examples

Might end up in local minimum, so compute cost and select.

For i = 1 to 100

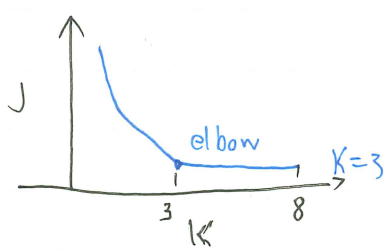
50~1000

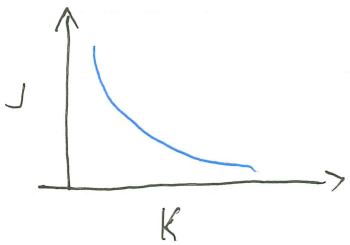
No.

Date

Choose the value of K

Elbow method





But often it has no clear elbow

The right "K" is ambiguous. don't choose K just to minimize cost J

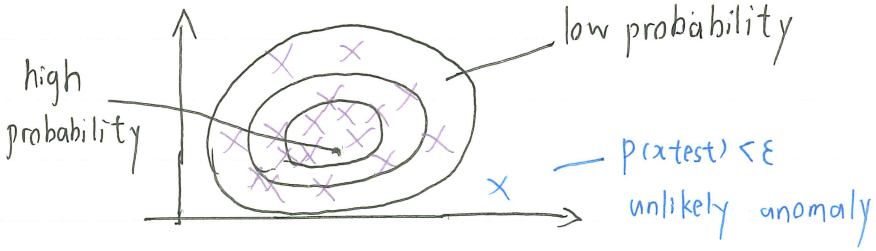
Evaluate K-means based on how well it performs on that later purpose

Based on how well it makes sense in practise.

e.g. image compression

Anomaly detection

Density estimation

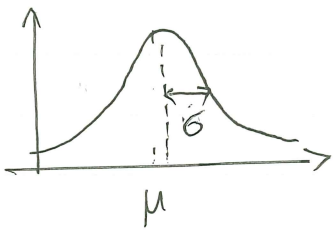


e.g: Fraud detection / airplane engine manufacturing

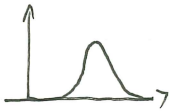
No.

Date

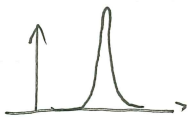
Gaussian(Normal) Distribution.



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

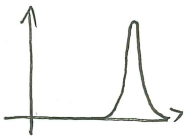


$\mu=0 \quad \sigma=1$



$\mu=0 \quad \sigma=0.5$

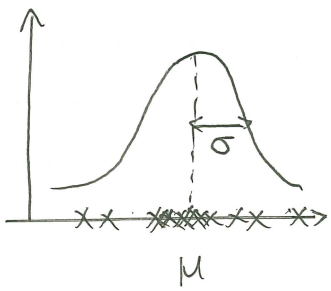
$\mu=0 \quad \sigma=2$



$\mu=3 \quad \sigma=0.5$

Parameter estimation.

Dataset: $\{x^{(1)}, x^{(2)}, \cdots, x^{(m)}\}$ (x has only one feature!)



$$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)^2$$

maximum likelihood for μ, σ,

but fine here, no need $\frac{1}{m-1}$

No.

Date

Density estimation.

Training set: $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \dots \vec{x}^{(m)}\}$

Each example $\vec{x}^{(i)}$ has n features $\quad \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$

Assume $x_1 \dots x_n$ are independent, but actually algorithm work even not

$$p(\vec{x}) = p(x_1; \mu_1, \sigma_1^2) * p(x_2; \mu_2, \sigma_2^2) * \dots * p(x_n; \mu_n, \sigma_n^2)$$

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

Algorithm.

1. choose n features x_i
2. Fit parameters $\mu_1, \dots, \mu_n, \sigma_1^2, \dots \sigma_n^2$

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

(vectorized : $\vec{\mu} = \frac{1}{m} \sum_{i=1}^{m} \vec{x}^{(i)}$ $\vec{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix}$

3. Given new examples x, compute $p(x)$

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

4. Anormaly if $p(x) < \varepsilon$

e.g.