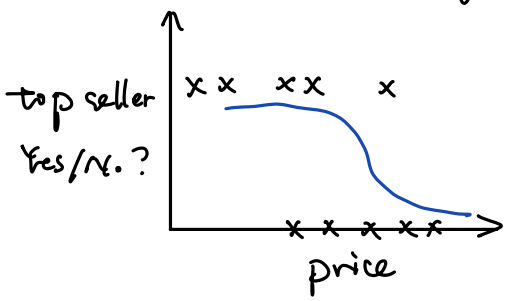
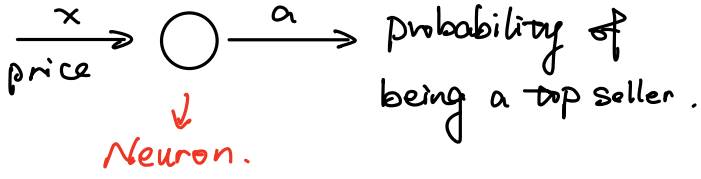
Demand Prediction.

e.g. You are selling T-shirts. Want to decide which will be top-seller



input: x = price.

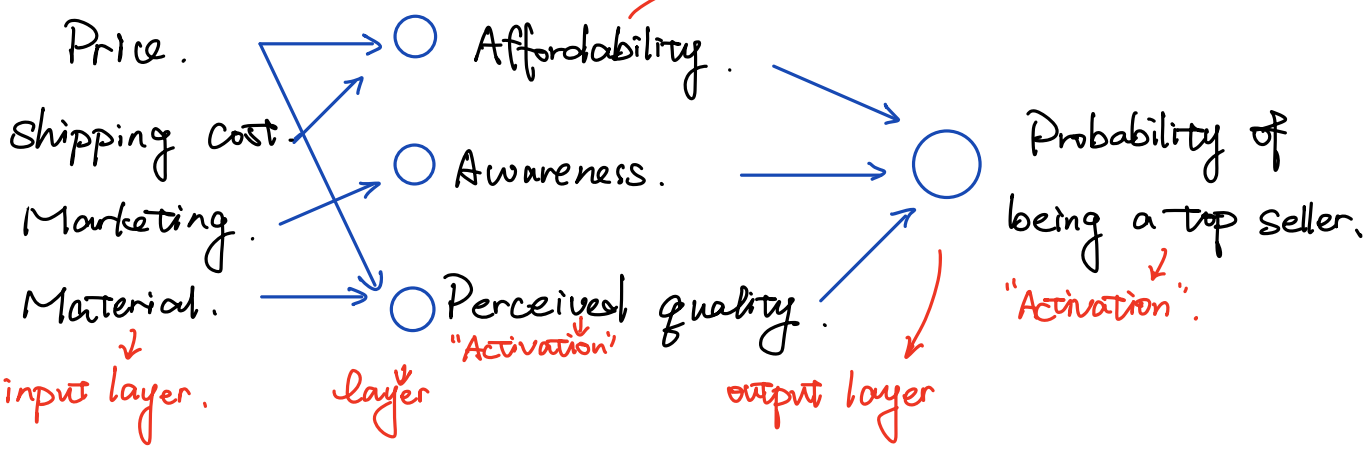
output: $a = f(x) = \frac{1}{1+e^{-(wx+b)}}$



Neuron.

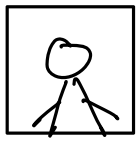
Now we have more feature:

e.g. Affordability is a function of price and shipping cost



Example: Image Recognition.

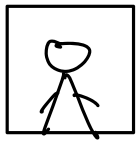
- face recognition.



pixel bright value range from 0 to 255.

transform into list:

This is the input.



→ layer → layer → ··· → layer → output layer → probability of being 人?

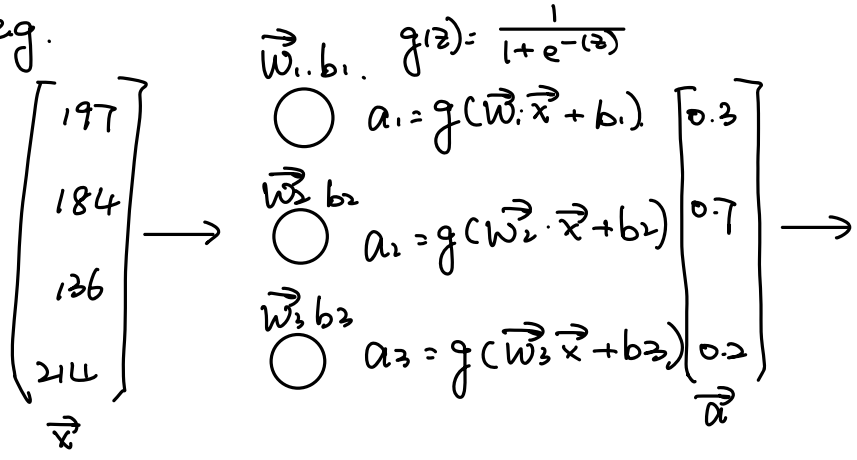
To visualizing the hidden layer.

① first hidden layer. You might find vertical lines. horizontal lines ···

② Second hidden layer. Short lines grouped together. To form shapes like eyes. nose.

② Later layers: Aggregates part · get faces: finally decide the prob.
Later look for larger part of the picture.

Neural Network Layers



Notation for layer numbering: $[l]$. eg. $\vec{w}_1^{[l]}$. $a_1^{[l]}$. Param and output in layer 1.
Finally at layer l. We have a scalar $a^{[l]}$. to make prediction.

General Notation: $a_j^{[l]} = g(\vec{w}_j \cdot \vec{a}^{[l-1]} + b_j^{[l]})$.

- $a_j^{[l]}$: Activation value of layer l. Unit
- g: Act. func.
- $\vec{a}^{[l-1]}$: Output vec from previous layer

Model Training Steps.

	logistic reg.	Neural Network.
① Specify how to compute output given input x and params w, b. (define model. $f_{\vec{w},b}(\vec{x}) = ?$	$z = np.dot(w, x) + b$. $f\_x = 1/(1+np.exp(-z))$.	model = Sequential([Dense(…) Dense(…) …])
② Specify loss and cost. $L(f_{\vec{w},b}(x), y)$. $J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m} L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})$	logistic loss: loss = $-y^*$ np.log(f_x) $-(1-y)^*$ np.log(1-f_x)	binaryCrossEntropy. model.compile(loss = BinaryCrossentropy())

③ Train on data to minimize $J(\vec{w}, b)$. e.g. GD.	$w = w - \text{alpha} * d_j\_dw$. $b = b - \text{alpha} * d_j\_db$.	model.fit(X, y, epoch=100)

Choosing Activation Function.

① Consider the output layer.

→ If you are working with binary classification.

Using Sigmoid as the output activation function.

→ Regression. Either + or − (e.g. Stock pred.)

Using linear activation function.

→ Regression Either 0 or +.

Using ReLU function.

② Hidden layer.

Most common choice: Using ReLU much more often.

- Reason:

① ReLU faster to compute.

② Completely flat one side. But Sigmoid flat only at two ended.
$\frac{\partial}{\partial w} J(w, b) \approx 0$ when $g(z)$ is flat.

Improved Softmax Implementation.

① $x = \frac{2}{10000}$

② $x = (1 + \frac{1}{10000}) - (1 - \frac{1}{10000})$

in computer. they are different: 0.0002000... 0.0001999999...78...

Numerical Roundoff errors.

More numerically accurate implementation of logistic loss.

Logistic Reg.

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

Original Loss: $loss = -y\log(a) - (1-y)\log(1-a)$.

More accurate loss (in code).

$$loss = -y\log\left(\frac{1}{1+e^{(-z)}}\right) - (1-y)\log\left(1-\frac{1}{1+e^{(-z)}}\right).$$

So. instead of: model.compile(loss=BinaryCrossEntropy())

we do: model.compile(loss=BinaryCrossEntropy(from_logits=True))

And in the last layer. we just use 'linear' activation function instead of 'sigmoid'

Same for softmax classification for multiple classes.

Multi-label classification.

e.g. Camera:

Is there a bus ?
a car ?
a pedestrain ?

$$Y = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \begin{matrix} \text{Yes} \\ \text{No} \\ \text{Yes} \end{matrix} \qquad Y = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \begin{matrix} \text{Yes} \\ \text{No} \\ \text{No.} \end{matrix}$$

You can build 3 Neural network. To predict separately.

Alternative: One NN with three outputs.

for each output. we all use sigmoid.

Say 3rd layer is output:

$$\vec{a}^{[3]} = \begin{bmatrix} a_1^{[3]} \\ a_2^{[3]} \\ a_3^{[3]} \end{bmatrix} \begin{matrix} \text{Car} \\ \text{Bus} \\ \text{Pedestrain.} \end{matrix}$$

Multi-label and Multi-class are different!

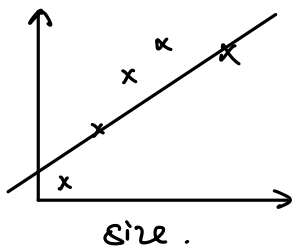
Evaluate the Model.

If you only have 3 or less features. You can plot them to see whether they overfit the training set.

Technique: Train / Validation. 70% / 30%. e.g.

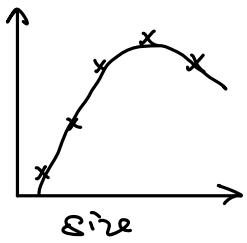
Bias and Variance. 偏差与方差.

e.g.



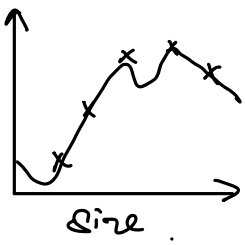
$f_{\vec{w},b}(x) = w_1 x + b$.

High bias. (underfit).



$f_{\vec{w},b}(x) = w_1 x + w_2 x^2 + b$.

Pretty Good.



$f_{\vec{w},b}(x) = w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$.

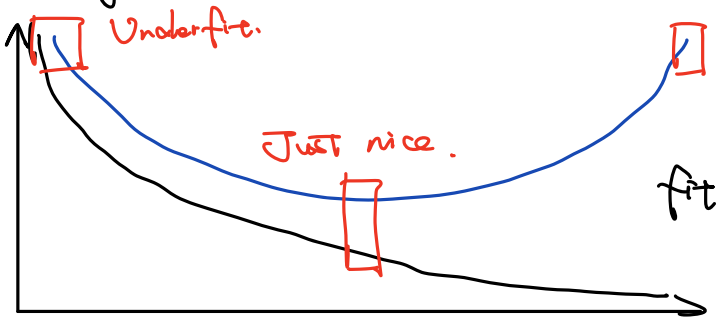
High Variance. (Overfit).

But. Still. one feature. we can plot. More features. Can't plot.

J_{train} is high	J_{train} is low	J_{train} is low
J_{cv} is high	J_{cv} is low.	J_{cv} is high.

↓ cross validation

Understanding Bias and Variance.



degree of polynomial. d.

① High bias ⇒ J_{train} high. ($J_{train} \approx J_{cv}$).

② High Variance. ⇒ $J_{cv} \gg J_{train}$. (J_{train} may be very low).

①② may happen simultaniously. e.g: Overfit part of the input.
& underfit part of the input.

Consider the relation with the regularization param λ.

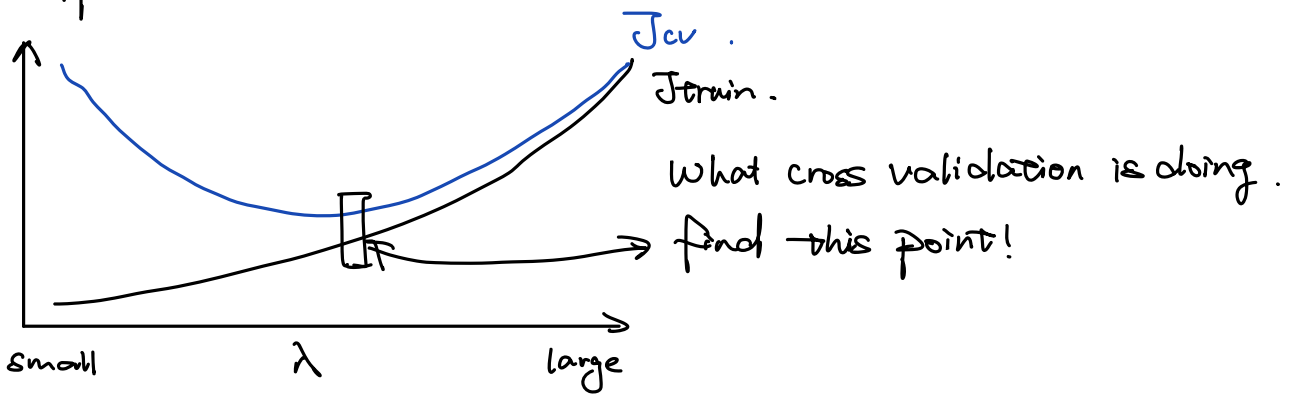
①. λ large. $\lambda \approx 1000$. High bias. $w_1, w_2, \cdots, w_n \approx 0$. $f_{\vec{w},b}(x) \approx b$.

② λ small. Might Overfit.

Cross validation will give you good choice of λ.

After we decide what degree (which model) we are going to use.

Try diff. λ.



Problem: How do we judge J_{train}. J_{cv} high and low?

↓

Establishing a baseline performance level.

e.g. Speech recognition:

J_{train}: 10.8% Human level: 10.6%.

J_{cv}: 14.8%.

Say if J_{train} high. we can judge by whether J_{train} is large than human level performance a lot.

Reasonable baseline.

→ Human level performance. e.g. speech. image. language.

→ Competing alg. performance.

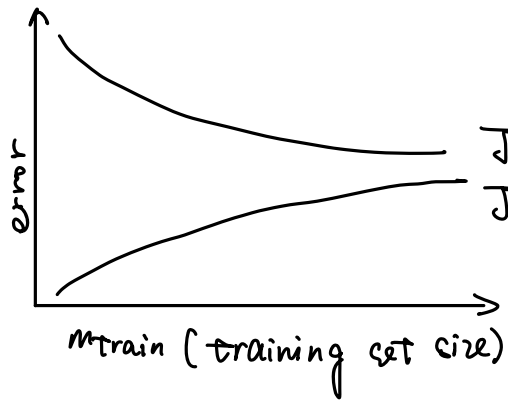
→ Guess based on experience.

Diff between baseline and J_{train}: High ⇒ High bias.

Diff between J_{train} and J_{cv}. High ⇒ High variance.

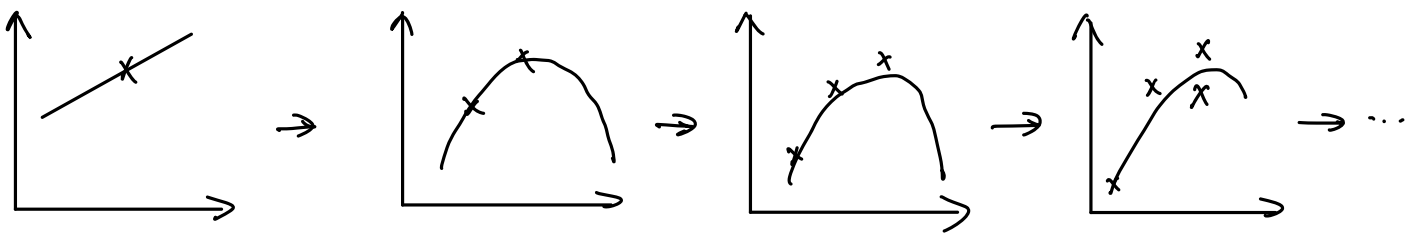
Learning Curves.

Suppose we are trying to fit a quad. func. $f_{\vec{w},b}(x) = w_1 x + w_2 x^2 + b$.



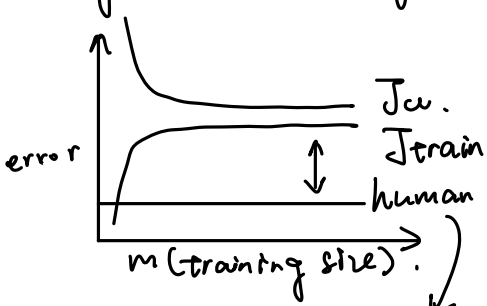
Why's the case.

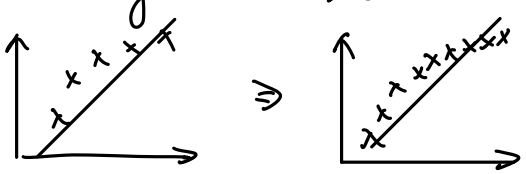
example ↑. J_{cv} ↓. J_{train} ↑.



with # train examples ↑. harder → perfect fit.

High bias. (e.g. fitting points using linear).

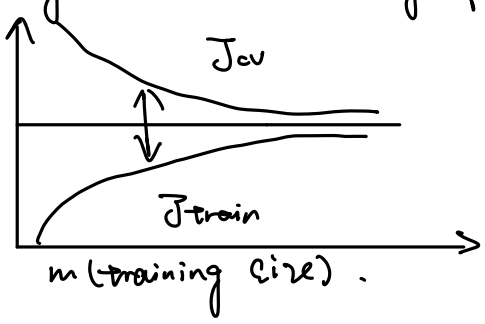


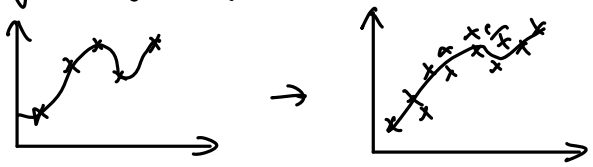


If an alg. suffers from high bias. Increasing # of training examples will not help much.

Flatten after a while

High variance. (e.g. fitting using high degree).





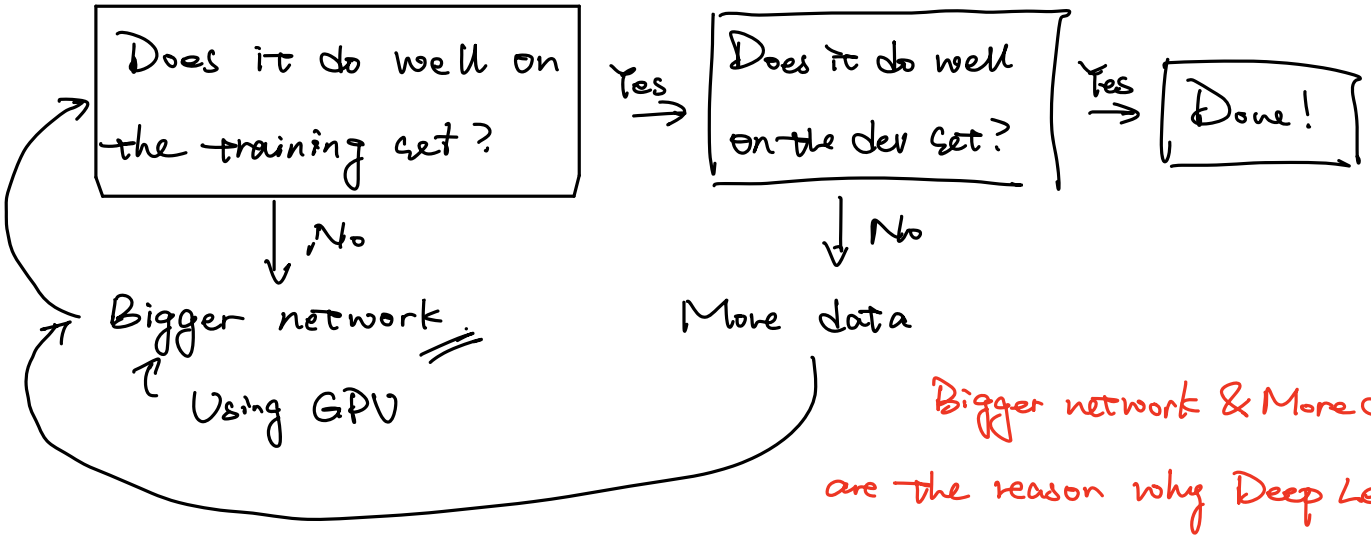
If an alg. suffer from high variance. then increase #train example will really help.

What to do next?

High bias	High variance.
Adding additional features.	Get more training examples.
Adding polynomial features	Try smaller set of features
Try decreasing λ	Try increasing λ.
↓ More attention on func.	
⇓	⇓
Need complex/powerful model.	Need simplify the model.

Bias & Variance with NN

Large NN are low bias machine: Large enough, always can fit your train set.



Bigger network & More data are the reason why Deep Learning rise recent years.

It turns out that a large NN will usually do as well or as better as a small NN as long as regularization term being chose appropriately.

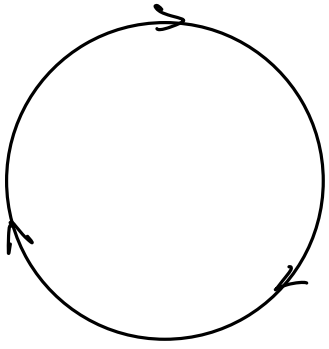
* larger NN almost don't hurt your learning. (But slow down).

Code: Dence(unit=25, activation = "relu", kernel_regularizer=L2(0.01))

Iterative loop of ML development.

Choose architecture.

(model, data, etc).



Diagnostics.

(bias, variances.

error analysis).

train model.

Error Analysis. (e.g. Span email detect)

m_{cv} = 100 examples in cross validation set.

Alg miss classified 100 of them.

Manually examine 100 examples and categorize them based on common traits.

e.g. Pharma. 21 → More data / features

Deliberate misspelling: 3.

Unusually email routing: 7

Steal passwords: 18.

Spam message in embedded images: 5

Can have overlapping!

.....

Based on the error analysis. we can tell what is the prob. (e.g. Bias & Variance)

Adding data.

Augmentation: modifying an existing training example to create a new training example.

e.g. Error analysis told you that a subset of the training is doing poorly. Maybe adding more data on that subset only will help.

e.g. of Augmentation: (Image).

Rotating. Mirror. Enlarge. Shrink. Adding noise. .....

Placing a grid. And do some distortions.

A ⇒ 

e.g. Data aug. for speech.

Noisy background. ① Crowd ② Car ③ bad connection ....

Usually doesn't help adding purely random/meaningless noise.

Engineering the data used by your system.

Conventional model-centric approach: AI = (Code) + Data.

↑ work on this

Data-centric approach. AI = Code + (Data)

↑ work on this

Transfer learning.

Suppose you are trying to work on digits recog.

But you don't have much data.

You have 1 million images of cats & dgs

You can train a DNN on these images ⇒ (Supervised pretraining).

Then, change the last output layer to digits. (10 units).

Option ①. Only train the output layer ⇒ (fine tuning).

Option ②. Train all the parameters.

* Even unrelated tasks can help.
* You can even use networks online, others trained for weeks. to fit your own network.

Why this work!?

Recall in DNN with images. Early layer only recognize simple things.

e.g. First layer: Edges. Second: Corners Third: Curves.

Early layers learning generic features.

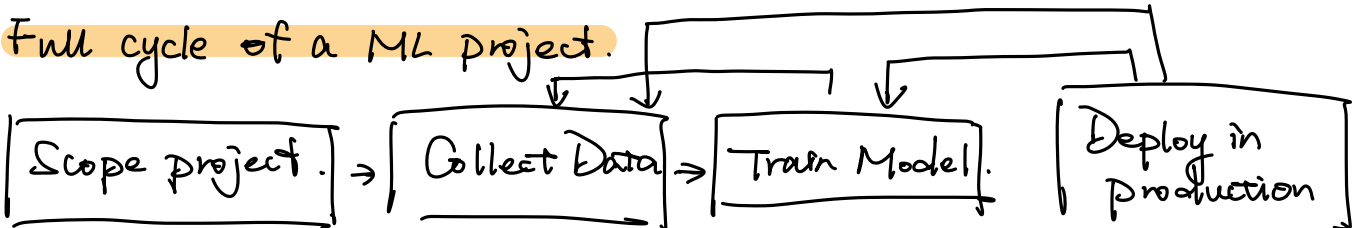
So we need use pre-trained network on the same kind of tasks.

Summary: ① Pre-train (Download) ② Fine-tuning on your own.

e.g. 1M examples. 10 examples.

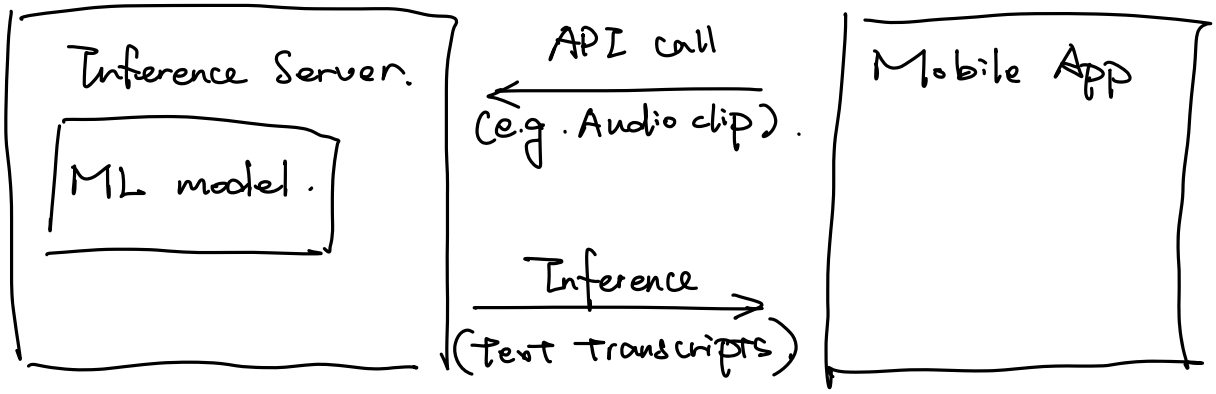
e.g. NN: BERT. GPT-3.

Full cycle of a ML project.



Define project. | Define and collect data | Train. error analysis & iterative improvement. | Deploy. monitor. and maintain system.

Deployment:



SE may be needed for:

→ Ensure reliable and efficient predictions.

→ Scaling.

→ logging.

→ System monitoring.

→ Model updates.

MLOps: machine learning operations.

Trading off between precision and recall.

precision = true pos / predicted pos.

recall = true pos / actual pos.

→ Suppose we want to pred. $y=1$ (rare disease) only if very confident.

Pred 1 if $f_{\vec{w},b}(x) \geq 0.7$. 0 if $f_{\vec{w},b}(x) < 0.7$.

⇒ higher precision. lower recall.

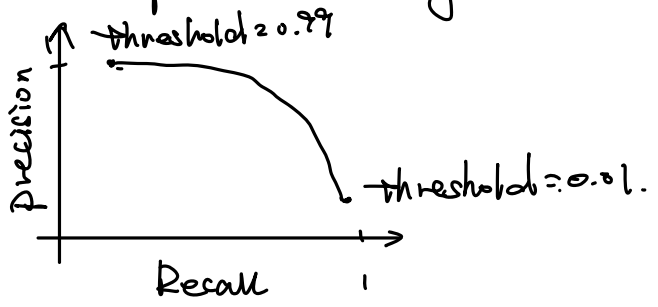
Because when you predict 1. it is more likely to be really 1.

→ Suppose we want to avoid missing too many cases of rare disease.

(when in doubt predict $y=1$).

Pred 1 if $f_{\vec{w},b}(\vec{x}) \geq 0.3$. 0 if $f_{\vec{w},b}(\vec{x}) < 0.3$.

lower precision. high recall.



F1-score used for "balance" to have a choice.

F1-score pays more "attention" on which one is lower.

$$F_1 = \frac{1}{\frac{1}{2}\left(\frac{1}{P}+\frac{1}{R}\right)} = 2\frac{PR}{R+P}$$

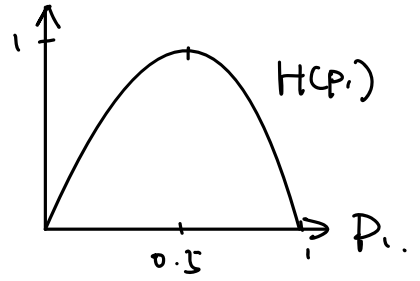
Harmonic mean

Decision Tree Model.

Measuring purity.

Entropy as a measure of impurity.

P_1 = fraction of examples that are cats.



Pure when $P_1 = 0$ or 1.

Most impure when $p_1 = 0.5$.

Define: $P_0 = 1 - P_1$.

$$H(P_1) = -P_1\log_2(P_1) - P_0\log_2(P_0)$$

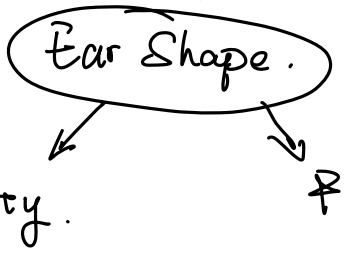
$$= -P_1\log_2(P_1) - (1-P_1)\log_2(1-P_1).$$

Actually. this definition won't allow $P_1 = 1$ or 0. we just define as 0.

"$0\log 0 = 0$" by convention.

Choosing a split: information gain.

e.g. Cat classification:



4/5. 1/5

$H(0.8) = 0.72$ $H(0.8) = 0.72$.

$\left(\frac{5}{10}H(0.8) + \frac{5}{10}H(0.2)\right) =$

4/7 1/3

$H(\frac{4}{7}) = 0.99$ $H(0.33) = 0.92$.

$\left(\frac{7}{10}H(0.57) + \frac{3}{10}H(0.33)\right)$

$\frac{3}{4}$ $\frac{2}{6}$

0.75 0.33

$H(0.75) = 0.81$ $H(0.33) = 0.92$.

$\frac{4}{10}H(0.75) + \frac{6}{10}H(0.33)$.

So which one do we want to use?

We want to use weighted average!

We want to compute the reduction of entropy compare to previous entropy.

At root node. we started with 10 animals.

5 cats and 5 dogs.

$P_i = 5/10 = 0.5$. $H(0.5) = 1$. Maximum impurity

Information Gain.

Ear shape : $H(0.5) - \left(\frac{5}{10}H(0.8) + \frac{5}{10}H(0.2)\right) = 0.28$ ✓ Split here.

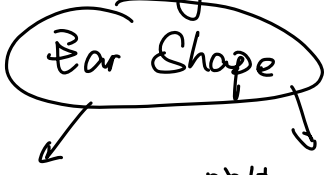
Face shape : $H(0.5) - \left(\frac{7}{10}H(0.57) + \frac{3}{10}H(0.33)\right) = 0.03$

Whiskers : $H(0.5) - \frac{4}{10}H(0.75) + \frac{6}{10}H(0.33) = 0.12$

Why we computing reduction not pick the lowest?

Because one of the way to make decision whether split further is to see if the reduction is smaller than certain threshold.

Information gain.



$P_i^{left} = 4/5$. $P_i^{right} = 1/5$

Def: Information gain:

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

$w^{left} = 5/10$. $w^{right} = 5/10$.

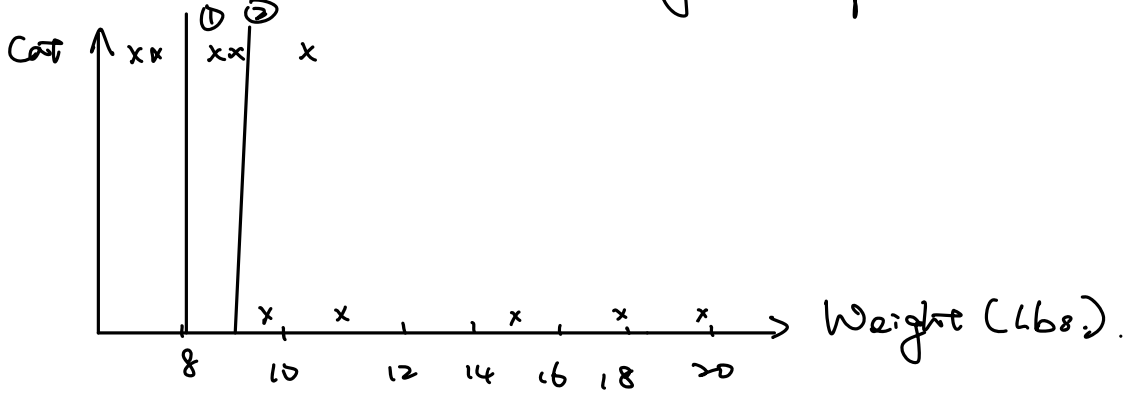
↓ examples that go here

Compare to the root.

Decision Tree Learning.

- Start with all examples at the root node.
- Calculate information gain for all possible features, and pick the one with the highest information gain.
- Split dataset according to selected feature, and create left and right branches of the tree.
- Keep repeating splitting process until stopping criteria is met.
  - ↳ When a node is 100% one class.
  - When splitting a node will result in the tree exceeding a maximum depth.
  - Information gain from additional splits is less than threshold.
  - When number of examples in a node is below a threshold.

Continuous Valued Feature. We just do splits!



① Weight ≤ 8 lbs. The information gain: $H(0.5) - \left(\frac{2}{10}H(\frac{2}{2}) + \frac{8}{10}H(\frac{3}{8})\right) = 0.34$

② Weight ≤ 9 lbs. $H(0.5) - \left(\frac{4}{10}H(\frac{4}{4}) + \frac{6}{10}H(\frac{1}{6})\right) = 0.61$.

③ Weight ≤ 13 lbs. $H(0.5) - \left(\frac{7}{10}H(\frac{5}{7}) + \frac{3}{10}H(\frac{0}{3})\right) = 0.4$

We can use each feature as the splitting point. Compute the info. gain.

e.g. 10 examples. we may have 9 possible splits points to compute.

Regression Tree. Not only category. but also predict number.

e.g. We want to predict the cat weight.

Keep spliting. and calculate the average weight.

Then how to split?

Ear Shape.

- pointy: weight: 7.2, 9.2, 8.4, 7.6, 10.2 → Variance: 1.47 → $w^{left} = \frac{5}{10}$
- floppy. 8.8, 15, 11, 18, 20 → 21.87 → $w^{right} = \frac{5}{10}$

$\frac{5}{10} \cdot 1.47 + \frac{5}{10} \cdot 21.87$.

Face Shape.

- Round: 7.2, 15, 8.4, 7.6, 10.2, 18, 20 → 27.80 → $w^{left} = \frac{7}{10}$
- Not Round. 8.8, 9.2, 11. → 1.37 → $w^{right} = \frac{3}{10}$

$\frac{7}{10} \cdot 27.80 + \frac{3}{10} \cdot 1.37$.

Whiskers.

- Present: 7.2, 8.8, 9.2, 8.4 → 0.75 → $w^{left} = \frac{4}{10}$
- Absent. 15, 7.6, 11, 102, 18, 20. → 23.32 → $w^{right} = \frac{6}{10}$

$\frac{4}{10} \cdot 0.75 + \frac{6}{10} \cdot 23.32$.

* Variance plays the role of information gain.

We still consider the reduction of variance.

Root variance: 20.51.

Ear Shape. Variance reduction: $20.51 - (\frac{5}{10} \cdot 1.47 + \frac{5}{10} \cdot 21.87) = 8.84$.

Face Shape: ... = 0.64

Whiskers: ... = 6.22.

So "Ear Shape" gives us the largest variance reduction. we take that as root.

The final prediction is the average

Tree ensembles.

Trees are highly sensitive to small change of the data.

Maybe change only one example cause the root split on another feature.

And finally the tree looks totally different.

→ Tree ensemble. Multiple trees prediction result.

And we just vote!

Make entire alg. less sensitive to simple change in dataset.

Technique. Sampling with Replacement.

Take one out. put back. take another.

Take our 10 dogs and cats in a bag.

Create a new training set with just sampling with replacement.

(Might have repeating e.g.s).

Random forest alg.

Given training set m. (Bagged decision tree).

For b = 1 to B.

Use sampling with replacement to create a new training set size m

Train a decision tree on the new dataset

* It turns out increasing B barely hurt performace. (But longer to compute)

It also turns out. we usually start with the same root node.

→ Randomizing the feature choice.

At each node. when choosing a feature to use to split. If n features are available. pick a random subset of $k<n$ features. and allow the algorithm to only choose from that subset of features

e.g. $k = \sqrt{n}$

XGBoost.

Remind:

Given training set m. (Boosted tree intuition)..

For b = 1 to B.

Use sampling with replacement to create a new training set size m

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

Think this as: deliberate practice. More attention on a subset we are not doing well at.

Train a decision tree on the new dataset

→ XGBoost. (eXtreme Gradient Boosting).

When to use DT.

DT vs NN.

- DT and tree ensembles:
  - → Work well on tabular (structured) data.
    - → Not good on unstructured data. (e.g. images. audio. text).
  - → fast.
  - → Small DT may be human interpretable.
- NN.
  - → Works well on all types of data

→ Train slower

→ Works with transfer learning.

→ When building a system of multiple models working together. it might be easier to string together multiple NNs.