Evaluating a learning algorithm

What happens when our learning algorithm does not predict well? What can we do?

→ You can try:
- Get more training data
- Try smaller sets of features
- Try getting additional features
- Try with polynomial features
- Try increasing/decreasing λ

But which one? Intuition? ← Better run diagnostic tests and get insights about learning alg. performance so we know how to improve it.

Evaluate a hypothesis

- Split the data set into:
  - training set (~70%)
  - test set (~30%)

← random selection is better

- Training/testing for linear regression
  - Learn Θ from training data minimizing $J(\Theta)$ (from 70% training data)
  - Compute test set error → $J_{test}(\Theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} \left(h_\Theta(x_{test}^{(i)}) - y_{test}^{(i)}\right)^2$

- Training/testing for logistic regression → usual formula without regul. term
  - Learn Θ from training data minimizing $J(\Theta)$ (from 70% training data)
  - Compute test set error → $J_{test}(\Theta) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} y_{test}^{(i)} \log h_\Theta(x_{test}^{(i)}) + (1 - y_{test}^{(i)}) \log h_\Theta(x_{test}^{(i)})$

  or

  Compute misclassification error:

$$err(h_\Theta(x), y) = \begin{cases} 1 \text{ (error)} & \text{if } h_\Theta(x) \geq 0.5, y = 0 \\ & \text{or if } h_\Theta(x) < 0.5, y = 1 \\ 0 \text{ (correct)} & \text{otherwise} \end{cases}$$

$$\text{Test error} = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err\left(h_\Theta(x_{test}^{(i)}), y_{test}^{(i)}\right)$$

= average number of misclassifications

Model selection with train/cross validation/test sets

We might want to use different models (e.g. different polynomial degrees). Each model gives a set of Θ, which one do we choose?

↓

We can select the model that has a lower error in the test set

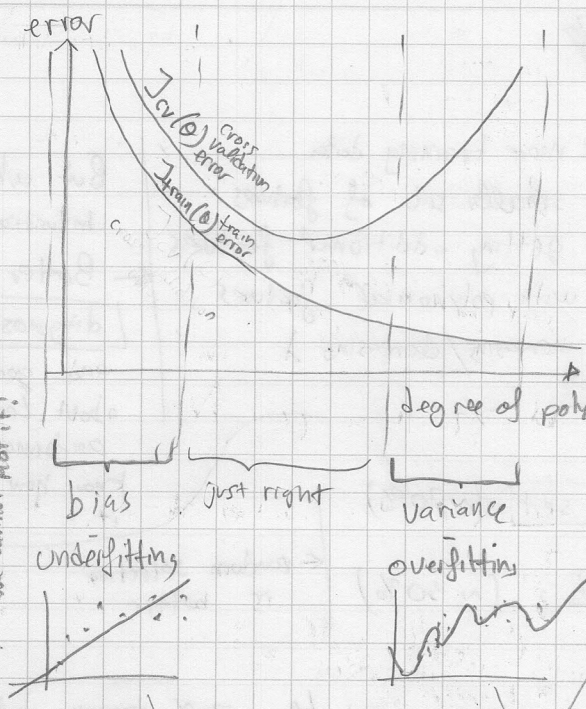
→ but we are using the test set on our learning process! :(

actually to fit the parameter d (poly. degree)

60% train ← train and get Θ for all models (fit Θ)
20% cross-validation ← choose the best model here! (fit d)
20% test ← then finally test the chosen model

Diagnose Bias (underfit) vs. Variance (overfit)

what does this mean?
bias ↔ underfit
variance ↔ overfit



would be nice to see how under/over-fitting a model is, but if we have more than 1-2 features... we cannot plot it!

If our learning algorithm is not working well (J_{cv} or J_{test} are high), are we suffering from bias (underfit) or variance (overfit) problem?

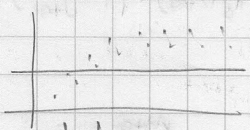
J_{train} & J_{cv} high $\Rightarrow$ high bias (underfit)

J_{train} low & J_{cv} high $\Rightarrow$ high variance (overfit)

How to pick the regularization parameter λ?

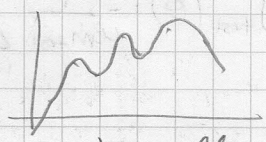
← regularization is used to prevent overfitting...

If λ is too large we 'kill' all $\theta_1 \dots \theta_n$ and $h_\theta(x) = \theta_0$

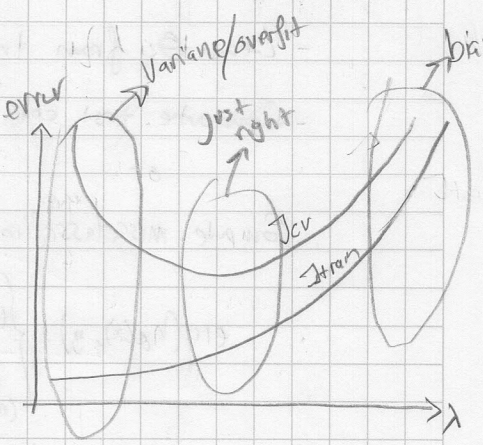


λ large
high bias
underfit

if λ is too small $\theta_1 \dots \theta_n$ are not regularized.



λ small
high variance
overfit



Try:

(1) $\lambda = 0 \longrightarrow \min J(\theta) \rightarrow \theta^{(1)} \rightarrow J_{cv}(\theta^{(1)})$
(2) $\lambda = 0.01 \longrightarrow$ " $\rightarrow \theta^{(2)} \rightarrow J_{cv}(\theta^{(2)})$
(3) $\lambda = 0.02 \longrightarrow$ " $\rightarrow \theta^{(3)} \rightarrow$
(4) $\lambda = 0.04 \longrightarrow$ " $\rightarrow \theta^{(4)} \rightarrow$
(5) $\lambda = 0.08 \longrightarrow$ " $\rightarrow \theta^{(5)} \rightarrow$
⋮
(12) $\lambda = 10.24 \longrightarrow$ " $\rightarrow \theta^{(12)} \rightarrow J_{cv}(\theta^{(12)})$

doubling...

minimum $J_{cv}(\theta^{(k)}) \rightarrow J_{test}(\theta^{(k)})$ is the error

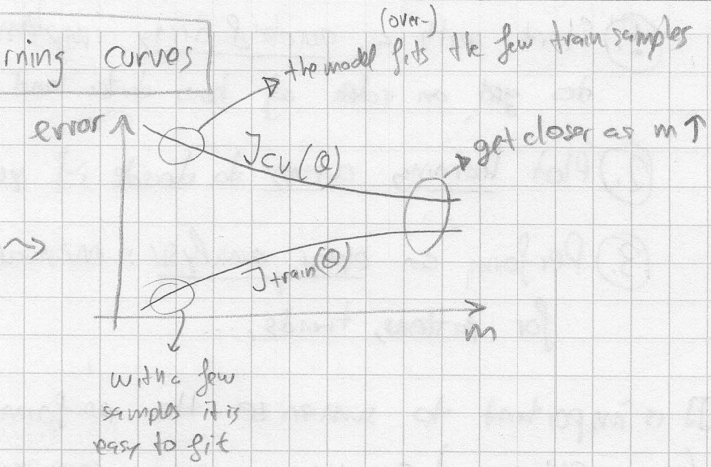
we fit 'k' to J_{cv}

J_{cv} neglects λ factor (just squared error)

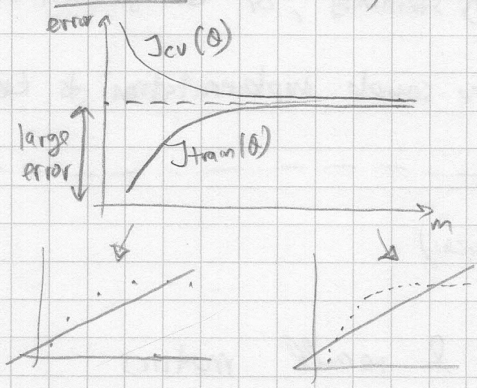
$$J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum (h_\theta(x) - y)^2$$

Diagnosing Bias/Variance with learning curves

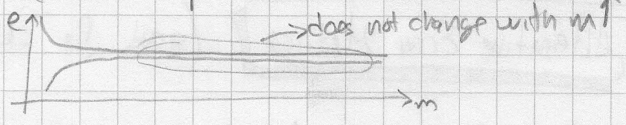
A learning curve plots the error for different ~~m~~ sizes of the training set (m). That is, using more samples or less samples.



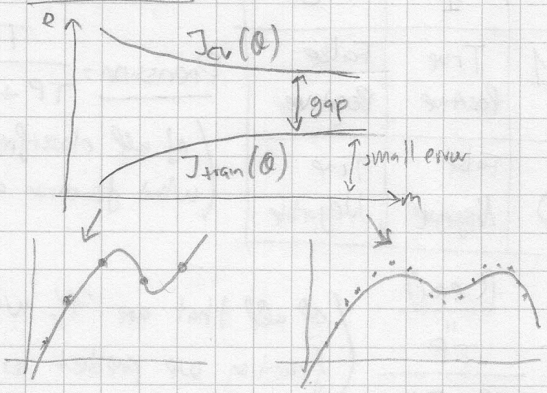
High bias (underfitting)



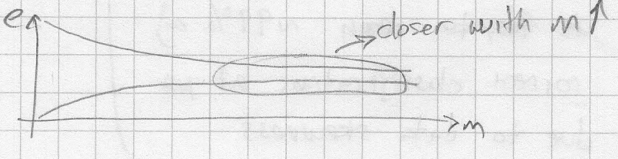
⇒ getting more training data will not help that much.



High variance (overfitting)



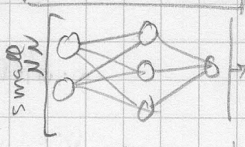
⇒ getting more training data might help



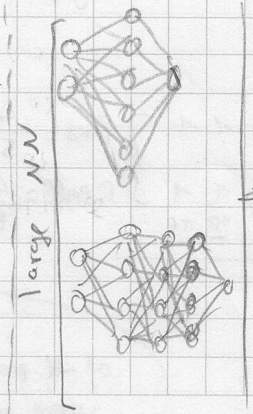
Revisiting 'what to do next'

	fix high variance (overfit)	fix high bias (underfit)
- Get more training data →	✓	
- Smaller set of features →	✓	
- Get additional features →		✓
- Add polynomial features →		✓
- Decrease λ →		✓
- Increase λ →	✓	

Back to NN



computationally cheap
prone to underfitting (high bias)



computationally expensive
prone to overfitting (high variance)
↓
use regularization (λ)

How to choose λ?
- make train/cv/test splits and test with different architectures

ML Projects: Recommended Approach

→ do not spend months on it, in one week you have it.

1. Start with a quick & dirty implementation of a simple algorithm to get an idea of how data and learning behaves (in CV)
2. Plot learning curves to decide if you need more data, more features, etc...
3. Perform an error analysis: manually examine missclassificated examples looking for patterns, trends, ...

It is important to sumarize the performance into a single numerical value (e.g. CV error) so we can easily compare different strategies. For instance, ma spam classifier, compare 'use stemming' vs 'not using stemming', or 'distinguish Capital' vs 'not distinguishing', ... ← try different ideas with the simple implementation to keep improving the algorithm.

Alternative Error Metrics (good for skewed classes)

When we have a classifier (with 2 classes) where in the data there is only 1% of one of the type classes, it is difficult to measure learning accuracy with common error rates, because it is easy to reach ~99% of correct classifications but just due to data skewness.

↓

Even a predictor that always predicts $y=C$ would be a better one...

Use precision & recall metrics

		Actual class 1	Actual class 0
Predicted class	1	True Positive	False Positive
	0	False Negative	True Negative

$$\text{Precision} = \frac{TP}{TP+FP}$$

(of all classified as '1', what fraction actually was '1')

(true positives / predicted positive)

$$\text{Recall} = \frac{TP}{TP+FN}$$

(of all that are '1', what fraction we predict as '1')

(true posit. / actual positive)

We want high precission & high recall

try Dijkstra!

prediction	0	0	0	0	0	1	1	1	1	1
actual	0	0	1	0	1	1	0	1	1	1
	TN	TN	FN	TN	FN	TP	FP	TP	TP	TP

$\text{Prec} = \frac{4}{5} = 0.8$

$\text{Recall} = \frac{4}{6} = 0.66$

Recall → of all that were '1', how many where predicted '1'

actual	1	1	1	1	1	0	0	0	0	0	0
	TP	FN	TP	TP	TP	FN	TN	FP	TN	FP	TN
pred	1	0	1	1	1	0	0	1	0	1	0

Precision ← of all predicted '1', how many were really '1'

→ but usually there is a tradeoff between precision and recall.

For example, in a Log Reg we can set a high threshold:

(A) $y=1$ if $h_\theta(x) \geq 0.8$; $y=0$ if $h_\theta(x) < 0.8$ → we are more confident of the $y=1$ predictions

higher precision, lower recall

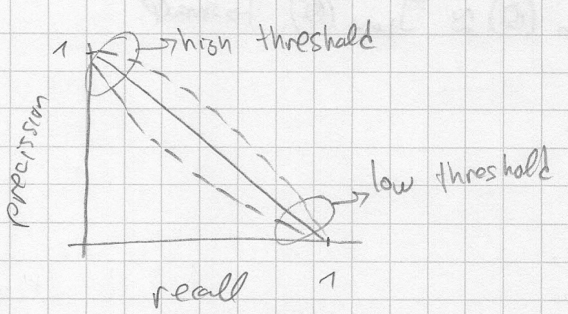
$\left(\frac{TP\downarrow}{\text{predicted Pos}\downarrow\uparrow}\right)$, $\left(\frac{TP\downarrow}{\text{actual posit.}=\downarrow}\right)$

or we can set a low threshold:

(B) $y=1$ if $h_\theta(x) \geq 0.3$; $y=0$ if $h_\theta(x) < 0.3$ → we avoid missing positive classifications ($\downarrow FN$)

lower precision, higher recall

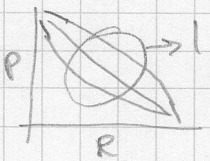
$\left(\frac{TP\uparrow}{\text{predict. Pos}\uparrow\uparrow}\downarrow\right)$ $\left(\frac{TP\uparrow}{\text{actual pos.}=}\uparrow\right)$



The problem with using Precision & Recall (including the tradeoffs) is that we have lost the 'single metric' evaluation that we had with the CV error. How can we transform P&R into a single metric?

~~Average = $\frac{P+R}{2}$~~ → bad metric because 'bad' classifiers with very high R & very low P (or viceversa) end up with medium average...

F_1 score $= 2\frac{PR}{P+R}$ → better measures when P&R are large...



I think that it detects this area

(or F score)

Use this in the CV set to choose among different algorithms.

Large Data Rationale

The key question is when it makes sense to collect a large amount of data. This is an important issue because not in all conditions having more data results in more accuracy.

The rationale says that:

1- First use a learning algorithm with many parameters
   → low bias, prone to high variance
   → $J_{train}(\theta)$ will be small

2- Then reduce the variance with more data
   → low bias, low variance
   → $J_{train}(\theta) \approx J_{test}(\theta)$ ↦ small