ML

Andrew Ng

Coursera

Week 6

Date:

Debugging a learning algorithm:

Suppose you have implemented regularized linear regression to predict housing prices

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

However when you test your hypothesis on a new set of houses, you find that it makes unacceptably large errors in its predictions. What should ~~we~~ you try next?

> Note: There is a difference between someone that really knows how to powerfully & effectively apply that algorithm versus someone that's less familiar with the material

Solutions:

1) Get more training examples
   - ↳ ~~may not~~ may not work out at times
2) Try smaller set of features to avoid over fitting

Date:

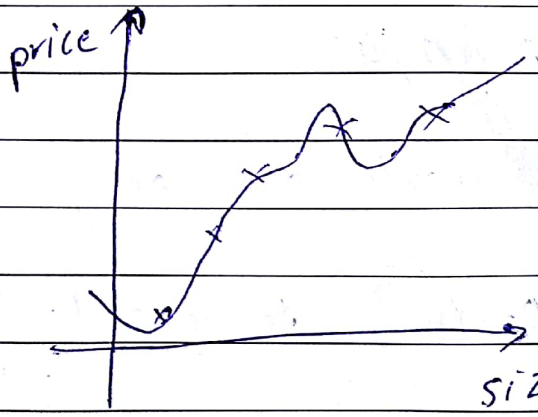
3) Try getting additional features
4) Try adding polynomial features
5) Try decreasing λ
6) Try increasing λ

Machine learning diagnostic

Diagnostic : a test that you can run to gain insight what is / isn't working with a learning algorithm, and gain guidance as to how best to improve its performance

↳ ~~which to~~ choose which avenue from above (out of 6)

Evaluating a hypothesis



Overfitting
Fails to generalize to new examples not in training set

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Date:

How to tell if hypothesis is overfitting with dataset with many features?

Size	Price	
2104	400	Training set (70%)
1600	330	
2400	532	
1416	572	
3000	142	
1985	132	
1534	511	
1427	112	Test set (30%)
1380	345	
1494	345	

Training set: $x^{(1)}, y^{(1)}$
$x^{(2)}, y^{(2)}$
$\vdots$
$x^{(m)}, y^{(m)}$

Test set: $x_{test}^{(1)}, y_{test}^{(1)}$
$x_{test}^{(2)}, y_{test}^{(2)}$
$\vdots$
$x_{test}^{(m_{test})}, y_{test}^{(m_{test})}$

m_{test} = no. of test examples

Date:

The records chosen for 70% & 30% must be random.

~~Testing~~ Training / testing procedure for linear regression

- learn parameter θ from training data (minimizing training error $J(\theta)$)
- 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 procedure for logistic regression

- learn parameter θ from training data
- compute test set error

$$J_{test}(\theta) = \frac{-1}{m_{test}} \sum_{i=1}^{m_{test}} y_{test}^{(i)} \log h_\theta(x_{test}^{(i)}) +$$

$$+ \left(1 - y_{test}^{(i)}\right) \log h_\theta(x_{test}^{(i)})$$

Date:

- Misclassification error (0/1 ~~etc~~ misclassification error):

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

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

↳ ~~fraction of~~ fraction of

$$\frac{\#\text{ of mislabelled predictions}}{\text{total predictions}}$$

Since $J_{train}(\theta)$ will always be low ~~for~~, $J_{test}(\theta)$ can be used to find out if there is any overfitting: when $J_{train}(\theta)$ is low & $J_{test}(\theta)$ is high

Date:

Model Selection & training/validation/test sets

Once parameters $\theta_0, \theta_1, \ldots, \theta_n$ are fit to some set of data (training set), the error of the parameters as measured on that data (the training error $J(\theta)$) is likely to be lower than the actual generalization error

Consider the model selection problem

$d=1$ 1) $h_\theta(x) = \theta_0 + \theta_1 x$

$d=2$ 2) $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$

⋮ ⋮

$d=10$ 10) $h_\theta(x) = \theta_0 + \theta_1 x + \ldots + \theta_{10} x^{10}$

d = degree of polynomial

↳ it seems as if there is an additional parameter d to the algorithm that is being determined by the data set

To choose a degree of polynomial & get estimate of how well hypothesis generalised to new examples:

~~Cne-tt peg~~

First take models & minimise training error

(continued on next page)

Victory

Page No.

Date:

1) $h_\theta(x) = \theta_0 + \theta_1 x \rightarrow \theta^{(1)}$

2) $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \rightarrow \theta^{(2)}$

⋮

3) $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \rightarrow \theta^{(10)}$

theta vectors after minimising training error

$\theta^{(1)} \rightarrow J_{test}(\theta^{(1)})$

$\theta^{(2)} \rightarrow J_{test}(\theta^{(2)})$

⋮ ⋮

$\theta^{(10)} \rightarrow J_{test}(\theta^{(3)})$

Choose model with lowest $J_{test}(\theta^{(i)})$

Assume that $J_{test}(\theta^{(5)})$ has the lowest test set error then choose $\theta_0 + \theta_1 x + \ldots + \theta_5 x^5$

Suppose I want to ask how well does my model generalize?

↳ possible answer: report test set error $J_{test}(\theta^{(5)})$

↳ not a fair estimate of hypothesis's generalization

Date:

Problem: $J_{test}(\theta^{(5)})$ is likely to be an optimistic estimate of generalization error i.e our extra parameter (d = degree of polynomial) is fit to data set

↳ we chose the value of d that gave the best possible performance on the test set

↳ hypothesis is likely to do better on test set than on new examples it hasn't seen before

- Parameter d is fit to the test set

To address this problem:
Given data set split into 3 pieces:
1) training set (60%)
2) cross validation set (CV) (20%)
3) test set (20%)

Date:

Dataset:

Size	Price	
2104	400	Training set (60%)
1600	330	
2400	369	
1416	232	
3000	540	
1985	300	
1534	315	CV (20%)
1427	199	
1380	212	test set (20%)
1494	243	

$(x^{(1)}, y^{(1)})$
$(x^{(2)}, y^{(2)})$
$\vdots$
$(x^{(m)}, y^{(m)})$

$(x_{cv}^{(1)}, y_{cv}^{(1)})$
$(x_{cv}^{(2)}, y_{cv}^{(2)})$
$\vdots$
$(x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$

$(x_{test}^{(1)}, y_{test}^{(1)})$
$(x_{test}^{(2)}, y_{test}^{(2)})$
$\vdots$
$(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

m_{cv} = no. of CV examples

Thus the cost/error can be calculated for training, cross validation & test data set: $J_{train}(\theta)$, $J_{cost}(\theta)$, $J_{test}(\theta)$

Date:

Model Selection

$\min_{\theta} J(\theta) \rightarrow \theta^{(1)}$

1) $h_\theta(x) = \theta_0 + \theta_1 x$
2) $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \rightarrow \theta^{(2)}$
⋮
10) $h_\theta(x) = \theta_0 + \theta_1 x + \ldots + \theta_{10} x^{10} \rightarrow \theta^{(10)}$

~~Use cross validation~~ Now instedd of using ~~test~~ training set to select the model use the validation set to select model

$\theta^{(1)} \rightarrow J_{cv}(\theta^{(1)})$
$\theta^{(2)} \rightarrow J_{cv}(\theta^{(2)})$
$\theta^{(3)} \rightarrow J_{cv}(\theta^{(3)})$
⋮
$\theta^{(10)} \rightarrow J_{cv}(\theta^{(10)})$

Pick the ~~lowest~~ hypothesis with lowest cross validation error. Suppose for this example $h_\theta(x) = \theta_0 + \ldots \theta_4 x^4$ is chosen hence 4th order polynomial model is chosen

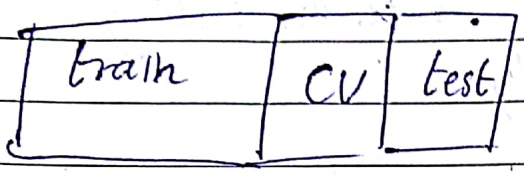
↳ $d = 4$

Date:

Use test set to estimate the generalization error of the model selected

Note: CV can't be used to report generalised error as it was used to determine d

Data set



Train algo on train data - determine θ on different d → Use CV to determine value of d on set θ (find error using θ)

Use test to determine generalised error on set d & set θ (find error using θ & d)

Using test set to get d & then as a way of finding generalised error is wrong

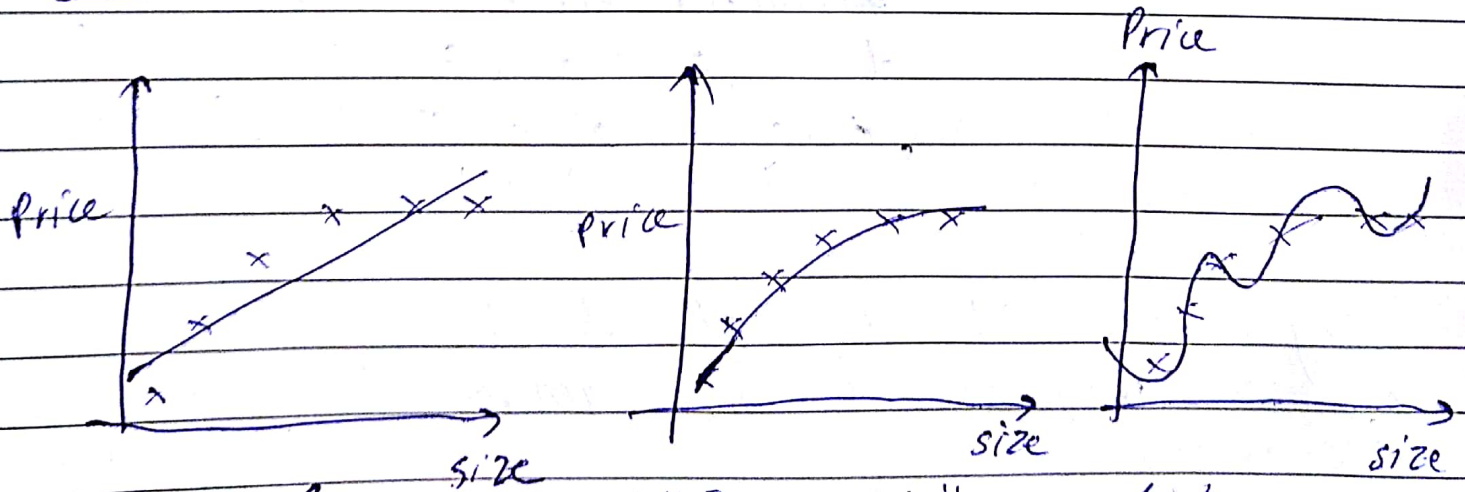
Diagnosing Bias Vs Variance

If a learning algorithm doesn't do well then almost all the time the model has a high bias problem or high variance problem → underfitting or overfitting respectively.

High bias = underfitting high variance = overfitting

Finding out which one you have will give useful ways of improving algorithm

Bias / Variance



High bias (underfit)
$d = 1$

"Just right"
$d = 2$

High variance (overfit)
$d = 4$

Victory

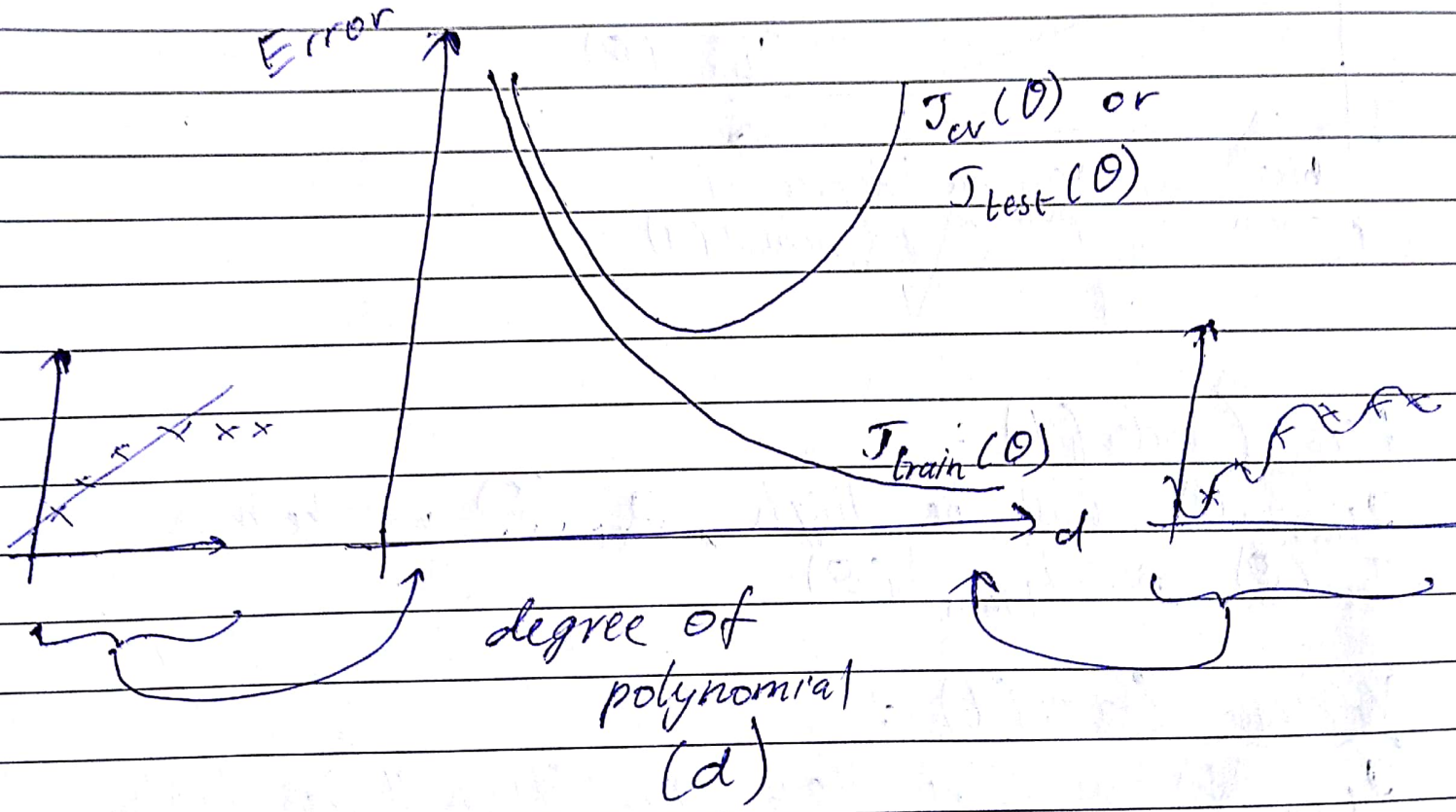
Date:

Training error: $J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left[h_\theta(x^{(i)}) - y^{(i)} \right]^2$

cross validation error: $J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} \left(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)} \right)^2$

(or $J_{test}(\theta)$)

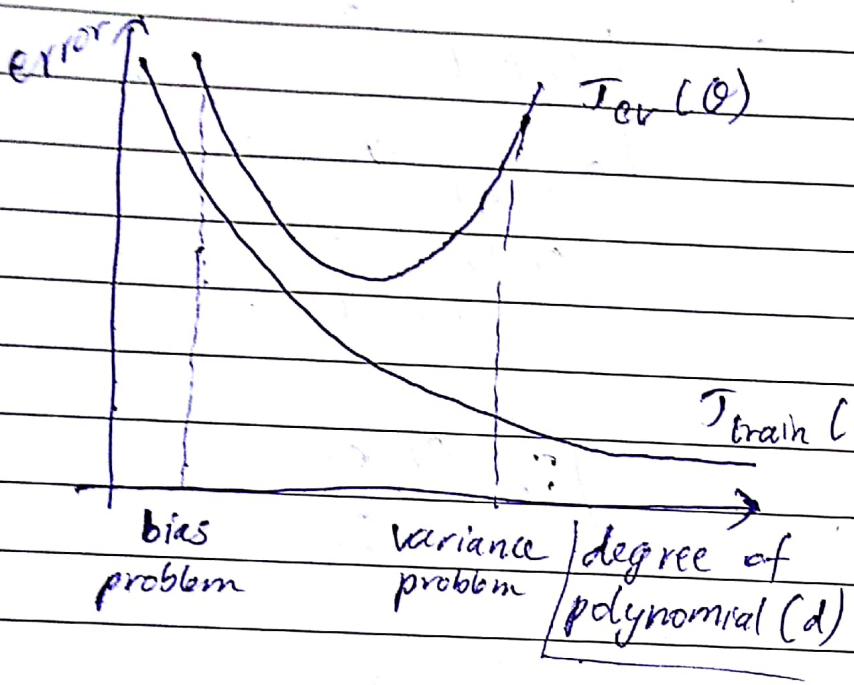
↳ $J_{test}(\theta)$ will output similar results



Date

Diagnosing bias vs Variance

Suppose your learning algorithm is performing less well than you were hoping ($J_{cv}(\theta)$ or $J_{test}(\theta)$ is high). Is it a bias or variance problem?



Bias (underfit):

$J_{train}(\theta)$ will be high, $J_{cv}(\theta)$ will be high

$J_{cv}(\theta) \approx J_{train}(\theta)$

Variance (overfit):

$J_{train}(\theta)$ will be low, $J_{cv}(\theta)$ will be high

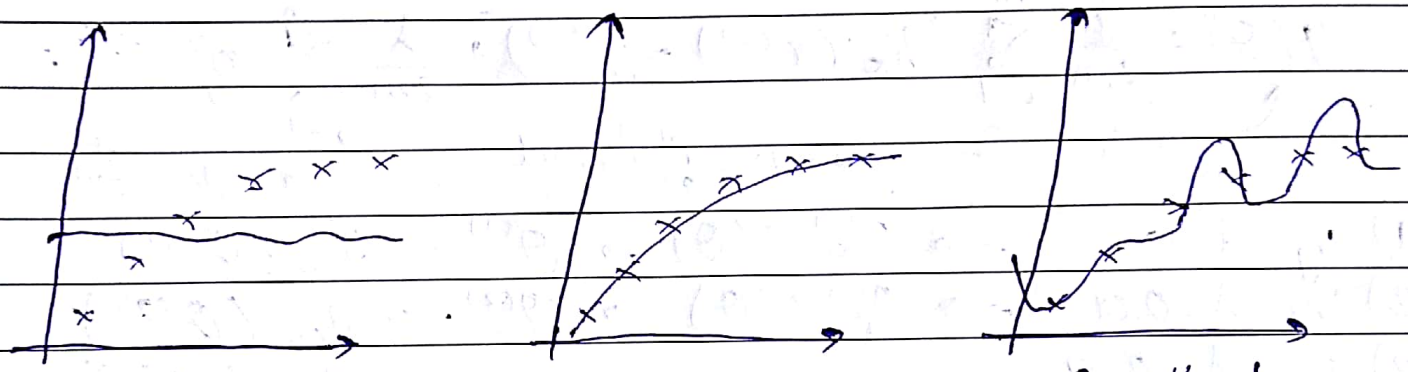
$J_{cv}(\theta) >> J_{train}(\theta)$

Date:

Regularization & Bias/Variance

Model: $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m} \sum_{j=1}^{m} \theta_j^2$$



Large λ
High bias (underfit)
$\lambda = 1000$, $\theta_1 \approx 0$, $\theta_2 \approx 0$...
$h_\theta(x) \approx \theta_0$

Intermediate λ
"Just right"

Small λ
High variance
(overfit)
$\lambda \approx 0$

How is this intermediate λ chosen?

To choose the regularization parameter λ ~~use~~ define the following:

1) $J_{train}(\theta)$
2) $J_{cv}(\theta)$
3) $J_{test}(\theta)$

} all average sum of squared error without regularization

Date:

Average sum of squared error:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Choosing the regularization parameter

Model: $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

↑ reg. applied on train set

↑ reg. not applied

1) Try $\lambda = 0$ → $\min_\theta J(\theta)$ → $\theta^{(1)}$ → $J_{cv}(\theta^{(1)})$
2) Try $\lambda = 0.01$ → $\min_\theta J(\theta)$ → $\theta^{(2)}$ → $J_{cv}(\theta^{(2)})$
3) Try $\lambda = 0.02$ → $J_{cv}(\theta^{(3)})$
4) Try $\lambda = 0.04$ → $J_{cv}(\theta^{(4)})$
5) Try $\lambda = 0.08$ → $J_{cv}(\theta^{(5)})$

⋮

12) Try $\lambda = 10.24$ → $J_{cv}(\theta^{(12)})$

Pick model with the lowest $J_{cv}(\theta)$ & use that value of λ. Suppose model 5 is picked

Use test set to find generalized error of the model chosen on previously unseen examples

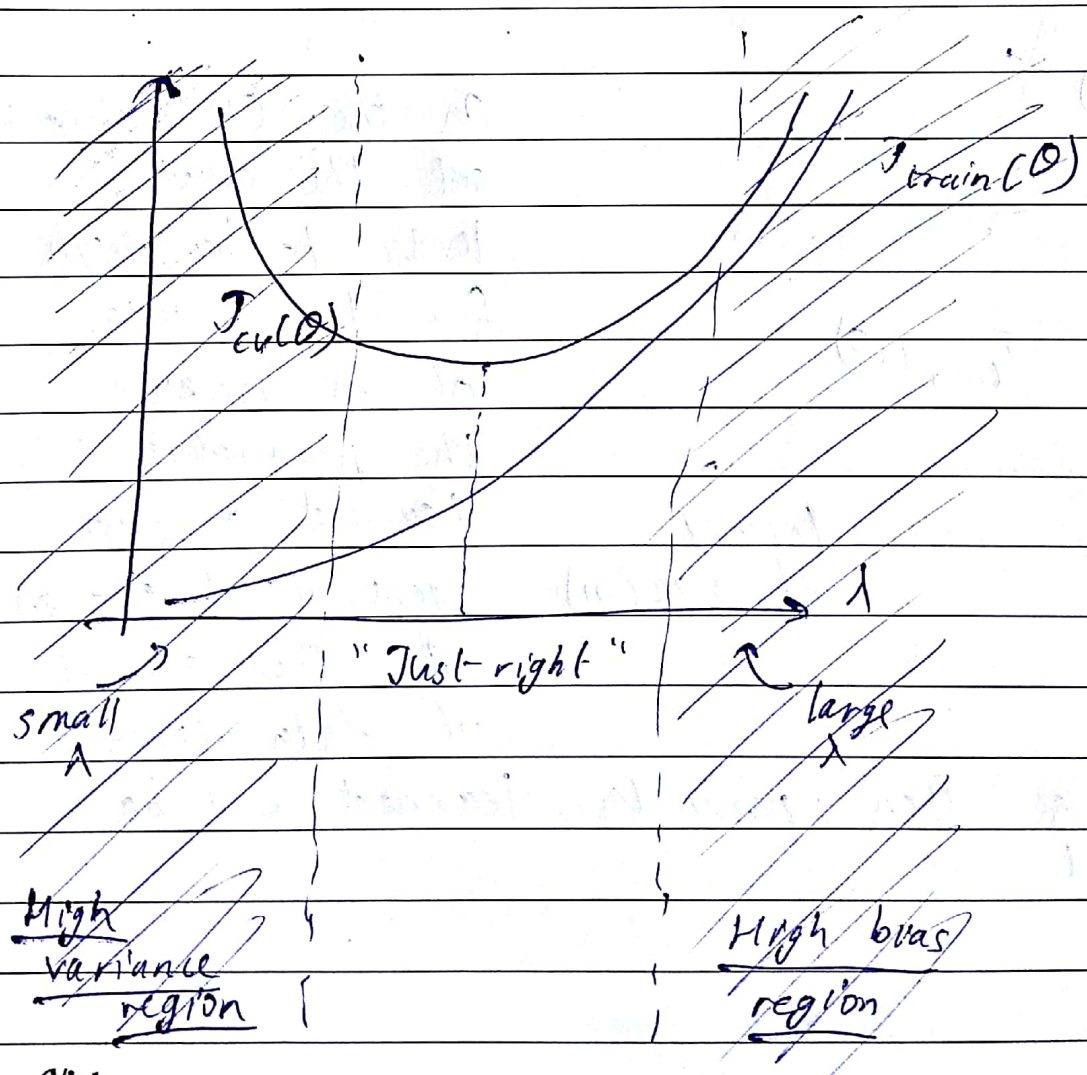
Date:

① $J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 + \frac{\lambda}{2m} \sum_{j=1}^{m} \theta_j^2$

② $J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$

③ $J_{cv}(\theta) = \frac{1}{2m} \sum_{i=1}^{m_{cv}} \left(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)} \right)^2$

Find ① first for different values of λ & train a specific $\theta^{(i)}$ then run ② & ③ with the specific $\theta^{(i)}$



Victory

Page No.

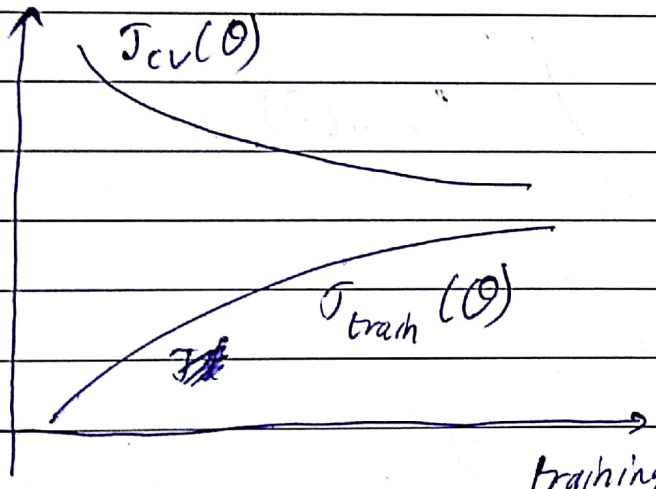
Date:

Learning Curves

- used as sanity check that algorithm is working properly
- or want to improve performance of algorithm

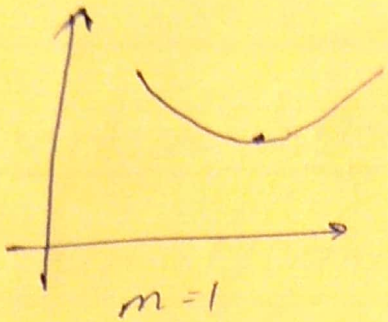
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

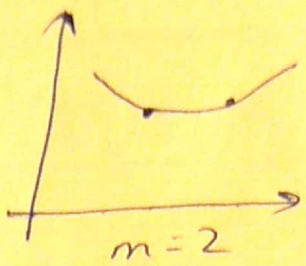
$$J_{cv}(\theta) = \frac{1}{2m} \sum_{i=1}^{m_{cv}} \left(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)}\right)^2$$

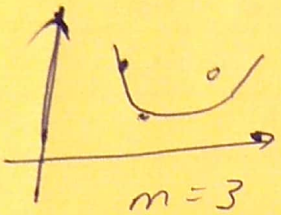


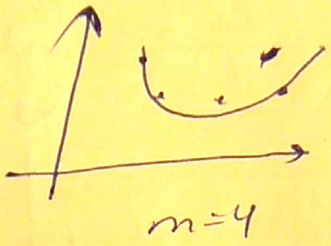
On the CV dataset the error tends to be high for low values of m because the parameters learned are not generalised enough. As the amount of data increase i.e. m increases then parameters learned will be well generalised

$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$







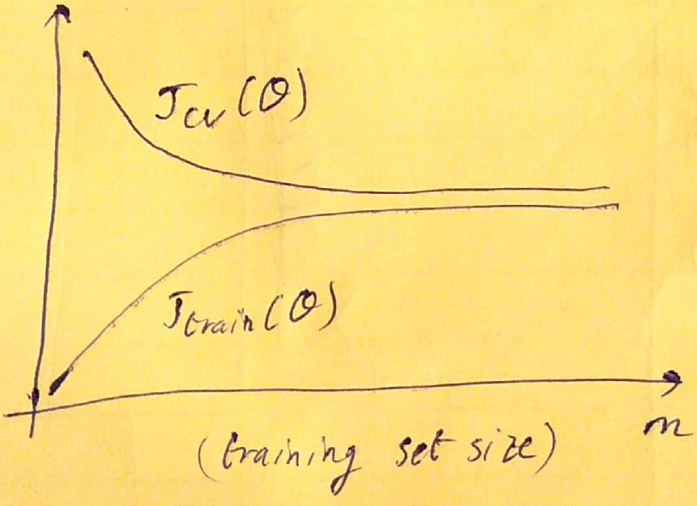


As the number of examples are low then fitting the data has ~~ver~~ low cost Hence for small values of m, $J_{train}(\theta)$ is low.

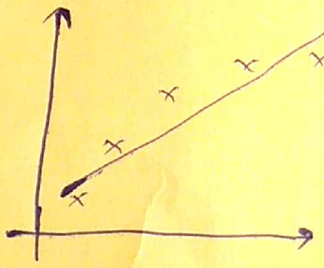
As ~~the~~ m increa

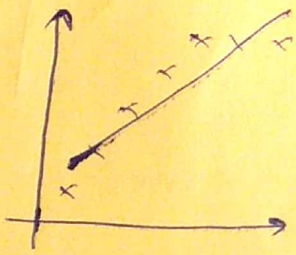
then fitting the dataset such that cost is low becomes difficult, thus $J_{train}(\theta)$ increase

High Bias



$h_\theta(x) = \theta_0 + \theta_1 x$





The shape of the fitted curve does not change much as more & more data points are added. Thus both J_{cv} & J_{train} plateau eventually.

Date:

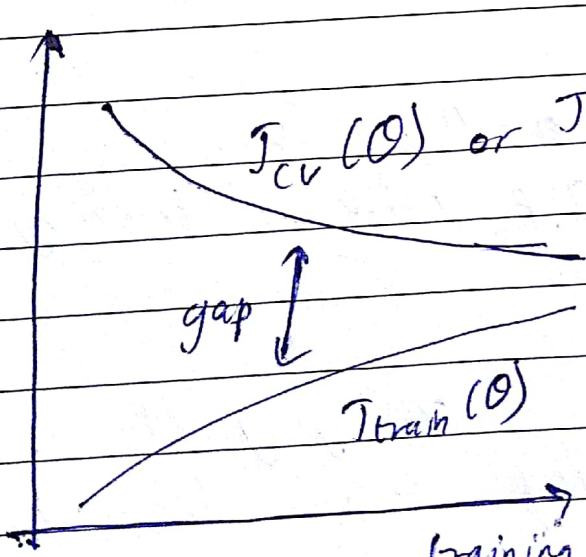
Initally J_{cv} decreases as the parameters ~~tend to~~ learned tend to work better on generalised data but it ~~plateaus~~ plateau's eventually.
As before the training error will be small initially & increase as more examples are added. It plateau's as well.
~~J_{test}~~ $J_{train} \approx J_{cv}$ as they plateau.

If a learning algorithm has high bias then getting more training data will not help → you will end up with similar values of J_{train} & J_{cv}

diagnostic

Both J_{cv} & J_{train} will be high in value

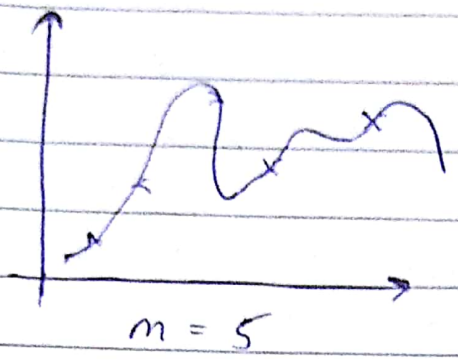
High Variance



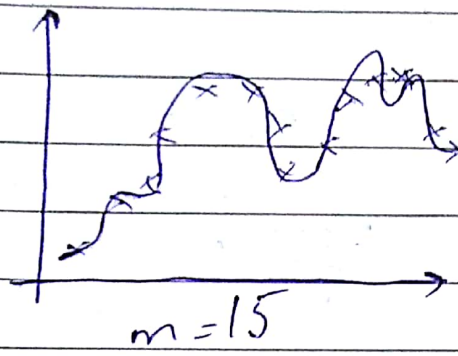
If a learning algorithm is suffering from high variance, getting training data is likely to help.

Page No.

$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_{100} x^{100}$ Date:



When m is small then $h_\theta(x)$ (in this case) can fit it well. As the data set increases (m increases) then fitting the data set becomes more difficult, $J_{train}(\theta)$ increases then



On small values of m we ~~wh~~ know that the model is being overfit hence the model will not generalize well to unseen data therefore $J_{cv}(\theta)$ will be high. As more data is added to training, the less the degree of overfitting ~~the~~ hence $J_{cv}(\theta)$ decreases.

Indicative diagnostic of a high variance problem is the large gap between training & cross validation.

Both J_{cv} & J_{train} are converging to each other hence adding more data helps.

Date:

The curves illustrated here are idealized, in reality you will encounter curves with more noise & messiness

Large training set size - $J_{train}(\theta)$ increases with training set size & $J_{cv}(\theta)$ continues to decrease without leveling off. Also, $J_{train}(\theta) < J_{cv}(\theta)$ but the difference between them remains significant.

Deciding What to do Next

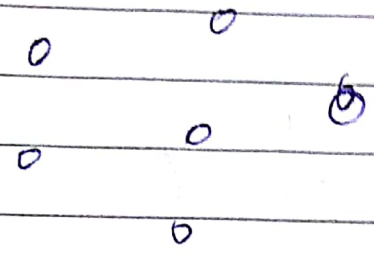
Debugging a learning algorithm:
Suppose you have implemented regularized linear regression to predict housing prices. However, when you test your hypothesis in a new set of houses, you find that it makes unacceptably large errors in its prediction. What should you try next?

- Get more training examples → fixes high variance
- Try smaller sets of features → fixes high variance
- Try getting additional features → fixes high bias
- Try adding polynomial features → fixes high bias
- Try decreasing λ → fixes high bias
- Try increasing λ → fixes high variance

Date:

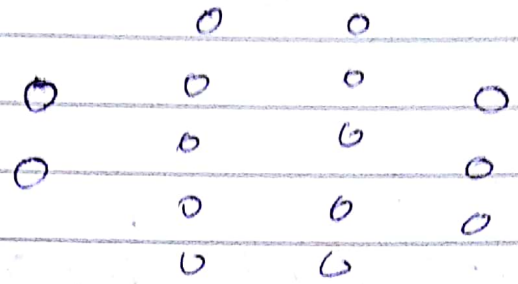
Neural Networks & overfitting

"Small" neural networks
(few nodes per layer & few hidden layers → fewer parameters) prone to underfitting



Computationally cheaper

"large" neural networks
(more parameters)
prone to overfitting



Computationally more expensive

Use regularization (λ) to address overfitting

Usually it's best to use a "large" neural network with regularization as opposed to using "small" ~~neural~~ neural networks.

Date:

If you want to find how many hidden layers would work best then the best thing would be to split the data into train, CV & test. Check ~~what~~ what amount of layers deliver the best results on cross validation data set as it was used to determine the degree of linear regression (d)

Prioriting what to work on

Suppose you want to build a spam ~~classifier~~ classifier on emails

Building a spam classifier

Supervised learning : x = feature of email, y = spam (1) or not spam (0)
Features x : Choose 100 words indicative of spam / not spam

↳ Eg: deal, buy, discount, Andrew, etc.

Create a feature vector from the email depending on which words are present: 0 for not present, 1 for present (regardless of # of times a word is mentioned)

Date:

Note: In practice, take most frequently occuring n words (10'000 to 50'000) in training set, rather than manually picking 100 words

How to spend time to make it have low error (spam detector)?

- Collect ~~test~~ lots of data
- develop sophisticated features based on email routing information (from email header)
- develop sophisticated features for message body e.g. should "discount" & "discounts" be treated as the same word? How about "deal" & "dealer"? features about pronunciation?
- develop sophisticated algorithm to detect misspellings ~~(e.g. m0rtgage)~~

A research group will fixate on one of these problems & try to increase the accuracy of the model this way

When faced with an ML problem there are usually many different ways to improve the algorithm

Date:

Error Analysis

↳ gives a more systematic way of choosing which way to improve algorithm out of the many different options available as shown previously

Recommended approach

- Start with a simple algorithm that you can implement quickly. Implement it & test it on your cross-validation data. (max take up to 1 day - 24 hours - on implementation for this)
- Plot learning curves to decide if more data, more features etc are likely to help. In the absence of the quick implementation & learning curves, it is very difficult to tell what to do next.
- Error analysis: Manually examine the examples (in cross validation set) that your algorithm made errors on. See if you spot any systematic trend in what type of examples it is making errors on.

↳ this process will inspire you to design new features or tell of the current shortcomings of the system.

error analysis evaluated on CV so that when new features are made/decided we can check generalised error as new features may be CV data set dependent.

Date:

Suppose we have a spam classification ML problem:

m_{cv} = 500 examples in cross validation set
Algorithm misclassifies 100 emails.
Manually examine the 100 errors, and categorize them based on:

(i) What type of email it is
(ii) What cues (features) you think would have helped the algorithm classify them correctly

→ spam from pharma companies, watch replica companies, phishing attempts & etc

Count according to each class misclassified:
Pharma: 12
Replica: 4
phishing: 53
Other: 31

→ It would be the best use of your time to ensure that spam classifier correctly classifies phishing attempts, accuracy will increase greatly.

→ Create classes as such:
- deliberate misspellings: 5
- unusual email routing: 16
- unusual (spamming) punctuation: 32

Date:

It seems that most spammers use spamming punctuation when they write their emails hence it's best to create sofisticated features to address this. misclassification.

Error analysis - process of manually examining the mistakes of the algorithm

↳ supports quick & dirty implementation of algorithm - we want to figure out what are the most difficult examples to classify so effort can be focused on these

The importance of numerical evaluation

Should discount / discounts / discounted / discounting be treated as the same word? Can use "stemming" software (E.g. "Porter stemmer")

↳ construct of NLP

"Porter stemmer" - tool that can be used to achieve the feature above.

Date:

Stemming software basically looks at the first few alphabets of the words & groups words together. This can lead to disadvantages:

- universe / university will be taken as same word when semantically they mean different things.
  - ↳ Thus not easy to tell if stemming software should be used

Error analysis may not be helpful for deciding if this is likely to improve performance. Only solution is to try it & see if it works.

↳ Need numerical evaluation (e.g. cross validation error) of algorithm's performance with & without stemming:

without stemming: 5% error
with stemming: 3% error
} Thus using stemming is a good idea

Gives you a very quick way of deciding to use stemming or not

The quick & dirty approach for creating algo allows you to use evidence to determine which direction to apply your time rather than using gut feeling

Error Metrics for Skewed Classes

~~Error metrics = numerical evaluation~~
numerical evaluation - error metrics

Cancer classification example
Train logistic regression model $h_\theta(x)$
($y = 1$ if cancer, $y = 0$ otherwise)
You find that you got 1% error on test set (99% correct diagnoses)

However only 0.50% patients have cancer

Suppose you use a non-learning function:
function y = predict Cancer (x)
    y = 0; % ignore x!
return

This function now has 0.50% error, better than the learning algo applied!

Skewed classes - ratio of positive to negative examples is at extremes

of +ve examples is much smaller than -ve in this example

Date:

If skewed classes are present in the data set it becomes hard to use simple classification accuracy:

Version 1: 99.2% (0.08% error)
Version 2: 99.5% (0.05% error)

Technically according to simple classification Version 2 is an improvement over version 1. But Version 2 can just simply be setting $y = 0$ for all x. as previously shown.

↳ difficult to say if there is any actual improvement with the learning classifier

Other error metrics are used for skewed classes

Date:

Precision / Recall

$y = 1$ in presence of rare class that we want to detect

~~Actual class~~ Actual class

~~Predicted class~~

Predicted class	Actual class: 1	Actual class: 0
1	True positive	false positive
0	false negative	True negative

false positive – prediction is 1 but it is wrong

false negative – prediction is 0 but it is wrong

Date:

Precision (of all patients where we predicted $y=1$, what fraction actually has cancer?):

$$\frac{\text{True positives}}{\#\text{ predicted +ve}} = \frac{\text{True +ve}}{\text{True +ve + false +ve}}$$

Recall (of all patients that actually have cancer, what fraction did we correctly detect as having cancer?):

$$\frac{\text{True +ve}}{\#\text{ actual +ve}} = \frac{\text{True +ve}}{\text{True +ve + false -ve}}$$

The higher the value of both, the better

If our algo predicts 0 all the time ($y=0$) then recall will be 0

↳ there will be no true +ves

If precision & recall are high then we know that even if we have skewed classes, the learning algorithm is doing well.

Date:

Trading off Precision & Recall

We want to control the trade-off between precision & recall

We continue with "cancer example as before.
Here logistic regression : $0 \leq h_\theta(x) \leq 1$
Predict 1 if $h_\theta(x) \geq 0.7$
Predict 0 if $h_\theta(x) < 0.7$

↳ we predict $y = 1$ (cancer) only if we are very confident

↳ higher precision now because there are more true +ves as compared to false +ves → predictions are only made when we are more confident

↳ lower recall now because we're predicting $y = 1$ on a small number of people who actually have cancer

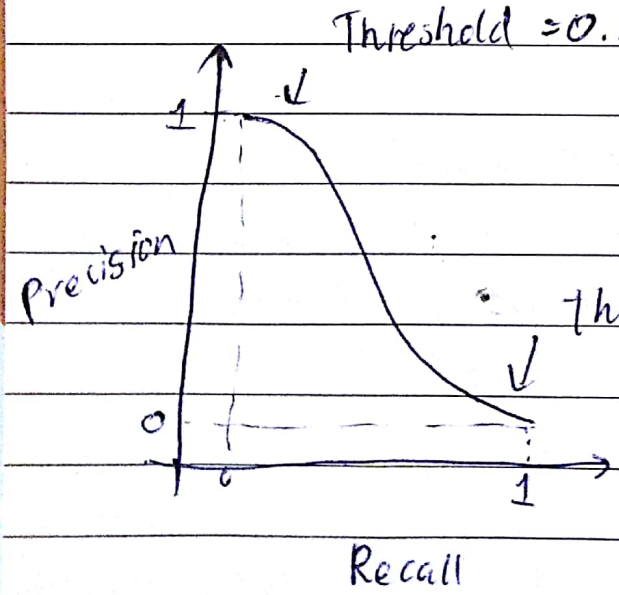
Suppose we want to avoid missing too many causes of cancer (avoid false negatives)

↳ when in doubt we want to predict that they have cancer so that at least they look further into it

Set the probability ~~threes~~ threshold lower:
Predict 1 if $h_\theta(x) \geq 0.3$
Predict 0 if $h_\theta(x) < 0.3$

↳ higher recall & lower precision

More generally: Predict 1 if $h_\theta(x) \geq$ threshold



Date:

F_1 score (F score)

How to compare precision/recall numbers? Which one is the best?

By use precision/recall we lost. The simplicity of a single real number metric → it is difficult to decide which algorithm with which threshold is better

↳ slows down the process of decision making ~~is~~ when evaluating different changes

	Precision	Recall	Avg	F_1 score
Algorithm 1	0.5	0.4	0.45	0.444
Algo 2	0.7	0.1	0.4	0.175
Algo 3	0.02	1.0	0.51	0.0392

It can ~~else~~ be clearly said that algo 3 is the worst as Recall can be 1 if we predict $y = 1$ all the time. Thus there will be no such instances where we predicted 0 hence there will be no false -ves. Hence Recall = 1.0

↳ the 2 extremes of a high threshold or a low threshold tend to be bad classifiers

Victory

Page No.

Date:

Now to make a single value metric from P & R we can try a normal average & F_1 score:

$$\text{Average} = \frac{P+R}{2}$$

$$F_1 \text{ score} = 2\frac{RP}{P+R}$$

With the average, algo 3 is considered the best which is obviously wrong & hence average is a bad measure. However the F_1 score correctly determines Algo 3 the worst.

↳ F score is widely used in the ML community

F score extremes:
- P = 0, R = 0 then F score = 0
- P = 1, R = 1 then F score = 1

There are many different ways of combining precision & recall, F score is just one way

F score $\in [0, 1]$

if enough data is given to an inferior algorithm, it can beat a superior algorithm

Date:

Try out various thresholds on cross validation set & pick whichever gives the greatest f score.

Data for ML

There have been many studies that show that many different learning algorithms tend to give similar ranges of performance ~~but performance is driven by~~ given a ^ training data to of

"It's not who has the best algorithm that wins. It's who has the most data"

This statement is true under certain conditions

Large data rationale

Assume feature $x \in \mathbb{R}^{n+1}$ has sufficient information to predict y accurately

Example: For breakfast I ate two eggs

choose either two, too or too into blank space

Date:

Counter example : predict housing price from only size ($feet^2$) & no other features

Useful test : given the input x, can a human expert confidently predict y?

↳ so a human expert will be able to fill in the black space in the example (a normal english speaker) while a human house expert will not be able to give an estimate price for the house with just the area.

Use a ~~large~~ learning algorithm with many parameters

↳ thus becomes low bias algorithm

hence $J_{train}(\theta)$ will be small

Use a ~~very~~ very large training set (unlikely to overfit) hence it has low variance thus:

$$J_{train}(\theta) \approx J_{test}(\theta)$$

~~so~~ hence $J_{test}(\theta)$ will be small as well

Victory

Page No.

Date:

~~Cont~~

Condition: can a human expert look at the features x & confidently predict y