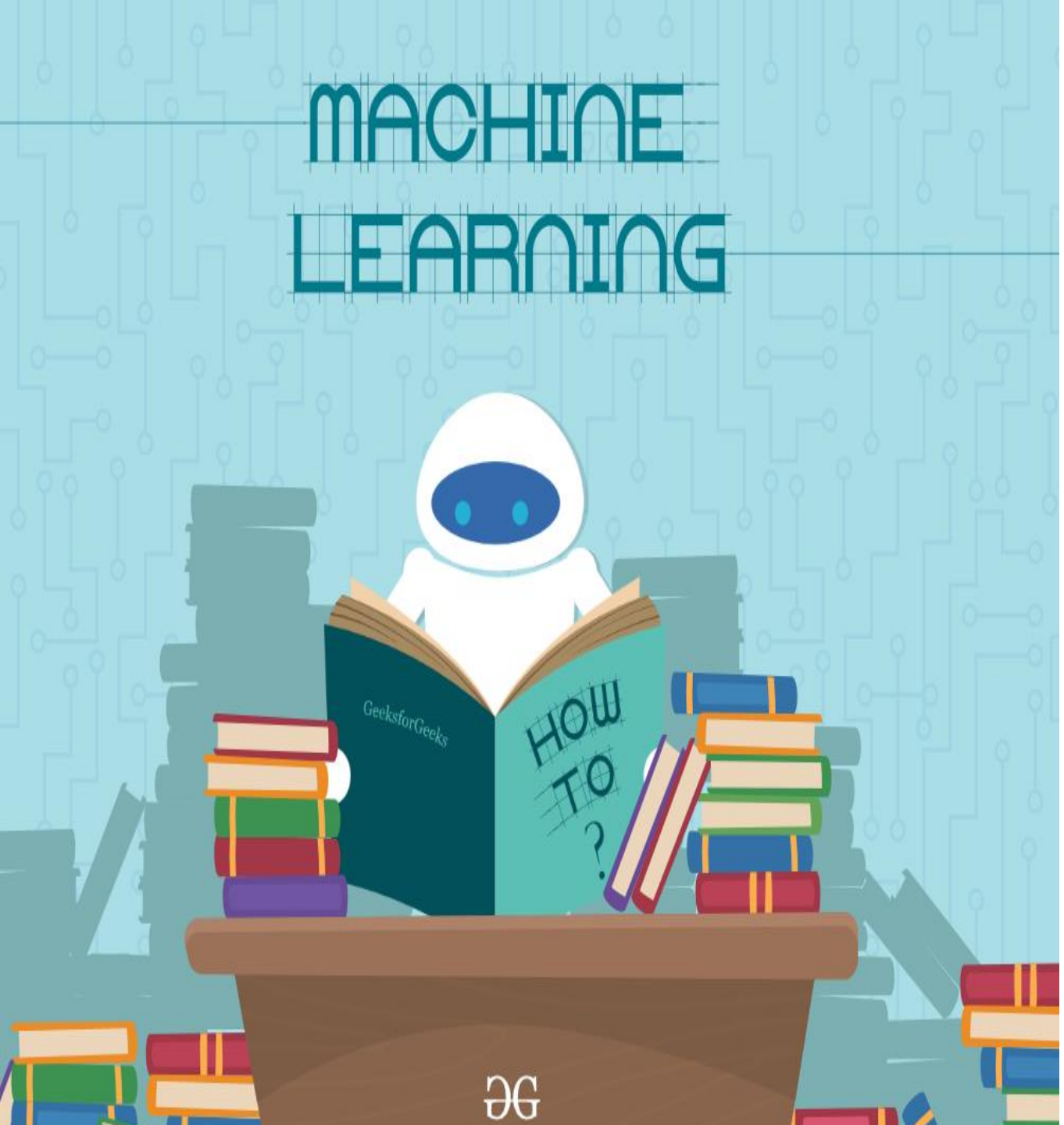


>TCC
THE CODERS' CLUB

Advice for Applying Machine Learning

WEEK 6

Deciding what to try next

Debugging a learning algorithm:

Suppose you have implemented regularized linear regression to predict housing prices.

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

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 you try next?

- Get more training examples
- Try smaller set of features
- Try getting additional features
- Try adding polynomial features (x_1^2, x_2^2, x_1x_2, etc.)
- Try decreasing λ
- 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.

Diagnostics can take time to implement but doing so can be a very good use of your time.

Which of the following statements about diagnostics are true? Check all that apply.

- It’s hard to tell what will work to improve a learning algorithm, so the best approach is to go with gut feeling and just see what works.

- Diagnostics can give guidance as to what might be more fruitful things to try to improve a learning algorithm.

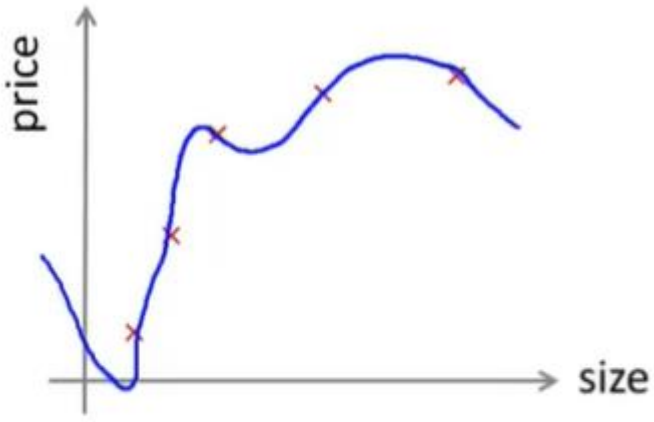
- Diagnostics can be time-consuming to implement and try, but they can still be a very good use of your time.

- A diagnostic can sometimes rule out certain courses of action (changes to your learning algorithm) as being unlikely to improve its performance significantly.

Evaluating a Hypothesis

Once we have done some trouble shooting for errors in our predictions, we can move on to evaluate our new hypothesis.

Consider an example as shown below:



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

This hypothesis fits well on the training example but fails to generalize to new examples not in training set.

x_1 = size of house
x_2 = no. of bedrooms
x_3 = no. of floors
x_4 = age of house
x_5 = average income in neighborhood
x_6 = kitchen size
.
.
x_{100}

The standard way to evaluate hypothesis is as follows:

Dataset:

Size	Price
2104	400
1600	330
2400	369
1416	232
3000	540
1985	300
1534	315
1427	199
1380	212
1494	243

70%

30%

Training Set →

Test Set →

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

m = no. of training examples

$(x_{test}^{(1)}, y_{test}^{(1)})$
$(x_{test}^{(2)}, y_{test}^{(2)})$
.
.
.
$(x_{test}^{(mtest)}, y_{test}^{(mtest)})$

m_{test} = no. of test examples

We split up the data into two sets: 70% **training set** and 30% **test set.**

Training/testing procedure for linear regression:

1. Learn parameter θ and minimize $J_{train}(\theta)$ using the training set.
2. Compute the test set error $J_{test}(\theta)$

$$J_{test}(\Theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\Theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

Training/testing procedure for logistic regression:

1. Learn parameter θ and minimize $J_{train}(\theta)$ using the training set.
2. Compute the test set error $J_{test}(\theta)$

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

Misclassification error (0/1 misclassification error):

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

This gives us a binary 0 or 1 error result based on a misclassification.

The average test error for the test set is given by,

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

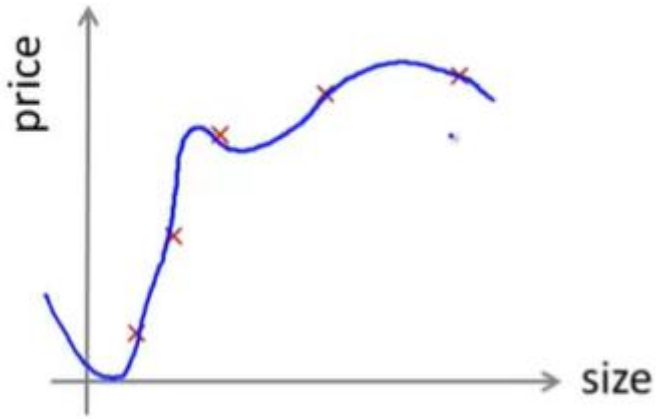
This gives us the proportion of the test data that was misclassified.

Suppose an implementation of linear regression (without regularization) is badly overfitting the training set. In this case, we would expect:

- The training error $J(\theta)$ to be **low** and the test error $J_{test}(\theta)$ to be **high.**
- The training error $J(\theta)$ to be **low** and the test error $J_{test}(\theta)$ to be **low.**
- The training error $J(\theta)$ to be **high** and the test error $J_{test}(\theta)$ to be **low.**
- The training error $J(\theta)$ to be **high** and the test error $J_{test}(\theta)$ to be **high.**

Model Selection and Train/Validation/Test Sets

Overfitting example:



Once parameters $\theta_0, \theta_1, \ldots, \theta_4$ were 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.

- Just because a learning algorithm fits a training set well, that does not mean it is a good hypothesis.
- It could over fit and as a result your predictions on the test set would be poor.
- The error of your hypothesis as measured on the data set with which you trained the parameters will be lower than the error on any other data set.

Model Selection:

1. $h_\Theta(x) = \theta_0 + \theta_1 x$ d=1
2. $h_\Theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$ d=2
3. $h_\Theta(x) = \theta_0 + \theta_1 x + \ldots + \theta_3 x^3$ d=3

.

.

.

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

d = degree of polynomial

- Given many models with different polynomial degrees, we can use a systematic approach to identify the 'best' function.
- In order to choose the model of your hypothesis, you can test each degree of polynomial and look at the error result.

One way to break down our dataset into the three sets is:

Size	Price
2104	400
1600	330
2400	369
1416	232
3000	540
1985	300
1534	315
1427	199
1380	212
1494	243

60%

20%

20%

Training Set →

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

Cross Validation (CV) Set →

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

Test Set →

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

m = no. of training examples

m_{cv} = no. of cross validation examples

m_{test} = no. of test examples

Train/validation/test error:

Training error:

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

Cross Validation error:

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

Test error:

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

We can now calculate three separate error values for the three different sets using the following method:

1. Optimize the parameters in Θ using the training set for each polynomial degree.
2. Find the polynomial degree d with the least error using the cross validation set.
3. Estimate the generalization error using the test set with $J_{test}(\Theta^{(d)})$ where d = theta from polynomial with lower error.

Example:

1. $h_\Theta(x) = \theta_0 + \theta_1 x$
2. $h_\Theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$
3. $h_\Theta(x) = \theta_0 + \theta_1 x + \ldots + \theta_3 x^3$ ← Suppose it is found that $J_{CV}(\theta)$ is minimum for this model where d=3

.

.

.

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

Hence, we pick $\theta_0 + \theta_1 x + \ldots + \theta_3 x^3$ and estimate generalization error for test set $J_{test}(\theta^{(3)})$

This way, the degree of the polynomial d has not been trained using the test set.

Consider the model selection procedure where we choose the degree of polynomial using a cross validation set. For the final model (with parameters θ), we might generally expect $J_{CV}(\theta)$ to be lower than $J_{test}(\theta)$ because:

- An extra parameter (d, the degree of the polynomial) has been fit to the cross validation set.
- An extra parameter (d, the degree of the polynomial) has been fit to the test set.
- The cross validation set is usually smaller than the test set.
- The cross validation set is usually larger than the test set.

Diagnosing Bias vs. Variance

In this section we examine the relationship between the degree of the polynomial d and the underfitting or overfitting of our hypothesis.

- We need to distinguish whether **bias** or **variance** is the problem contributing to bad predictions.
- High bias is underfitting and high variance is overfitting. Ideally, we need to find a golden mean between these two.

Bias:
Bias is the difference between the average prediction of our model and the correct value which we are trying to predict.

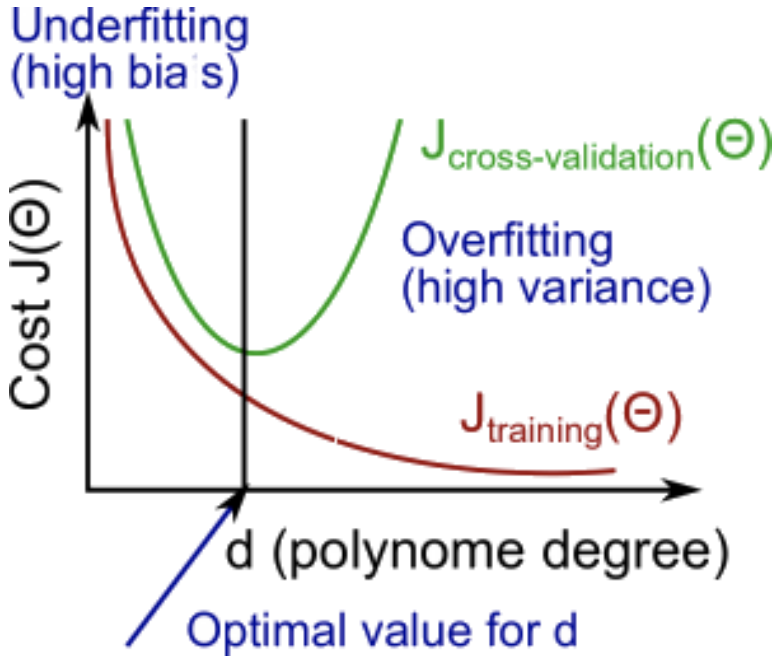
Variance:
Variance is the variability of model prediction for a given data point or a value which tells us spread of our data.

- The training error will tend to **decrease** as we increase the degree d of the polynomial.
- At the same time, the cross validation error will tend to **decrease** as we increase d up to a point, and then it will **increase** as d is increased, forming a convex curve.

High bias (underfitting): Both $J_{train}(\theta)$ and $J_{CV}(\theta)$ will be high and $J_{CV}(\theta) \approx J_{train}(\theta)$

High variance (overfitting): $J_{train}(\theta) << J_{CV}(\theta)$

This is summarized in the figure below:



Suppose you have a classification problem. The (misclassification) error is defined as $\frac{1}{m}\sum_{i=1}^{m} \mathrm{err}(h_\theta(x^{(i)}), y^{(i)})$, and the cross validation (misclassification) error is similarly defined, using cross validation examples $(x_{CV}^{(i)}, y_{CV}^{(i)}), \ldots, (x_{CV}^{(mcv)}, y_{CV}^{(mcv)})$.

Suppose your training error is 0.10, and your cross validation error is 0.30.
What problem is the algorithm most likely to be suffering from?

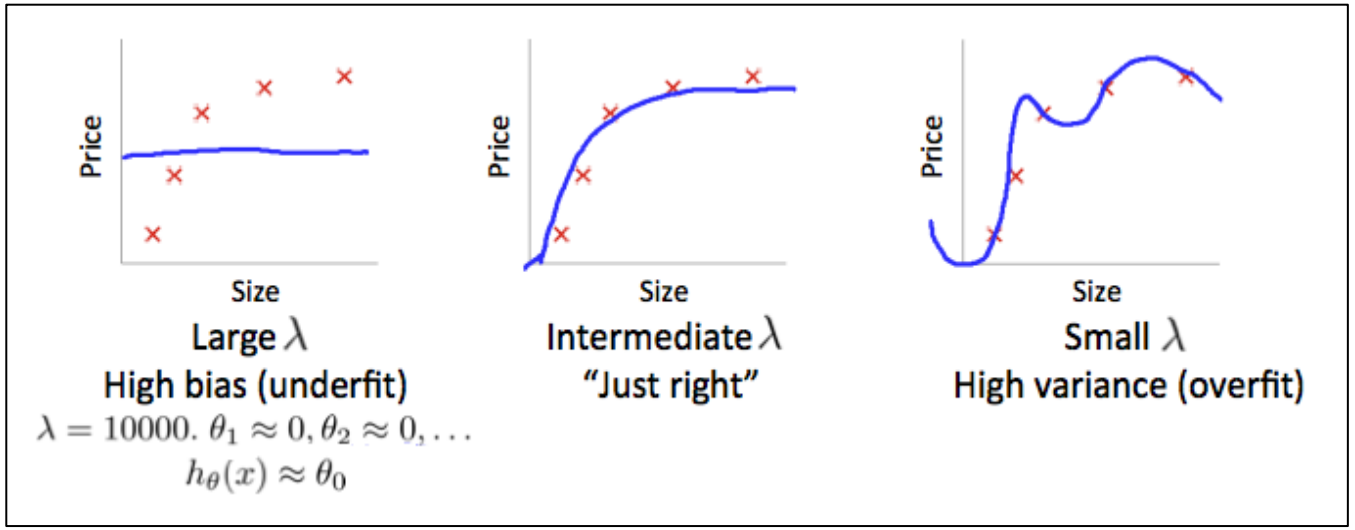
- High bias (overfitting)
- High bias (underfitting)
- High variance (overfitting)
- High variance (underfitting)

Regularization and Bias/Variance

Consider a model as follows:

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

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



- In the figure above, we see that as λ increases, our fit becomes more rigid.
- On the other hand, as λ approaches 0, we tend to over overfit the data.
- So how do we choose our parameter λ to get it 'just right' ?

Choosing the regularization parameter λ:

In order to choose the model and the regularization term λ, we need to:

1. Create a list of lambdas i.e.
   $\lambda \in \{0, 0.01, 0.02, 0.04, 0.08, 0.16, 0.32, 0.64, 1.28, 2.56, 5.12, 10.24\}$

2. Create a set of models with different degrees or any other variants.

3. Iterate through the λs and for each λ go through all the models to learn some Θ.

4. Compute the cross validation error using the learned Θ (computed with λ) on the $J_{CV}(\Theta)$ without regularization or $\lambda = 0$.

5. Select the best combo that produces the lowest error on the cross validation set.

6. Using the best combo Θ and λ, apply it on $J_{test}(\Theta)$ to see if it has a good generalization of the problem.

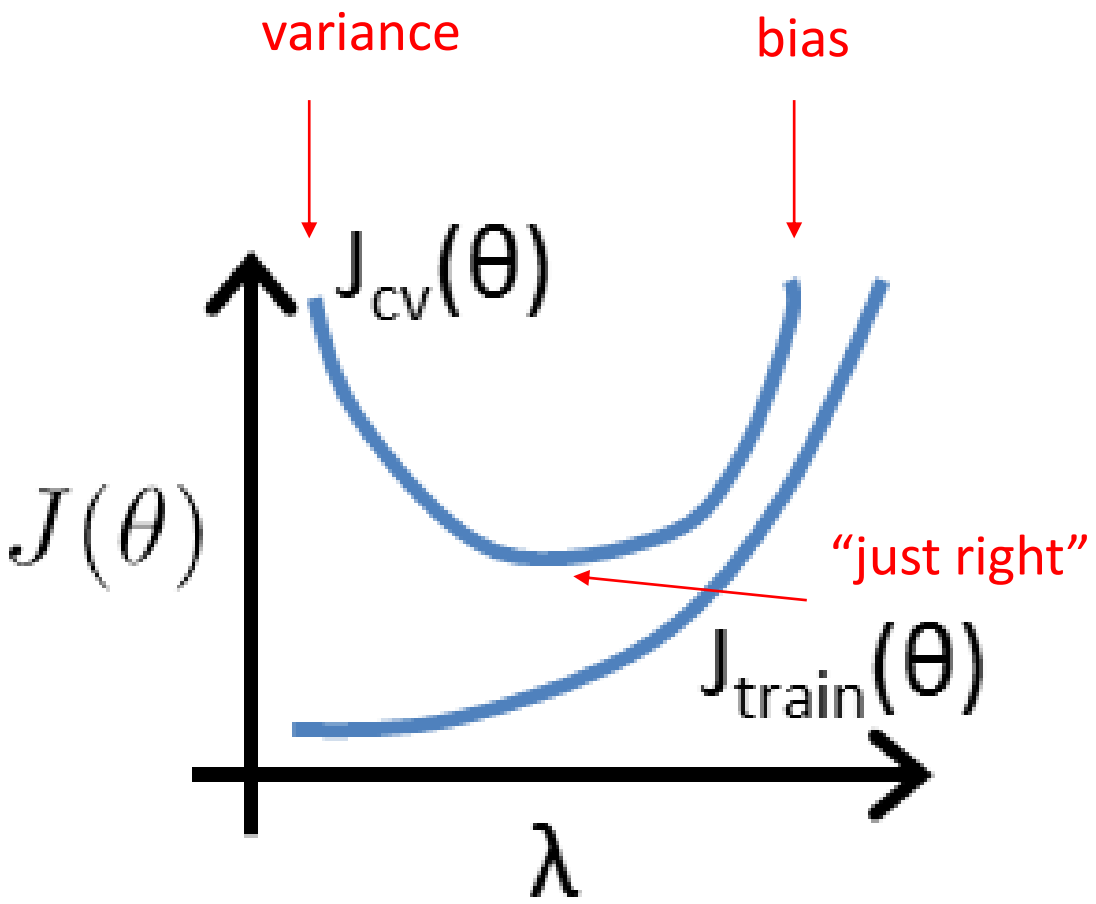
Bias/Variance as a function of regularization parameter λ:

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

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

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

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

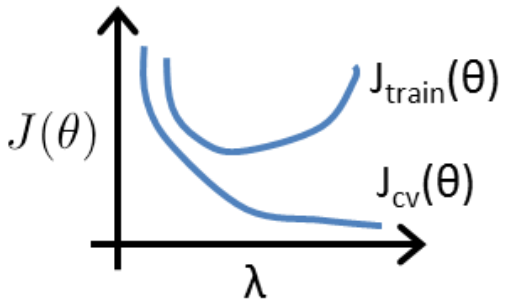


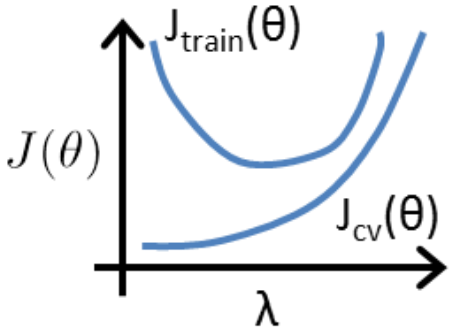
- The training error will tend to **increase** as we increase the regularization parameter λ.
- At the same time, the cross validation error will tend to **decrease** as we increase λ up to a point, and then it will **increase** as λ is increased, forming a convex curve.

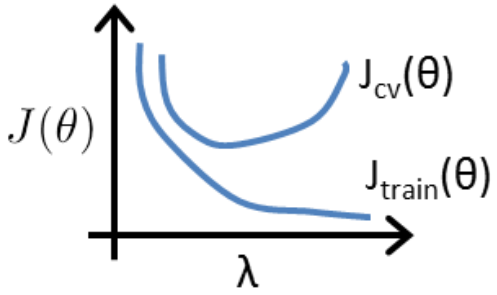
Consider regularized logistic regression. Let

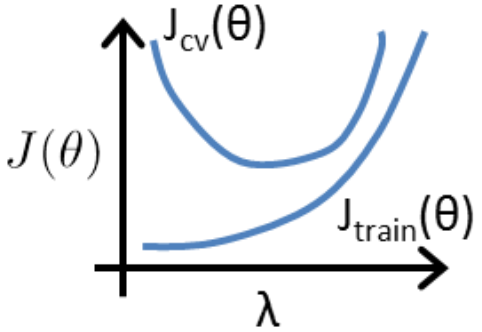
- $J(\theta) = \frac{1}{2m}[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=2}^{n} \theta_j^2]$
- $J_{\text{train}}(\theta) = \frac{1}{2m_{\text{train}}}[\sum_{i=1}^{m_{\text{train}}}(h_\theta(x_{\text{train}}^{(i)}) - y_{\text{train}}^{(i)})^2]$
- $J_{\text{CV}}(\theta) = \frac{1}{2m_{\text{CV}}}[\sum_{i=1}^{m_{\text{CV}}}(h_\theta(x_{\text{CV}}^{(i)}) - y_{\text{CV}}^{(i)})^2]$

Suppose you plot J_{train} and J_{CV} as a function of the regularization parameter λ. which of the following plots do you expect to get?









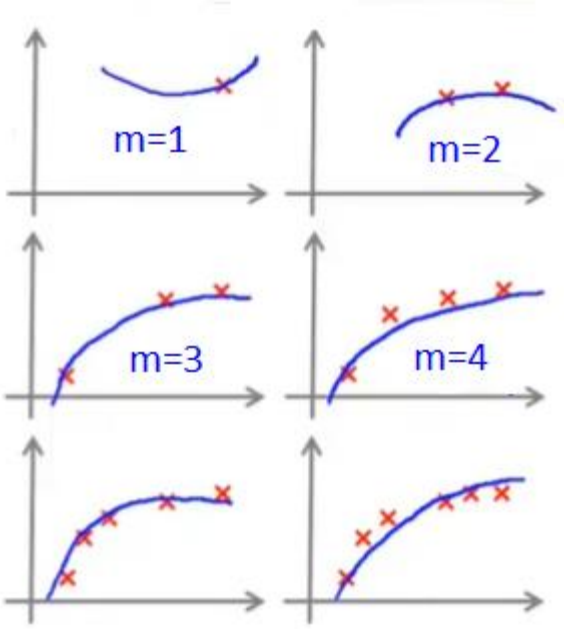
Learning Curves

Training an algorithm on a very few number of data points (such as 1, 2 or 3) will easily have 0 errors because we can always find a quadratic curve that touches exactly those number of points.

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

Hence:

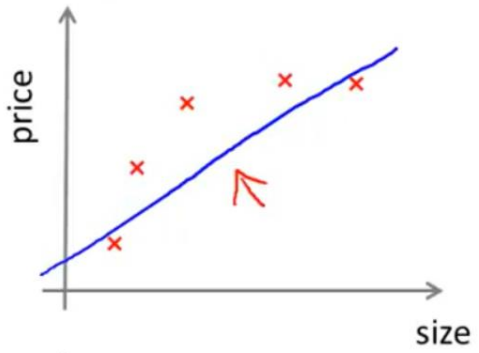
- As the training set gets larger, the error for a quadratic function increases.
- The error value will plateau out after a certain m, or training set size.



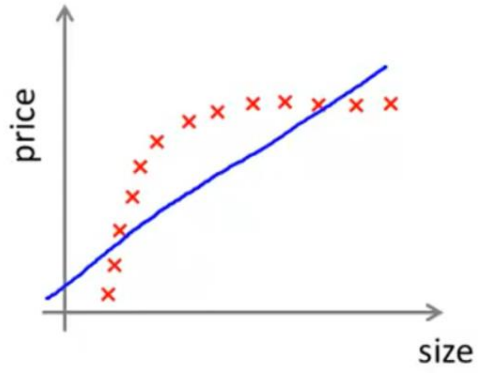
Experiencing high bias:

- **Low training set size:** causes $J_{train}(\Theta)$ to be low and $J_{CV}(\Theta)$ to be high.
- **Large training set size:** causes both $J_{train}(\Theta)$ and $J_{CV}(\Theta)$ to be high with $J_{train}(\Theta) \approx J_{CV}(\Theta)$.

Typical learning curve for high bias (at fixed model complexity)



$h_\Theta(x) = \theta_0 + \theta_1 x$
(linear hypothesis)

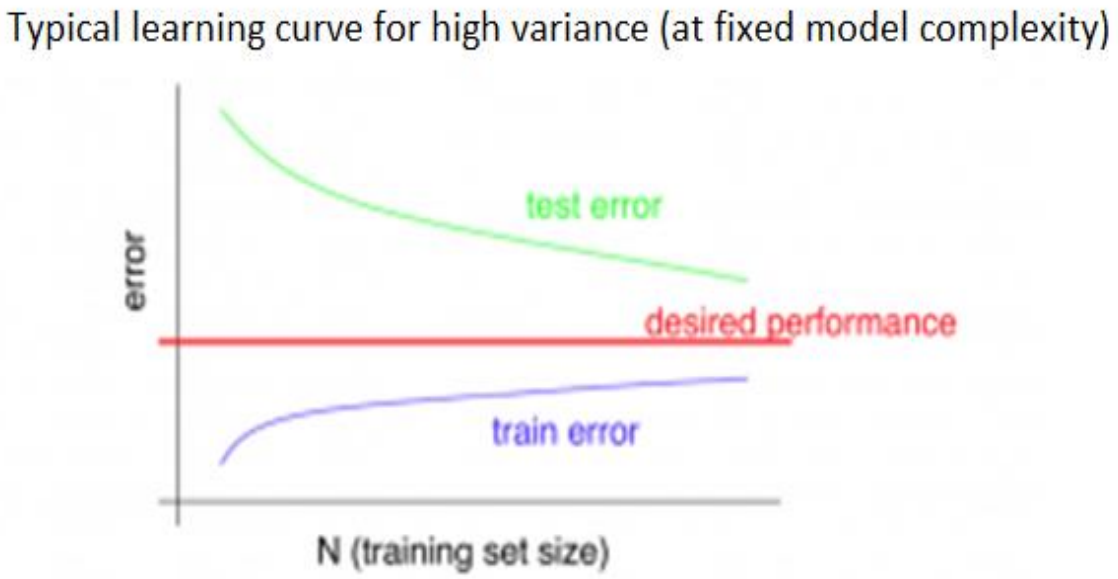


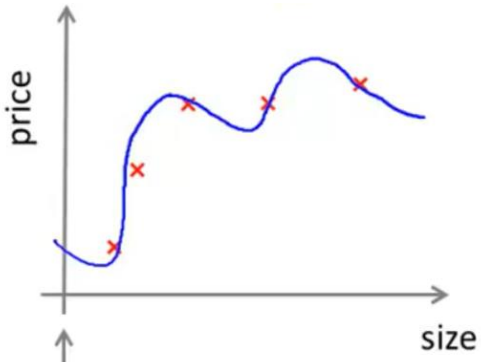
No improvement despite increasing training data

If a learning algorithm is suffering from high bias, getting more training data will not (by itself) help much.

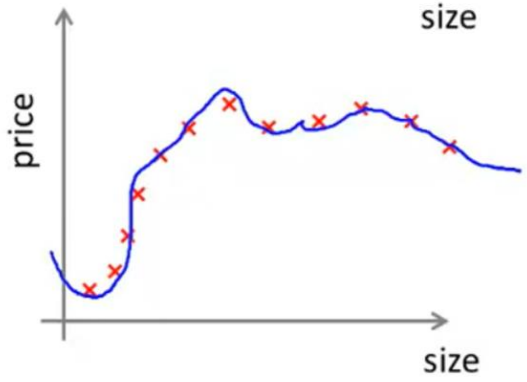
Experiencing high variance:

- **Low training set size:** $J_{train}(\Theta)$ will be low and $J_{CV}(\Theta)$ will be high.
- **Large training set size:**
  $J_{train}(\Theta)$ increases with training set size and $J_{CV}(\Theta)$ continues to decrease without leveling off.
  Also, $J_{train}(\Theta) < J_{CV}(\Theta)$ but the difference between them remains significant.





$h_\Theta(x) = \theta_0 + \theta_1 x + \ldots + \theta_{100} x^{100}$
(polynomial hypothesis and small λ)



Increasing the training data improves the fit

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

In which of the following circumstances is getting more training data likely to significantly help a learning algorithm's performance?

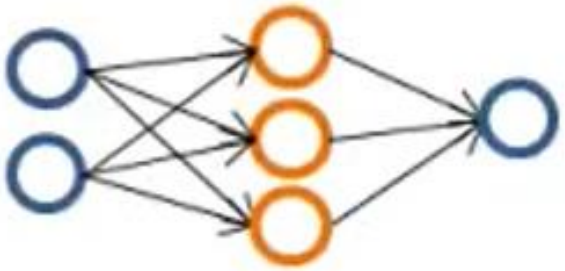
- Algorithm is suffering from high bias.
- Algorithm is suffering from high variance.
- $J_{CV}(\theta)$ (cross validation error) is much larger than $J_{train}(\theta)$ (training error).
- $J_{CV}(\theta)$ (cross validation error) is about the same as $J_{train}(\theta)$ (training error).

Deciding what to do next:

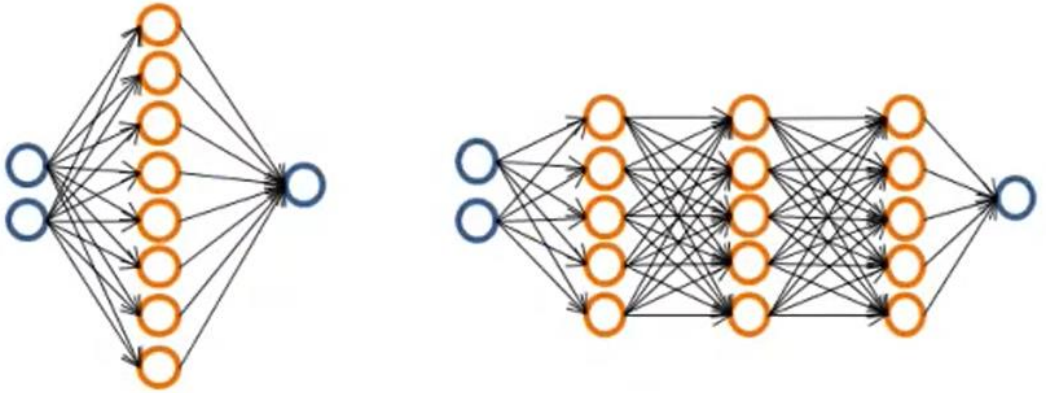
Our decision process can be broken down as follows:

- **Getting more training examples:** Fixes high variance
- **Trying smaller set of features:** Fixes high variance
- **Adding features:** Fixes high bias
- **Adding polynomial features:** Fixes high bias
- **Increasing λ:** Fixes high bias
- **Decreasing λ:** Fixes high variance

Neural networks and overfitting:



Small Neural Network



Large Neural Network

- A neural network with fewer parameters is prone to underfitting. It is also computationally cheaper.
- A large neural network with more parameters is prone to overfitting. It is also computationally expensive. In this case you can use regularization (increase λ) to address the overfitting.

Using a single hidden layer is a good starting default. You can train your neural network on a number of hidden layers using your cross validation set. You can then select the one that performs best.

Model Complexity Effects:

- Lower-order polynomials (low model complexity) have high bias and low variance.
  In this case, the model fits poorly consistently.

- Higher-order polynomials (high model complexity) fit the training data extremely well and the test data extremely poorly. These have low bias on the training data, but very high variance.

- In reality, we would want to choose a model somewhere in between, that can generalize well but also fits the data reasonably well.

Suppose you fit a neural network with one hidden layer to a training set. You find that the cross validation error $J_{CV}(\theta)$ is much larger than the training error $J_{train}(\theta)$.
Is increasing the number of hidden units likely to help?

- Yes, because this increases the number of parameters and lets the network represent more complex functions.
- Yes, because it is currently suffering from high bias.
- No, because it is currently suffering from high bias, so adding hidden units is unlikely to help.
- No, because it is currently suffering from high variance, so adding hidden units is unlikely to help.

Machine Learning System Design

Prioritizing what to work on:

Example: Spam Classifier (Supervised Learning)

- Given a data set of emails, we could construct a vector for each email.
- Each entry in this vector represents a word.
- The vector normally contains 10,000 to 50,000 entries gathered by finding the most frequently used words in our data set.
- If a word is to be found in the email, we would assign its respective entry a 1, else if it is not found, that entry would be a 0.
- Once we have all our x vectors ready, we train our algorithm and finally, we could use it to classify if an email is a spam or not.

From: cheapsales@buystufffromme.com
To: abc@cs.stanford.edu
Subject: Buy now!

Deal of the week! Buy now!
Rolex w4tches - $100
Med1cine (any kind) - $50
Also low cost M0rtgages available

SPAM

From: XYZ
To: abc@cs.stanford.edu
Subject: Summer vacation

Hey ABC,
This is your cousin XYZ. It's been a long time since we met. Want you to come over my place this summer to spend some time.

- XYZ

NON-SPAM

Spam emails often consist of $ symbol, words like "buy", "offer", "discount", irregular punctuations and deliberate misspellings.

x = features of email

y = spam (1) or not spam (0)

Features x: Choose 100 words indicative of spam/not spam.
E.g.: deal, buy, discount, offer, now, etc.

$$X = \begin{bmatrix} \text{buy} & 1 \\ \text{discount} & 0 \\ \dots & \\ \text{deal} & 1 \\ \text{offer} & 0 \\ \text{now} & 1 \\ \dots & \end{bmatrix} \quad \text{where, } X \in R^{100}$$

From: cheapsales@buystufffromme.com
To: abc@cs.stanford.edu
Subject: Buy now!

Deal of the week! Buy now!

$$\text{Thus, } x_j = \begin{cases} 1 & \text{if word j appears in email} \\ 0 & \text{otherwise} \end{cases}$$

Note: In practice, take most frequently occurring *n* words (10,000 to 50,000) in training set, rather than manually pick 100 words.

So how could you spend your time to improve the accuracy of this classifier?

- Collect lots of data
  Example: “Honeypot" project but doesn't always work.

- Develop sophisticated features based on email routing information
  Example: Using email header data in spam emails.

- Develop sophisticated features for message body.
  - Should “discount” and “discounts” be treated as the same word?
  - How about “deal” and “Dealer”?
  - Features about punctuation?

- Develop algorithms to process your input in different ways (recognizing misspellings in spam).
  Example: m0rtgage, med1cine, w4tches, etc.

It is difficult to tell which of the options will be most helpful.

Which of the following statements do you agree with? Check all that apply.

- For some learning applications, it is possible to imagine coming up with many different features (e.g. email body features, email routing features, etc.). But it can be hard to guess in advance which features will be the most useful.

- For spam classification, algorithms to detect and correct deliberate misspellings will make a significant improvement in accuracy.

- Because spam classification uses very high dimensional feature vectors (e.g. n=50,000 if the features capture the presence or absence of 50,000 different words), a significant effort to collect a massive training set will always be a good idea.

- There are often many possible ideas about how to develop a high accuracy learning system; “gut feeling” is not a recommended way to choose among the alternatives.

Error Analysis

Recommended approach:

- Start with a simple algorithm that you can implement quickly.
  Implement it and test it on your cross-validation data.

- Plot learning curves to decide more data, more features, etc. are likely to help.

- 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.

Error Analysis:

Assume that we have 500 emails i.e. $m_{CV} = 500$ examples in cross-validation set and the algorithm misclassifies 100 emails.

We can manually examine the 100 errors and categorize them based on:

(i) What type of email it is

- Pharma: 12
- Replica/Fake: 4
- Steal passwords: 53
- Other: 31

(ii) What cues (features) would help us classify these emails correctly

- Deliberate misspellings (m0rtgage, med1cine, etc.): 5
- Unusual email routing: 16
- Unusual punctuation: 32

- We find that by counting the number of types of email incorrectly classified, the algorithm is doing particularly poor on emails trying to steal passwords. Thus, we could find some features that are particular to those emails and add them to our model.
- Also, it would be better to focus on features like unusual punctuation to increase the model accuracy.

The importance of numerical evaluation:

- It is very important to get error results as a single, numerical value. Otherwise it is difficult to assess your algorithm's performance.

- For example if we use stemming, which is the process of treating the same word with different forms (fail/failing/failed) as one word (fail), and get a 3% error rate instead of 5%, then we should add it to our model.

- However, if we try to distinguish between upper-case and lower-case letters (Apple/apple) and end up getting a 3.2% error rate instead of 3%, then we should avoid using this new feature.

- Hence, we should try new things, get a numerical value for our error rate, and based on our result decide whether we want to keep the new feature or not.

Note: Porter stemmer is the most commonly used algorithm for stemming.

Why is the recommended approach to perform error analysis using the cross validation data used to compute $J_{CV}(\theta)$ rather than the test data used to compute $J_{test}(\theta)$?

- The cross validation data set is usually large.
- This process will give a lower error on the test set.
- If we develop new features by examining the test set, then we may end up choosing features that work well specifically for the test set, so $J_{test}(\theta)$ is no longer a good estimate of how well we generalize to new examples.
- Doing so is less likely to lead to choosing an excessive number of features.

Handling Skewed Data

Error Metrics for Skewed Classes:

Consider a cancer classification example.

- We train the logistic regression model $h_\theta(x)$ (y=1 if cancer and y=0 otherwise).
- Let's say we test our classifier on a test set and find that we get 1% error. Hence, we get 99% correct diagnosis.
- But now, let's say we find out that only 0.50% of patients in our training set actually have cancer.

Consider a piece of code of a non-learning algorithm as follows which takes input feature x but ignores it i.e. it always predicts y=0 (no cancer) irrespective of the input feature x. This algorithm would get 0.5% error.

```
function y = predictCancer(x)
      y = 0;          %ignore x
return
```

- A class is said to be skewed class if the no. of positive examples is much smaller than the no. of negative examples and vice-versa.
- In other words, we just have a lot more of examples from one class than from the other class.
- Using a skewed class may give high accuracy and low error but it's not always clear if doing so is really improving the quality of your classifier because predicting y = 0 all the time doesn't seem like a particularly good classifier.
- Hence, using classification accuracy metric is not always a good idea when the classes are skewed.

When we face such a skewed classes, we would want to come up with a different error metric called “Precision/Recall”.

Precision/Recall:

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

Predicted Class \ Actual Class	1	0
1	TRUE POSITIVE	FALSE POSITIVE
0	FALSE NEGATIVE	TRUE NEGATIVE

Precision:

Of all the patients where we predicted y=1, what fraction actually has cancer?

$$\frac{\text{TRUE POSITIVES}}{\text{\# PREDICTED POSITIVES}} = \frac{\text{TRUE POSITIVE}}{\text{TRUE POSITIVE} + \text{FALSE POSITIVE}}$$

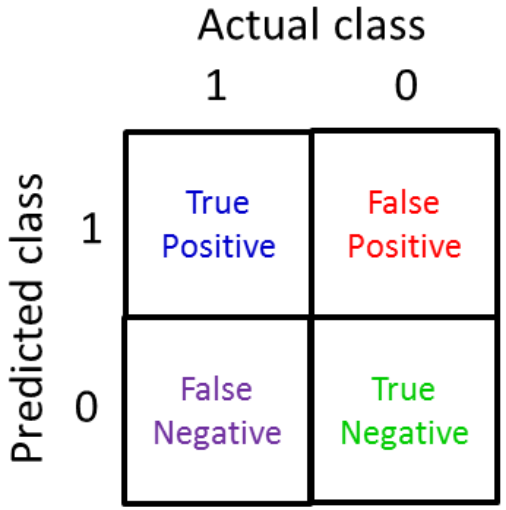
Recall:

Of all the patients that actually have cancer, what fraction did we correctly detect as having cancer?

$$\frac{\text{TRUE POSITIVES}}{\text{\#ACTUAL POSITIVES}} = \frac{\text{TRUE POSITIVE}}{\text{TRUE POSITIVE} + \text{FALSE NEGATIVE}}$$

Hence, Recall=0 for y=0 since there is no true positive case.

Precision and recall are defined according to:

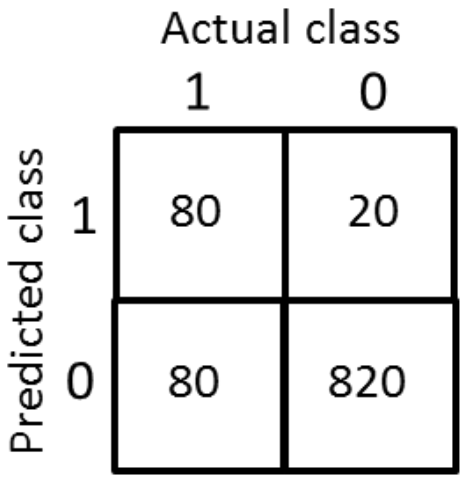


$$\text{Precision} = \frac{\text{TRUE POSITIVES}}{\text{\# PREDICTED POSITIVES}} = \frac{\text{TRUE POSITIVE}}{\text{TRUE POSITIVE+FALSE POSITIVE}}$$

$$\text{Recall} = \frac{\text{TRUE POSITIVES}}{\text{\#ACTUAL POSITIVES}} = \frac{\text{TRUE POSITIVE}}{\text{TRUE POSITIVE+FALSE NEGATIVE}}$$

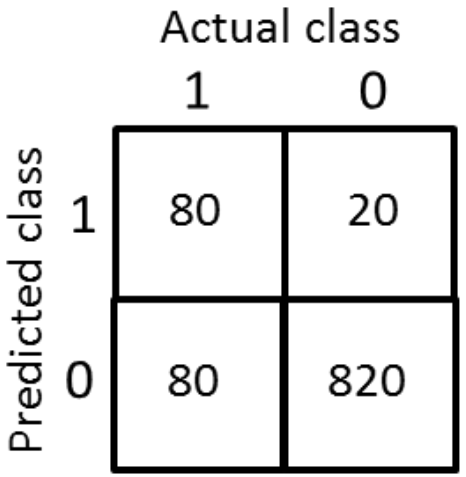
Your algorithm's performance on the test set is given to the right. What is the algorithm's precision?

- 0.8
- 0.5
- 0.08
- 0.1



Your algorithm’s performance on the test set is given to the right. What is the algorithm’s recall?

- 0.5
- 0.8
- 0.08
- 0.1



Trading off Precision and Recall

Consider the example of cancer classification.

Logistic regression: $0 \leq h_\theta(x) \leq 1$

- Predict 1 if $h_\theta(x) \geq 0.5$
- Predict 0 if $h_\theta(x) < 0.5$

Case I: Predicting cancer only if we are certain

Suppose we want to predict y=1 (cancer) only if we are very confident so that a patient who does not have cancer should not be wrongly classified as a cancer patient and get unnecessary treatment.
Hence, we increase the threshold to 0.7.

- Predict 1 if $h_\theta(x) \geq 0.7$ i.e. predict cancer only if we are more than 70% certain
- Predict 0 if $h_\theta(x) < 0.7$

Thus, precision increases and recall decreases.

Note: Precision increases and recall decreases with increase in threshold.

Case II: Avoid false negatives

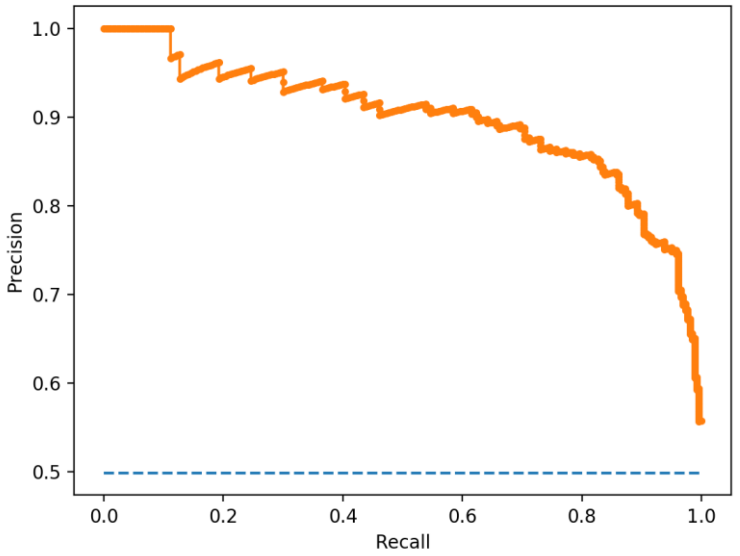
Suppose we want to avoid missing too many cases of cancer (avoid false negatives) i.e. patients who actually had cancer but were wrongly classified as non-cancer. This is dangerous as the patients would not get treatment which may lead to their death.

Hence, we decrease the threshold to 0.3.

- Predict 1 if $h_\theta(x) \geq 0.3$
- Predict 0 if $h_\theta(x) < 0.3$

Thus, precision decreases and recall increases.

- As the value of threshold varies, there is a trade-off between precision and recall which is plotted as shown.
- Note that the precision-recall curve can look in many different shapes. The curve shown here is just an example.



Precision vs Recall Curve

Note: Precision decreases and recall increases with decrease in threshold.

F_1 Score (F score):

How to compare precision/recall numbers?

Suppose we have three different algorithms with their precision and recall values given. Which algorithm is better to use?

We calculate a term called F_1 Score or F score and choose that algorithm having the maximum value of F_1 Score

$$F_1 \text{ Score} = 2\frac{PR}{P+R}$$

	Precision (P)	Recall (R)	F_1 Score
Algorithm 1	0.5	0.4	0.444
Algorithm 2	0.7	0.1	0.175
Algorithm 3	0.02	1.0	0.0392

← maximum

- F_1 Score = 0 if P=0 or R=0
- F_1 Score = 1 if P=1 and R=1

You have trained a logistic regression classifier and plan to make predictions according to:

- Predict y=1 if $h_\theta(x) \geq$ threshold
- Predict y=0 if f $h_\theta(x) <$ threshold

For different values of the threshold parameter, you get different values of precision (P) and recall (R). Which of of the following would be a reasonable way to pick the value to use for the threshold?

- Measure precision (P) and recall (R) on the **test set** and choose the value of threshold which maximizes $\frac{P+R}{2}$
- Measure precision (P) and recall (R) on the **test set** and choose the value of threshold which maximizes $2\frac{PR}{P+R}$
- Measure precision (P) and recall (R) on the **cross validation set** and choose the value of threshold which maximizes $\frac{P+R}{2}$
- Measure precision (P) and recall (R) on the **cross validation set** and choose the value of threshold which maximizes $2\frac{PR}{P+R}$

Data for Machine Learning

Designing a high accuracy learning system

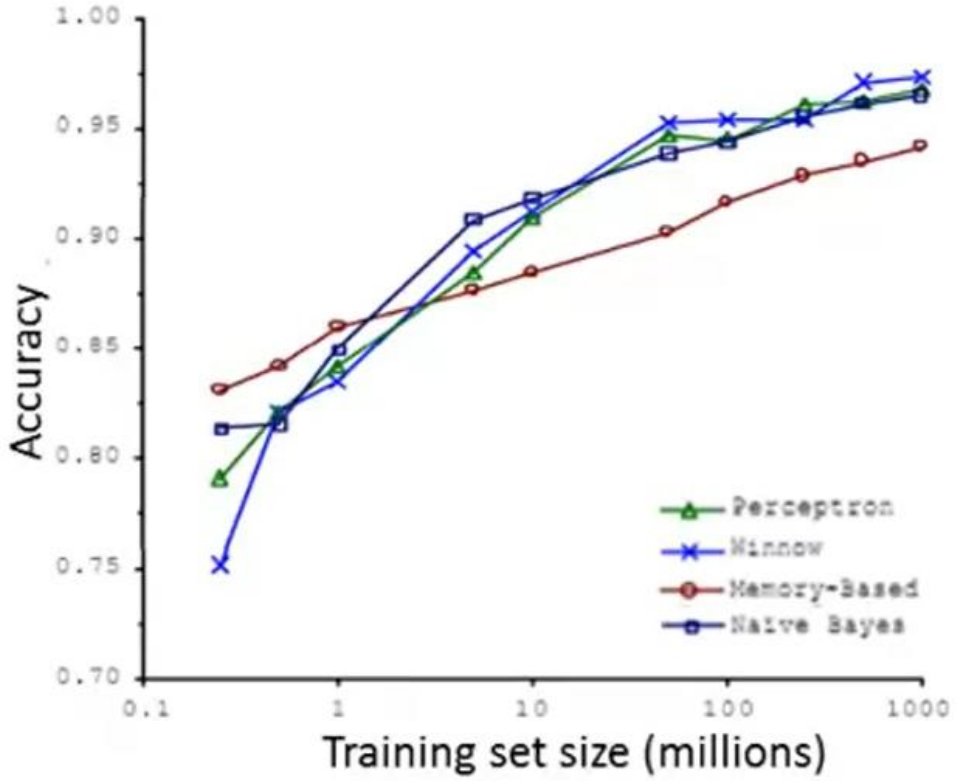
Example:

Classify between confusable words.
{to, too, two}, {then, than}

For breakfast I ate ____ eggs.

Algorithms:

- Perceptron (Logistic Regression)
- Winnow
- Memory-based
- Naïve-Bayes



All these algorithms give a similar performance for a large dataset.

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

[Banko and Brill, 2001]

Large data rationale

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

Example:
For breakfast I at ____ eggs. {two, too, to}

- We have enough information to find which word is suitable.
- Hence, increasing the dataset will also increase the performance of algorithm.

Counterexample:
Predict housing price from only size (sq. feet) and no other features.

- We don't have enough information to predict the price of house.
- Hence, even though we increase the size of data, there will be no improvement because we don't have information to tell.

Useful test: Given the input x, can a human expert confidently predict y? If yes, then y can be accurately predicted from the features x using an algorithm.

Large data rationale

- Suppose the features have enough information to predict the value of y.

- Use a learning algorithm with large number of parameters (e.g. logistic regression/linear regression with many features; neural network with many hidden units).
  These algorithms are very powerful if they are trained with lot of parameters that give rich information. Hence, they are "low bias" algorithms.
  Hence, $J_{train}(\theta)$ will be small.

- For a very large training set, overfitting is highly unlikely to happen. This eliminates the possibility of high variance too.
  Hence, $J_{train}(\theta) \approx J_{test}(\theta)$

- Since $J_{train}(\theta)$ is small and $J_{train}(\theta) \approx J_{test}(\theta)$, therefore, $J_{test}(\theta)$ will also be small.

- Hence, if we have a lot of data and we train a learning algorithm with lot of parameters, the performance of the algorithm would be high.

Having a large training set can help significantly improve a learning algorithm's performance. However, the large training set is **unlikely** to help when:

- The features x do not contain enough information to predict y accurately (such as predicting a house's price from only its size), and we are using a simple learning algorithm such as logistic regression.

- We are using a learning algorithm with a larger number of features (i.e. one with "low bias").

- The features x do not contain enough information to predict y accurately (such as predicting a house's price from only its size), even if we are using a neural network with a large number of hidden units.

- We are not using regularization (e.g.: the regularization parameter $\lambda = 0$).