G. Advice for applying the algorithm

I. Evaluating a learning algorithm

1. Evaluating a hypothesis

Have done some trouble shooting for errors

+ Getting more training examples

+ Trying smaller sets of features

+ Trying additional features

+ Trying polynomial features

+ Increasing or decreasing λ

A hypothesis may have a low error for the training examples but still be inaccurate (because of overfitting). Thus, to evaluate a hypothesis, given a dataset of training examples, split up the data into two sets: a training set and a test set. Typically, the training set consists of 70% of the data and the test set is the remaining 30%.

The new procedure using these two sets is then:

+ Learn Θ and minimize $J_{train}(\Theta)$ using the training set

+ Compute the test set error $J_{test}(\Theta)$

The test set error:

+ For linear regression:

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

+ For classification ~ Misclassification error (aka 0/1 misclassification error):

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

This gives us a binary 0 or 1 error result based on a misclassification. The average test error for the test set is:

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

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

2. Model selection and Training/Validation/Test sets

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 the hypothesis as measured on the data set with which were trained the parameters will be lower than the error on any other data set.

Given many models with different polynomial degrees to identify the 'best' function. In order to choose the model of hypothesis, test each degree of polynomial and look at the error result.

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

+ Training set: 60%
+ Cross validation set: 20%
+ Test set: 20%

Calculate three separate error values for the three different sets using the following method:

+ Optimize the parameters in Θ using training set for each polynomial degree.

+ Find the polynomial degree d with the least error using the cross validation set.

+ Estimate the generalization error using the test set with $J_{test}(\Theta^{(d)})$, (d = theta from polynomial with lower error);

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

II. Bias vs. Variance

1. Diagnosing Bias vs. Variance

The relationship between the degree of the polynomial d and the underfitting or overfitting of the hypothesis.

+ Distinguish whether bias or variance is the problem contributing to bad predictions.

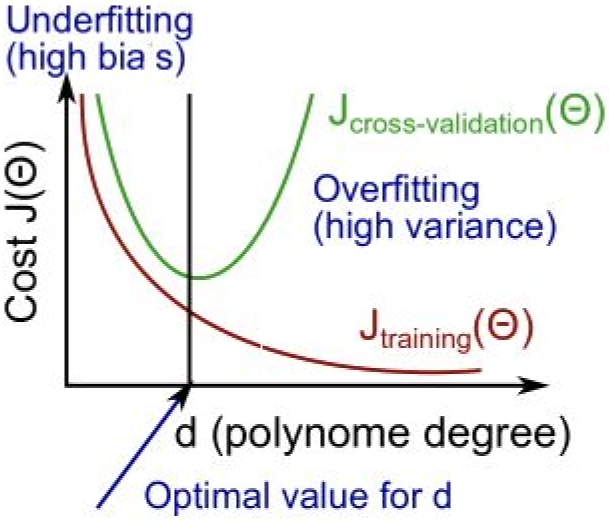
+ High bias is underfitting and high variance is overfitting. Ideally, find a golden mean between these two.

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

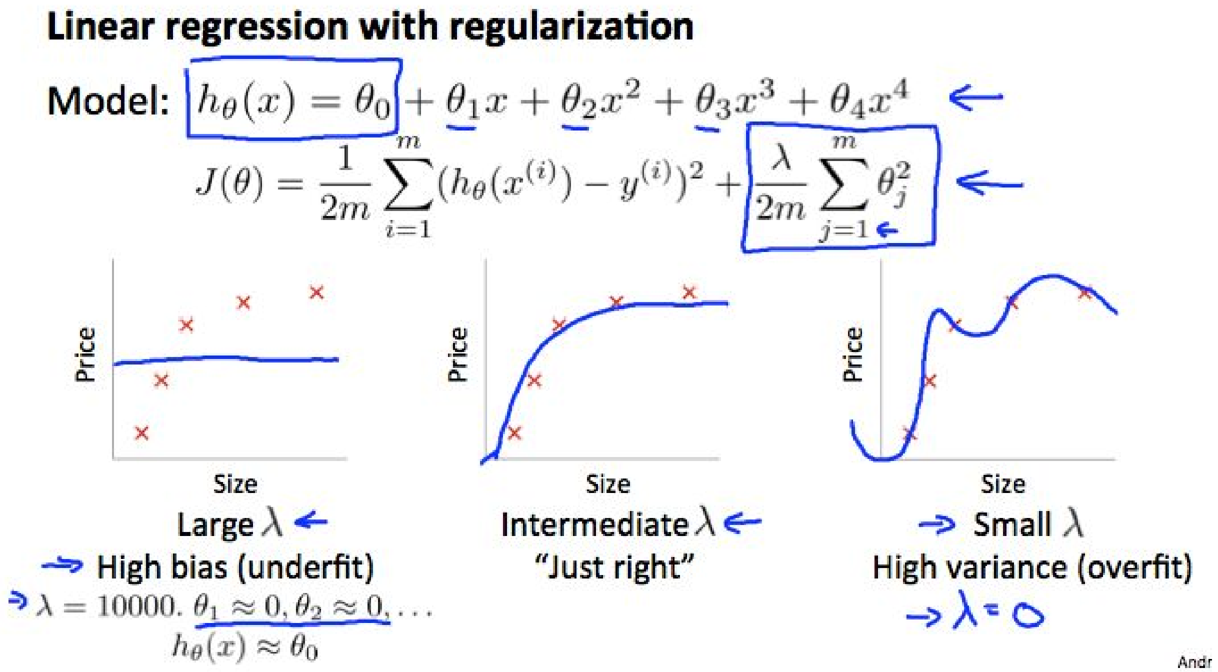
High bias (underfitting): both $J_{train}(\Theta)$ and $J_{CV}(\Theta)$ will be high. $J_{CV}(\Theta) \approx J_{train}(\Theta)$.

High variance (overfitting): $J_{train}(\Theta)$ will be low and $J_{CV}(\Theta)$ will be much greater than $J_{train}(\Theta)$.

Summarize:



2. Regularization and Bias/Variance



Linear regression with regularization

Model: $\boxed{h_\theta(x)=\theta_0}+\underline{\theta_1}x+\underline{\theta_2}x^2+\underline{\theta_3}x^3+\theta_4x^4 \leftarrow$

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

Large $\lambda \leftarrow$

$\rightarrow$ High bias (underfit)

$\rightarrow \lambda = 10000.\ \underline{\theta_1 \approx 0, \theta_2 \approx 0}, \ldots$

$h_\theta(x) \approx \theta_0$

Intermediate $\lambda \leftarrow$

"Just right"

$\rightarrow$ Small λ

High variance (overfit)

$\rightarrow \lambda = 0$

Andr

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

i/ Create a list of lambdas.

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

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

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

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

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

3. 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 always find a quadratic curve that touches exactly those number of points. Hence:

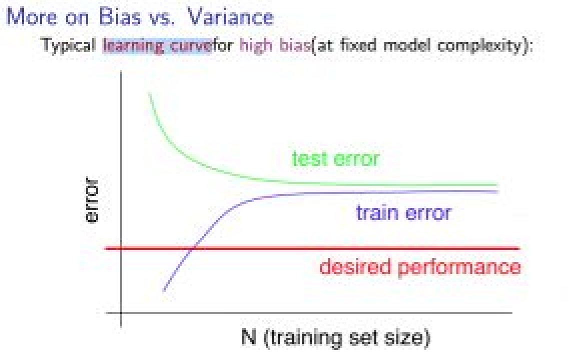
+ As the training set gets larger, the error for a quadratic function increases.

+ 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 $J_{train}(\Theta) \approx J_{CV}(\Theta)$.

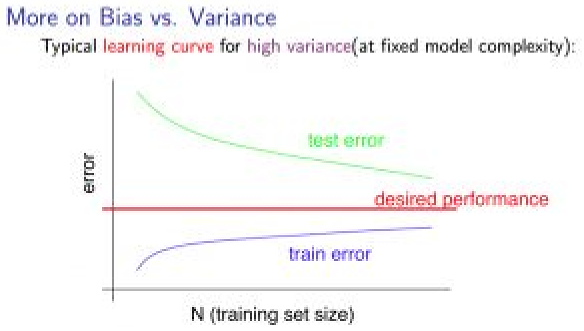


Experiencing high variance:

+ 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.

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



4. Deciding what to do next revisited

The decision process can be broken down as follows:

+ Getting more training examples: Fixes high variance

+ Trying smaller sets of features: Fixes high variance

+ Adding features: Fixes high bias

+ Adding polynomial features: Fixes high bias

+ Decreasing λ: Fixes high bias

+ Increasing λ: Fixes high variance.

Diagnosing Neural Networks

+ 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, use regularization (increase λ) to address the overfitting.

Using a single hidden layer is a good starting default. Train the neural network on a number of hidden layers using a cross validation set. 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.

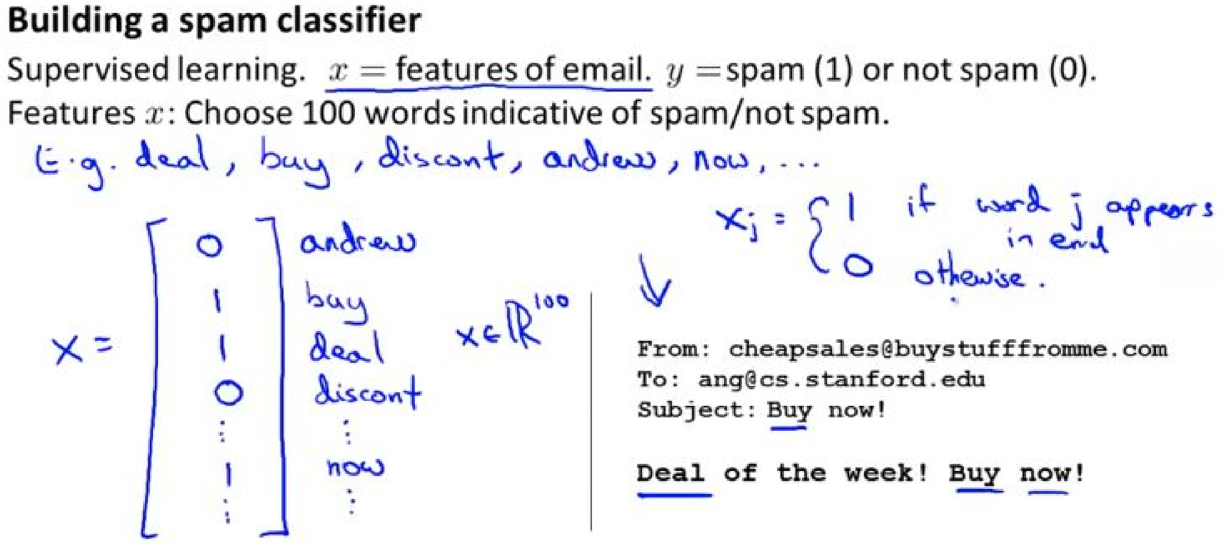
+ In reality, choose a model somewhere in between, that can generalize well but also fits the data reasonably well.

H. Machine learning system design

I. Building a spam classifier

1. Prioritizing What to Work On

System design example: 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 spam or not.



Building a spam classifier

Supervised learning. 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, discont, andrew, now, ...

$$x = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 1 \\ \vdots \end{bmatrix} \begin{matrix} \text{andrew} \\ \text{buy} \\ \text{deal} \\ \text{discont} \\ \vdots \\ \text{now} \\ \vdots \end{matrix} \qquad x \in \mathbb{R}^{100}$$

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

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

Deal of the week! Buy now!

So how to improve the accuracy of this classifier?

+ Collect lots of data

+ Develop sophisticated features (for example: using email header data in spam emails)

+ Develop algorithms to process the input in different ways (recognizing misspellings in spam).

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

2. Error analysis

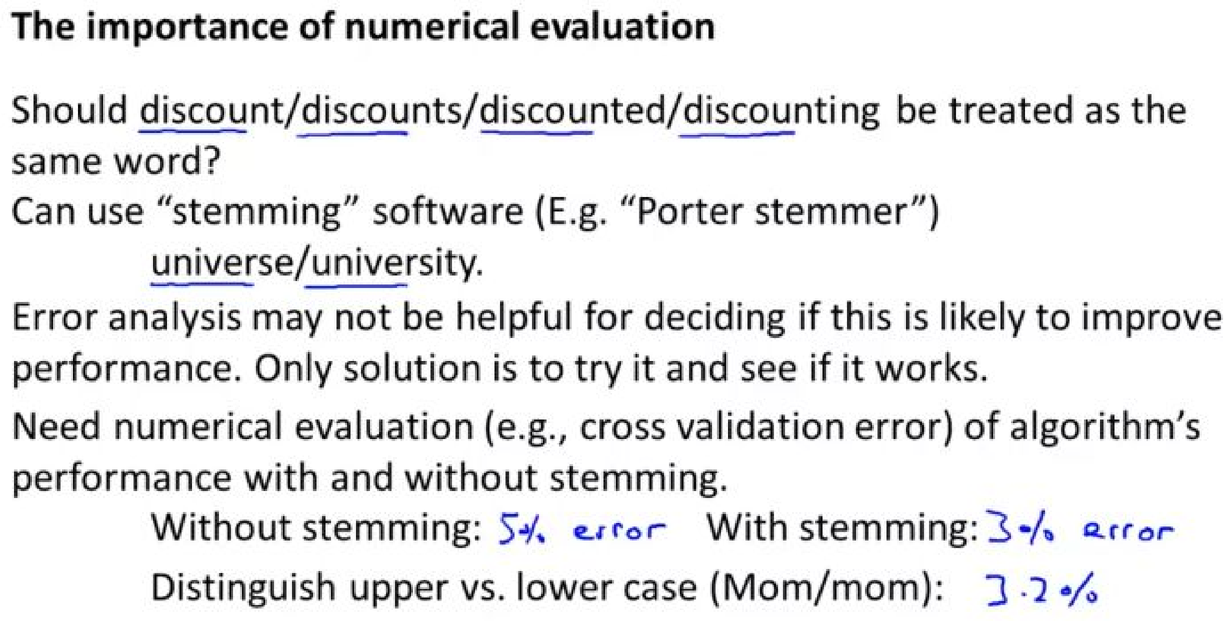
The recommended approach to solving machine learning problems is to:

+ Start with a simple algorithm, implement it quickly, and test it early on the cross validation data.

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

+ Manually examine the errors on examples in the cross validation set and try to spot a trend where most of the errors were made.

For example, assume that we have 500 emails and our algorithm misclassifies a 100 of them. We could manually analyze the 100 emails and categorize them based on what type of emails they are. We could then try to come up with new cues and features that would help us classify these 100 emails correctly. Hence, if most of our misclassified emails are those which try to steal passwords, then we could find some features that are particular to those emails and add them to our model. We could also see how classifying each word according to its root changes our error rate:



The importance of numerical evaluation

Should discount/discounts/discounted/discounting be treated as the same word?

Can use "stemming" software (E.g. "Porter stemmer")

universe/university.

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

Need numerical evaluation (e.g., cross validation error) of algorithm's performance with and without stemming.

Without stemming: 5% error With stemming: 3% error

Distinguish upper vs. lower case (Mom/mom): 3.2%

It is very important to get error results as a single, numerical value. Otherwise it is difficult to assess the algorithm's performance. Should try new things, get a numerical value for error rate, and based on the result decide whether to keep the new feature or not.