Week 6

Advice for applying Machine Learning

→ Evaluating a learning Algorithm

→ 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 you hypothesis on a new set of houses, you find that it makes unacceptably large errors in its predictions. What should I you try next in order to improve the learning algorithm?

- Get more training examples
- Try smaller set of features
- Trying getting additional features
- Try adding polynomial features
- Try decreasing λ
- Try increasing λ

加training examples
改features

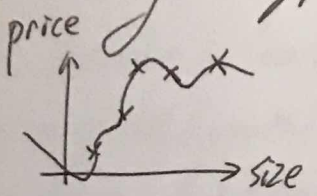
调参数

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.

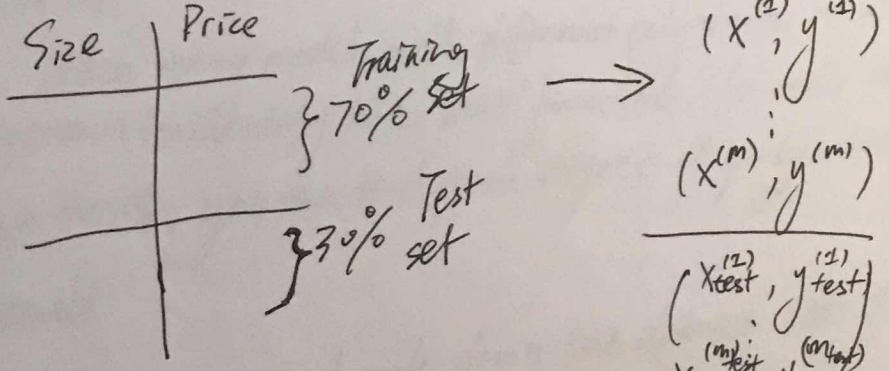
Diagnostics can take time to implement, but doing so can be a very good use of your time.

→ Evaluating a hypothesis



Fails to generalize to new examples not in training set.

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



$\rightarrow$ $(x^{(1)}, y^{(1)})$ ⋮ $(x^{(m)}, y^{(m)})$

$(x^{(1)}_{test}, y^{(1)}_{test})$ ⋮ $(x^{(m_{test})}_{test}, y^{(m_{test})}_{test})$

m_{test} = no. of test example

$(x^{(1)}_{test}, y^{(1)}_{test})$

Training/testing procedure for linear regression

- Learn parameter θ from training data (minimizing training error $J(\theta)$)
- Compute test set error

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\theta(x^{(i)}_{test}) - y^{(i)}_{test})^2$$

Similarly. Training/testing procedure for logistic regression

- Learn parameter θ from training data
- Compute test set error:

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

- Misclassification error (0/1 misclassification error):

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

$$\text{Test error} = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h_\theta(x^{(i)}_{test}), y^{(i)}_{test}).$$

→ Model Selection and Train/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 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.

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

Training set : 60%

Cross validation set : 20%

Test set : 20%

We can now calculate three seperate 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 $J_{test}(\Theta^{(d)})$, (d = theta from polynomial with lower error)

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

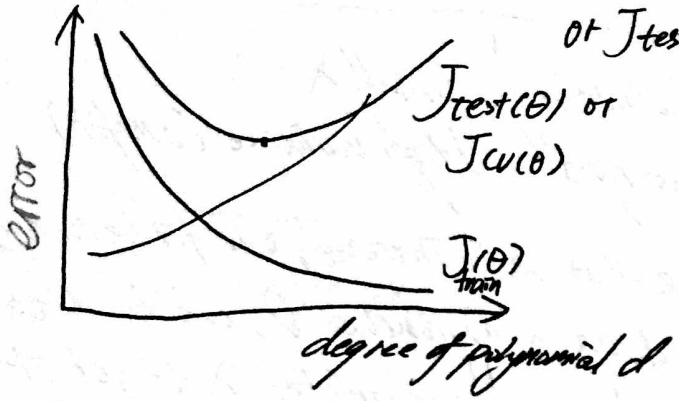
Advice for applying machine learning:

Diagnosing Bias vs. Variance

Bias / Variance:

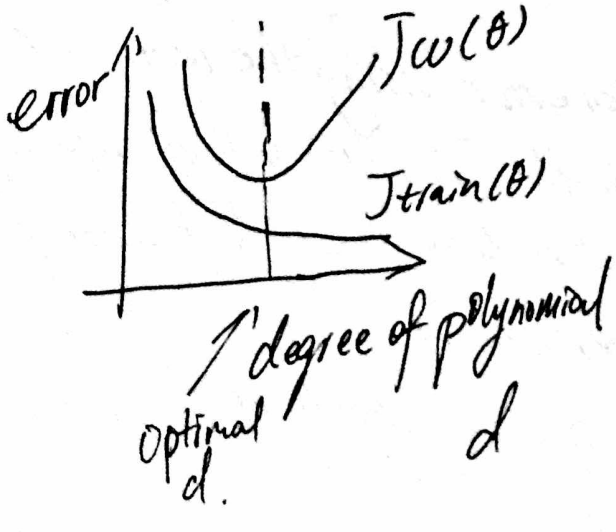
Training error: $J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$

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



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 problem or a variance problem?



Bias (underfit):

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

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

Variance (overfit):

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

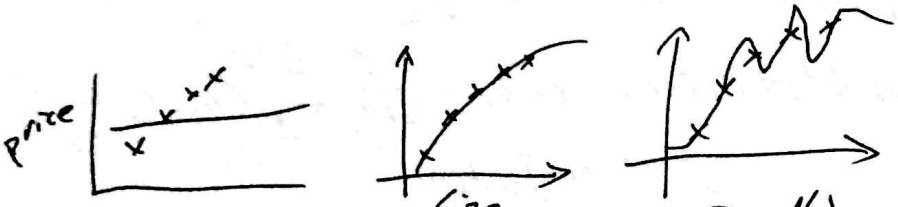
$J_{cv}(\theta) >> J_{train}(\theta)$

Regularization with Bias/Variance

Linear regression with regularization

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

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



Large λ — High bias (underfit)

Intermediate λ — "Just right"

Small λ — High variance (overfit)

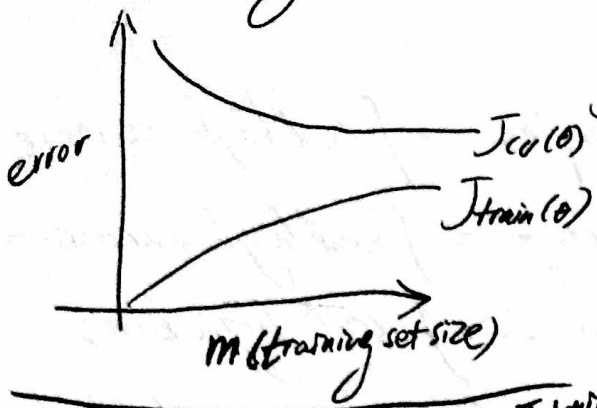
In the figure above, we see that as λ increase, our fit becomes more rigid. On the other hand, as λ approaches 0, we tend to overfit the data. So how do we choose our parameter λ to get it 'just right?' In order to choose the model and the regularization term λ, we need to:

1. Create a list of lambdas (i.e. $\lambda \in \{0, 0.01, 0.02, \ldots, 5.12, 10.24\}$)
2. Create a set of models with different degress or 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.

Learning Curves

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

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

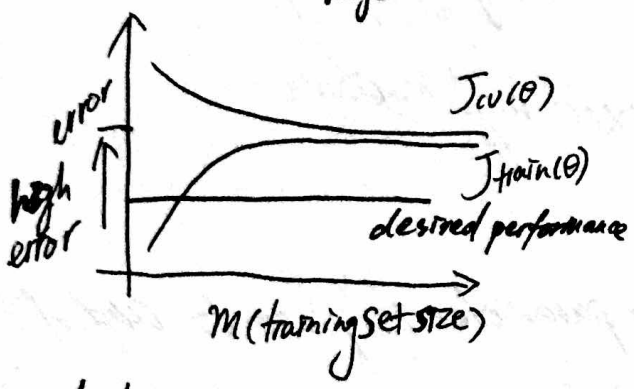


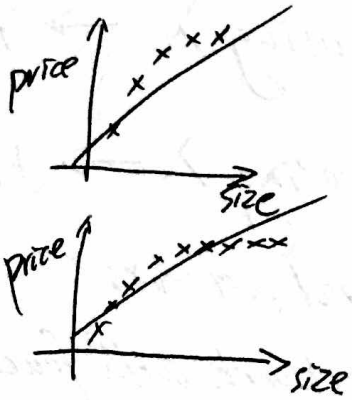
m Low: cause $J_{train}(\theta)$ to be low and $J_{cv}(\theta)$ to be high

m High: cause both $J_{train}(\theta)$ and $J_{cv}(\theta)$ to be high with $J_{train}(\theta) \approx J_{cv}(\theta)$

High bias, ~~m high~~ Large

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

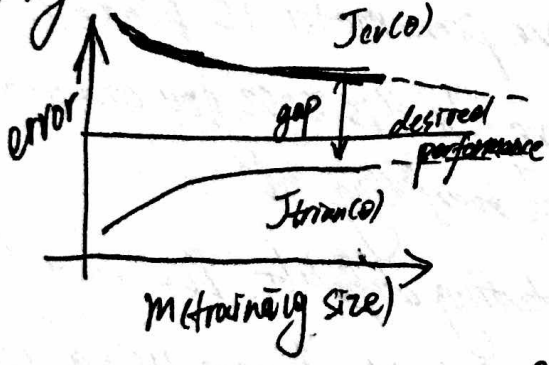


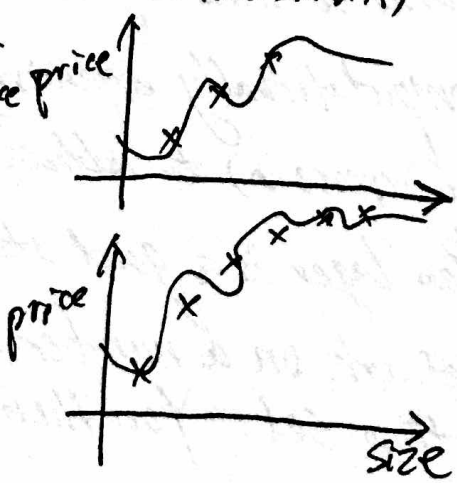


If a learning algorithm is suffering from high bias, getting more training data will not (by itself) help much

High variance:

$h_\theta(x) = \theta_0 + \theta_1 x + \dots + \theta_{100} x^{100}$ (and small λ)





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

m low: $J_{train}(\theta)$ will be low and $J_{cv}(\theta)$ will be high

m large: $J_{train}(\theta)$ 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.

Deciding What to do next

several solutions.

- Get ~~some~~ more training examples → fixed high variance
- Try small set of features → fixed high variance
- Try getting additional features → fixed high bias
- Try adding polynomial feature (x_1^2, x_2^2, x_1x_2, etc) → fixed high bias
- Try decreasing λ → Fixed high bias
- Try increasing λ → Fixed high variance

Diagnosing Neural Networks:

- A neural network with fewer parameter is prone to underfitting. It is also computationally cheaper.
- A large neural network with more parameter 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 then select the one that performs best.

Model Complexity effects:

- Lower-order polynomials (lower model complexity) have high bias and low variance. In this case, the model fits poor consistently.
- High-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, then can ~~categ~~ generalize well but also fits the data reasonably well.

Prioritizing What to Work On

System Design Example:

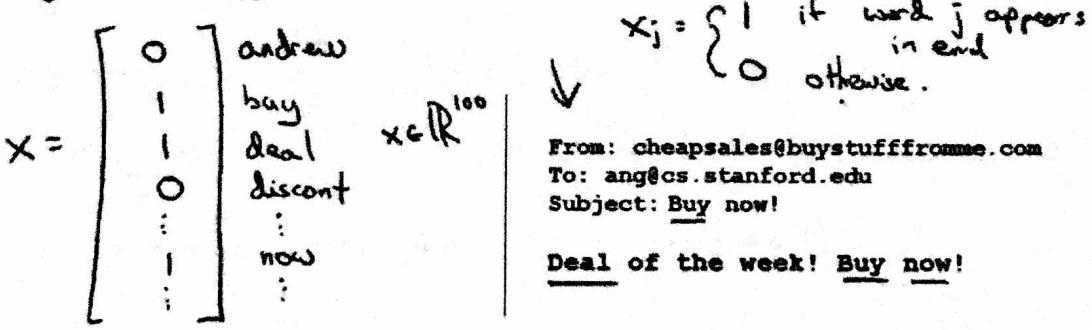
Given a data set of emails, we could construct a vector for each email. Each entry in this vector represents a word. The vector normally contains 10,000 to 50,000 entries gathered by finding the most frequently used words in our data set. If a word is to be found in the email, we would assign its respective 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 a 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, ...



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

- Collect lots of data (for example "honeypot" project but doesn't always work)
- Develop sophisticated features (for example: using email header data in spam emails)
- Develop algorithms to process your input in different ways (recognizing misspellings in spam).

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

✓ 完成

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 your 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.

Error analysis

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 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 definitely add it to our model. However, if we try to distinguish between upper case and lower case letters 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.

✓ 完成

error metrics for analysis

It is sometimes difficult to tell whether a reduction in error is actually a improvement of the algorithm.

. For example; In predicting a cancer diagnoses where 0.5% of the examples have cancer, we find our learning algorithm has a 1% error. However, if we were to simply classify every single example as a 0, then our error would reduce to 0.5% even though we did not improve the algorithm.

This usually happens with skewed classes, that is when our classes is very rare in the entire data set. or to say it another way, when we have lot more examples from one class than from the other class.

Precision / Recall:

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

		Actual class: 1	Actual class: 0
predicted class	1	True positive	False positive
	0	False Negative	True Negative

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

$$\frac{\text{True positives}}{\#\text{ predicted positive}} = \frac{\text{True Pos}}{\text{True Pos} + \text{False Pos}}$$

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

$$\frac{\text{True Positives}}{\#\text{ actual positive}} = \frac{\text{True Pos}}{\text{True Pos} + \text{False Neg}}$$

These two metrics give us a better sense of how our classifier is doing. We want both precision and recall to be high.

In the example at the beginning of the section, if we classify all patients as 0, then our recall will be $\frac{0}{0+f} = 0$, so despite having a lower error percentage, we can quickly see it has worse recall.

$$\text{Accuracy} = \frac{\text{true Pos} + \text{False Neg}}{\text{Total population}}$$

Note 1: If an algorithm predicts only negatives like it does in one of exercises, the precision is not defined, it is impossible to divide by 0. F1 score will not be defined too.

Tradding off precision and recall

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

Predict 1 if $h_\theta(x) \ge 0.5$ 0.7? 0.9?

Predict 0 if $h_\theta(x) < 0.5$ 0.7? 0.9?

Suppose we want to predict $y=1$ (cancer) only if very confident

Higher precision, lower recall

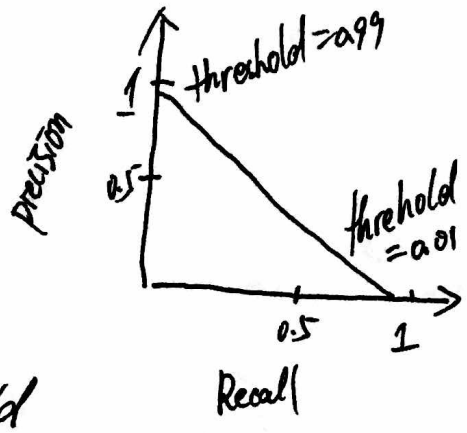
Suppose we want to avoid missing too many cases of cancer (avoid false negatives).

Higher recall, lower precision

More generally: Predict 1 if $h_\theta(x) \ge$ threhold

— precision $= \frac{\text{true Pos}}{\text{no. of predicted pos}}$

— recall $= \frac{\text{true Pos}}{\text{no. of actual pos.}}$



F1 Score (F score)

How to compare precision/recall numbers?

	Precision (P)	Recall (R)	Average	F1 Score
Algorithm 1	0.5	0.4	0.45	0.444
Algorithm 2	0.7	0.1	0.4	0.175
Algorithm 3	0.02	1.0	0.51	0.0392

Average: $\frac{P+R}{2}$ not a good way to evaluate

F1 Score: $2\frac{PR}{P+R}$

Designing a high accuracy learning system

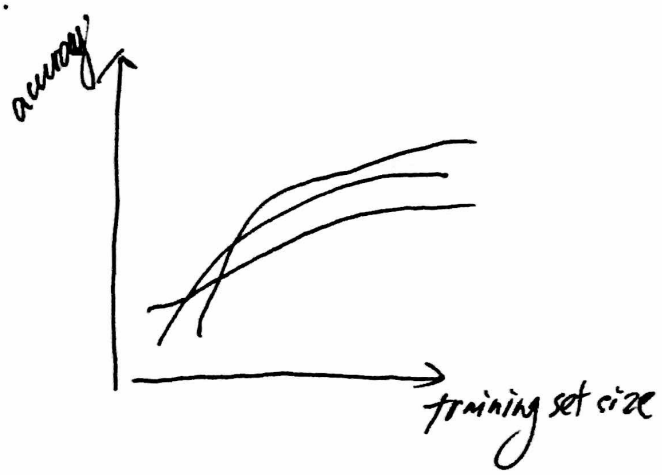
E.g. Classify between confusable words.

{to, two, too} {then, than}

For breakfast I ate __ eggs.

Algorithms

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



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.

Counterexample: Predict housing price from only size (feet^2) and no other features.

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

Use a learning algorith with many parameters (e.g. logistic regression/linear regression with many features: neural network with many hidden units)

low bias algorithm

→ $J_{train}(\theta)$ will be small

Use a very large training set (unlikely to overfit) → low variance

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

$J_{test}(\theta)$ will be small