Evaluating a Hypothesis

Once we have done some trouble shooting for errors in our predictions by:

- Getting more training examples
- Trying smaller sets of features
- Trying additional features
- Trying polynomial features
- Increasing or decreasing λ

We can move on to evaluate our new hypothesis.

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, we can split up the data into two sets: a **training set** and a **test set**. Typically, the training set consists of 70 % of your data and the test set is the remaining 30 %.

The new procedure using these two sets is then:

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

The test set error

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

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.
- The error of your hypothesis as measured on the data set with which you trained the parameters will be lower than any other data set.

In order to choose the model of your hypothesis, you can test each degree of polynomial and look at the error result.

Without the Validation Set (note: this is a bad method - do not use it)

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

In this case, we have trained one variable, d, or the degree of the polynomial, using the test set. This will cause our error value to be greater for any other set of data.

Use of the CV set

To solve this, we can introduce a third set, the **Cross Validation Set**, to serve as an intermediate set that we can train d with. Then our test set will give us an accurate, non-optimistic error.

One example 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 separate error values for the three different sets.

With the Validation Set (note: this method presumes we do not also use the CV set for regularization)

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 the test set.

(Mentor note: be aware that using the CV set to select 'd' means that we cannot also use it for the validation curve process of setting the lambda value).

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. We need to find a golden mean between these two.

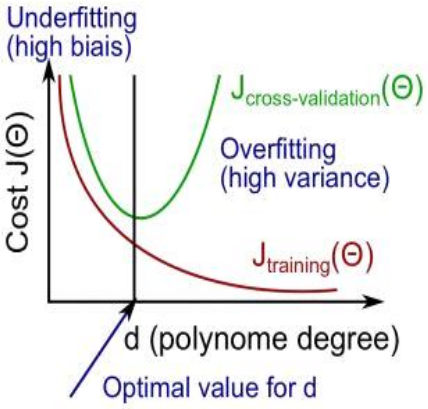
The training error will tend to **decrease** as we increase the degree d of the polynomial.

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

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

The is represented in the figure below:



Regularization and Bias/Variance

Instead of looking at the degree d contributing to bias/variance, now we will look at the regularization parameter λ.

- Large λ: High bias (underfitting)
- Intermediate λ: just right
- Small λ: High variance (overfitting)

A large lambda heavily penalizes all the Θ parameters, which greatly simplifies the line of our resulting function, so causes underfitting.

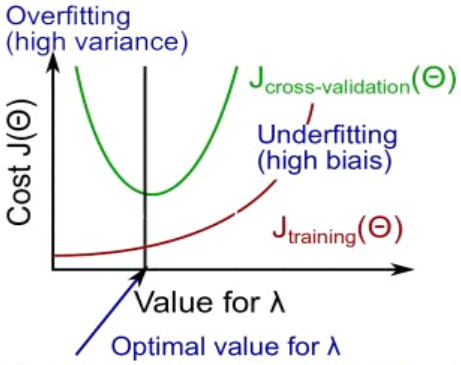
The relationship of λ to the training set and the variance set is as follows:

Low λ: $J_{train}(\Theta)$ is low and $J_{CV}(\Theta)$ is high (high variance/overfitting).

Intermediate λ: $J_{train}(\Theta)$ and $J_{CV}(\Theta)$ are somewhat low and $J_{train}(\Theta) \approx J_{CV}(\Theta)$.

Large λ: both $J_{train}(\Theta)$ and $J_{CV}(\Theta)$ will be high (underfitting /high bias)

The figure below illustrates the relationship between lambda and the hypothesis:



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

1. Create a list of lambdas (i.e. λ∈{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 λ = 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

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. Hence:

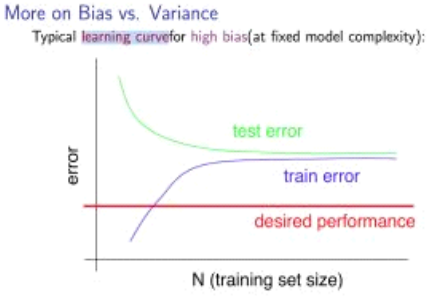
- As the training set gets larger, the error for a quadratic function increases.
- The error value will plateau out after a certain m, or training set size.

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

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

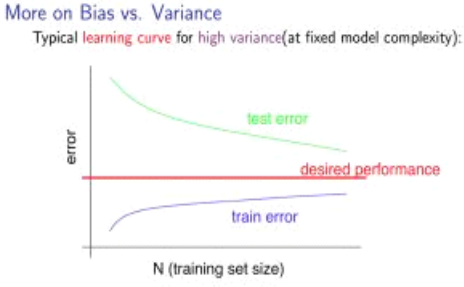


Experiencing high variance:

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.

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



Deciding What to Do Next Revisited

Our 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 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.

Thursday, May 20, 2021 10:52 PM

Model Selection:

Choosing M the order of polynomials.

How can we tell which parameters Θ to leave in the model (known as "model selection")?

There are several ways to solve this problem:

- Get more data (very difficult).
- Choose the model which best fits the data without overfitting (very difficult).
- Reduce the opportunity for overfitting through regularization.

Bias: approximation error (Difference between expected value and optimal value)

- High Bias = UnderFitting (BU)
- $J_{train}(\Theta)$ and $J_{CV}(\Theta)$ both will be high and $J_{train}(\Theta) \approx J_{CV}(\Theta)$

Variance: estimation error due to finite data

- High Variance = OverFitting (VO)
- $J_{train}(\Theta)$ is low and $J_{CV}(\Theta) \gg J_{train}(\Theta)$

Intuition for the bias-variance trade-off:

- Complex model => sensitive to data => much affected by changes in X => high variance, low bias.
- Simple model => more rigid => does not change as much with changes in X => low variance, high bias.

One of the most important goals in learning: finding a model that is just right in the bias-variance trade-off.

Regularization Effects:

- Small values of λ allow model to become finely tuned to noise leading to large variance => overfitting.
- Large values of λ pull weight parameters to zero leading to large bias => underfitting.

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.

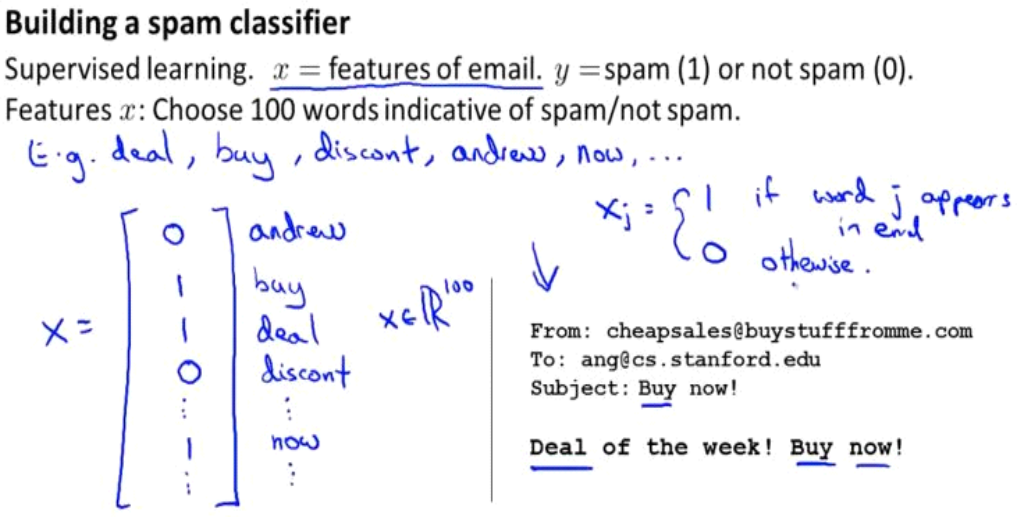
A typical rule of thumb when running diagnostics is:

- More training examples fixes high variance but not high bias.
- Fewer features fixes high variance but not high bias.
- Additional features fixes high bias but not high variance.
- The addition of polynomial and interaction features fixes high bias but not high variance.
- When using gradient descent, decreasing lambda can fix high bias and increasing lambda can fix high variance (lambda is the regularization parameter).
- When using neural networks, small neural networks are more prone to under-fitting and big neural networks are prone to over-fitting. Cross-validation of network size is a way to choose alternatives.

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.



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.

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.

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:

Error Metrics for Skewed Classes

It is sometimes difficult to tell whether a reduction in error is actually an 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 class 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.

For this we can use **Precision/Recall**.

- Predicted: 1, Actual: 1 --- True positive
- Predicted: 0, Actual: 0 --- True negative
- Predicted: 0, Actual, 1 --- False negative
- Predicted: 1, Actual: 0 --- False positive

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

$$\frac{\text{True Positives}}{\text{Total number of predicted positives}} = \frac{\text{True Positives}}{\text{True Positives} + \text{False positives}}$$

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

$$\frac{\text{True Positives}}{\text{Total number of actual positives}} = \frac{\text{True Positives}}{\text{True Positives} + \text{False negatives}}$$

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.

Accuracy = $\frac{truepositive+truenegative}{totalpopulation}$

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.

Trading Off Precision and Recall

We might want a **confident** prediction of two classes using logistic regression. One way is to increase our threshold:

- Predict 1 if: $h_\theta(x) \geq 0.7$
- Predict 0 if: $h_\theta(x) < 0.7$

This way, we only predict cancer if the patient has a 70% chance.

Doing this, we will have **higher precision** but **lower recall** (refer to the definitions in the previous section).

In the opposite example, we can lower our threshold:

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

That way, we get a very **safe** prediction. This will cause **higher recall** but **lower precision**.

The greater the threshold, the greater the precision and the lower the recall.

The lower the threshold, the greater the recall and the lower the precision.

In order to turn these two metrics into one single number, we can take the **F value**.

One way is to take the **average**:

$$\frac{P+R}{2}$$

This does not work well. If we predict all y=0 then that will bring the average up despite having 0 recall. If we predict all examples as y=1, then the very high recall will bring up the average despite having 0 precision.

A better way is to compute the **F Score** (or F1 score):

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

In order for the F Score to be large, both precision and recall must be large.

We want to train precision and recall on the **cross validation set** so as not to bias our test set.