Advice for Applying Machine Learning

Contents

1 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
- 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 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 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):

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

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

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

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

2 Model Selection 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 three sets is:

- Training set: 60%

- Cross Validation set: 20%
- Test set: 20%

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

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

3 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.

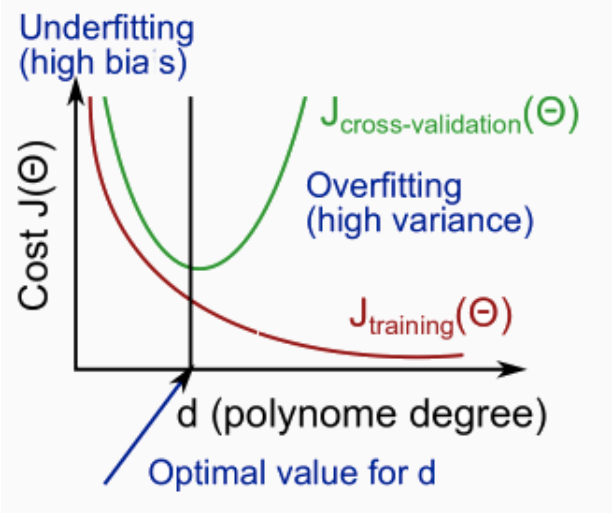
The training error will tend to **decrease** as we increase the degree d of the polynomial.

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

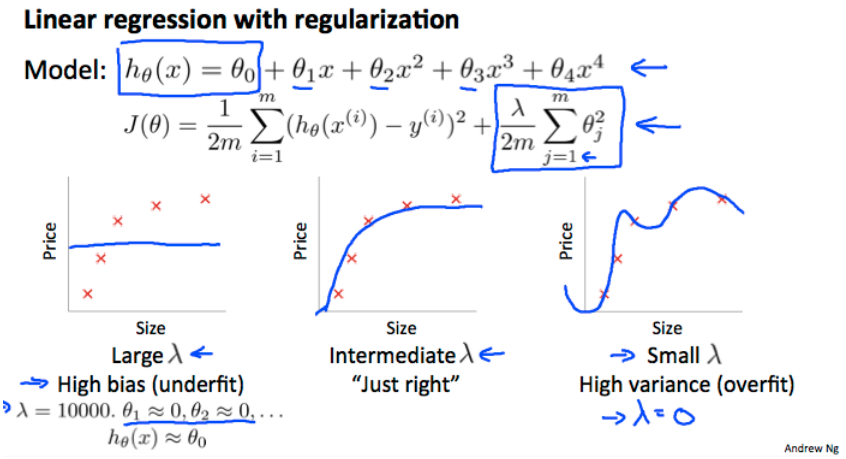
High variance (overfitting): $J_{train}(\Theta)$ will be low and $J_{CV}(\Theta)$ will be much greater than $J_{train}(\Theta)$.

This is summarized in the figure below:



4 Regularization and Bias/Variance

Note: The regularization term below and through out the video should be $\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$ and **NOT** $\frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$.



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 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. $\lambda \in \{0, 0.01, 0.02, 0.04, 0.08, 0.16, 0.32, 0.64, 1.28, 2.56, 5.12, 10.24\}$);
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.

5 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 platgeau out after a certain m, or trining set size.

Experincing 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.

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

Experincing 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.

Typical learning curve for high variance (at fixed model complexity):

6 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. You can then select the one that performas 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 trining 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.

7 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.

8 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.

Based on the following confusion matrix we 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

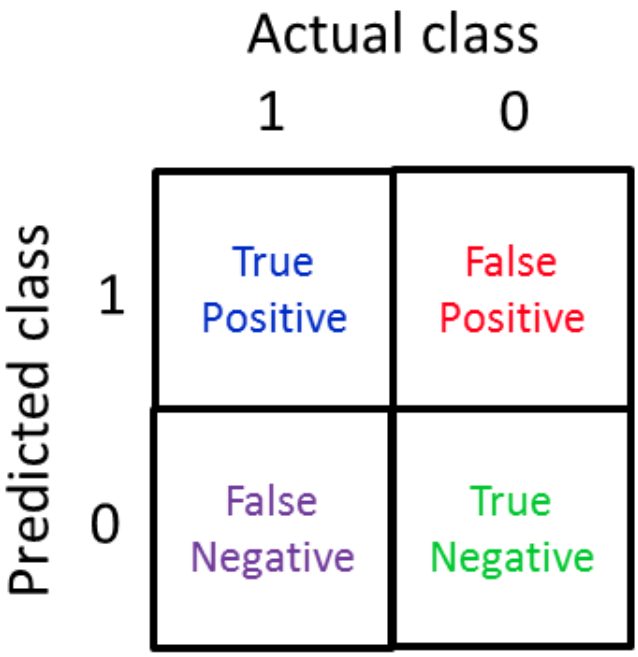
letters and end up getting a 3.2% error rate instead of 3%, then we should avoid using this new feature. Hence, we should try new things, get a numerical value for our error rate, and based on our result decide whether we want to keep the new feature or not.

9 Handling Skewed Data

Error metrics for skewed classes

With skewed data the **accuracy** metric $= \frac{\text{(true positives + true negatives)}}{\text{(total examples)}}$ may not give you the true picture, since the proportion of true negatives is the highest and the same result could be achieve simply classifying all examples as y = 0. Instead, **Precision** and **Recall** are used as evaluation metrics for classification problems with rare class ($y = 1$) that we want to detect.

Precision and **Recall** are defined according to the following confusion matrix:



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

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

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

$$Recall = \frac{\text{True positive}}{\text{\# actual positive}} = \frac{\text{True positives}}{\text{True positives + False negatives}}$$

10 Trading off precision and recall

Predict 1 if $h_\theta(x) \geq$ threshold. By varying the threshold, you can control a trade off between precision and recall.

- With threshold > 0.5, we want to predict y = 1 (e.g. cancer) only if very confident. $\Rightarrow$ Higher precision, lower recall
- With threshold < 0.5, we want to avoid missing too many cases of cancer (avoid false negatives). $\Rightarrow$ Higher recall, lower precision.

To automatically set the threshold to decide what's really y=1 and y=0 would be to try a range of different values of thresholds and evaluate these on the cross validation set and choose the value of threshold which maximizes $F_1 Score = 2\frac{PR}{P+R}$.

11 Data for machine learning

Large data rationale

Assume feature $x \in \Re^{n+1}$ has sufficient information to predict y accurataly. An useful test for the above assumption is to ask: Given the input x, can a human expert confidently predict y?

Then, using a learning algorithm with many parameters(e.g. logistic regression/linear regression with many features; neural network with many hidden units) gives **low bias algoritms** and a small $J_{train}(\Theta)$.

Adding examples and working with a very large training set, becames very unlikely to overfit, so we get **low variance** and $J_{train}(\Theta) \approx J_{CV}(\Theta)$.

At the end, using this large data rationale we can achieve a small $J_{test}(\Theta)$.