Advice for applying Machine Learning

Contents

Summary

So after implementation, the algorithm is making unacceptably large errors: what to do next? Two key concepts are high bias and high variance, which are explained in detail in the following pages.

But, as a summary, the following rectangle contains the basic strategies to tackle this:

To fix high bias/underfitting:

- **Try getting aditional features**: For example, if you try to predict the price of a house just by the size, you might be ignoring other crucial factors like zone, age of the house, etc, and your model will have high bias. By adding extra features, in this case the data will fit much better.
- **Make your model more complex:**
  - **Try adding polynomial features**: Is similar to adding aditional features - a linear function might not be able to fit the data well, and by adding polinomial features the function will become more complex and fit the training data better.
  - **Try decreasing the regularization parameter λ**: It will increase the size of the weights and lead to a more complex function able to fit the data better.

To fix high variance/overfitting:

- **Get more training examples**: Will help making the function more complex, improving how it fits the training set while also generalizing better.
- **Simplify your model:**
  - **Try a smaller set of features**: Sometimes having too many features makes the algorithm try to fit very complex functions, increasing the variance. If you suspect your algorithm has plenty of features which are not helpful or redundant, then eliminating them will remove the flexibility of the algorithm to fit the data.
  - **Try increasing the regularization parameter λ**: This will reduce the weights, making the function smoother and reducing its complexity and high variance.

"*The concept of high bias and high variance takes a short time to learn, but takes a whole life to master*"

Let's explore this further!

Machine Learning diagnostics

We split the data into 2 sets: training and test.

With **linear regression**, we fit the model with the training set (and with regularization, if applicable) and compute both, the training error and test error (note: when computing the error, the regularization is not included):

Train/test procedure for linear regression (with squared error cost)

Fit parameters by minimizing cost function $J(\vec{w}, b)$

$$J(\vec{w}, b) = \left[\frac{1}{2m_{train}} \sum_{i=1}^{m_{train}} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m_{train}} \sum_{j=1}^{n} w_j^2\right]$$

Compute test error:

$$J_{test}(\vec{w}, b) = \frac{1}{2m_{test}} \left[\sum_{i=1}^{m_{test}} \left(f_{\vec{w},b}\left(\vec{x}_{test}^{(i)}\right) - y_{test}^{(i)}\right)^2\right]$$

Compute training error:

$$J_{train}(\vec{w}, b) = \frac{1}{2m_{train}} \left[\sum_{i=1}^{m_{train}} \left(f_{\vec{w},b}\left(\vec{x}_{train}^{(i)}\right) - y_{train}^{(i)}\right)^2\right]$$

With **classification**, we can apply very similar ideas:

Train/test procedure for classification problem

0/1

Fit parameters by minimizing $J(\vec{w}, b)$ to find $\vec{w}, b$

E.g.,

$$J(\vec{w}, b) = -\frac{1}{m_{train}} \sum_{i=1}^{m_{train}} \left[y^{(i)} \log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) + (1 - y^{(i)}) \log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right)\right] + \frac{\lambda}{2m_{train}} \sum_{j=1}^{n} w_j^2$$

Compute test error:

$$J_{test}(\vec{w}, b) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} \left[y_{test}^{(i)} \log\left(f_{\vec{w},b}\left(\vec{x}_{test}^{(i)}\right)\right) + \left(1 - y_{test}^{(i)}\right) \log\left(1 - f_{\vec{w},b}\left(\vec{x}_{test}^{(i)}\right)\right)\right]$$

Compute train error:

$$J_{train}(\vec{w}, b) = -\frac{1}{m_{train}} \sum_{i=1}^{m_{train}} \left[y_{train}^{(i)} \log\left(f_{\vec{w},b}\left(\vec{x}_{train}^{(i)}\right)\right) + \left(1 - y_{train}^{(i)}\right) \log\left(1 - f_{\vec{w},b}\left(\vec{x}_{train}^{(i)}\right)\right)\right]$$

Log loss will work well, although in classification is more commonly used the proportion of the data that has been misclassified:

We want the test error to be as low as possible, in which case the model generalizes well.

Train/test procedure for classification problem

fraction of the test set and the fraction of the train set that the algorithm has misclassified.

$$\hat{y} = \begin{cases} 1 \text{ if } f_{\vec{w},b}(\vec{x}^{(i)}) \geq 0.5 \\ 0 \text{ if } f_{\vec{w},b}(\vec{x}^{(i)}) < 0.5 \end{cases}$$

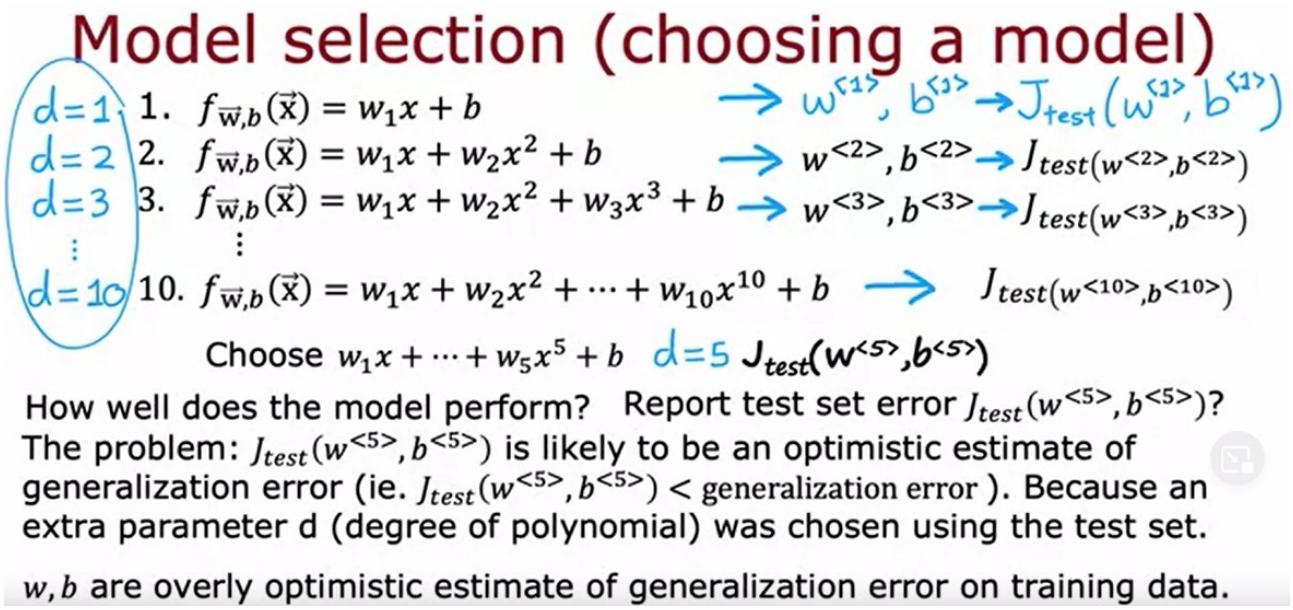
count $\hat{y} \neq y$

$J_{test}(\vec{w}, b)$ is the fraction of the test set that has been misclassified.

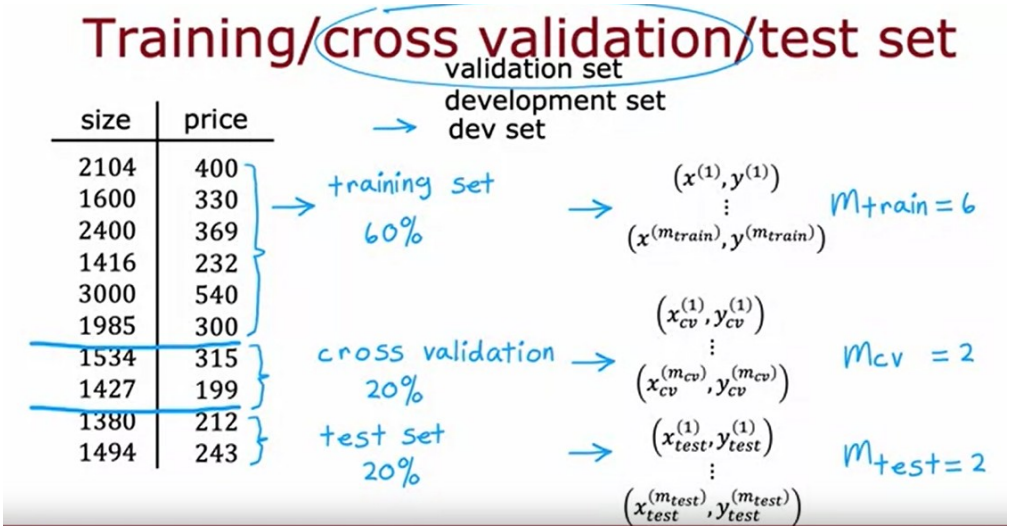
$J_{train}(\vec{w}, b)$ is the fraction of the train set that has been misclassified.

Model selection

Dividing into training and test sets when selecting a model (for example, the degree of a polynomial) is a flawed approach, as our test error will be optimistic compared with the generalization error (this is, the error for completely new or unknown data). This is because out test data was used to select out model, which means that this data fits particularly well with this polynomial, and some skewedness is inevitable. Andrew Ng explains it as follows:



Solution? Add another set: we split the data into training, cross-validation and test sets (Andrew does 60% training, 20% cv, 20% test).



Andrew explains the cross-validation set is also called in other ways; he likes *dev set* because is shorter.

Training/cross validation/test set

Training error: $J_{train}(\vec{w}, b) = \frac{1}{2m_{train}} \left[\sum_{i=1}^{m_{train}} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right)^2 \right]$

Cross validation error: $J_{cv}(\vec{w}, b) = \frac{1}{2m_{cv}} \left[\sum_{i=1}^{m_{cv}} \left(f_{\vec{w},b}(\vec{x}_{cv}^{(i)}) - y_{cv}^{(i)} \right)^2 \right]$ (validation error, dev error)

Test error: $J_{test}(\vec{w}, b) = \frac{1}{2m_{test}} \left[\sum_{i=1}^{m_{test}} \left(f_{\vec{w},b}(\vec{x}_{test}^{(i)}) - y_{test}^{(i)} \right)^2 \right]$

We then use:

1. Training data: to fit all the different models
2. Validation/Dev data: To choose the model with lowest dev error.
3. Test data: The test error is a fair value for our generalizarion error (the one we expect for new data)

Model selection

$d=1$ 1. $f_{\vec{w},b}(\vec{x}) = w_1 x + b$ $w^{\langle 1 \rangle}, b^{\langle 1 \rangle}$ → $J_{cv}(w^{\langle 1 \rangle}, b^{\langle 1 \rangle})$

$d=2$ 2. $f_{\vec{w},b}(\vec{x}) = w_1 x + w_2 x^2 + b$ → $J_{cv}(w^{\langle 2 \rangle}, b^{\langle 2 \rangle})$

$d=3$ 3. $f_{\vec{w},b}(\vec{x}) = w_1 x + w_2 x^2 + w_3 x^3 + b$

⋮ ⋮ ⋮

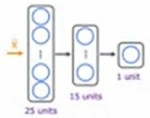
$d=10$ 10. $f_{\vec{w},b}(\vec{x}) = w_1 x + w_2 x^2 + \cdots + w_{10} x^{10} + b$ $J_{cv}(w^{\langle 10 \rangle}, b^{\langle 10 \rangle})$

→ Pick $w_1 x + \cdots + w_4 x^4 + b$ $(J_{cv}(w^{<4>}, b^{<4>}))$

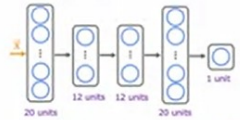
Estimate generalization error using test the set: $J_{test}(w^{<4>}, b^{<4>})$

This can be applied for other models:

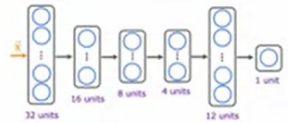
Model selection – choosing a neural network architecture

1. 

$w^{\langle 1 \rangle}, b^{\langle 1 \rangle}$ $J_{cv}(w^{\langle 1 \rangle}, b^{\langle 1 \rangle})$

→ 2. 

$w^{\langle 2 \rangle}, b^{\langle 2 \rangle}$ $J_{cv}(w^{\langle 2 \rangle}, b^{\langle 2 \rangle})$

3. 

$w^{\langle 3 \rangle}, b^{\langle 3 \rangle}$ $J_{cv}(w^{\langle 3 \rangle}, b^{\langle 3 \rangle})$

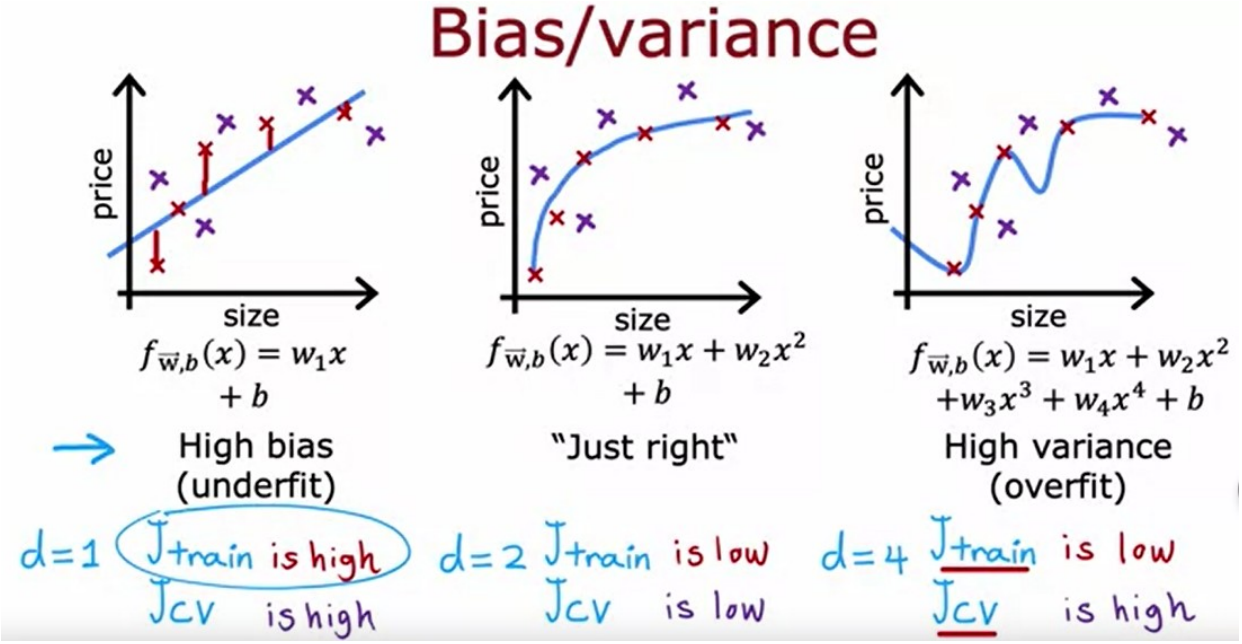
Pick $w^{<2>}, b^{<2>}$

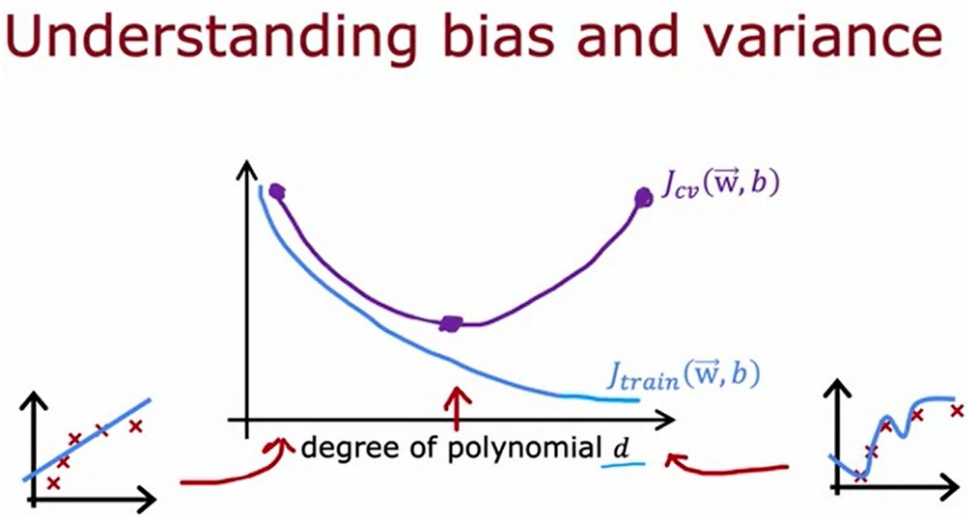
Estimate generalization error using the test set: $J_{test}(w^{<2>}, b^{(<2>)})$

Bias and Variance

This section is presented with the assumption there is no regularization.

High bias: the model does not perform well even in the training set.
High variance: the model performs very well with the training set, but fails at generalizing.



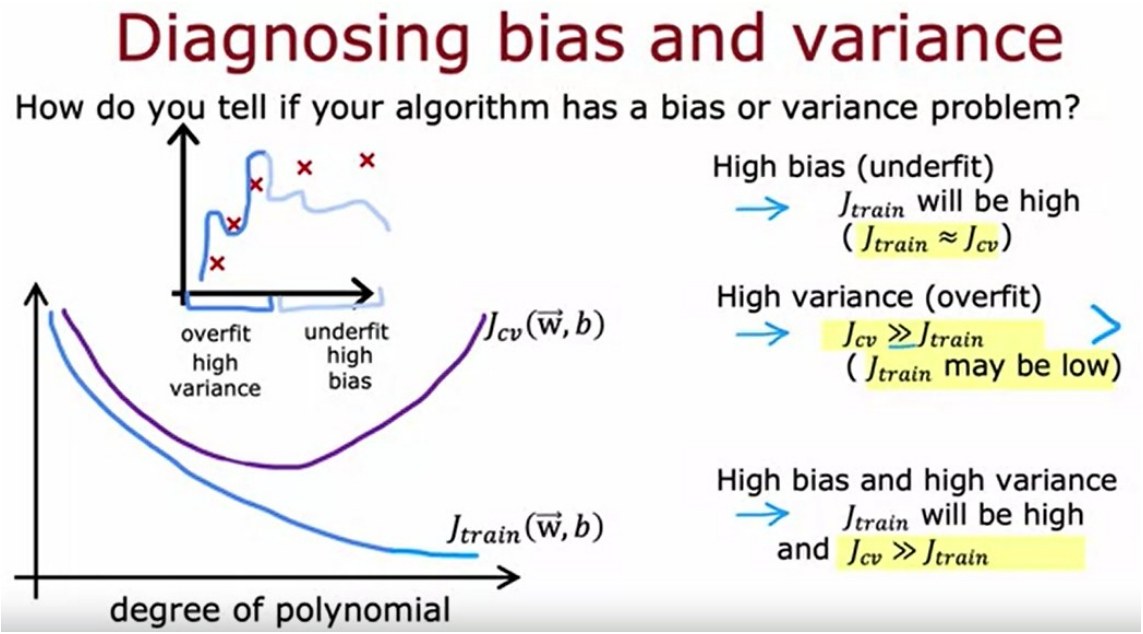


The lowest point of the Jcv curve (the "valley" spot) is where we find the best performing polynomial. Before that, the model is prone to high bias (it underfits), and after it to high variance(it overfits).

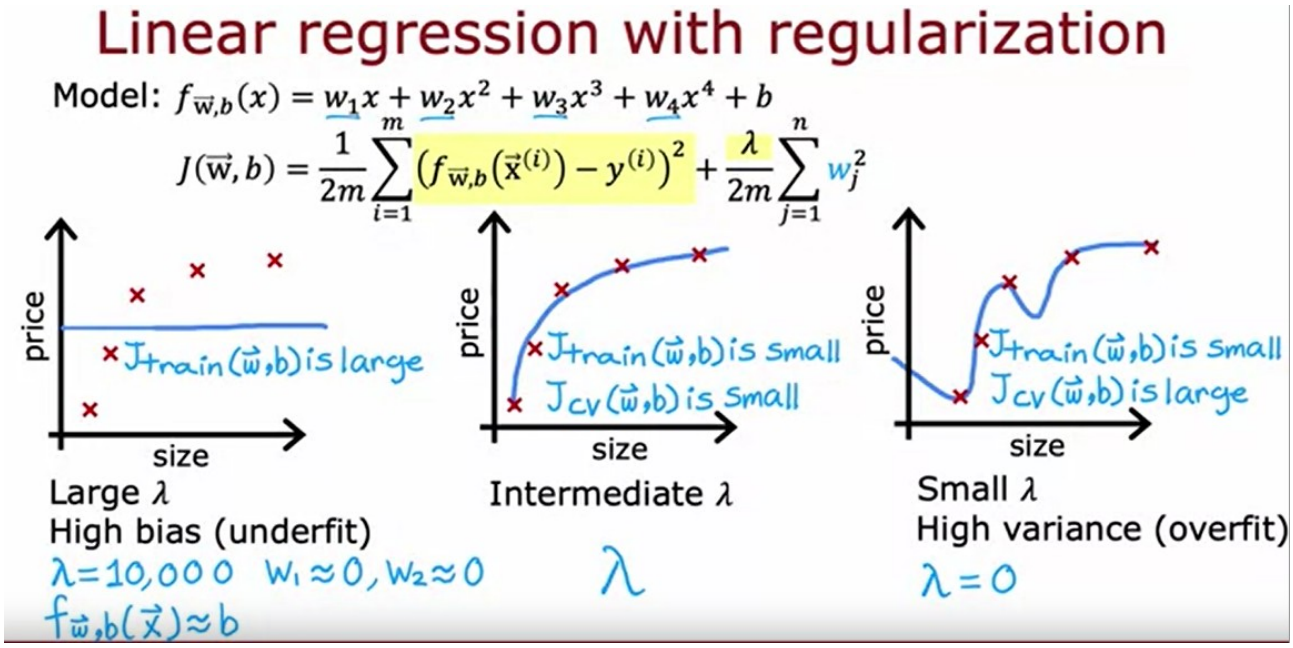
Sometimes you can have simultaneously high vias and high variance; you won't see this happen as much in linear regression, but if you are training neural networks, there are some appications where you might have high vias and high variance.To recognize this, you see the training error is high, but the cv error is even much higher. An explanation is that for

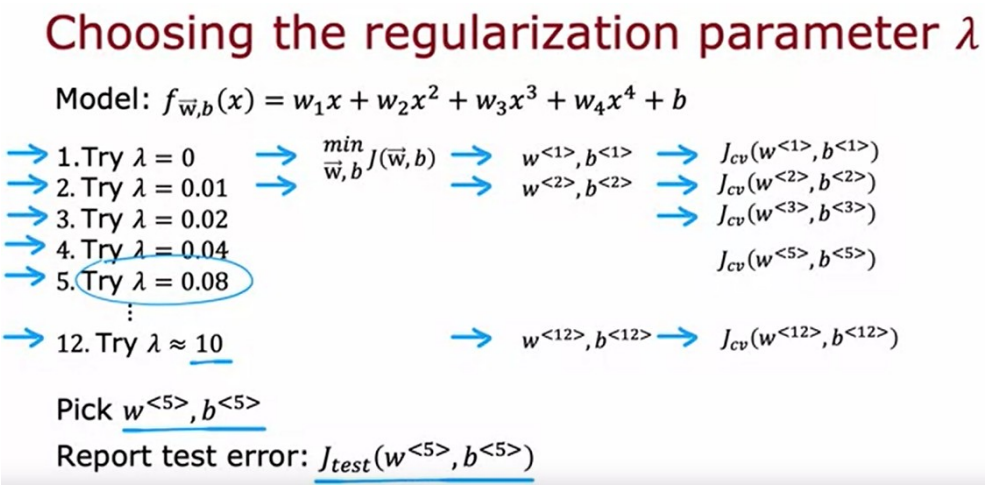
some data the model clearly underfits but, for the other, it overfits (which is very unlikely to happen in linear regression).

This is explained in the next image, with a small picture replicating what in linear regression would mean simultaneus high bias and high variance (a bit of an artifitial example):

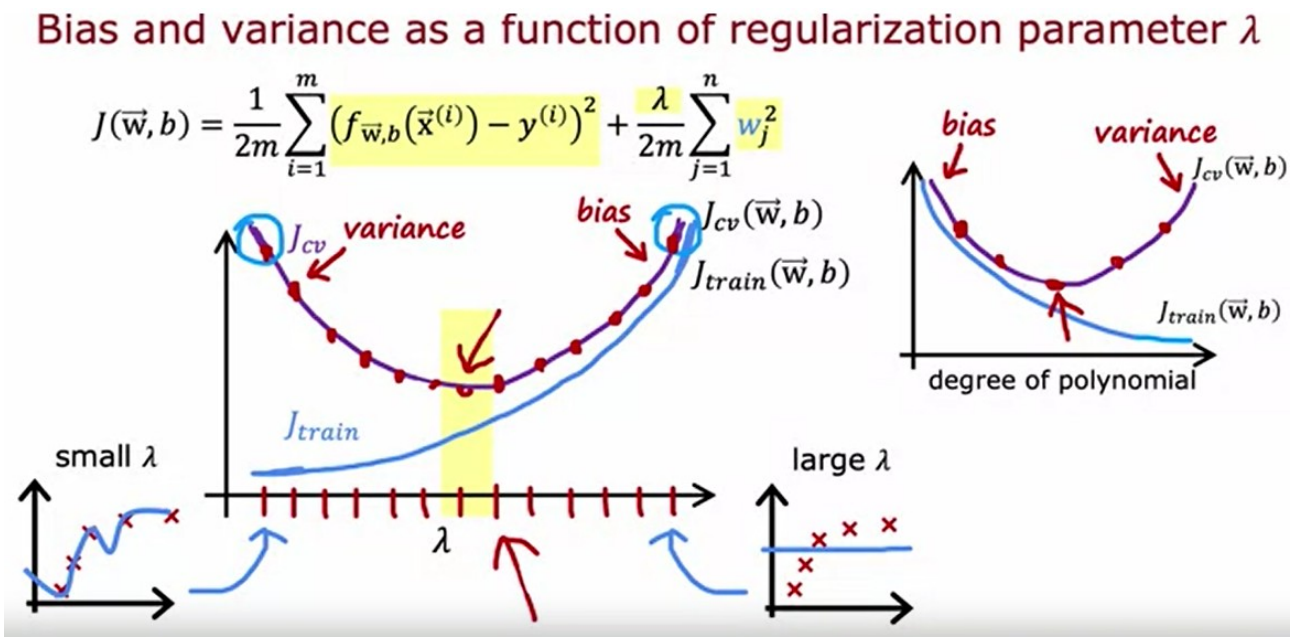


Bias,variance and regularization





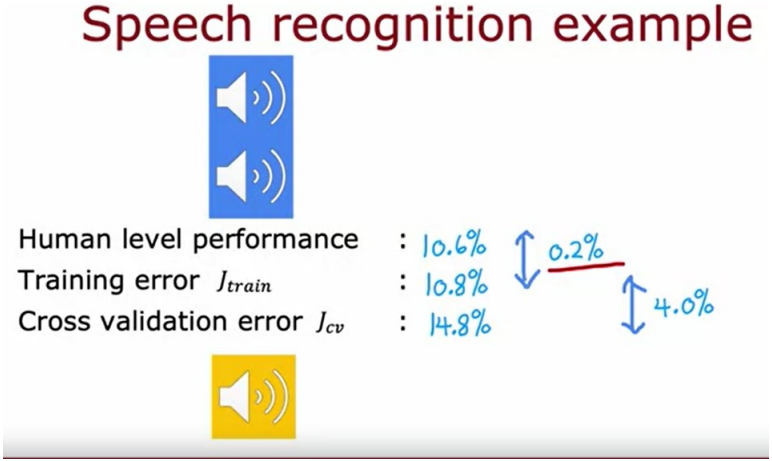
So, how does the training and cv error vary as a function of the parameter λ?



When λ is very small the model has high variance and when is big it has high bias; it looks similar to a mirrored version of the plot for the optimal degree of polynomial.

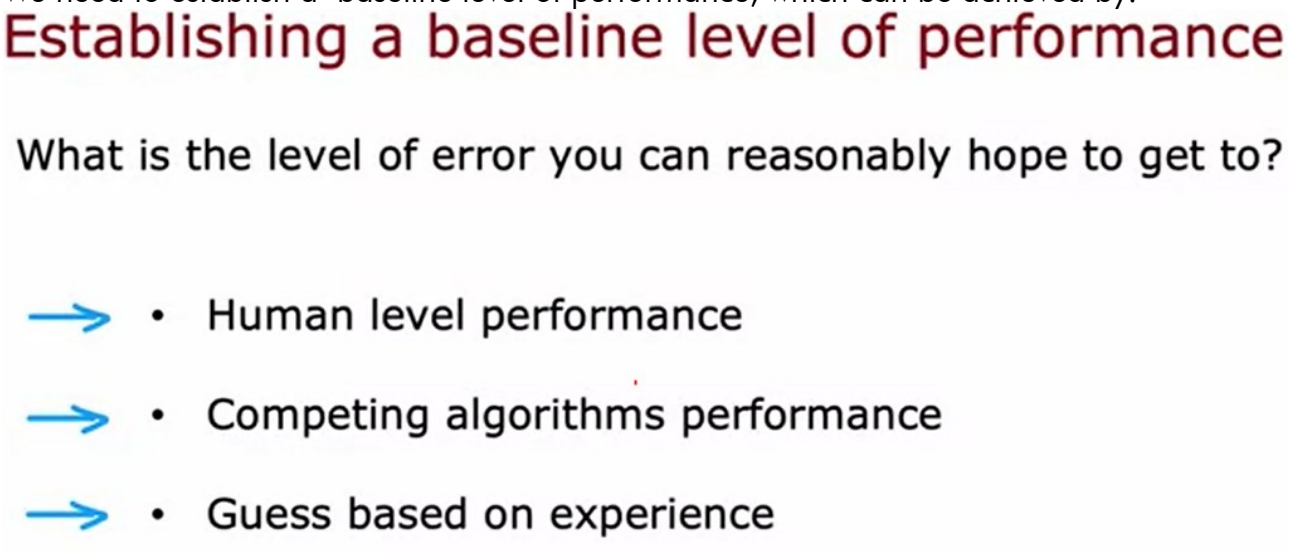
Establishing a baseline level of performance

Andrew explains how a first approach to evaluate the performance of a model is to compare its error to the one a human expert could make. For example, in speech recognition:

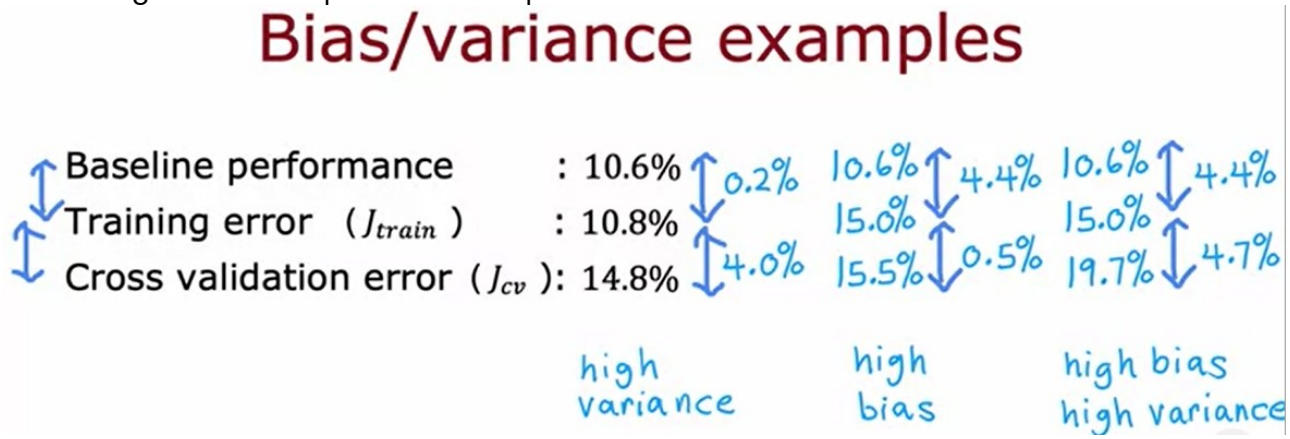


We could think that, in this example, a model with a training error of 10.8% is very high, but actually humans have an error rate of 10.6% when transcribing these messages (for example, messages with lots of noise, or cut in the middle, etc). But then, if the cv error is of 14.8%, that is actually quite high; in this case, we might have a high variance problem.

We need to establish a baseline level of performance, which can be achieved by:



So coming back to the previous example:

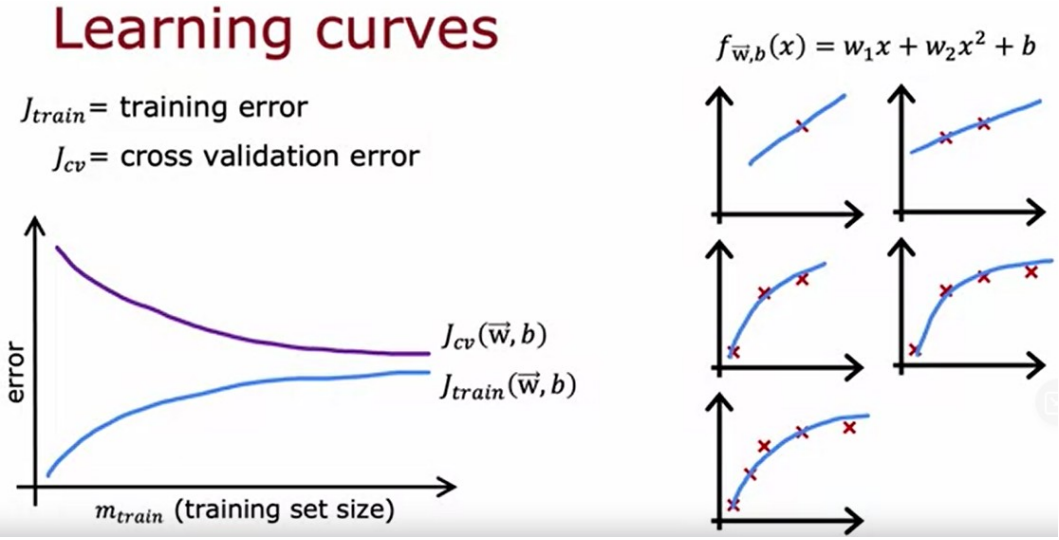


So instead of asking ourselves: is the error high? We want to be able to ask: **is the error high compared to human performance/competition results? (for example)**

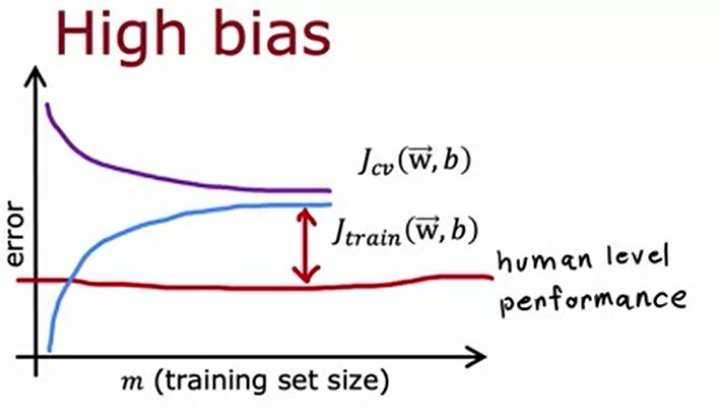
Learning curves

When you have a very small number of training examples, is relative easy to get zero or very small training error, but as the examples increase it becomes more difficult to fit all those examples, so the error will increase.

On the other side, the cv error will tipically be higher than the training error, as the parameters are expected to be at least a bit better for the training examples. When plotting this, we will have a curve similar to this one:

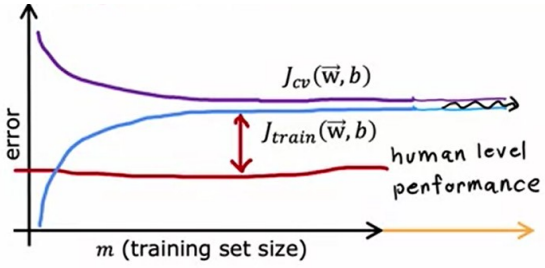


This curve looks like this for high bias:



The training error and cv error are both much higher than the expected performance, with both being quite close to each other. Generally we would need a more complex function or strategy to improve performance.

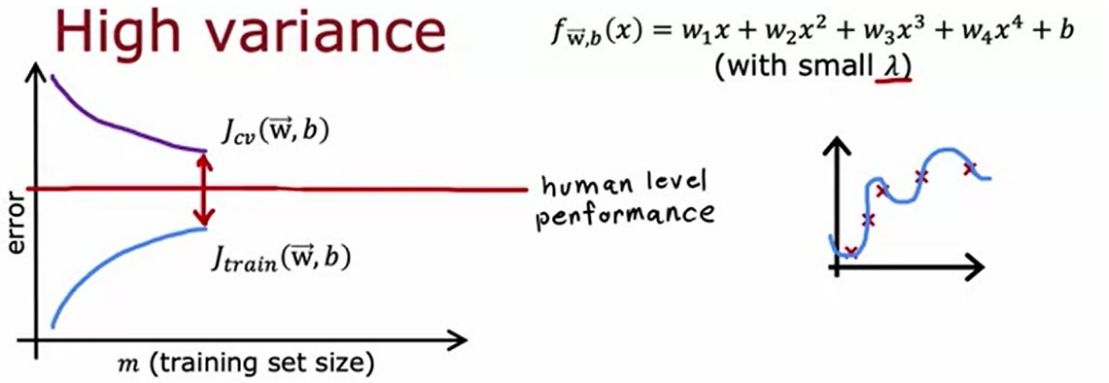
One serious mistake we can do in this case is to, in order to improve the performance of the model, increase the training size... which won't affect performance as this is what will happen:



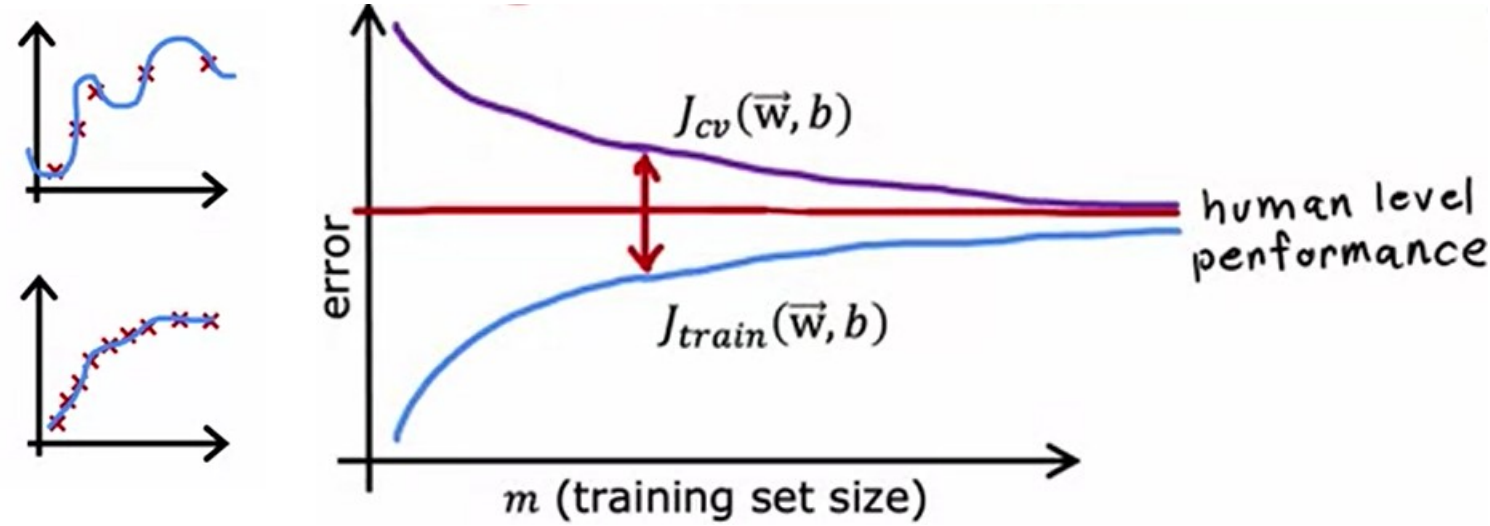
Both errors have reached a plateau and they will not just go down by increasing the examples.

If your algorithm have high bias, bringing more examples will not improve performance by itself.

What stands out in the high variance case is how the cv error will be much higher than the training error:



In the case of high variance, increasing the training set can help a lot: the training error will go up, while the cv error will hopefully go down:



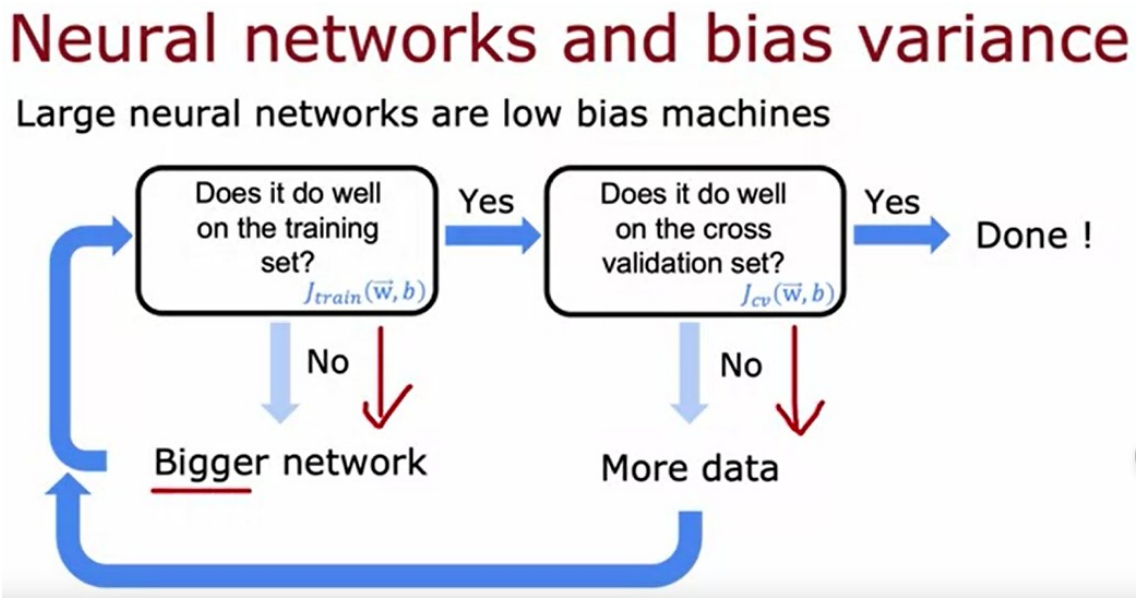
One important downside of learning curves is that they are computationally expensive, so they are not used that often; even so, Andrew encourages to keep this picture in mind to imagine what you algorithm is doing and what high bias and high variance looks like.

Bias/Variance applied to neural networks

One of the things that explains the success of neural networks is how, when they are combined with big data, high bias and variance also becomes addressed.

Large neural networks, when trained on small to moderate size dataset, are low bias machines. If you make your neural network large enough, you can almost always fit your training set well.

A general routine to address high variance and high bias in neural networks is as follows:



One problem with this approach is that as neural networks become larger, they also become computationally very expensive; the use of GPUs for speeding up the calculations has been crucial in the development of these models, but eventually neural networks can become so large that are not feasable.

Another problem is if at some point you can not get more data.

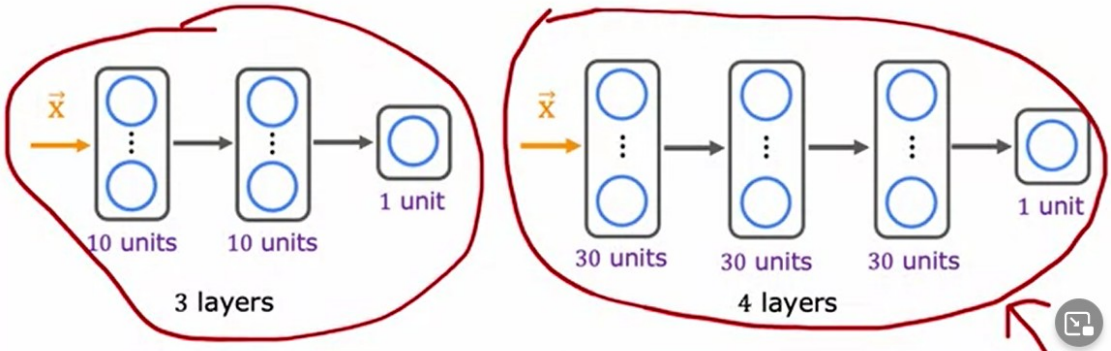
But the reason behind the rise of deep learning all these previous years is that with big enough data and computational power you can have very good performance in plenty of applications.

One possible question is: what if my neural network is too big? Will that create a high variance problem?

It turns out that a large network with well chosen regularization will usually do as well or better than a smaller one.

It almost never hurts to go to a larger neural network so long as you regularize appropiately. Although the computational cost will increase.

Neural networks and regularization



A large neural network will usually do as well or better than a smaller one so long as regularization is chosen appropriately.

Neural network regularization

$$J(\mathbf{W},\mathbf{B}) = \frac{1}{m}\sum_{i=1}^{m} L\left(f\left(\vec{x}^{(i)}\right), y^{(i)}\right) + \frac{\lambda}{2m}\sum_{\text{all weights } \mathbf{W}} \left(w^2\right)$$

b

Unregularized MNIST model

```
layer_1 = Dense(units=25, activation="relu")
layer_2 = Dense(units=15, activation="relu")
layer_3 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2, layer_3])
```

λ

Regularized MNIST model

```
layer_1 = Dense(units=25, activation="relu", kernel_regularizer=L2(0.01))
layer_2 = Dense(units=15, activation="relu", kernel_regularizer=L2(0.01))
layer_3 = Dense(units=1, activation="sigmoid", kernel_regularizer=L2(0.01))
model = Sequential([layer_1, layer_2, layer_3])
```

Error analysis

By manually examining a set of examples that the algorithm is misclassifying, it will often create inspiration about what to try next. The strategy is:

1. Get the data with errors:
    - Classification problems: Missclassified examples.
    - Regression problems: Examples with an error over a certain threshold.
2. If the error examples are too many, or you don't have enough time to go through them -> choose a random set of a size more manageable.
3. Manually go through all the examples and categorize them based on common traits.
4. Analize the groups:
    - Focus in the ones with lots of errors (don't invest time if, let's say, there is a group with 2 examples out of 1000 total error examples)
    - Take into consideration that some of these categories can be overlapping, or what is the same: not mutually exclusive.

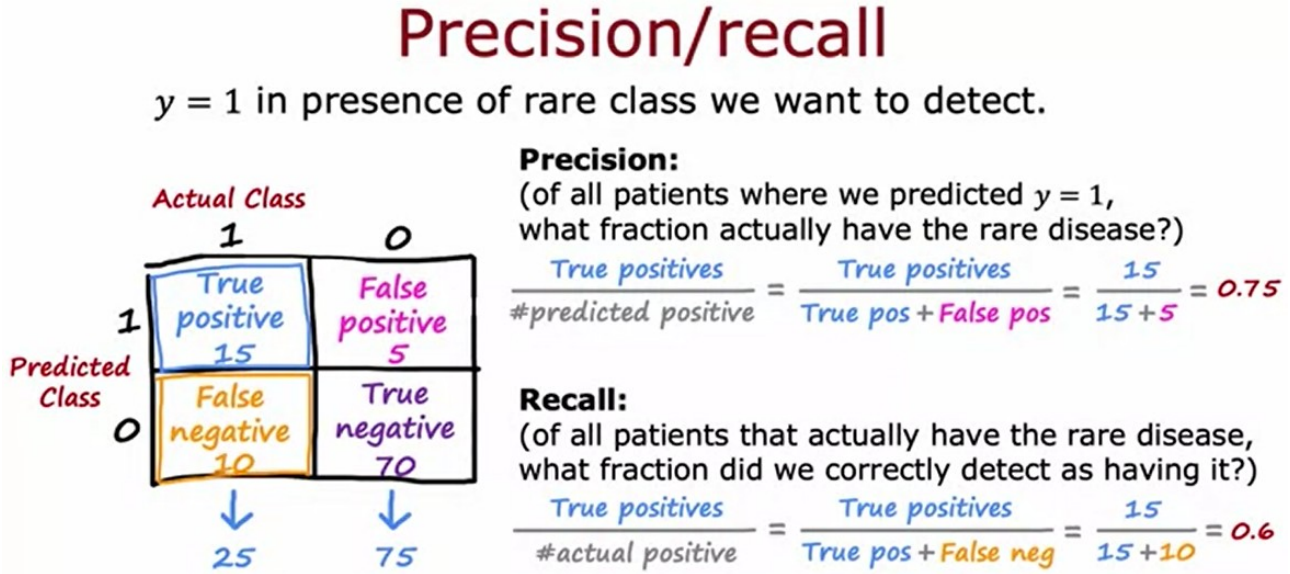
This might encourage us to:

- Look for new features.
- Look for new examples more focused in a particular trait.
- See that certain type of errors are too rare to try to fix them.

Dealing with skewed datasets

If you have in your data, for example, just 0.5% positive cases (let's say, a rare disease), then an algorithm that just predicts y = 0 always will have a 99.5% accuracy and 0.5% error. So if you have an algorithm with 1% error, or 1.2%, or 1.6%, it is very difficult to actually know which one performs better.

How to deal with this? We use precision/recall as indicators.



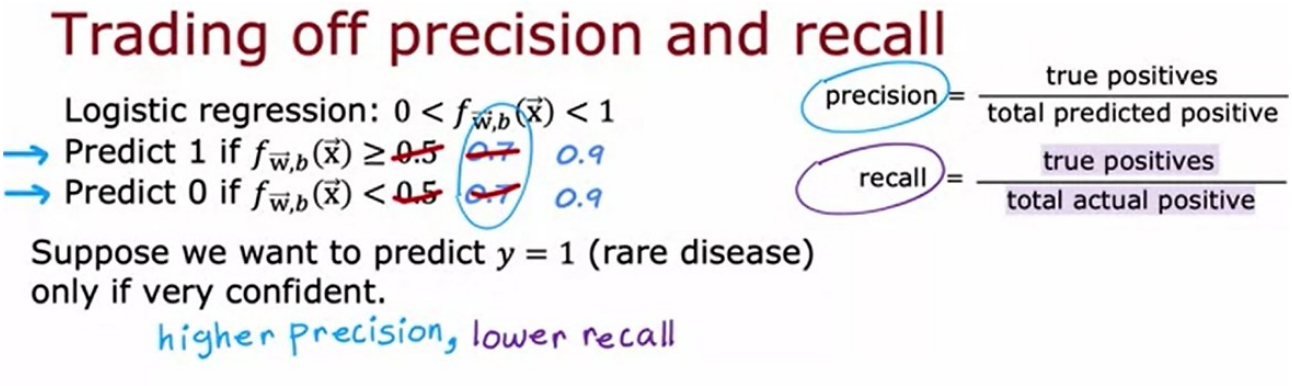
With this metrics, the algorithm that predicts y = 0 for all examples would have recall and precision 0 - confirming it as an extremely poor model.

High precision means that if it diagnoses a patient with the rare disease, probably the patient does have it and is an accurate diagnosis.

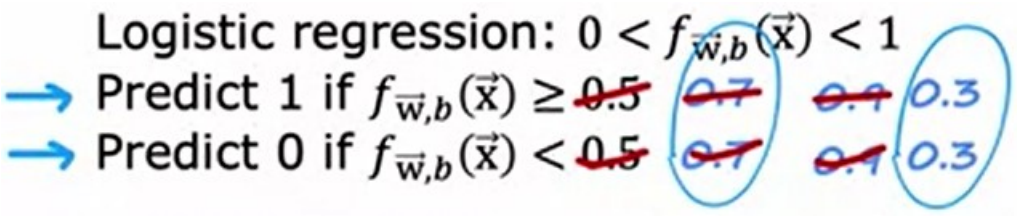
High recall means that if there is a patient with that rare disease probably the algorithm will correctly diagnose him/her.

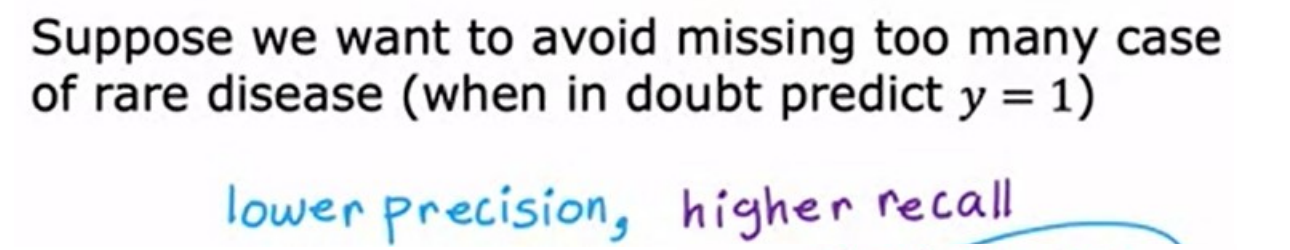
We would like an algorithm to have very high values for both, but actually there is a tradeoff between them.

If we want to predict, as the image says, y = 1 only when we are very confident (this means, we aim for high precision), increasing the threshold of the logistic regression will increase the precision. This will reduce the recall value, as we might miss more positive cases.

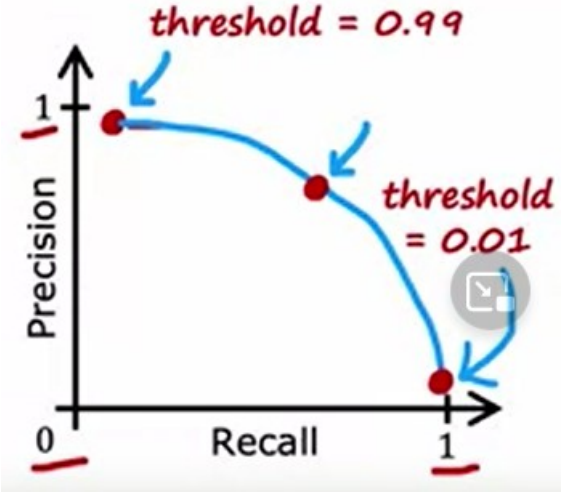


On the other hand, if we wanto to void missing any patient with the disease, we can do the opposite: reduce the threshold. This will increase the recall while reducing the precision:

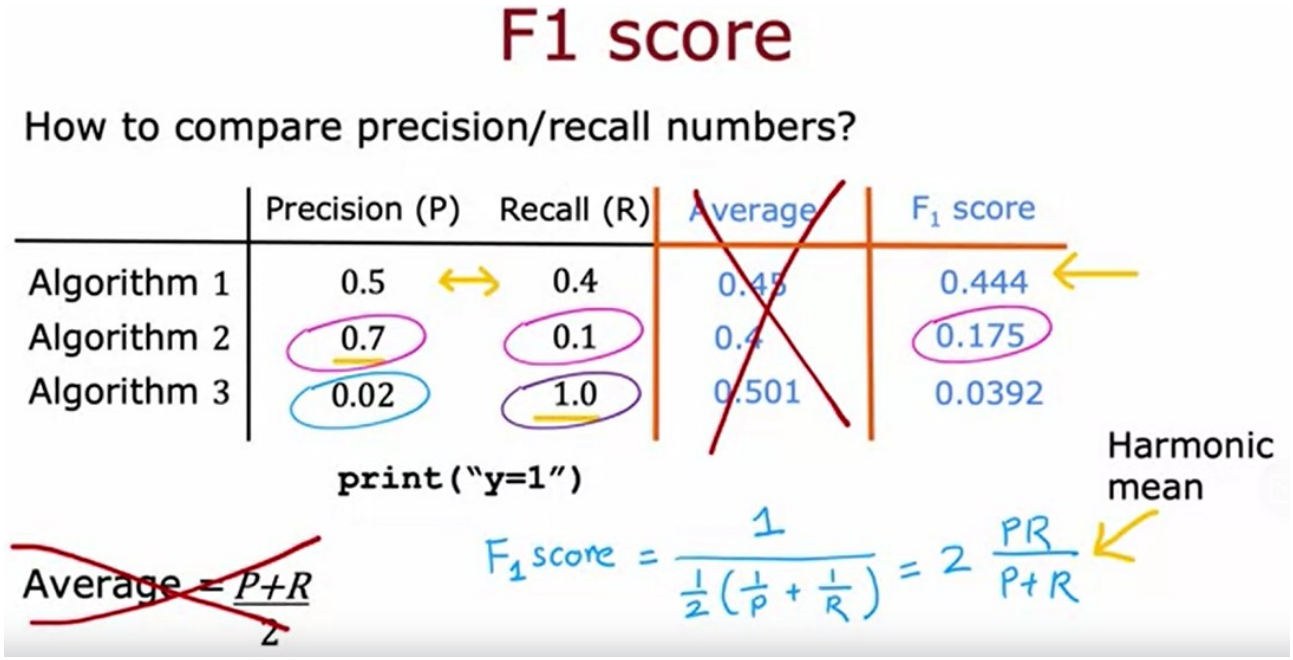




This is how the tradeoff looks depending on the threshold of the classification algorithm:



To choose an algorithm depending on these 2 values can be tricky; to simplify, we could do the average, but that doesn't work well: we can use the F1 score:



The F1 score is also called Harmonic mean, which is a way of calculating an average that emphasizes the smaller values more. We choose the highest F1 score, which is the one with the smaller tradeoff.