Interview Question for Machine Learning

#### **1.1 - What are parametric models? Give an example.**

*Parametric* models are those with a finite number of parameters. To predict new data, you only need to know the parameters of the model. Examples include linear regression, logistic regression, and linear SVMs.

*Non-parametric* models are those with an unbounded number of parameters, allowing for more flexibility. To predict new data, you need to know the parameters of the model and the state of the data that has been observed. Examples include decision trees, k-nearest neighbors, and topic models using latent dirichlet analysis.

In short, think of it this way. For a parametric model to predict new data, knowing just the parameters is enough (think of linear regression based on a set of parameters). For a non parametric model, predicting future data is based on not just the parameters but also in the current state of data that has been observed (think of topic modelling that is based on latent dirichlet distributions).

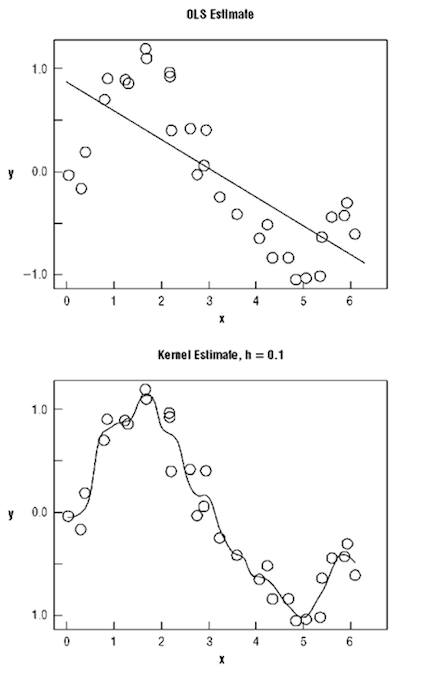
**Parametric Model:** yi=β0+β1xi+eiyi=β0+β1xi+ei

Here you know what the regression will look like: a linear line.

**Non-Parametric Model:** yi=f(xi)+eiyi=f(xi)+ei where f(.) can be any function.

A parametric model is one where we assume the ‘shape’ of the data, and therefore only have to estimate the coefficients of the model.

A non-parametric model is one where we do not assume the ‘shape’ of the data, and we have to estimate the most suitable form of the model, along with the coefficients.



<https://www.quora.com/What-is-the-difference-between-a-parametric-model-and-a-non-parametric-model>

#### **1.2 - What is the "Curse of Dimensionality?**

### **The Curse of Dimensionality**

In machine learning, “dimensionality” simply refers to the number of features (i.e. input variables) in your dataset.

When the number of features is very large relative to the number of observations in your dataset, *certain* algorithms struggle to train effective models. This is called the “Curse of Dimensionality,” and it’s especially relevant for [clustering](http://elitedatascience.com/machine-learning-algorithms#clustering) algorithms that rely on distance calculations.

<https://elitedatascience.com/dimensionality-reduction-algorithms>

#### **1.3 - Explain the Bias-Variance Tradeoff.**

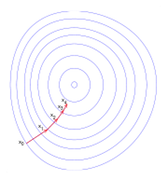
Predictive models have a tradeoff between bias (how well the model fits the data) and variance (how much the model changes based on changes in the inputs).

*Simpler models* are stable (low variance) but they don't get close to the truth (high bias).

More *complex models* are more prone to being overfit (high variance) but they are expressive enough to get close to the truth (low bias).

The best model for a given problem usually lies somewhere in the middle.

* [Learn more about the Bias-Variance Tradeoff](http://elitedatascience.com/bias-variance-tradeoff)

[](http://elitedatascience.com/bias-variance-tradeoff)

## **There are 3 types of prediction error: bias, variance, and irreducible error.**

## Irreducible error is also known as "noise," and it can't be reduced by your choice in algorithm. It typically comes from inherent randomness, a mis-framed problem, or an incomplete feature set.

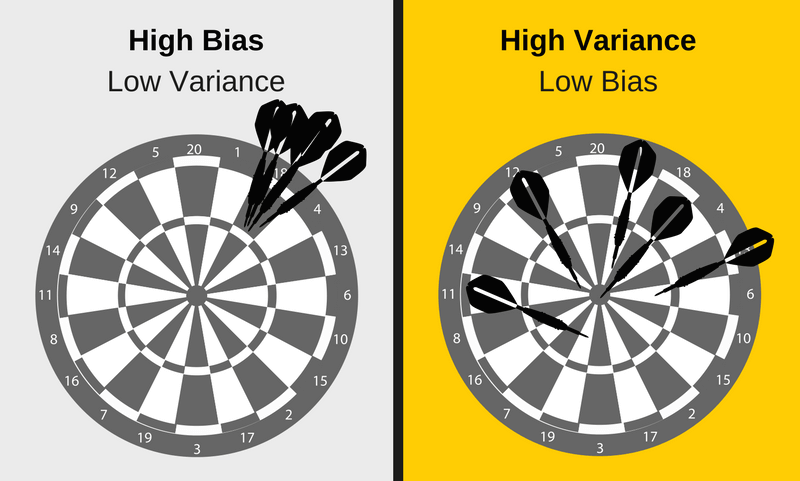
## **Error from Bias → known as under-fitting.**

Bias is the difference between your model's expected predictions and the true values.

## **Error from Variance→ known as over-fitting.**

Variance refers to your algorithm's sensitivity to specific sets of training data.

## **The Bias-Variance Tradeoff**



Low variance (high bias) algorithms tend to be **less complex**, with simple or rigid underlying

This **tradeoff in complexity** is why there's a tradeoff in bias and variance - an algorithm cannot simultaneously be more complex and less complex.

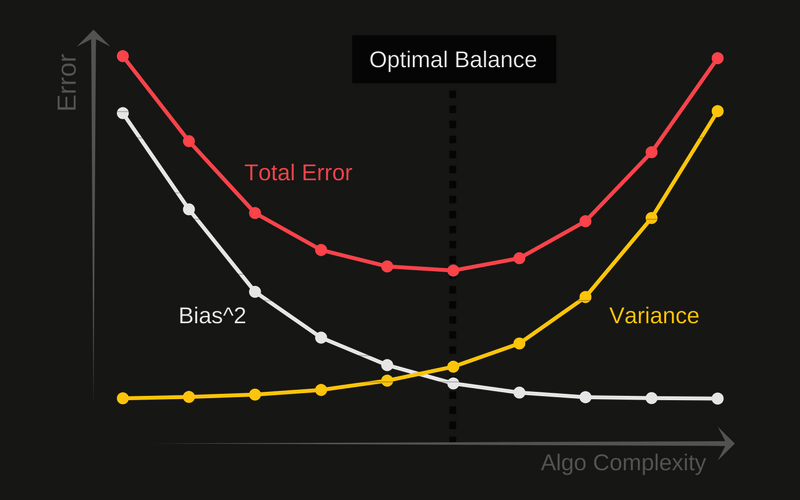
*\*Note: For certain problems, it's possible for some algorithms to have less of both errors than others. For example, ensemble methods (i.e. Random Forests) often perform better than other algorithms in practice. Our recommendation is to always try multiple reasonable algorithms for each problem.*

## **Total Error**

To build a good predictive model, you'll need to find a balance between bias and variance that minimizes the total error.

**Total Error = Bias^2 + Variance + Irreducible Error**

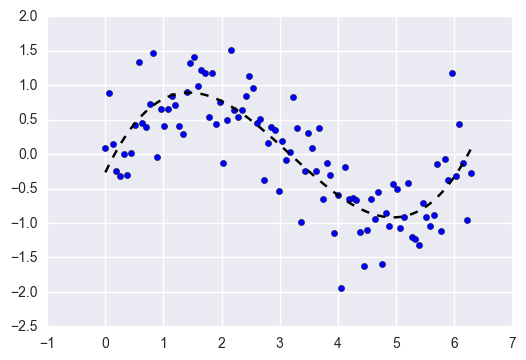
Machine learning processes find that optimal balance:



A proper machine learning workflow includes:

* Separate training and test sets
* Trying appropriate algorithms ([No Free Lunch](http://elitedatascience.com/machine-learning-algorithms))
* Fitting model parameters
* Tuning impactful hyperparameters
* Proper performance metrics
* Systematic cross-validation

Finally, as you might have already concluded, an optimal balance of bias and variance leads to a model that is neither overfit nor underfit:



This is the ultimate goal of supervised machine learning - to isolate the **signal** from the dataset while ignoring the noise!

## **2. Optimization**

*Algorithms for finding the best parameters for a model.*

#### ***2.1 - What is the difference between stochastic gradient descent (SGD) and gradient descent (GD)?***

Both algorithms are methods for finding a set of parameters that minimize a loss function by evaluating parameters against data and then making adjustments.

In standard gradient descent, you'll evaluate all training samples for each set of parameters. This is akin to taking big, slow steps toward the solution.

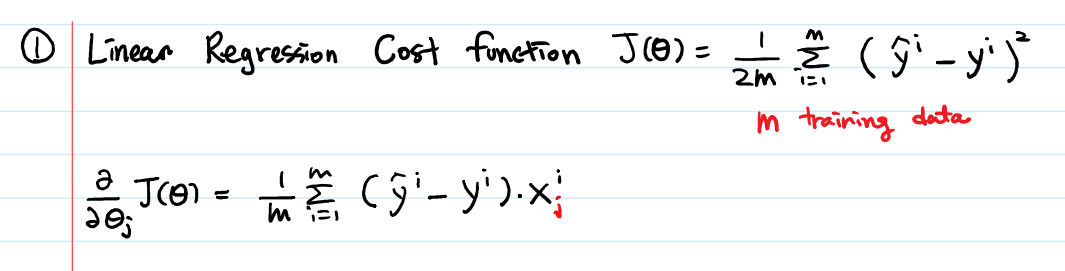
In stochastic gradient descent, you'll evaluate only 1 training sample for the set of parameters before updating them. This is akin to taking small, quick steps toward the solution.

# ***Difference between Batch Gradient Descent and Stochastic Gradient Descent***

## *[WARNING: TOO EASY!]*

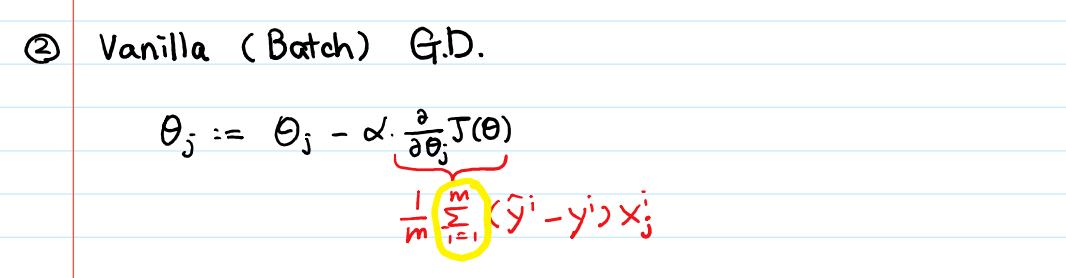
*Let’s take the simplest example, which is Linear Regression.*

*As always, we start with the cost function.*

**

*Linear Regression Recap done.*

*Now, what was the Gradient Descent algorithm?*

**

*Above algorithm says,* ***to perform the GD****, we need to calculate the gradient of the cost function J. And* ***to calculate the gradient of the cost function****, we need to sum (****yellow circle!****) the cost of each sample. If we have 3 million samples, we have to loop through 3 million times or use the dot product.*

*Here is Python code:*

*def gradientDescent(X, y, theta, alpha, num\_iters):*

*"""*

*Performs gradient descent to learn theta*

*"""*

*m = y.size # number of training examples*

*for i in range(num\_iters):*

*y\_hat = np.dot(X, theta)*

*theta = theta - alpha \* (1.0/m) \** ***np.dot(X.T, y\_hat-y)***

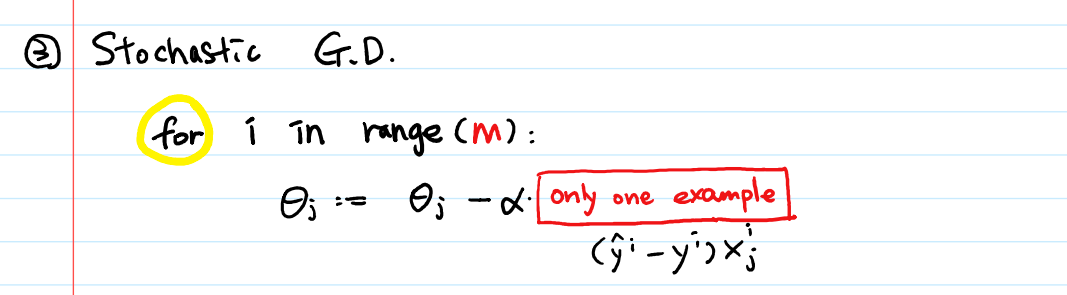
*return theta*

*Do you see* ***np.dot(X.T, y\_hat-y)*** *above? That’s the* ***vectorized version of “looping through (summing) 3 million samples”****.*

*Wait.. just to move a single step towards the minimum, do we really have to calculate each cost 3 million times?*

*Yes. If you insist to use GD.*

*But if you use Stochastic GD, you don’t have to!*

**

*Basically, in SGD, we are using the cost gradient of* ***1******example*** *at each iteration, instead of using the sum of the cost gradient of* ***ALL*** *examples.*

*def SGD(f, theta0, alpha, num\_iters):*

*"""*

*Arguments:*

*f -- the function to optimize, it takes a single argument*

*and yield two outputs, a cost and the gradient*

*with respect to the arguments*

*theta0 -- the initial point to start SGD from*

*num\_iters -- total iterations to run SGD for*

*Return:*

*theta -- the parameter value after SGD finishes*

*"""*

*start\_iter = 0*

*theta= theta0*

*for iter in xrange(start\_iter + 1, num\_iters + 1):*

*\_, grad = f(theta)*

*theta = theta - (alpha \* grad)* ***# there is NO dot product!***

*return theta*

*Well, Stochastic Gradient Descent has a fancy name but I guess it’s a pretty simple algorithm!*

*Few things to note:*

*a) In SGD, before for-looping, you need to randomly shuffle the training examples.*

*b) In SGD, because it’s using only one example at a time, its path to the minima is noisier (more random) than that of the batch gradient. But it’s ok as we are indifferent to the path, as long as it gives us the minimum AND the shorter training time.*

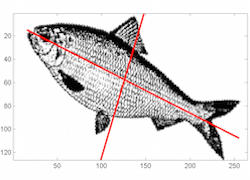
*c) Mini-batch gradient descent uses* ***n*** *data points (instead of* ***1*** *sample in SGD) at each iteration.*

#### ***2.2 - When would you use GD over SDG, and vice-versa?***

*GD theoretically minimizes the error function better than SGD. However, SGD converges much faster once the dataset becomes large.*

*That means GD is preferable for small datasets while SGD is preferable for larger ones.*

*In practice, however, SGD is used for most applications because it minimizes the error function well enough while being much faster and more memory efficient for large datasets.*

**

## ***3. Data Preprocessing***