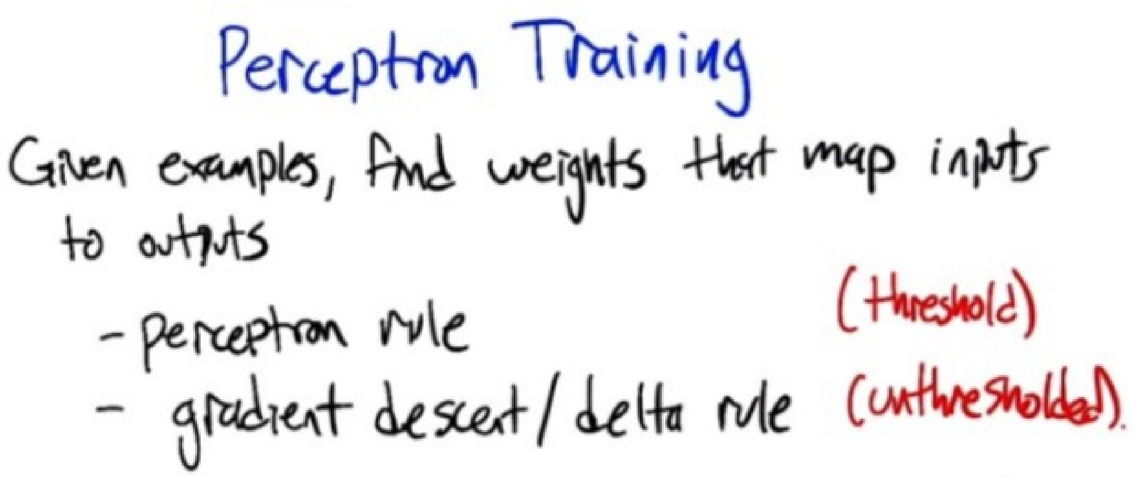
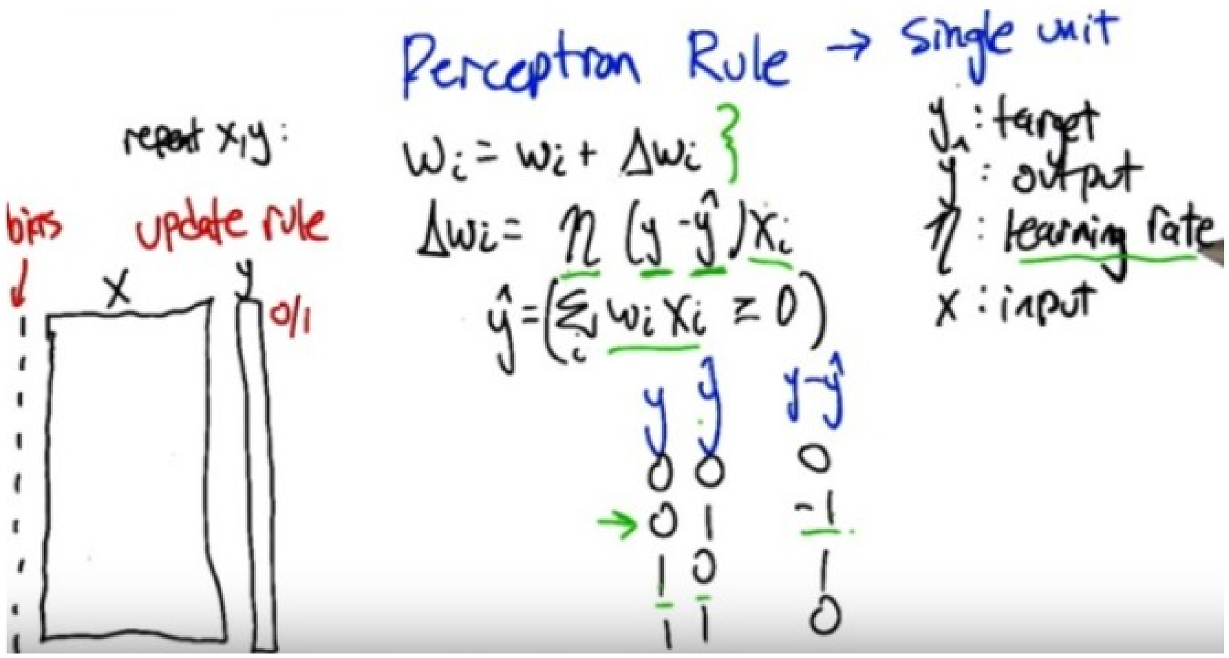
1. Last time we talked about metrics to evaluate predictive models (比如一個 decision tree 就是一個 model). Now that we know what we're looking for in our models, let's talk about how we actually develop those models using big data. We'll start by talking about some of the algorithms used in big data analytics like stochastic gradient descent. We'll then talk about ensemble method. A powerful class of machinery algorithms that often lead to the best performing models. In particular, we'll talk about two criteria, bias and variance, and the trade off between them. And finally, we'll talk about bagging and boosting, two ensemble methods.



Perceptron Training

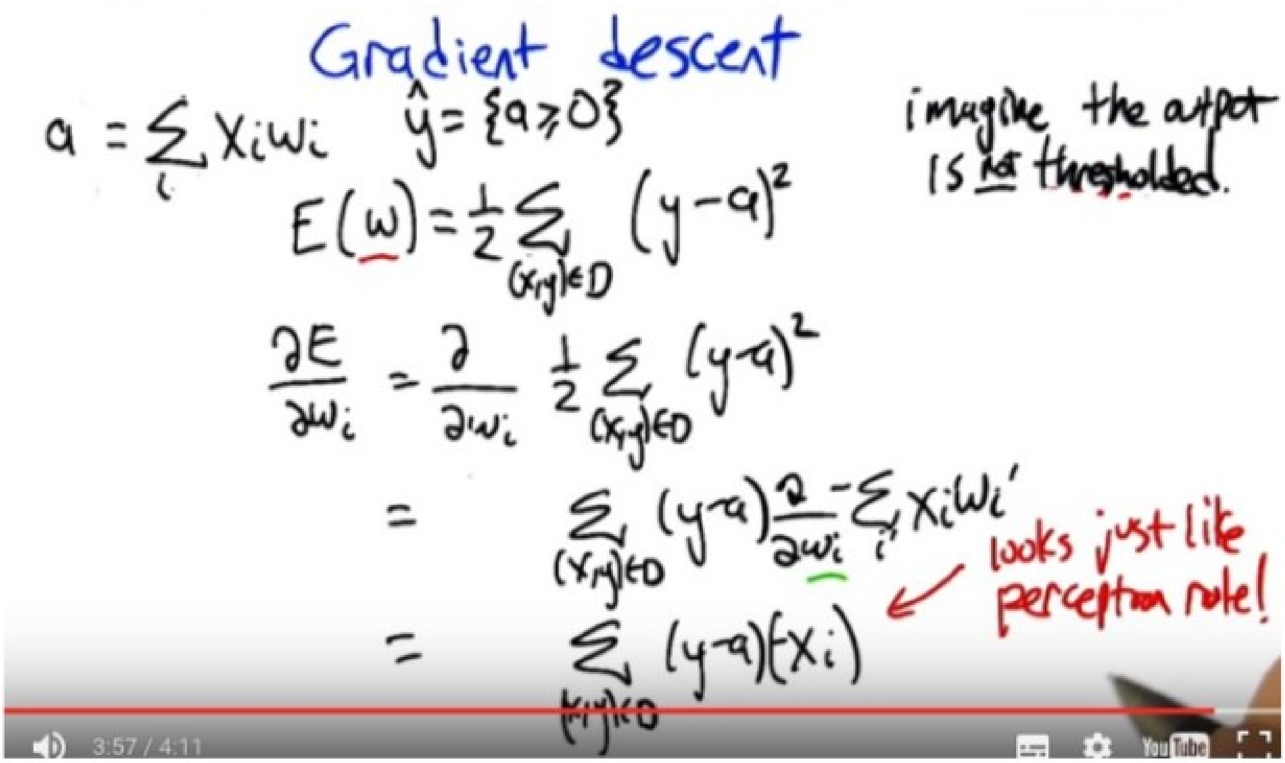
Given examples, find weights that map inputs to outputs

- perceptron rule (threshold)
- gradient descent / delta rule (unthresholded)



Perceptron Rule → single unit
repeat x,y:
bias
update rule
x
y
0/1
$w_i := w_i + \Delta w_i$
$\Delta w_i = \eta (y - \hat{y}) x_i$
$\hat{y} = (\sum_i w_i x_i \geq 0)$
y $\hat{y}$ $y-\hat{y}$
0 0 0
→ 0 1 -1
1 0 1
1 1 0
y : target
$\hat{y}$: output
η : learning rate
x : input

上圖來自 ML 課 note 的 SL3(neural network)中



Gradient descent

$a = \sum_i x_i w_i \qquad \hat{y} = \{a \geq 0\}$

imagine the output is not thresholded.

$$E(\underline{w}) = \frac{1}{2}\sum_{(x,y)\in D}(y-a)^2$$

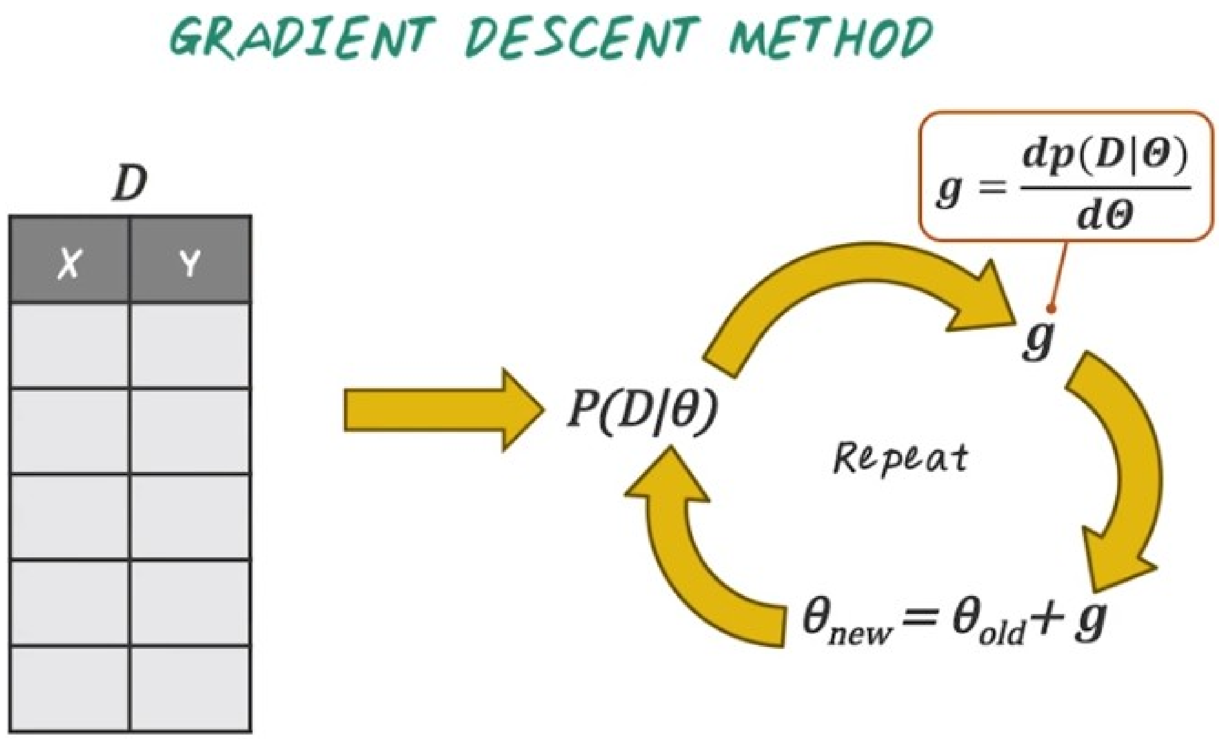
$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\,\frac{1}{2}\sum_{(x,y)\in D}(y-a)^2$$

$$= \sum_{(x,y)\in D}(y-a)\frac{\partial}{\partial w_i} - \sum_{i'} x_{i'} w_{i'}$$

$$= \sum_{(x,y)\in D}(y-a)(-x_i)$$

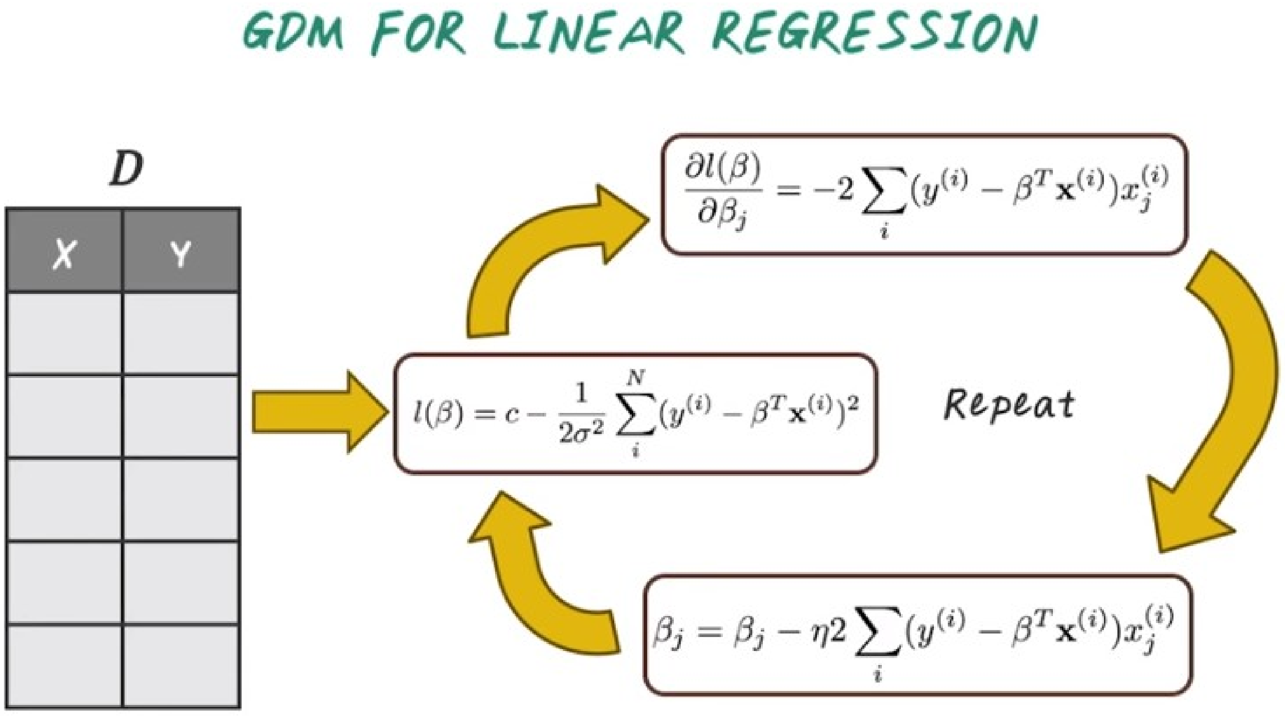
looks just like perceptron rule!

上圖來自 ML 課 note 的 SL3(neural network)中


GRADIENT DESCENT METHOD
D
X
Y
$P(D|\theta)$
$g = \frac{dp(D|\Theta)}{d\Theta}$
g
Repeat
$\theta_{new} = \theta_{old} + g$

上圖來自本課

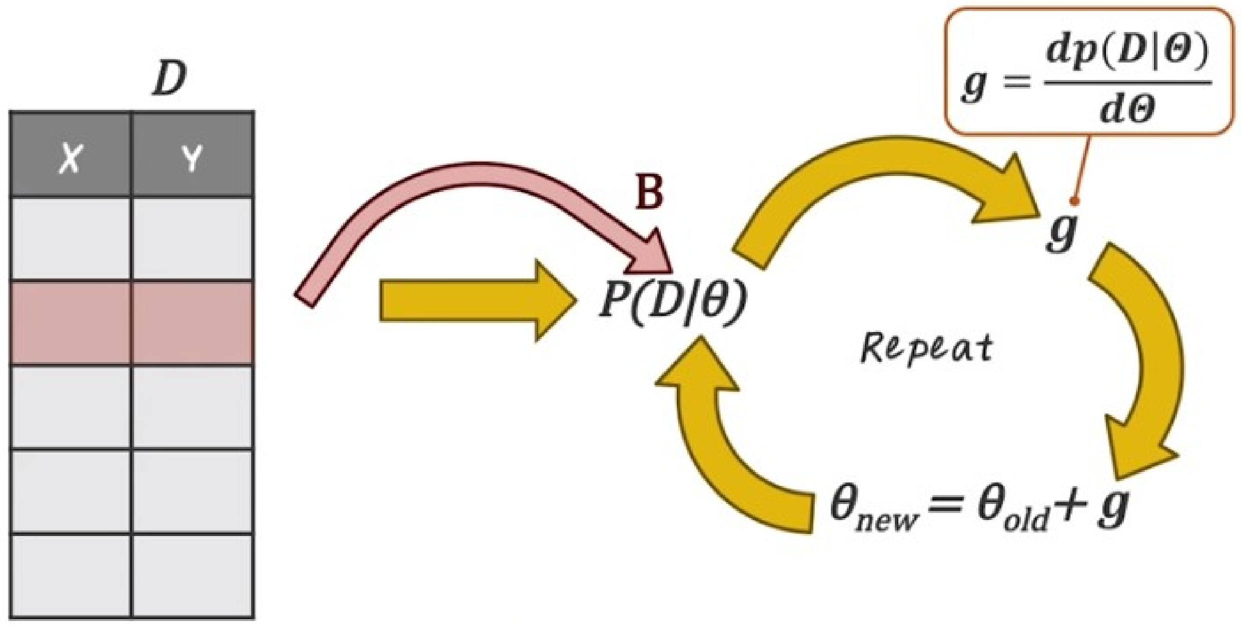
2. Now lets start with gradient descent method for classification. Gradient descent is a basic optimization method that has been widely used in machine learning and data mining applications. Next, lets go to the high level procedure about how to apply gradient descent method for classification and regression problem. So remember, the input to any classification or regression problem is a training data set consisting of n pairs of data points, x, and the corresponding target, y. And the final output of the model is usually involve some parameter theta. For example, the theta in linear regression are the linear coefficient beta. (以下的步驟會在下段中給出俱體例子) And the first step of gradient descent is to specify the likelihood function of input data D given parameter theta. The likelihood function is really the joint distribution of the data points, given the model parameter theta. In these steps, sometimes it's easier to use log likelihood function instead of likelihood function, since log transformation is monotonically increasing and leads to the same optimal as the original likelihood function. But the resulting terms are often a lot easier to manipulate. After we specify the likelihood function, next, we want to find the derivative, also called gradient, of the likelihood function, given theta. This step is a crucial step. Most of the computation really happens in this step, and depending on how likelihood function is specified, finding the derivative can be easy or can be very hard. Once we completed gradient, we'll update the parameter theta by moving the old parameter towards the direction of the gradient. Finally, we repeat this process until it converges which means the theta is no longer changing. And this process illustrates the high level idea of gradient descent. But the algorithm actually requires some additional tuning parameters. For example, the learning rate or the step size (見下段), which controls how far we update the theta based on the gradient. And the step size can be learned through cross-validation. And also note that gradient descent is the simplest gradient based optimization algorithm. There are many other algorithms, are more advanced but still based on gradient computation, such as conjugate gradient, and ~powtzei-Newton method. So in fact, as long as you can specify the formula for computing the gradient, and you can use more advanced optimization method as black box by just simply plugging those gradient computation as a function.


GDM FOR LINEAR REGRESSION
D
X
Y
$l(\beta) = c - \frac{1}{2\sigma^2}\sum_i^N (y^{(i)} - \beta^T \mathbf{x}^{(i)})^2$
$\frac{\partial l(\beta)}{\partial \beta_j} = -2\sum_i (y^{(i)} - \beta^T \mathbf{x}^{(i)}) x_j^{(i)}$
$\beta_j = \beta_j - \eta 2 \sum_i (y^{(i)} - \beta^T \mathbf{x}^{(i)}) x_j^{(i)}$
Repeat

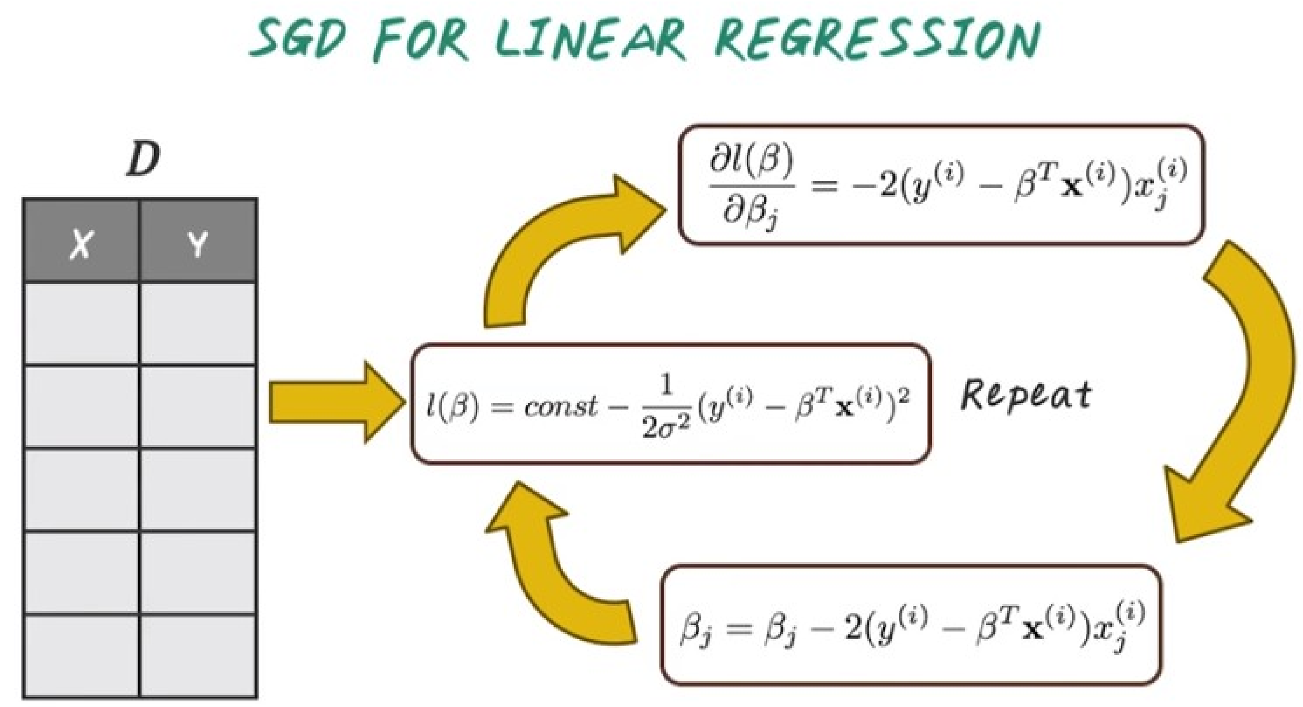
GDM: gradient descent method

3. Now let's illustrate the gradient descent method using linear regression. Imagine we're giving a data set D, every row corresponding to a patient and every patient is characterized by a set of features, specified in X and we also have an outcome variable Y. For example, in this case, it could be the cost of this patient, and the goal of the model is to map the input feature X from the patient to the output cost Y that this patient incurs. So first, we want to specify the log likely to function for linear regression. In this case, linear regression assumes Gaussian distribution. If you carry out all the calculation based on Gaussian probabilistic distribution, then you will see this log likelihood function involves some constants, subtract a sum over a set of squared terms. And these summation terms is the sum over all the patients. And for each patient, we find the true cost this patient i incurred (y^(i)), subtract what the model predicts (β^T x^(i)). Here this beta are the coefficient from this regression model, and x^(i) are all the features from patient i. So the intuition is, we want to learn a linear model that convert the input features such as age, gender, demographics, from this patient to the cost this patient incurred at the hospital. Once we have the log likelihood, we can take the derivative of this log likelihood to find the gradient, which is specified over here. This notation means partial derivative of beta on it's base element. The entire gradient is also a vector over all the beta j's. So the intuition is, we have a set of parameters, one for each feature and we want to create a gradient for each one of those parameters as well. If we look closely at this gridding computation, we notice that it involve a summation over all the patients. And then for each patient, we compute this linear term and scale that by a scaler. So the intuition is if we want to compute the gradient of a specific feature, for example age, want to know how important is age in this linear regression model, then we have to go through the entire data set to compute this sum and also for each of this term with scaled by the H feature for a specific patient. Once we have this gradient, then we can just update each beta coefficient by moving towards this grading direction. Here, we added the step size, eta, as additional parameter so that you realize, in the real implementation, you have to specify the step size as well. Then again, this process has to be repeated many times until all the beta are converged. So note that the gridding computation involve going through the entire data set and we have compute this gridding many many times. So this algorithm can be very expensive given a large data set. So next we'll see how we can address that challenge using a slightly different method.

STOCHASTIC GRADIENT DESCENT (SGD) METHOD


D
X
Y
B
$P(D|\theta)$
$g = \frac{dp(D|\Theta)}{d\Theta}$
g
Repeat
$\theta_{new} = \theta_{old} + g$

4. So, Stochastic Gradient Descent, or SGD Method, is a variant of gradient descent for handling big data set. So, in traditional Gradient Descent Method, we have to compute the likelihood of the entire data set, then compute the gradient of the entire data set, then repeat, this, many, many times. Which can be too expensive to do on a large data set. The idea of SGD is actually quite simple. Instead of computing the likelihood over the entire data set, the idea is to compute the likelihood function and the gradient on a random subset of data points. For example, B data points. Sometimes, SGD referring specifically to when we update one data point at a time, while for larger B it referred as a mini batch update. But the intuition are the same. We take a subset of data points, compute the likelihood on that subset, then compute the gradient on that subset, then perform that date and repeat. And sample another set of random data points then repeat this process. So compared to traditional gradient descent, SGD method can compute this update much more quickly.

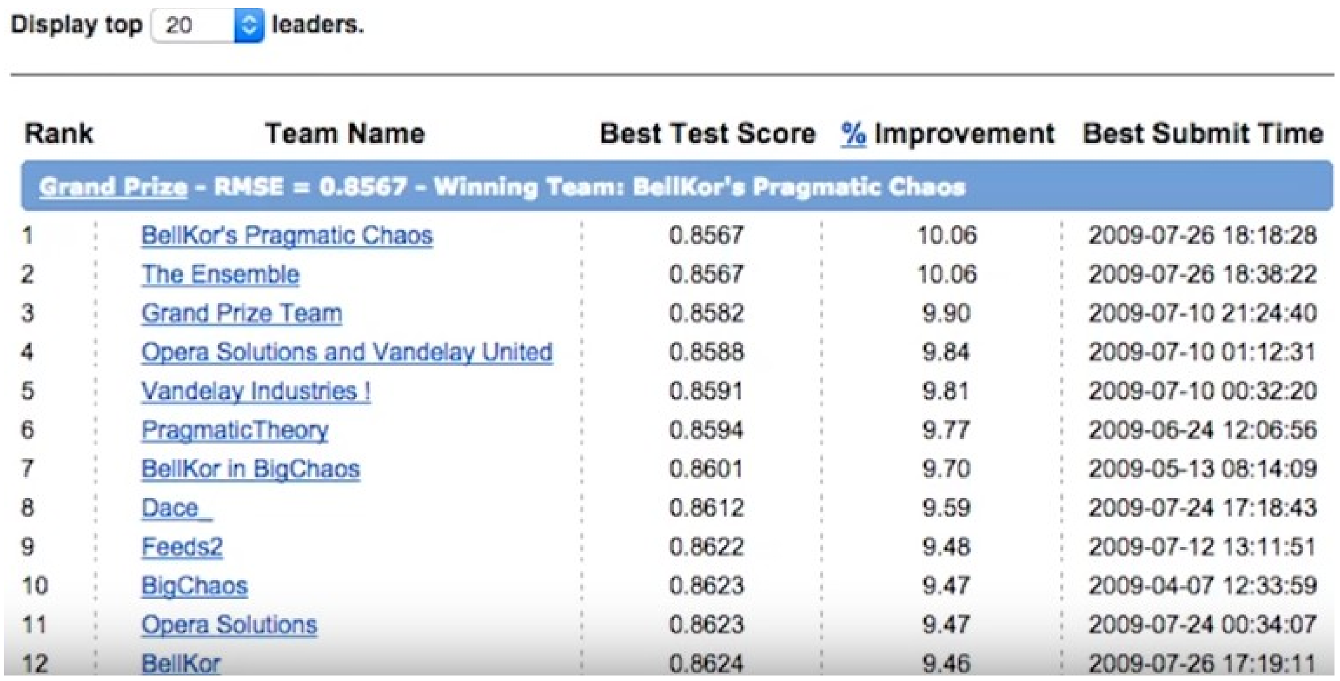

SGD FOR LINEAR REGRESSION
D
X
Y
$\frac{\partial l(\beta)}{\partial \beta_j} = -2(y^{(i)} - \beta^T \mathbf{x}^{(i)})x_j^{(i)}$
$l(\beta) = const - \frac{1}{2\sigma^2}(y^{(i)} - \beta^T \mathbf{x}^{(i)})^2$
Repeat
$\beta_j = \beta_j - 2(y^{(i)} - \beta^T \mathbf{x}^{(i)})x_j^{(i)}$

上圖中的三個式子跟前面比, 都沒了 Σ_i

5. Next, let's see how we can run stochastic gradient descent for linear regression. Here, we're using the same dataset as before. Every row is a patient, and we have a set of features associated with each patient, such as age, gender, what disease they have and what medication they're taking and the goal is to predict a target Y, which is the cost they will incur at the hospital. So next, if we want to compute SGD on this data set, we randomly sample one patient and compute a log likelihood (即中間那個 l(β)式子) of this patient. If we look closely at this log likelihood term, we notice that this only involve a single patient, patient i. And intuition here is, it measures how well the model performed for this specific patient. Then we compute the gradient on this patient and the gradient in this case is this linear term, scaled by this corresponding feature j. Then we can perform the update just like before. Using the gradient, and then repeat the process by sampling another patient. Then go through this update. So this process is very similar to gradient descent for linear regression. So the main difference is, when we compute the log likelihood and the gradient, only single patient is involved. We don't have to do this for the entire dataset. And this is much more efficient.

ENSEMBLE METHOD

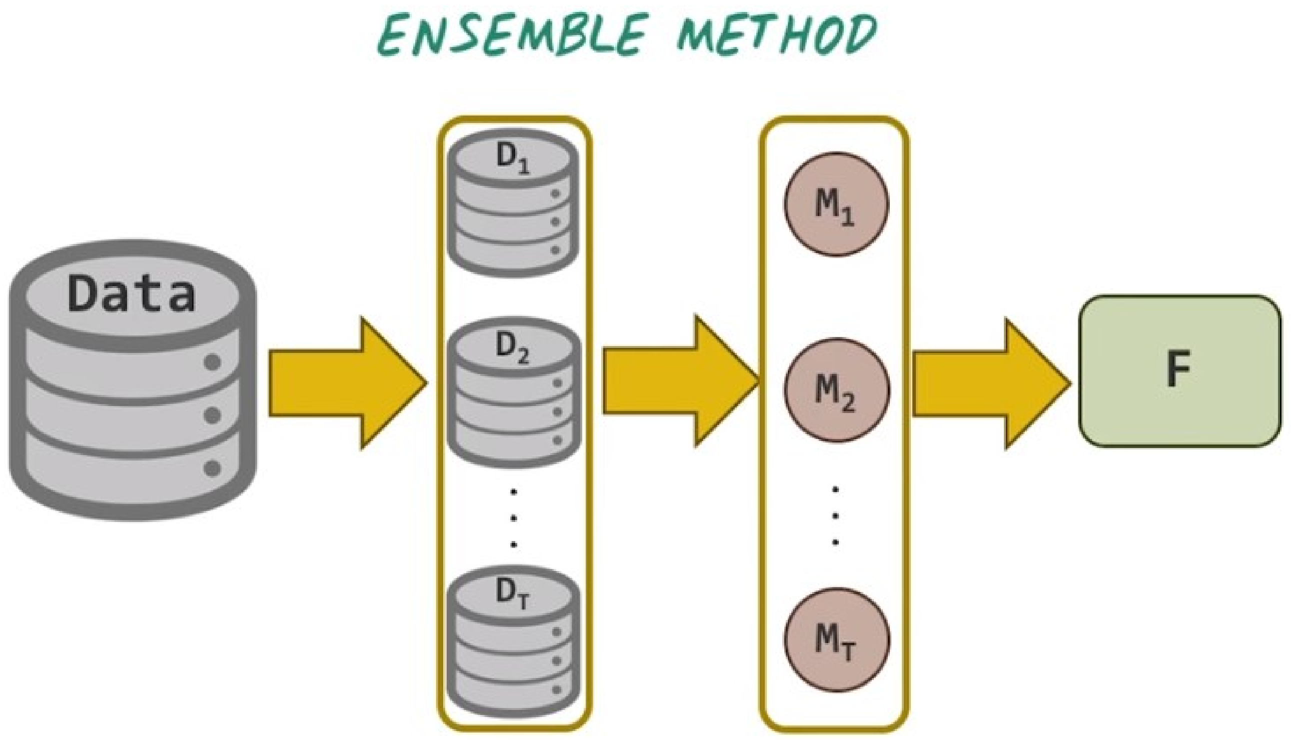
6. So far, we've talked about stochastic gradient descent as a efficient method for dealing with large dataset. Next, we introduce ensemble method which is another popular method for classification. Ensemble method combines multiple models to form a better model. Traditionally ensemble method refers to combining models using the same base learning algorithms. Such as random forest. However, more generally it can also refer to combining models using different algorithms.



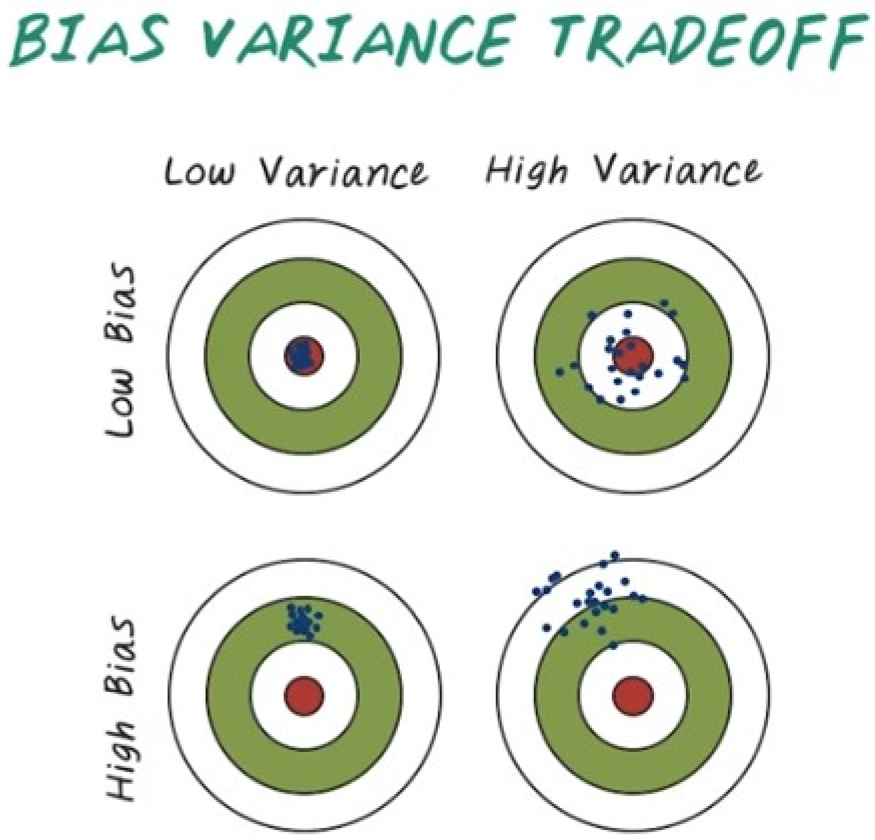
Display top 20 leaders.

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace_	0.8612	9.59	2009-07-24 17:18:43
9	Feeds2	0.8622	9.48	2009-07-12 13:11:51
10	BigChaos	0.8623	9.47	2009-04-07 12:33:59
11	Opera Solutions	0.8623	9.47	2009-07-24 00:34:07
12	BellKor	0.8624	9.46	2009-07-26 17:19:11

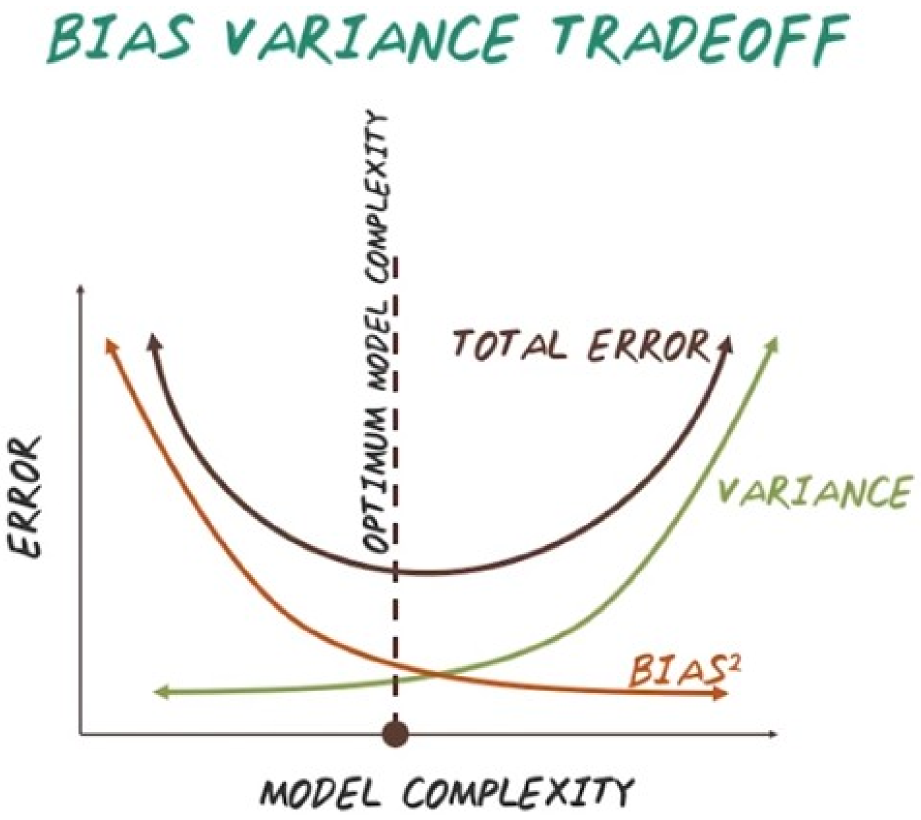
Ensemble model often outperform other classification method. For example, Netflix price an open competition for the best recommendation algorithm to predict user's rating for movies, Netflix provided a training data set over 100 million ratings that close to half a million user give to 17,000 movies. The goal is to improve 10% in root mean square errors, or RMSE, over the existing Netflix internal algorithm. The competition began on October 2006, and ended on September, 2009. Which lasted almost three years. And here are the leaderboards. Notice that the top two teams had the same exact performance gain over the baseline method. And the number one team won simply because they submit earlier. In fact, all the top ranked teams are based on ensemble methods. If we look closely, we can even see the ensemble in their team names. So initially, number 12 BellKor merged with number 10, BigChaos and they became number seven, Bellkor in BigChaos. Then later on BellKor in BigChaos combine with Pragmatic Theory, it becomes the final winner, BellKor's Pragmatic Chaos, and likewise the number two teams is also named The Ensemble. So as you can see, method really work in practice. Now let's learn some of the most popular ensemble method together.



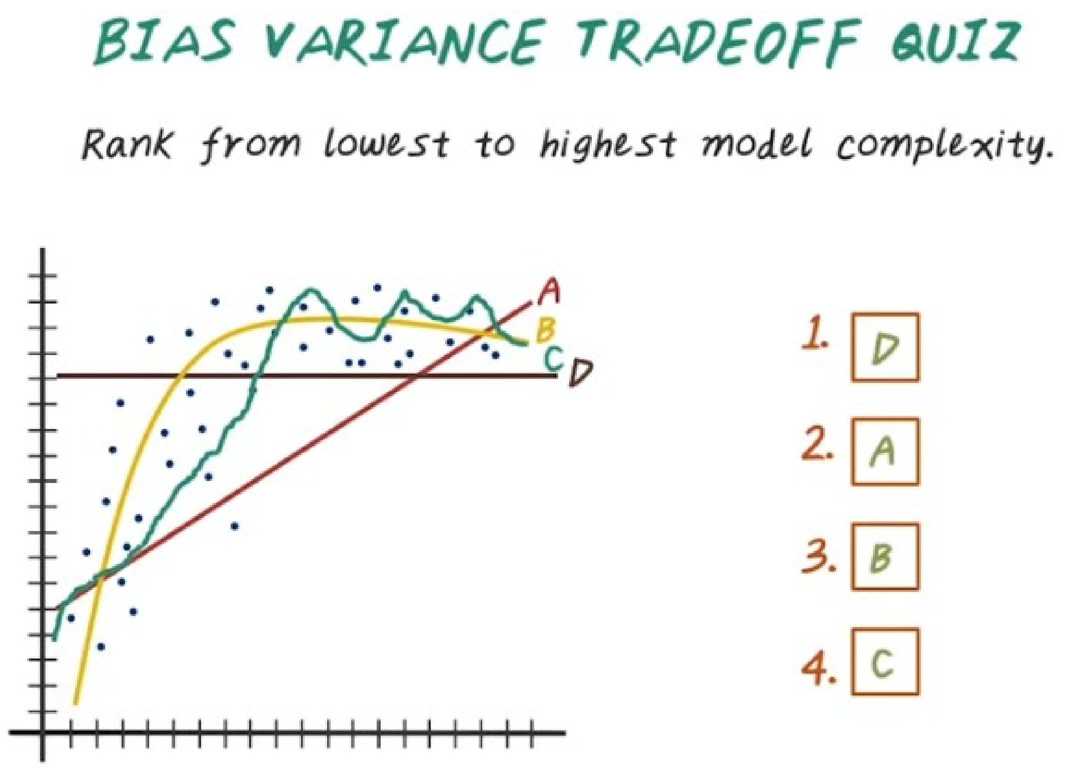
7. In general ensemble method involves three steps. First given an input data set, we need to generate a set of data set, D1 to DT for subsequent model training. In this step, those data set can be either generate independently like in bagging, or sequentially like in boosting. And bagging and busting are two different methods we will talk about in this lesson. Second, each data set will be used to train a separate model. M1 to MT. Those models can be independently trained from the input data or there can be dependency among those models as well. Finally, we need to construct an aggregation function F to combine the result from all those T models. This aggregation function can be as simple as taking simple average or taking a weighted average, and the weight are determined by the algorithm. Depending on how we generate the data sets and how we train those models and what aggregation function to use. Different ensemble algorithms can be developed.



8. In order to explain why ensemble method work, we need to understand bias and variance tradeoff. Here's the visual illustration of the concept of bias versus variance. Bias refers to the prediction error due to the wrong modeling assumption. For example, if the model assumes linear relationship between the target. However, in reality the data do not follow the linear trend, then the difference will be due to the bias of the model. The variance on the other hand, refers to the error from 「the sensitivity to small fluctuation in the training data set」. Ideally, we want a model to have both low variance and low bias. Intuitively, it can be illustrated by the following example. Here we have a two by two example, and x-axis shows the low variance versus high variance. While the y-axis shows the low bias versus high bias. The red center in every figure indicate the ground truths target, and all those blue dots are prediction from the model. In this first example, all the model predictions, those blue dots, are concentrated on the red target. This means the model has low bias and low variance. And this is the optimal scenario for modeling. In the second example, the model prediction are scattered around the center, but not really concentrated in the center. So this means the model has low bias, but high variance. In this third example the model prediction are clustered at one position, but not on the target. So this means the model has high bias but low variance. In this case the model can be easily fixed by shifting the prediction towards the center. In the fourth example, the model predictions are scattered and away from the target. This means the model has high bias and high variance, and this is the worst possible situation.



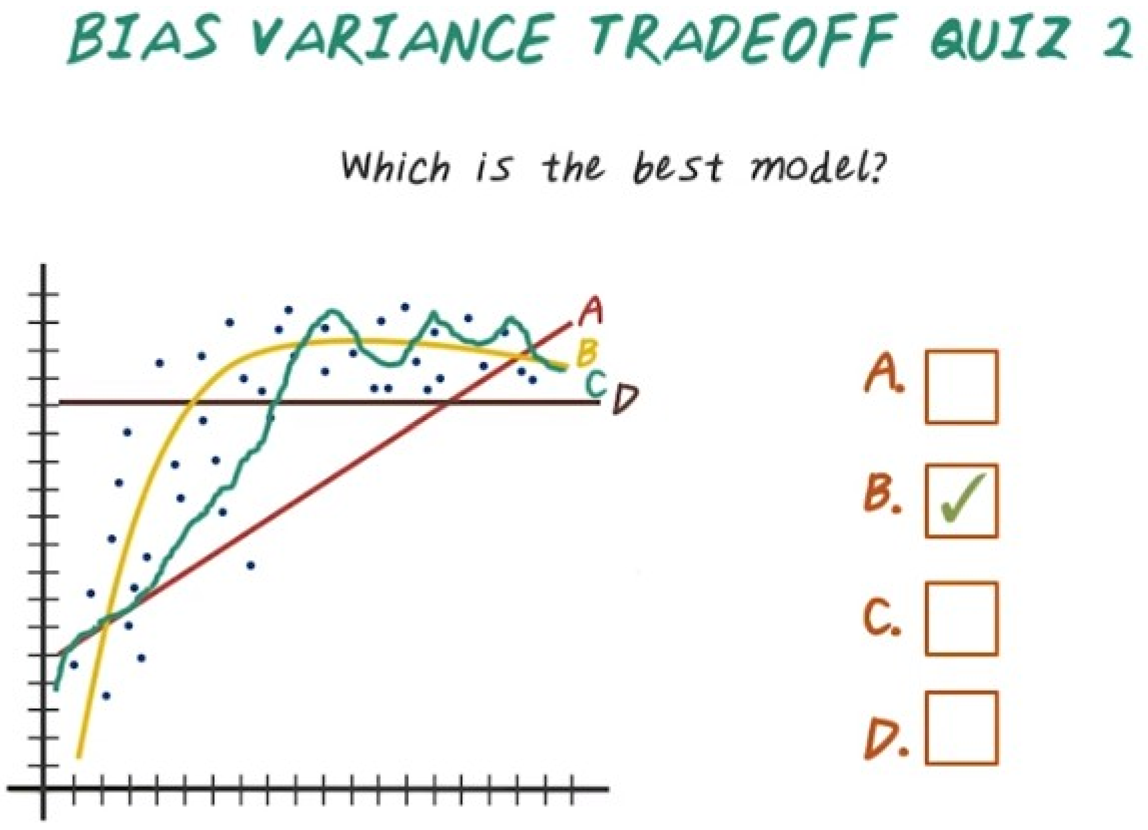
Now we understand the intuition behind the bias and variance. Now let's look at the relationship between bias variance. I want to explore the relationship using this plot. The x-axis is model complexity, and the y-axis is the error from the model. In general, there's a tradeoff between bias and variance. As we can see from this curve, as we increase the complexity of the model, bias decreases. This is because a flexible or complex model can often fit the data better. Therefore, low bias. However, the error coming from variance will increase as we increase the complexity of the model. This is because a complex model is more susceptible to over fitting, hence high variance. So the total error is error due to the bias, plus the error coming from the variance. Our goal is to find optimal model complexity that balance the right amount of bias and variance that lead to the minimal error. And note that the best model will change from task to task, and from data set to data set. So, it's not possible to expect the same algorithm that always perform well in all cases. Therefore, it's important to understand the trade off and search for the best model for your task.



本題是根据函數的形狀做出來的, 不用數据

9. Now we understand the bias and variance tradeoff. Let's do a quiz about model complexity. Here are four regression models, and x axis is input feature and y axis is the output target. Rank all those four models from lowest to highest model complexity. And we have four models here. A is this linear line. B is this curved line. And C, is this wiggly line and D is this flat line. So rank these four models from lowest to the highest model complexity.

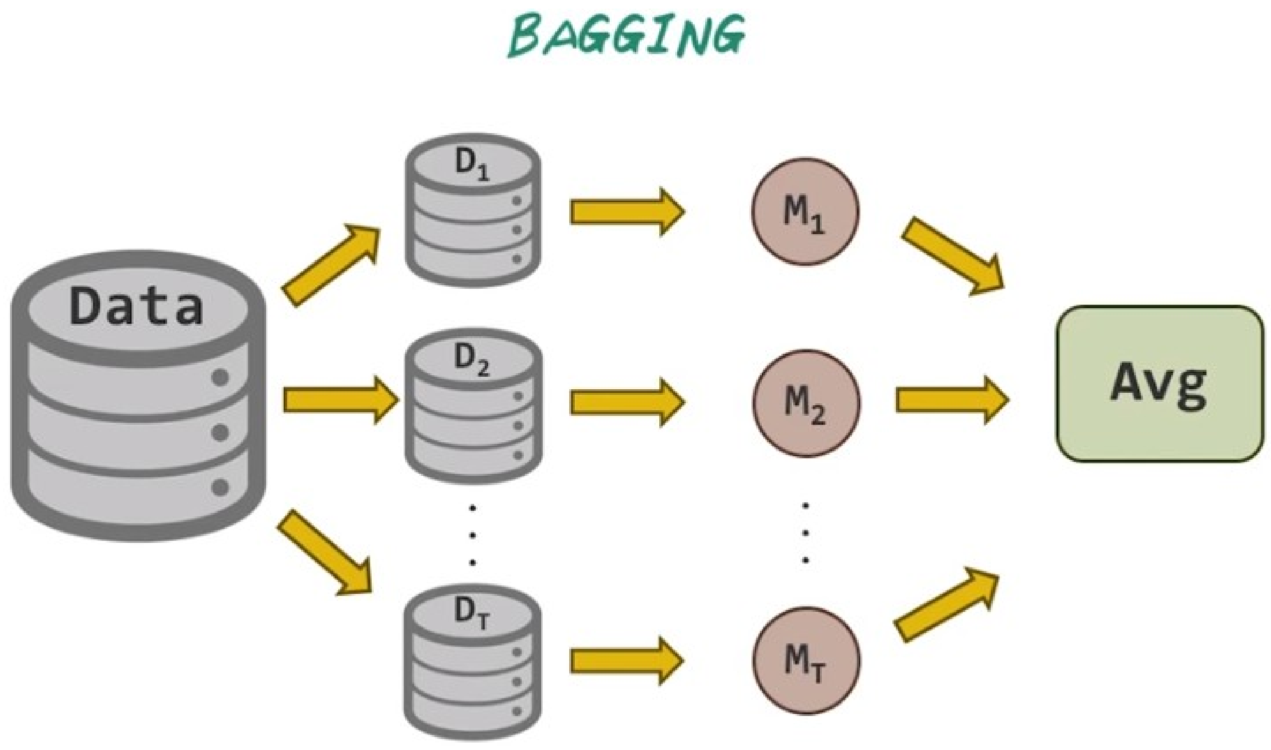
10. And here are the answers. The flat line, D, is the model with the lowest complexity because it has smallest variance, with just a single parameter to specify. That is the height of this line. And this linear line, A, is the second because it is also pretty simple model and can be specified with just two parameters. And the variance of this model is also quite low. And the smooth curve, B, ranked as third. Because it is more complex than the linear line and the flat line. And finally, the wiggly line C is the most complex one, due to the complex shape of this function.



本題是根据 數据 和 函數的形狀 的比較做出來的

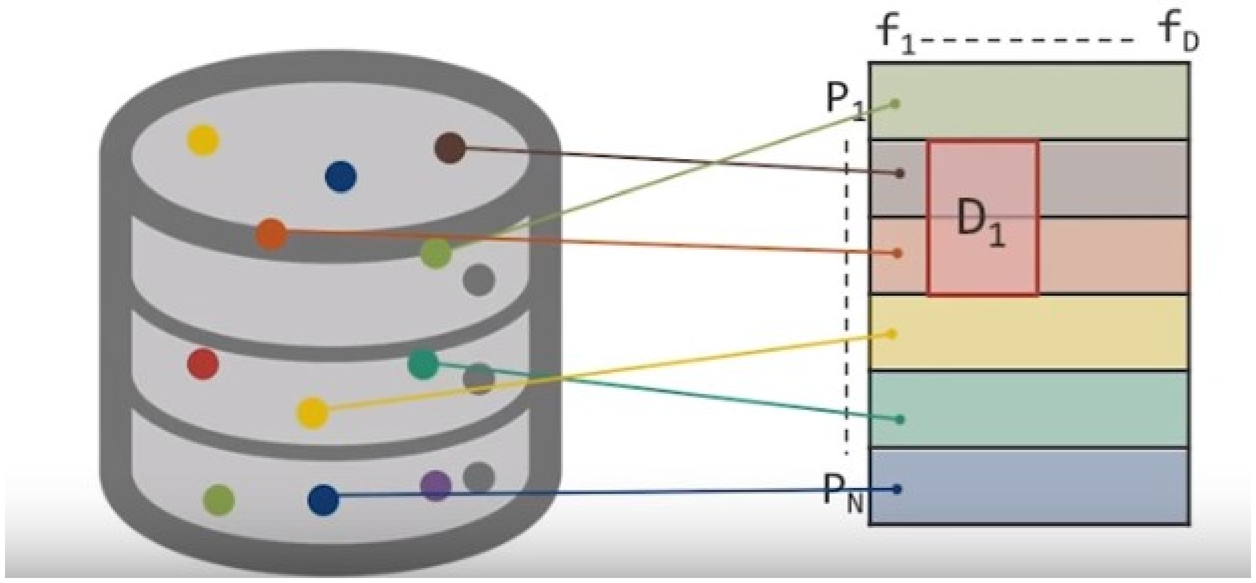
11. Lets do an other quiz, using the same examples. Note that, all those four models are trying to approximate under line data, which are those loop dots. So, which of the models is the best for approximating the underlying data?

12. The answer is B, because B fits the data better than A and D, which means the modeling assumption of B is closer to the underlying data than those linear models. At the same times, the complexity of B is much better than this wiggly line that's specified by C. So, B actually balanced the bias and variance trade off and it's the best model.

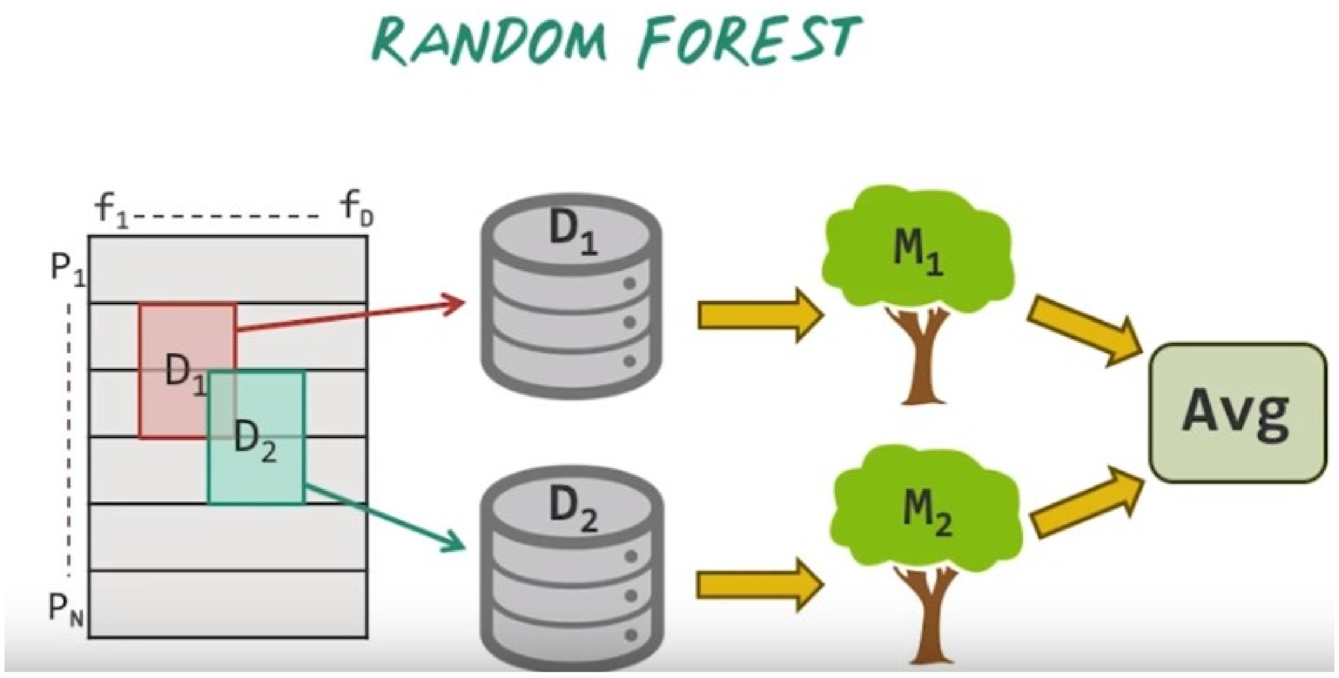


13. Now let's talk about some popular ensemble method. Let's start with bagging. Bagging is a simple, yet powerful ensemble strategy, which is named by Leo Brayman. Given an input data set, the idea behind bagging is to take repeated bootstrap (後面馬上講意思, 别查, 無用) samples from input data set to generate all those sample data set: D1 to DT. Here bootstrap sampling means sampling with replacement from the original data set, then based on the sample data set, we separate models, M1 to MT. Then, finally we classify a new data point by taking the majority vote or average of all those models.

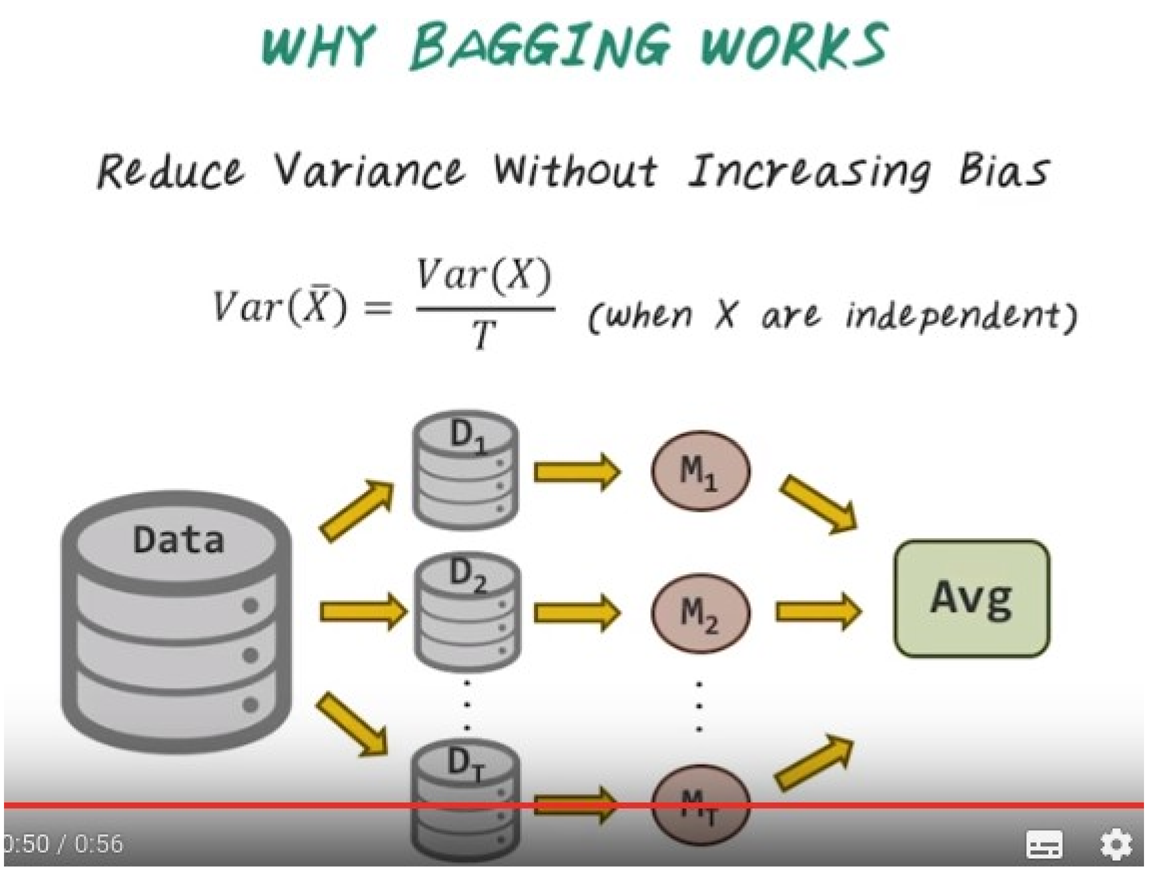
RANDOM FOREST



14. Bagging is one general strategy for ensemble. Random forest is a classical bagging algorithm, where the underlying models are decision trees. So, now let's talk about random forest, how it works. Imagine we have a large patient database. Every dot's here corresponding to a patient, and each patient is represented by a high dimensional feature vectors, such as age, gender, diagnosis, medication. Then we can represent those patients by a matrix, for example, we have N patients and D features. Every row is a patient, every column is the features. That's our entire data set. Then Random Forest will randomly sample subset of patients in every row corresponding to a patient in the original database and every column corresponding to a feature. Then Random Forest will randomly sample a subset of patient and a subset of features to construct a sub-matrix, like D1 or D2.



Then Random Forest will use those sub-matrices as their input data to build separate decision tree. And the way how the tree are constructed is simply recursively select a feature from the sub-matrix and split based on that feature and repeat this process until all the features in this sub-matrix are used. In this manner we would generate multiple trees. Once we have all those models, when a new patient comes we can score that patient against all those models, then taking an average, and use that average as our final prediction. Note that the algorithm for constructing those trees in Random Forest is much simpler than a traditional decision tree algorithm, such as C5. This choice is intentional. In fact, there are theoretical studies showing that it's actually important to use those simple methods, so that we generate a diverse set of models in bagging. The other benefit for using those simple algorithm is computational cost will be dramatically reduced.

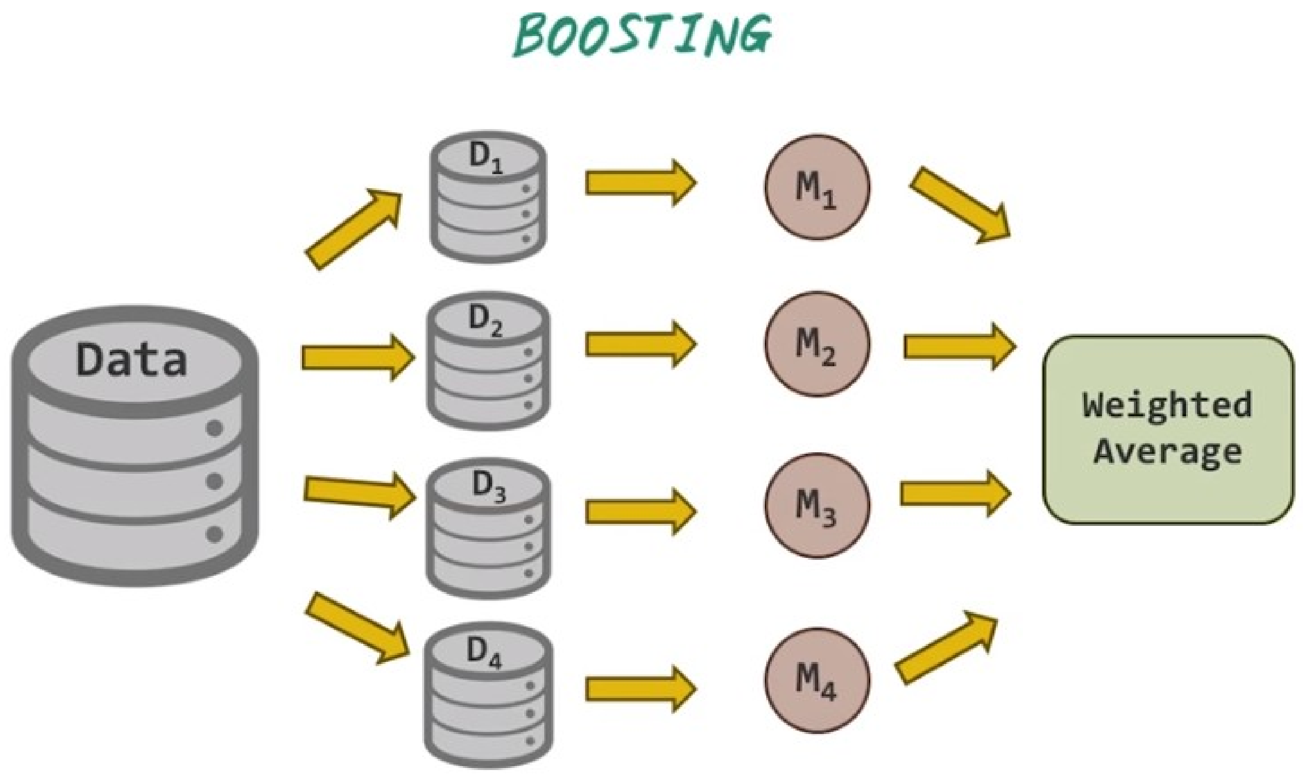


WHY BAGGING WORKS

Reduce Variance Without Increasing Bias

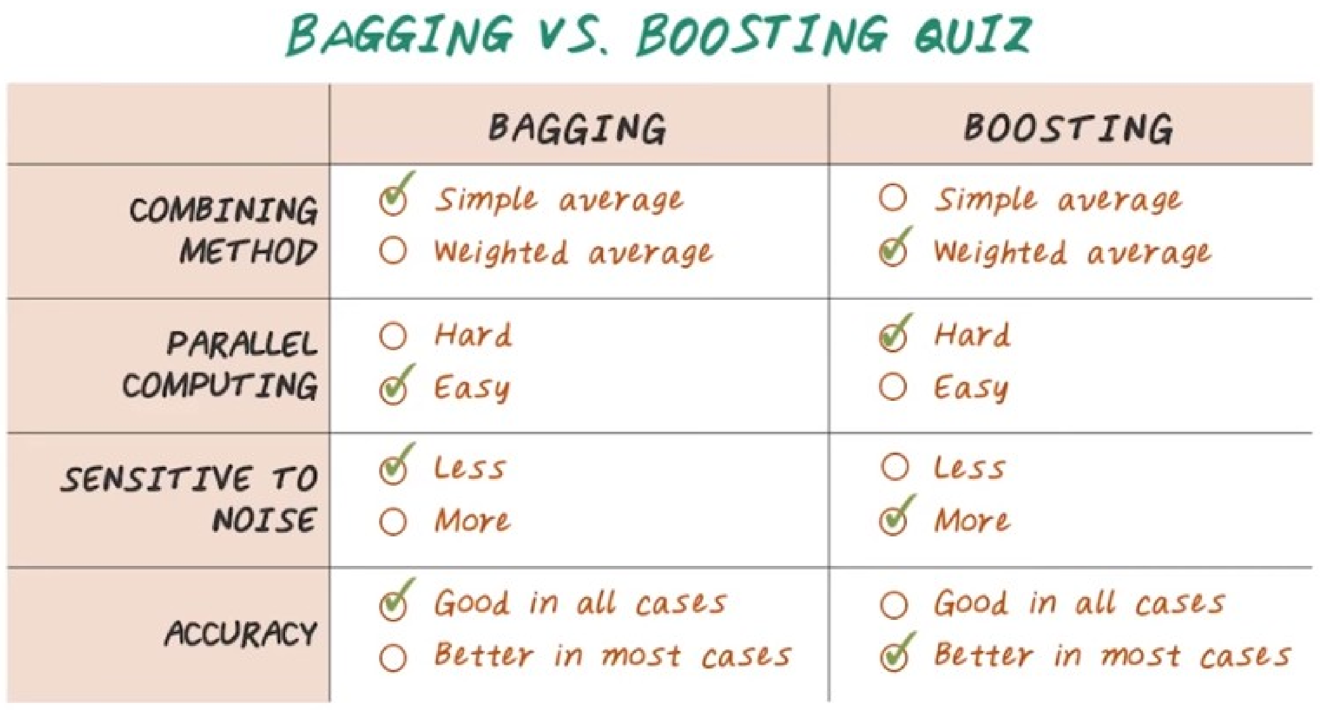
$$Var(\bar{X}) = \frac{Var(X)}{T} \quad \text{(when X are independent)}$$

15. So why does bagging work? The reason is because, bagging reduces the variance of the final model without increasing the bias. Since the final model is just take the mean over multiple model, the mean of the final model is the same as the individual model. Therefore, bias is the same. However, the variance of the model can be reduced by averaging. So the intuition come from simple statistic. That the variance of a random variable X, can be reduced by a factor of T, if we measure the variance of the mean over T independent identically distributed samples of x. So in backing, all the models are viewed along both trap samples, which can be considered close to IID. Therefore, the variance of the final model can be greatly reduced by averaging.



16. So boosting involve incrementally building those models one at a time and emphasized the later model to the training instances that the previous model misclassified. For example, so we start with building model M1 on the first data set, D1. Then we test the performance of model M1 on data set D2. Then based on what mistakes, what misclassification has happened on D2, we train another model M2 to really focus on correct those misclassification has happened because of M1. Then we repeat this process by combining M1 and M2 together and test it against the third data set, D3. Again, these three is trying to correct the mistakes, the combined model of M1 and M2 misclassified, then repeat this process again on D4, and to get the final model M4. And the final model become a weighted average of all the past model we have trained. So in some cases, boosting has shown to be yield better accuracy than bagging, but it also tend to be more likely to over fit the training data. By far the most popular boosting algorithm is Eta-Boost. But there are many other boosting algorithms out there as well. So, more details will be provided in the instructor note.

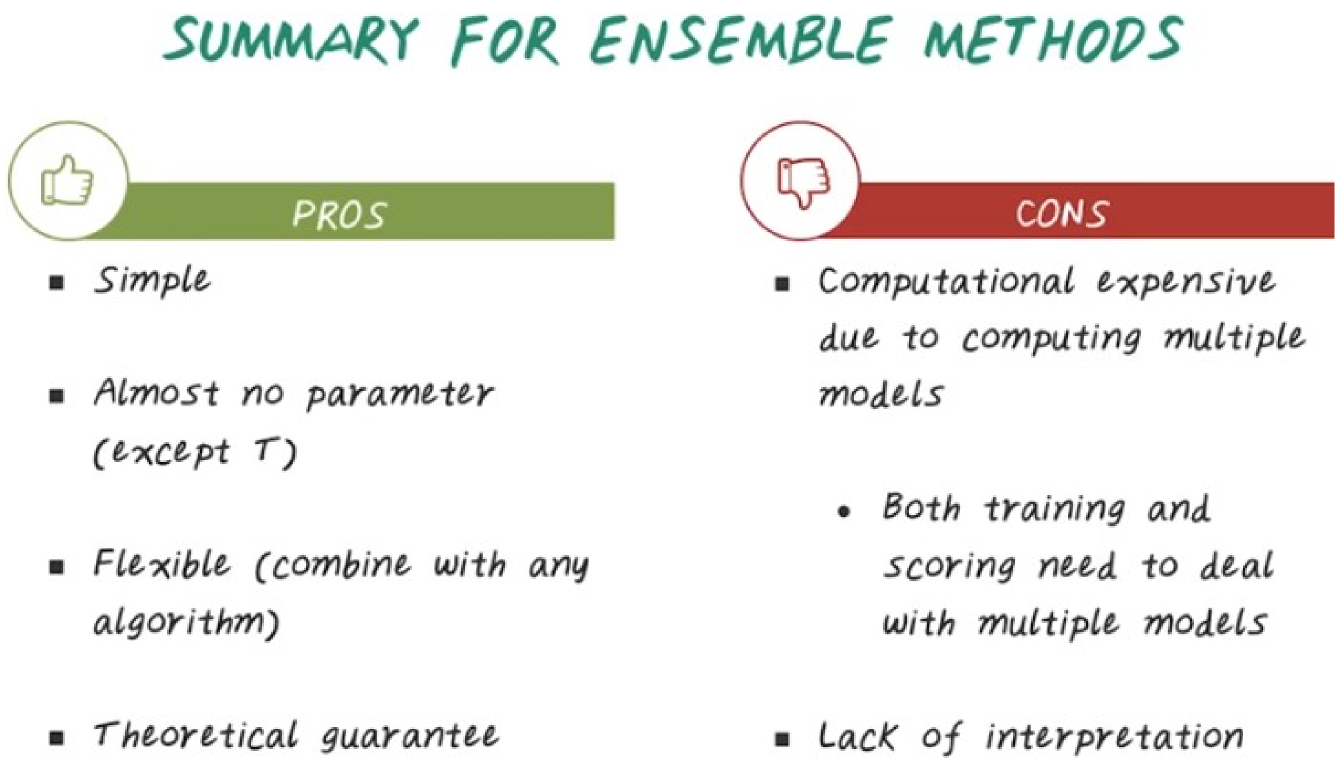
17. Here's a quiz to compare bagging and boosting. Here, every rows are some characteristic of the method. First, how do we combine those different models, it says by taking simple average, or weighted average, or can the method be done in parallel? Is it easy or hard? And how sensitive is the method towards noise? It's more sensitive or less sensitive? And finally, what about the accuracy?



BAGGING VS. BOOSTING QUIZ

	BAGGING	BOOSTING
COMBINING METHOD	[✓ Simple average] Weighted average	Simple average [✓ Weighted average]
PARALLEL COMPUTING	Hard [✓ Easy]	[✓ Hard] Easy
SENSITIVE TO NOISE	[✓ Less] More	Less [✓ More]
ACCURACY	[✓ Good in all cases] Better in most cases	Good in all cases [✓ Better in most cases]

18. So here are the answers. In bagging, we use simple average to combine all those models, because the models are developed independently. And in boosting, we take weighted average, because there is a sequential dependency among all those models. In terms of parallel computing, bagging is very easy because all those models are independent. Then you can use those models separately on different cores, or different machines. Parallel computing for bagging is easy. But, for boosting, parallel computing is hard because there is a sequential dependency between the early model to the later model. We can't really compute the later model in parallel with early model. In term of noise, bagging is less sensitive to noise because the model being combined are oftentimes much more diverse, since they're viewed independently on the separate samples. Well, for boosting, it's more sensitive to noise because the sequential dependency and later model, try to correct errors that made by the earlier models, and it can over fit the data, therefore sensitive to noise. Finally, in term of accuracy, bagging often achieve good performance in all cases, while boosting, in most of the cases, give a better result, but because of over fitting problem, sometimes they perform much worse in some cases. So the analogy is bagging is like a Japanese car. It always gets the job done. It's not always the fastest car, but it's a pretty reliable car. And boosting is like a sports car. It's much faster when it works, but when it has a problem, it can cost you a lot. [SOUND]


SUMMARY FOR ENSEMBLE METHODS

PROS

- Simple
- Almost no parameter (except T)
- Flexible (combine with any algorithm)
- Theoretical guarantee

CONS

- Computational expensive due to computing multiple models
  - Both training and scoring need to deal with multiple models
- Lack of interpretation

19. So in this lecture, we talked about ensemble method. So now, let's summarize the overall pros and cons of ensemble method. In term of advantages, ensemble method are often simple. Almost have no parameters, except how many models to be combined (T). And it's quite flexible, you can combine many algorithms together. And there's many different ways how to combine them. And there are also theoretical guarantee wide works. In term of disadvantages, it's mostly coming from computational challenges. Because now, we have to build multiple models, not only at the the training time, but also at the scoring time. Every time we want to score a patient instead of score that patient against single model. Now we have to score that against three different models. The other disadvantage is, is lack of interpretation. Because the final model is a combination of multiple models. The interpretation oftentimes can be very difficult.