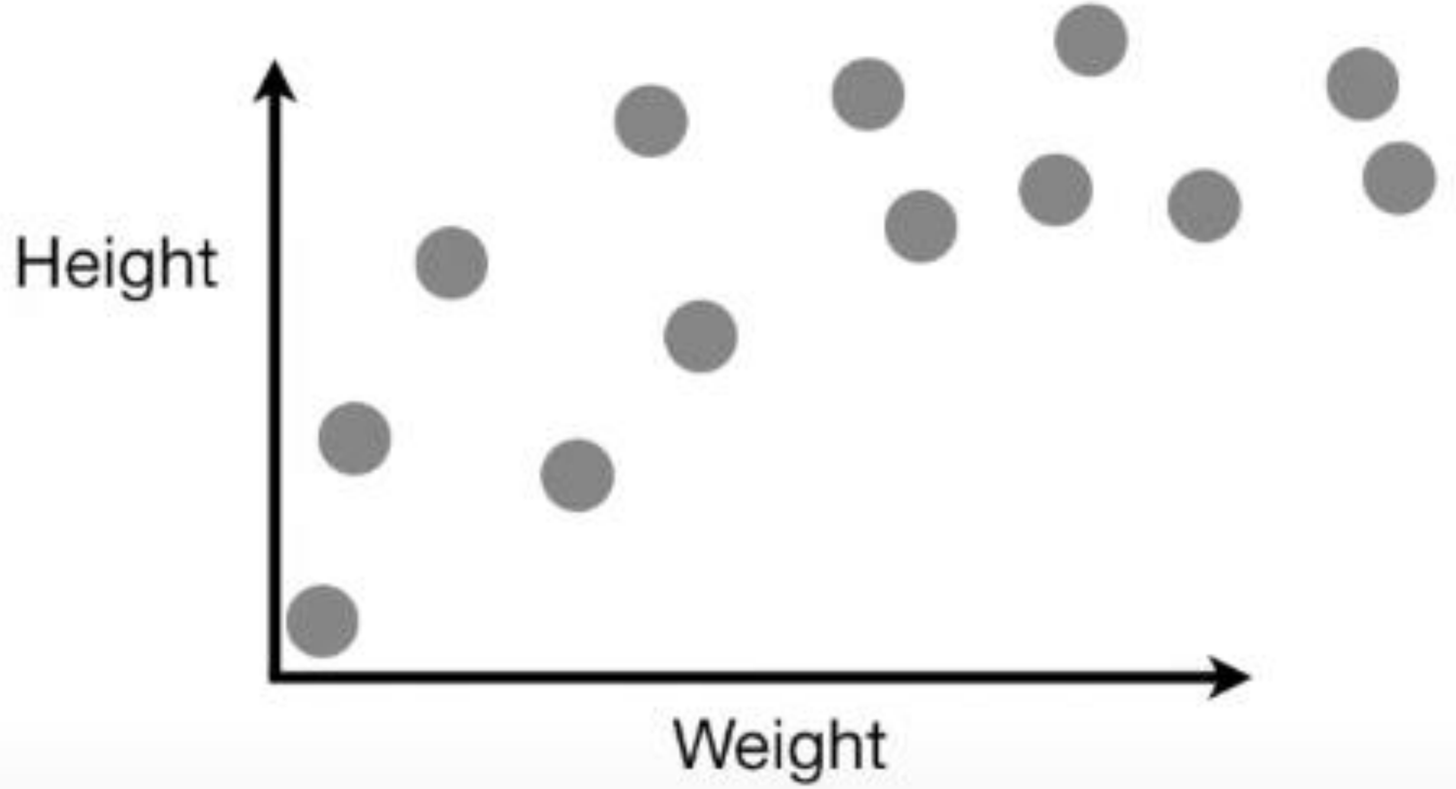
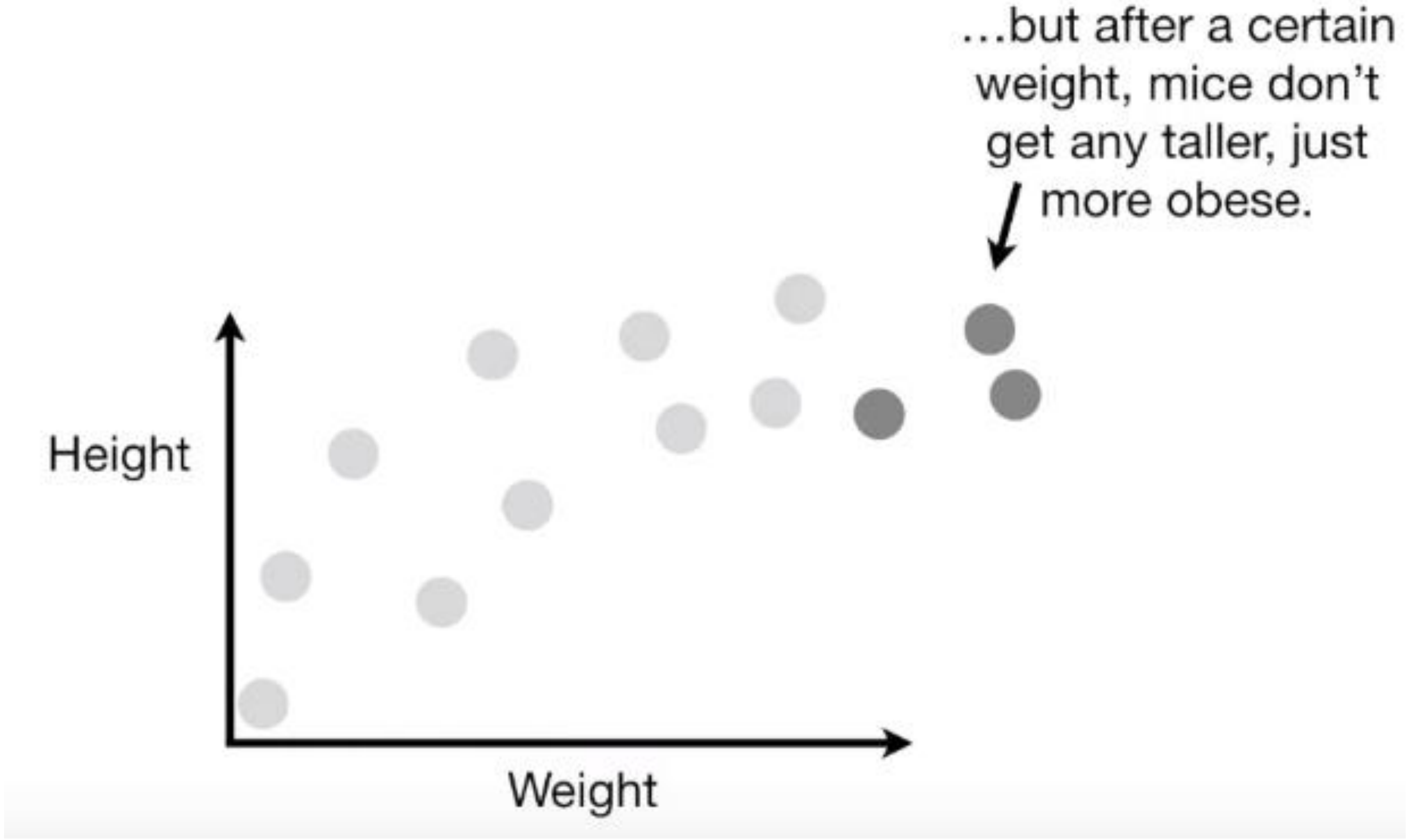
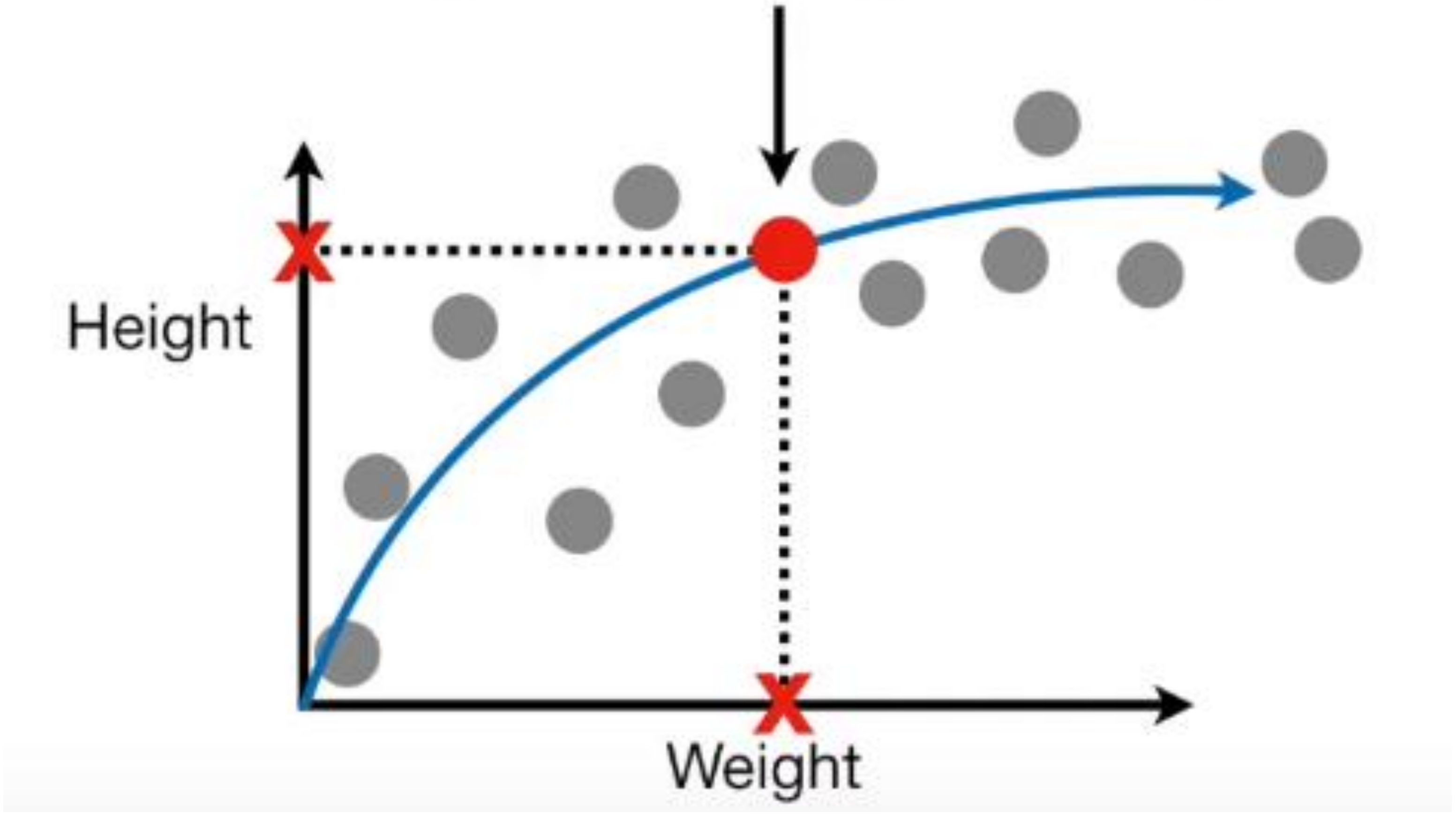
Bias and Variance and Ensembles of Classifiers

Imagine we measured the weight and height of a bunch of mice and plotted the data on a graph…

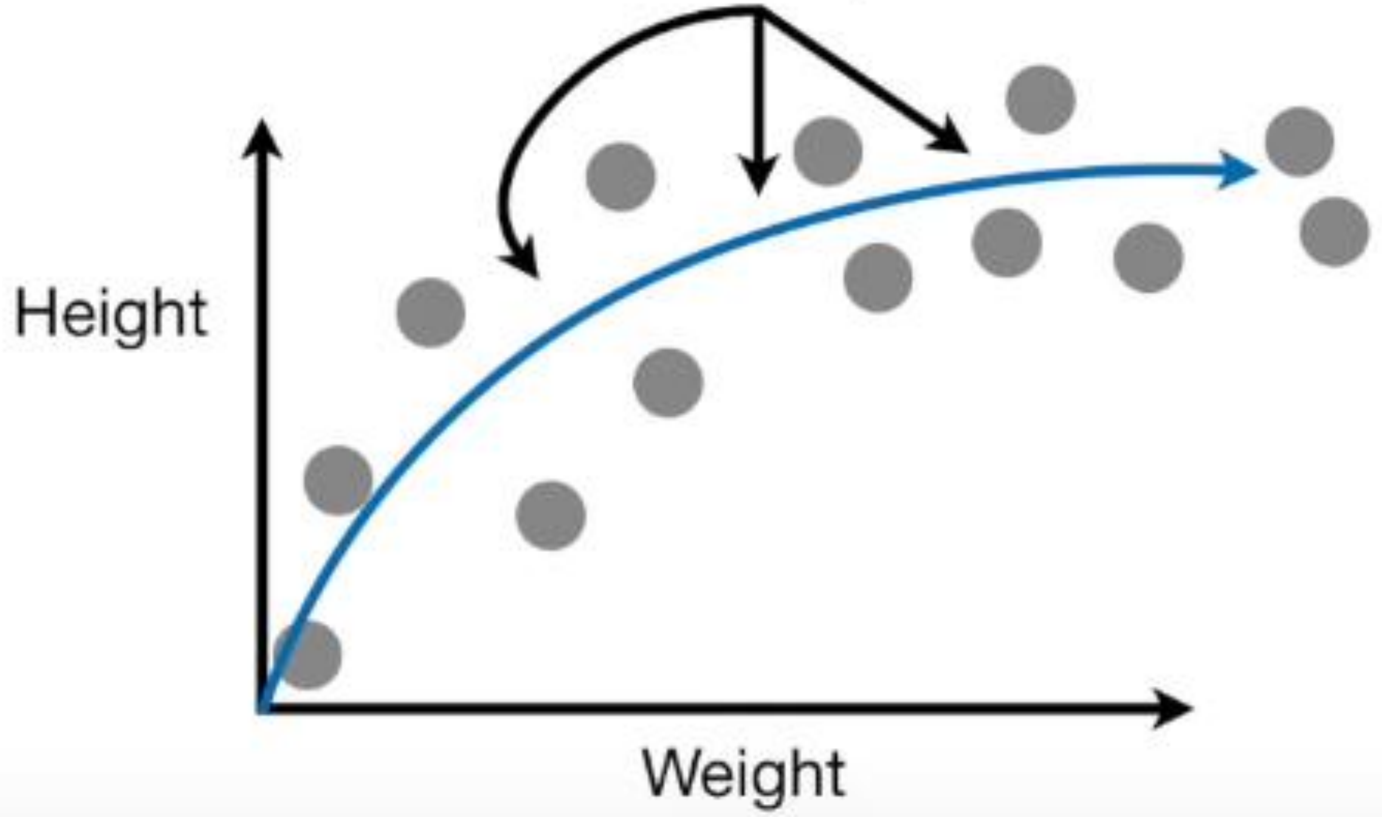



…but after a certain weight, mice don't get any taller, just more obese.
Height
Weight

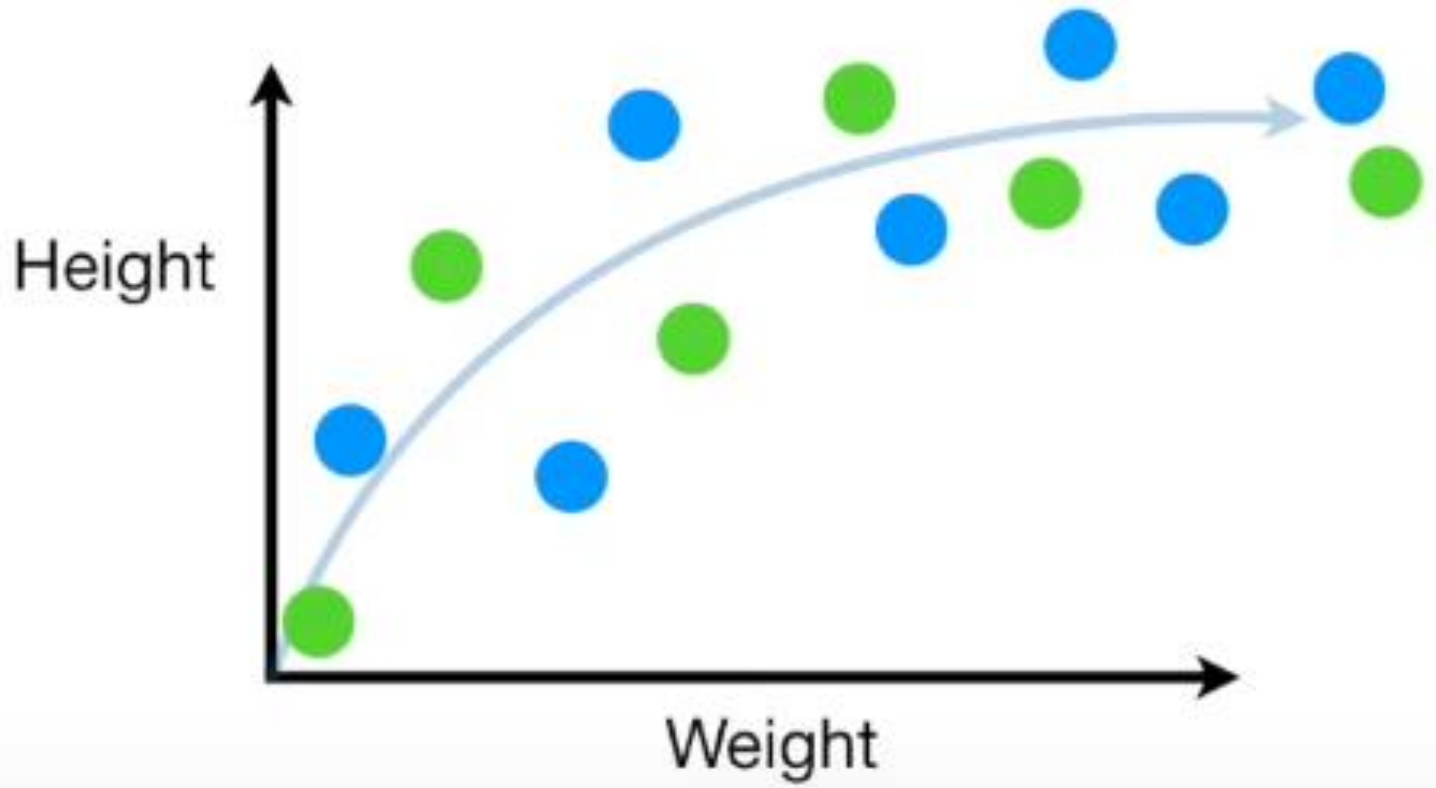
Ideally, we would know the exact mathematical formula that describes the relationship between weight and height…

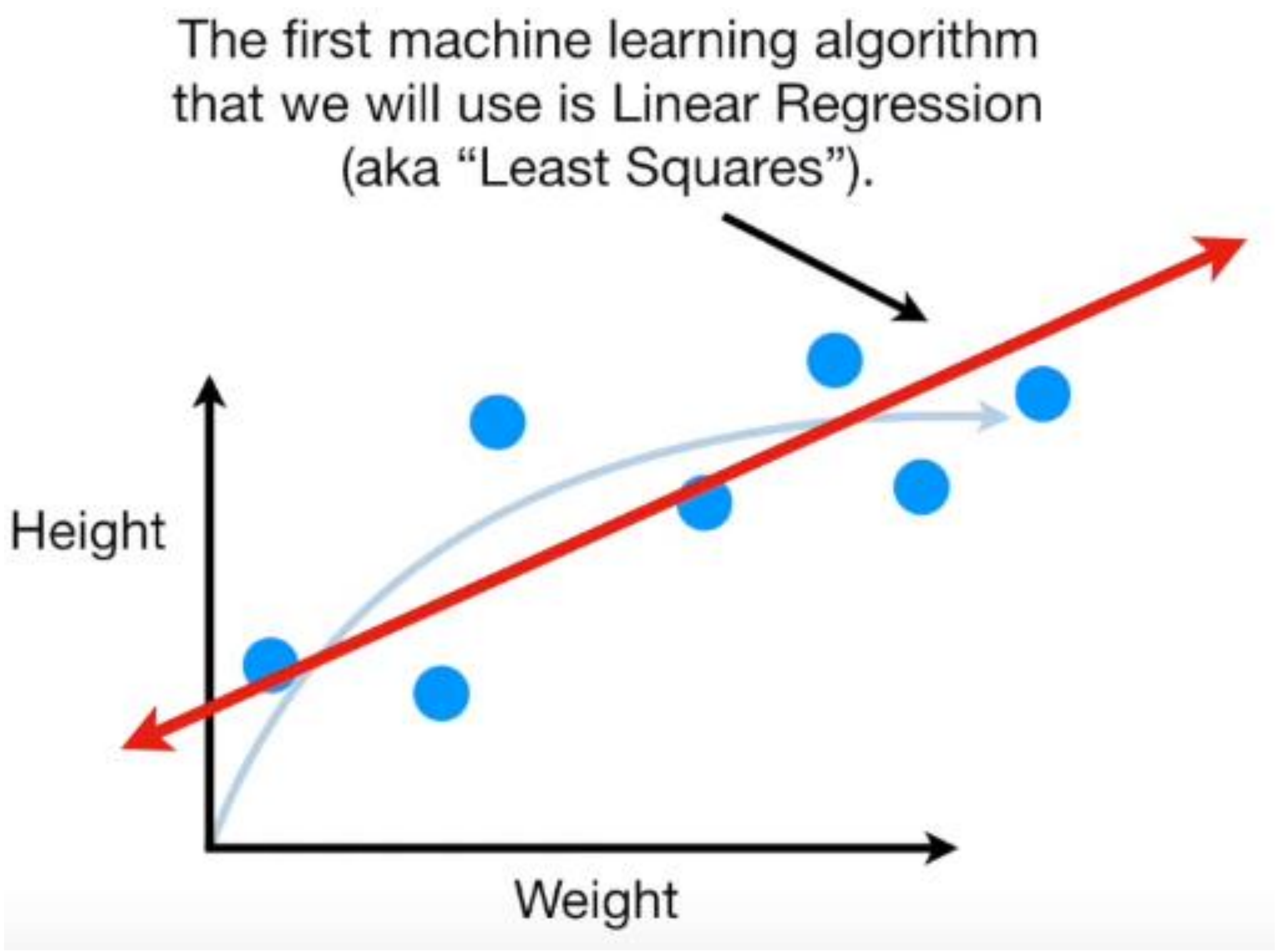


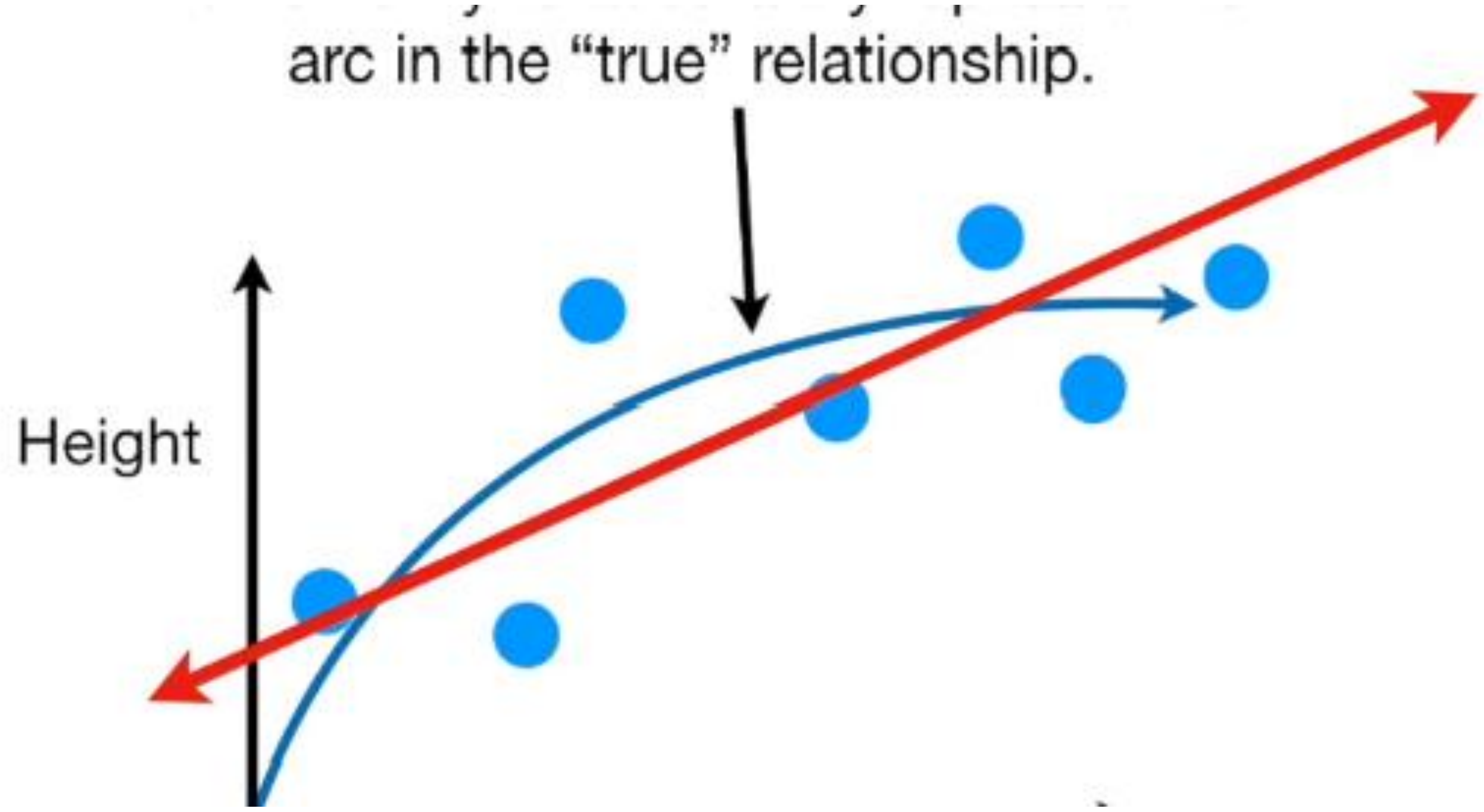
…but, in this case, we don't know the formula, so we're going to use two machine learning methods to approximate this relationship.



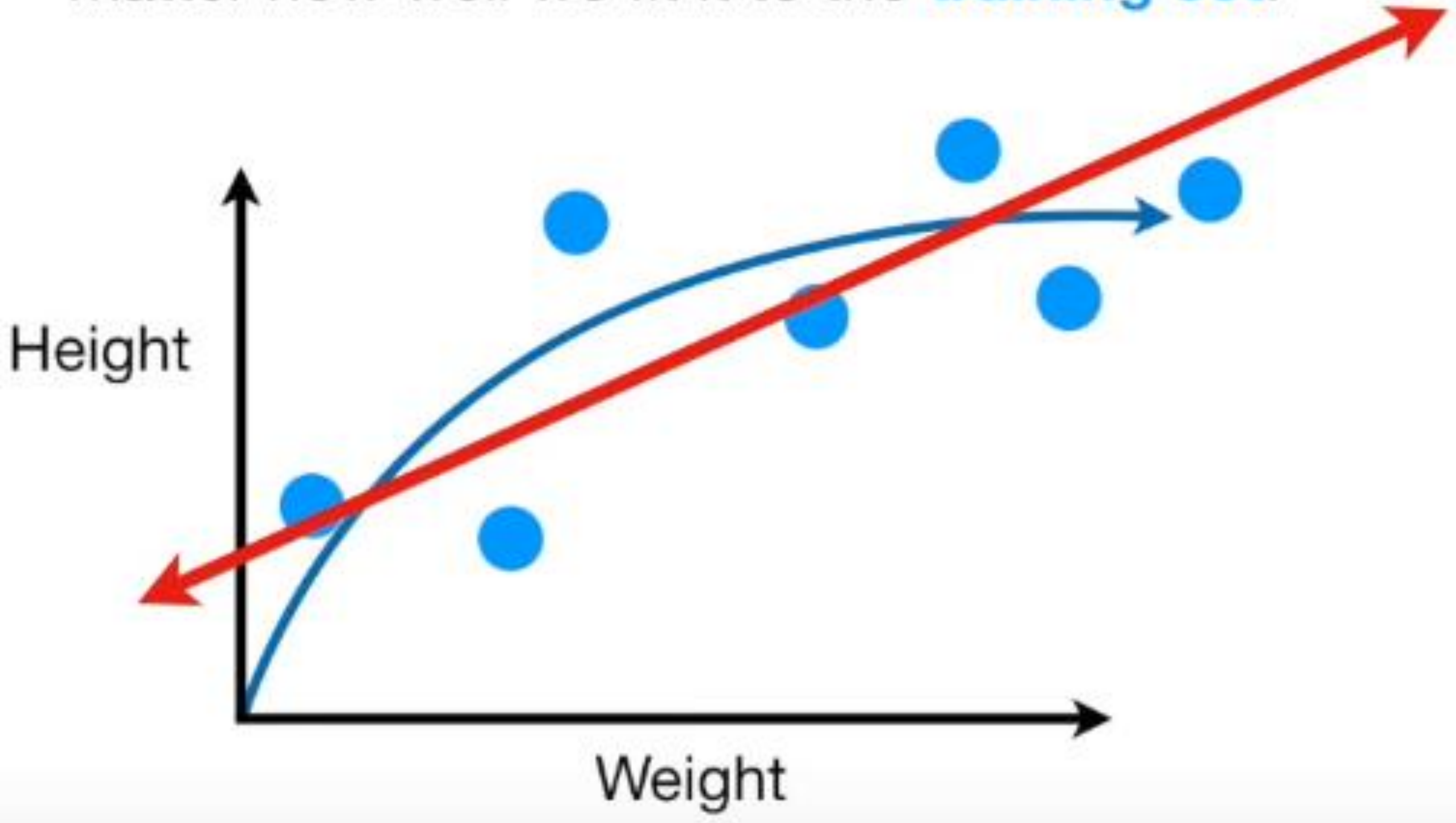
The first thing we do is split the data into two sets, one for training the machine learning algorithms and one for testing them.



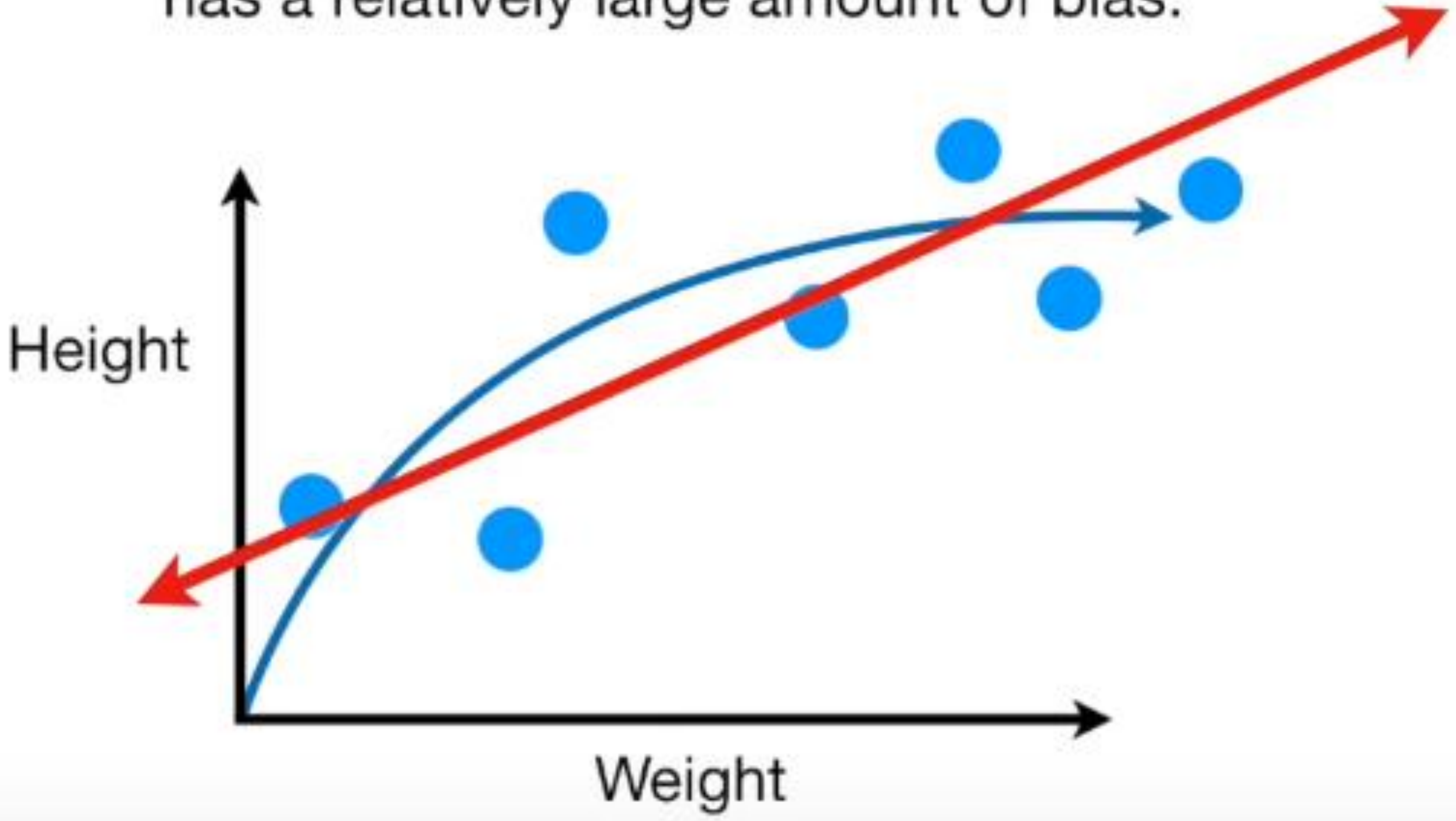

The first machine learning algorithm that we will use is Linear Regression (aka "Least Squares").


arc in the "true" relationship.
Height

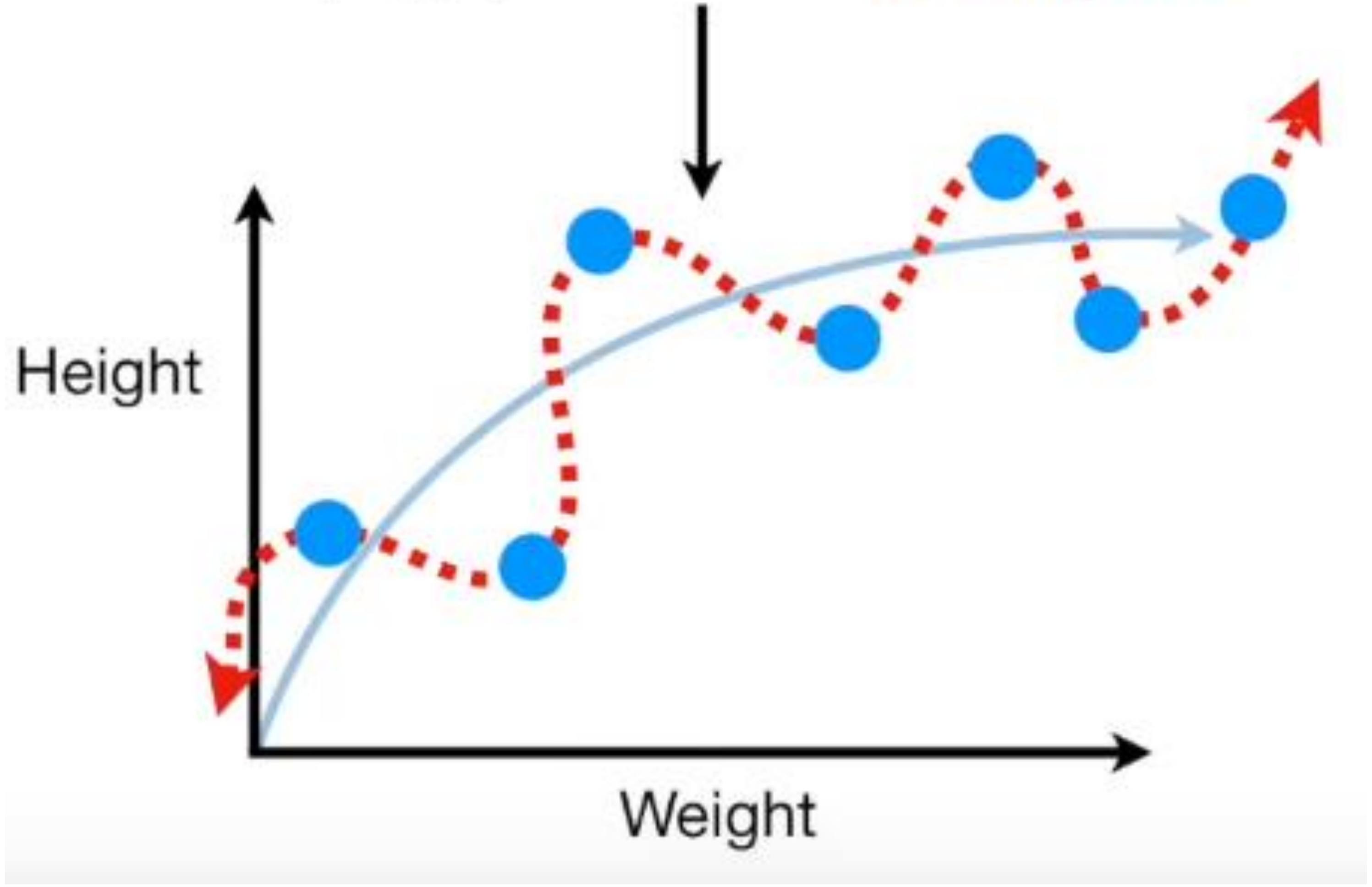
Thus, the **Straight Line** will never capture the true relationship between weight and height, no matter how well we fit it to the training set.



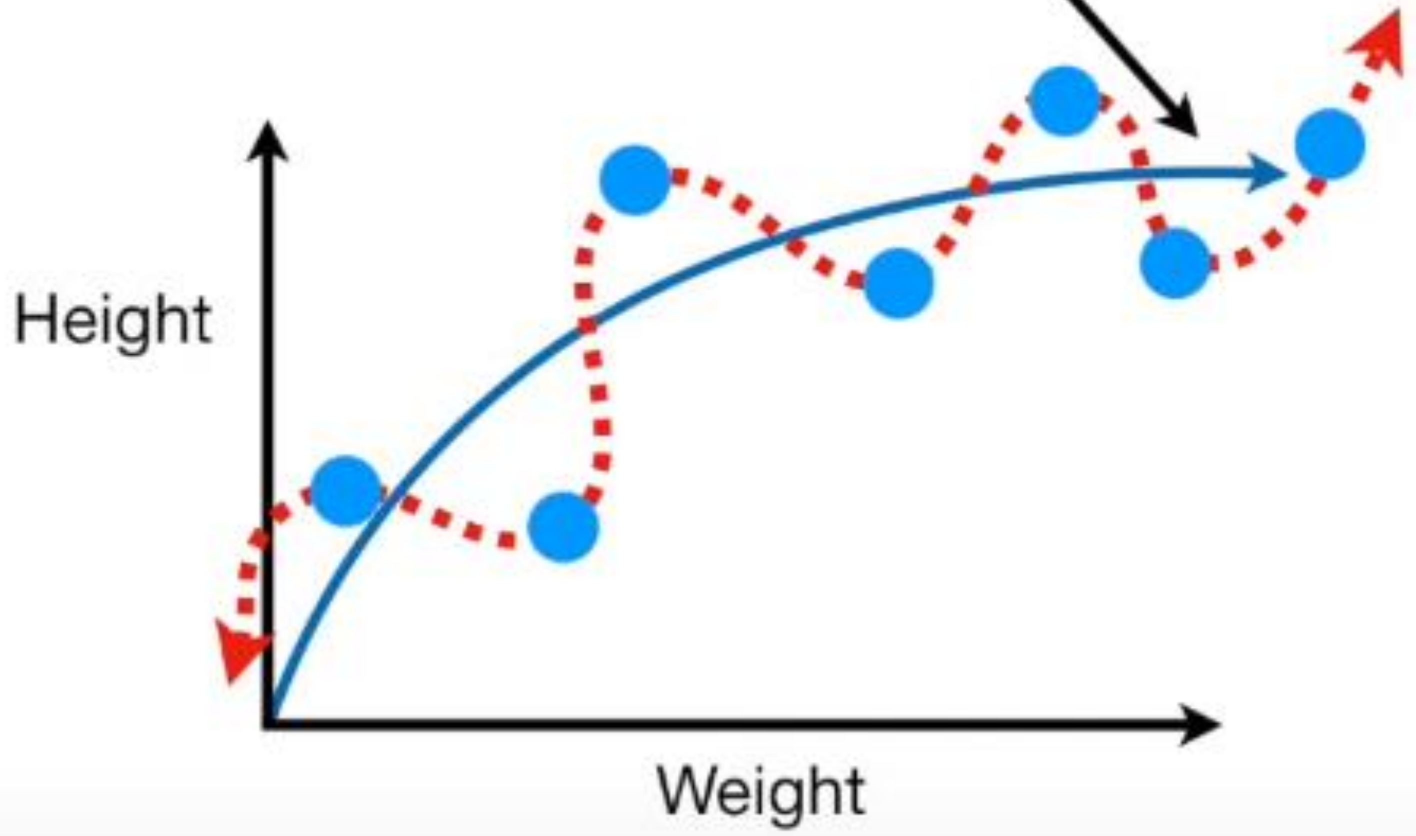
Because the **Straight Line** can't be curved like the "true" relationship, it has a relatively large amount of bias.



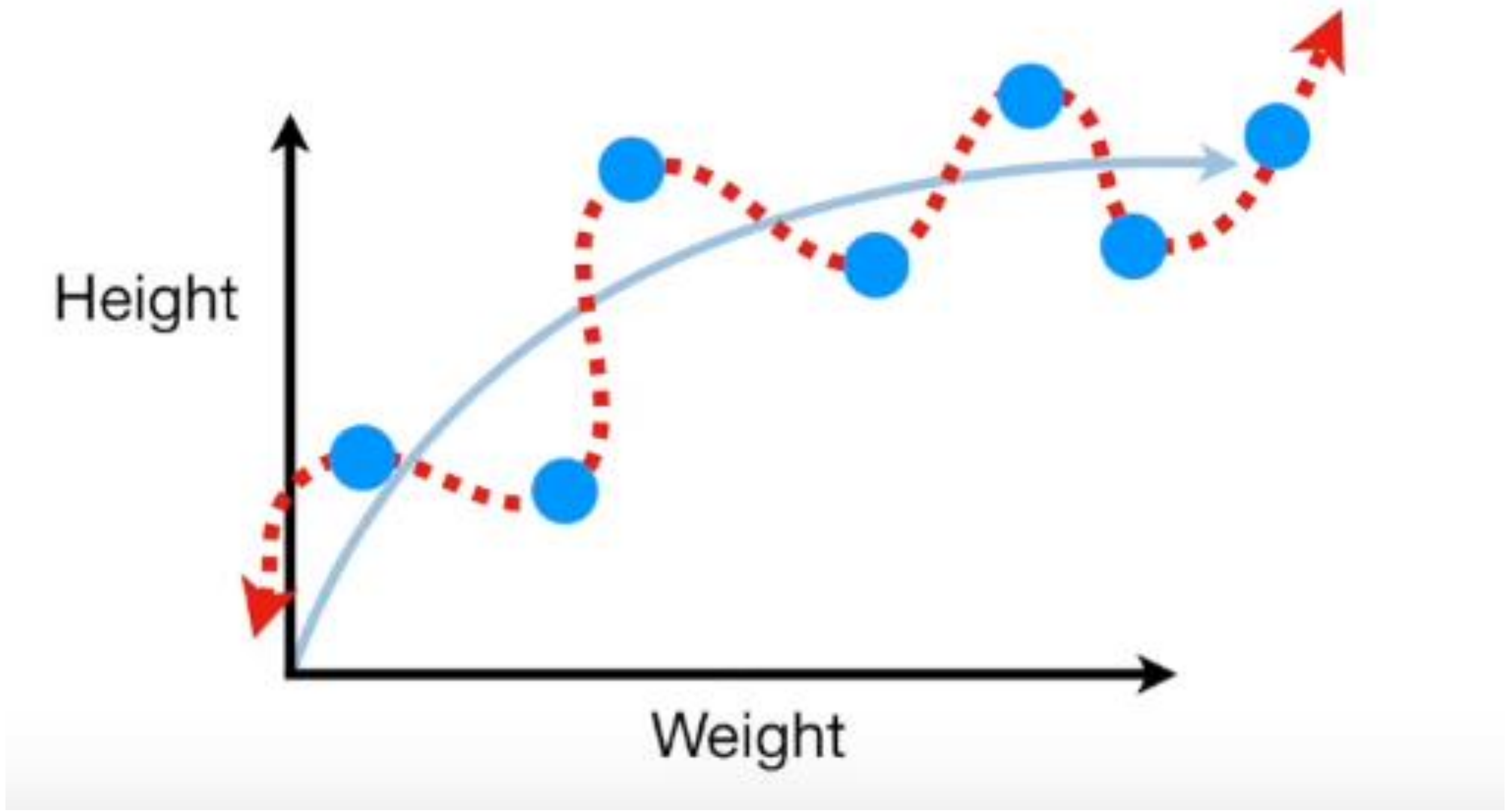
Another machine learning method might fit a **Squiggly Line** to the **training set**...



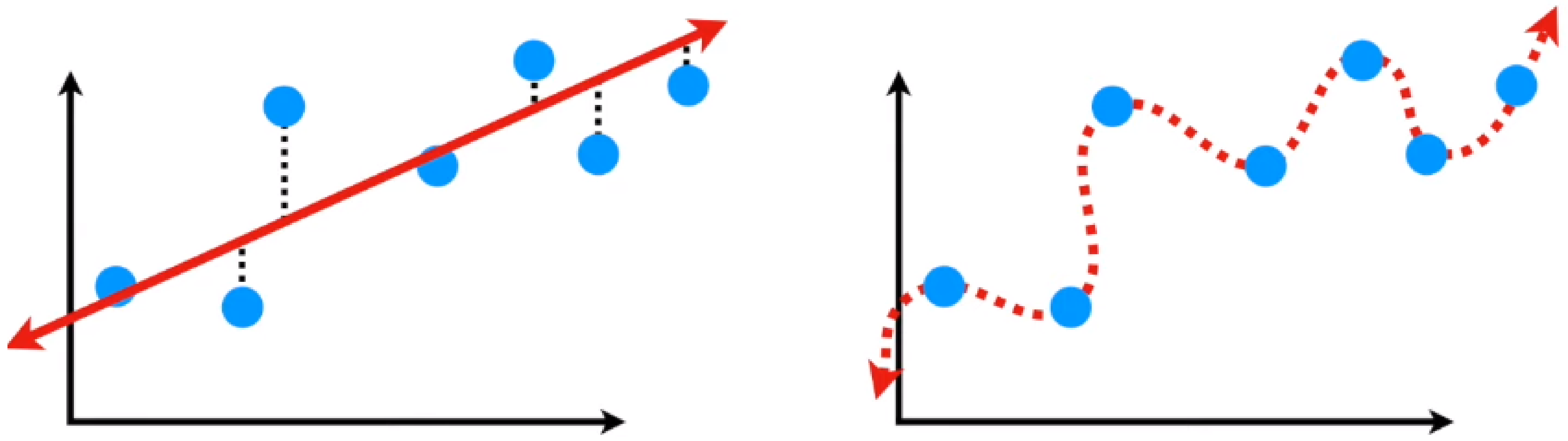
The **Squiggly Line** is super flexible and hugs the **training set** along the arc of the true relationship.



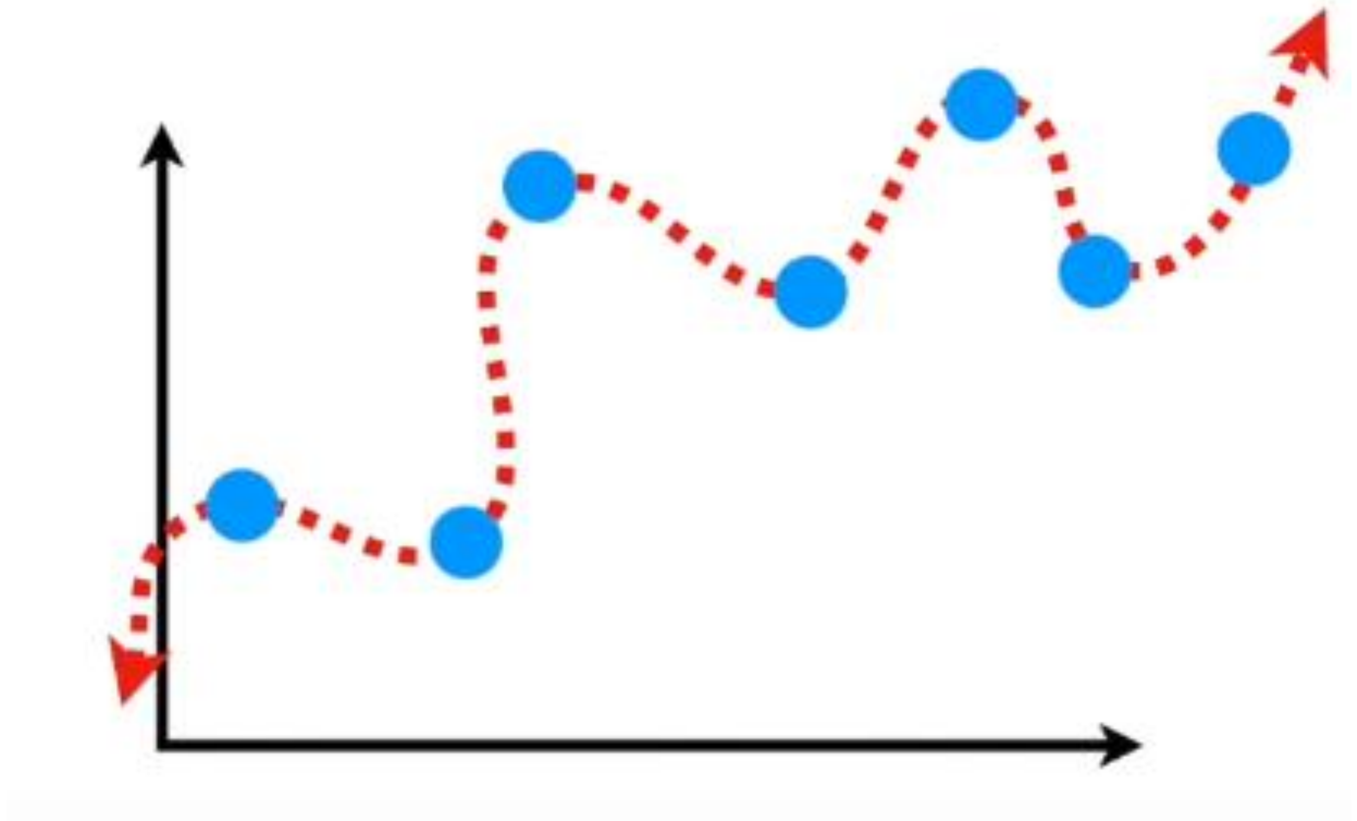
Because the **Squiggly Line** can handle the arc in the true relationship between weight and height, it has very little **bias**.



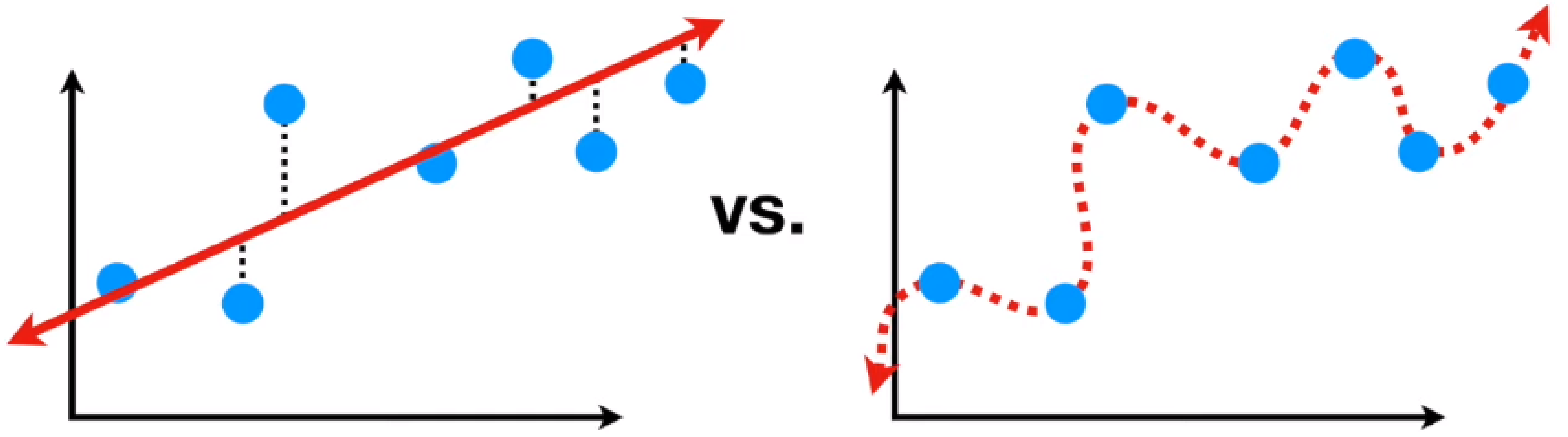
We can compare how well the **Straight Line** and the **Squiggly Line** fit the **training set** by calculating their sums of squares.

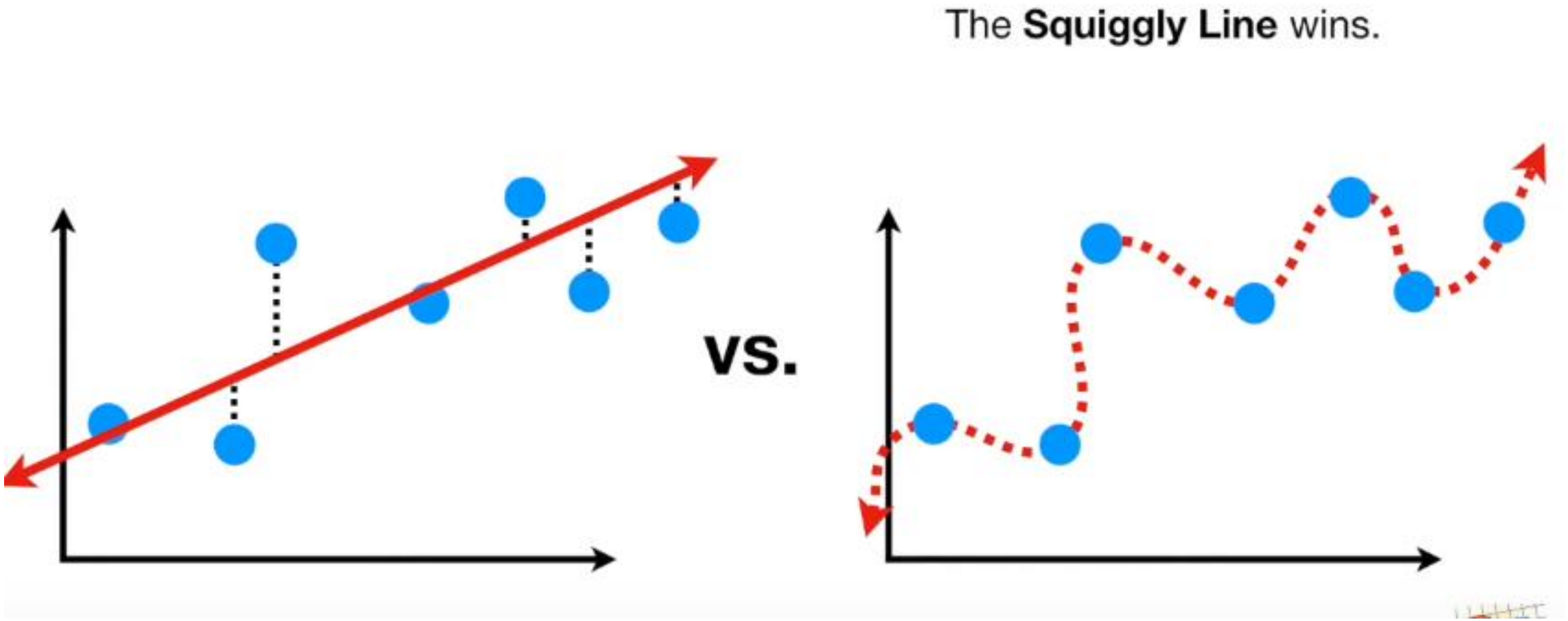


Notice how the **Squiggly Line** fits the data so well that the distances between the line and the data are all 0.

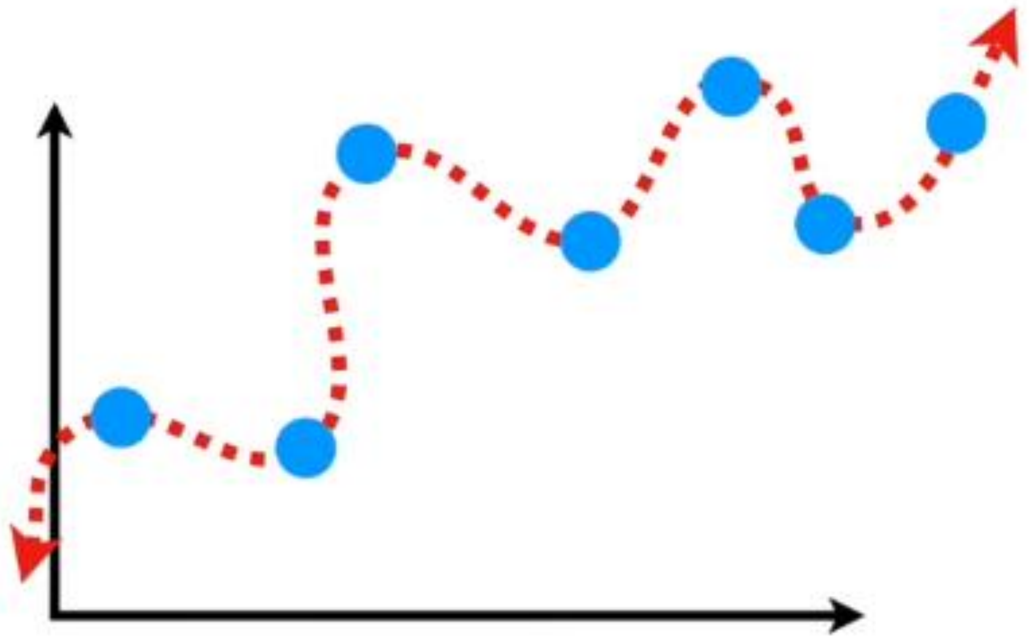


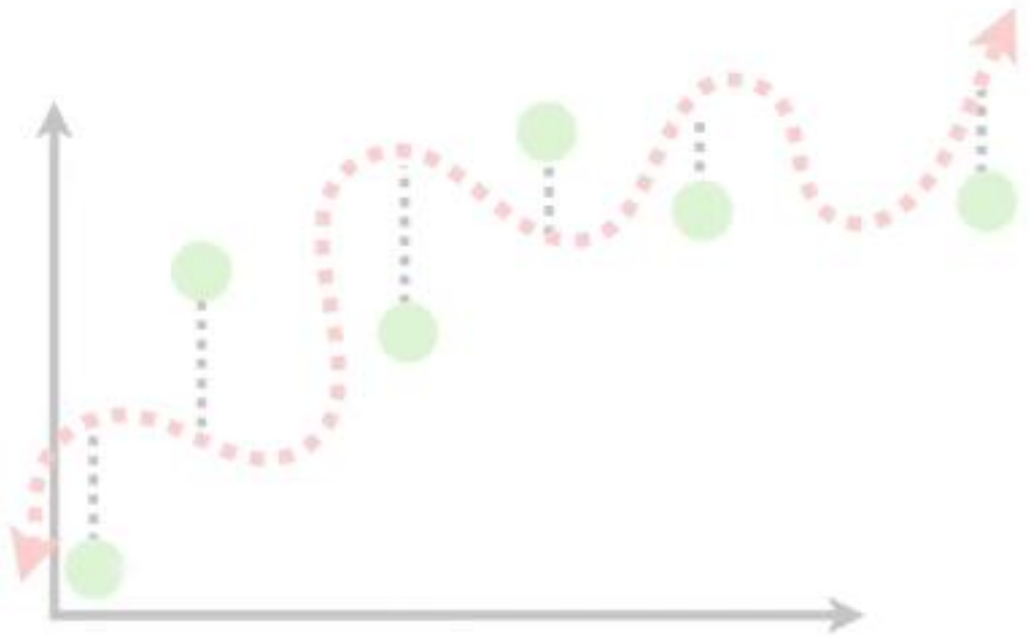
In the contest to see whether the **Straight Line** fits the **training set** better than the **Squiggly Line**…

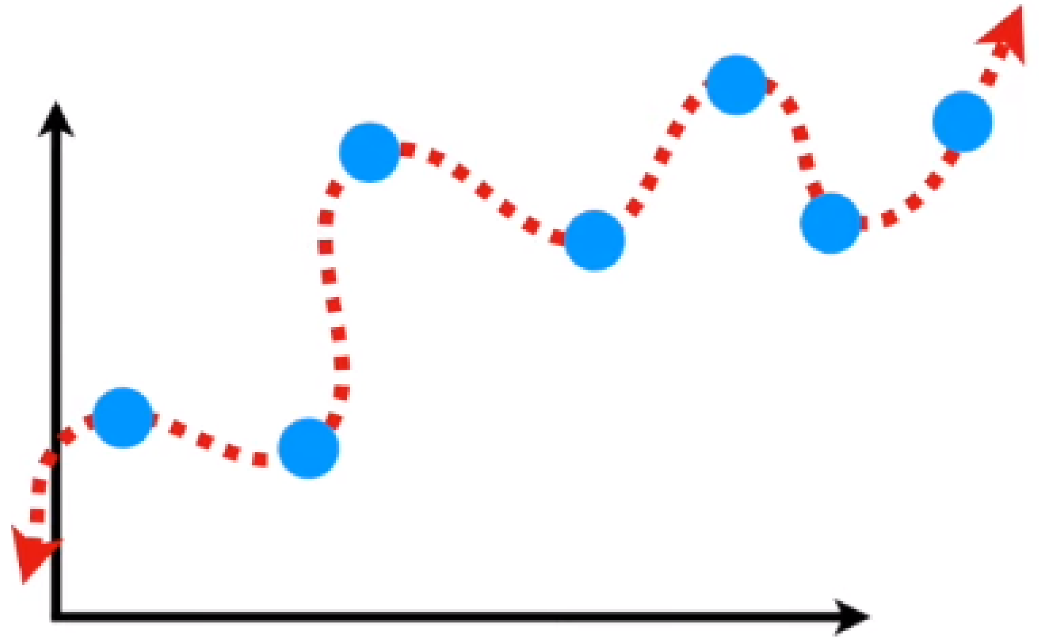



The Squiggly Line wins.
vs.

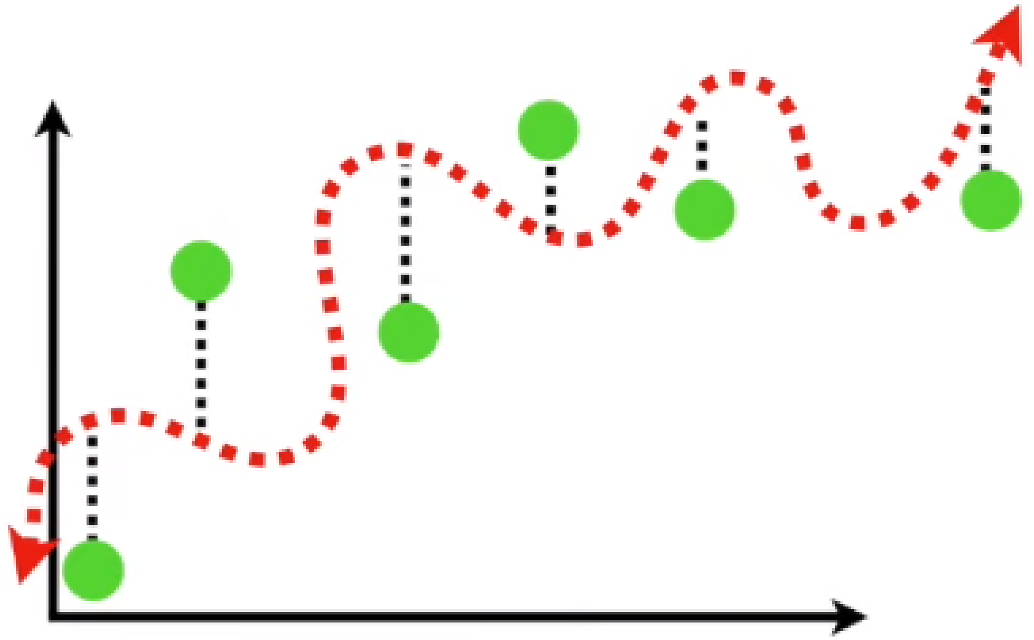
Even though **Squiggly Line** did a great job fitting the **training set**…



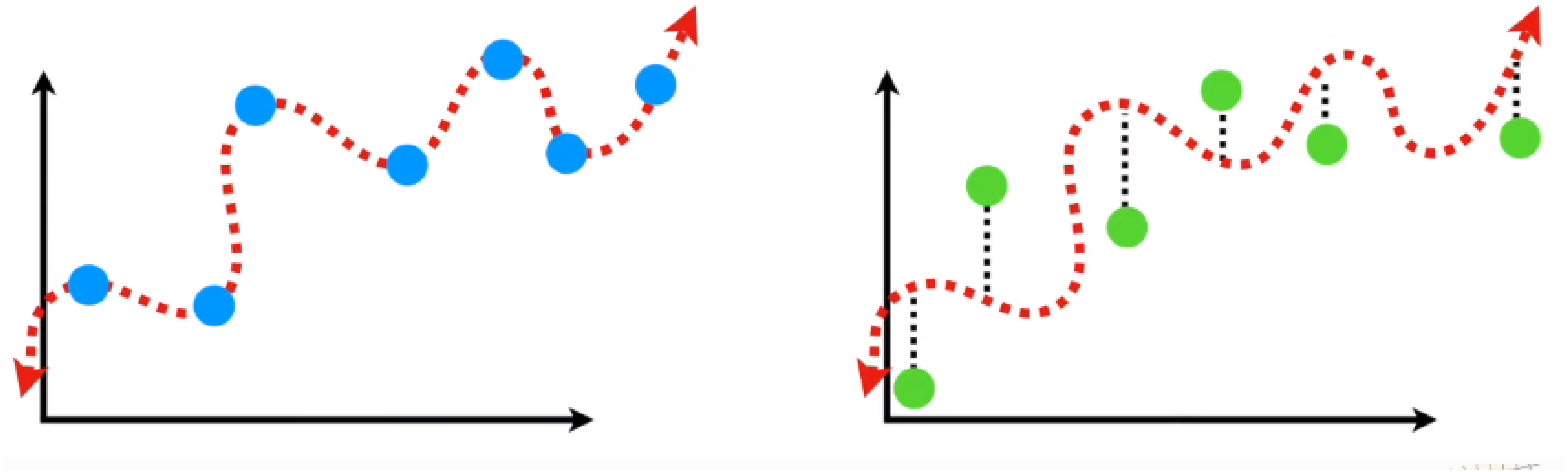




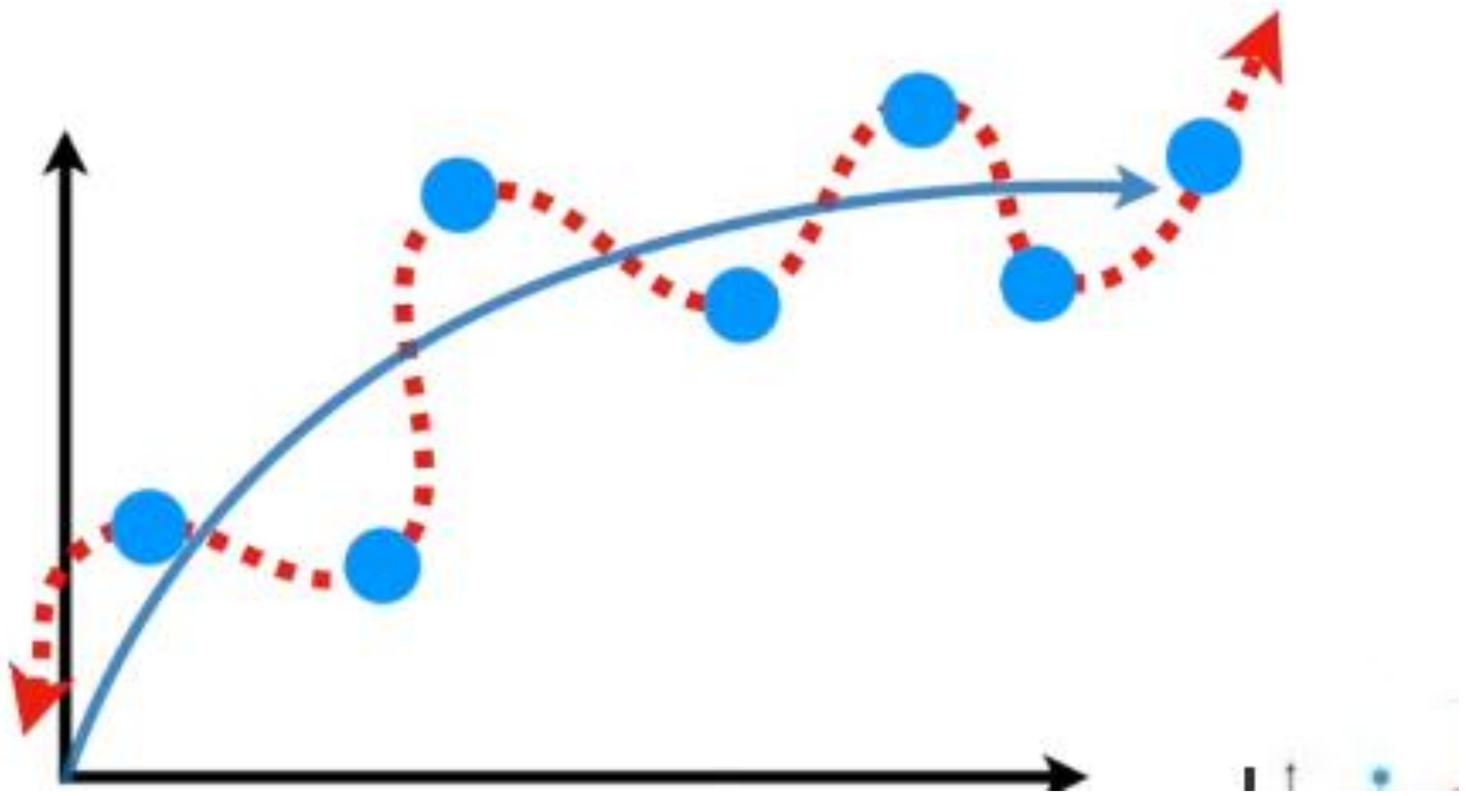
…it did a terrible job fitting the **testing set**…



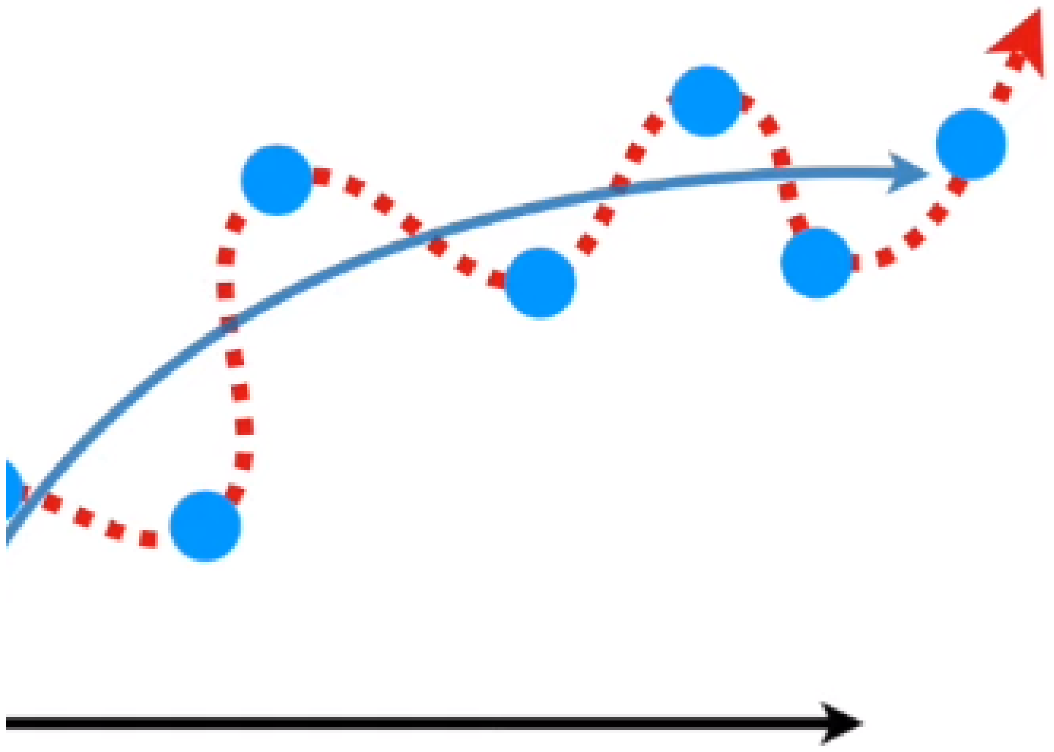
In Machine Learning lingo, the difference in fits between data sets is called **Variance**.

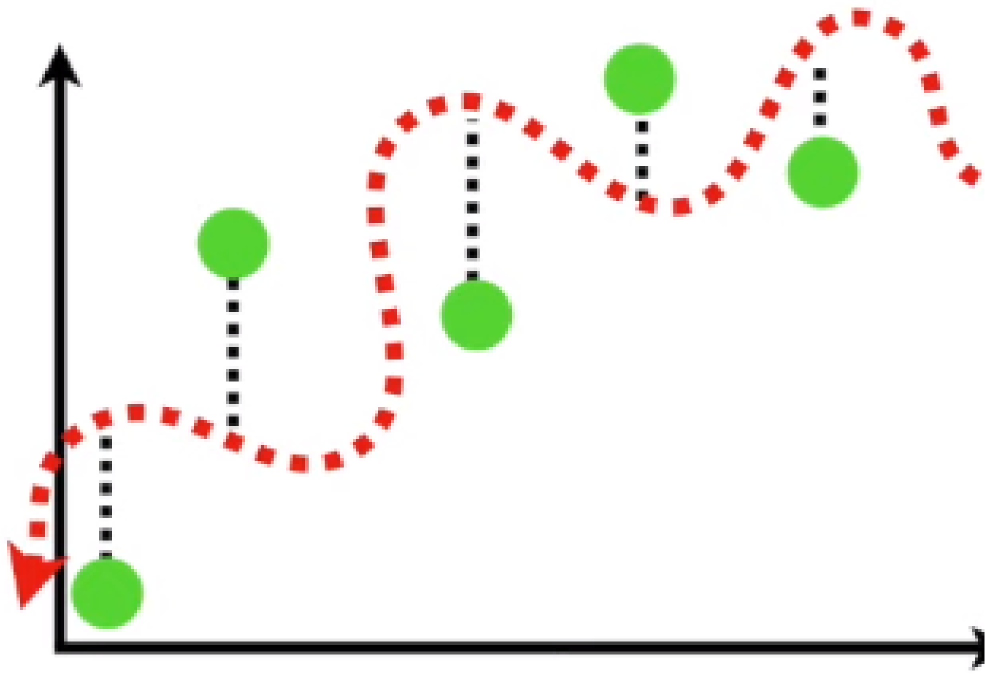


The **Squiggly Line** has **low bias**, since it is flexible and can adapt to the curve in the relationship between weight and hight…

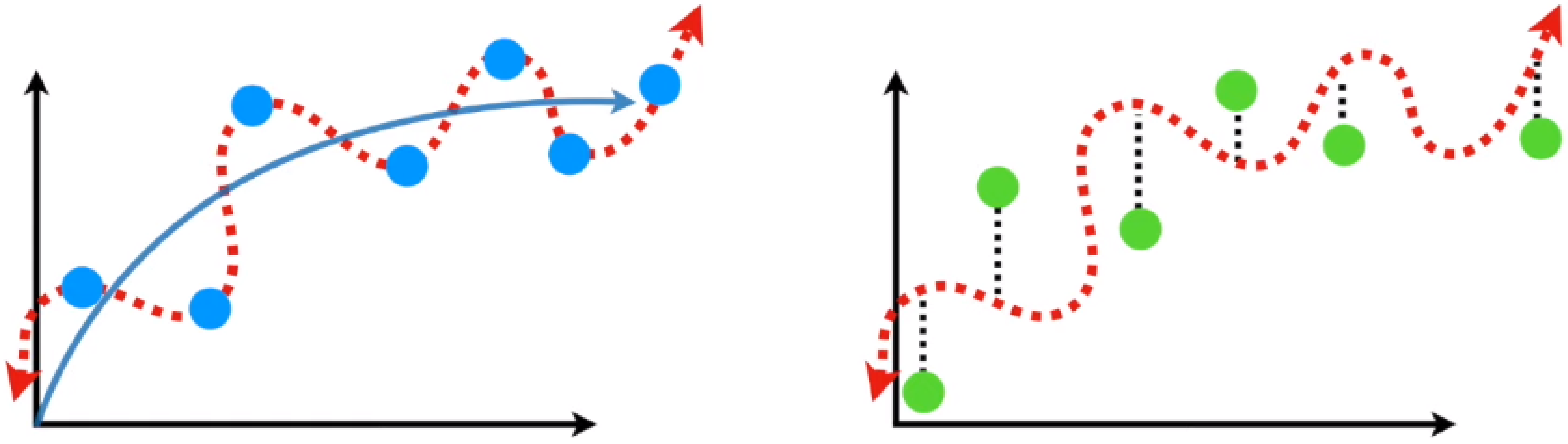


…but the **Squiggly Line** has **high variability**, because it results in vastly different Sums of Squares for different datasets.

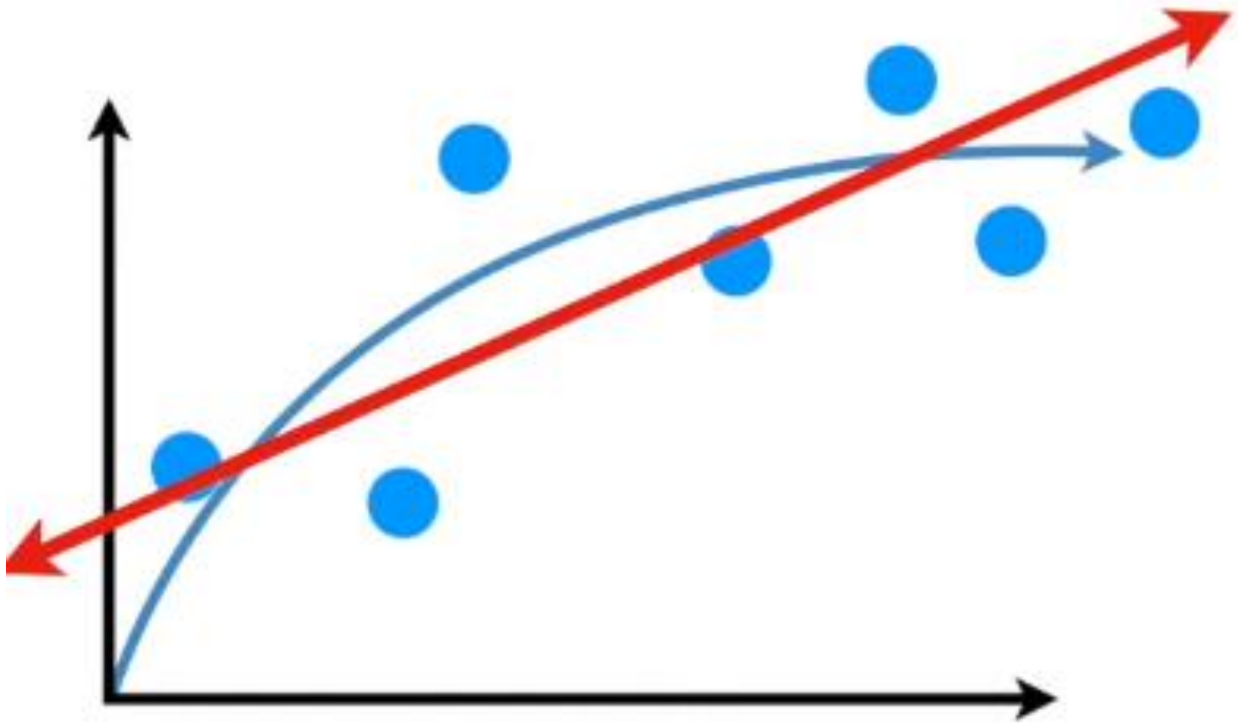




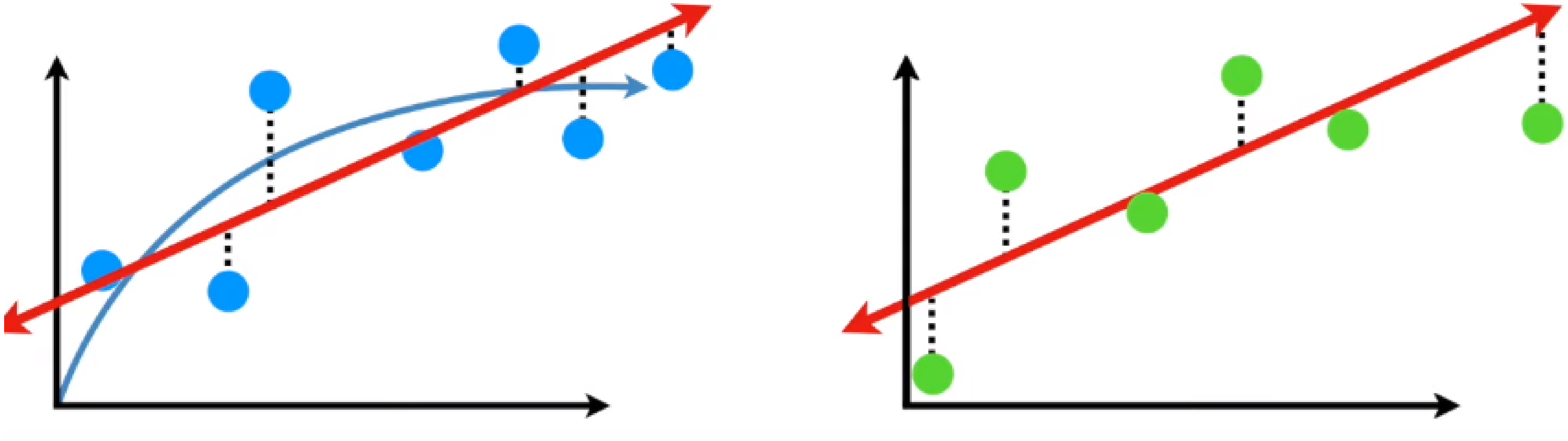
In other words, it's hard to predict how well the **Squiggly Line** will perform with future data sets. It might do well sometimes, and other times it might do terribly.



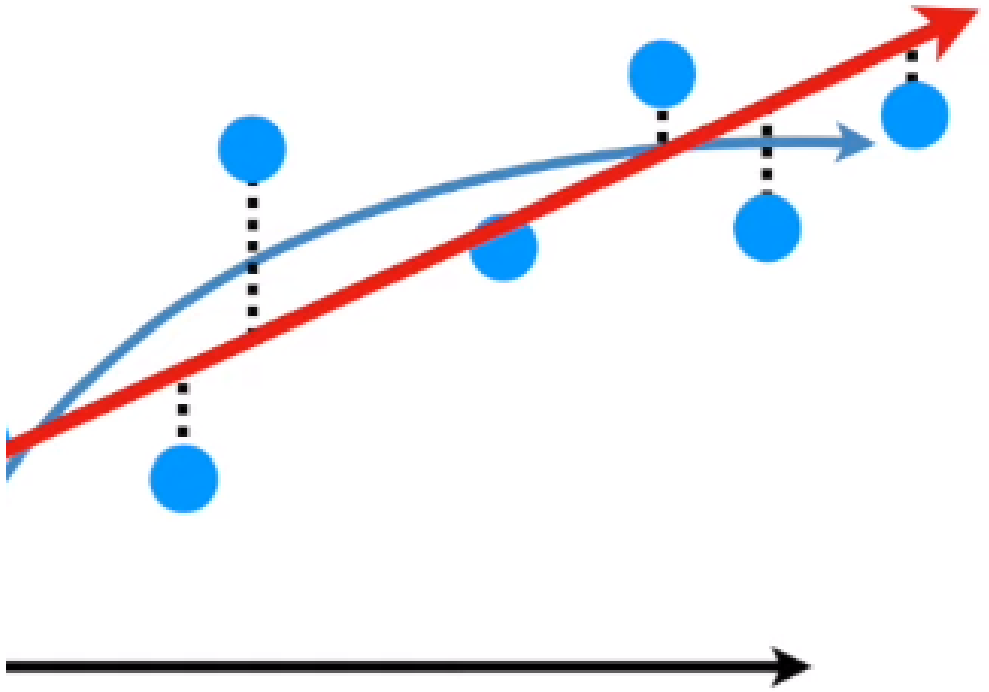
In contrast, the **Straight Line** has relatively **high bias**, since it can not capture the curve in the relationship between weight and height...

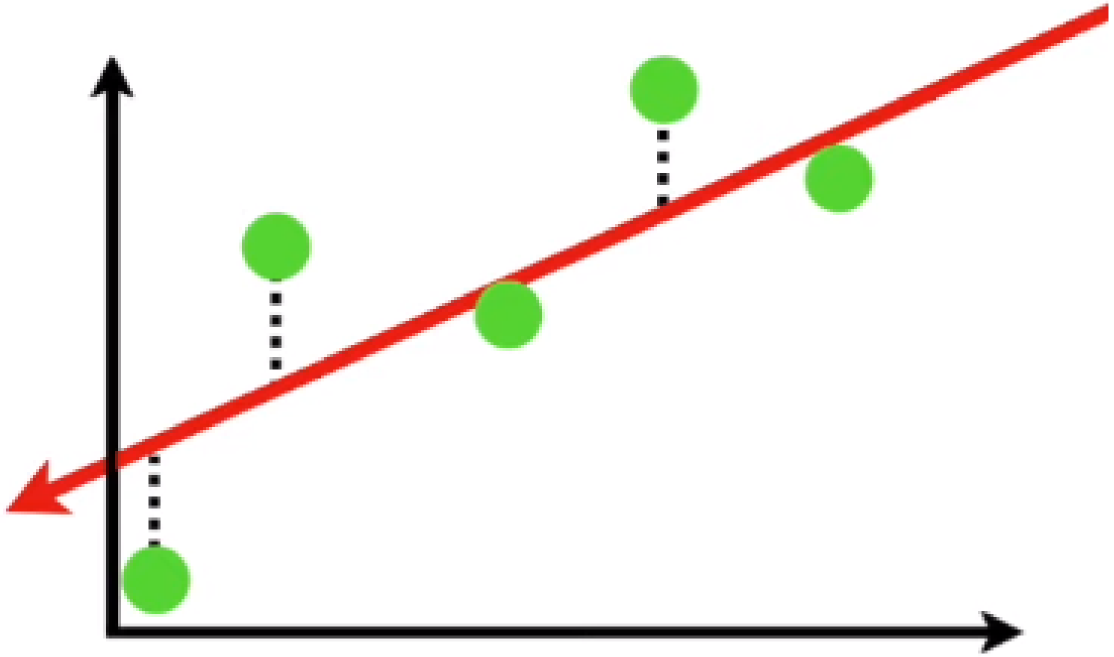


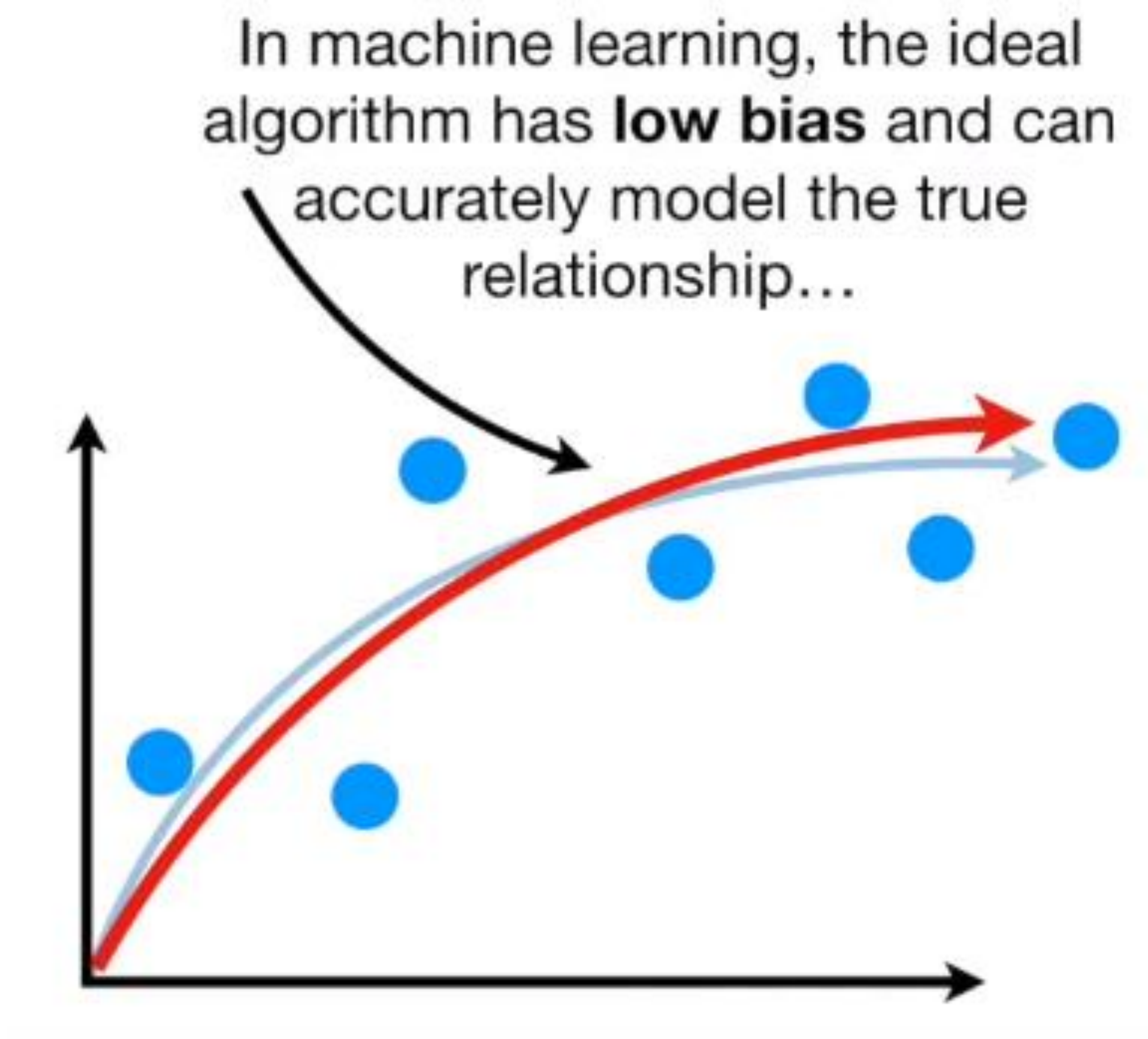
…but the **Straight Line** has relatively **low variance**, because the Sums of Squares are very similar for different datasets.



In other words, the **Straight Line** might only give good predictions, and not great predictions. But they will be consistently good predictions.

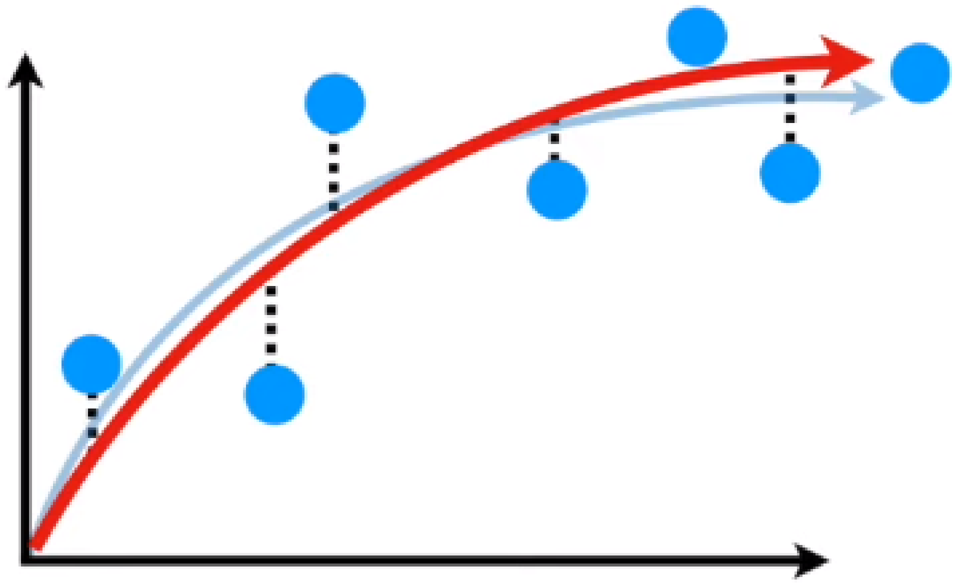


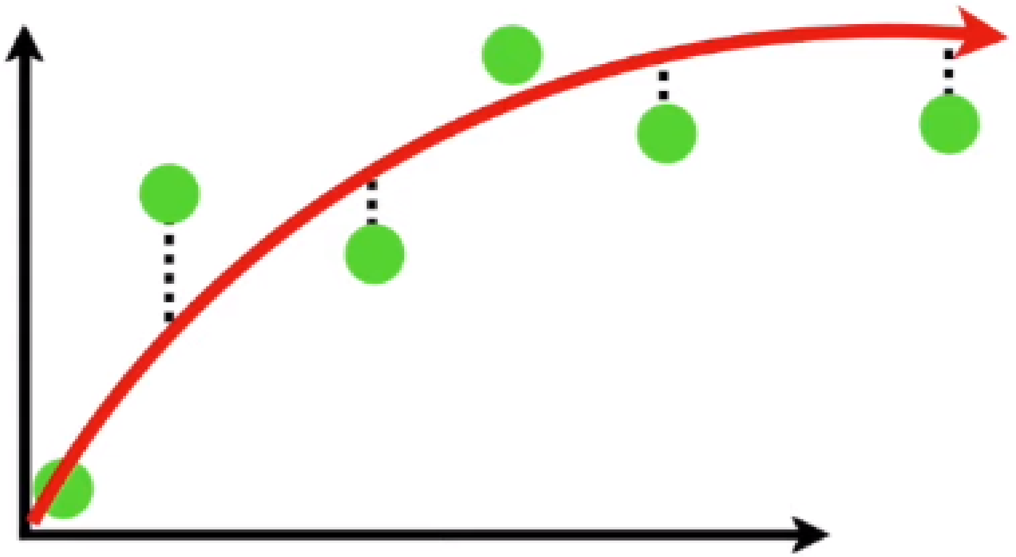




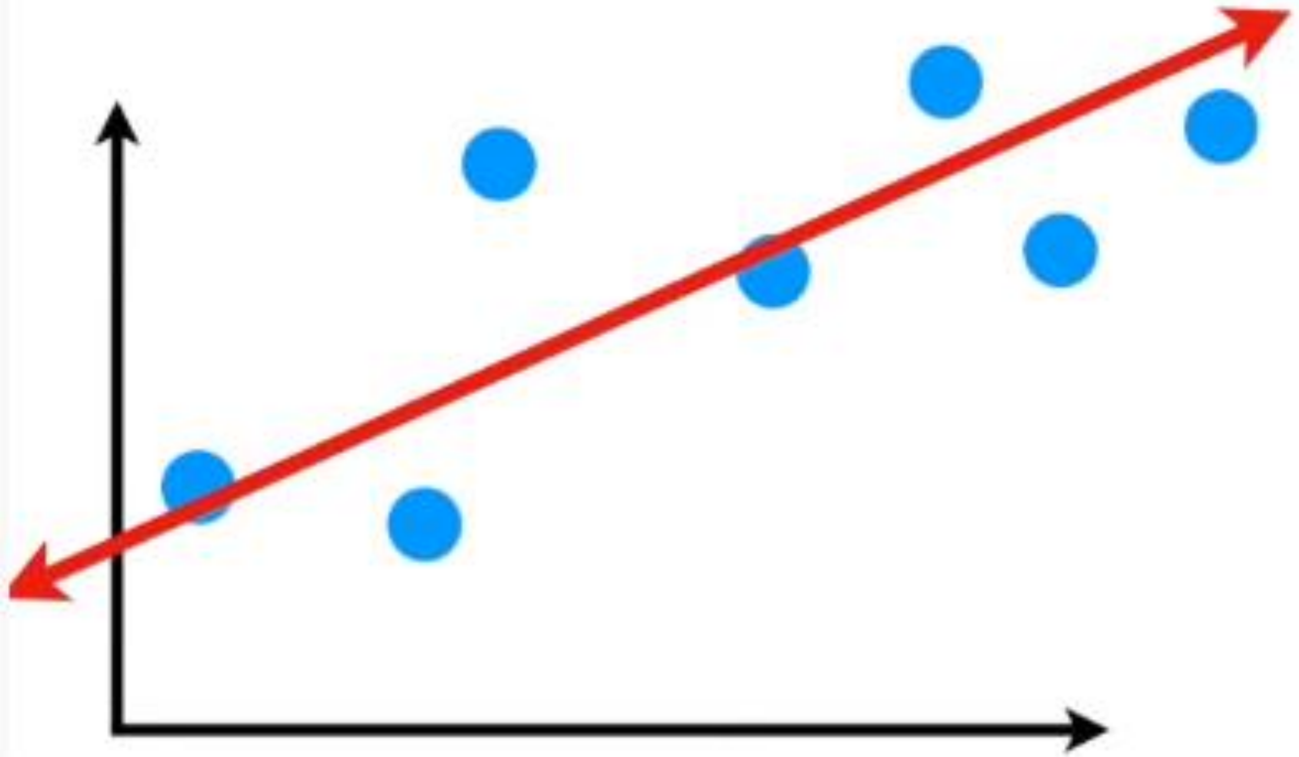
In machine learning, the ideal algorithm has **low bias** and can accurately model the true relationship…

…and it has **low variability**, by producing consistent predictions across different datasets.

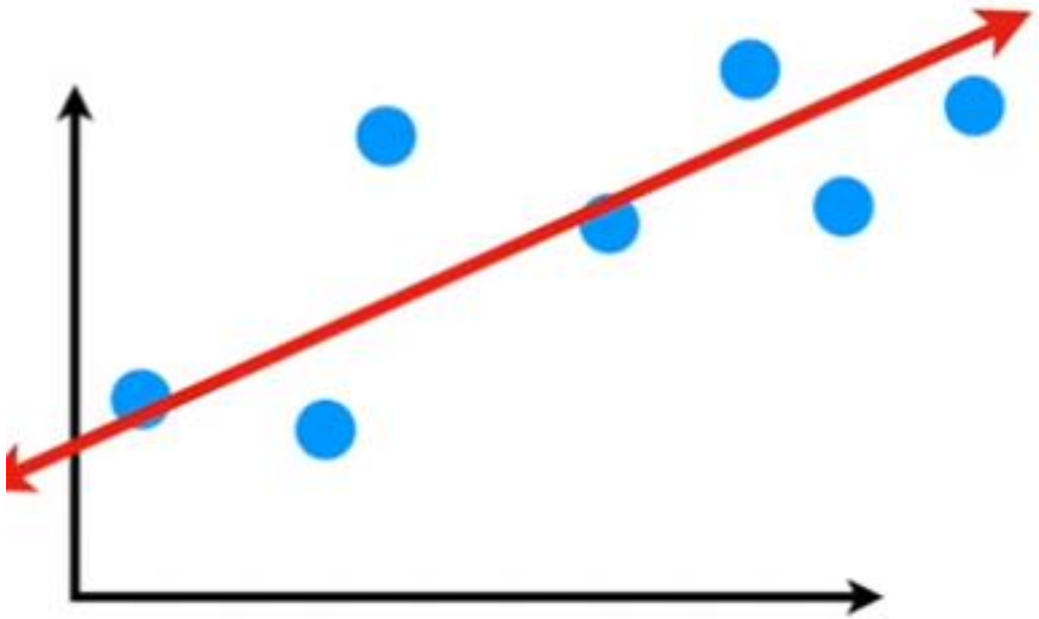


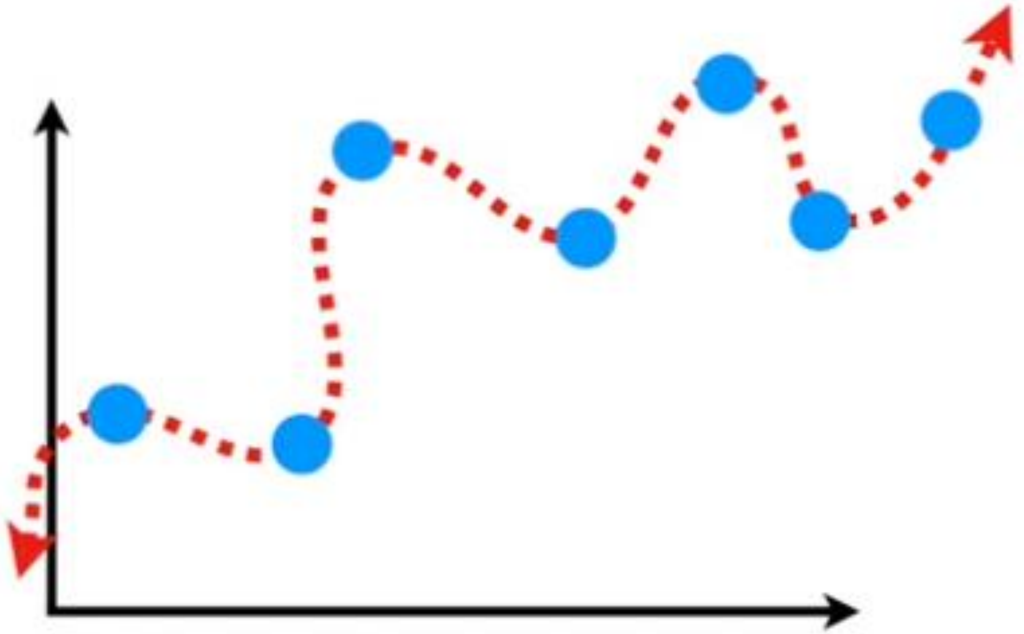


This is done by finding the sweet spot between a simple model…



…and a complex model.





How to know if the Bias is High or the Variance is High?

Training Error (Bias)	Low (e.g. 0.05)	High (e.g 0.15)	Low (e.g. 0.1)	High
Validation Set Error (Variance)	Low (e.g. 0.1)	Low (e.g. 0.16)	High (e.g. 25)	High
Comment	Good	High bias (Underfitting)	High variance (Overfitting)	Try to fix the underfitting problem first

- Underfitting (High Bias))
    - In ANN, use more hidden Units and/or more layers
    - More training epochs
    - Use better parameter values (e.g. the learning rate, and the alpha momentum)
    - In general use a more expressive ML model e.g.
        - A quadratic model instated of a linear model)
        - First order logic rules instead of propositional rules

- Overfitting (High Variance)
    - Increase the training data set (does not hurt the bias; not really a tradeoff)
    - Build an ensemble of classifiers (works for most ML methods)
    - In ANN, use regularization (e.g. L_2 reg. or Dropout) to simplify the network.
    - Use a less expressive ML model

- See
- https://www.youtube.com/watch?v=EuBBz3bI-aA

- Common methods for finding low bias and low variance
- Regularization in ANN.
- Finding the appropriate value for k in instance based learning.
- Building ensembles of classifiers.

Bias

- Low bias
  - linear regression applied to linear data
  - 2nd degree polynomial applied to quadratic data
  - ANN with many hidden units trained to completion
- High bias
  - constant function
  - linear regression applied to non-linear data
  - ANN with few hidden units applied to non-linear data

Variance

- Low variance
  - constant function
  - model independent of training data
  - model depends on stable measures of data
    - mean
    - median
- High variance
  - high degree polynomial
  - ANN with many hidden units trained to completion

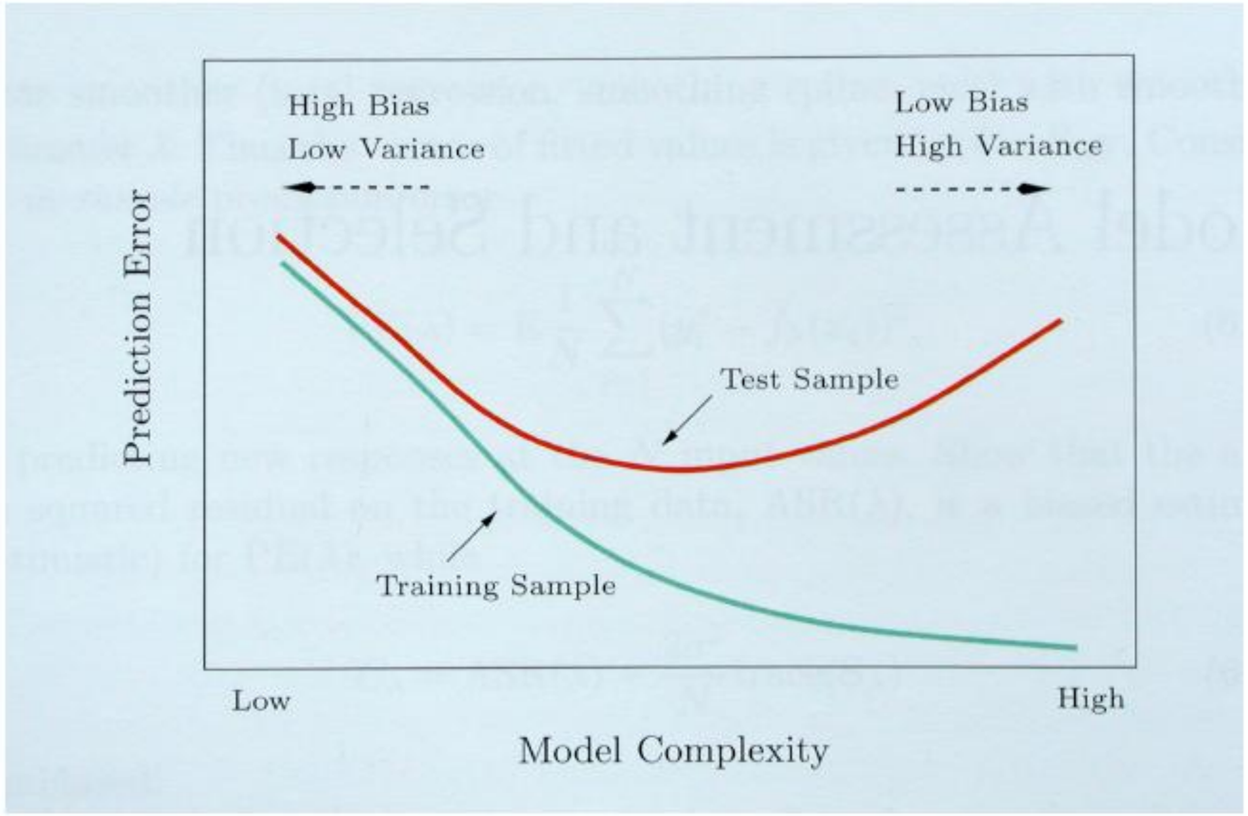
Sources of Variance in Supervised Learning

- noise in targets or input attributes
- bias (model mismatch)
- training sample
- randomness in learning algorithm
  - neural net weight initialization
- randomized subsetting of train set:
  - cross validation, train and early stopping set

Bias/Variance Tradeoff

- (bias +variance) is what counts for prediction
- Often:
  - low bias => high variance
  - low variance => high bias
- Tradeoff:
  - bias^2 vs. variance

Bias/Variance Tradeoff



Hastie, Tibshirani, Friedman "Elements of Statistical Learning" 2001

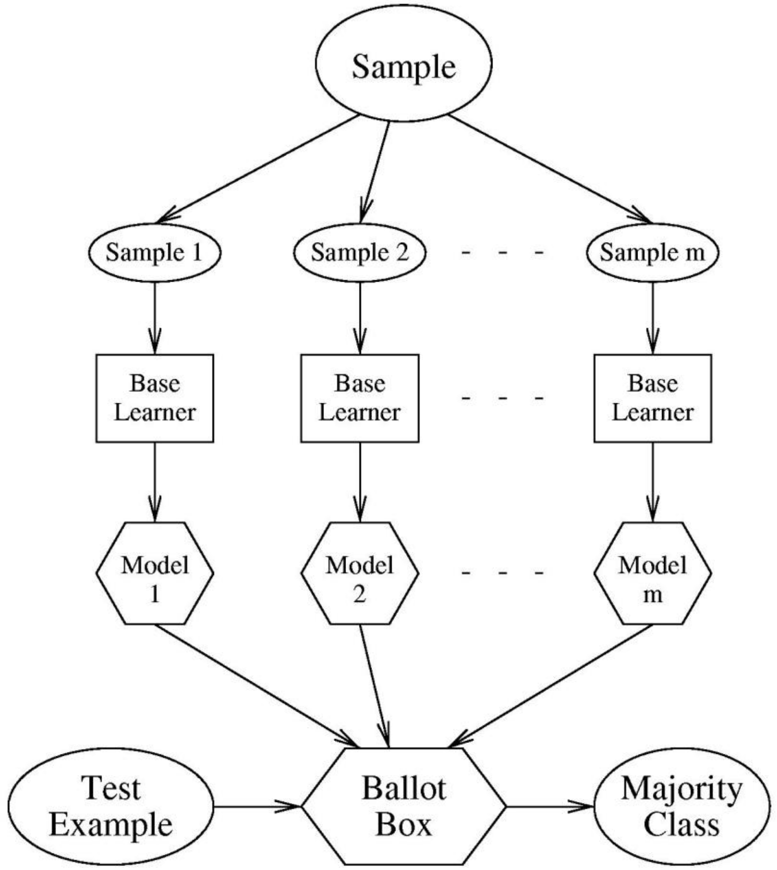
Reduce Variance Without Increasing Bias

- Averaging reduces variance:

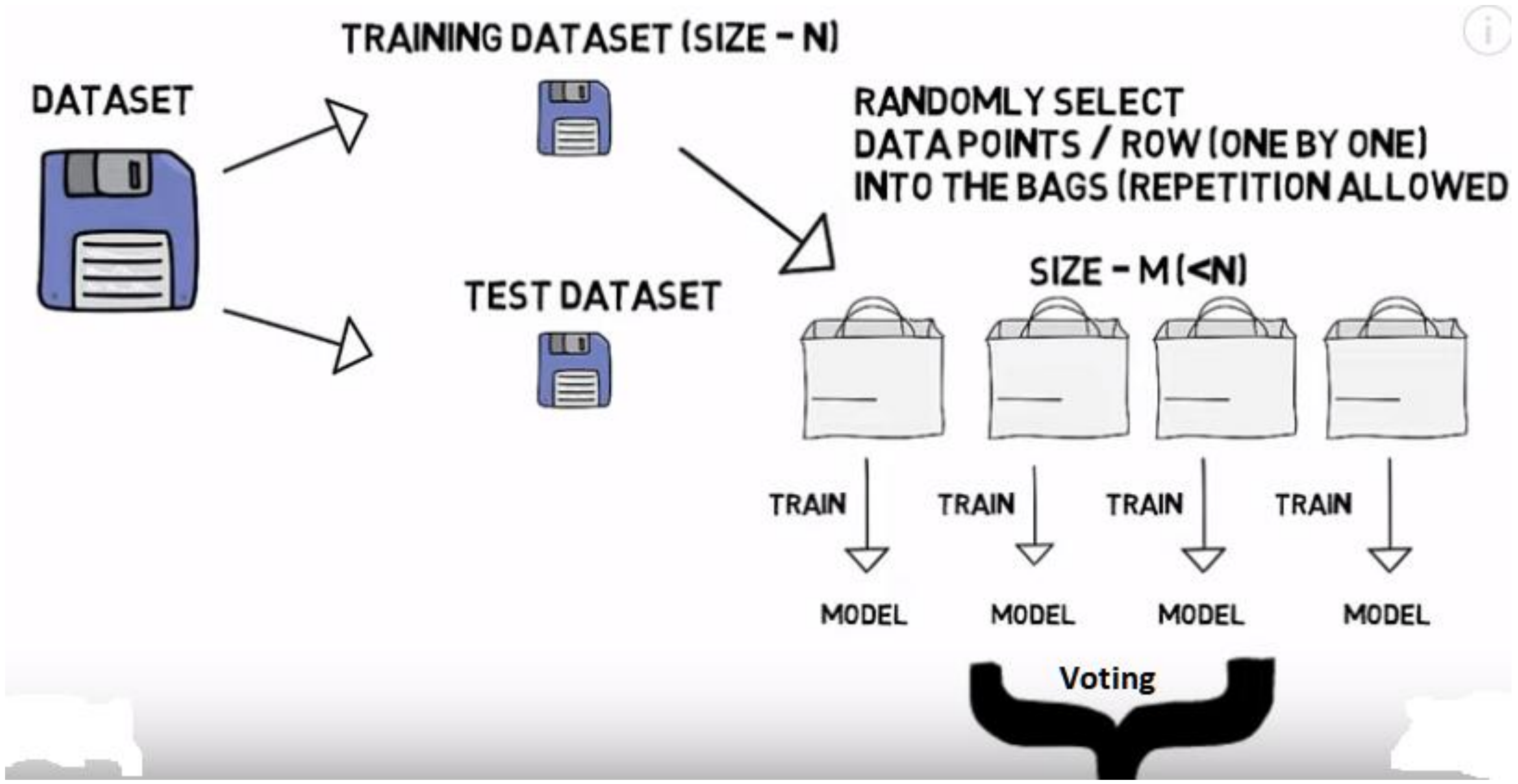
$$Var(\overline{X}) = \frac{Var(X)}{N}$$

- Average models to reduce model variance
- One problem:
  - only one train set
  - where do multiple models come from?

Bagging


Sample
Sample 1
Sample 2
Sample m
Base Learner
Base Learner
Base Learner
Model 1
Model 2
Model m
Test Example
Ballot Box
Majority Class

Bagging


TRAINING DATASET (SIZE - N)
DATASET
RANDOMLY SELECT
DATA POINTS / ROW (ONE BY ONE)
INTO THE BAGS (REPETITION ALLOWED
TEST DATASET
SIZE - M (<N)
TRAIN
TRAIN
TRAIN
TRAIN
MODEL
MODEL
MODEL
MODEL
Voting

Bagging: Bootstrap Aggregation

- Leo Breiman (1994)
- Bootstrap Sample:
  - draw sample of size |D| with replacement from D

$$\text{Train } L_i\left(BootstrapSample_i(D)\right)$$

$$\text{Regression}: L_{bagging} = \overline{L_i}$$

$$\text{Classification}: L_{bagging} = Plurality\left(L_i\right)$$

Bagging

- Best case:

$$Var(Bagging(L(x, D))) = \frac{Variance(L(x, D))}{N}$$

- In practice:
  - models are correlated, so reduction is smaller than 1/N
  - variance of models trained on fewer training cases usually somewhat larger
  - stable learning methods have low variance to begin with, so bagging may not help much

Bagging Results

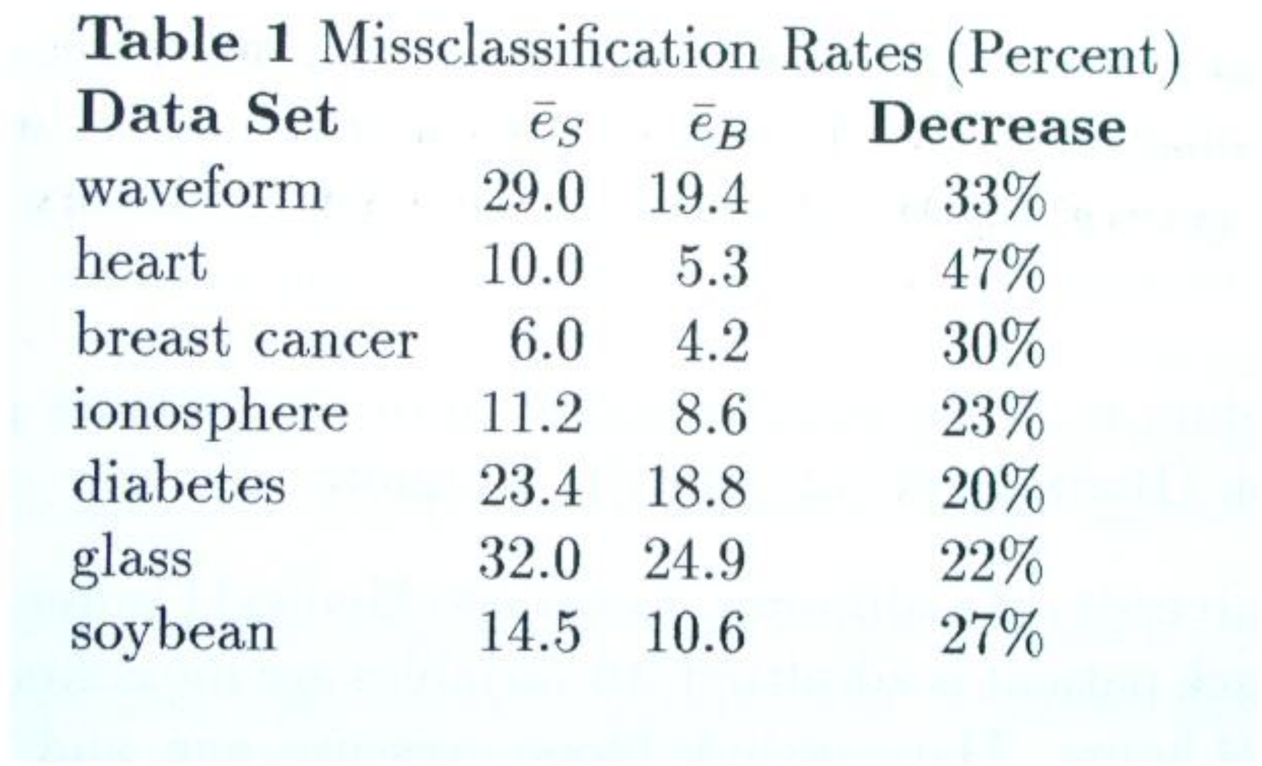


Table 1 Missclassification Rates (Percent)

Data Set	$\bar{e}_S$	$\bar{e}_B$	**Decrease**
waveform	29.0	19.4	33%
heart	10.0	5.3	47%
breast cancer	6.0	4.2	30%
ionosphere	11.2	8.6	23%
diabetes	23.4	18.8	20%
glass	32.0	24.9	22%
soybean	14.5	10.6	27%

Breiman "Bagging Predictors" Berkeley Statistics Department TR#421, 1994

How Many Bootstrap Samples?

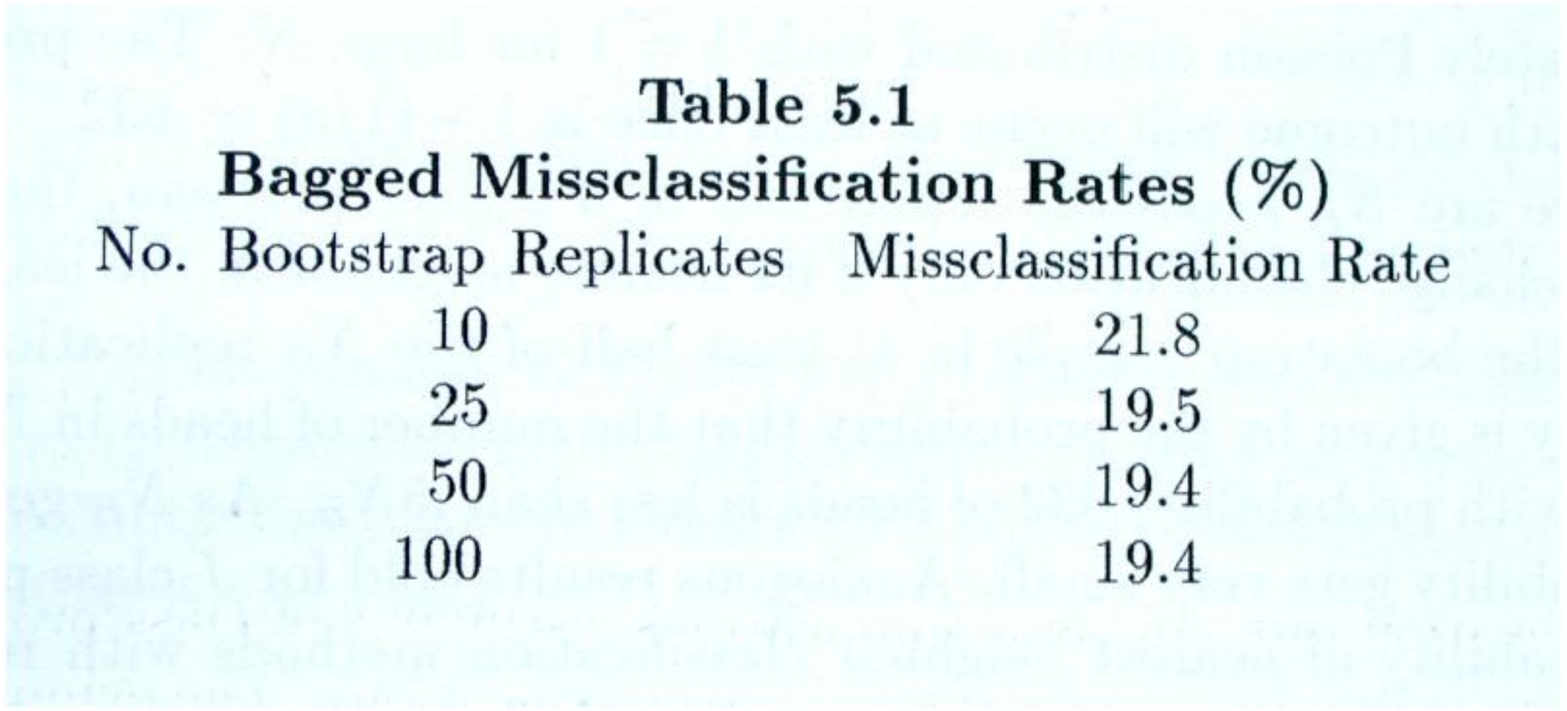


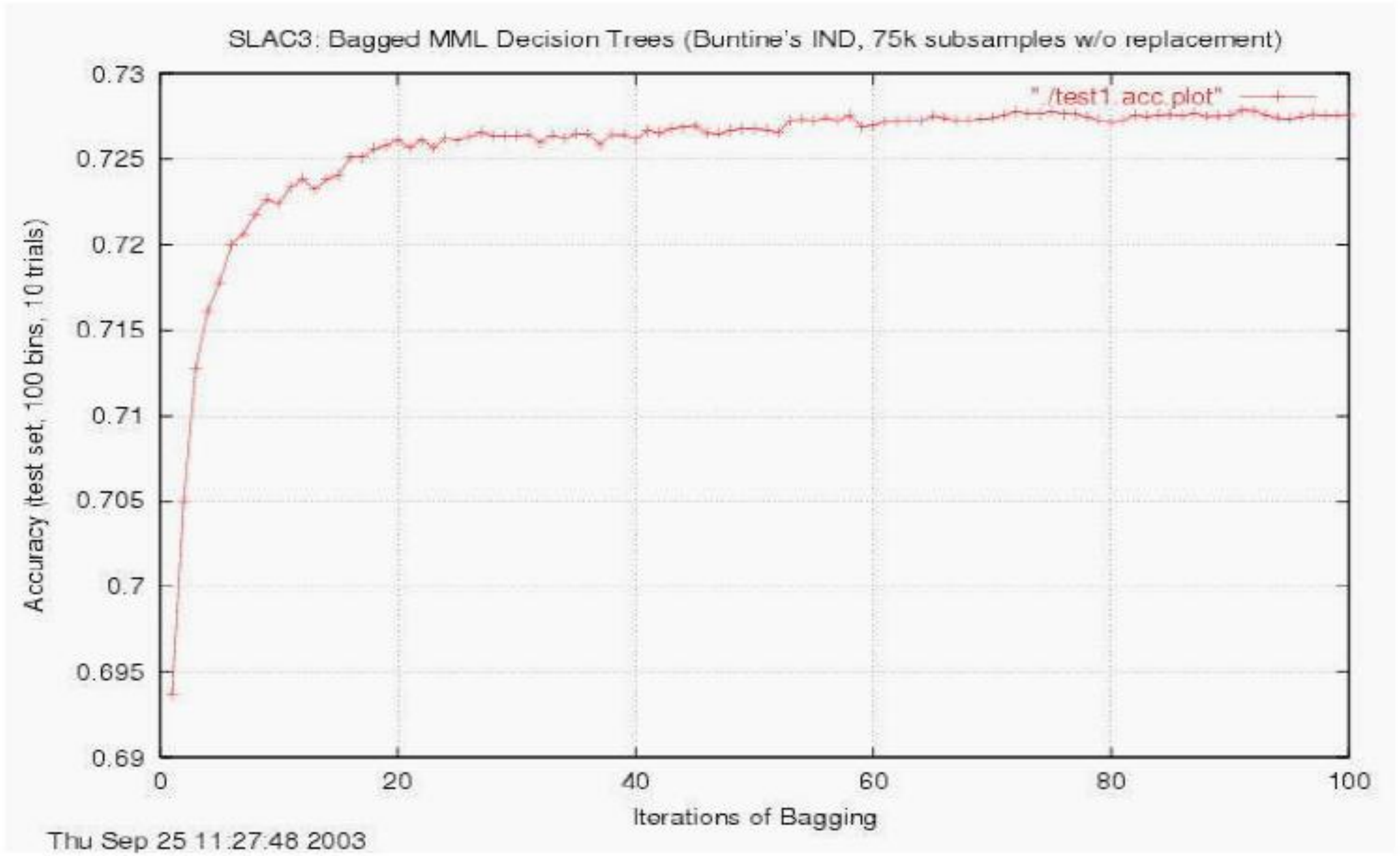
Table 5.1

Bagged Missclassification Rates (%)

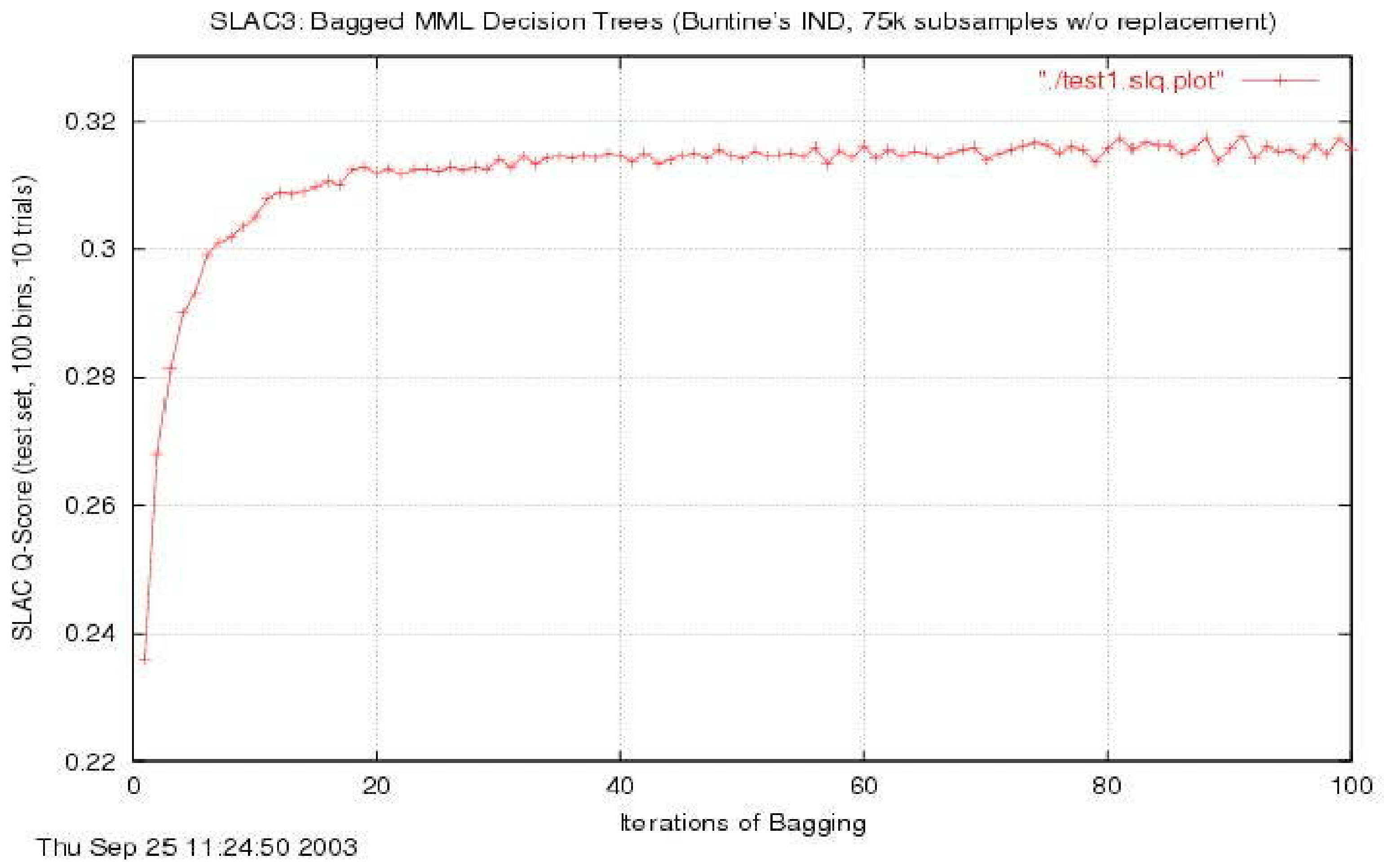
No. Bootstrap Replicates	Missclassification Rate
10	21.8
25	19.5
50	19.4
100	19.4

Breiman "Bagging Predictors" Berkeley Statistics Department TR#421, 1994

More bagging results



More bagging results



Bagging with cross validation

- Train neural networks using 4-fold CV
  - Train on 3 folds earlystop on the fourth
  - At the end you have 4 neural nets

- How to make predictions on new examples?
  - Train a neural network until the mean earlystopping point
  - Average the predictions from the four neural networks

Can Bagging Hurt?

- Each base classifier is trained on less data
  - Only about 63.2% of the data points are in any bootstrap sample
- However the final model has seen all the data
  - On average a point will be in >50% of the bootstrap samples

It can hurt in some cases

- Noise Amplification: if the data set is noisy bagging may amplify the effect of noise.
- Overfitting in Small datasets:
  - in this case the bootstrap samples used to train individual models may not be diverse enough.
  - This can lead to a scenario where all models in the ensemble overfit in a similar way, thereby not effectively reducing overfitting or improving model generalization.
- Degradation of Performance in Imbalanced Datasets
  - If one class is significantly underrepresented, the process of creating bootstrap samples might produce subsets with even poorer representation of the minority class.
  - This can lead to models that are biased towards the majority class, thereby reducing the ensemble's ability to accurately predict minority class instances.

Random Forest

- **Builds upon the concept of bagging but specifically applies it to decision trees**

1. **Bootstrap Sampling:** Like bagging, Random Forest creates multiple bootstrap samples from the original training data. Each sample is used to train a separate decision tree, leading to a "forest" of trees.
2. **Feature Randomness:** In addition to bootstrapping samples, Random Forest introduces another layer of randomness by selecting a random subset of features at each split in the decision tree. This ensures that the trees in the forest are de-correlated, making the ensemble less prone to overfitting and more robust to variance in the data. For example for a dataset with 100 features instead of evaluating all 100 features to determine the best split, we might randomly select 10 features and only consider those for splitting.
3. **Aggregation of Predictions:** Once the forest of trees is grown, Random Forest aggregates the predictions of all trees to form the final prediction. For classification tasks, this typically involves a majority voting mechanism, where the class that gets the most votes from individual trees is chosen as the final prediction. For regression tasks, it usually involves averaging the predictions from all trees.

Reduce Bias and Decrease Variance?

- Bagging reduces variance by averaging
- Bagging has little effect on bias
- Can we average *and* reduce bias?
- Yes:

Boosting

Boosting

- Freund & Schapire:
  - theory for “weak learners” in late 80’s
- Weak Learner: performance on ***any*** train set is slightly better than chance prediction
- intended to answer a theoretical question, not as a practical way to improve learning
- tested in mid 90’s using not-so-weak learners
- works anyway!

Boosting

- Weight all training samples equally
- Train model on train set
- Compute error of model on train set
- Increase weights on train cases model gets wrong
- Train new model on re-weighted train set
- Re-compute errors on weighted train set
- Increase weights again on cases model gets wrong
- Repeat until tired (100+ iteraations)
- Final model: weighted prediction of each model

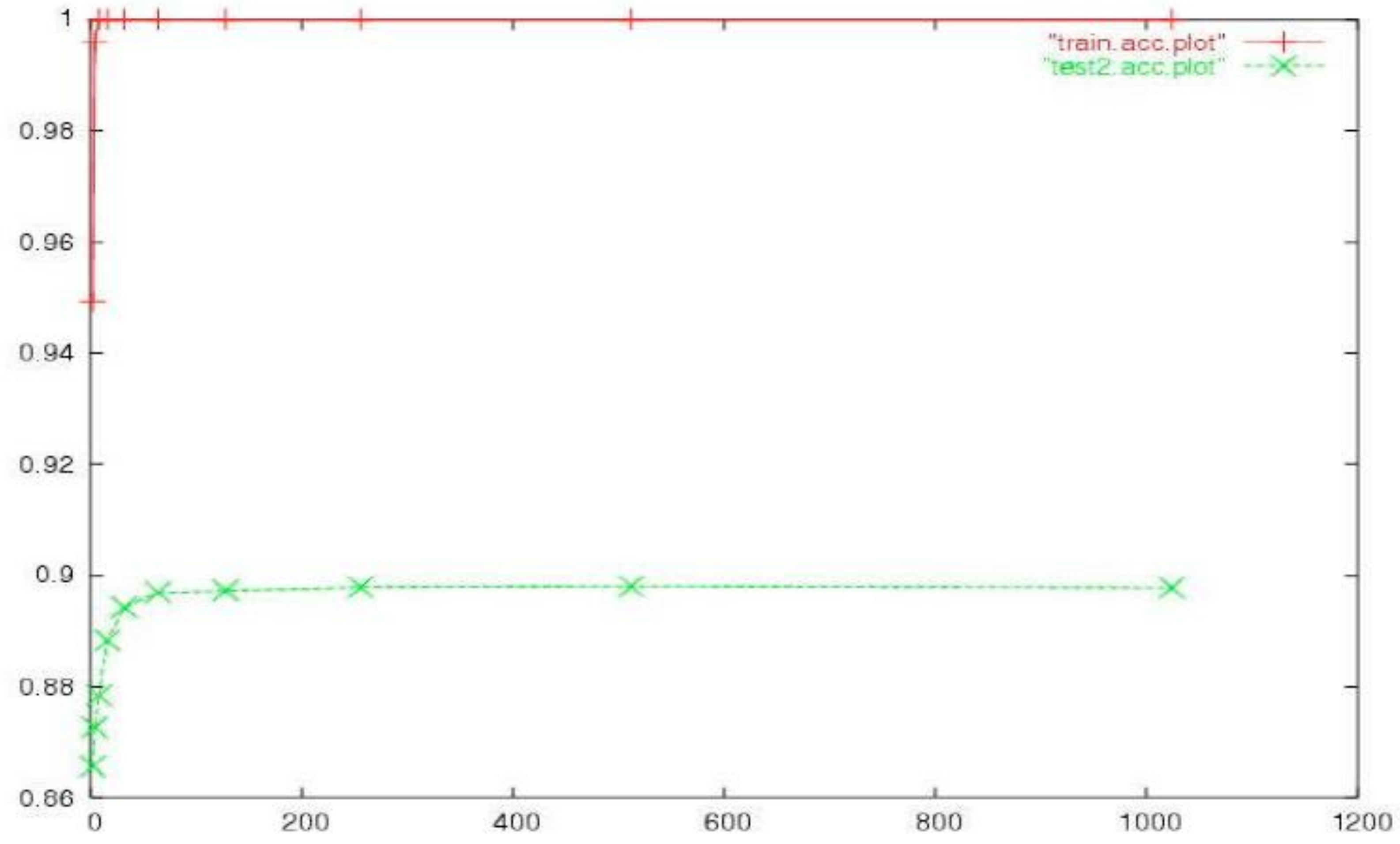
AdaBoost Algorithm Steps

1. **Initialize Weights:** Start with equal weights for all instances in the training dataset. If there are N instances, each instance i gets an initial weight $w_i = 1/N$.
2. **For each iteration t in 1 to T:**
   - **a. Train a Weak Learner:** Use the training data to fit a weak model. The weak model can be any classifier that performs slightly better than random guessing. Decision stumps (one-level decision trees) are commonly used.
   - **b. Calculate Error:** Compute the weighted error rate (err_t) of the weak model on the training data. The error is calculated based on the current weights of the instances, emphasizing the model's performance on instances that are harder to predict.
   - **c. Compute Model Weight:** Calculate the weight (α_t) of this weak model, which is a measure of its importance in the final prediction. It's usually set to $\alpha_t = 0.5 \times \log((1 - err_t)/err_t)$. Models with lower error rates get higher weights.
   - **d. Update Instance Weights:** Increase the weights of incorrectly predicted instances and decrease the weights of correctly predicted instances. This makes the algorithm focus more on the hard-to-predict instances in the next iteration. The update rule is $w_i \leftarrow w_i \times \exp(\alpha_t \times I(y_i \neq \hat{y}_i))$, where I is the indicator function that is 1 if the condition is true and 0 otherwise.
   - **e. Normalize Weights:** Adjust the instance weights so that the total sum of weights equals 1. This step ensures that the weights are properly normalized to form a probability distribution.
3. **Combine the Weak Models:** After T iterations, aggregate the weak models using a weighted majority vote. The final prediction is made by summing the weights (α_t) of the weak models where the prediction is positive and subtracting the weights where the prediction is negative. The sign of the resulting sum (positive or negative) determines the class label.

Reweighting vs Resampling

- Example weights might be harder to deal with
  - Some learning methods can’t use weights on examples
  - Many common packages don’t support weighs on the train
- We can resample instead:
  - Draw a bootstrap sample from the data with the probability of drawing each example is proportional to it’s weight
- Reweighting usually works better but resampling is easier to implement

Boosting Performance



Boosting vs. Bagging

- Bagging doesn’t work so well with stable models. Boosting might still help.

- Boosting might hurt performance on noisy datasets. Bagging doesn’t have this problem

Boosting vs. Bagging

- On average, boosting helps more than bagging, but it is also more common for boosting to hurt performance.
- The weights grow exponentially.
- Bagging is easier to parallelize.

Some Boosting Variants

1. **Gradient Boosting**
   - **Description:** Gradient Boosting is a generalization of boosting to arbitrary differentiable loss functions. It builds models sequentially, each new model correcting errors made by previously trained models. Models are added to the ensemble based on the gradient of the loss function, which represents the direction of greatest improvement.

2. **XGBoost (eXtreme Gradient Boosting)**
   - **Description:** XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solves many data science problems in a fast and accurate way.
3. **LightGBM (Light Gradient Boosting Machine)**
   - **Description:** LightGBM is a gradient boosting framework that uses tree-based learning algorithms. It's designed for distributed and efficient training, especially for high-dimensional data. LightGBM improves on the traditional gradient boosting method by using a novel tree algorithm called Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB), which significantly increases the efficiency of model training.

4. **CatBoost (Categorical Boosting)**
   - **Description:** CatBoost is an open-source gradient boosting library that provides high-performance implementation of gradient boosting on decision trees, with a special emphasis on working with categorical data efficiently.

Stacking

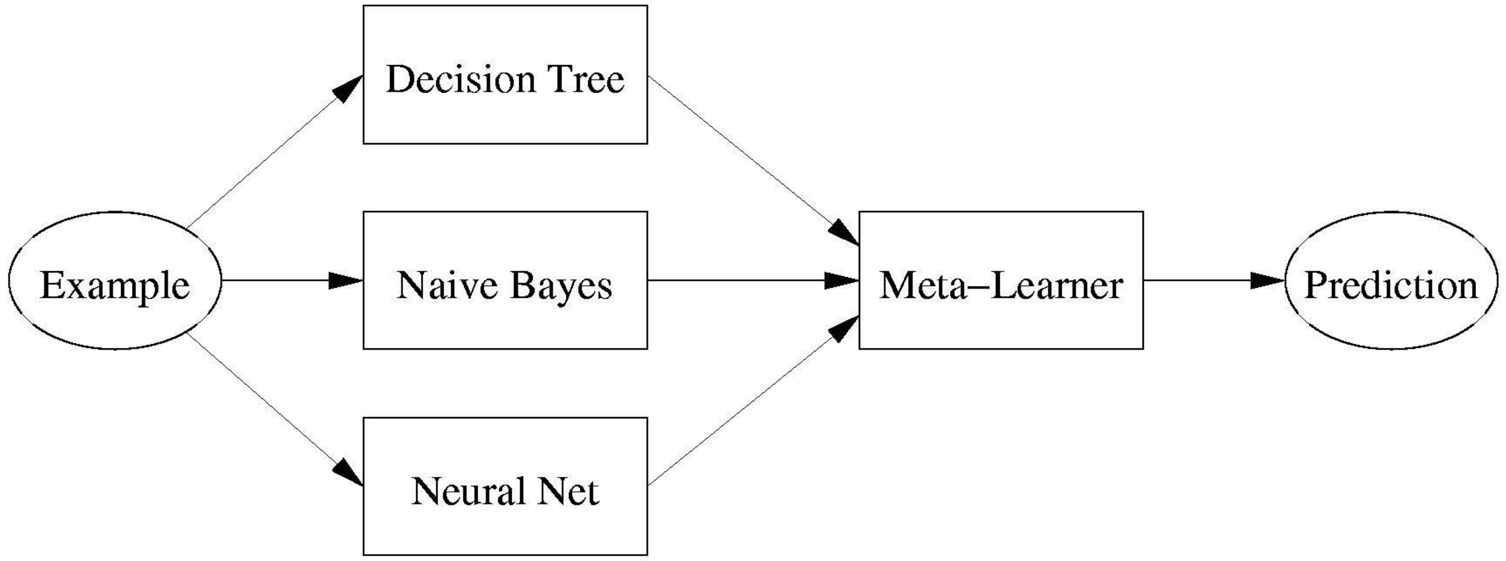
- used to combine multiple different models
- use a meta-learner (or a blender) to learn how to best combine the predictions from several base models.
- This approach leverages the strengths of various models

How Stacking Works

- **Train Base Models:**
  - Train multiple base models. These models can be of different types (e.g., decision trees, support vector machines, neural networks) and are trained on the entire training dataset or through a technique like cross-validation to generate out-of-sample predictions.
- **Generate Meta-Features:**
  - Once the base models are trained, they are used to make predictions on a separate dataset (this could be a validation set or the same training set used in a cross-validated manner). The predictions from the base models serve as the meta-features for the next step. The idea is that these meta-features represent the input from various models' perspectives on the data.
- **Train Meta-Learner:**
  - A meta-learner (another model) is then trained on the meta-features. The target is still the original target variable, but now the features are the predictions from the base models. The meta-learner's job is to learn the best way to combine these predictions to make a final prediction.
- **Final Prediction:** For new data, predictions from the base models are first generated and then fed into the meta-learner, which makes the final prediction.

Stacking

- Apply multiple base learners
  (e.g.: decision trees, naive Bayes, neural nets)
- Meta-learner: Inputs = Base learner predictions
- Training by leave-one-out cross-validation:
  Meta-L. inputs = Predictions on left-out examples



Two methods for Stacking

1. Input features of the meta-classifier = the output of the base learners
2. Or the input of the meta-classifier = probability of the classes as predicted by the base learners.

- Of course a validation set is used to for the training data of the meta-classifier
- The base learners are used to classify the validation set.
- The output of the classifiers (the classes or their probabilities) form the input features of the meta-classifier
- To classify a new instance it is first classified by the base learners to form the input vector for the meta-classifier.
- The meta-classifier is then used to classify the vector of classes (or the probabilities) as produced by the base learners.