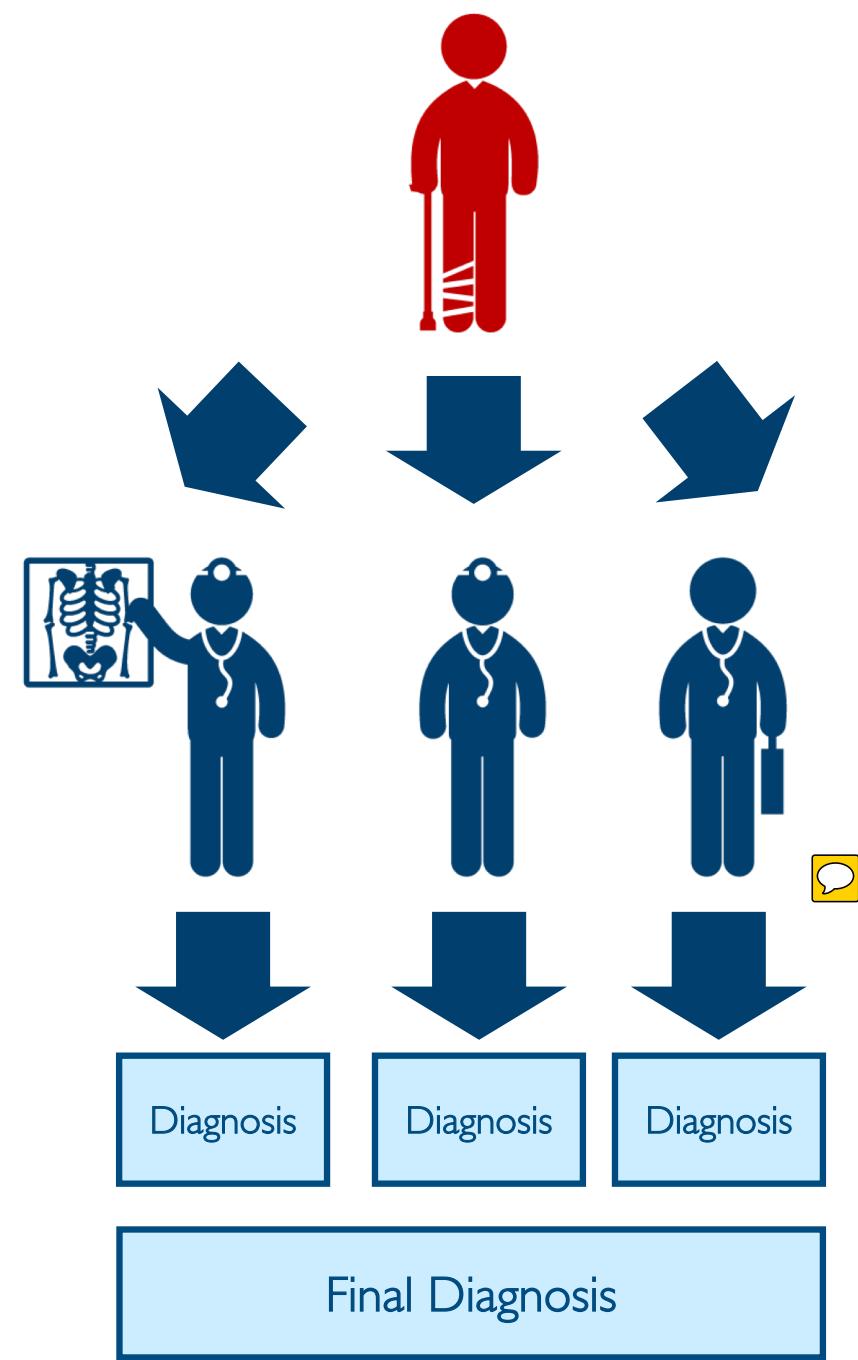
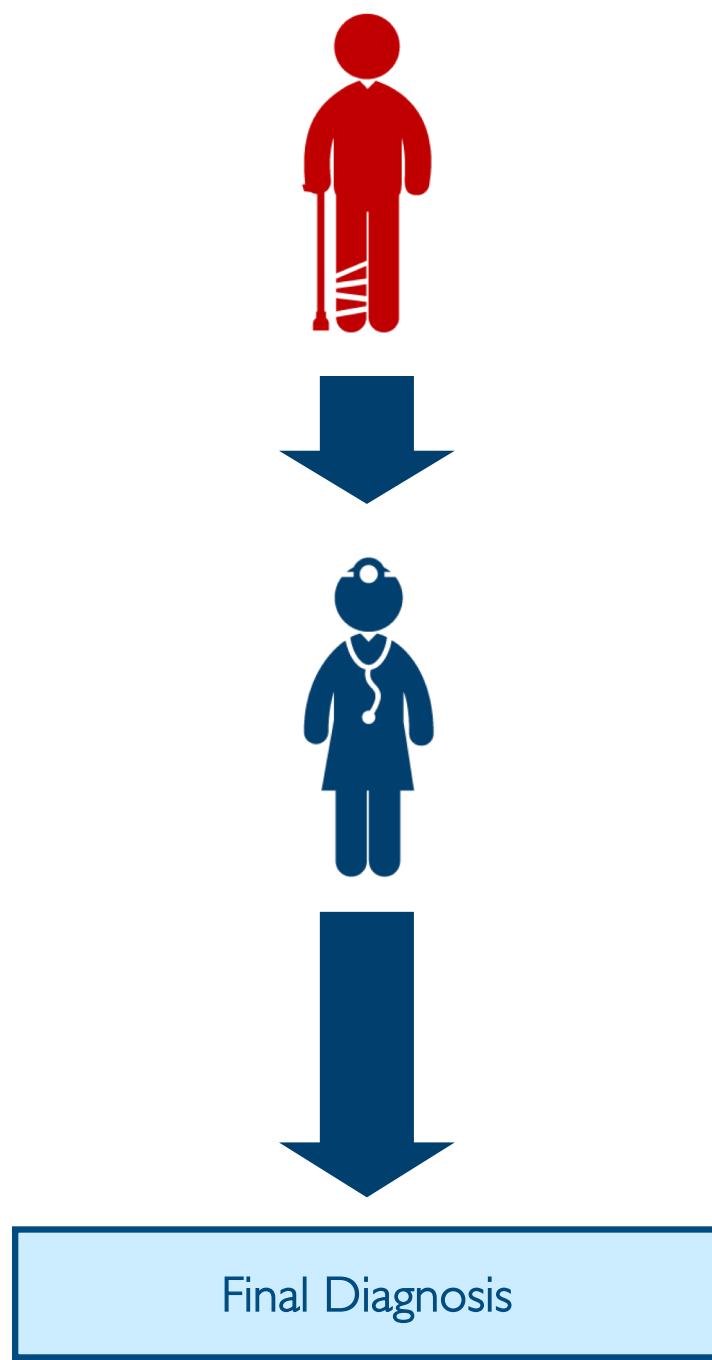




# Classification Ensembles

Master in Analytics and Business Intelligence – Machine Learning

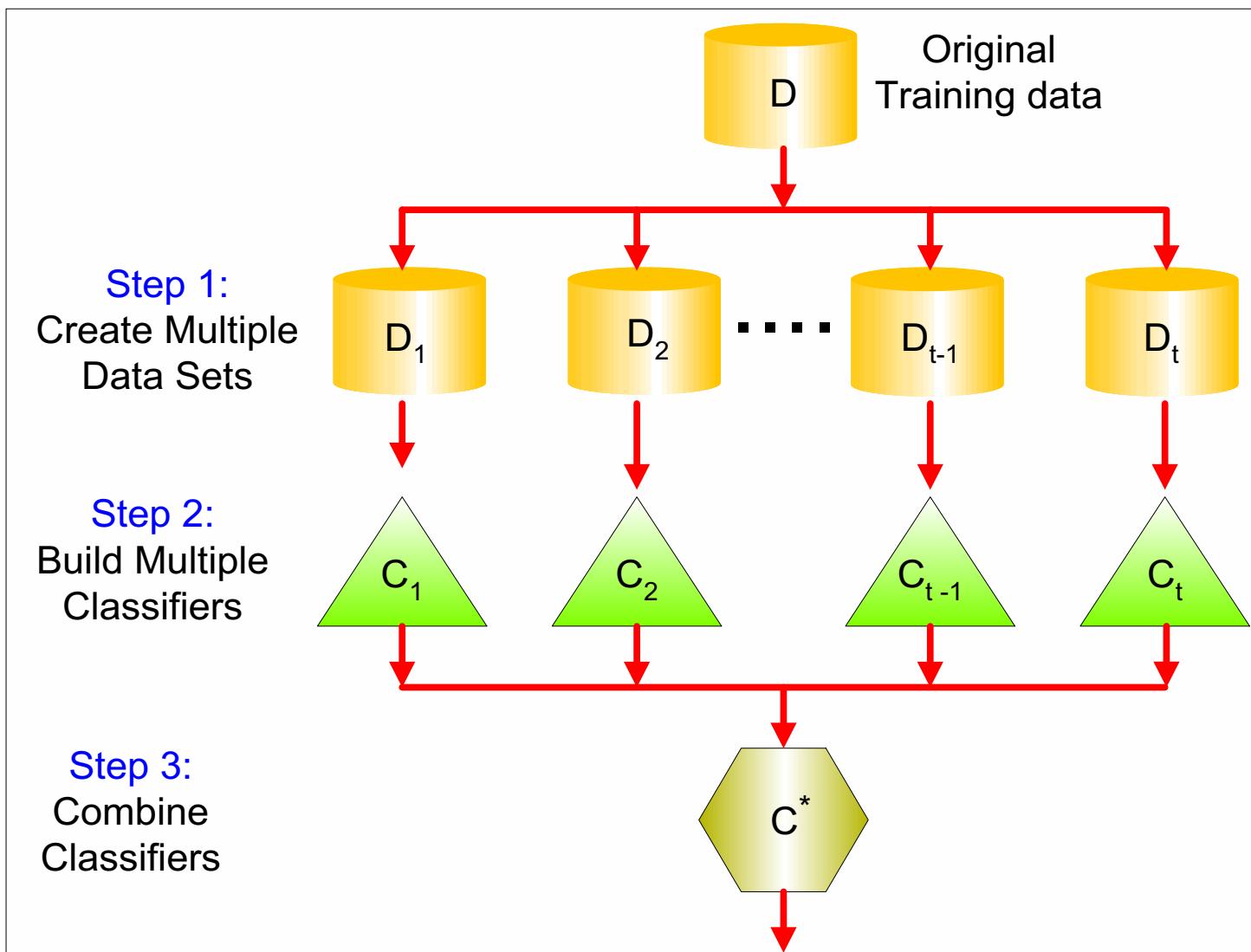
# What are Ensemble Methods?



# Ensemble Methods

Generate a set of classifiers from the training data

Predict class label of previously unseen cases by aggregating predictions made by multiple classifiers



- Basic idea
  - Build different “experts”, let them vote
- Advantage
  - Often improves predictive performance
- Disadvantage
  - Usually produces output that is very hard to analyze

However, there are approaches that aim to produce a single comprehensible structure



- Suppose there are 25 base classifiers
- Each classifier has error rate,  $\epsilon = 0.35$
- Assume classifiers are independent
- The probability that the ensemble makes a wrong prediction is

$$\sum_{i=13}^{25} \binom{25}{i} \epsilon^i (1 - \epsilon)^{25-i} = 0.06$$

how can we generate several models using  
the same data and the same approach (e.g., Trees)?

we might ...

randomize the selection of training data?

randomize the approach?

...

# Bagging

# What is Bagging? (Bootstrap Aggregation)

- Analogy
  - Diagnosis based on multiple doctors' majority vote
- Training
  - Given a set  $D$  of  $d$  tuples, at each iteration  $i$ , a training set  $D_i$  of  $d$  tuples is sampled with replacement from  $D$  (i.e., bootstrap)
  - A classifier model  $M_i$  is learned for each training set  $D_i$
- Classification (classify an unknown sample  $X$ )
  - Each classifier  $M_i$  returns its class prediction
  - The bagged classifier  $M^*$  counts the votes and assigns the class with the most votes to  $X$
- Combining predictions by voting/averaging
  - The simplest way is to give equal weight to each model
- Predicting continuous values
  - It can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple

## Model generation

Let  $n$  be the number of instances in the training data

For each of  $t$  iterations:

    Sample  $n$  instances from training set  
    (with replacement)

    Apply the learning algorithm to the sample

    Store the resulting model

## Classification

For each of the  $t$  models:

    Predict class of instance using model

    Return class that is predicted most often

Bagging works because it reduces variance by voting/averaging

usually, the more classifiers the better

it can help a lot if data are noisy

however, in some pathological hypothetical situations the overall error might increase

# When Does Bagging Work?

## Stable vs Unstable Classifiers

14

- A learning algorithm is unstable, if small changes to the training set cause large changes in the learned classifier
- If the learning algorithm is unstable, then bagging almost always improves performance
- Bagging stable classifiers is not a good idea
- Decision trees, regression trees, linear regression, neural networks are examples of unstable classifiers
- K-nearest neighbors is a stable classifier



- We could randomize the learning algorithm instead of the data
- Some algorithms already have a random component,  
e.g. initial weights in neural net
- Most algorithms can be randomized (e.g., attribute selection in decision trees)
- More generally applicable than bagging, e.g. random subsets in nearest-neighbor scheme
- It can be combined with bagging

the more uncorrelated the trees,  
the better the variance reduction

# Random Forests

learning ensemble consisting of a bagging  
of unpruned decision tree learners with  
randomized selection of features at each split

- Random forests (RF) are a combination of tree predictors 
- Each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest
- The generalization error of a forest of tree classifiers depends on the strength of the individual trees in the forest and the correlation between them
- Using a random selection of features to split each node yields error rates that compare favorably to Adaboost, and are more robust with respect to noise

- $D$  = training set  $F$  = set of variables to test for split
- $k$  = number of trees in forest  $n$  = number of variables
- for  $i = 1$  to  $k$  do
  - build data set  $D_i$  by sampling with replacement from  $D$
  - learn tree  $T_i$  from  $D_i$  using at each node only a subset  $F$  of the  $n$  variables
  - save tree  $T_i$  as it is, do not perform any pruning
- Output is computed as the majority voting (for classification) or average (for prediction) of the set of  $k$  generated trees

---

**Algorithm 15.1** Random Forest for Regression or Classification.

1. For  $b = 1$  to  $B$ :
  - (a) Draw a bootstrap sample  $\mathbf{Z}^*$  of size  $N$  from the training data.
  - (b) Grow a random-forest tree  $T_b$  to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size  $n_{min}$  is reached.
    - i. Select  $m$  variables at random from the  $p$  variables.
    - ii. Pick the best variable/split-point among the  $m$ .
    - iii. Split the node into two daughter nodes.
2. Output the ensemble of trees  $\{T_b\}_1^B$ .

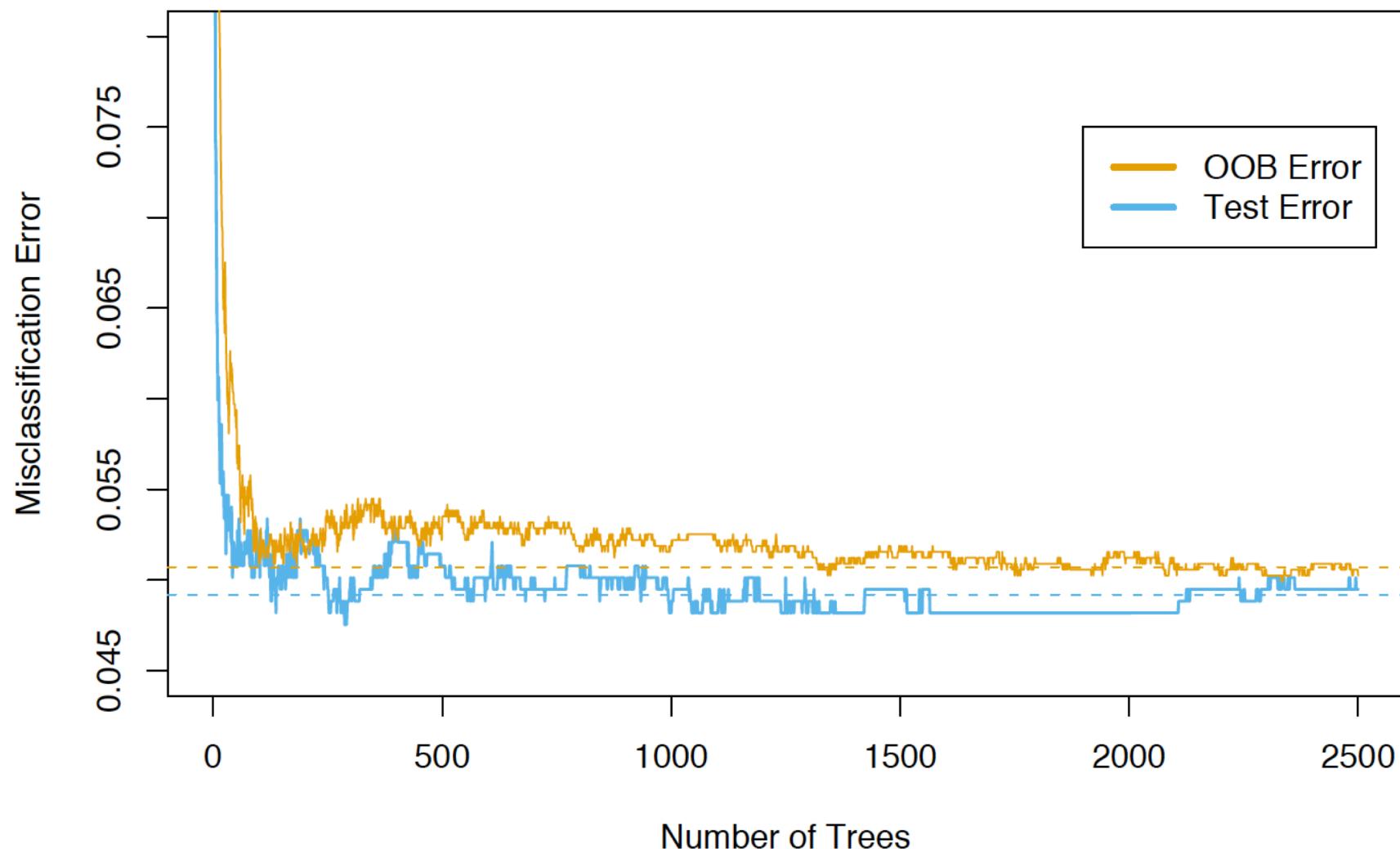
To make a prediction at a new point  $x$ :

$$\text{Regression: } \hat{f}_{\text{rf}}^B(x) = \frac{1}{B} \sum_{b=1}^B T_b(x).$$

*Classification:* Let  $\hat{C}_b(x)$  be the class prediction of the  $b$ th random-forest tree. Then  $\hat{C}_{\text{rf}}^B(x) = \text{majority vote } \{\hat{C}_b(x)\}_1^B$ .

---

- For each observation  $(x_i, y_i)$ , construct its random forest predictor by averaging only those trees corresponding to bootstrap samples in which the observation did not appear
- The OOB error estimate is almost identical to that obtained by n-fold crossvalidation and related to the leave-one-out evaluation
- Thus, random forests can be fit in one sequence, with cross-validation being performed along the way
- Once the OOB error stabilizes, the training can be terminated



- Easy to use ("off-the-shelf"), only 2 parameters (no. of trees, %variables for split)
- Very high accuracy
- No overfitting if selecting large number of trees (choose high)
- Insensitive to choice of split% ( $\sim 20\%$ )
- Returns an estimate of variable importance
- Random forests are an effective tool in prediction.
- Forests give results competitive with boosting and adaptive bagging, yet do not progressively change the training set.
- Random inputs and random features produce good results in classification - less so in regression.
- For larger data sets, we can gain accuracy by combining random features with boosting.

# Boosting

headache & stomach  
pain & leg hurts



headache is surely caused  
by X but cannot explain  
others



stomach pain & leg hurts  
(headache because of X)



agree for X, leg is because  
of Y but cannot explain  
stomach





headache is surely caused  
by X but cannot explain  
others



agree on X, leg is because  
of Y but cannot explain  
stomach



agree on X sort of agree on  
Y, sure stomach is Z



**Final Diagnosis**  
result of the weighted  
opinions that the doctors  
(sequentially) gave you

- AdaBoost computes a strong classifier as a combination of weak classifiers,

$$H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x) \right)$$

- Where
  - $h_t(x)$  is the output of the weak classifier  $t$
  - $\alpha_t$  is the weight assigned to the weak classifier  $t$  from its estimated error  $\epsilon_t$  as

$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$$

- Analogy
  - Consult several doctors, sequentially trying to find a doctor that can explain what other doctors cannot
  - Final diagnose is based on a weighted combination of all the diagnoses
  - Weights are assigned based on the accuracy of previous diagnoses
- How does Boosting work?
  - Weights are assigned to each training example
  - A series of  $k$  classifiers is iteratively learned
  - After a classifier  $M_i$  is learned, the weights are updated to allow the subsequent classifier  $M_{i+1}$  to pay more attention to the training tuples that were misclassified by  $M_i$
  - The final  $M^*$  combines the votes of each individual classifier, where the weight of each classifier's vote is a function of its accuracy

- Suppose our data consist of ten points with a +|-| label

x	0	1	2	3	4	5	6	7	8	9
y	+	+	+	-	-	-	-	-	+	+

- At the beginning, all the data points have the same weight and thus the same probability of being selected for the training set. With 10 data points the weight is 1/10 or 0.1

w	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100
x	0	1	2	3	4	5	6	7	8	9
y	+	+	+	-	-	-	-	-	+	+

- Suppose that in the first round, we generated the training data using the weights and generated a weak classifier  $h_1$  that returns  $+|$  if  $x < 3$

	$h_1$									
w	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100	0.100
x	0	1	2	3	4	5	6	7	8	9
y	+	+	+	-	-	-	-	-	+	+
	+	+	+	-	-	-	-	-	-	-

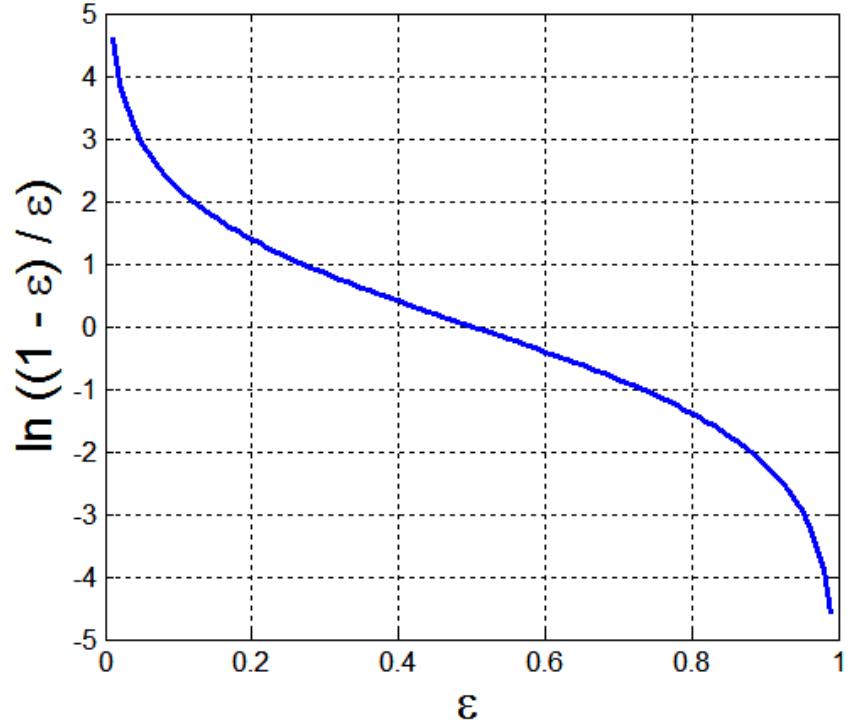
- The error  $\epsilon_1$  on the weighted dataset is 2/10 and  $\alpha_1$  is 0.69
- The weights are updated by multiplying them with
  - $\exp(-\alpha_1)$  if the example was correctly classified
  - $\exp(\alpha_1)$  if the example was incorrectly classified
- Finally, the weights are normalized since they have to sum up to 1

- The importance of a classifier is computed as

$$\alpha_i = \frac{1}{2} \ln \left( \frac{1 - \epsilon_i}{\epsilon_i} \right)$$

- Where the error rate is computed as

$$\epsilon_i = \frac{1}{N} \sum_{j=1}^N w_j \delta(h_i(x_j) \neq y_j)$$



- The new weights are thus,

w	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.25	0.25	
x	0	1	2	3	4	5	6	7	8	9
y	+	+	+	-	-	-	-	-	+	+

- Suppose now that we computed the second weak classifier  $h_2$  as

w	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.0625	0.25	0.25	
x	0	1	2	3	4	5	6	7	8	9
y	+	+	+	-	-	-	-	-	+	+
	-	-	-	-	-	-	-	+	+	

- That has an error  $\epsilon_2$  of 0.1875 and  $\alpha_2$  is 0.7332

- The new weights are thus,

w	0.167	0.167	0.167	0.038	0.038	0.038	0.038	0.154	0.154
x	0	1	2	3	4	5	6	7	8
y	+	+	+	-	-	-	-	+	+

- Finally, we sample the dataset using the weights and compute the third weak classifier  $h_3$  as

 **$h_3$** 

w	0.167	0.167	0.167	0.038	0.038	0.038	0.038	0.154	0.154
x	0	1	2	3	4	5	6	7	8
y	+	+	+	-	-	-	-	+	+
	+	+	+	+	+	+	+	+	+

- That has an error  $\epsilon_3$  of 0.19 and an  $\alpha_3$  of 0.72

- The final model is thus computed as,

$$H(x) = \text{sign}(0.69h_1(x) + 0.73h_2(x) + 0.72h_3(x))$$

## Model generation

```
Assign equal weight to each training instance
For t iterations:
    Apply learning algorithm to weighted dataset,
        store resulting model
    Compute model's error e on weighted dataset
    If e = 0 or e ≥ 0.5:
        Terminate model generation
    For each instance in dataset:
        If classified correctly by model:
            Multiply instance's weight by e / (1-e)
    Normalize weight of all instances
```

## Classification

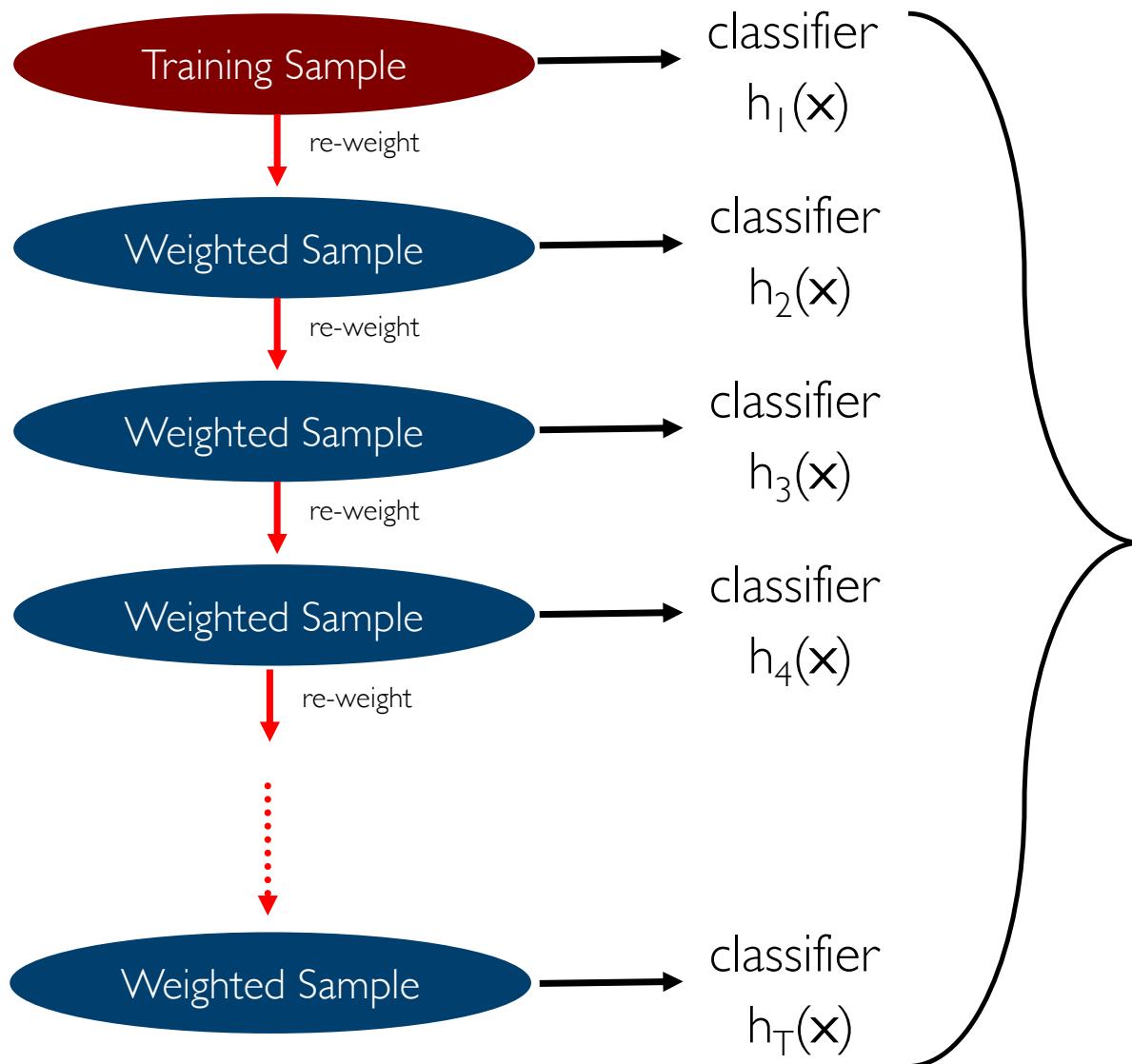
```
Assign weight = 0 to all classes
For each of the t (or less) models:
    For the class this model predicts
        add -log e / (1-e) to this class's weight
Return class with highest weight
```

---

**Algorithm 10.1** AdaBoost.M1.

1. Initialize the observation weights  $w_i = 1/N$ ,  $i = 1, 2, \dots, N$ .
  2. For  $m = 1$  to  $M$ :
    - (a) Fit a classifier  $G_m(x)$  to the training data using weights  $w_i$ .
    - (b) Compute
$$\text{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}.$$
    - (c) Compute  $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$ .
    - (d) Set  $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$ ,  $i = 1, 2, \dots, N$ .
  3. Output  $G(x) = \text{sign} \left[ \sum_{m=1}^M \alpha_m G_m(x) \right]$ .
-

- Boosting needs weights ... but
- Can adapt learning algorithm ... or
- Can apply boosting **without** weights
  - Resample with probability determined by weights
  - Disadvantage: not all instances are used
  - Advantage: if error > 0.5, can resample again
- Stems from computational learning theory
- Theoretical result: training error decreases exponentially
- It works if base classifiers are not too complex, and their error doesn't become too large too quickly
- Boosting works with weak learners only condition: error doesn't exceed 0.5
- In practice, boosting sometimes overfits (in contrast to bagging)

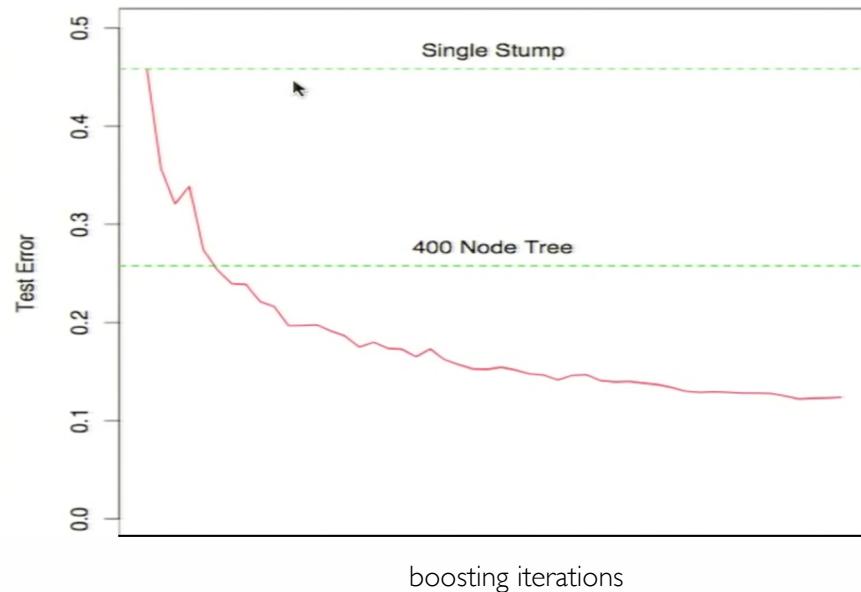


$$H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x) \right)$$

$$\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$$

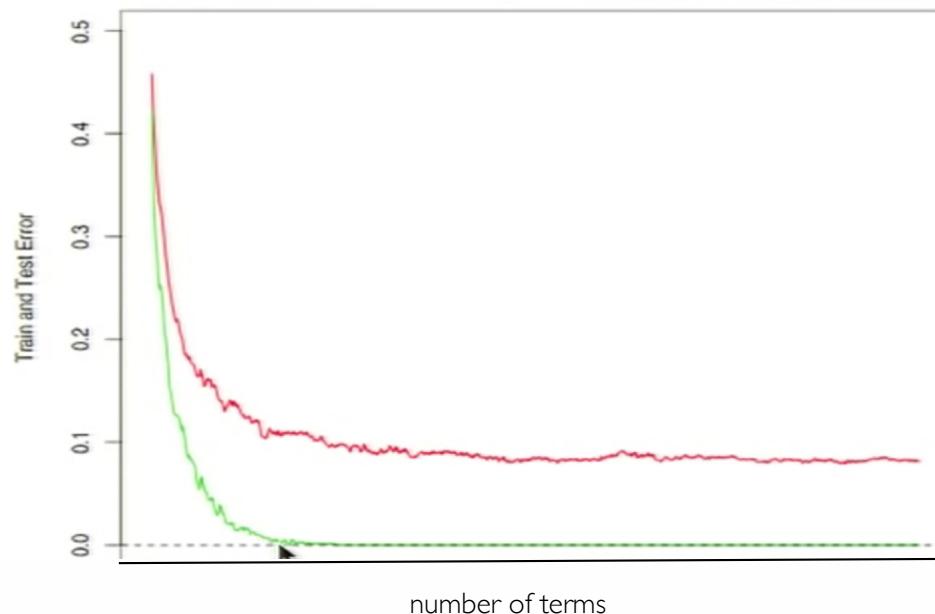
$$\epsilon_t = \frac{1}{N} \sum_{j=1}^N w_j \delta(h_t(x_j) \neq y_j)$$

- Random forests tend to grow rather bushy trees
- Boosting can work well even with shallow trees
- Even with stumps, simple two nodes trees involving just one split



## Boosting

- Boosting continues to reduce test error even if the training error reaches zero
- This suggests that boosting is not using training error as the basis for learning its trees



- Also uses voting/averaging
- Weights models according to performance
- Iterative: new models are influenced by the performance of previously built ones
  - Encourage new model to become an “expert” for instances misclassified by earlier models
  - Intuitive justification: models should be experts that complement each other
- Several variants
  - Boosting by sampling,  
the weights are used to sample the data for training
  - Boosting by weighting,  
the weights are used by the learning algorithm

# Gradient Boosting

- Build a sequence of predictors by repeating three simple steps
  1. Learn a regression predictor
  2. Compute the error residual
  3. Learn to predict the residual
- The sum of the predictions is increasingly accurate
- The predictive function is increasingly complex

# Why is it Called Gradient Boosting? (Intuition using MSE)

45

- To compute a model to predict a target value  $y_i$  that minimizes a loss function, for instance the mean square error

$$MSE(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

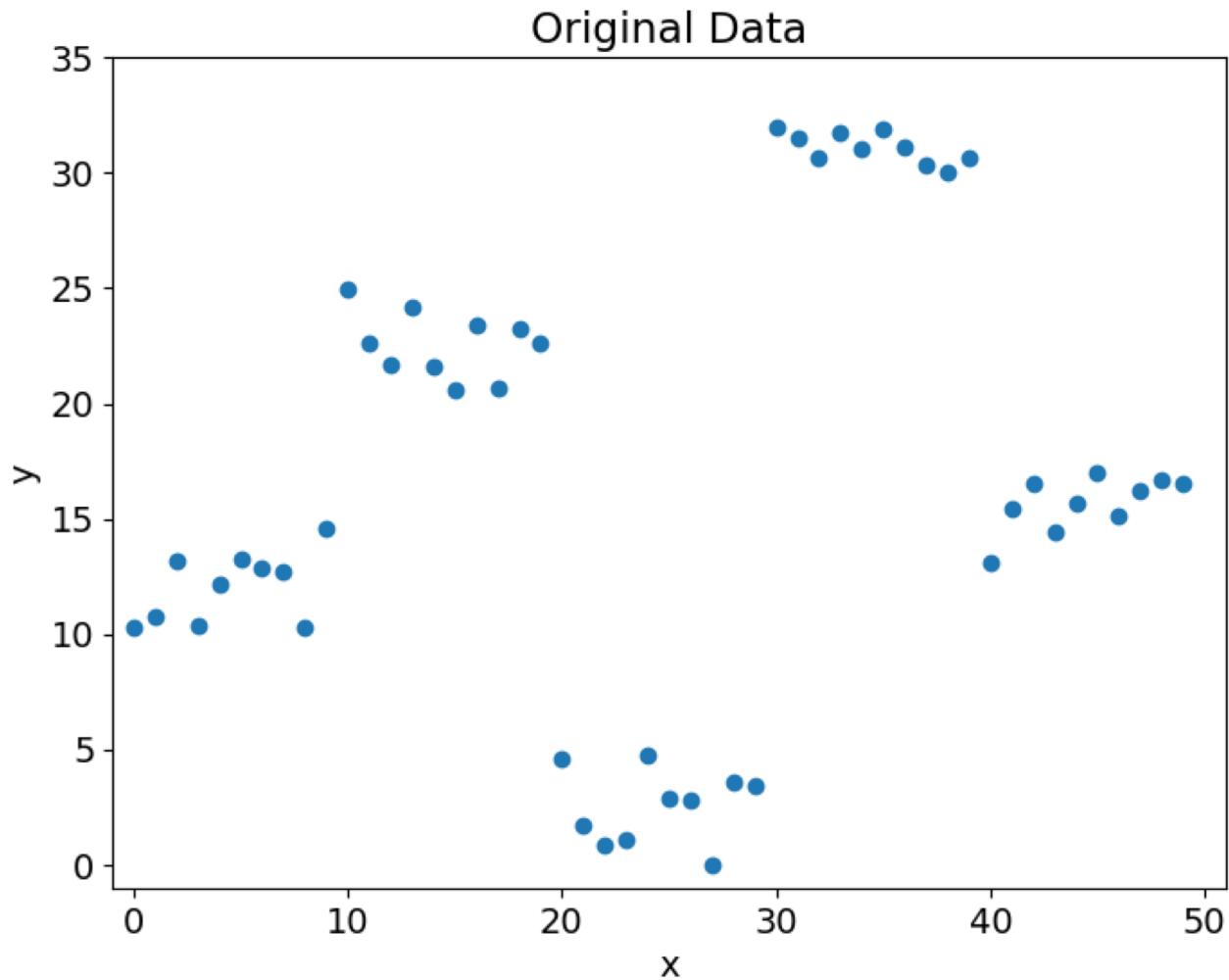
- We can adjust  $\hat{y}_i$  to try to reduce the error using the gradient

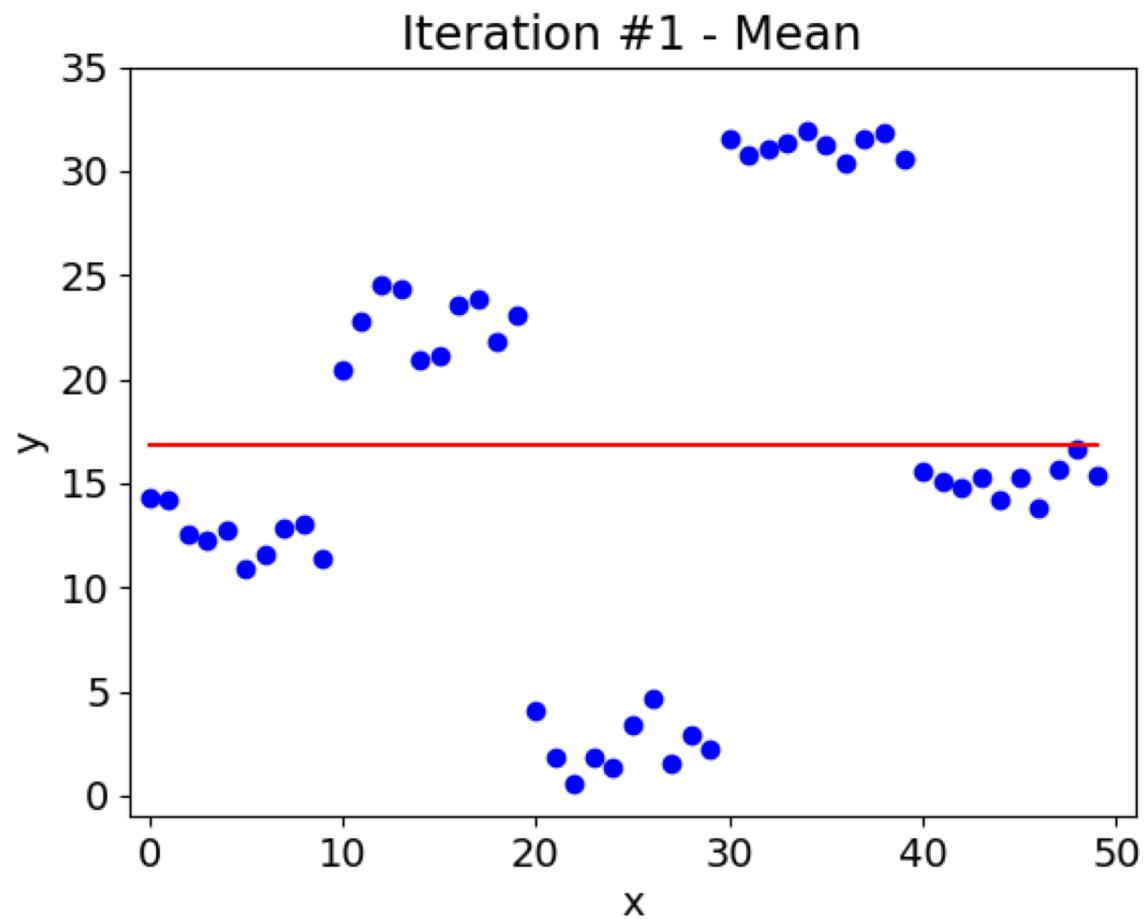
$$\hat{y}_i = \hat{y}_i + \alpha \nabla MSE(y, \hat{y})$$

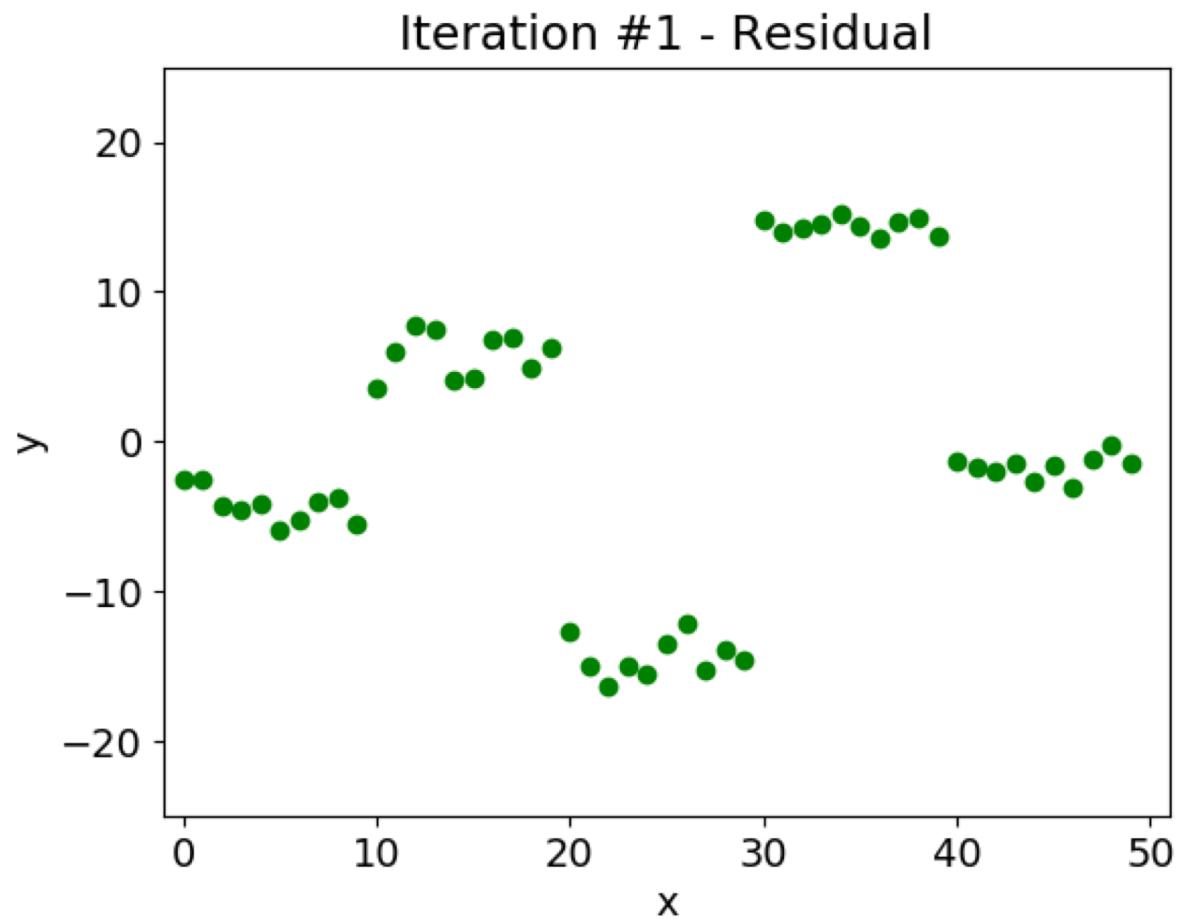
- The gradient for the MSE is a function of

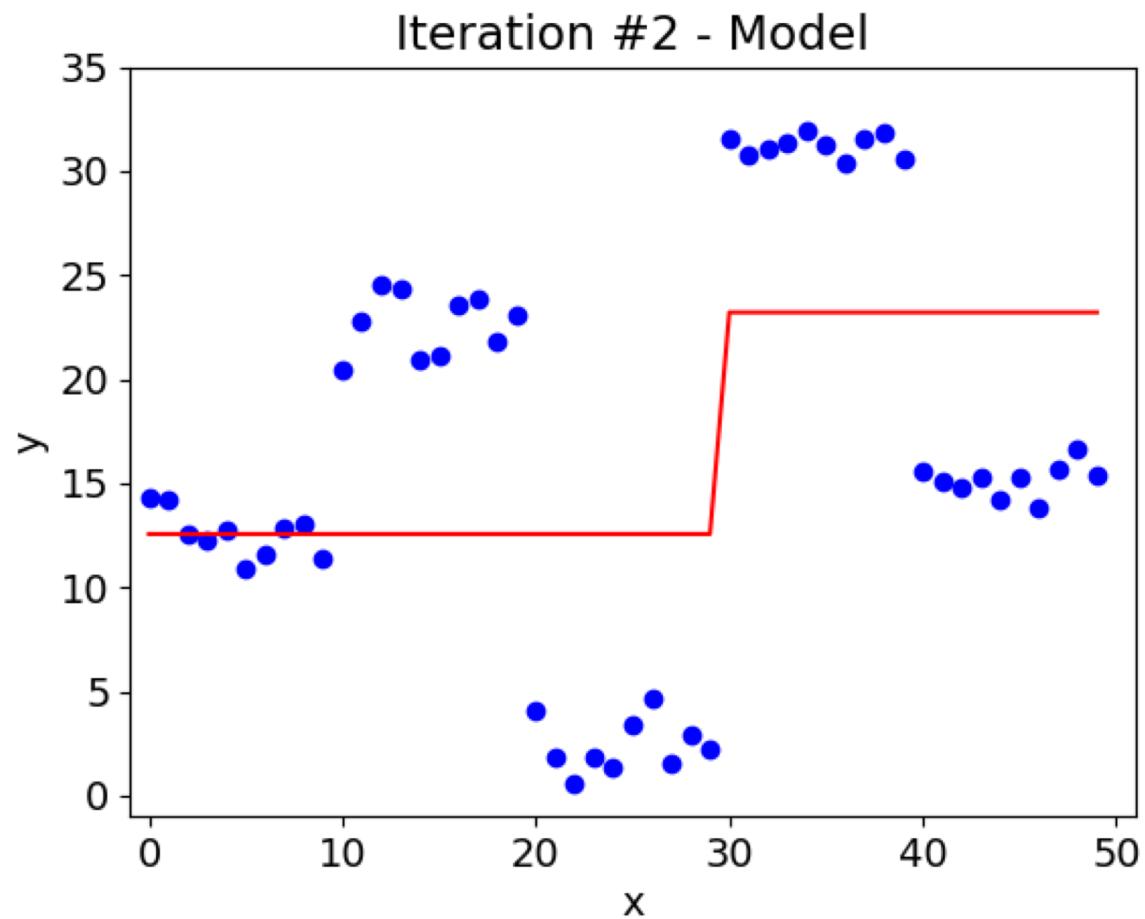
$$y_i - \hat{y}_i$$

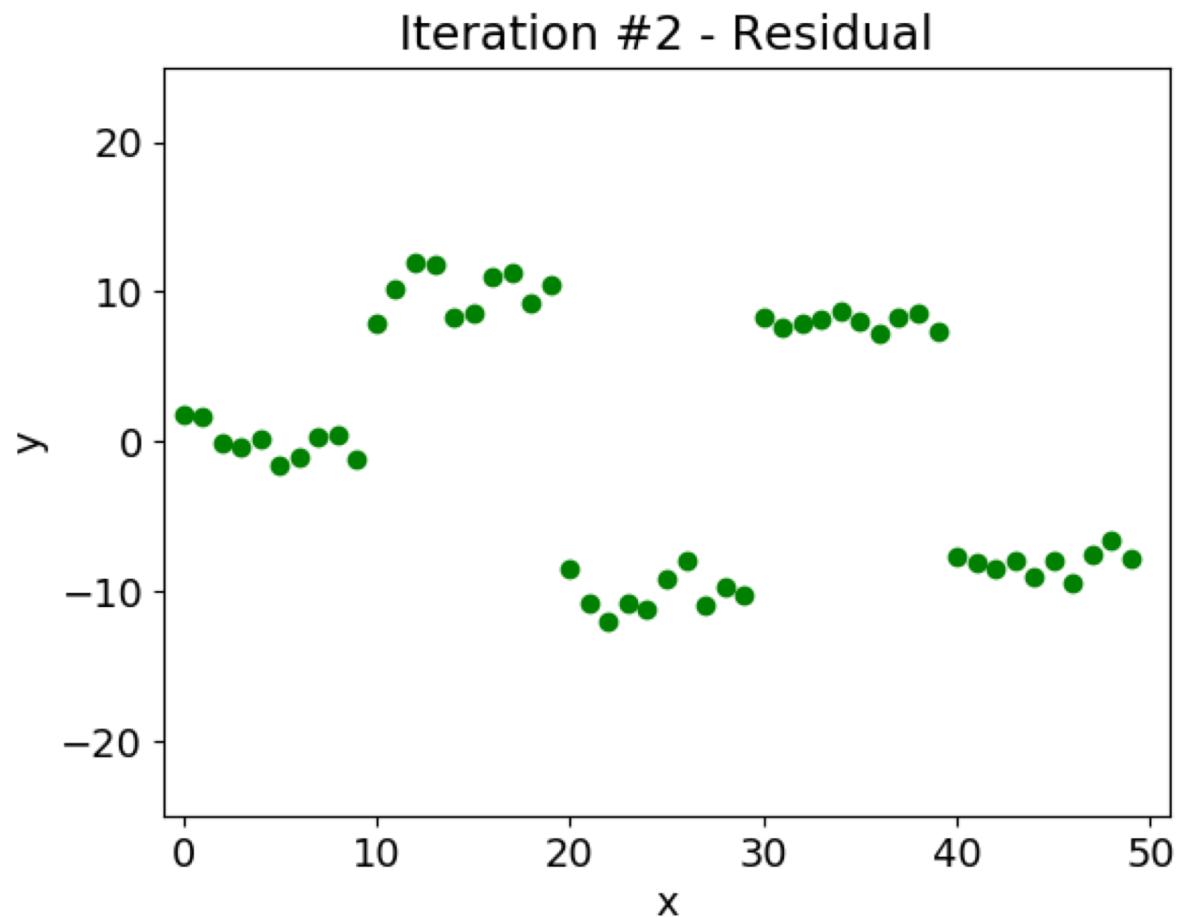
- Each learner is estimating the gradient of the loss. Larger  $\alpha$  means larger steps, smaller  $\alpha$  smaller steps and smoothing effect

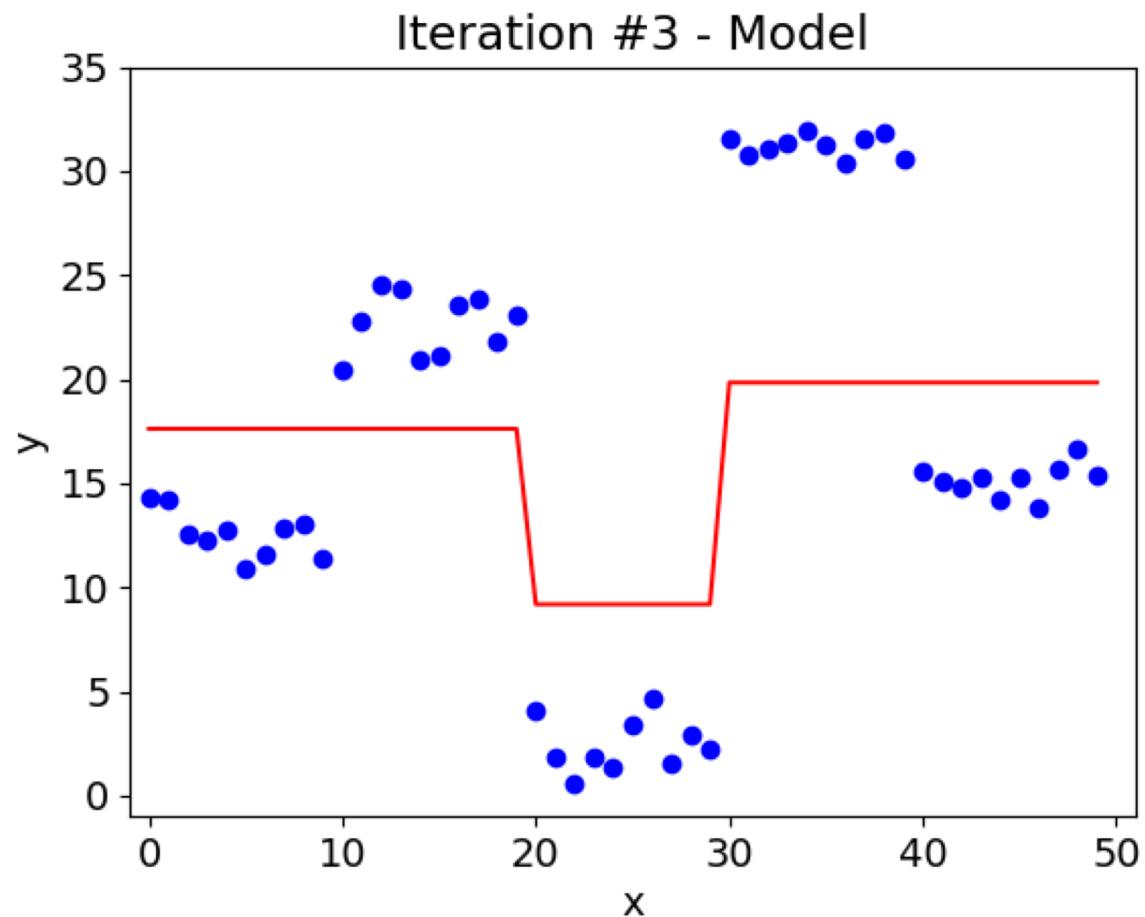


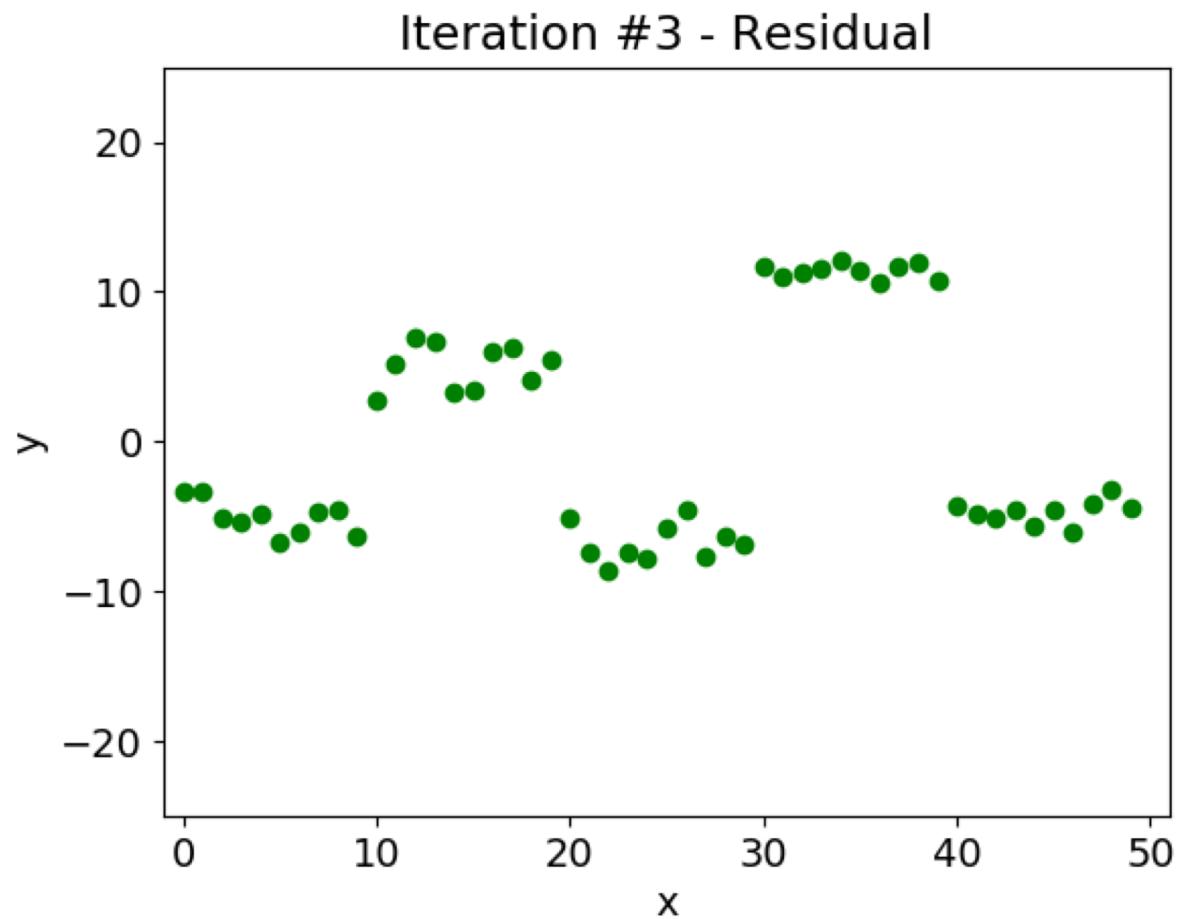


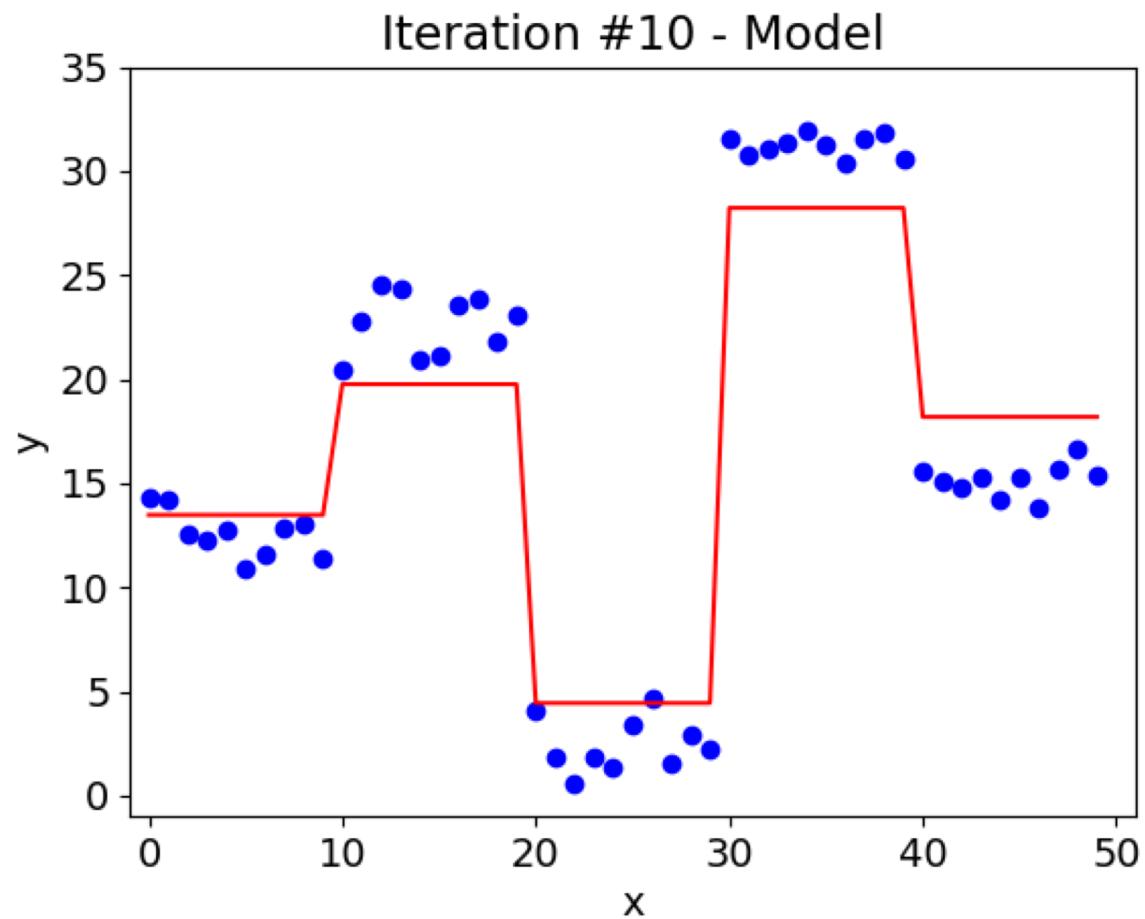


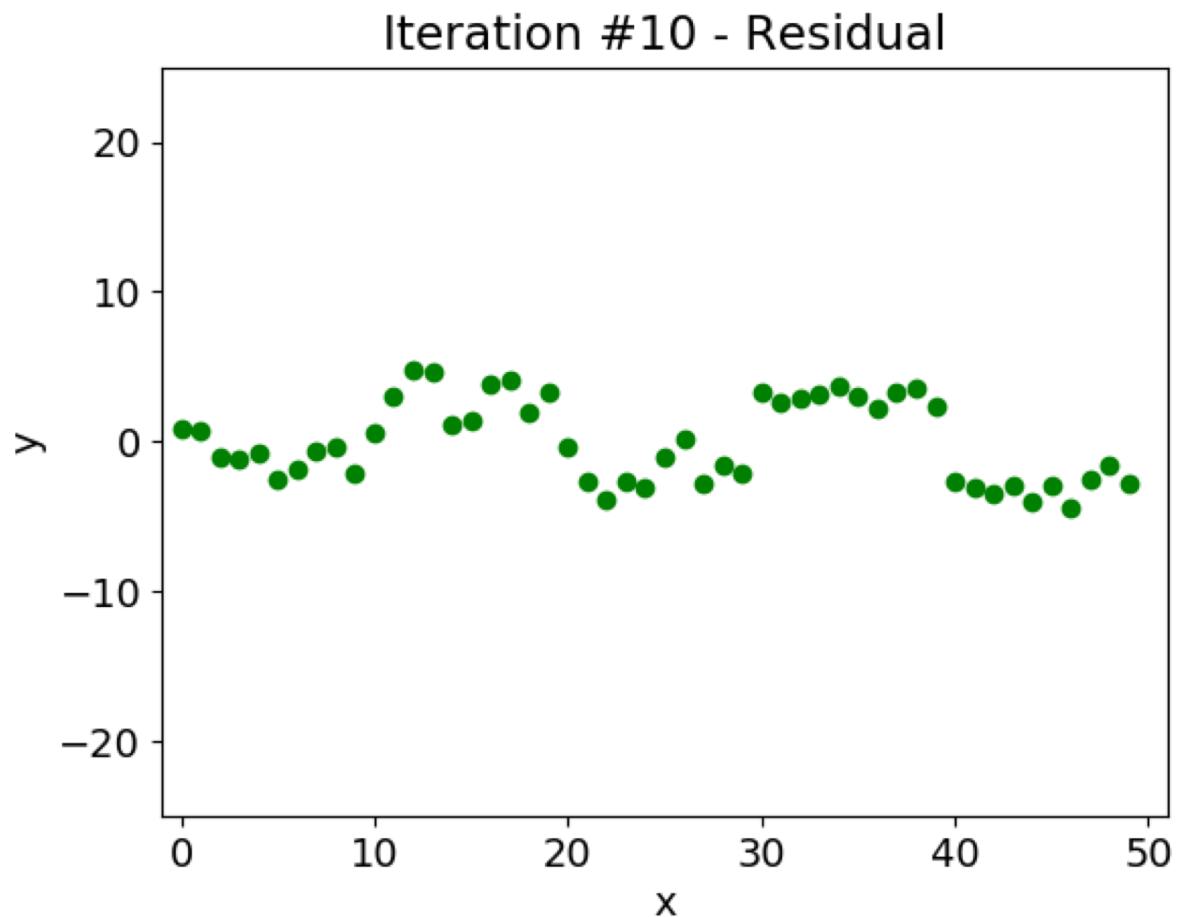


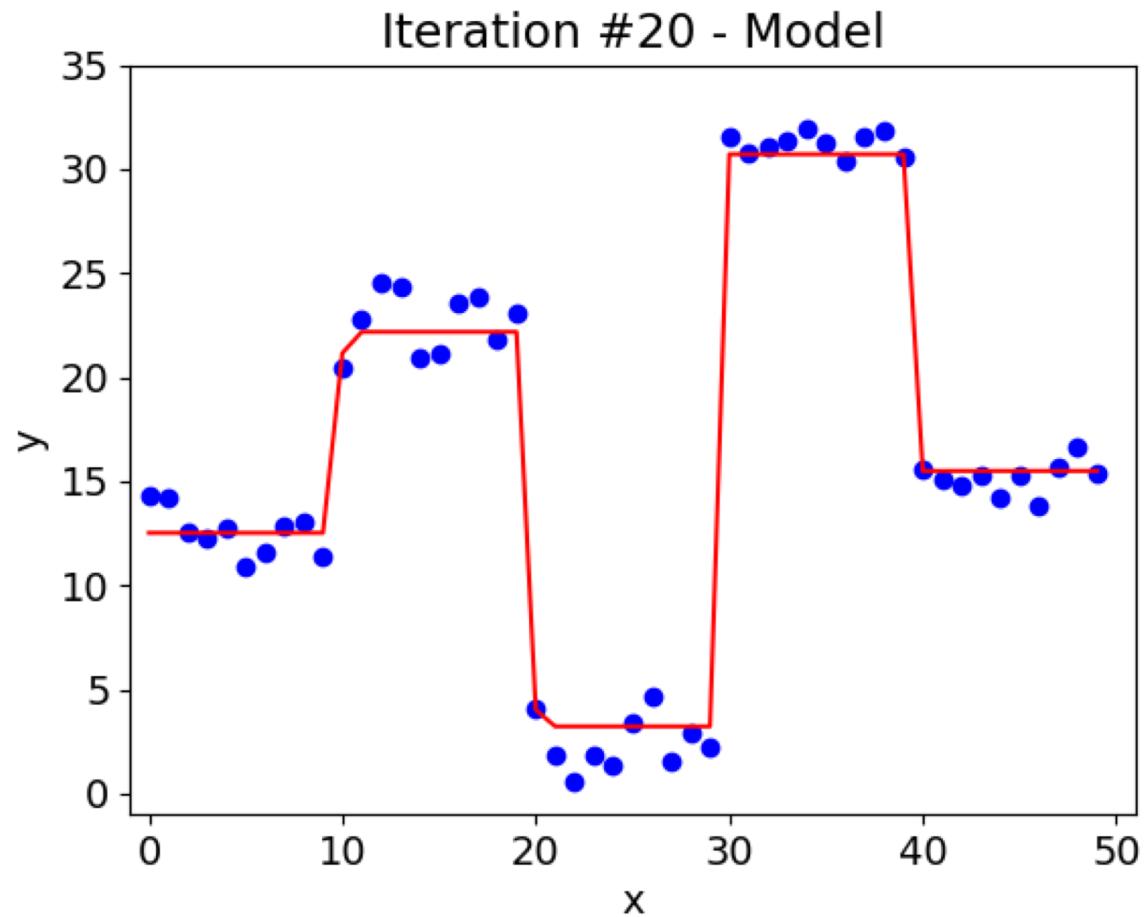


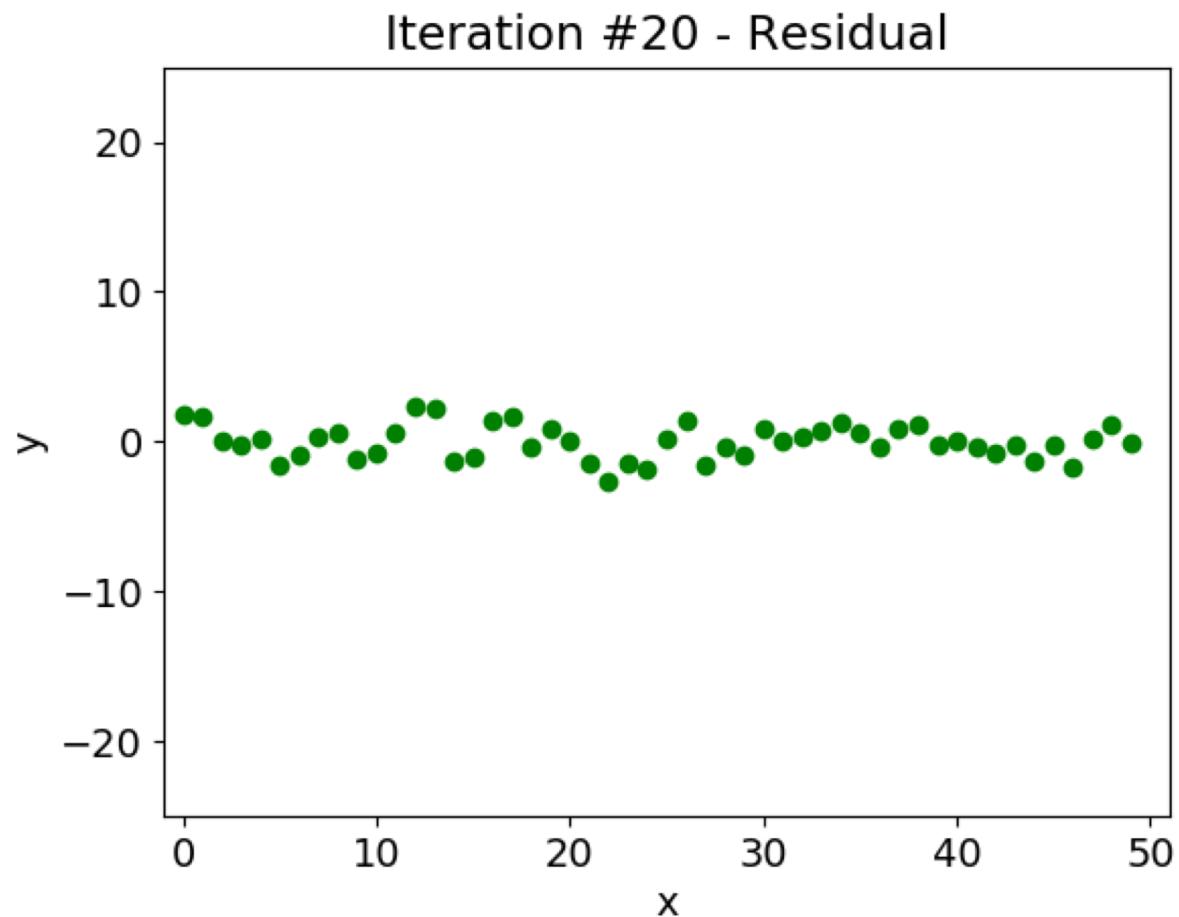






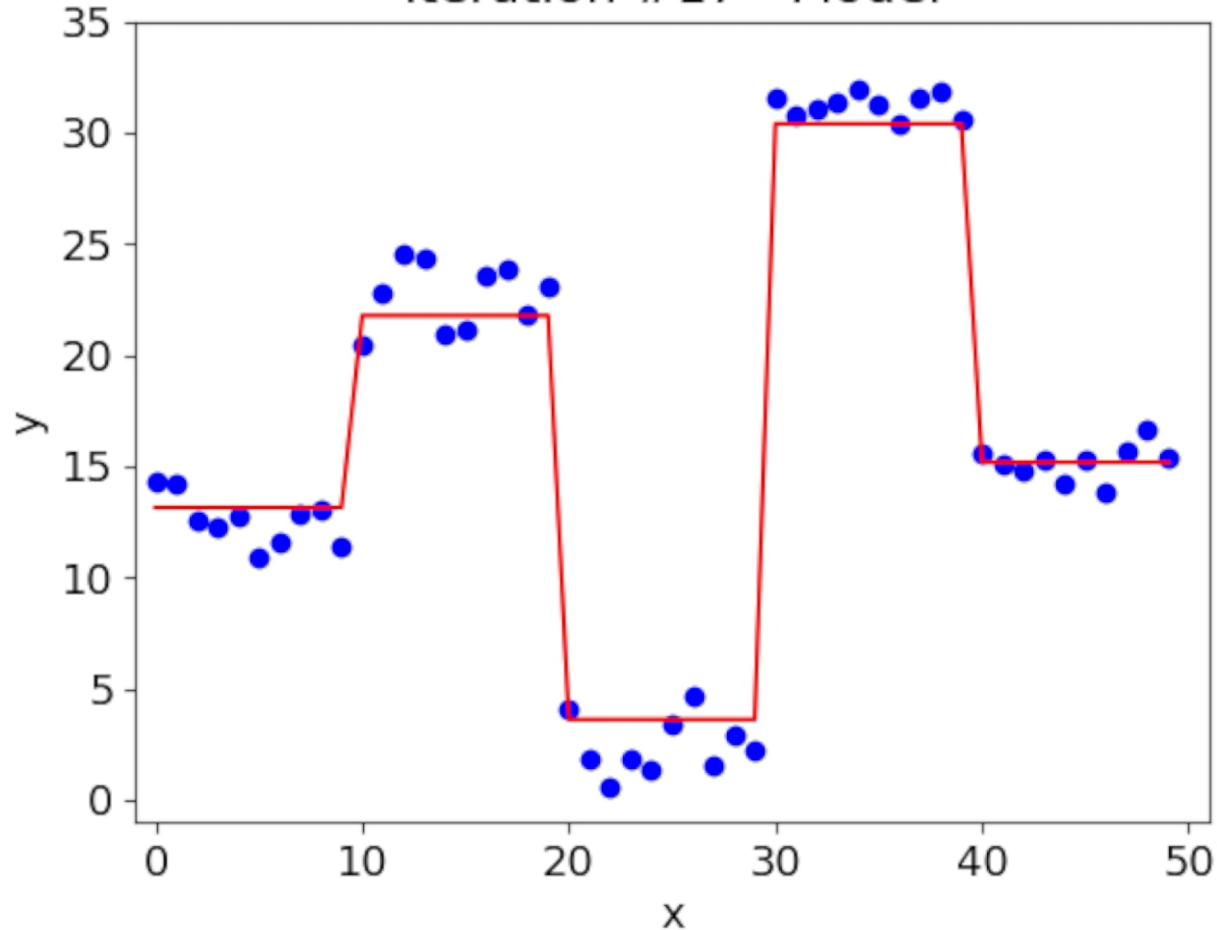








Iteration #17 - Model



# Gradient Boosting Example

<https://youtu.be/rvM648cedXU>

```
# start with the basic predictor (the mean)
mu = mean(y)
dy = y - mu

# number of models
n_boost = 50

for k in range(n_boost):

    # fit the residual
    learner[k] = model.fit(X,dy)

    # set the learning rate
    alpha[k] = 1

    # update the residual
    dy = dy - alpha[k] * learner[k].predict(X)
```

```
# x_test and y_test are the test data
predict = zeros(len(y_test))

for k in range(n_boost):
    predict = predict + alpha[k]*learner[k].predict(x_test)
```

- Efficient and scalable implementation of gradient boosting applied to classification and regression trees
- Used by most of the winning solutions and runner up of the recent data science competitions
- It only deals with numerical variables
- Features
  - Novel tree learning algorithm to handle sparse data
  - Approximate tree learning using quantile sketch
  - More than ten times faster on single machines
  - More regularized model formalization to control over-fitting, which gives it better performance.

# Learning Ensembles

- Random forests and boosting computes a set of weak models and combines them to build a stronger model through an incremental optimization process
- Random forests averages the models, boosting uses a weighted average
- We can use such weak models as a dictionary to build a stronger model by performing a global optimization over the ensemble
- First, we generate the dictionary of models (using random forests or boosting)

$$D = \{T_1(X), T_2(X), \dots, T_M(X)\}$$

- Next we fit a model to the data

$$F(X) = \sum_{m \in D} \alpha_m T_m(X)$$

# Stacked Generalization (Stacking)

- Generalization of previous ensemble methods and resemble many similarities to the concept of learning ensembles
- Trains a learning algorithm to combine the predictions of an **heterogeneous** set of learning algorithms
  - First, a set of base learners are trained over the data (to generate level-0 models)
  - Then, a meta learner is trained using the predictions of base classifiers as additional inputs (to generate a level-I model)
- Applies crossvalidation-like scheme
- Typically yields performance better than any single one of the trained models

Run the python notebooks and the  
KNIME workflows for this lecture