# Supervised Machine Learning - D

## Road Map

- Basic concepts
- Regression
- Naïve Bayesian classification
- K-nearest neighbor
- Support vector machines
- Decision tree induction
- **Ensemble methods: Bagging and Boosting**
- Summary

## Combining classifiers

- So far, we have only discussed individual classifiers, i.e., how to build them and use them.
- Can we combine multiple classifiers to produce a better classifier?
- Yes, sometimes
- We discuss two main methods:
    - Bagging
    - Boosting

## Bagging

- Breiman, 1996
- Bootstrap Aggregating = Bagging
  - Application of bootstrap sampling
    - Given: set *D* containing *m* training examples
    - Create a sample *S*[*i*] of *D* by drawing *m* examples at random *with replacement* from *D*

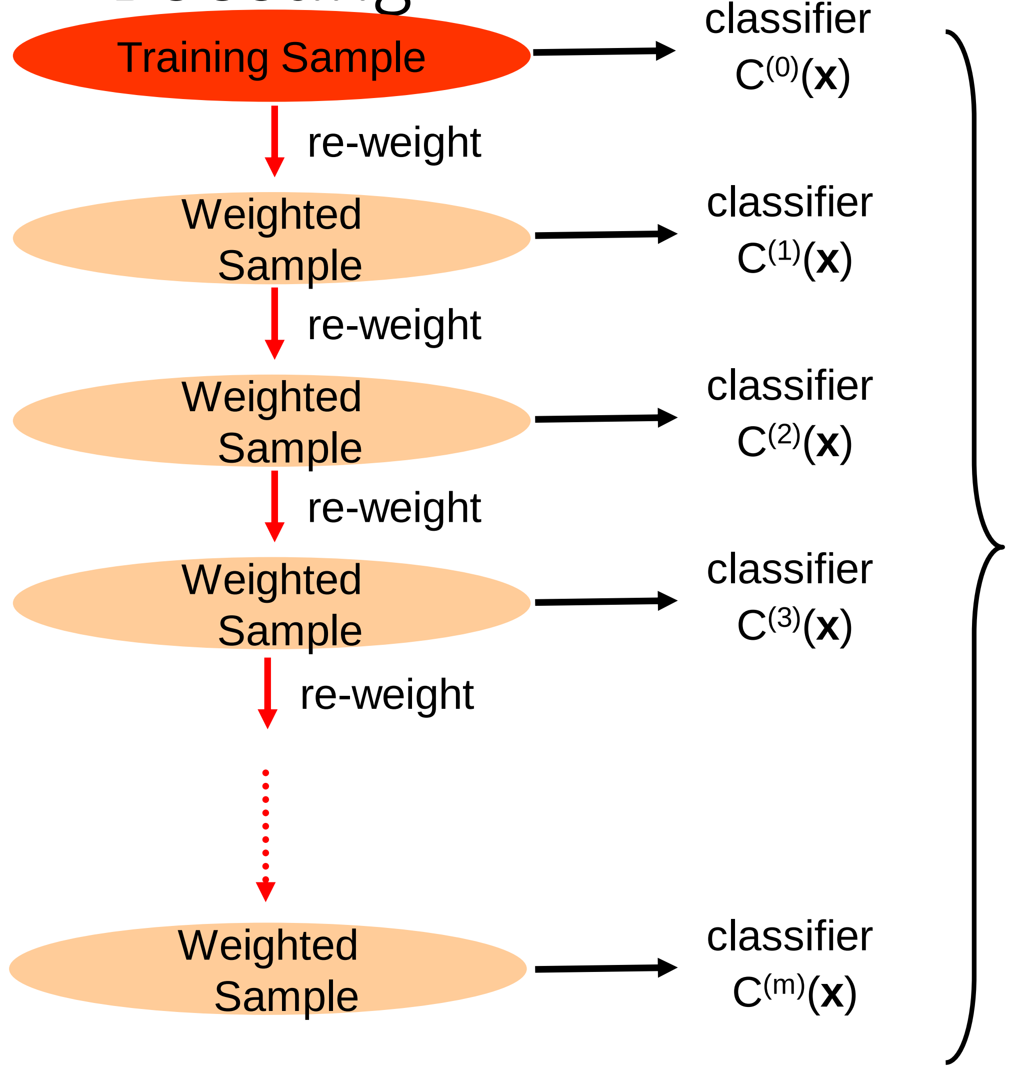
## Bagging (cont…)

- Training
  - Create *k* bootstrap samples *S*[1], *S*[2], …, *S*[*k*]
  - Build a distinct classifier on each *S*[*i*] to produce *k* classifiers, using the same learning algorithm.
- Testing
  - Classify each new instance by voting of the *k* classifiers (equal weights)

## Bagging (cont …)

- When does it help?
  - When learner is unstable
    - Small change to training set causes large change in the output classifier
    - True for decision trees; not true for *k*-nearest neighbor, naïve Bayesian
  - Experimentally, bagging can help substantially for unstable learners, may somewhat degrade results for stable learners

# Boosting


Training Sample
re-weight
Weighted Sample
re-weight
Weighted Sample
re-weight
Weighted Sample
re-weight
Weighted Sample
classifier $C^{(0)}(\mathbf{x})$
classifier $C^{(1)}(\mathbf{x})$
classifier $C^{(2)}(\mathbf{x})$
classifier $C^{(3)}(\mathbf{x})$
classifier $C^{(m)}(\mathbf{x})$

$$y(x) = \sum_{i}^{N_{\text{Classifier}}} w_i C^{(i)}(x)$$

## Boosting

**AdaBoost** (Freund & Schapire, 1996)

- Training
    - Produce a sequence of classifiers (the same base learner)
    - Each classifier is dependent on the previous one, and focuses on the previous one's errors
    - Examples that are incorrectly predicted in previous classifiers are given higher weights
- Testing
    - For a test case, the results of the series of classifiers are combined to determine the final class of the test case.

## AdaBoost

**Weighted**

**<u>training set</u>**

$(x_1, y_1, w_1)$
$(x_2, y_2, w_2)$
…
$(x_n, y_n, w_n)$

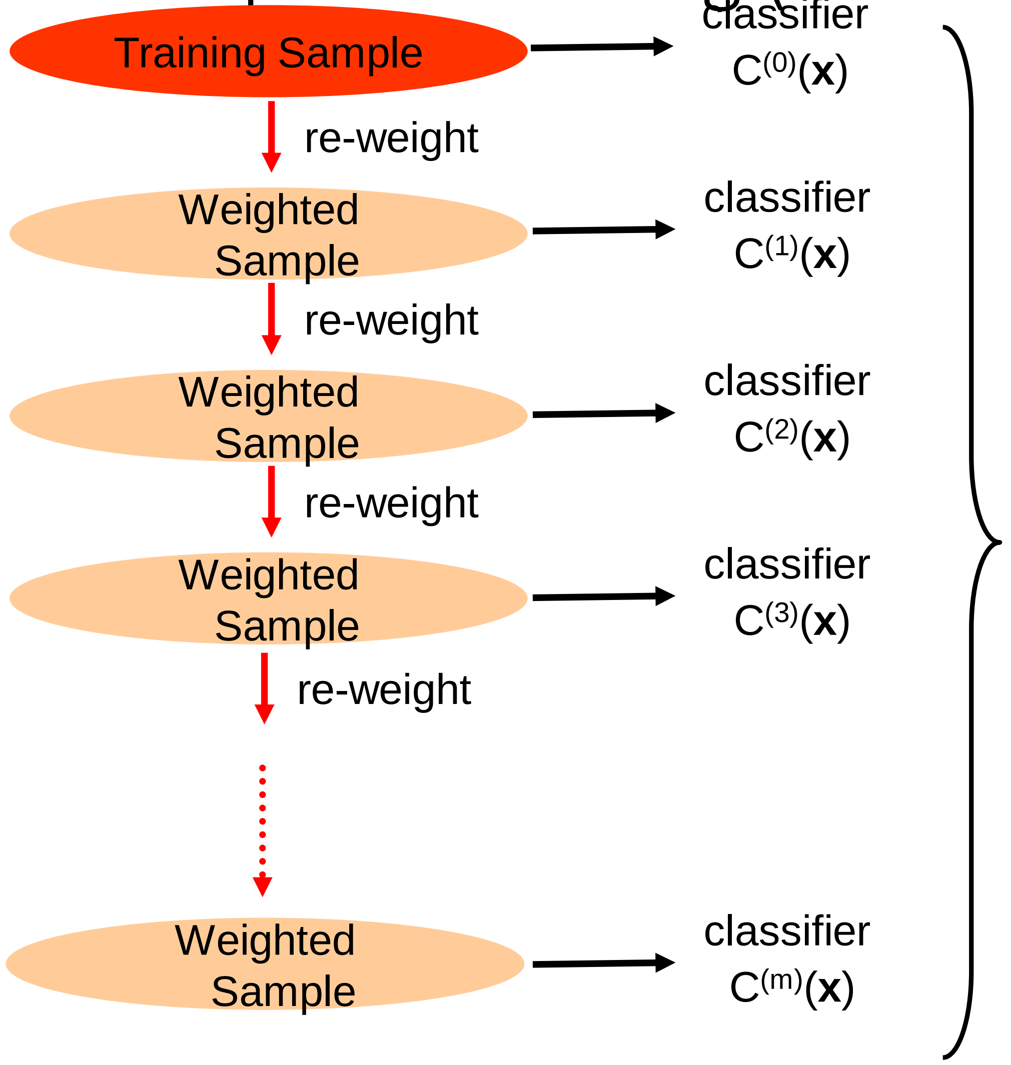
Non-negative weights
sum to 1

called a weaker classifier

- Build a classifier $\boldsymbol{h_t}$ whose accuracy on training set > ½ (better than random)

Change weights

# Adaptive Boosting (AdaBoost)


Training Sample
re-weight
Weighted Sample
re-weight
Weighted Sample
re-weight
Weighted Sample
re-weight
Weighted Sample
classifier $C^{(0)}(\mathbf{x})$
classifier $C^{(1)}(\mathbf{x})$
classifier $C^{(2)}(\mathbf{x})$
classifier $C^{(3)}(\mathbf{x})$
classifier $C^{(m)}(\mathbf{x})$

AdaBoost re-weights events misclassified by previous classifier by:

$$\frac{1-f_{err}}{f_{err}} \quad \text{with:}$$

$$f_{err} = \frac{\text{misclassified events}}{\text{all events}}$$

## Does AdaBoost always work?

- The actual performance of boosting depends on the data and the base learner.
  - It requires the base learner to be unstable as bagging.
- Boosting seems to be susceptible to noise.
  - When the number of outliners is very large, the emphasis placed on the hard examples can hurt the performance.

## Gradient tree boosting

- The key intuition behind boosting is that one can take an ensemble of simple models and additively combine them into a single, more complex model.
- Each model might be a poor fit for the data, but a linear combination of the ensemble can be expressive/flexible.
- Question: But which models should we include in our ensemble? What should the coefficients or weights in the linear combination be?

## Gradient tree boosting

Gradient boosting is a method for iteratively building a complex regression model by adding simple models. Each new simple model added to the ensemble compensates for the weaknesses of the current ensemble

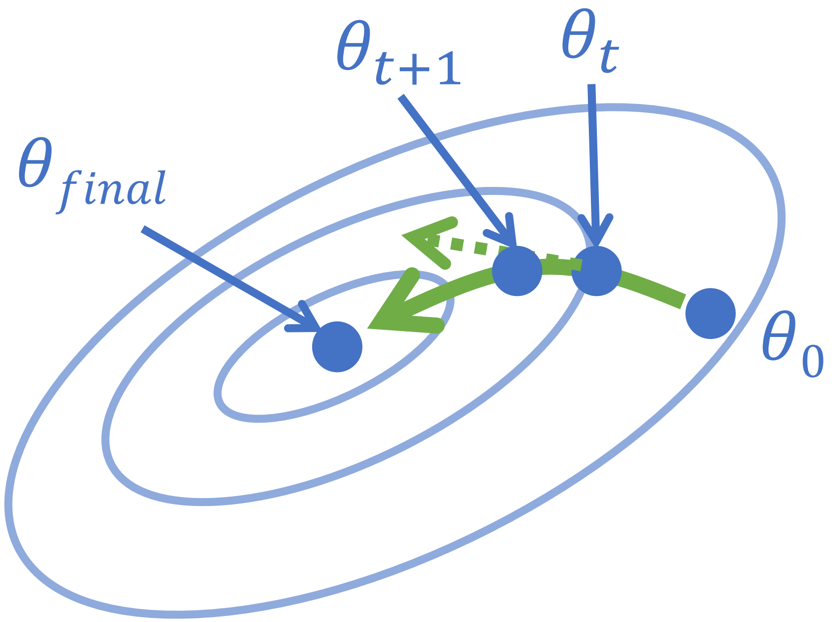
## Why Does Gradient Boosting Work?

- Intuitively, each simple model that we add to our ensemble model, models the errors.

- Thus, with each addition, the residual is reduced

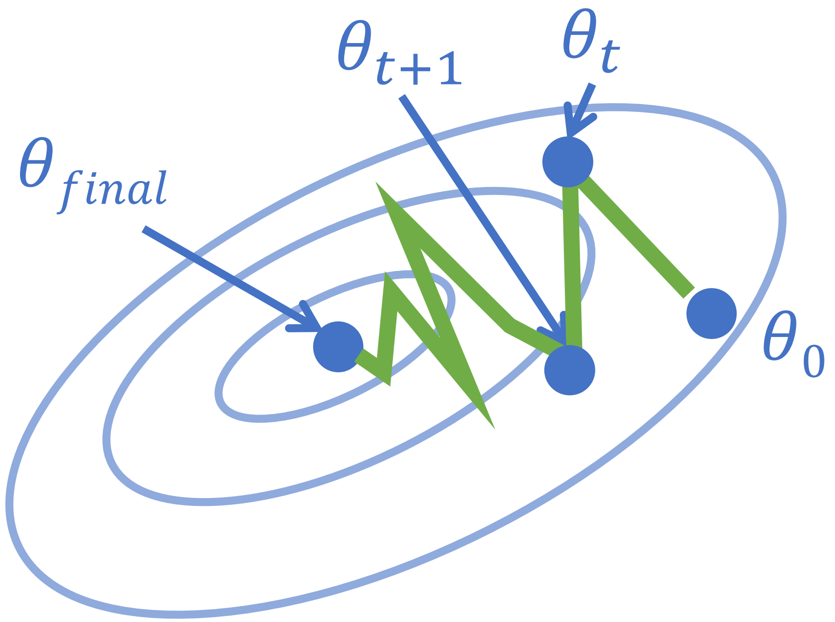
- How can we effectively descend the residual?
- We need to formulate gradient boosting as a type of gradient descent.

# Gradient descent



- Initialize $\theta_0$ randomly
- For *t* in 0,…, $T_{\text{maxiter}}$

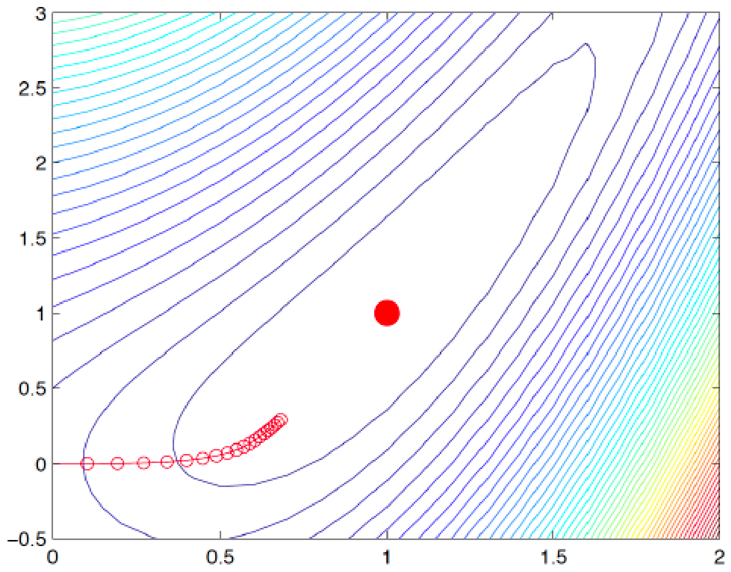
# Stochastic gradient descent (SGD)

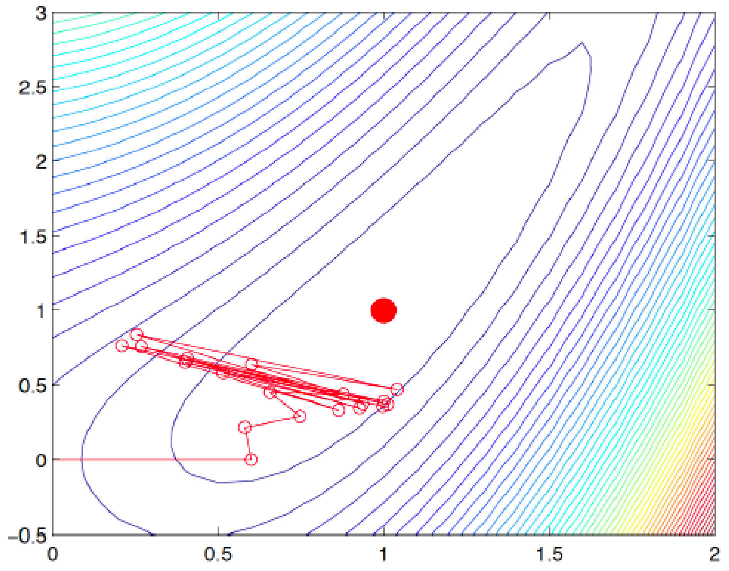


- Initialize $\theta_0$ randomly
- For $t$ in 0,…, $T_{\text{maxiter}}$

## Learning Rate

- ***Learning rate***
  - The gradient tells us the direction, but it does not tell us how far we should step
  - Choosing the learning rate (also called the step size) is one of the most important hyper-parameter settings



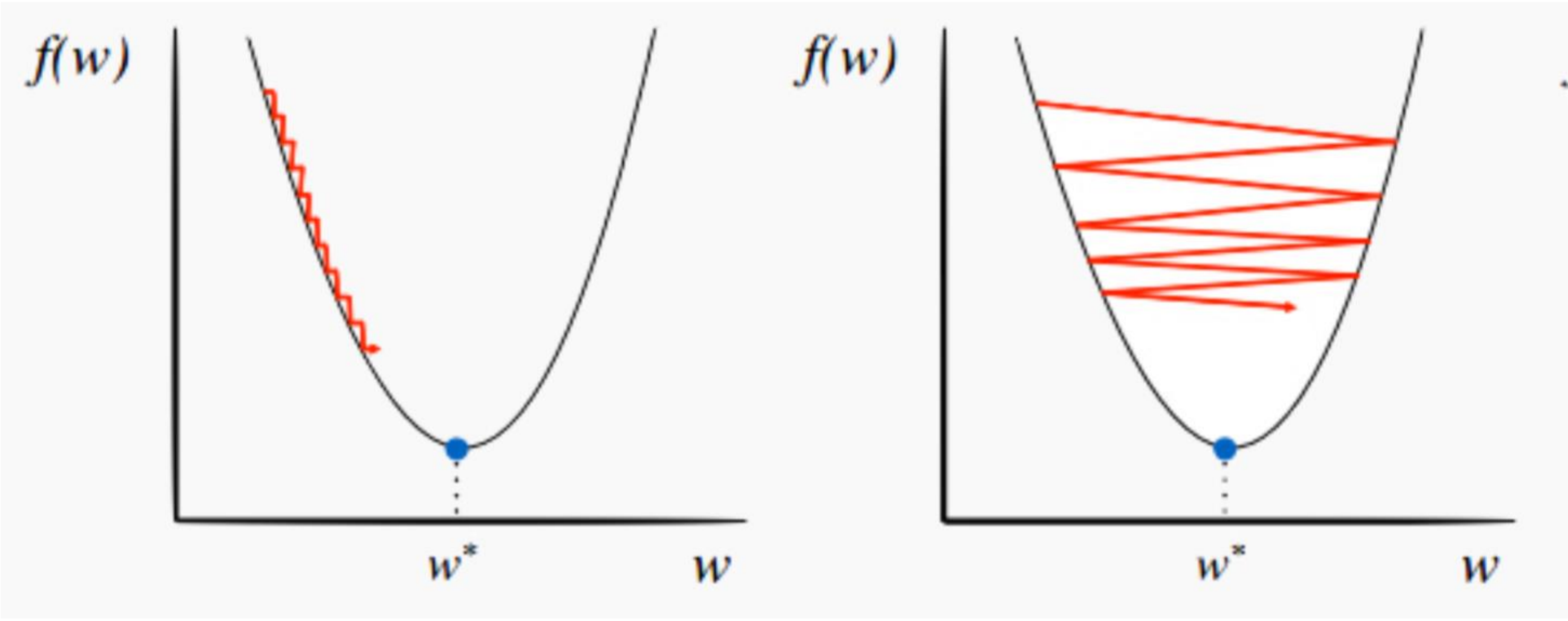


## Choosing a Learning Rate

- Under ideal conditions, gradient descent iteratively approximates and converges to the optimum.
- When do we terminate gradient descent?
  - We can limit the number of iterations in the descent. But for an arbitrary choice of maximum iterations, we cannot guarantee that we are sufficiently close to the optimum in the end.
  - If the descent is stopped when the updates are sufficiently small (e.g. the residuals are small), we encounter a new problem: the algorithm may never terminate!
- Both problems have to do with the magnitude of the learning rate.

## Choosing a Learning Rate

- For a constant learning rate, $\lambda$, if $\lambda$ is too small, it takes too many iterations to reach the optimum.
- If $\lambda$ is too large, the algorithm may 'bounce' around the optimum and never get sufficiently close.



## Choosing a Learning Rate

- If $\lambda$ is a constant, then it should be tuned through cross validation.
- For better results, use a variable $\lambda$. That is, let the value of $\lambda$ depend on the gradient
    - around the optimum, when the gradient is small, $\lambda$ should be small
    - far from the optimum, when the gradient is large, $\lambda$ should be larger

## Summary

- Applications of supervised learning are in almost any field or domain.
- We studied classic machine learning techniques.
- There are still many other methods
- It remains to be an active research area.