

Lecture 13

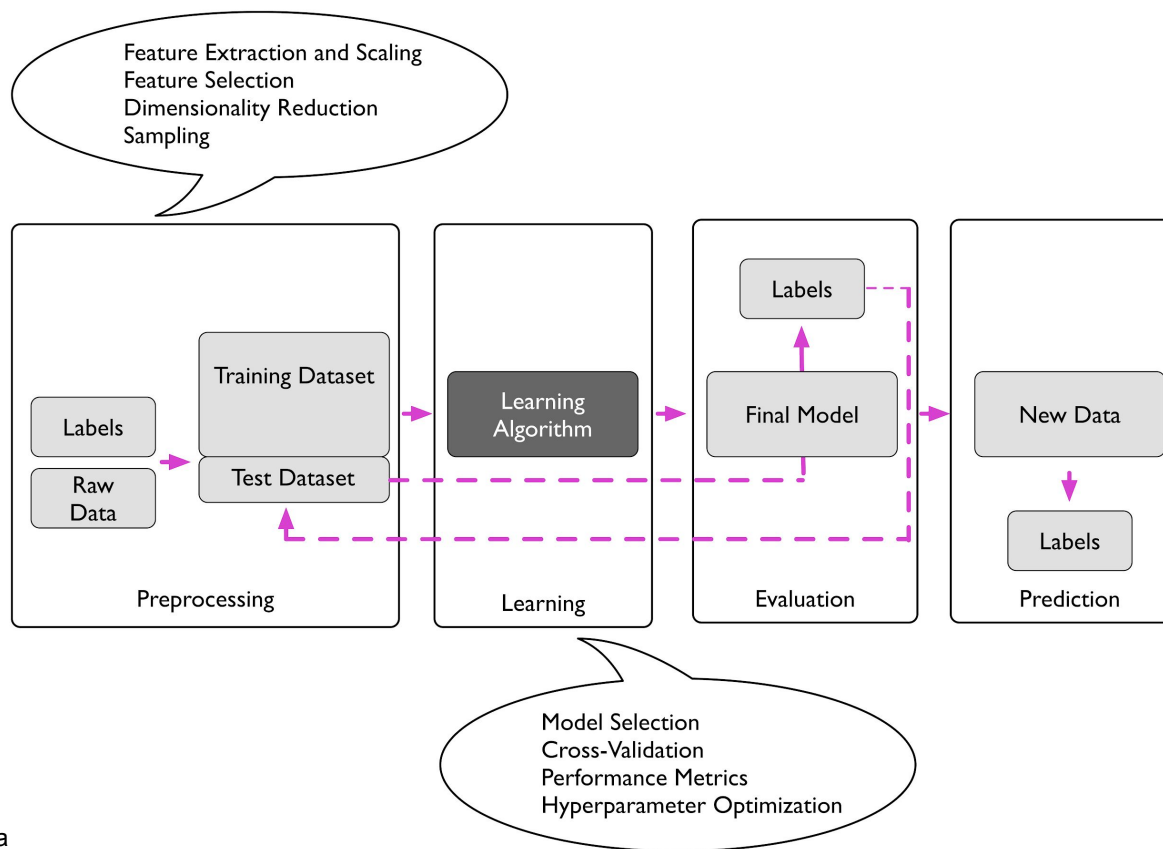
Ensemble methods

<https://github.com/dalcimar/MA28CP-Intro-to-Machine-Learning>

UTFPR - Federal University of Technology - Paraná

<https://www.dalcimar.com/>

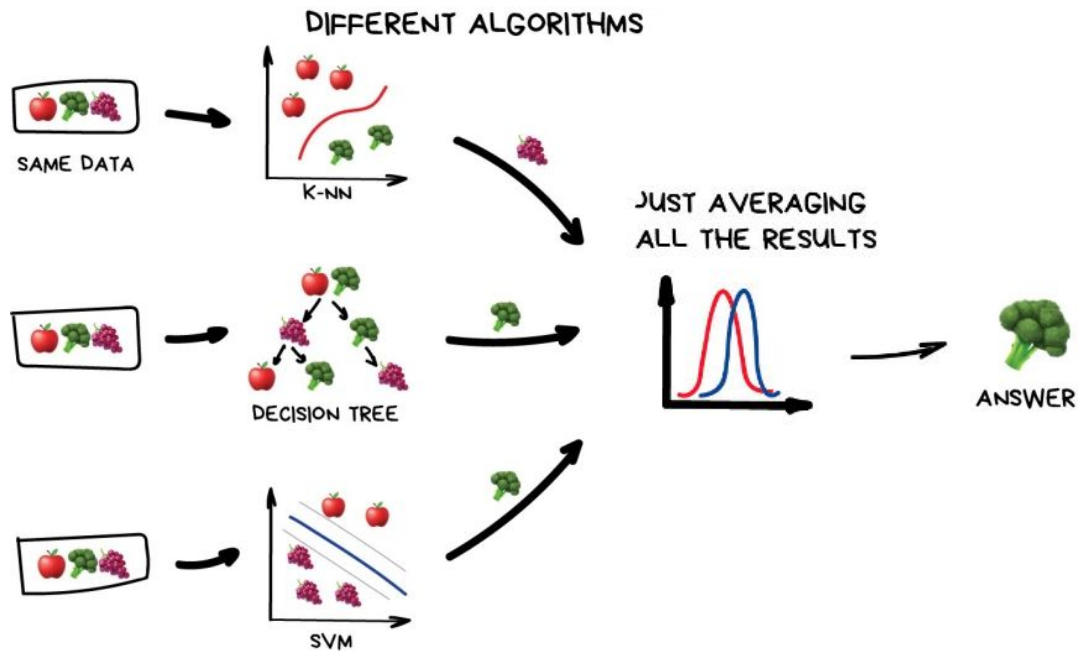
Machine learning pipeline



Lecturer Overview

- **Intro and overview**
- Voting
- Bagging
- Random Forests
- Adaboost
- Gradient Boosting
- Stacking

Ensemble



Ensemble

Two families of ensemble methods are usually distinguished:

- In **averaging methods**, the driving principle is to build **several estimators independently** and then to **average their predictions**. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced.
 - Examples: Bagging methods, Forests of randomized trees, ...
- By contrast, in boosting methods, base estimators are **built sequentially** and **one tries to reduce the bias of the combined estimator**. The motivation is to combine several weak models to produce a powerful ensemble.
 - Examples: AdaBoost, Gradient Tree Boosting, ...

Ensemble

Accuracy and diversity are two vital requirements for constructing classifier ensembles. Many methods for constructing ensembles have been developed. All methods aim to construct good individual hypotheses with uncorrelated errors (diversity). General methods:

- Manipulating the hypotheses/classifier
 - Voting and Stacking
- Manipulating the training examples
 - Bagging and Boosting
- Manipulating the input features
 - Feature Subspace
- Manipulating the training examples and features
 - Random Forest
- Manipulating the output targets
- Injecting randomness
 - Neural network ensembles

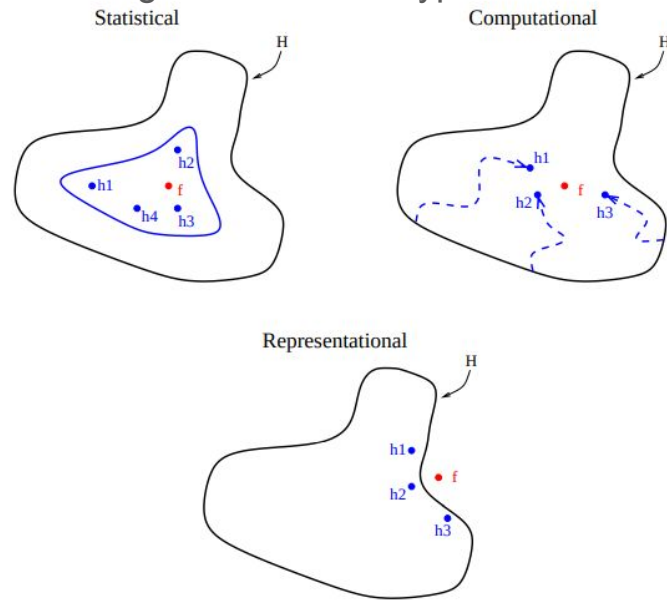


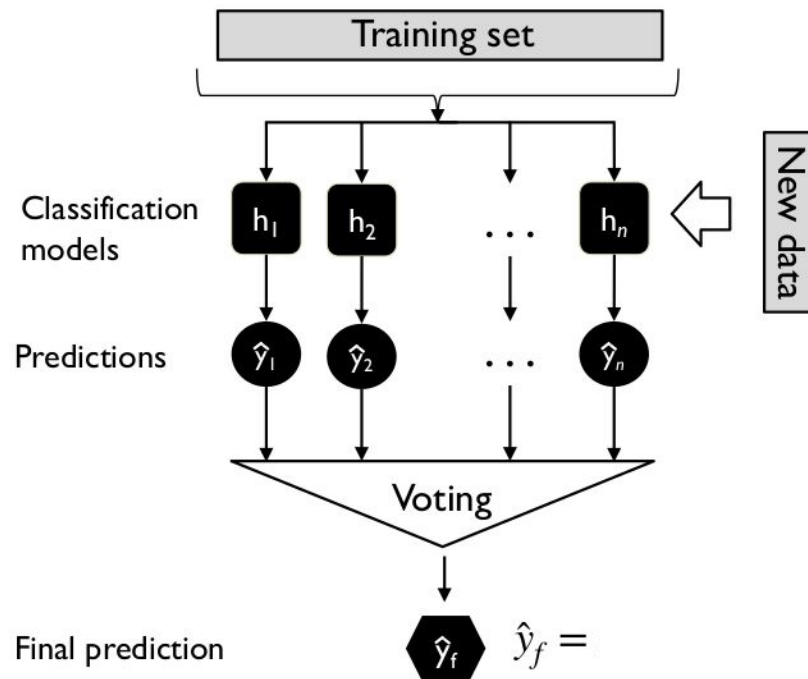
Fig. 2. Three fundamental reasons why an ensemble may work better than a single classifier

Lecturer Overview

- Intro and overview
- **Voting**
- Bagging
- Random Forests
- Adaboost
- Gradient Boosting
- Stacking

Voting Classifier

- The idea behind the Voting Classifier is to **combine conceptually different machine learning classifiers** and use a majority vote or the average predicted probabilities (soft vote) to predict the class labels. Such a classifier can be useful for a set of equally well performing model in order to balance out their individual weaknesses.
 - Majority/Hard voting
 - Soft voting

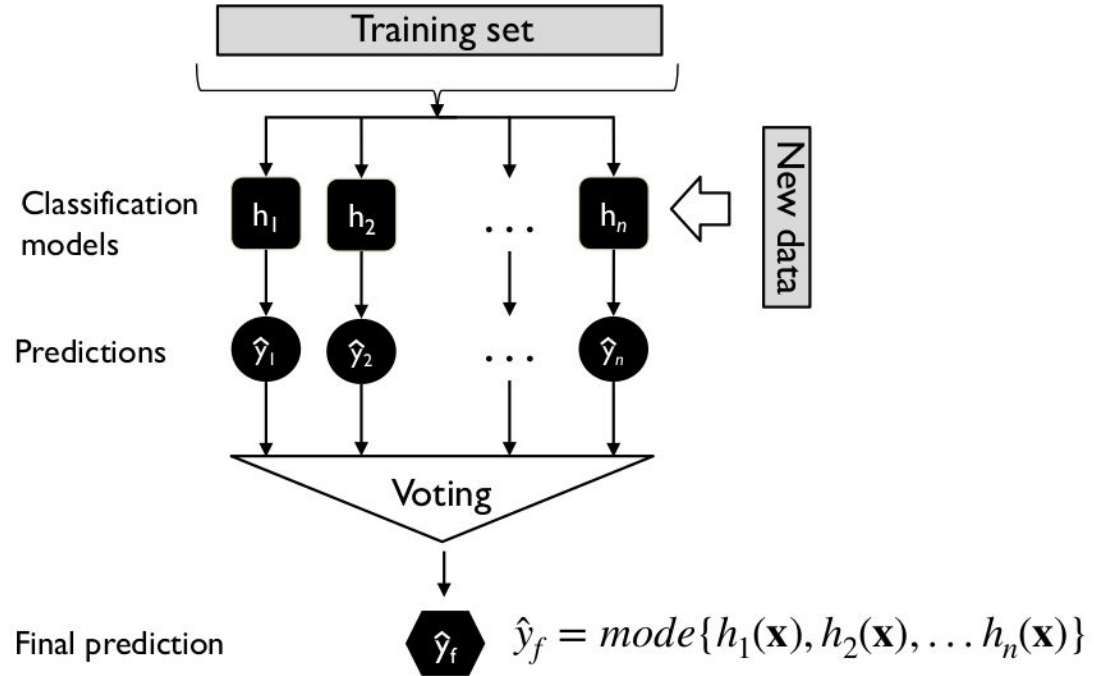


Majority Class Labels (Majority/Hard Voting)

In majority voting, the predicted class label for a particular sample is the class label that represents the majority (mode) of the class labels predicted by each individual classifier.

If the prediction for a given sample is

- classifier 1 -> class 1
- classifier 2 -> class 1
- classifier 3 -> class 2
- the VotingClassifier (with voting='hard') would classify the sample as “class 1” based on the majority class label.



where $h_i(\mathbf{x}) = \hat{y}_i$

Majority Class Labels (Majority/Hard Voting)

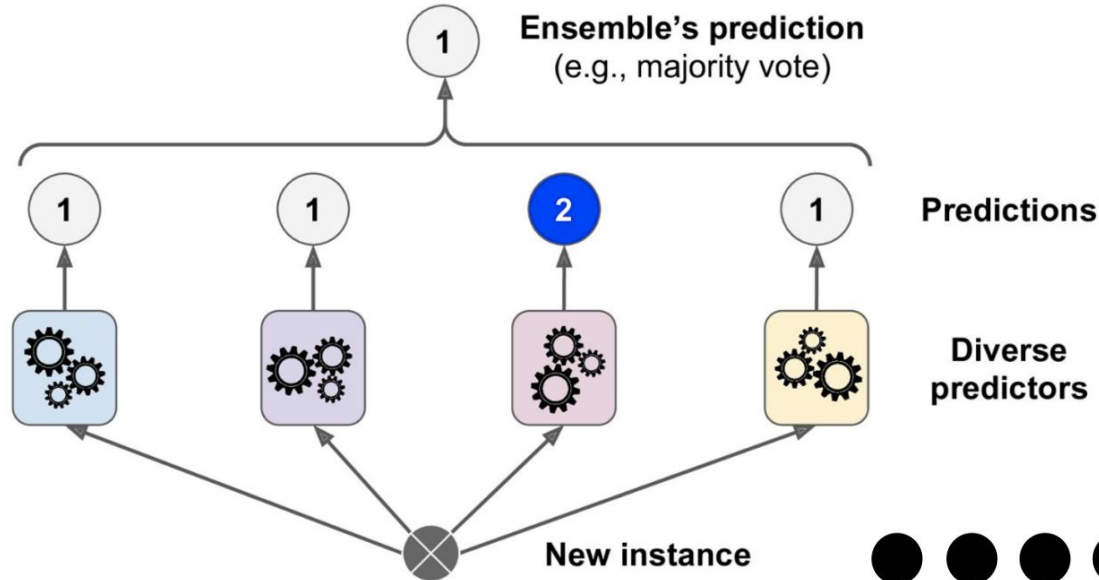
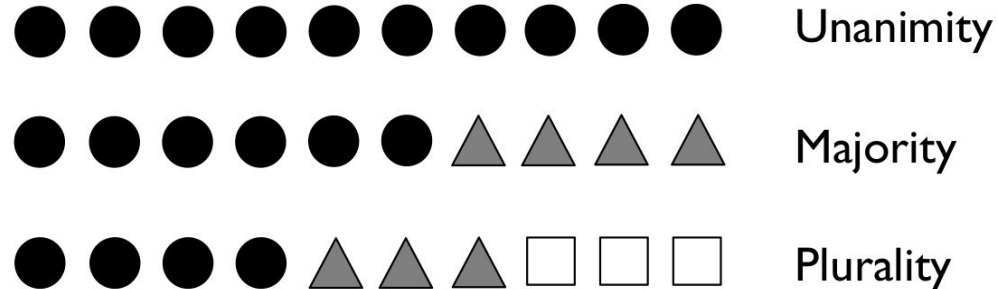


Figure 7-2. Hard voting classifier predictions



Why ensembles (majority voting) works?

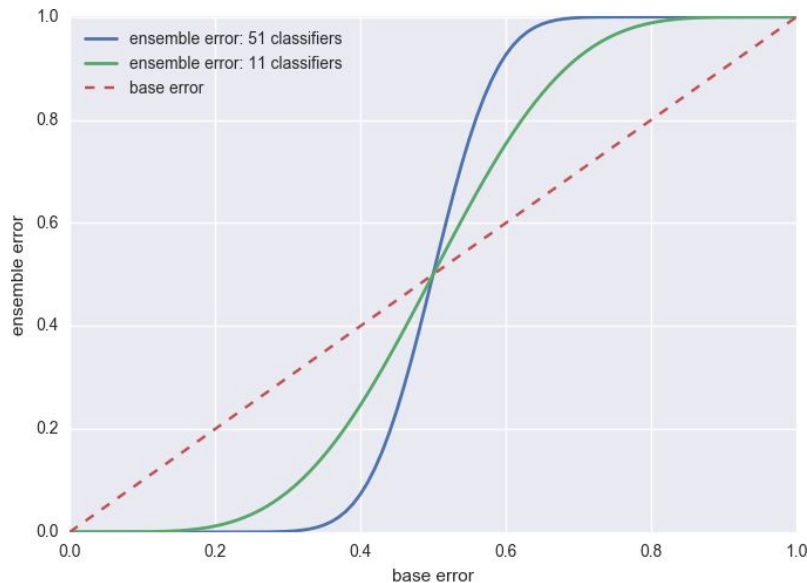
As long as each base classifier does better than random guess ($\epsilon < 0.5$), the voting classifier further reduces the error. And when that happens, the more base classifiers the better. In a binary classifier we have:

Ensemble error:

$$\epsilon_{ens} = \sum_k^n \binom{n}{k} \epsilon^k (1 - \epsilon)^{n-k}$$
$$\epsilon_{ens} = \sum_{k=6}^{11} \binom{11}{k} 0.25^k (1 - 0.25)^{11-k} = 0.034$$

The assumption in this error reduction though:

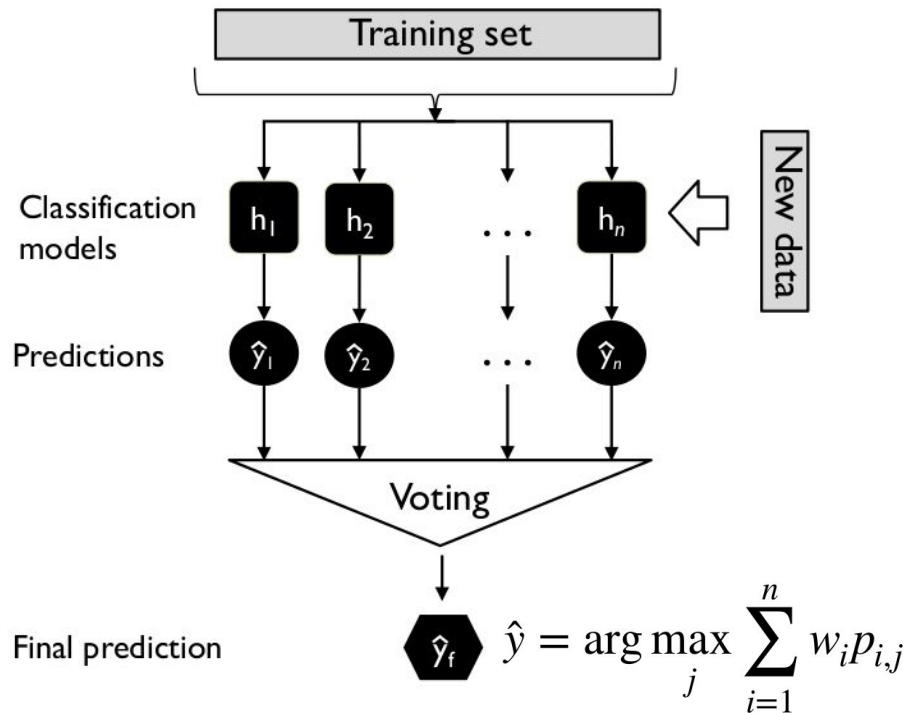
- the base classifiers are not correlated



Weighted Average Probabilities (Soft Voting)

In contrast to majority voting (hard voting), soft voting returns the class label as argmax of the sum of predicted probabilities.

- $p_{i,j}$ predicted class membership probability of the i th classifier for class label j
- w_i optional weighting parameter, default $w_i = 1/n$, $\forall w_i \in \{w_1, \dots, w_n\}$



Weighted Average Probabilities (Soft Voting)

Assuming the example in the previous section was a binary classification task with class labels $i \in \{0, 1\}$, our ensemble could make the following prediction:

- $C_1(\mathbf{x}) \rightarrow [0.9, 0.1]$
- $C_2(\mathbf{x}) \rightarrow [0.8, 0.2]$
- $C_3(\mathbf{x}) \rightarrow [0.4, 0.6]$

$$\hat{y} = \arg \max_j \sum_{i=1}^n w_i p_{i,j}$$

Using uniform weights, we compute the average probabilities:

$$p(i_0 | \mathbf{x}) = \frac{0.9 + 0.8 + 0.4}{3} = 0.7$$

$$p(i_1 | \mathbf{x}) = \frac{0.1 + 0.2 + 0.6}{3} = 0.3$$

$$\hat{y} = \arg \max_i [p(i_0 | \mathbf{x}), p(i_1 | \mathbf{x})] = 0$$

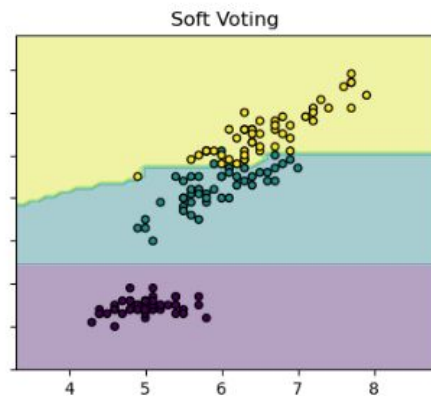
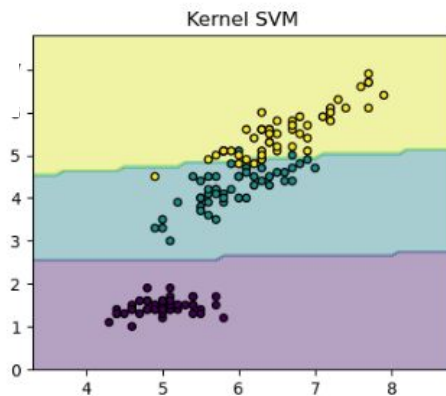
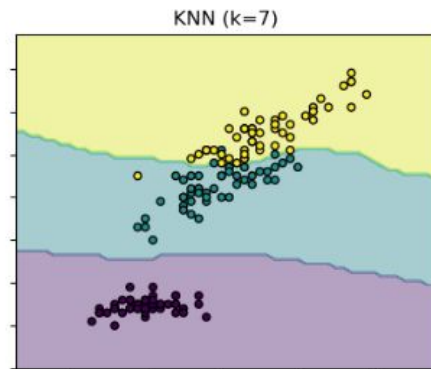
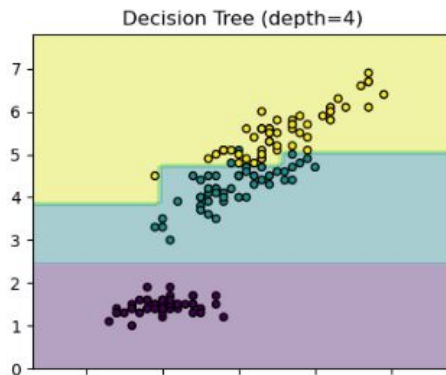
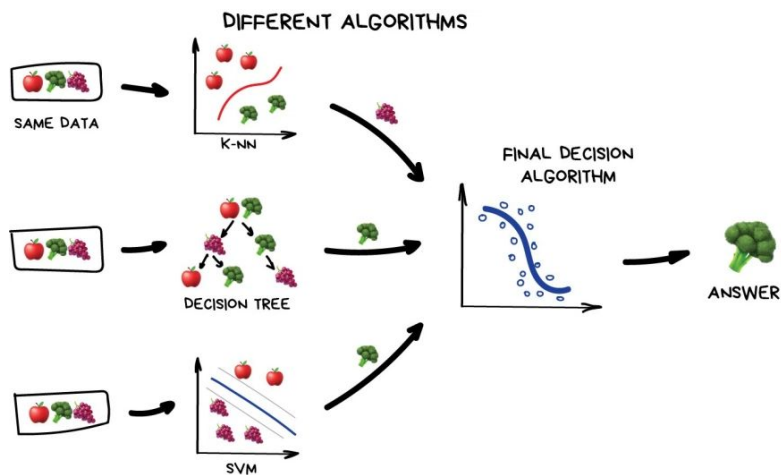
However, assigning the weights $\{0.1, 0.1, 0.8\}$ would yield a prediction $\hat{y} = 1$:

$$p(i_0 | \mathbf{x}) = 0.1 \times 0.9 + 0.1 \times 0.8 + 0.8 \times 0.4 = 0.49$$

$$p(i_1 | \mathbf{x}) = 0.1 \times 0.1 + 0.2 \times 0.1 + 0.8 \times 0.6 = 0.51$$

$$\hat{y} = \arg \max_i [p(i_0 | \mathbf{x}), p(i_1 | \mathbf{x})] = 1$$

Weighted Average Probabilities (Soft Voting)



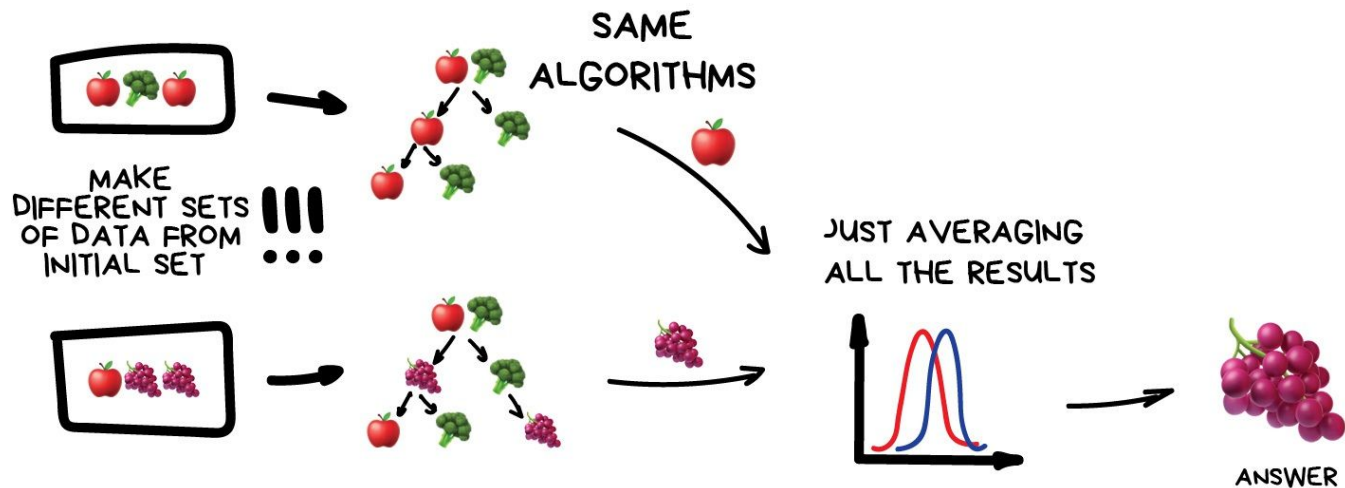
Lecturer Overview

- Intro and overview
- Voting
- **Bagging**
- Random Forests
- Adaboost
- Gradient Boosting
- Stacking

Bagging

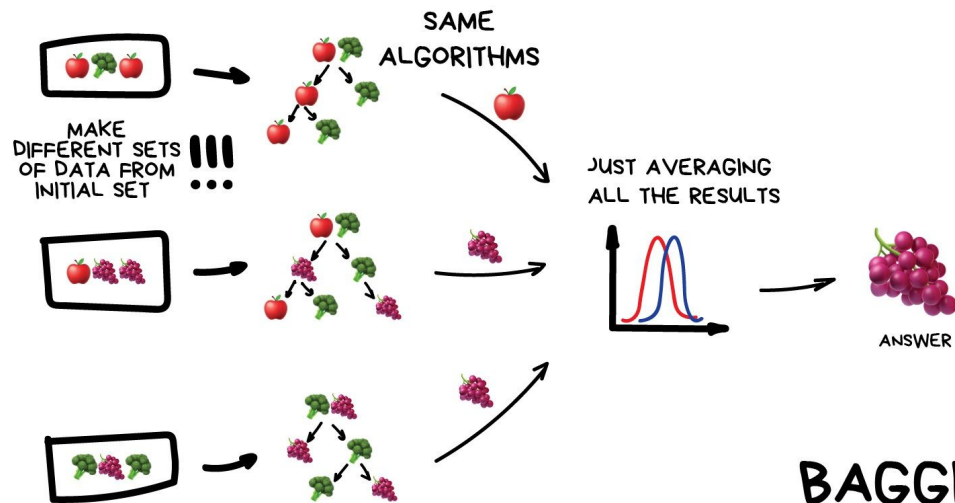
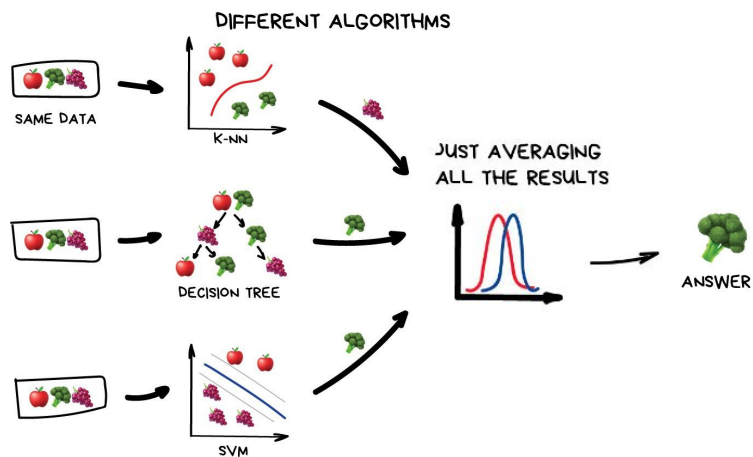
- Bootstrap Aggregating
 - Breiman, L. (1996). Bagging predictors. Machine learning, 24(2), 123-140.
- Ensemble family
 - Average methods (several estimators independently)
- Uncorrelated errors (diversity)
 - manipulating the training examples
 - manipulating the input features

Bagging



BAGGING

Voting x Bagging

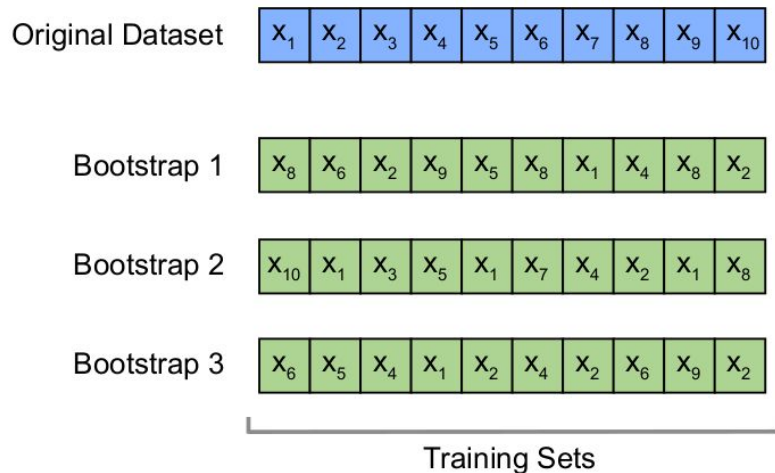


Bagging

Average methods

Algorithm 1 Bagging

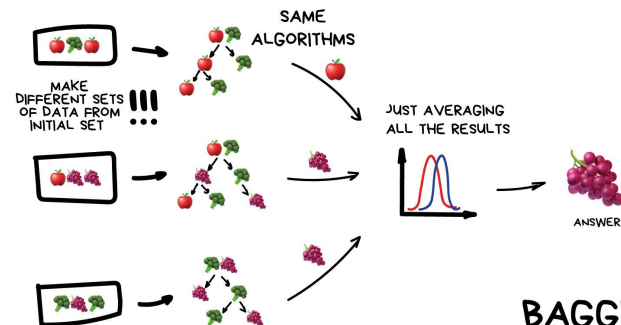
- 1: Let n be the number of bootstrap samples
 - 2:
 - 3: **for** $i=1$ to n **do**
 - 4: Draw bootstrap sample of size m , \mathcal{D}_i
 - 5: Train base classifier h_i on \mathcal{D}_i
 - 6: $\hat{y} = \text{mode}\{h_1(\mathbf{x}), \dots, h_n(\mathbf{x})\}$
-



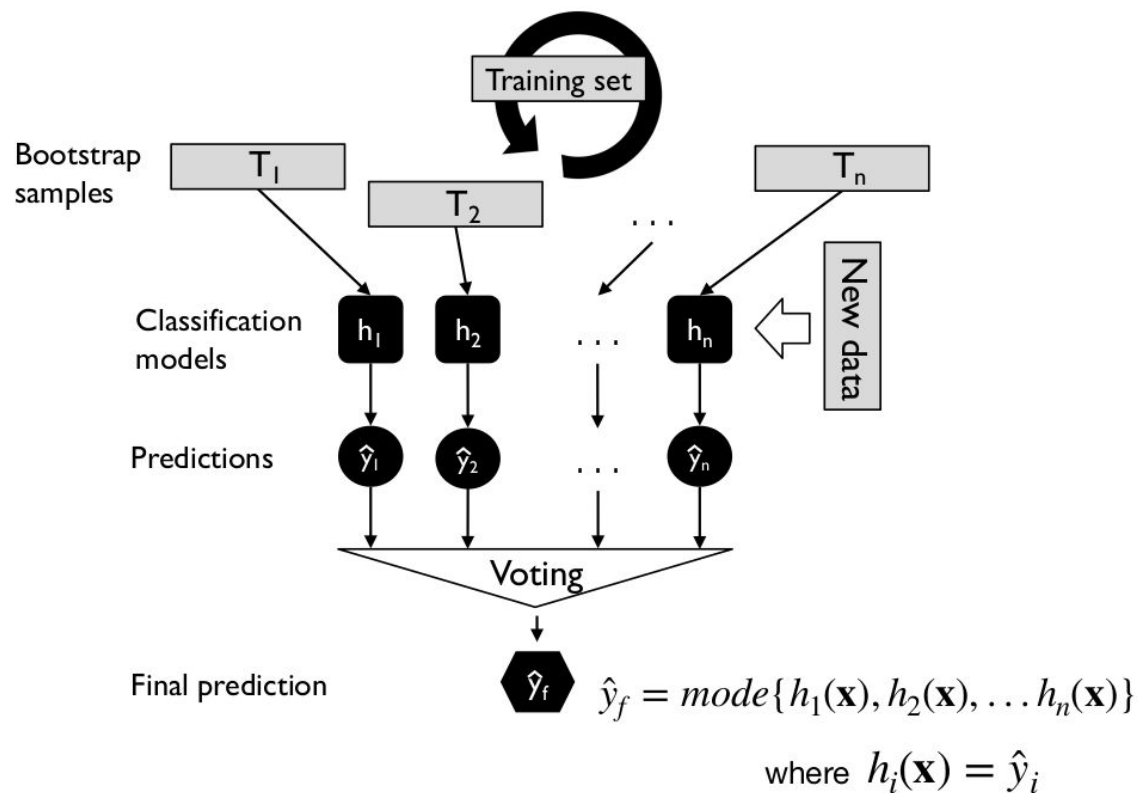
x_3	x_7	x_{10}
-------	-------	----------

x_6	x_9
-------	-------

x_3	x_7	x_8	x_{10}
-------	-------	-------	----------



Bagging



Training example indices	Bagging round 1	Bagging round 2	...
1	2	7	...
2	2	3	...
3	1	2	...
4	3	1	...
5	7	1	...
6	2	7	...
7	4	7	...

\downarrow
 h_1

\downarrow
 h_2

\downarrow
 h_n

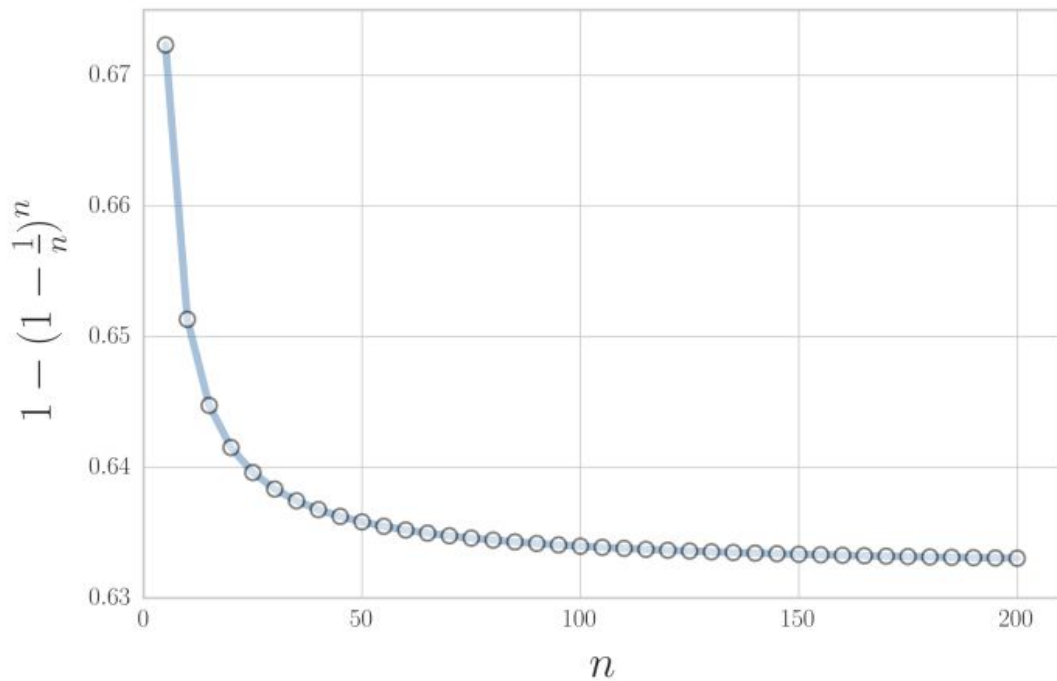
Bagging

Sampling probability

$$P(\text{not chosen}) = \left(1 - \frac{1}{n}\right)^n,$$

$$\frac{1}{e} \approx 0.368, \quad n \rightarrow \infty.$$

$$P(\text{chosen}) = 1 - \left(1 - \frac{1}{n}\right)^n \approx 0.632$$



Bagging methods

Bagging methods form a class of algorithms which build several instances of a black-box estimator on random subsets of the original training set and then aggregate their individual predictions to form a final prediction.

Bagging methods come in **many flavours** but mostly **differ** from each other by the **way they draw random subsets** of the training set:

- When random subsets of the dataset are drawn as **random subsets of the samples**, then this algorithm is known as **Pasting** [B1999].
- When **samples are drawn with replacement**, then the method is known as **Bagging** [B1996].
- When random subsets of the dataset are drawn as **random subsets of the features**, then the method is known as **Random Subspaces** [H1998].
- Finally, when base estimators are built on **subsets of both samples and features**, then the method is known as **Random Patches** [LG2012].

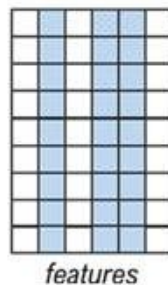
Bagging methods



bagging

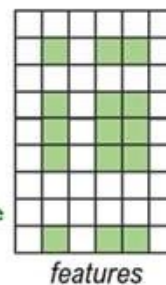
(sample instances)

`max_samples=0.75,`
`bootstrap=True,`
`max_features=1.0,`
`bootstrap_features=False`



random subspaces
(sample features)

`max_samples=1.0,`
`bootstrap=False,`
`max_features=0.5,`
`bootstrap_features=True`



random patches
(sample both)

`max_samples=0.75,`
`bootstrap=True,`
`max_features=0.5,`
`bootstrap_features=True`

Bias-Variance Decomposition

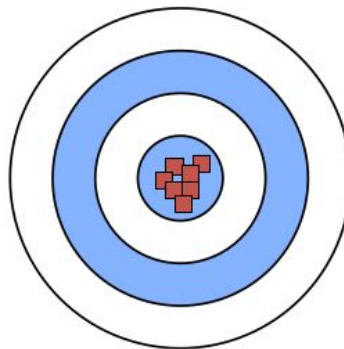
Loss = Bias + Variance + Noise

(more technical details in next lecture on model evaluation)

Low Variance
(Precise)

High Variance
(Not Precise)

Low Bias
(Accurate)



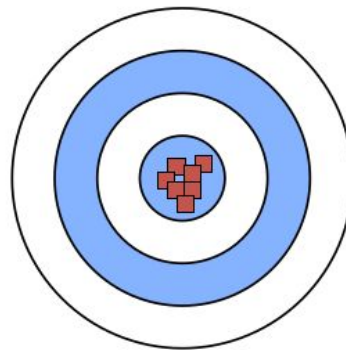
High Bias
(Not Accurate)

Bias-Variance Decomposition

Loss = Bias + Variance + Noise

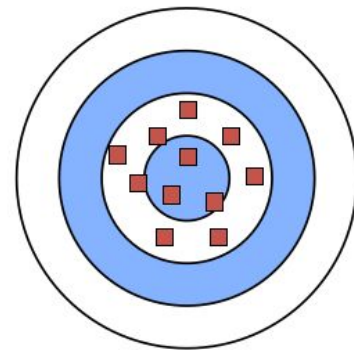
(more technical details in next lecture on model evaluation)

Low Bias
(Accurate)



Low Variance
(Precise)

High Bias
(Not Accurate)



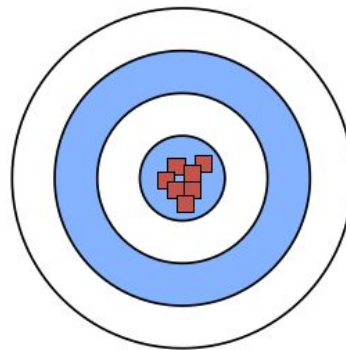
High Variance
(Not Precise)

Bias-Variance Decomposition

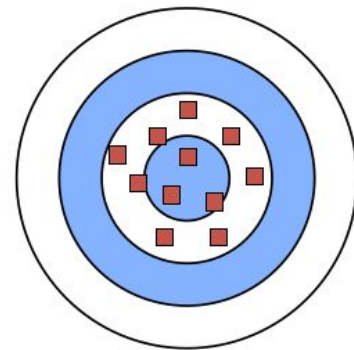
Loss = Bias + Variance + Noise

(more technical details in next lecture on model evaluation)

Low Bias
(Accurate)

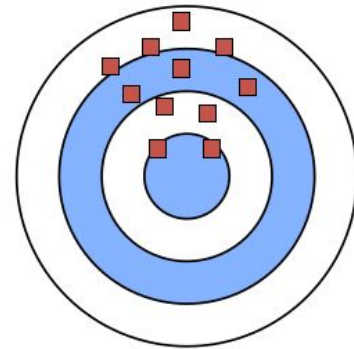


Low Variance
(Precise)



High Variance
(Not Precise)

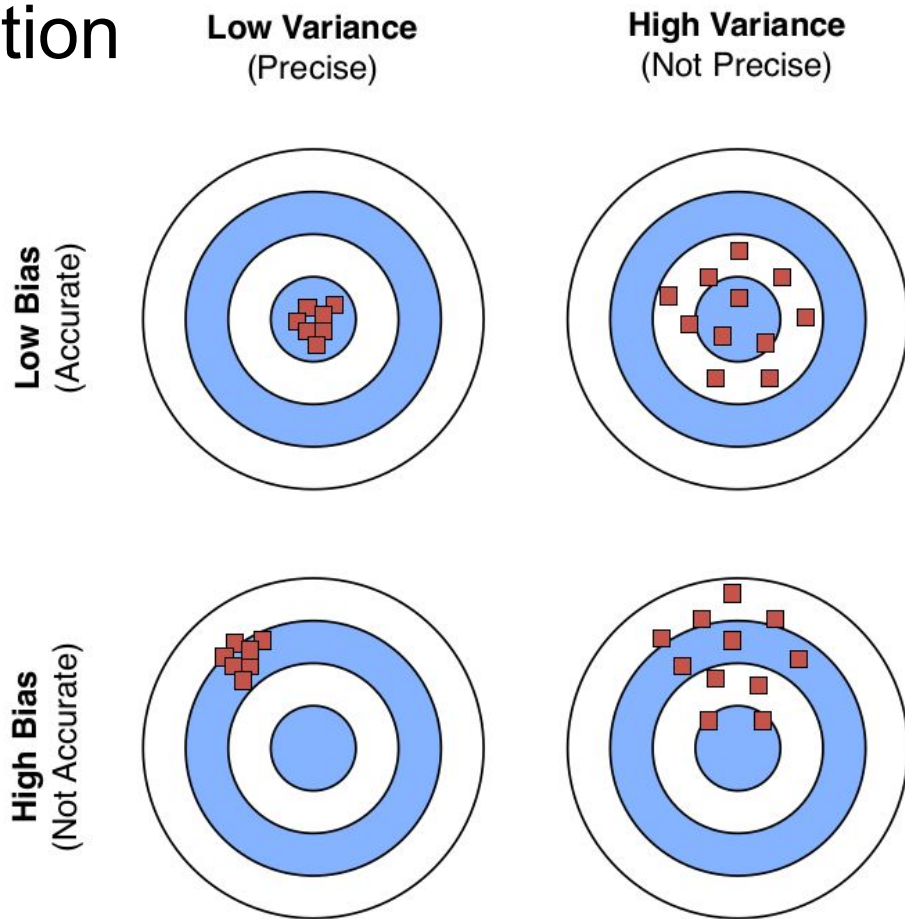
High Bias
(Not Accurate)



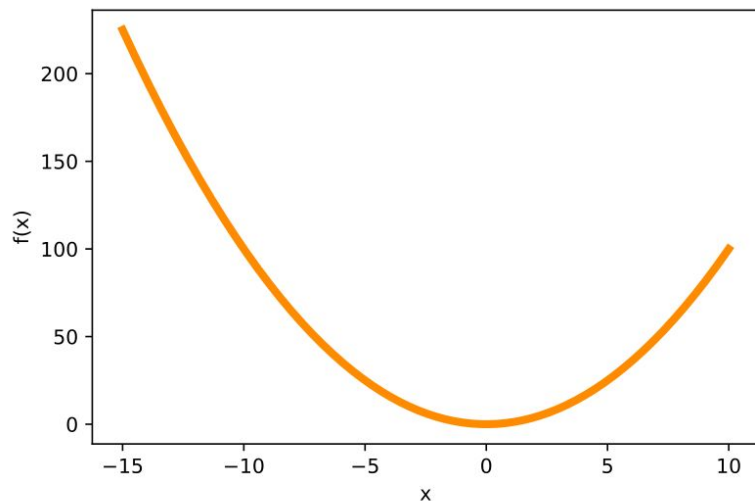
Bias-Variance Decomposition

Loss = Bias + Variance + Noise

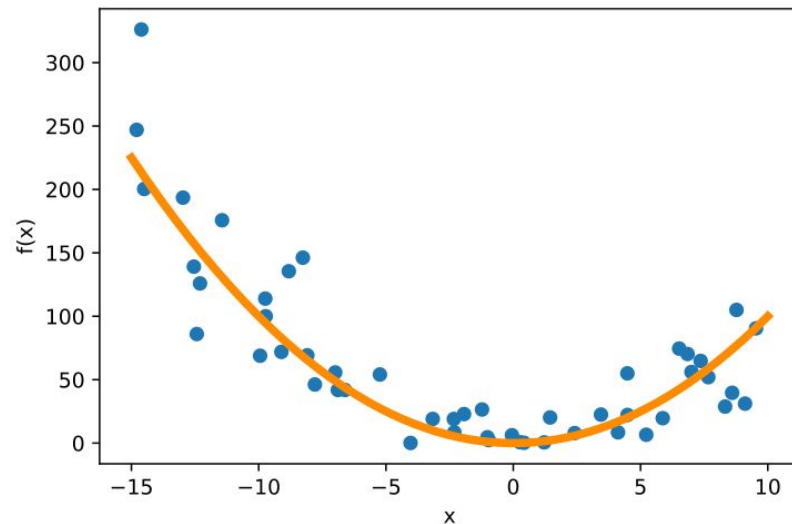
(more technical details in next lecture on model evaluation)



Bias and Variance Example



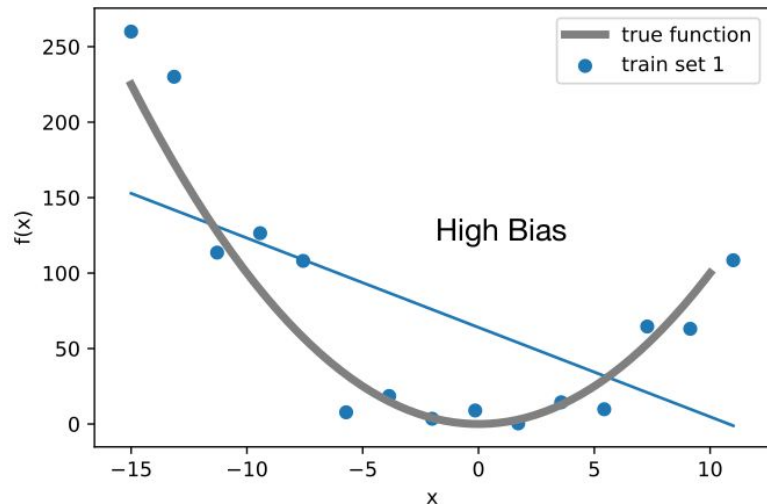
where $f(x)$ is some true (target) function



where $f(x)$ is some true (target) function

the blue dots are a training dataset;
here, I added some random Gaussian noise

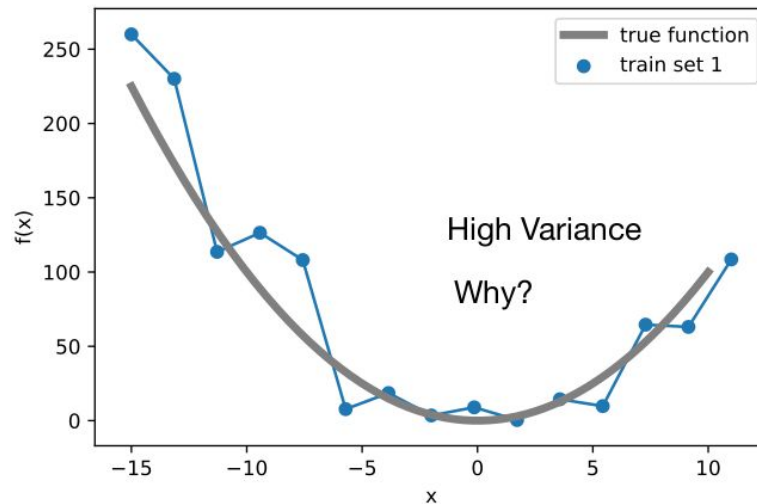
Bias and Variance Example



where $f(x)$ is some true (target) function

the blue dots are a training dataset;
here, I added some random Gaussian noise

here, suppose I fit a simple linear model (linear regression)
or a decision tree stump

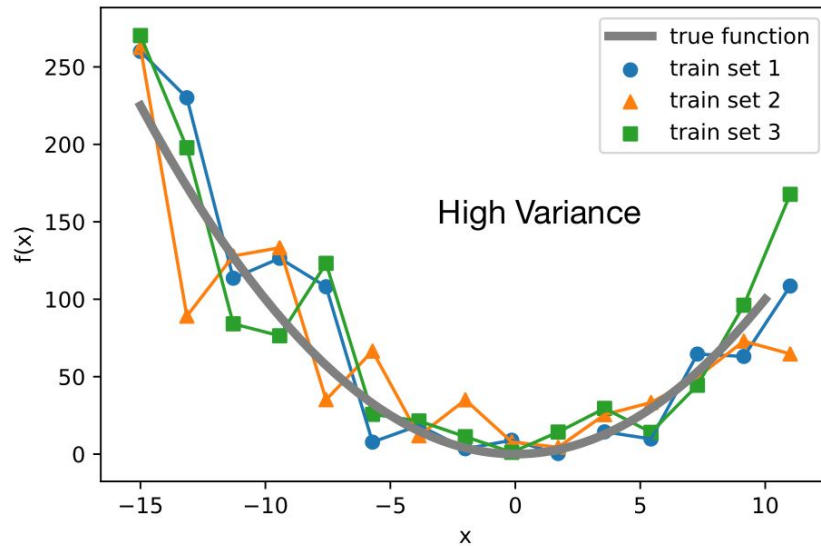
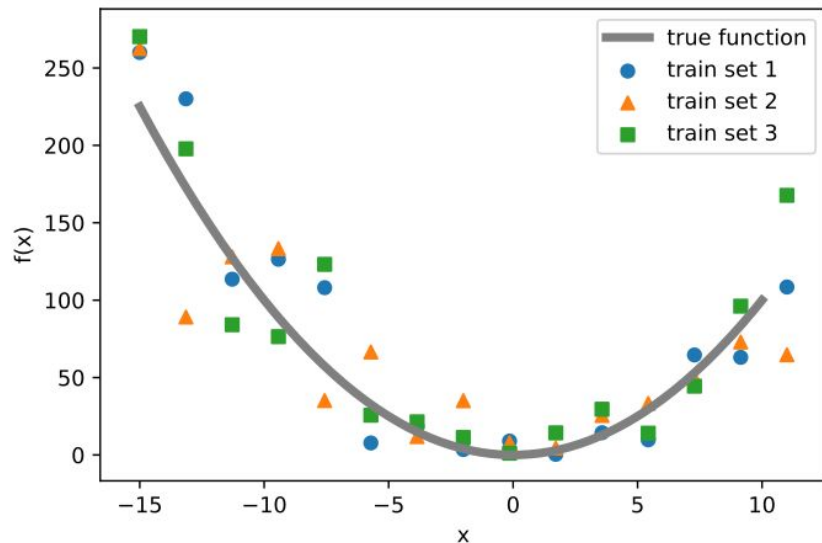


where $f(x)$ is some true (target) function

the blue dots are a training dataset;
here, I added some random Gaussian noise

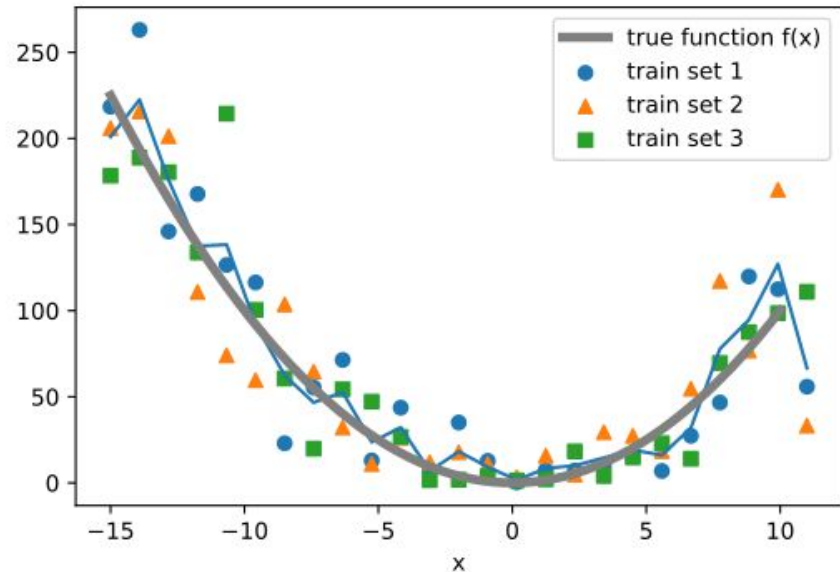
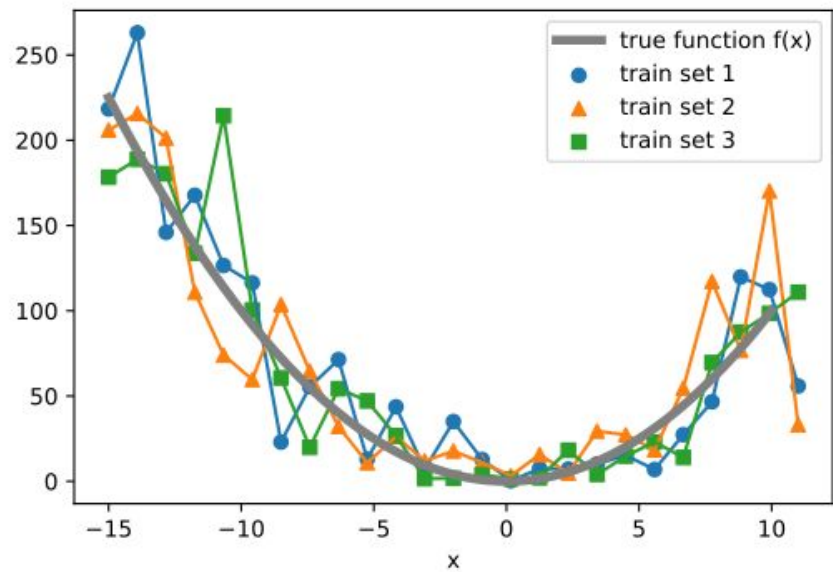
here, suppose I fit an unpruned decision tree

Bias and Variance Example



where $f(x)$ is some true (target) function

suppose we have multiple training sets

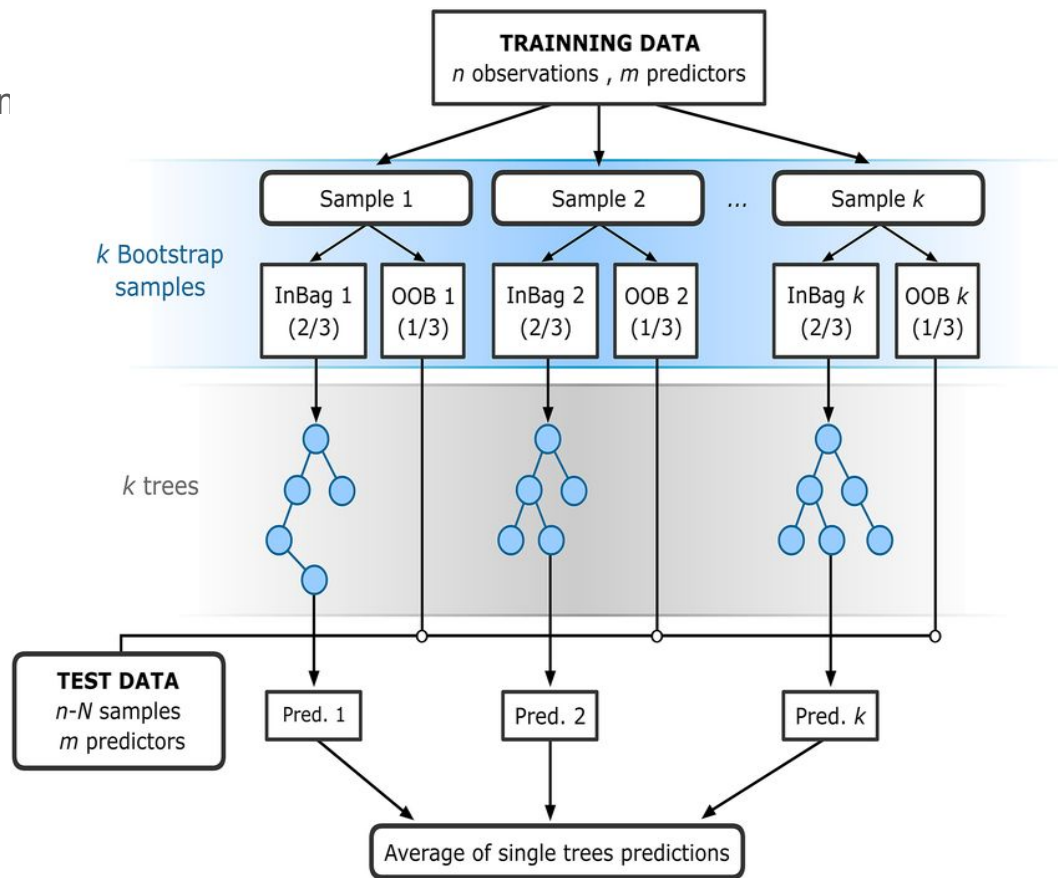


Lecturer Overview

- Intro and overview
- Voting
- Bagging
- **Random Forests**
- Adaboost
- Gradient Boosting
- Stacking

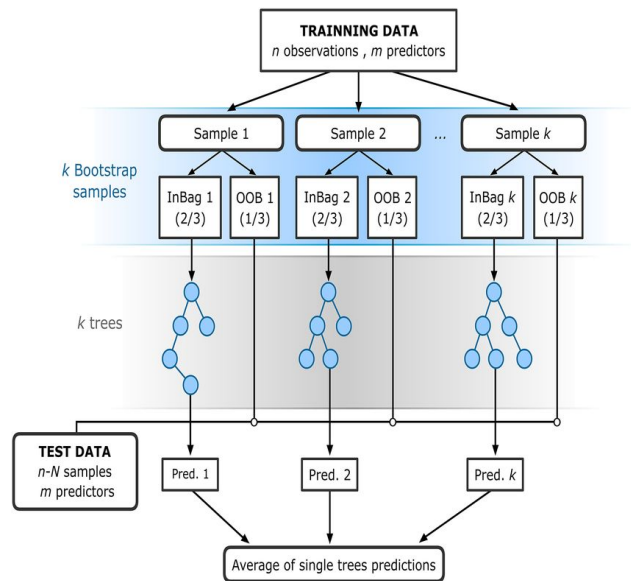
Random Forests

Each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set.



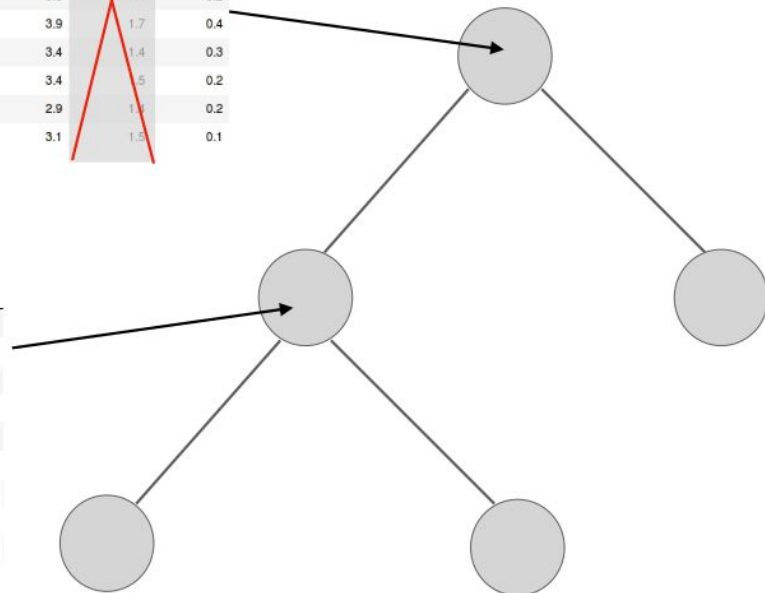
Random Forests

Furthermore, when splitting each node during the construction of a tree, the **best split** is **found** either from **all input** features or a **random subset of features**



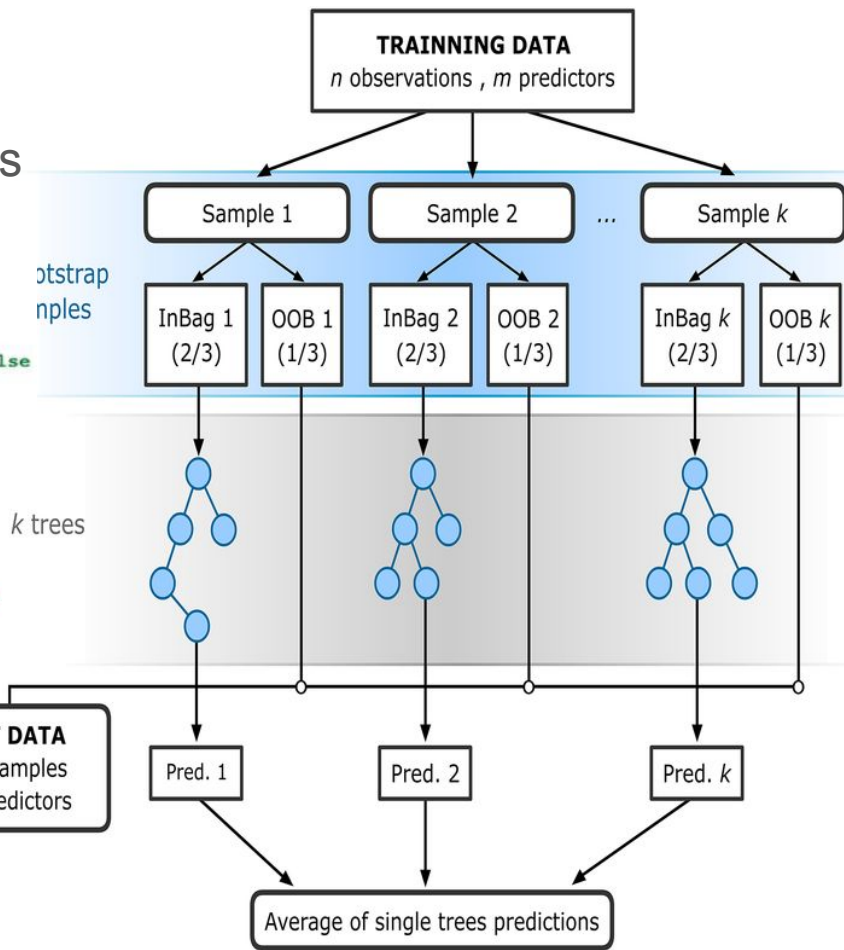
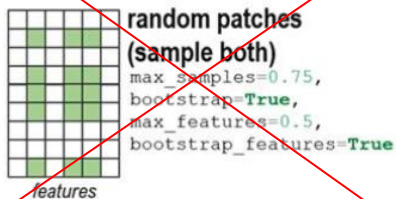
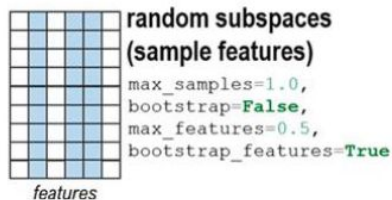
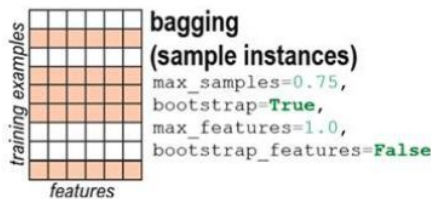
	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
6	4.6	3.4	1.4	0.3
7	5.0	3.4	1.5	0.2
8	4.5	2.9	1.1	0.2
9	4.9	3.1	1.5	0.1

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
2	4.7	3.2	1.3	0.2
4	5.0	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4
9	4.5	3.1	1.5	0.1



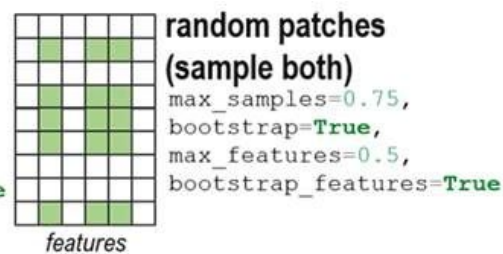
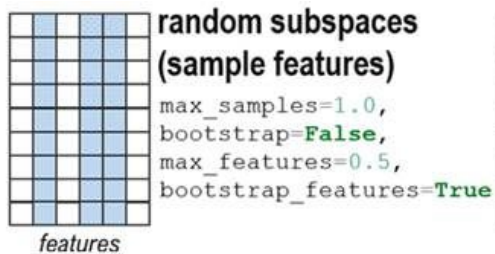
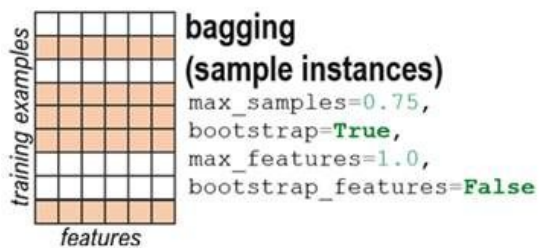
Random Forests

Bagging w. trees + random feature subsets



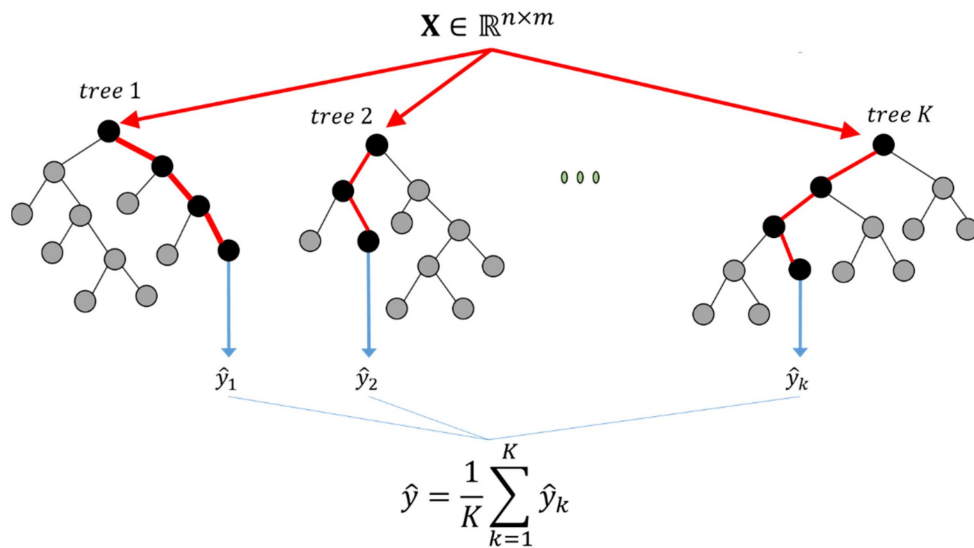
Random Feature Subset for each Tree or Node?

- Ho, Tin Kam. “The **random subspace** method for constructing decision forests.” IEEE transactions on pattern analysis and machine intelligence 20.8 (1998): 832-844.
 - “Our method relies on an autonomous, pseudo-random procedure to select a small number of dimensions from a given feature space ...”
 - Tin Kam Ho used the “random subspace method,” where **each tree got a random subset of features**.
- Breiman, Leo. “**Random Forests**” Machine learning 45.1 (2001): 5-32.
 - “...random forest with random features is formed by selecting at random, at **each node**, a **small group of input variables** to split on.”



Soft voting x Hard Voting

- Breiman, Leo. “**Random Forests**” Machine learning 45.1 (2001): 5-32
 - In contrast to the original publication the scikit-learn implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class.
 - Soft voting x Hard Voting



(Loose) Upper Bound for the Generalization Error

Breiman, Leo. "Random Forests" Machine learning 45.1 (2001): 5-32

$$\text{PE} \leq \frac{\bar{\rho} \cdot (1 - s^2)}{s^2}$$

$\bar{\rho}$: Average correlation among trees

s : "Strength" of the ensemble

Lecturer Overview

- Intro and overview
- Voting
- Bagging
- Random Forests
 - **Extremely Randomized Trees**
- Adaboost
- Gradient Boosting
- Stacking

Extremely Randomized Trees (ExtraTrees)

Random Forest random components:

- **each tree** in the ensemble is built from a sample drawn with replacement (i.e., a **bagging**) from the training set
- when splitting each node during the construction of a tree, the **best split** is found on random subset of features (i.e. **random subspaces**)

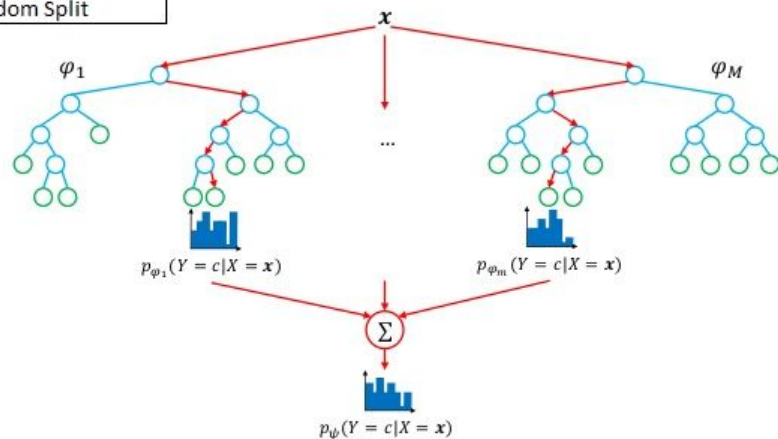
ExtraTrees algorithm adds one more random component

- instead of looking for the most discriminative thresholds, **thresholds are drawn at random** for each candidate feature and **the best of these randomly-generated thresholds is picked as the splitting rule**

This usually allows to **reduce the variance** of the model a bit more, **at the expense of a slightly greater increase in bias**

Extremely Randomized Trees (ExtraTrees)

	Decision Tree	Random Forest	Extra Trees
Number of trees	1	Many	Many
No of features considered for split at each decision node	All Features	Random subset of Features	Random subset of Features
Boostrapping(Drawing Sampling without replacement)	Not applied	Yes	No
How split is made	Best Split	Best Split	Random Split



Randomization

- Bootstrap samples
 - Random selection of $K \leq p$ split variables
 - Random selection of the threshold
- } Random Forests
 } Extra-Trees

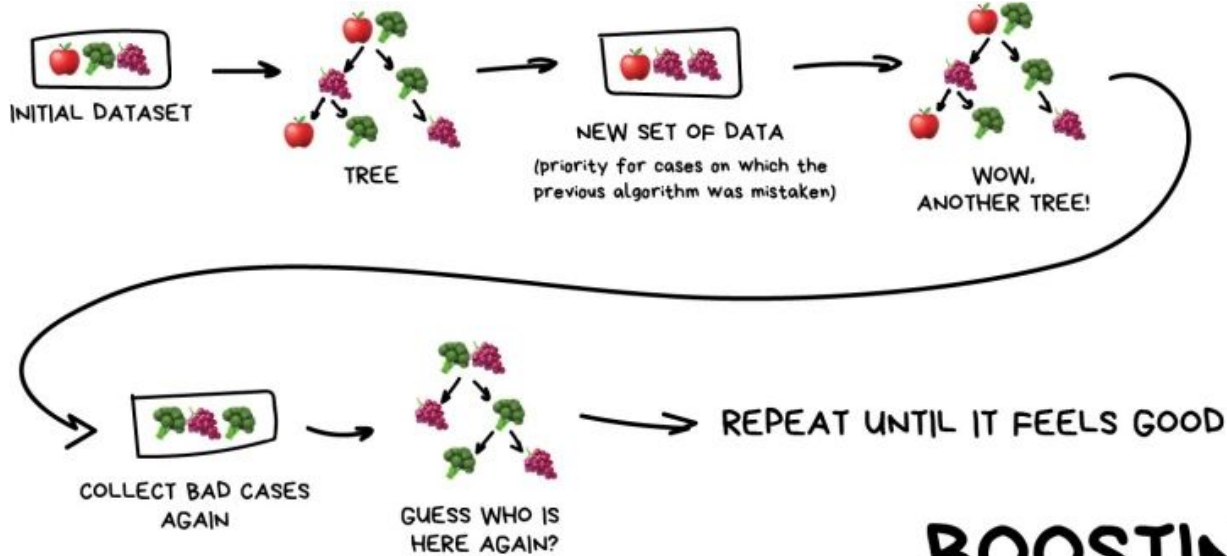
Lecturer Overview

- Intro and overview
- Voting
- Bagging
- Random Forests
- **Adaboost**
- Gradient Boosting
- Stacking

AdaBoost

- Adaptive Boosting
 - Freund, Y., & Schapire, R. E. (1997). **A decision-theoretic generalization of on-line learning and an application to boosting**. Journal of computer and system sciences, 55(1), 119-139.
- Ensemble family
 - Boosting methods (several estimators sequentially)
- Uncorrelated errors (diversity)
 - manipulating the training examples
 - weak learners

AdaBoost



BOOSTING

AdaBoost

The core principle of AdaBoost is to fit a **sequence of weak learners** (i.e., models that are only slightly better than random guessing, such as small decision trees) on **repeatedly modified versions of the data**

Average x sequential methods

Average methods (Voting, Bagging, Random Trees)

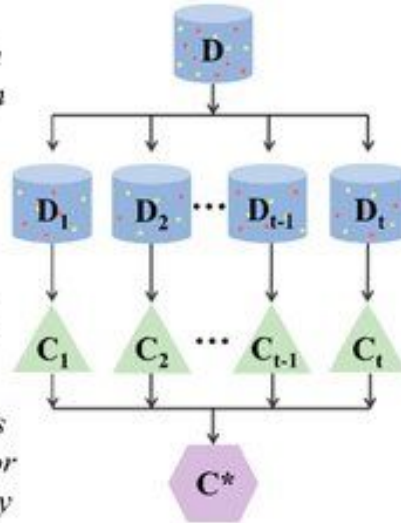
Sequential methods (Adaboost)

(A) bagging

step 1
create multiple data sets through random sampling with replacement

step 2
build multiple learners in parallel

step 3
combine all learners using an averaging or majority-vote strategy

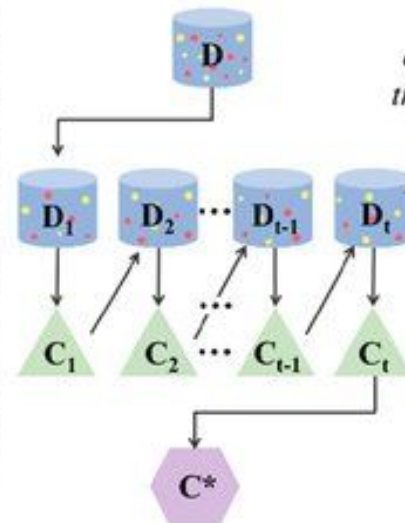


(B) boosting

step 1
create multiple data sets through random sampling with replacement over weighted data

step 2
build learners sequentially

step 3
combine all learners using a weighted-averaging strategy



AdaBoost

- Initialize a weight vector with uniform weights
- Loop
 - Apply weak learner* to weighted training examples
 - instead of orig. training set, may draw bootstrap samples with weighted probability
 - Increase weight for misclassified examples
- Soft voting on trained classifiers

* a learner slightly better than random guessing

Weak learners

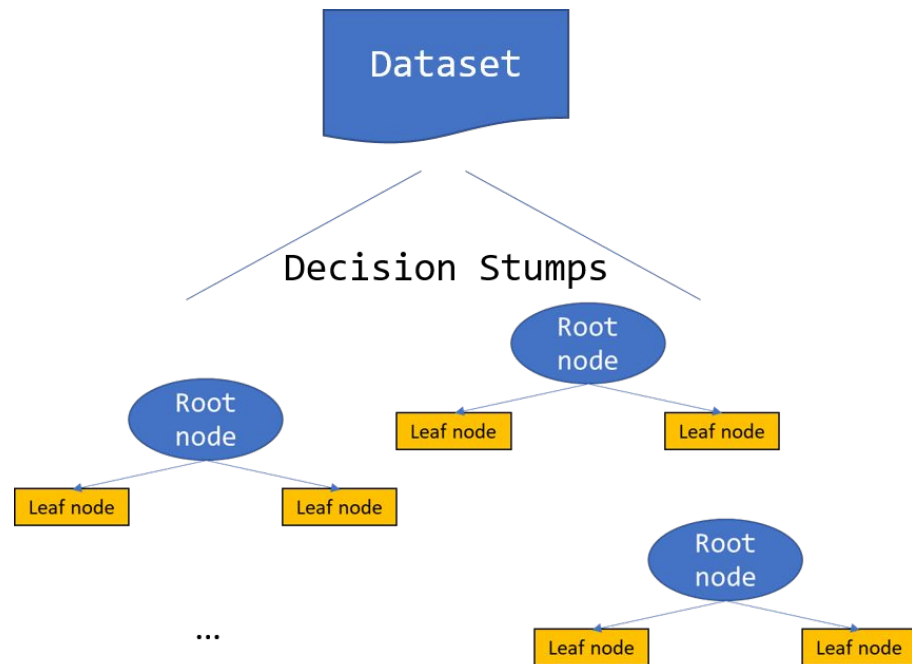
Weak classifier, here: **decision tree stump** for binary classification problem with labels **-1, 1**

$$h(\mathbf{x}) = s(\mathbf{1}(x_k \geq t))$$

where

$$s(x) \in \{-1, 1\}$$

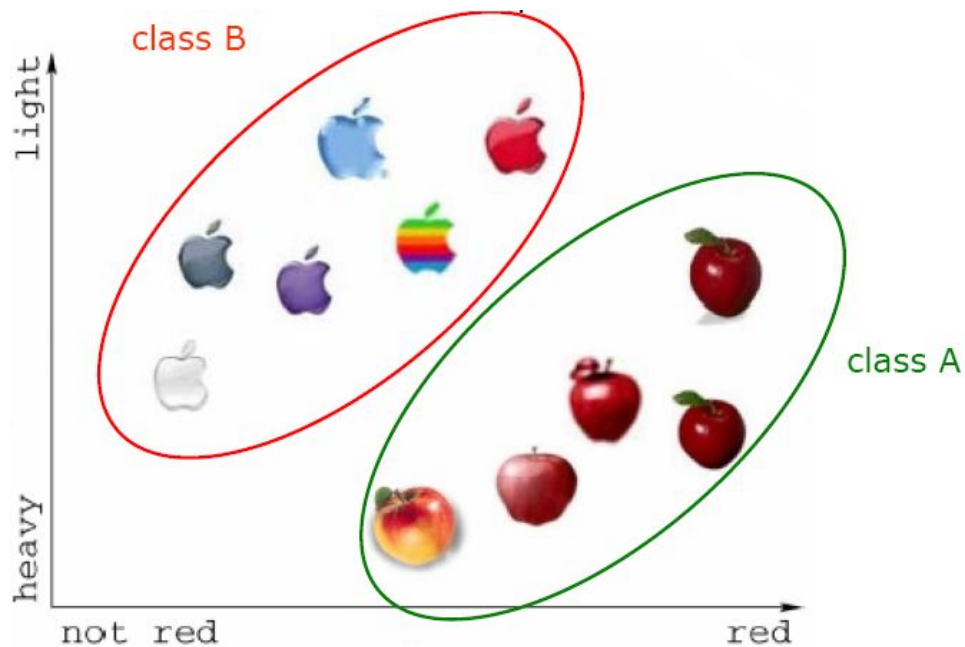
$$k \in \{1, \dots, K\} \text{ (} K \text{ is the number of features)}$$



Adaboost

Example 1

- plastic apples x real apples



AdaBoost

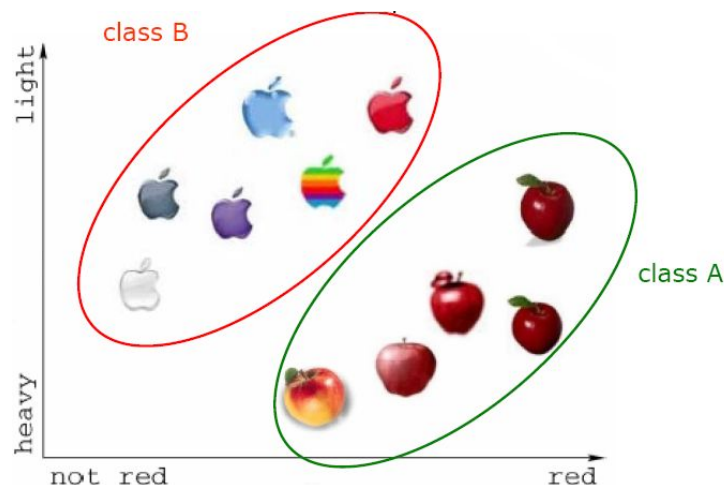
Algorithm 1 AdaBoost

- 1: Initialize k : the number of AdaBoost rounds
 - 2: Initialize \mathcal{D} : the training dataset, $\mathcal{D} = \{\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, \dots, \langle \mathbf{x}^{[n]}, y^{[n]} \rangle\}$
 - 3: Initialize $w_1(i) = 1/n$, $i = 1, \dots, n$, $\mathbf{w}_1 \in \mathbb{R}^n$
 - 4:
 - 5: **for** $r=1$ to k **do**
 - 6: For all i : $\mathbf{w}_r(i) := w_r(i) / \sum_i w_r(i)$ [normalize weights]
 - 7: $h_r := \text{FitWeakLearner}(\mathcal{D}, \mathbf{w}_r)$
 - 8: $\epsilon_r := \sum_i w_r(i) \mathbf{1}(h_r(i) \neq y_i)$ [compute error]
 - 9: if $\epsilon_r > 1/2$ then stop
 - 10: $\alpha_r := \frac{1}{2} \log[(1 - \epsilon_r)/\epsilon_r]$ [small if error is large and vice versa]
 - 11: $w_{r+1}(i) := w_r(i) \times \begin{cases} e^{-\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) = y^{[i]} \\ e^{\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) \neq y^{[i]} \end{cases}$
 - 12: Predict: $h_{ens}(\mathbf{x}) = \arg \max_j \sum_r \alpha_r \mathbf{1}[h_r(\mathbf{x}) = j]$
 - 13:
-

Initialization

- 1: Initialize k : the number of AdaBoost rounds
- 2: Initialize \mathcal{D} : the training dataset, $\mathcal{D} = \{\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, \dots, \mathbf{x}^{[n]}, y^{[n]} \rangle\}$
- 3: Initialize $w_1(i) = 1/n$, $i = 1, \dots, n$, $\mathbf{w}_1 \in \mathbb{R}^n$

- $w_1(i) = 1/11 \quad \forall i \in \mathcal{D}$
 - $w_1(i) = \mathbf{0,091}$

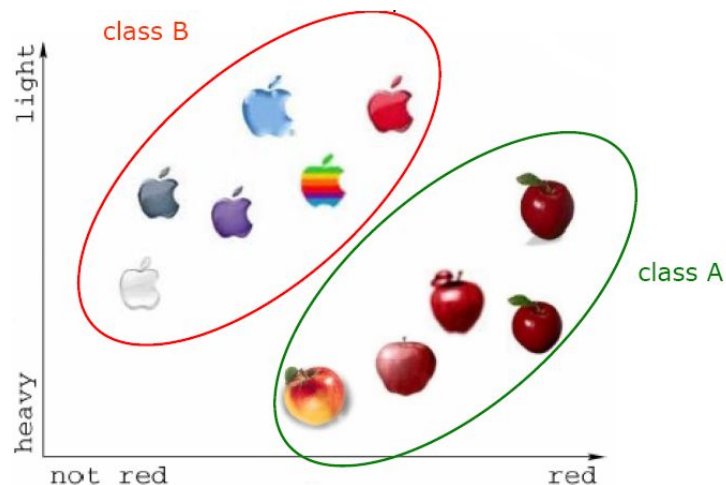


Normalize weights

5: **for** $r=1$ to k **do**

6: For all i : $\mathbf{w}_r(i) := w_r(i) / \sum_i w_r(i)$ [normalize weights]

- $w_1(i) = 0,091/1$
 - $w_1(i) = \mathbf{0,091}$

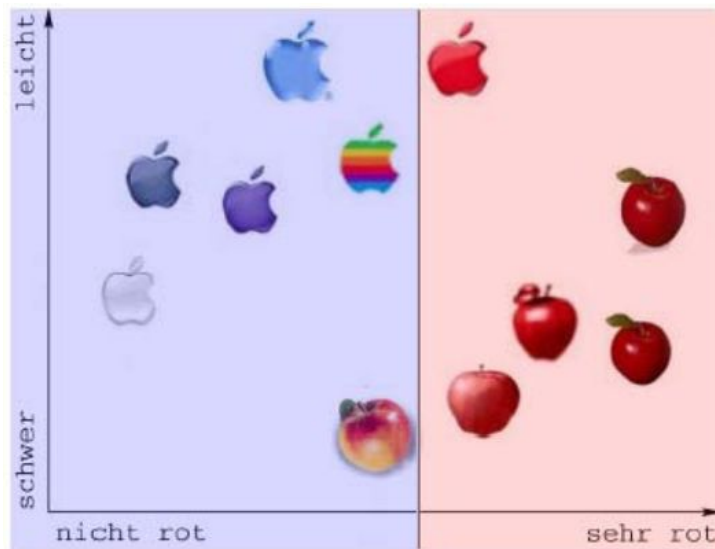


Fit a weak Learner and compute error

7: $h_r := \text{FitWeakLearner}(\mathcal{D}, \mathbf{w}_r)$

8: $\epsilon_r := \sum_i w_r(i) \mathbf{1}(h_r(i) \neq y_i)$ [compute error]

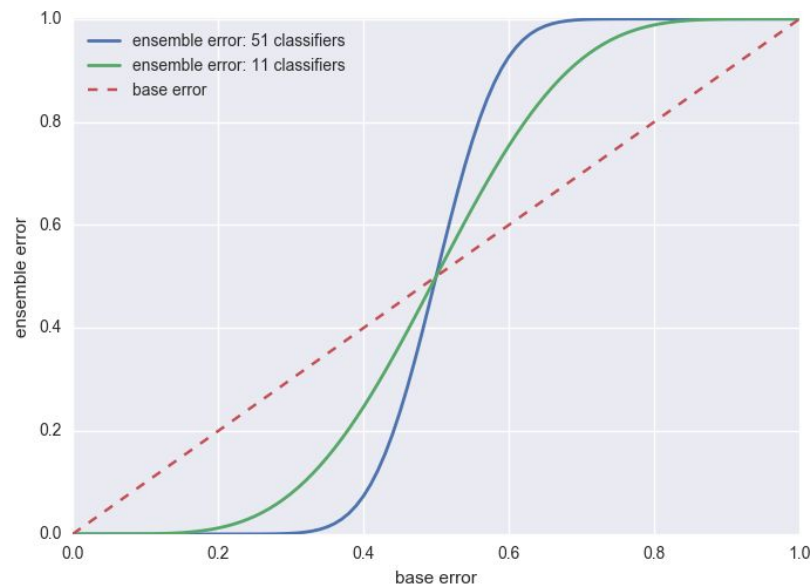
- Error (ϵ): 2/11
 - $\epsilon = 0.18$



Verify the sanity

9: if $\epsilon_r > 1/2$ then stop

If the error is greater than a random guessing, combine these classifier leads to an increase in the classification error

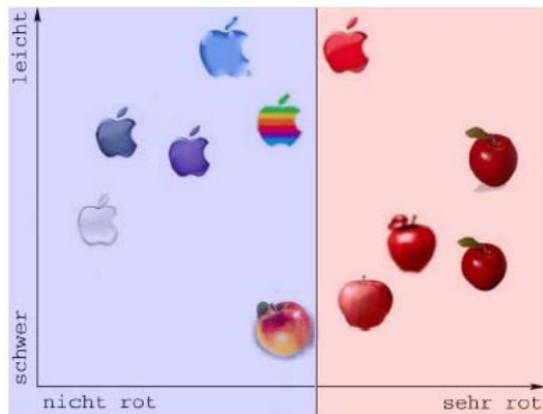


Estimador weight α

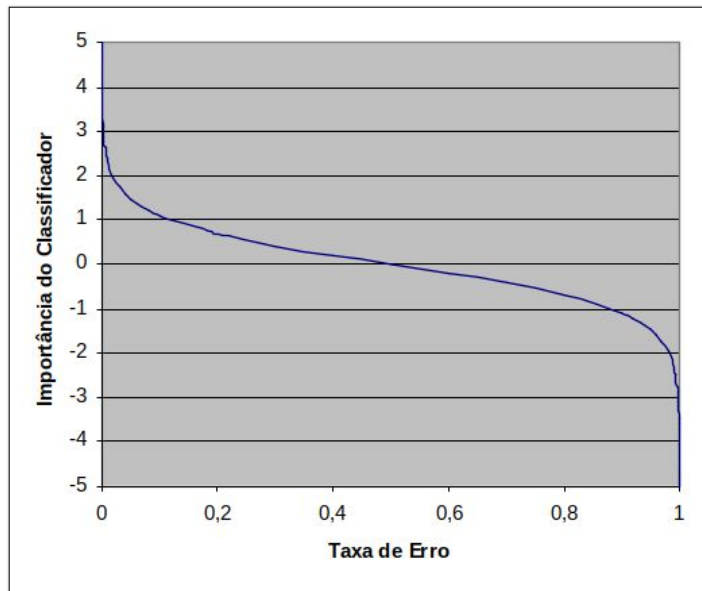
10: $\alpha_r := \frac{1}{2} \log[(1 - \epsilon_r)/\epsilon_r]$ [small if error is large and vice versa]

The **lower the error rate**, the **greater the importance** attributed to the classifier!

- $\epsilon = 0.18$
- Estimador weight (α): $\frac{1}{2} \log(1-0,18/0,18)$
 - $\alpha_1 = 0,75$



$$\alpha_r = \frac{1}{2} \log \left(\frac{1 - \epsilon_r}{\epsilon_r} \right)$$



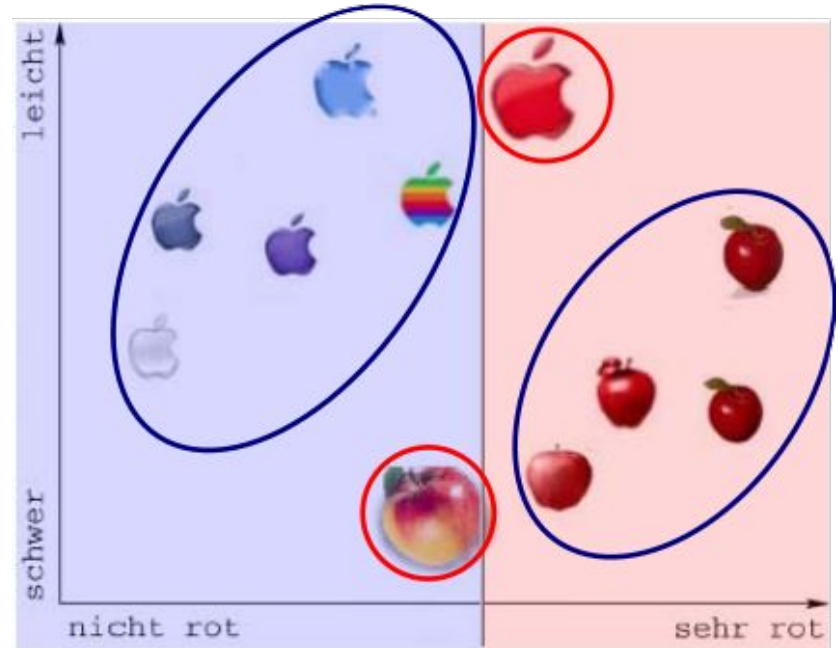
Weights update

- $w_1(i) = 0,091 \quad \forall i \in D$
- $\alpha_1 = 0,75$

$$11: \quad w_{r+1}(i) := w_r(i) \times \begin{cases} e^{-\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) = y^{[i]} \\ e^{\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) \neq y^{[i]} \end{cases}$$

The parameter α_r is used to update the weights of the training examples

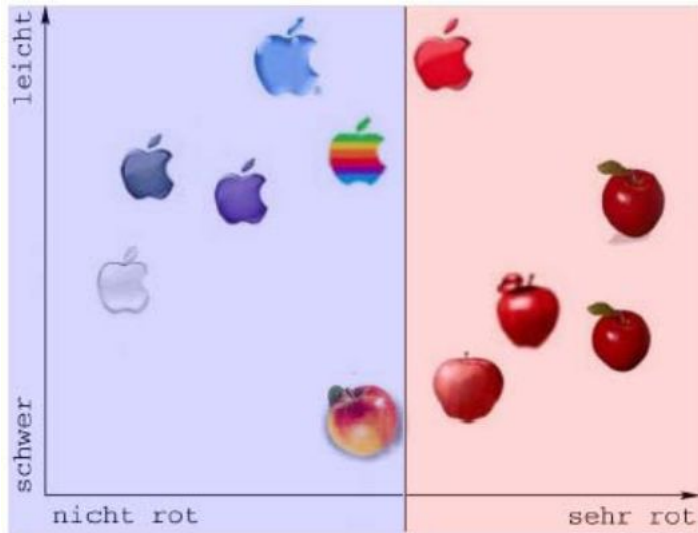
- Correctly classified records ($h_r(\mathbf{x}^i)=y^i$):
 - weight decreases;
 - $w_{r+1}(i) = 0,091 * \exp(-0,75)$
 - $w_{r+1}(i) = 0,091 * 0,047 = \mathbf{0,043}$
- Wrongly classified records ($h_r(\mathbf{x}^i) \neq y^i$):
 - weight increases;
 - $w_{r+1}(i) = 0,091 * \exp(0,75)$
 - $w_{r+1}(i) = 0,091 * 2,117 = \mathbf{0,193}$



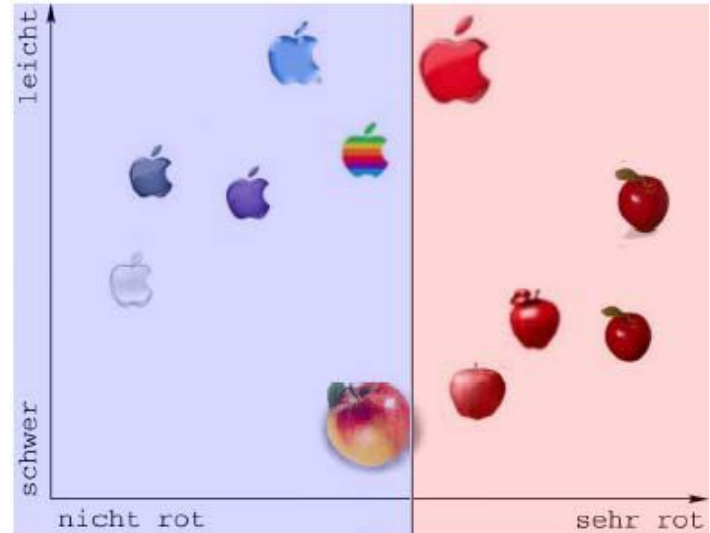
Weights update

$$11: \quad w_{r+1}(i) := w_r(i) \times \begin{cases} e^{-\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) = y^{[i]} \\ e^{\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) \neq y^{[i]} \end{cases}$$

- $w_1(i) = 0,091 \quad \forall i \in D$



- $w_2(i) = 0,043$ if correct $w_2(i) = 0,193$ if not



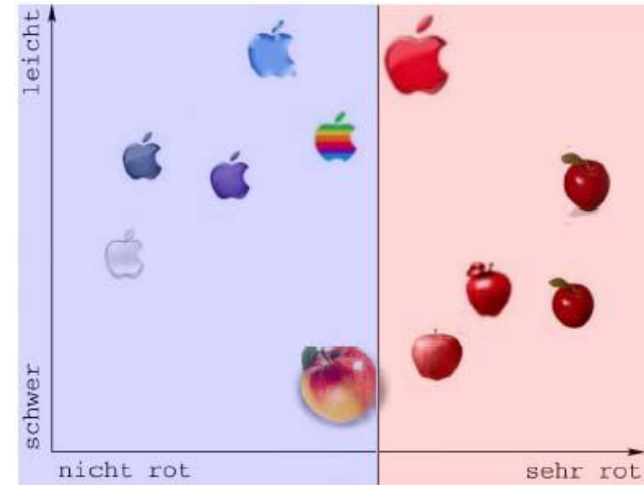
Repeat / Normalize weights

5: **for** $r=1$ to k **do**

6: For all i : $\mathbf{w}_r(i) := w_r(i) / \sum_i w_r(i)$ [normalize weights]

$$(0,043 * 9) + (0,193 * 2) = 0,773$$

- Weights of correctly classified samples
 - $w_2(i) = 0,043/0,773 = \mathbf{0,056}$
- Weights of incorrectly classified samples
 - $w_2(i) = 0,193/0,773 = \mathbf{0,248}$



Fit a weak Learner and compute error

7: $h_r := \text{FitWeakLearner}(\mathcal{D}, \mathbf{w}_r)$

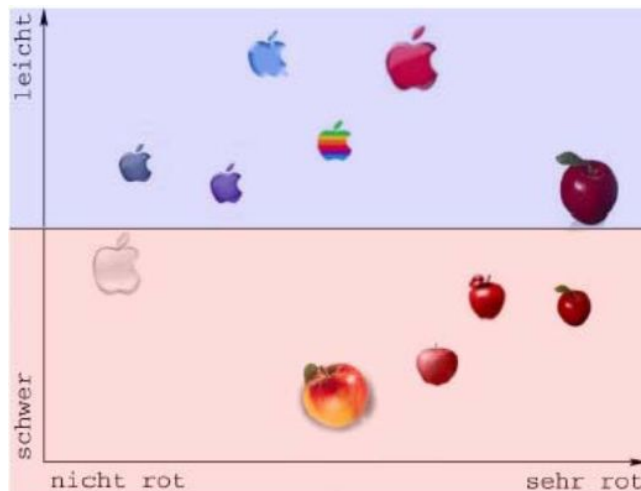
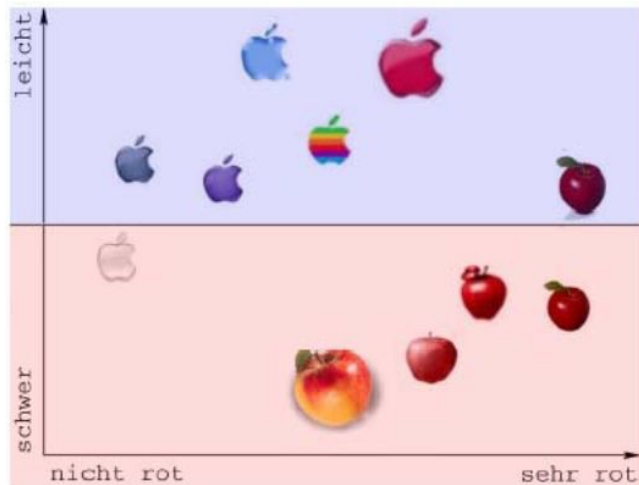
8: $\epsilon_r := \sum_i w_r(i) \mathbf{1}(h_r(i) \neq y_i)$ [compute error]

- Error (ϵ): 2/11
 - $\epsilon = 0.18$



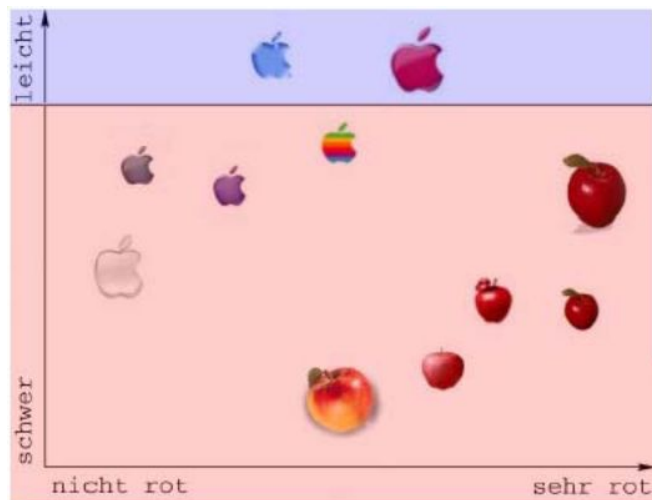
Estimador weight α and Weights update

- 9: if $\epsilon_r > 1/2$ then stop
- 10: $\alpha_r := \frac{1}{2} \log[(1 - \epsilon_r)/\epsilon_r]$ [small if error is large and vice versa]
- 11: $w_{r+1}(i) := w_r(i) \times \begin{cases} e^{-\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) = y^{[i]} \\ e^{\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) \neq y^{[i]} \end{cases}$



Repeat

```
5: for  $r=1$  to  $k$  do  
6:   For all  $i$  :  $\mathbf{w}_r(i) := w_r(i) / \sum_i w_r(i)$    [normalize weights]  
7:    $h_r := \text{FitWeakLearner}(\mathcal{D}, \mathbf{w}_r)$   
8:    $\epsilon_r := \sum_i w_r(i) \mathbf{1}(h_r(i) \neq y_i)$    [compute error]  
9:   if  $\epsilon_r > 1/2$  then stop  
10:   $\alpha_r := \frac{1}{2} \log[(1 - \epsilon_r) / \epsilon_r]$    [small if error is large and vice versa]  
11:   $w_{r+1}(i) := w_r(i) \times \begin{cases} e^{-\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) = y^{[i]} \\ e^{\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) \neq y^{[i]} \end{cases}$ 
```



Final classifier

12: Predict: $h_{ens}(\mathbf{x}) = \arg \max_j \sum_r \alpha_r \mathbf{1}[h_r(\mathbf{x}) = j]$

- $\alpha_1 = 0,75$
- $\alpha_2 = 0,75$
- $\alpha_3 = 0,29$
- $\alpha_4 = 0,75$

$$\mathbf{1}(h_r(i) = y_i) = \begin{cases} 0 & \text{if } h_r(i) \neq y_i \\ 1 & \text{if } h_r(i) = y_i \end{cases}$$

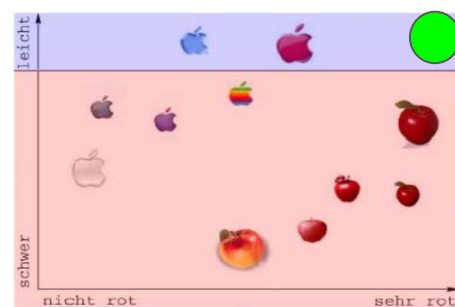
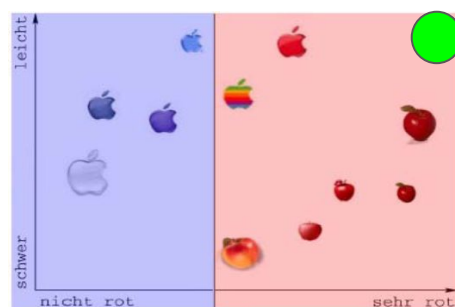
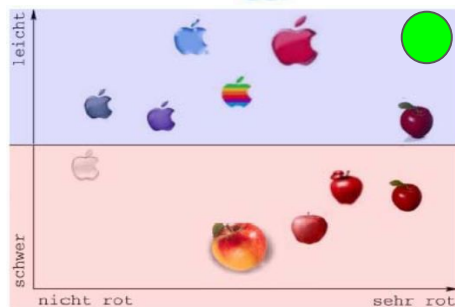
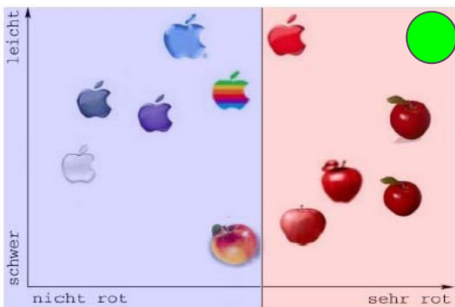


$$\arg \max_1 = 0,75 \cdot 1(0) + 0,75 \cdot 1(1) + 0,29 \cdot 1(0) + 0,75 \cdot 1(1) = 1,50$$



$$\arg \max_2 = 0,75 \cdot 1(1) + 0,75 \cdot 1(0) + 0,29 \cdot 1(1) + 0,75 \cdot 1(0) = 1,04$$

$h_{ens} = -1$




Final classifier

$$h_m(\mathbf{x}) = \text{sign}\left(\sum_{j=1}^m w_j h_j(\mathbf{x})\right) \quad \text{for } h(\mathbf{x}) \in \{-1, 1\}$$

$$h_m(\mathbf{x}) = \arg \max_i \left(\sum_{j=1}^m w_j \mathbf{1}[h_j(\mathbf{x}) = i] \right) \text{ for } h(\mathbf{x}) = i, \quad i \in \{1, \dots, n\}$$

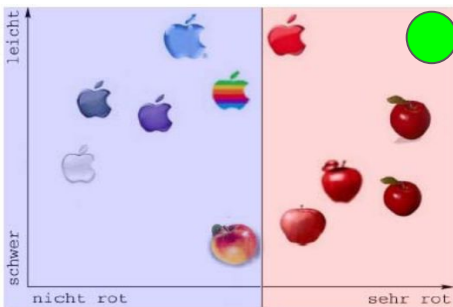
$$\text{sign} = 0,75 \cdot 1 + 0,75 \cdot -1 + 0,29 \cdot 1 + 0,75 \cdot -1 = -0,46$$

$h_{\text{ens}} = -1$ 

$$\arg \max_{-1} = 0,75 \cdot 1(0) + 0,75 \cdot 1(1) + 0,29 \cdot 1(0) + 0,75 \cdot 1(1) = 1,50$$

$$\arg \max_1 = 0,75 \cdot 1(1) + 0,75 \cdot 1(0) + 0,29 \cdot 1(1) + 0,75 \cdot 1(0) = 1,04$$

$h_{\text{ens}} = -1$



AdaBoost

0/1 loss $\mathbf{1}(h_r(i) \neq y_i) = \begin{cases} 0 & \text{if } h_r(i) = y_i \\ 1 & \text{if } h_r(i) \neq y_i \end{cases}$

Algorithm 1 AdaBoost

- 1: Initialize k : the number of AdaBoost rounds
 - 2: Initialize \mathcal{D} : the training dataset, $\mathcal{D} = \{\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, \dots, \langle \mathbf{x}^{[n]}, y^{[n]} \rangle\}$
 - 3: Initialize $w_1(i) = 1/n$, $i = 1, \dots, n$, $\mathbf{w}_1 \in \mathbb{R}^n$
 - 4:
 - 5: **for** $r=1$ to k **do**
 - 6: For all i : $\mathbf{w}_r(i) := w_r(i) / \sum_i w_r(i)$ [normalize weights]
 - 7: $h_r := \text{FitWeakLearner}(\mathcal{D}, \mathbf{w}_r)$
 - 8: $\epsilon_r := \sum_i w_r(i) \mathbf{1}(h_r(i) \neq y_i)$ [compute error]
 - 9: if $\epsilon_r > 1/2$ then stop
 - 10: $\alpha_r := \frac{1}{2} \log[(1 - \epsilon_r)/\epsilon_r]$ [small if error is large and vice versa]
 - 11: $w_{r+1}(i) := w_r(i) \times \begin{cases} e^{-\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) = y^{[i]} \\ e^{\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) \neq y^{[i]} \end{cases}$
 - 12: Predict: $h_{ens}(\mathbf{x}) = \arg \max_j \sum_r \alpha_r \mathbf{1}[h_r(\mathbf{x}) = j]$
 - 13:
- Assumes binary classification problem**

Sample weight

Estimator weight

Boosting

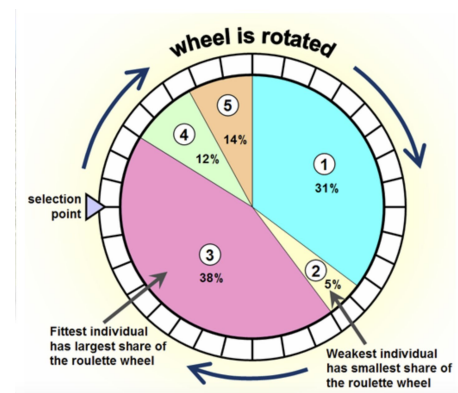
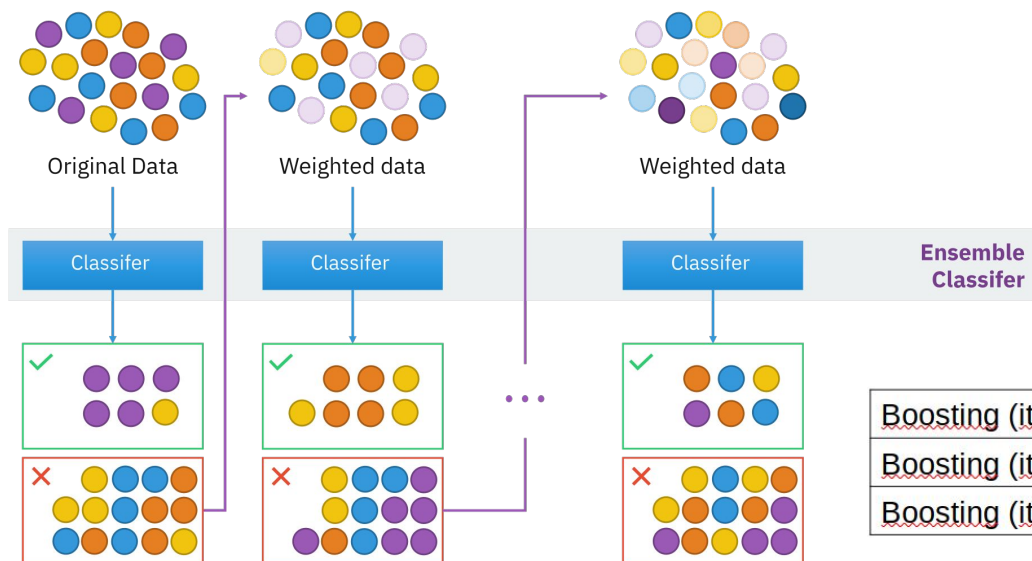
The **data modifications** at each so-called **boosting iteration** consist of **applying weights** to each of the training samples. Initially, those weights are all set to, so that the first step simply trains a weak learner on the original data. For each **successive iteration**, the **sample weights are individually modified** and the learning algorithm is **reapplied to the reweighted data**.

The weight assigned to the each training sample can be used in the following ways:

1. It can be used as a sampling distribution to do a bootstrap from the sample set of the original data
2. It can be used by the base classifier to learn a model that attaches high weight to examples of difficult classification

Boosting

As a sampling distribution to do a bootstrap from the sample set of the original data



Boosting (iteration 1)	7	3	2	8	7	9	4	10	6	3
Boosting (iteration 2)	5	4	9	4	2	5	1	7	4	2
Boosting (iteration 3)	4	4	8	10	4	5	4	6	3	4

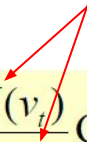
Boosting

In base classifier to learn a model that attaches high weight to examples of difficult classification

- weighted knn $d_w(x, y) = \sum_{i=1}^n w_i (x_i - y_i)^2$

replace with weights

- weighted decision tree


$$\text{Gini}_{\text{divis\~ao}} = \sum_{t=1}^k \frac{N(v_t)}{N} \text{Gini}(v_t)$$

Multiclass Adaboost (SAMME)

Stagewise Additive Modeling using a Multi-class Exponential loss function
(SAMME)

Statistics and Its Interface

Volume 2 (2009)

Number 3

Multi-class AdaBoost

Pages: 349 – 360

DOI: <https://dx.doi.org/10.4310/SII.2009.v2.n3.a8>

Authors

Trevor Hastie (Department of Statistics, Stanford University, Stanford, Calif., U.S.A.)

Saharon Rosset (Department of Statistics, Tel Aviv University, Tel Aviv, Israel)

Ji Zhu (Department of Statistics, University of Michigan, Ann Arbor, Mich., U.S.A.)

Hui Zou (School of Statistics, University of Minnesota, Minneapolis, Minn., U.S.A.)

Algorithm 1. AdaBoost [8]

1. Initialize the observation weights $w_i = 1/n$, $i = 1, 2, \dots, n$.
2. For $m = 1$ to M :

- (a) Fit a classifier $T^{(m)}(\mathbf{x})$ to the training data using weights w_i .
- (b) Compute

$$err^{(m)} = \sum_{i=1}^n w_i \mathbb{I}(c_i \neq T^{(m)}(\mathbf{x}_i)) / \sum_{i=1}^n w_i.$$

- (c) Compute

$$\alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}}.$$

- (d) Set

$$w_i \leftarrow w_i \cdot \exp\left(\alpha^{(m)} \cdot \mathbb{I}(c_i \neq T^{(m)}(\mathbf{x}_i))\right),$$

for $i = 1, 2, \dots, n$.

- (e) Re-normalize w_i .

3. Output

$$C(\mathbf{x}) = \arg \max_k \sum_{m=1}^M \alpha^{(m)} \cdot \mathbb{I}(T^{(m)}(\mathbf{x}) = k).$$

Algorithm 2. SAMME

1. Initialize the observation weights $w_i = 1/n$, $i = 1, 2, \dots, n$.
2. For $m = 1$ to M :

- (a) Fit a classifier $T^{(m)}(\mathbf{x})$ to the training data using weights w_i .
- (b) Compute

$$err^{(m)} = \sum_{i=1}^n w_i \mathbb{I}(c_i \neq T^{(m)}(\mathbf{x}_i)) / \sum_{i=1}^n w_i.$$

- (c) Compute

$$(1) \quad \alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}} + \log(K - 1).$$

- (d) Set

$$w_i \leftarrow w_i \cdot \exp\left(\alpha^{(m)} \cdot \mathbb{I}(c_i \neq T^{(m)}(\mathbf{x}_i))\right),$$

for $i = 1, \dots, n$.

- (e) Re-normalize w_i .

3. Output

$$C(\mathbf{x}) = \arg \max_k \sum_{m=1}^M \alpha^{(m)} \cdot \mathbb{I}(T^{(m)}(\mathbf{x}) = k).$$

Note that Algorithm 2 (SAMME) shares the same simple modular structure of AdaBoost with a *simple but subtle* difference in (1), specifically, the extra term $\log(K - 1)$. Obviously, when $K = 2$, SAMME reduces to AdaBoost. However, the term $\log(K - 1)$ in (1) is critical in the multi-class case ($K > 2$). One immediate consequence is that now in order

Multiclass Adaboost (SAMME)

12: Predict: $h_{ens}(\mathbf{x}) = \arg \max_j \sum_r \alpha_r \mathbf{1}[h_r(\mathbf{x}) = j]$

- $\ln(j-1) = \ln(2) = 0.69$
- $\alpha_1 = 0.30 + 0.69 = \mathbf{0.99}$
- $\alpha_2 = 0.30 + 0.69 = \mathbf{0.99}$
- $\alpha_3 = 0.30 + 0.69 = \mathbf{0.99}$
- $\alpha_4 = 0.14 + 0.69 = \mathbf{0.83}$

$$0/1 \text{ loss } \mathbf{1}(h_r(i) \neq y_i) = \begin{cases} 0 & \text{if } h_r(i) = y_i \\ 1 & \text{if } h_r(i) \neq y_i \end{cases}$$



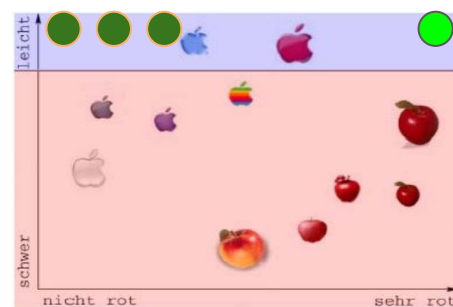
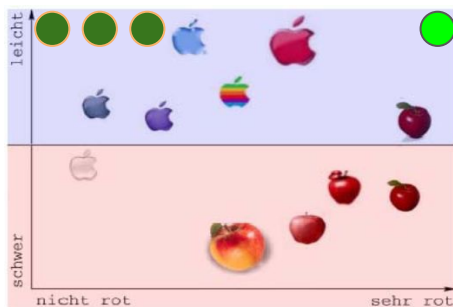
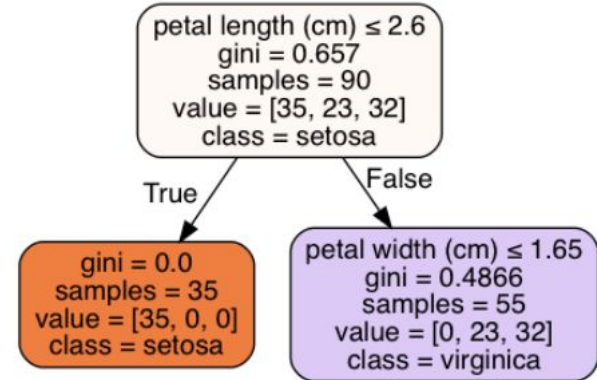
$$\arg \max_1 = 0.99 \cdot 1(0) + 0.99 \cdot 1(1) + 0.99 \cdot 1(0) + 0.83 \cdot 1(0) = 0.99$$



$$\arg \max_2 = 0.99 \cdot 1(1) + 0.99 \cdot 1(0) + 0.99 \cdot 1(1) + 0.83 \cdot 1(0) = 1.98$$

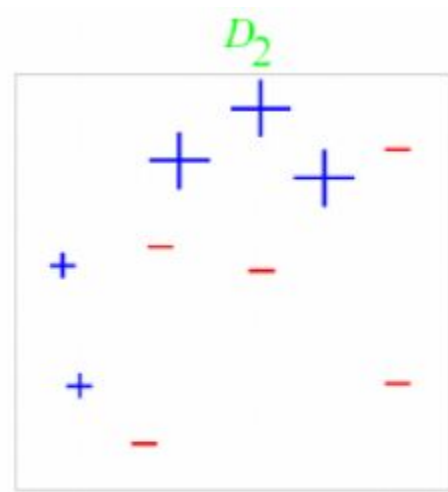
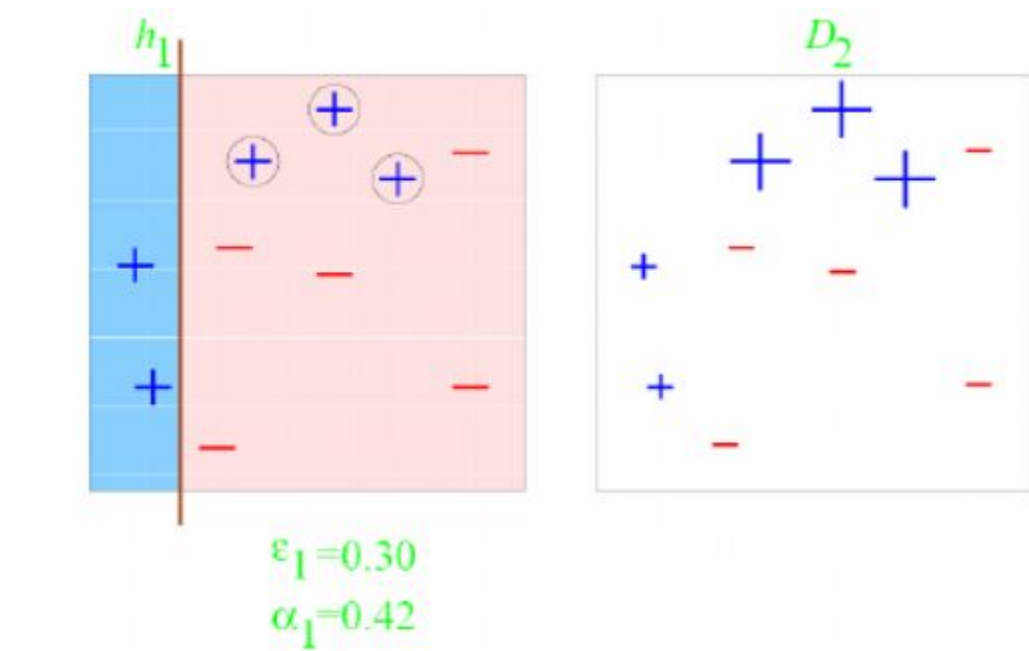
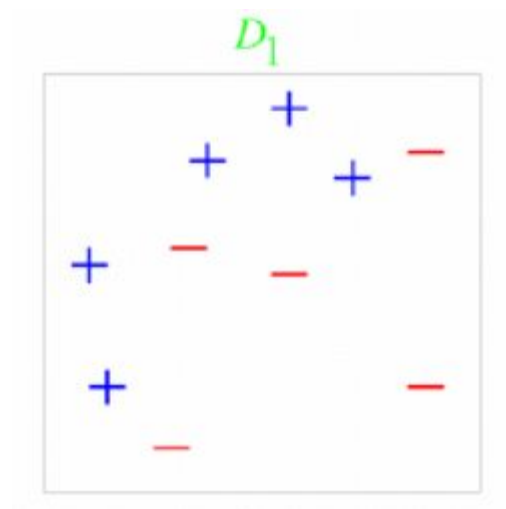


$$\arg \max_3 = 0.99 \cdot 1(0) + 0.99 \cdot 1(0) + 0.99 \cdot 1(0) + 0.83 \cdot 1(1) = 0.83$$



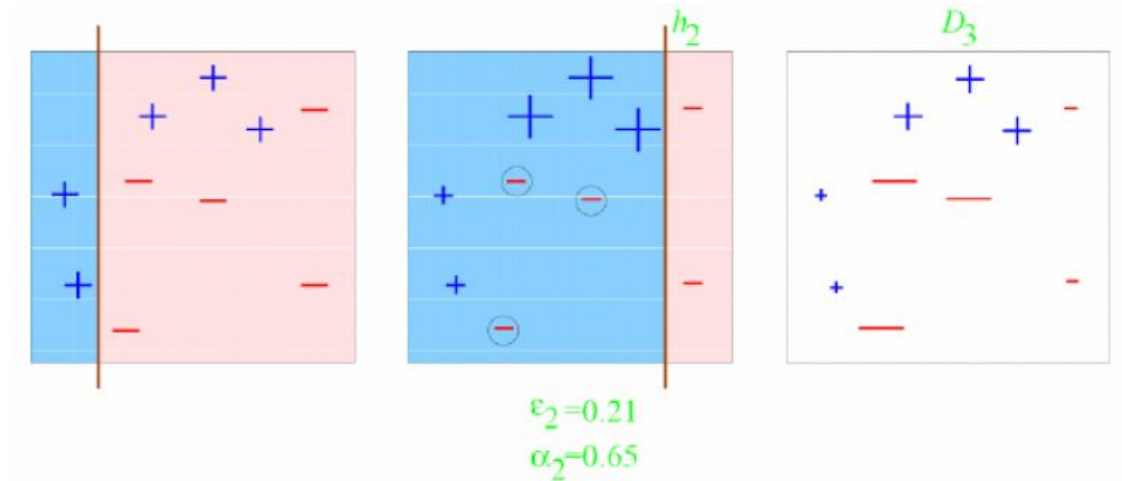
Example 2

Initialization

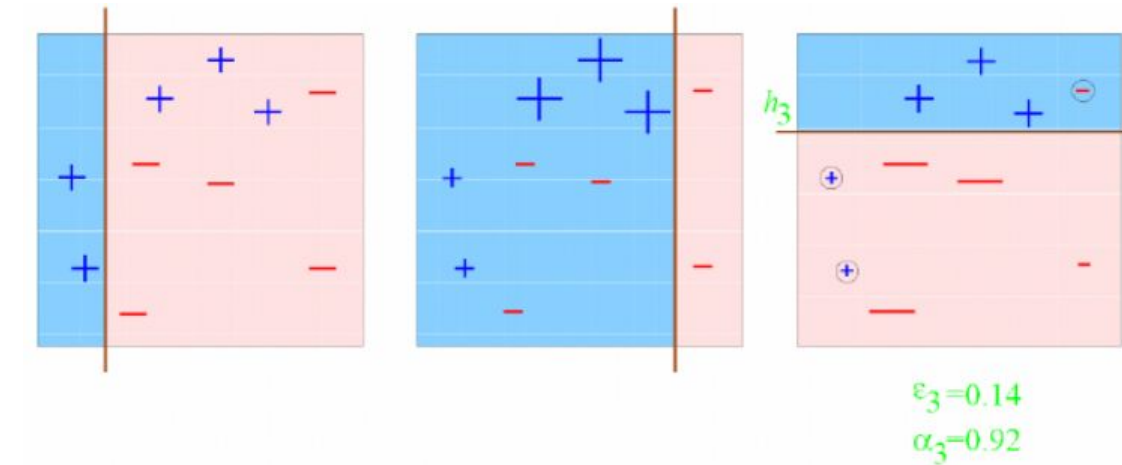


Example 2

Iteration 2

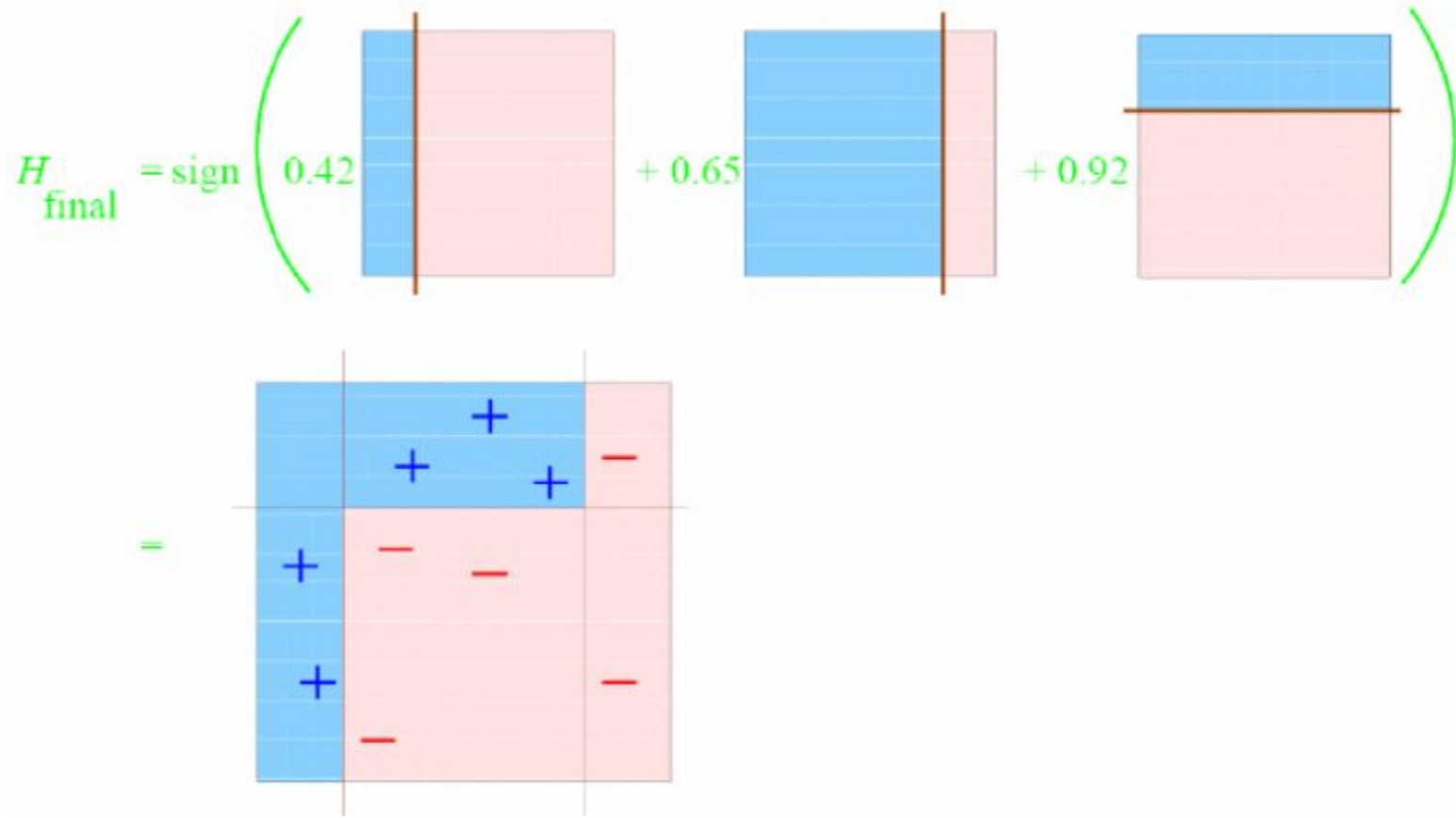


Iteration 3



Example 2

Inference



Decision Tree Summary: Pros and Cons

- (+) Simple and easy to implement algorithm;
- (+) Flexible to combine with the various basic learning algorithms available;
- (+) As the algorithm focuses on examples that are difficult to classify, the correct identification of noise is well handled;
- (-) Depending on the good choice of the basic learning algorithms (weak learners), because if they perform poorly in general, the classifier resulting from AdaBoost tends to be worse, exponentially;
- (-) When the amount of noise is high, the performance of the algorithm tends to degrade, due to the base classifiers having high error rates.

Lecturer Overview

- Intro and overview
- Voting
- Bagging
- Random Forests
- Adaboost
- **Gradient Boosting**
- Stacking