Ensemble Methods

Applied machine learning (EDAN96)
Lecture 12 — Ensemble Methods
2023–12–06
Elin A. Topp

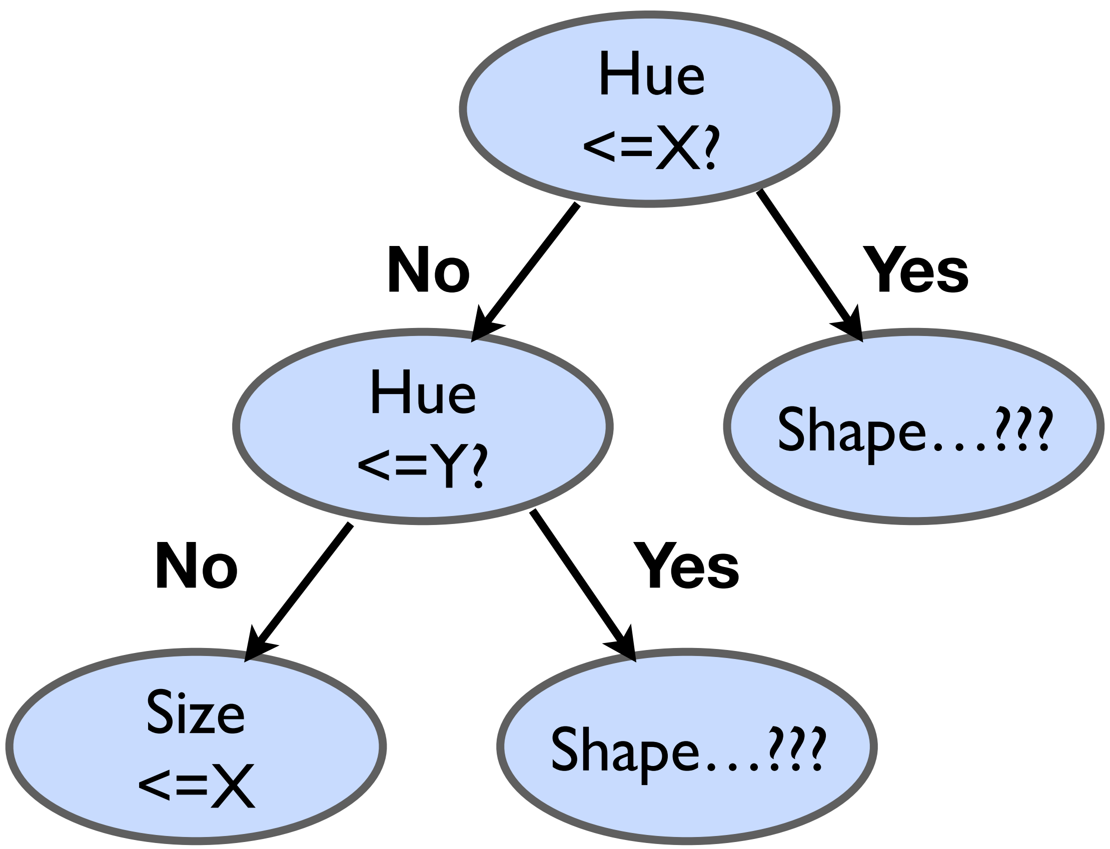
various sources

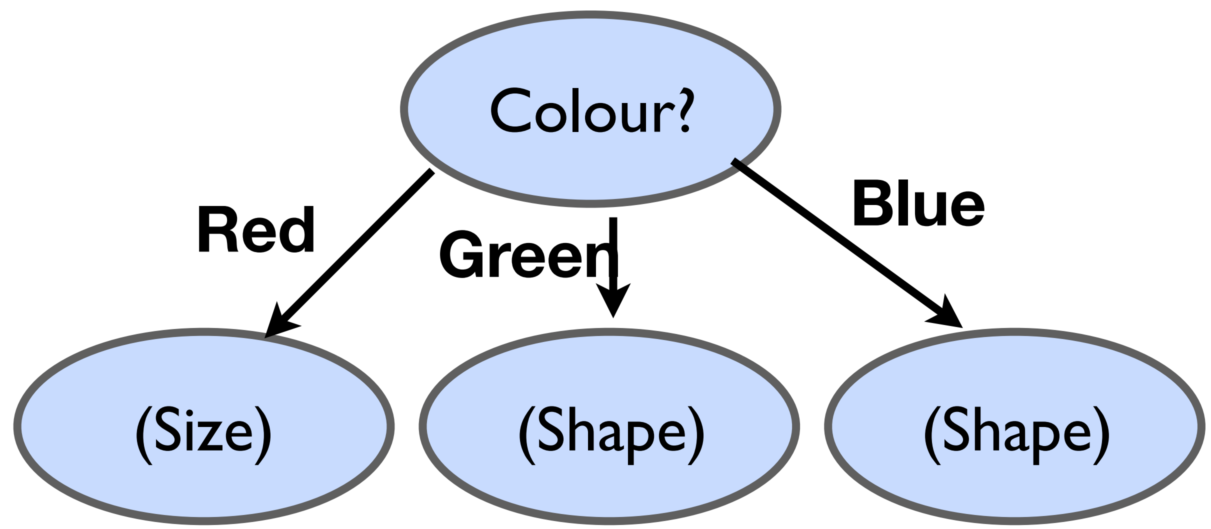
Today's agenda

- Recap Decision / Regression Trees
- Improvements for DTs / RTs (parameters, criteria)
- Ensemble methods (incl random forests)
  - Aggregating
  - Bootstrapping
  - Bagging
  - Boosting

Tree types in the assignment (CART vs ID3 or rather sklearn vs you)

- CART (sklearn): Classification and Regression Tree. Typically binary, goes for continues values, at least in sklearn-version.
- ID3 (you): Iterative Dichotomiser 3. **Can** however be multivalued and handles categorical / conceptual values, at least when you build it yourself.





Issues with Decision Trees

- Consider a new example with which you want to modify your tree…
- Consider a very unbalanced data set (like the concept learning example)
- Consider really unseen attribute values in examples …
- …?

Issues with Decision Trees

Issues with Decision Trees

- Non-existing values for certain attributes during training
  (OBS, not to confuse with attribute values that are not represented in the sample set)

Issues with Decision Trees

- Non-existing values for certain attributes during training
  (OBS, not to confuse with attribute values that are not represented in the sample set)
  - Missing values are replaced by the most common value in the considered set for the given attribute

Issues with Decision Trees

- Non-existing values for certain attributes during training
  (OBS, not to confuse with attribute values that are not represented in the sample set)
  - Missing values are replaced by the most common value in the considered set for the given attribute
- Continuous values

Issues with Decision Trees

- Non-existing values for certain attributes during training
  (OBS, not to confuse with attribute values that are not represented in the sample set)
  - Missing values are replaced by the most common value in the considered set for the given attribute
- Continuous values
  - Use decisions not based on equality, but on intervalls (see the SciKitLearn implementation)

Issues with Decision Trees

- Non-existing values for certain attributes during training
(OBS, not to confuse with attribute values that are not represented in the sample set)
  - Missing values are replaced by the most common value in the considered set for the given attribute
- Continuous values
  - Use decisions not based on equality, but on intervalls (see the SciKitLearn implementation)
- Natural bias of Information Gain towards attributes with many values, tree grows too specific

Issues with Decision Trees

- Non-existing values for certain attributes during training
  (OBS, not to confuse with attribute values that are not represented in the sample set)
  - Missing values are replaced by the most common value in the considered set for the given attribute
- Continuous values
  - Use decisions not based on equality, but on intervalls (see the SciKitLearn implementation)
- Natural bias of Information Gain towards attributes with many values, tree grows too specific
  - Limiting breadth of the tree by looking at groups of values (intervals)

Issues with Decision Trees

- Non-existing values for certain attributes during training
  (OBS, not to confuse with attribute values that are not represented in the sample set)
  - Missing values are replaced by the most common value in the considered set for the given attribute
- Continuous values
  - Use decisions not based on equality, but on intervalls (see the SciKitLearn implementation)
- Natural bias of Information Gain towards attributes with many values, tree grows too specific
  - Limiting breadth of the tree by looking at groups of values (intervals)
  - Using other methods to find the best split attribute than Information Gain

Issues with Decision Trees

- Non-existing values for certain attributes during training
  (OBS, not to confuse with attribute values that are not represented in the sample set)
  - Missing values are replaced by the most common value in the considered set for the given attribute
- Continuous values
  - Use decisions not based on equality, but on intervalls (see the SciKitLearn implementation)
- Natural bias of Information Gain towards attributes with many values, tree grows too specific
  - Limiting breadth of the tree by looking at groups of values (intervals)
  - Using other methods to find the best split attribute than Information Gain
- Overfitting the data (actually an issue with most methods in ML)

Issues with Decision Trees

- Non-existing values for certain attributes during training
  (OBS, not to confuse with attribute values that are not represented in the sample set)
  - Missing values are replaced by the most common value in the considered set for the given attribute
- Continuous values
  - Use decisions not based on equality, but on intervalls (see the SciKitLearn implementation)
- Natural bias of Information Gain towards attributes with many values, tree grows too specific
  - Limiting breadth of the tree by looking at groups of values (intervals)
  - Using other methods to find the best split attribute than Information Gain
- Overfitting the data (actually an issue with most methods in ML)
  - Pruning (going through the tree and taking out subtrees that do not contribute to performance, i.e., the new tree performs no worse than the original one on the validation data)

Issues with Decision Trees

- Non-existing values for certain attributes during training
(OBS, not to confuse with attribute values that are not represented in the sample set)
  - Missing values are replaced by the most common value in the considered set for the given attribute
- Continuous values
  - Use decisions not based on equality, but on intervalls (see the SciKitLearn implementation)
- Natural bias of Information Gain towards attributes with many values, tree grows too specific
  - Limiting breadth of the tree by looking at groups of values (intervals)
  - Using other methods to find the best split attribute than Information Gain
- Overfitting the data (actually an issue with most methods in ML)
  - Pruning (going through the tree and taking out subtrees that do not contribute to performance, i.e., the new tree performs no worse than the original one on the validation data)
  - Limits and stop criteria

Methods to mitigate overfitting (limits and stopping criteria)

Methods to mitigate overfitting (limits and stopping criteria)

- Maximum depth

Methods to mitigate overfitting (limits and stopping criteria)

- Maximum depth
- Minimum number of samples in (leaf) node

Methods to mitigate overfitting (limits and stopping criteria)

- Maximum depth
- Minimum number of samples in (leaf) node
- Minimum number of samples in node to allow for creating subtree

Methods to mitigate overfitting (limits and stopping criteria)

- Maximum depth
- Minimum number of samples in (leaf) node
- Minimum number of samples in node to allow for creating subtree
- Threshold for split criterion in regression tree (MSE < threshold)

Methods to mitigate overfitting (limits and stopping criteria)

Limiting the depth due to simplifying the model, make it less likely to fit noise in the training data

- Maximum depth
- Minimum number of samples in (leaf) node
- Minimum number of samples in node to allow for creating subtree
- Threshold for split criterion in regression tree (MSE < threshold)
- Preprocessing the data: "Binning" attributes (instead of having red, yellow, green and blue, go for "red-yellow" and "green-blue")

the minimum amount of improvement in the mean squared error (or another regression loss function) required to make a split at a node in a decision tree.

Why then Decision / Regression Trees?

Why then Decision / Regression Trees?

- Intuitive

Why then Decision / Regression Trees?

- Intuitive
- Explainable *by nature*

Why then Decision / Regression Trees?

- Intuitive
- Explainable *by nature*
- Tedious to compute, but very efficient to use

Single opinion vs collective decision

Single opinion vs collective decision

- One single classifier / regressor (we assume a decision tree for now, but it can be anything) can make a mistake, but hopefully that happens with a likelihood below random

Single opinion vs collective decision

- One single classifier / regressor (we assume a decision tree for now, but it can be anything) can make a mistake, but hopefully that happens with a likelihood below random
- Several single classifiers—if trained with some variation—will most likely make different mistakes

Single opinion vs collective decision

- One single classifier / regressor (we assume a decision tree for now, but it can be anything) can make a mistake, but hopefully that happens with a likelihood below random
- Several single classifiers—if trained with some variation—will most likely make different mistakes
- If there is some sense in the classifiers (mistake less likely than random!), there will be a majority of correct / sensible answers in the crowd.

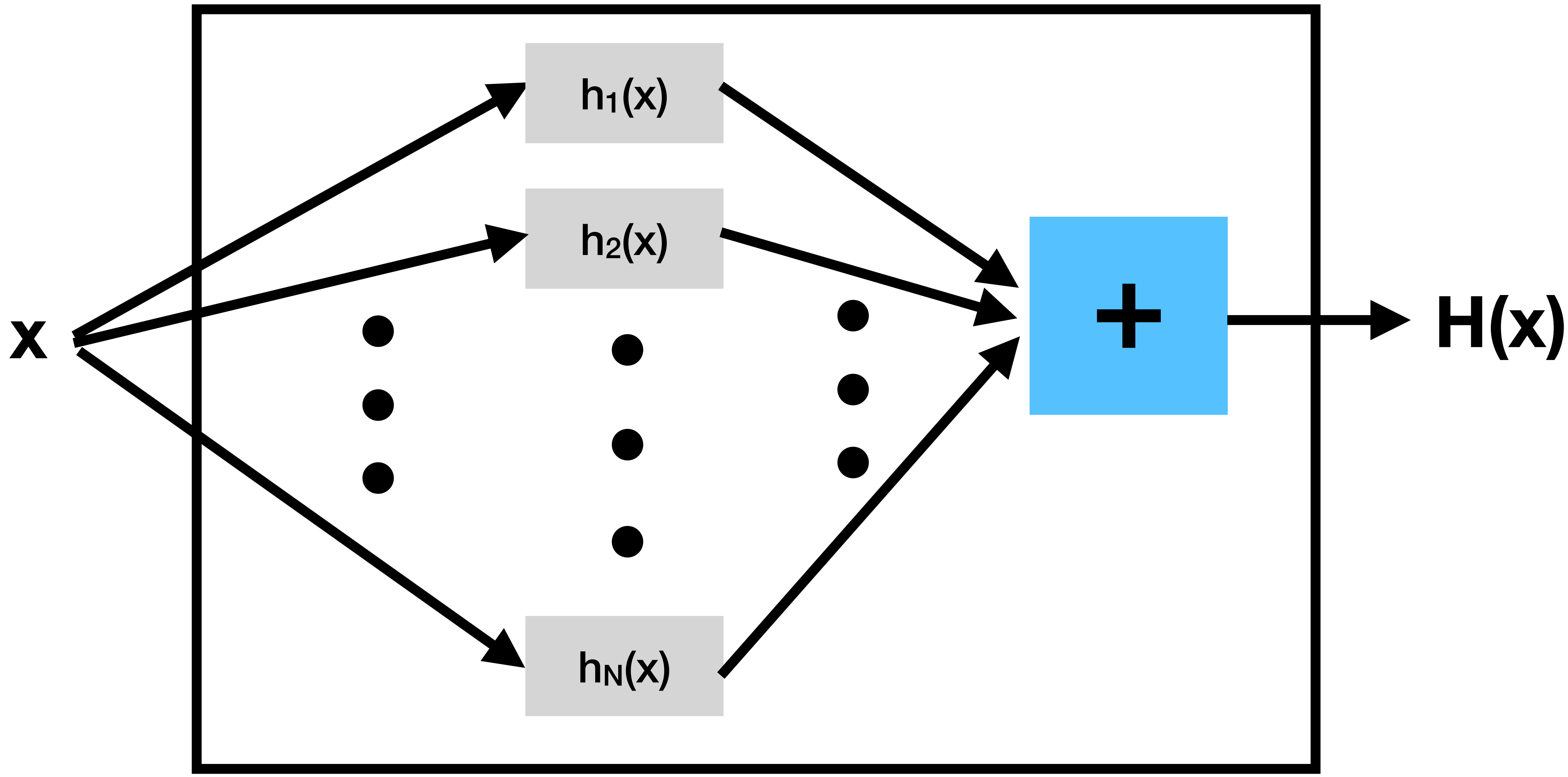
Single opinion vs collective decision

- One single classifier / regressor (we assume a decision tree for now, but it can be anything) can make a mistake, but hopefully that happens with a likelihood below random
- Several single classifiers—if trained with some variation—will most likely make different mistakes
- If there is some sense in the classifiers (mistake less likely than random!), there will be a majority of correct / sensible answers in the crowd.
- Train *N* classifiers (e.g. not one tree, but a forest) and have them somehow come to a collective conclusion

Single opinion vs collective decision

- One single classifier / regressor (we assume a decision tree for now, but it can be anything) can make a mistake, but hopefully that happens with a likelihood below random
- Several single classifiers—if trained with some variation—will most likely make different mistakes
- If there is some sense in the classifiers (mistake less likely than random!), there will be a majority of correct / sensible answers in the crowd.
- Train N classifiers (e.g. not one tree, but a forest) and have them somehow come to a collective conclusion
- Several ways of producing the final output, given a set of hypotheses $\{h_i\}$ for the answer from each of the N classifiers, $i = 1,\ldots,N$

Method 1: Averaging / Voting

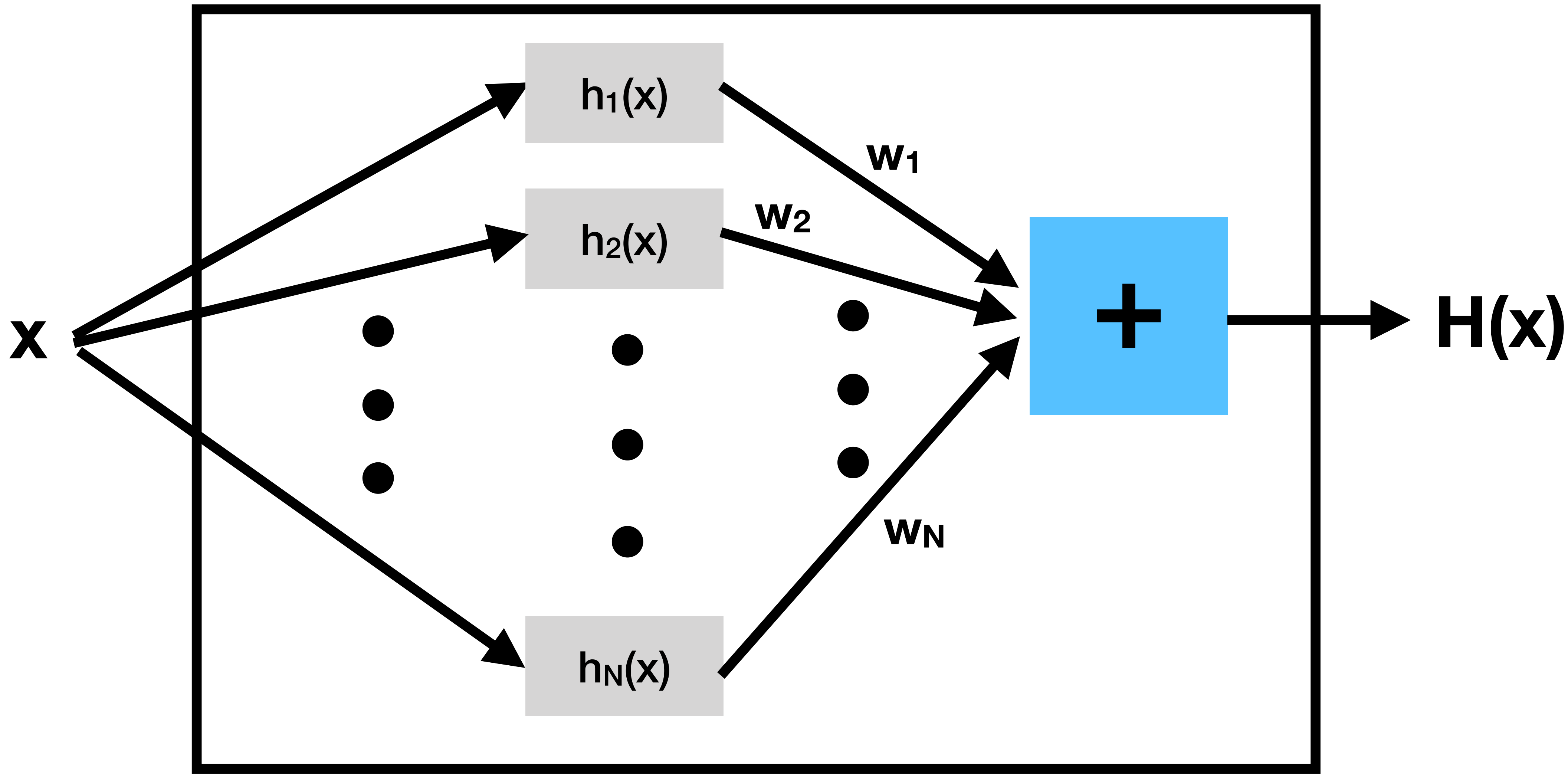


Ensamble method in ML refers to combining the predictions of multiplee models to create a stronger, more robust model. The idea is to leverage the diversity among individual models to improve overall predictive performance.

Voting involves aggregating the predicted class labels from multiple classifiers. In a majority voting, each classifier in the ensemble predicts a class label, the class label that receives the majority of votes is chosen as the final prediction.

Slide based on material from Eric Eaton (see lecture slides EDAN95 2018, lecture 4)

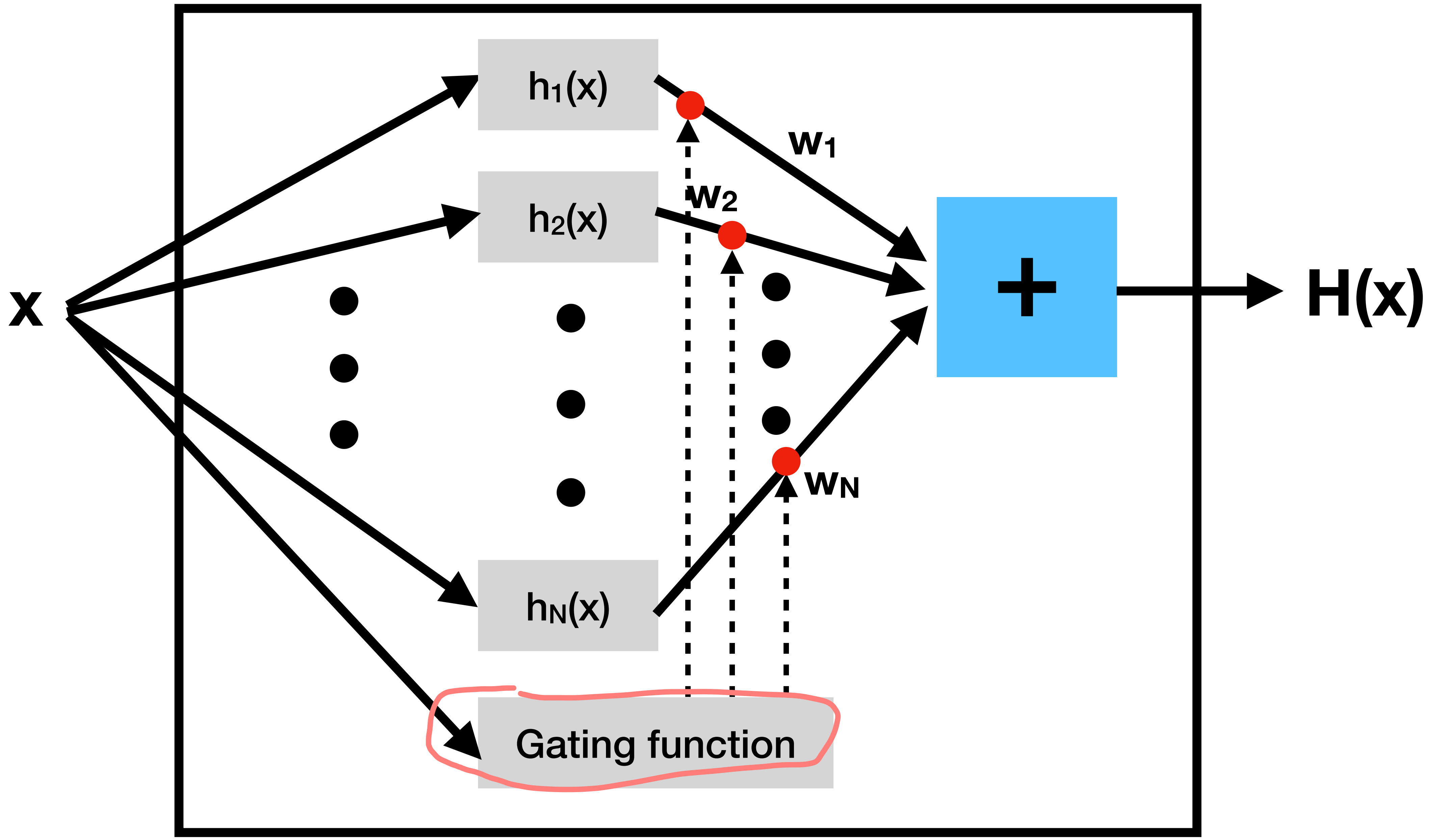
Method 2: Weighted Averaging / Voting



In weighted voting, each classifiers vote is weighted based on its confidence or performance. The final prediction is a weighted sum or average of the predicted class labels.

Slide based on material from Eric Eaton (see lecture slides EDAN95 2018, lecture 4)

Method 3: Gating

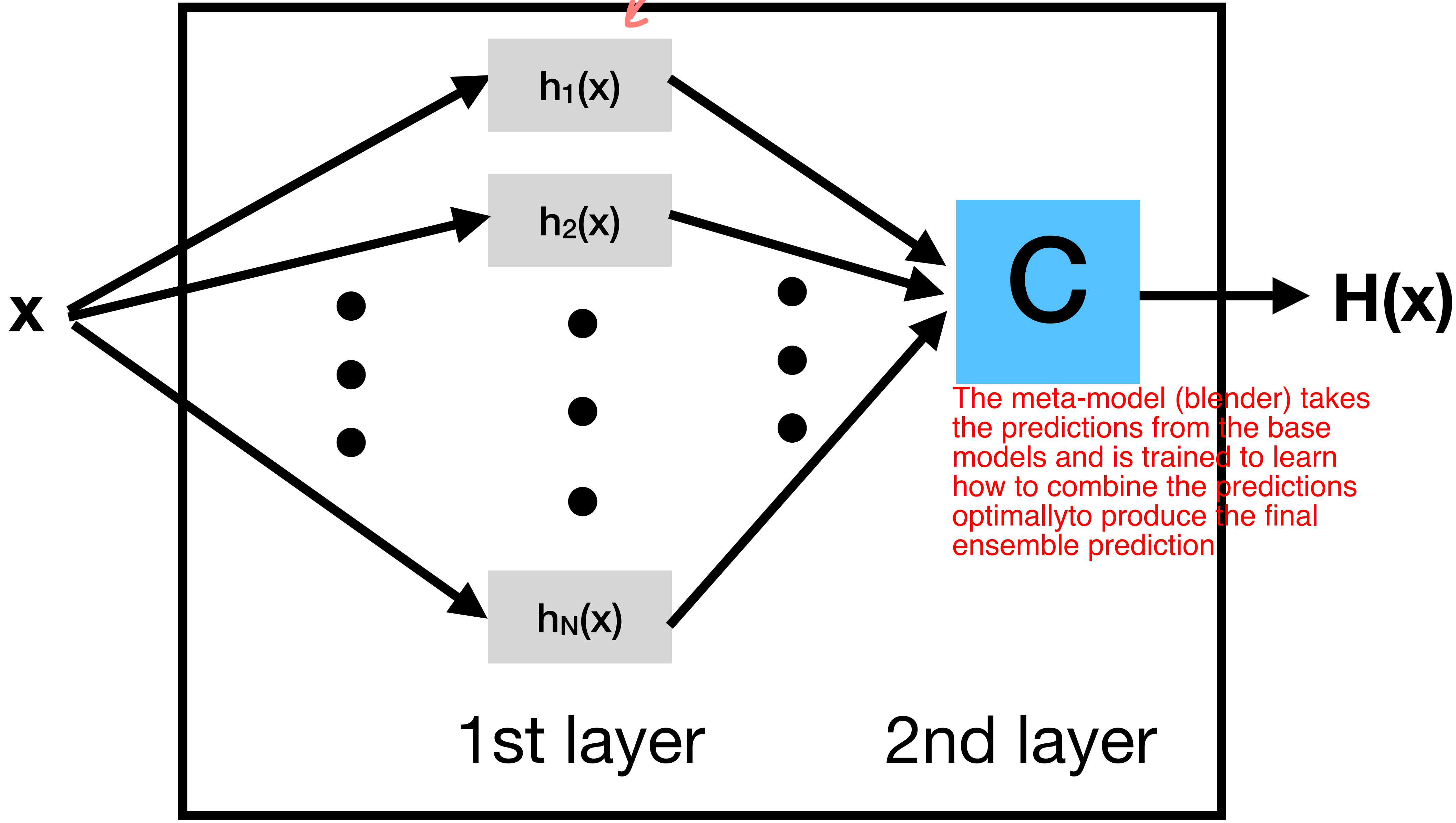


Gating is commonly used in neural network architectures and refers to the introduction of a meta-model that combines or votes of the base models. The meta-model takes the predictions of the base models as inputs and prices a final combined predictions. The gating strategy can vary, involving assigningment of weights to the predictions of each base model to learn how to combine predictions dynamically

Slide based on material from Eric Eaton (see lecture slides EDAN95 2018, lecture 4)

Method 4: Stacking

Each base models are trained independently on the training data on the first layer



The meta-model (blender) takes the predictions from the base models and is trained to learn how to combine the predictions optimallyto produce the final ensemble prediction

Stacking layers refer to the hierarchical structure of models in a stacked ensemble, where multiple layers of models are trained to make predictions. In a stacked ensemble, base models, intermediate models, and possible a final meta-model are organised in layers, each with a specific role in the ensemble.

Creating several different classifiers

- There might be different causes for the mistakes your original single classifier makes.
  - Difficult samples (sometimes, reality is nasty): No recipe, you will have to live with this, and be aware that there are these uncertainties
  - Overfitting (even after modelling quite carefully): Vary the training sets
  - Noise in the data / some features: Vary sets of input features
  - More or less suitable model: Vary the type of classifier (voting, stacking, gating, … all can be done with different classifiers / regressors plugged into them), or vary the parameters for the same type of model

Boosting (e.g. AdaBoost)

Boosting (e.g. AdaBoost)

- Improves (weak) classifier(s) over time (steps $t = 1,\dots,T$) by emphasizing mispredicted instances (in the training set!)

Boosting (e.g. AdaBoost)

- Improves (weak) classifier(s) over time (steps $t = 1,\ldots,T$) by emphasizing mispredicted instances (in the training set!)
- Assume classes $y_i \in \{-1,1\}$ and classification hypotheses $h_t(x_i) \in \{-1,1\}$

Boosting (e.g. AdaBoost)

- Improves (weak) classifier(s) over time (steps $t = 1, \ldots, T$) by emphasizing mispredicted instances (in the training set!)
- Assume classes $y_i \in \{-1, 1\}$ and classification hypotheses $h_t(x_i) \in \{-1, 1\}$
- Initially, all instances are of the same importance, e.g.

$$w_{t,i} = \frac{1.0}{|samples|} \quad for \quad t = 0$$

Boosting (e.g. AdaBoost)

- Improves (weak) classifier(s) over time (steps $t = 1,\ldots,T$) by emphasizing mispredicted instances (in the training set!)
- Assume classes $y_i \in \{-1,1\}$ and classification hypotheses $h_t(x_i) \in \{-1,1\}$
- Initially, all instances are of the same importance, e.g.

$$w_{t,i} = \frac{1.0}{|samples|} \quad for \quad t = 0$$

- In each iteration, compute the weighted training error (opposite of accuracy) ϵ (each instance x_i in the training set contributes with its weight to the error, misclassified instances, where $h_t(x_i) \neq y_i$, do so obviously more than correctly classified ones)

Boosting (e.g. AdaBoost)

- Improves (weak) classifier(s) over time (steps t = $1,\ldots,T$) by emphasizing mispredicted instances (in the training set!)
- Assume classes $y_i \in \{-1,1\}$ and classification hypotheses $h_t(x_i) \in \{-1,1\}$
- Initially, all instances are of the same importance, e.g.

$$w_{t,i} = \frac{1.0}{|samples|} \quad for \quad t = 0$$

- In each iteration, compute the weighted training error (opposite of accuracy) ϵ (each instance x_i in the training set contributes with its weight to the error, misclassified instances, where $h_t(x_i) \neq y_i$, do so obviously more than correctly classified ones)
- Update the weights so that a wrongly classified sample gets a higher weight

$$w_{t+1,i} = w_{t,i} \cdot e^{-\beta_t y_i h_t(x_i)} \quad with \quad \beta_t = \frac{1}{2} ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

Boosting (e.g. AdaBoost)

- Improves (weak) classifier(s) over time (steps $t = 1,\ldots,T$) by emphasizing mispredicted instances (in the training set!)
- Assume classes $y_i \in \{-1,1\}$ and classification hypotheses $h_t(x_i) \in \{-1,1\}$
- Initially, all instances are of the same importance, e.g.

$$w_{t,i} = \frac{1.0}{|samples|} \quad for \quad t = 0$$

- In each iteration, compute the weighted training error (opposite of accuracy) ϵ (each instance x_i in the training set contributes with its weight to the error, misclassified instances, where $h_t(x_i) \neq y_i$, do so obviously more than correctly classified ones)
- Update the weights so that a wrongly classified sample gets a higher weight

$$w_{t+1,i} = w_{t,i} \cdot e^{-\beta_t y_i h_t(x_i)} \quad with \quad \beta_t = \frac{1}{2} ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$$

- normalise the weights, and iterate (i.e. produce stepwise an ensemble classifier as $H_t(\mathbf{x}) = H_{t-1}(\mathbf{x}) + \beta_t h_t(\mathbf{x})$)

Boosting (e.g. AdaBoost)

- Improves (weak) classifier(s) over time (steps $t = 1,\ldots,T$) by emphasizing mispredicted instances (in the training set!)
- Assume classes $y_i \in \{-1,1\}$ and classification hypotheses $h_t(x_i) \in \{-1,1\}$
- Initially, all instances are of the same importance, e.g.

$$w_{t,i} = \frac{1.0}{|samples|} \quad for \quad t = 0$$

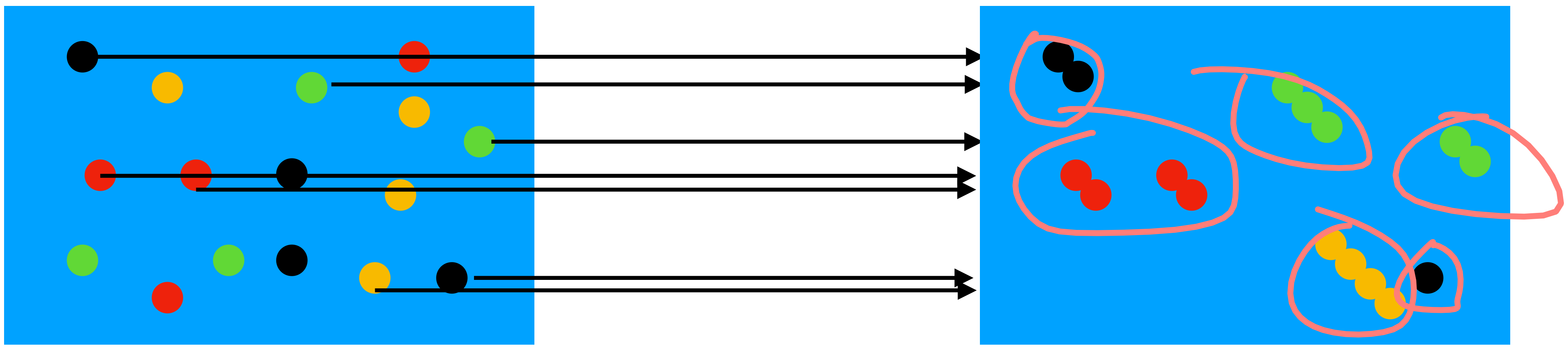
- In each iteration, compute the weighted training error (opposite of accuracy) ϵ (each instance x_i in the training set contributes with its weight to the error, misclassified instances, where $h_t(x_i) \neq y_i$, do so obviously more than correctly classified ones)
- Update the weights so that a wrongly classified sample gets a higher weight

$$w_{t+1,i} = w_{t,i} \cdot e^{-\beta_t y_i h_t(x_i)} \quad with \quad \beta_t = \frac{1}{2} ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

- normalise the weights, and iterate (i.e. produce stepwise an ensemble classifier as $H_t(\mathbf{x}) = H_{t-1}(\mathbf{x}) + \beta_t h_t(\mathbf{x})$)
- when done, produce hypothesis as $\boxed{H_T(\mathbf{x}) = sign\left(\sum_{t=1}^{T} \beta_t h_t(\mathbf{x})\right)}$

Manipulating training data: Bootstrap replication

Bootstrap replication involves creating multiple subsets of the dataset by randomly sampling with replacement. The term "bootstrap" comes from the phrase "pulling oneself up by one's bootstraps," suggesting a process of self-generation.



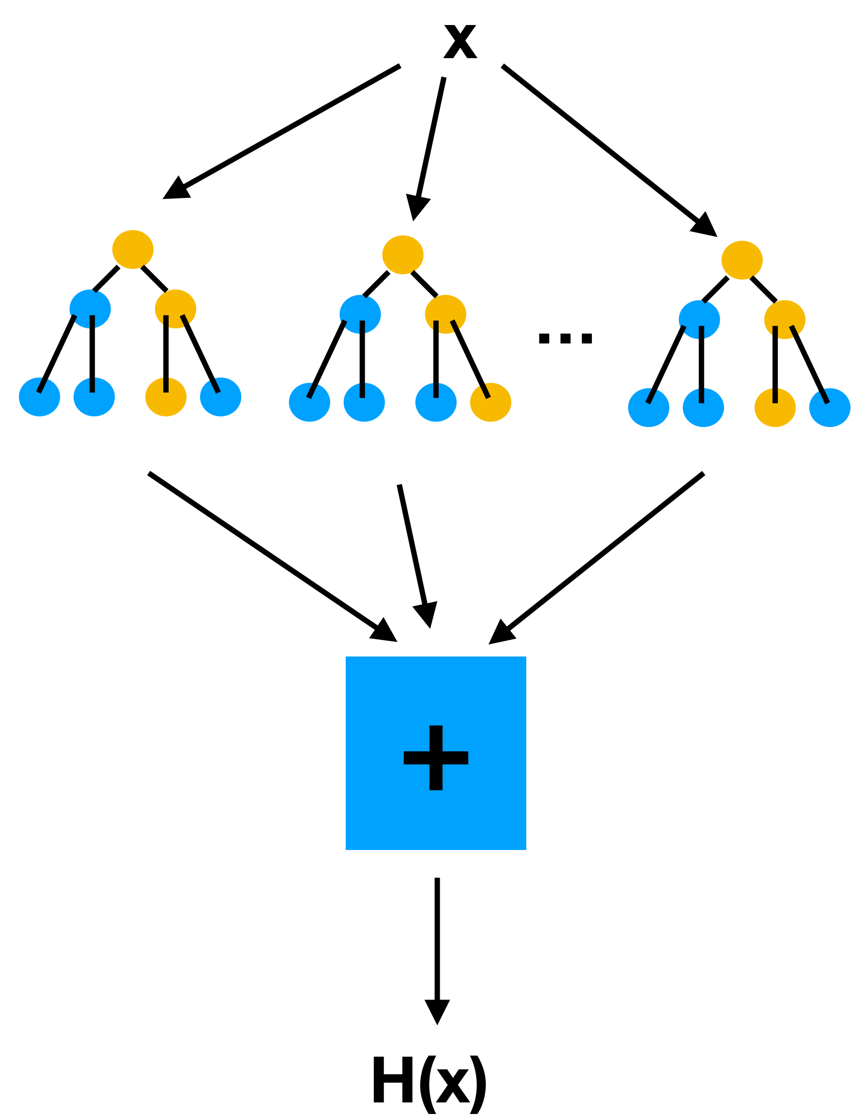
Exclude some (e.g. 30%) of data from the bootstrapping

Bagging (Bootstrap AGGregatING)

- Do a Bootstrap replicating round, create thus N training sets
- Train N classifiers, one on each set
- Estimate performance on the out-of-bootstrap data (the ~30%)
- Combine output according to previously suggested methods

Random Forest

- In principle, a bagging approach:
  - Do a **Bootstrap** replicating round, create thus *N* training sets
  - Train *N* DTs or RTs, one on each set, **but**
    - Use **only a randomly picked set of attributes** for each DT/RT
    - Do **not** prune the trees and **estimate performance on the out-of-bootstrap data** (the ~30%)
  - **Combine output** according to previously suggested methods (e.g. averaging/voting)



Today's summary

- Discussed further Decision Trees / Regression Trees and possibilities to improve them
- Presented ensemble methods, incl boosting as improvement approach, as well as bagging in general and Random Forests as a specific bagging technique for DTs/RTs.

- Reading:
  - Lecture slides lecture 4, 2018
  - Mitchell, chapter 3, Decision Trees
  - Lindholm et al, chapter 2

Outlook on homework assignment 3

- Take role of a researcher: Compare two approaches for regression trees for the **California housing data** and write a short paper
- Focus for the entire report ("paper") is Part 2 of programming assignment 5, i.e. the "general workings" of an RT should be discussed for basic implementations, where differences can be described between the **two types** of trees
- Make use of the hints in the instructions!

- Reading:
  - SciKitLearn documentation
  - ID3 description, e.g. Wikipedia, lecture material, pseudo code on Canvas page.