Ensuring More Accurate, Generalisable, and Interpretable Machine Learning Models for Bioinformatics

Wandrille Duchemin, Van Du, Markus Müller

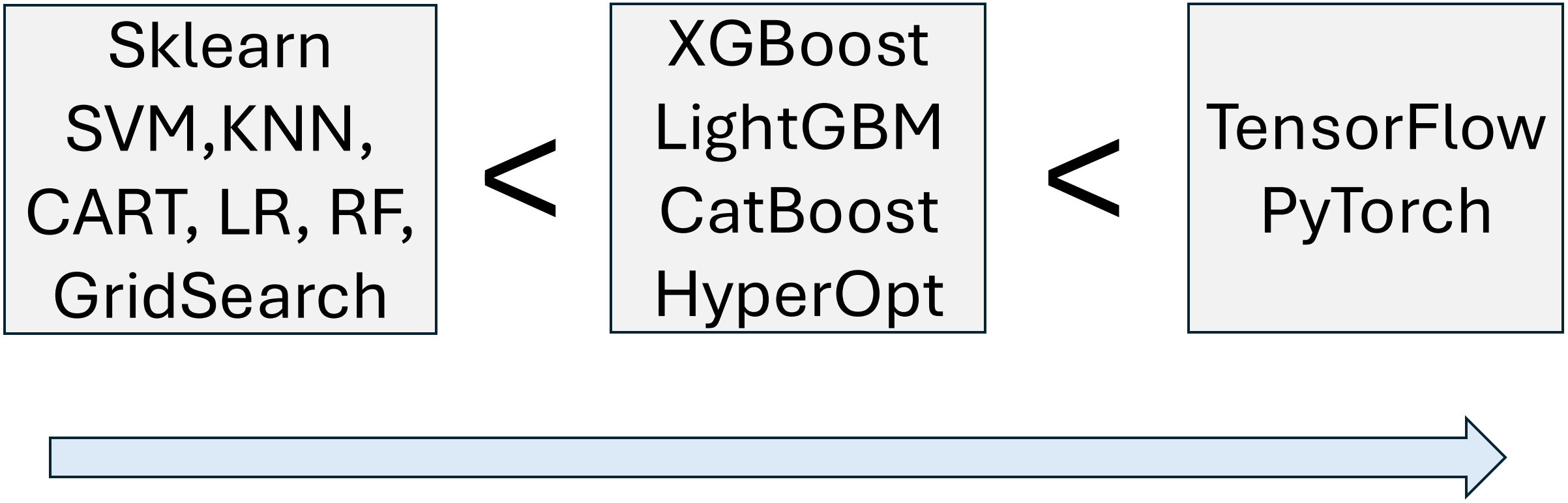
Swiss Institute of Bioinformatics

Swiss Institute of Bioinformatics

Outline

- Intermediate machine learning (ML)
- Trees
- Bagging and boosting
- Classification and regression with XGBoost
- Hyperparameter optimization with Hyperopt
- Interpretable machine learning with LIME and SHAP

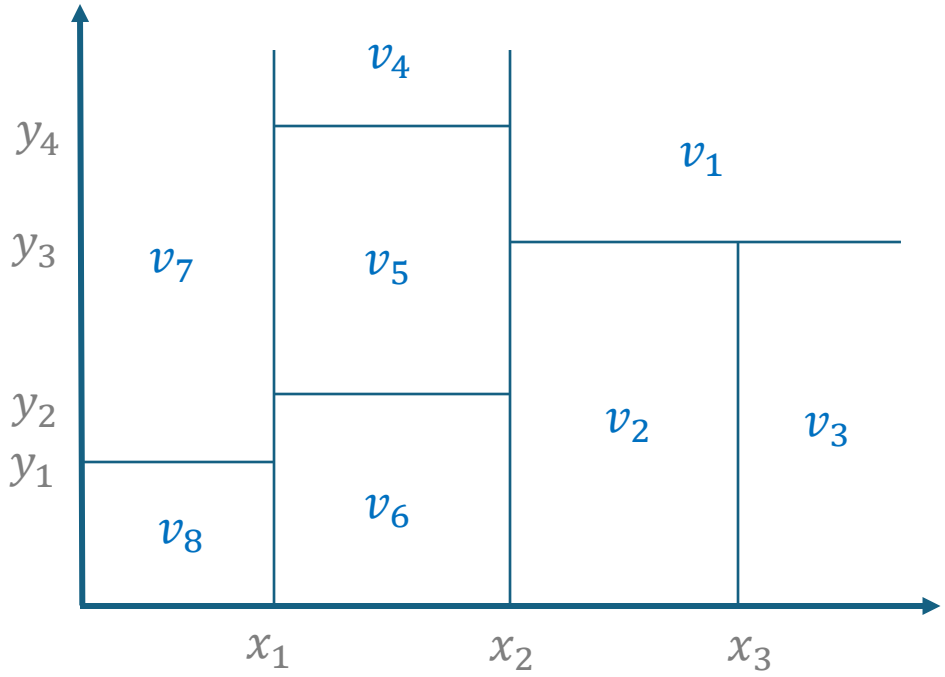
Intermediate ML

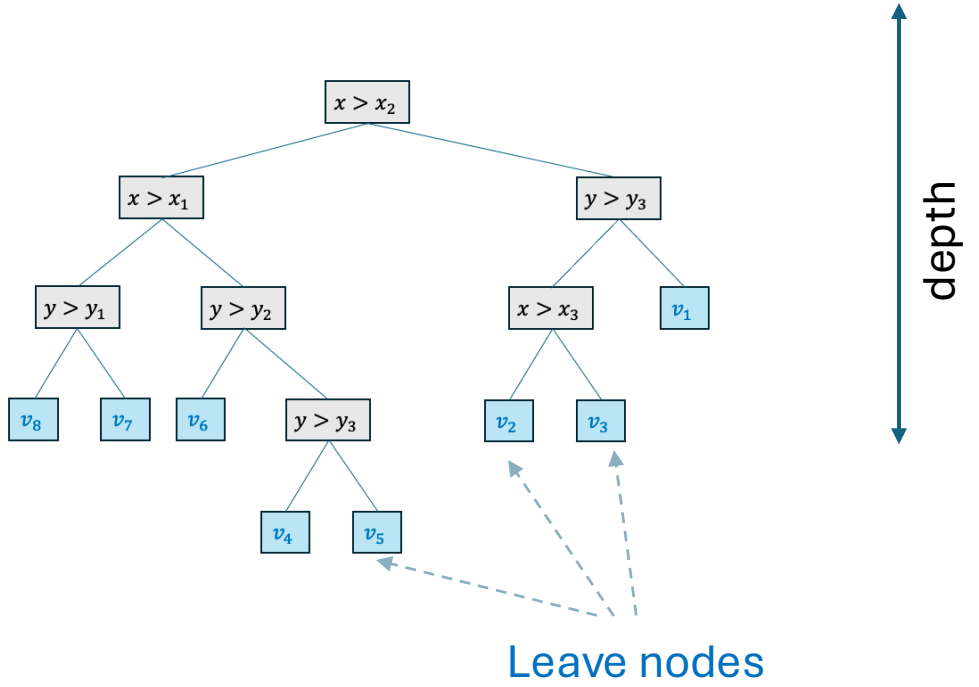


Markus Müller
SIB Swiss Institute of Bioinformatics

Trees

- $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}, 1 \leq i \leq n, \boldsymbol{x}_i \in \mathbb{R}^m, y_i \in \mathbb{R}$
- $\hat{y}_i = f(\boldsymbol{x}_i)$,
  - $f \in \mathcal{T} = \{f(\boldsymbol{x}) = \boldsymbol{v}_{q(\boldsymbol{x})}\}, (q: \mathbb{R}^m \to \{1..T\}, \boldsymbol{v} \in \mathbb{R}^T), T$ number of leafes



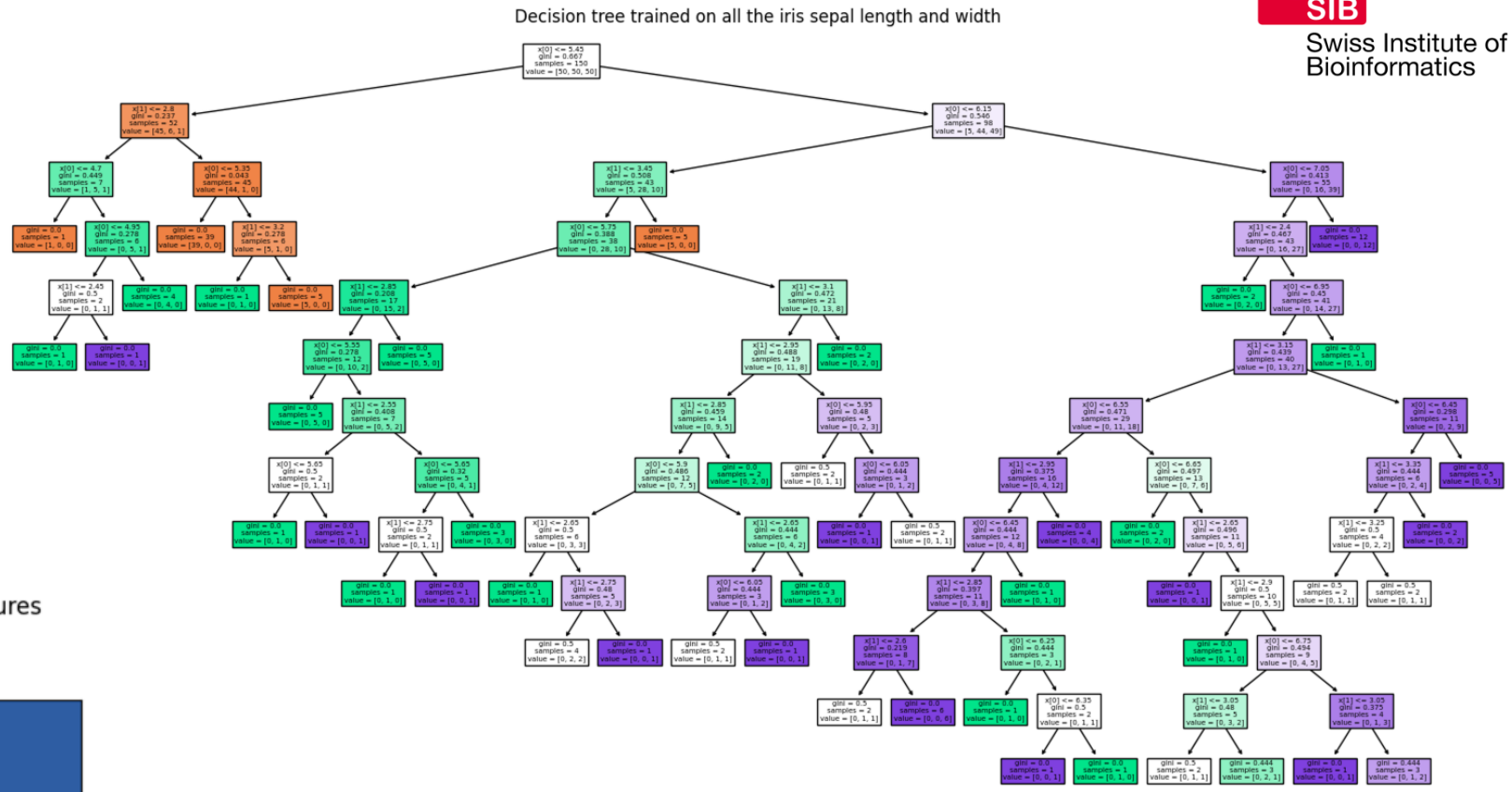


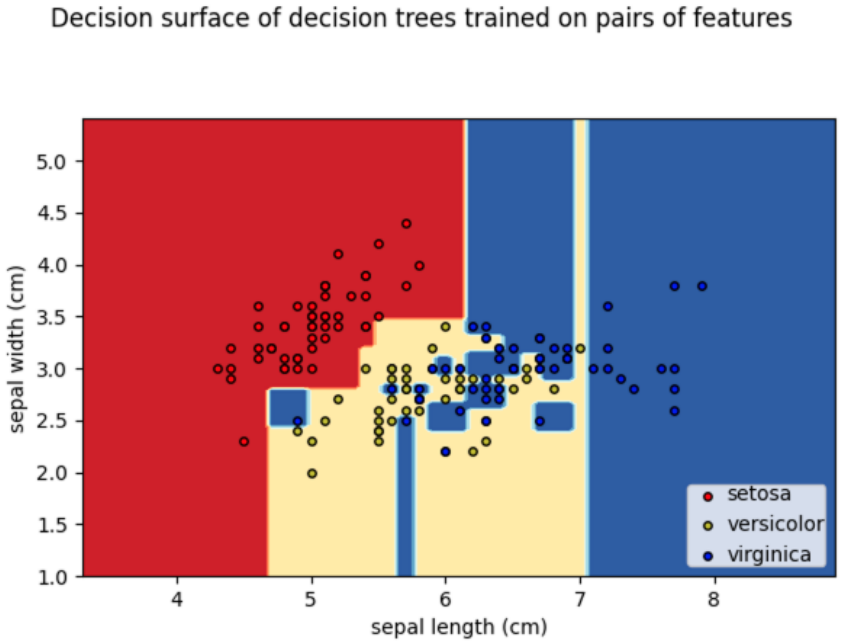
Trees

- Regression and classification trees
- Trees are universal approximators (no bias) if the size is not restricted
- Tree's are able to deal with categorical data. For numerical data they are not sensitive to the distribution of features values
- Tree hyperparameters add constraints to its growth and weights:
  - max_depth: maximal number of internal nodes before leaf
  - max_leaf_nodes: maximal number of leaf nodes in the tree
  - min_samples_leaf: minimal number of data points in the training set within leaf
  - L1 or L2 regularization of weights

Markus Müller
Swiss Institute of Bioinformatics

Trees

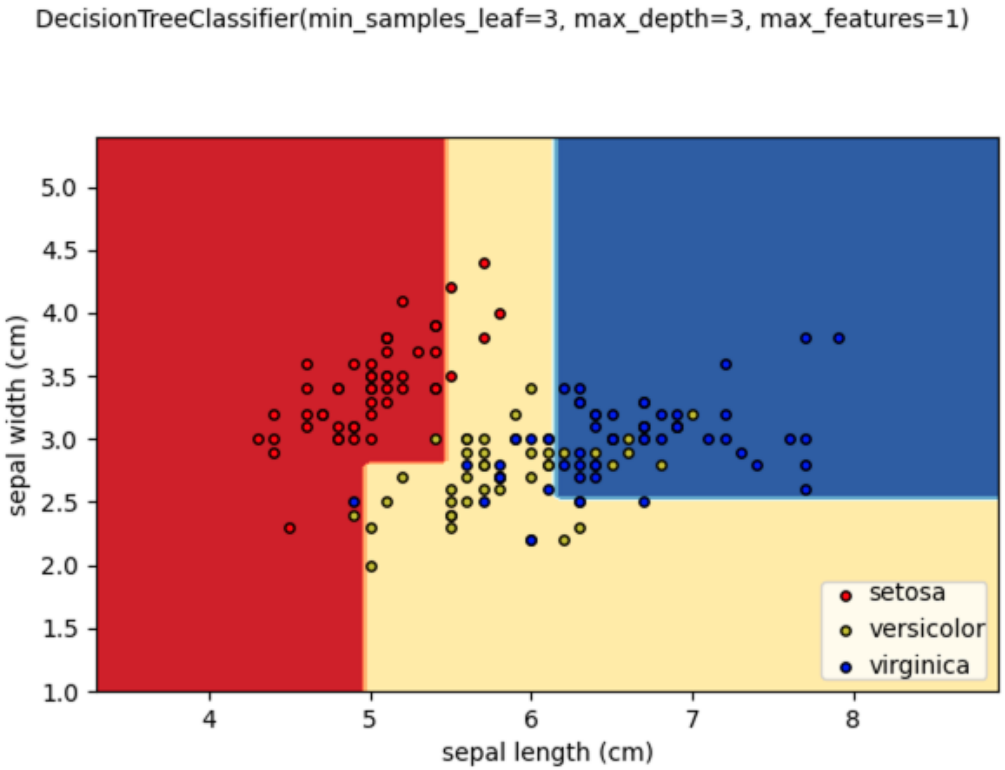


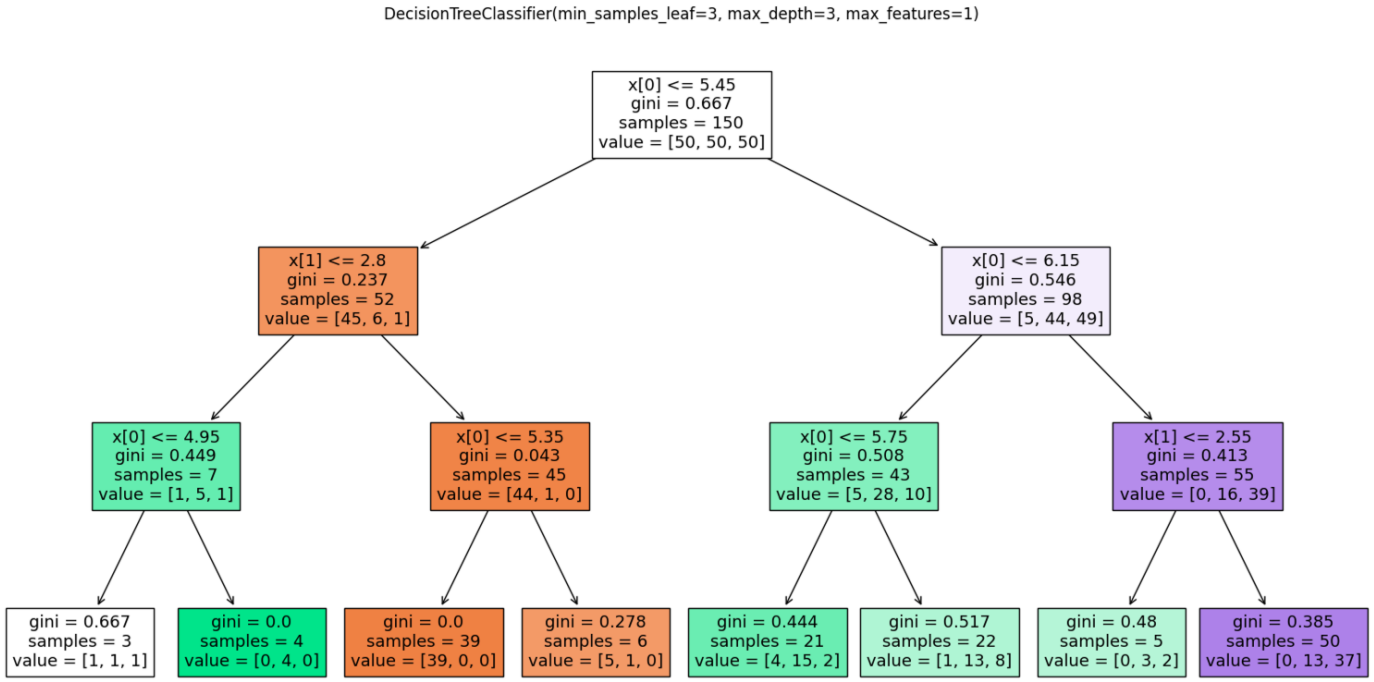


[Code on sklearn page](#) !

Markus Müller
SIB Swiss Institute of Bioinformatics

Tree hyperparameters





Markus Müller
SIB Swiss Institute of Bioinformatics

Bias and Variance

- The *bias* error is an error that occurs if the assumptions in the learning algorithm do not match the data. High bias can cause an algorithm to misclassify a significant number of data items (underfitting). The bias does not disappear if you increase the size of your training data.
- The *variance* is an error from sensitivity of the algorithms to small fluctuations in the training set. High variance may result from an algorithm modeling the random noise in the training data (overfitting). The variance becomes smaller if we increase the size of your training data.

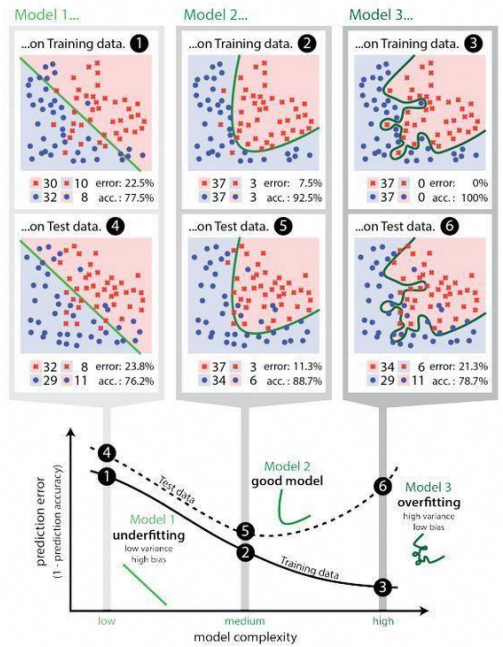
Markus Müller
Swiss Institute of Bioinformatics

Bias and Variance

- $\mathbb{E}_{D,\epsilon}\left[\left(\boldsymbol{y} - \hat{f}(x;D)\right)^2\right] = \left(Bias_D\left[\hat{f}(x;D)\right]\right)^2 + Var_D\left[\hat{f}(x;D)\right] + \sigma^2$

Prediction error | Bias error | Variance error | Data error

- Unfortunately, we cannot calculate the bias and variance if we don't know the true classification function



https://en.wikipedia.org/wiki/Bias-variance_tradeoff

Markus Müller
Swiss Institute of Bioinformatics

Bias and Variance

$$(\text{Model prediction error})^2 = (\text{Model bias error})^2 + (\text{Model variance}) + (\text{Data variance})$$

More flexible models
Boosting

More data
Bagging

Bagging and Boosting

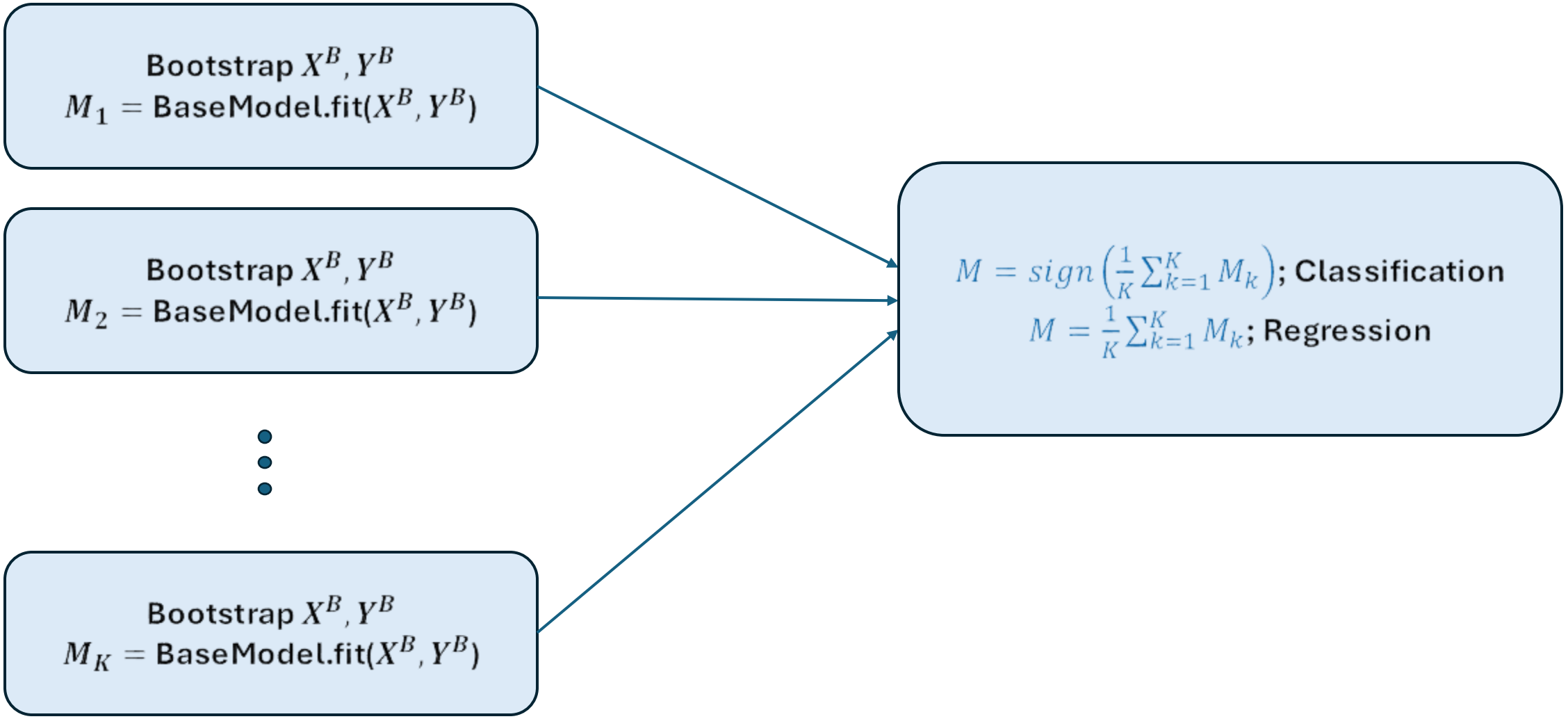
- Obtain a good classifier by combining many basic, weak classifiers
- Classifiers have two types of errors:
  - Bias (e.g. linear classifiers)
  - Variance (e.g. kNN, trees, unstable classifiers)
- Weak classifiers are for example linear classifiers (high bias for non-linear problems) or tree classifiers (high variance without pruning)
- Weak classifiers have either a high bias and/or high variance
- Weak classifiers have to be better than random labels

Bagging

- Classifiers with high variance can be improved by bootstrapping (sample with replacement) the data items $(\boldsymbol{x}_i, y_i), 1 \leq i \leq N, \boldsymbol{x}_i \in \mathbb{R}^n, y_i \in \{-1,1\}^N$, and training a new classifier C_k. This procedure is repeated K-times.
- The voting classifier $C = sign\left(\frac{1}{K}\sum_{k=1}^{K} C_k\right)$ has lower variance than individual C_k, if the C_k are unstable, and produces better predictions on test sets. If the C_k are stable (low variance), the voting classifier does not perform better.
- Bagging is not efficient in correcting bias.
- See Breiman, Machine Learning, 1996

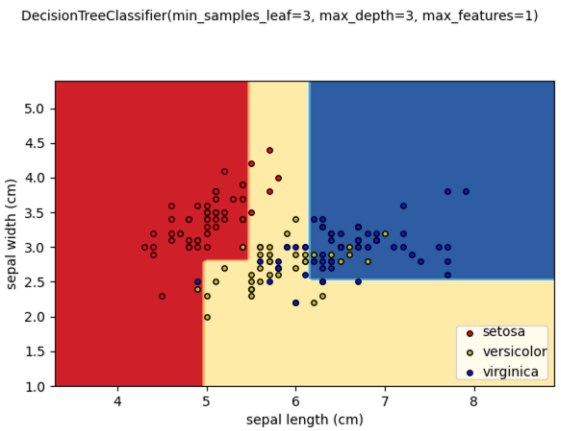
Markus Müller
Swiss Institute of Bioinformatics

Bagging

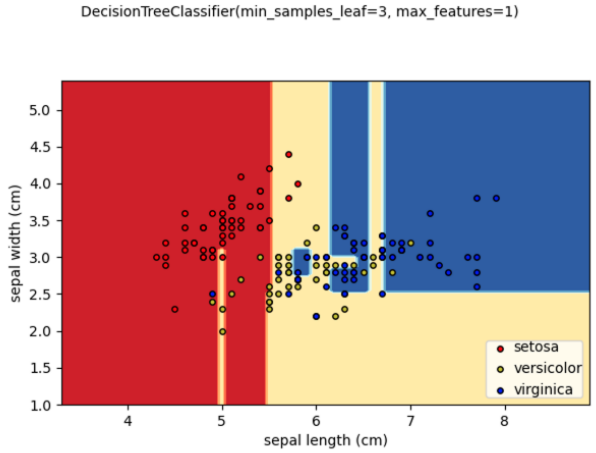


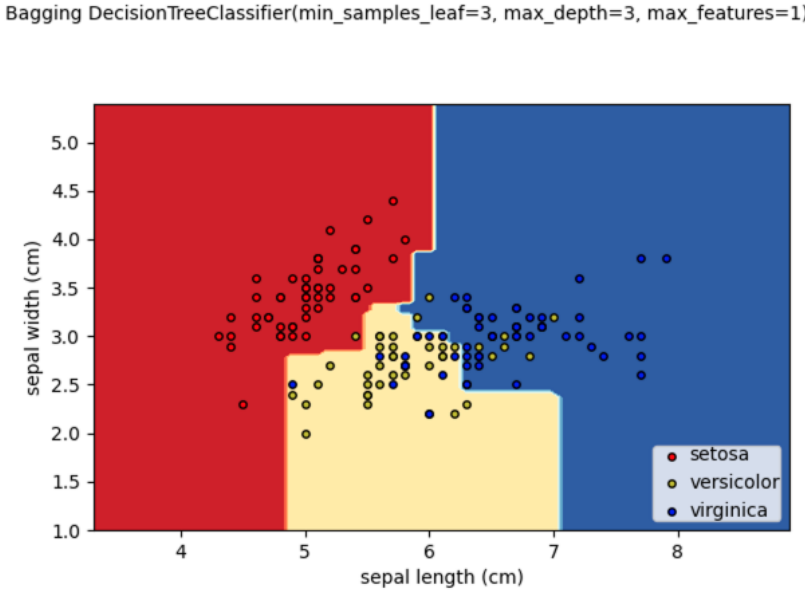
Bagging

Markus Müller
Swiss Institute of Bioinformatics

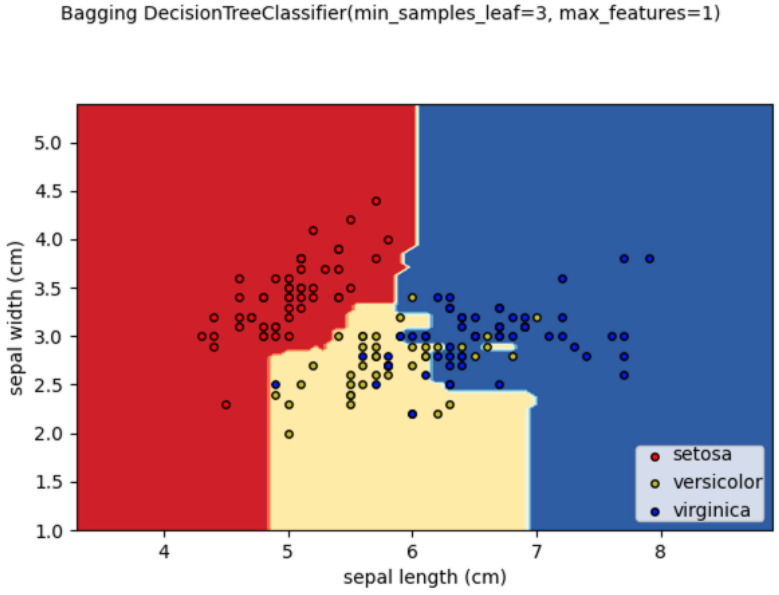


K = 1





K = 100



Markus Müller
SIB Swiss Institute of Bioinformatics

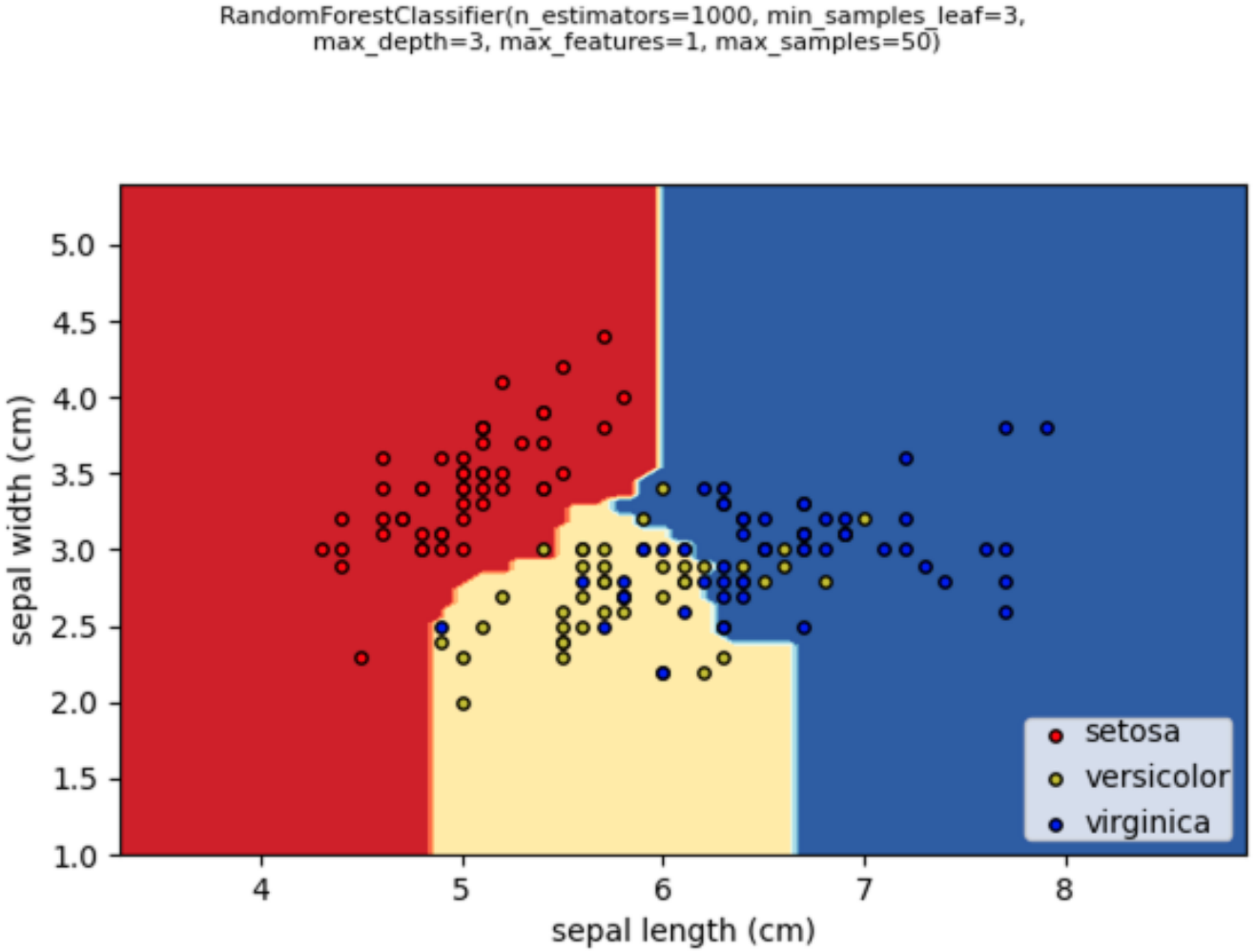
Random Forest

- Bootstrapped samples in bagging are similar and produce correlated base model outputs, which decreases the efficiency of bagging.
- Random Forest is a bagging approach where both data items and features are sampled in order to de-correlate the trees.
- A predefined number of training data points is sampled to build each tree.
- A predefined number of features is sampled at each decision node in the tree. The splits in that node can only involve the selected features.
- Base trees can be pruned with the max_depth, max_leaf_nodes, min_samples_leaf, … hyperparameters

- Breiman, Machine Learning, 2001

Markus Müller
SIB
Swiss Institute of Bioinformatics

Random Forest



RandomForestClassifier on sklearn

Markus Müller
SIB Swiss Institute of Bioinformatics

Boosting

- Bagging algorithms and RF can decrease the variance, but they cannot efficiently correct the bias of a weak classifier.
- The AdaBoost algorithm (Freund & Shapire, ICML, 1996) corrects the bias too, and offers a very powerful method, which works well for many data types.
- In each iteration, boosting tries to correct the error of the previous iteration by increasing the weights of the misclassified data items.
- The current classifier focuses on those misclassified data points and is added to the previous classifiers.

AdaBoost

- Let's assume that weak classifiers can deal with weighted data items, where each $(\boldsymbol{x}_i, y_i), 1 \leq i \leq N, \boldsymbol{x}_i \in \mathbb{R}^n, y_i \in \{-1,1\}^n$ has a different weight D_i.
- If all weak classifiers are better than random AdaBoost will quickly converge to the correct solution!

Algorithm AdaBoost.M1

Input: sequence of m examples $\langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$ with labels $y_i \in Y = \{1, \ldots, k\}$
weak learning algorithm **WeakLearn**
integer T specifying number of iterations

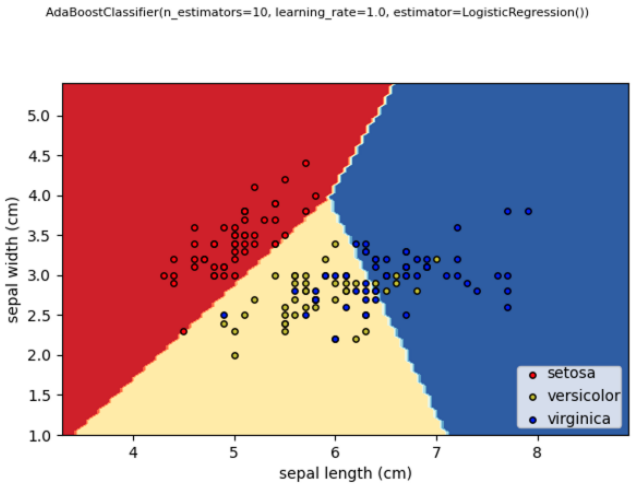
Initialize $D_1(i) = 1/m$ for all i.

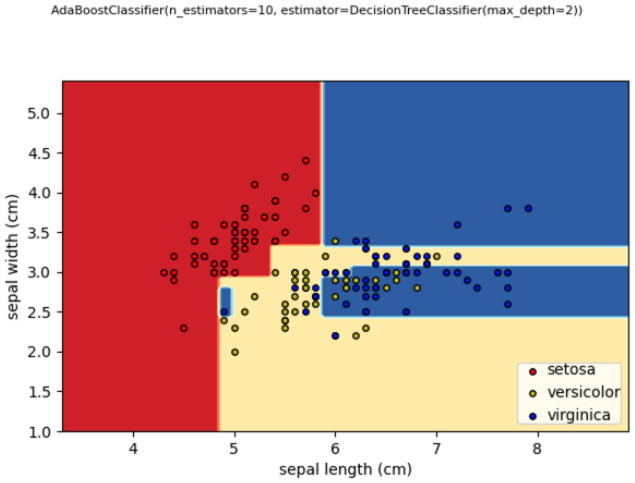
Do for $t = 1, 2, \ldots, T$

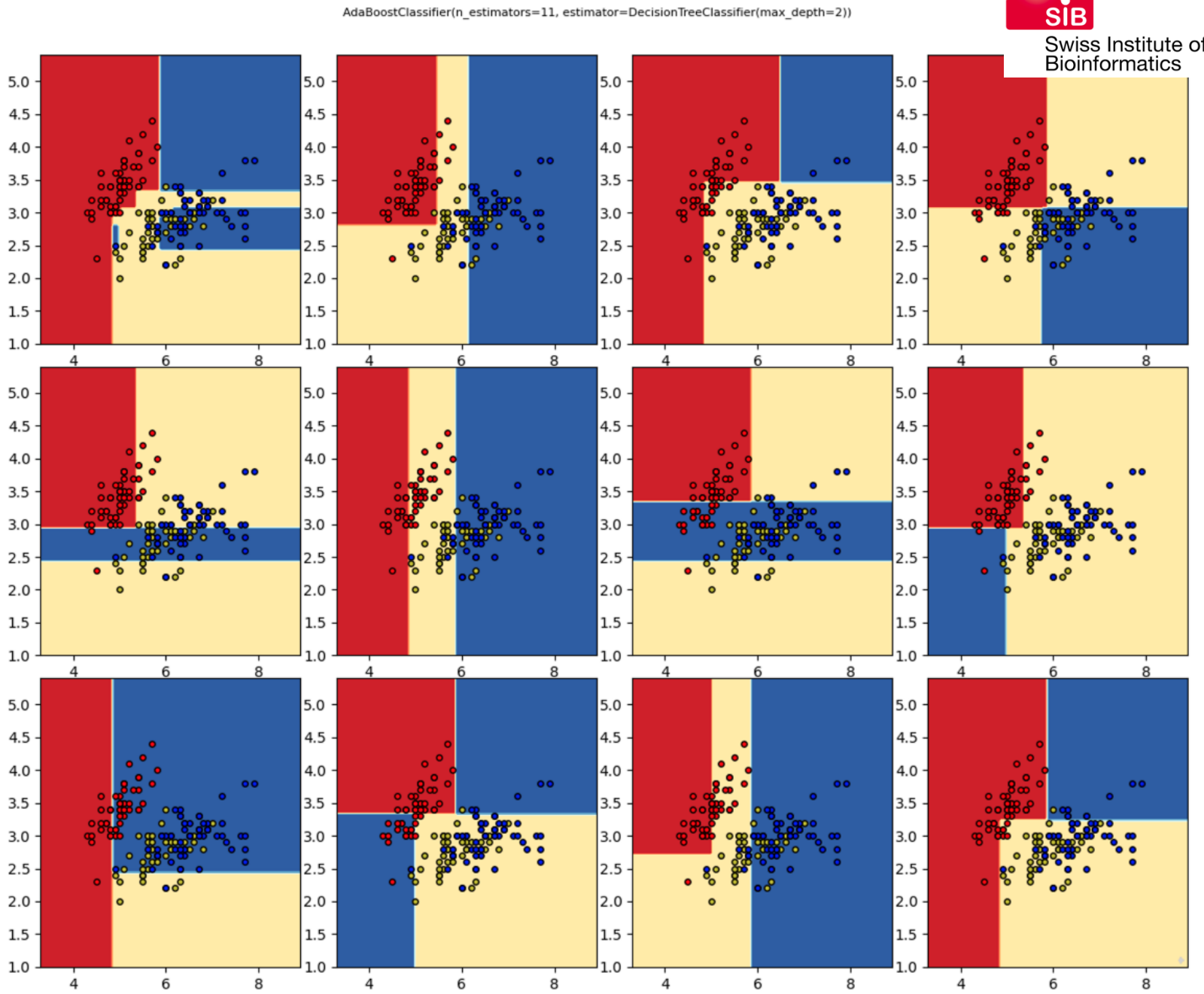
1. Call **WeakLearn**, providing it with the distribution D_t.
2. Get back a hypothesis $h_t : X \to Y$.
3. Calculate the error of h_t: $\epsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$. If $\epsilon_t > 1/2$, then set $T = t - 1$ and abort loop.
4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$.
5. Update distribution D_t: $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_t & \text{if } h_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$
   where Z_t is a normalization constant (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis: $h_{fin}(x) = \arg\max_{y \in Y} \sum_{t:h_t(x)=y} \log \frac{1}{\beta_t}$.

AdaBoost







Markus Müller
SIB Swiss Institute of Bioinformatics

Gradient Boosting

- Mason et al. (Mason et al., NIPS, 1999) showed that AdaBoost is a special case of the Gradient Boosting framework.
- In the gradient boosting framework, you choose a cost function and select a weak classifier, which minimizes the gradient of the cost function (in function space)
- Mason et al. showed that some cost functions can perform better than AdaBoost, which is more prone to overfitting.
- For a thorough comparison of bagging and boosting methods see Dietterich, Machine Learning, 2000.

Markus Müller
Swiss Institute of Bioinformatics

XGBoost: a scalable and flexible gradient boosting system

- Extreme gradient boosting system
- Introduced by Chen & Guestrin in 2016 (Chen & Guestrin, KDD 16, 2016)
- Efficient implementation for tree and linear boosting for regression, classification, and ranking.
- Support for parallelization, cluster computing, and GPU.
- Very flexible and configurable API.
  - Callbacks, checkpoints, early stopping, dropout (dart), incremental learning, configurable sampling, regularization, many losses and metrics, choice of algorithm, …
- XGBoost provides robust and complementary results when compared to neural network models on tabular data

Markus Müller
SIB Swiss Institute of Bioinformatics

XGBoost Framework

- Exact greedy (slower) and approximate (faster) algorithms
- Deal with missing values (sparsity aware)
- Include feature interaction constraints
- Include monotonicity constraints
- Include regularization (shrinkage, sampling, penalties, dropout)
- Callbacks, early stopping
- Update existing model with new data
- Parallel on multiple CPUs, support for Spark distributed computing
- GPU support
- Out-of-core computing (data larger than memory)
- Multiple outputs
- Since 2.0: optimize histo size, learning to rank, quantile regression, federated learning

Markus Müller
SIB Swiss Institute of Bioinformatics

XGBoost

- $\hat{y}_i(\boldsymbol{x}_i) = \sum_{k=1}^{K} f_k(\boldsymbol{x}_i)$; $f_k(\boldsymbol{x}_i) =$ base tree classifier
- Boosting algo: fit tree to error $y_i - \hat{y}_i^{t-1}$ of previous iteration:

$$\mathcal{L}^t\left(\hat{y}_i^{t-1}, f_t\right) = \sum_{i=1}^{N} l\left(y_i, \hat{y}_i^{t-1} + f_t(\boldsymbol{x}_i)\right) + \Omega(f_t)$$

  - loss function $l(\hat{y}_i, y_i)$ (e.g. square loss $l(\hat{y}_i, y_i) = \frac{1}{2}(\hat{y}_i - y_i)^2$)
  - regularization term $\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{i=1}^{T} v_i^2$
    - T: number of leaves; $\boldsymbol{v}$: values at leaves

Markus Müller
SIB Swiss Institute of Bioinformatics

XGBoost

- Gradient boosting:
  - $\mathcal{L}^t\left(\hat{y}_i^{t-1}, f_t\right) \approx \sum_{i=1}^{N}\left\{l\left(y_i, \hat{y}_i^{t-1}\right) + g_i f_t(\boldsymbol{x}_i) + \frac{1}{2} h_i f_t^2(\boldsymbol{x}_i)\right\} + \Omega(f_t)$

$$= \sum_{i=1}^{N} l\left(y_i, \hat{y}_i^{t-1}\right) + \underbrace{\sum_{i=1}^{N}\left\{g_i f_t(\boldsymbol{x}_i) + \frac{1}{2} h_i f_t^2(\boldsymbol{x}_i)\right\} + \Omega(f_t)}_{\tilde{\mathcal{L}}^t\left(\hat{y}_i^{t-1}, f_t\right)}$$

$$g_i = \frac{\partial}{\partial \hat{y}_i^{t-1}} l\left(y_i, \hat{y}_i^{t-1}\right); \; h_i = \frac{\partial^2}{\partial^2 \hat{y}_i^{t-1}} l\left(y_i, \hat{y}_i^{t-1}\right)$$

gradients hessians

Markus Müller
SIB Swiss Institute of Bioinformatics

XGBoost - square loss

- $l(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$
- Gradient boosting: $g_i = \left(\hat{y}_i^{t-1} - y_i\right); h_i = 1$
  - $\tilde{\mathcal{L}}^t\left(\hat{y}_i^{t-1}, f_t\right) = \sum_{i=1}^{N} \left\{ \left(\hat{y}_i^{t-1} - y_i\right) f_t(\boldsymbol{x}_i) + \frac{1}{2} f_t^2(\boldsymbol{x}_i) \right\} + \Omega(f_t)$

$\left(\hat{y}_i^{t-1} - y_i\right) > 0: f_t(\boldsymbol{x}_i) < 0$
$\left(\hat{y}_i^{t-1} - y_i\right) \sim 0: f_t(\boldsymbol{x}_i) \sim 0$
$\left(\hat{y}_i^{t-1} - y_i\right) < 0: f_t(\boldsymbol{x}_i) > 0$

Regularization term

Markus Müller
SIB Swiss Institute of Bioinformatics

XGBoost - negative log likelyhood

- $l\left(y, f(\boldsymbol{x})\right) = -[y \log \hat{p} + (y-1)\log(1-\hat{p})]; \;\; \hat{p} = \frac{1}{1+e^{-f(\boldsymbol{x})}}$
- Gradient boosting: $g_i = (\hat{p}_i - y_i); h_i = \hat{p}_i(1-\hat{p}_i)$
  - $\tilde{\mathcal{L}}^t\left(\hat{y}_i^{t-1}, f_t\right) = \sum_{i=1}^{N} \left\{ \underbrace{\left(\hat{p}_i^{t-1} - y_i\right) f_t(\boldsymbol{x}_i)}_{\left(\hat{p}_i^{t-1} - y_i\right) > 0: f_t(\boldsymbol{x}_i) < 0} + \underbrace{\frac{1}{2}\hat{p}_i^{t-1}\left(1-\hat{p}_i^{t-1}\right) f_t^2(\boldsymbol{x}_i)}_{\left(\hat{p}_i^{t-1} - y_i\right) \sim 0: f_t(\boldsymbol{x}_i) \sim 0} \right\} + \Omega(f_t)$

$\left(\hat{p}_i^{t-1} - y_i\right) < 0: f_t(\boldsymbol{x}_i) > 0$

Regularization term

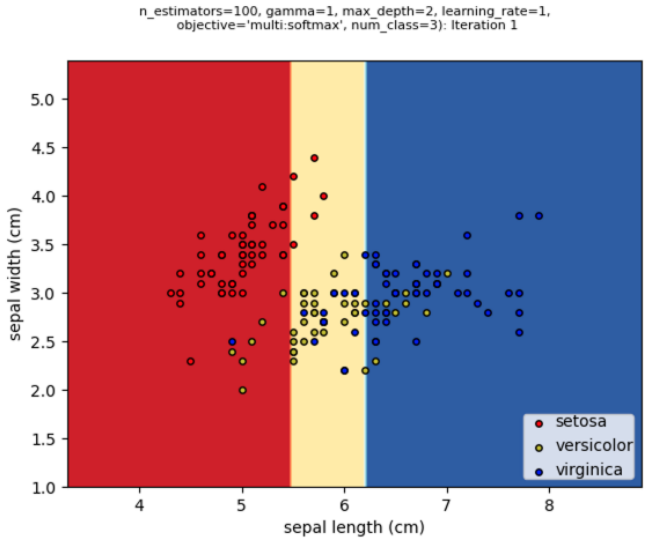
Markus Müller
Swiss Institute of Bioinformatics

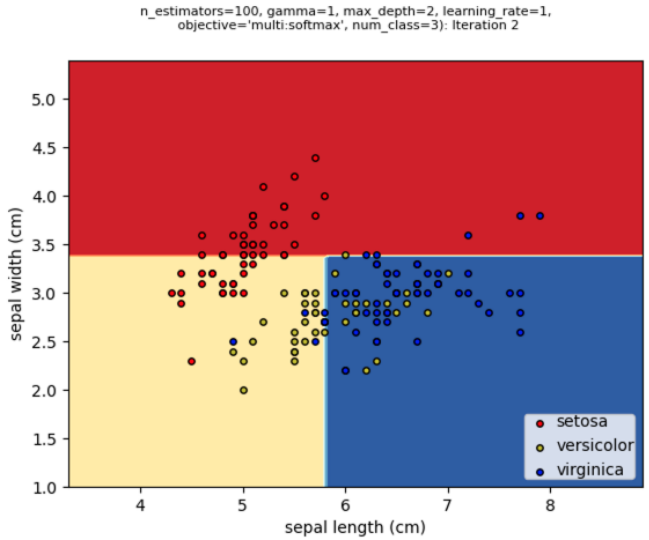
XGBoost

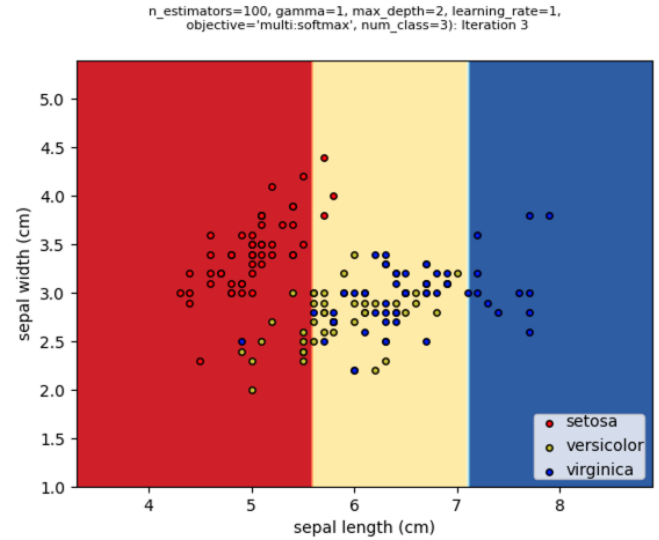
- Initialize $f_0(\boldsymbol{x}) = \underset{\boldsymbol{v},Tree}{\operatorname{argmin}} \mathcal{L}\left(\hat{y}_i^0, f\right)$
- for $t = 1..K$
  - Sample columns of $\boldsymbol{X}$ and/or rows of $\boldsymbol{X}, \mathbf{y}$
  - $f_t(\boldsymbol{x}) = \underset{\boldsymbol{v},Tree}{\operatorname{argmin}} \tilde{\mathcal{L}}^t\left(\hat{y}_i^{t-1}, f_t\right)$

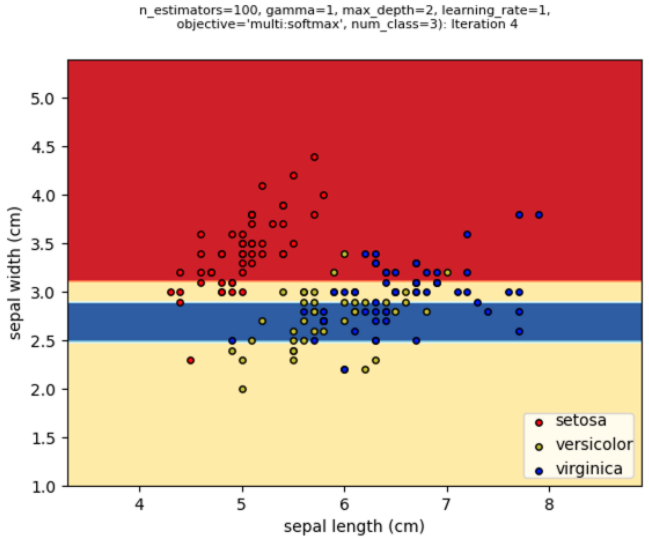
    $\hat{y}_i^t(\boldsymbol{x}_i) = \hat{y}_i^{t-1}(\boldsymbol{x}_i) + \eta\, f_t(\boldsymbol{x})$, shrinkage parameter eta: $0 < \eta \leq 1$
- Output $\hat{y}_i^K(\boldsymbol{x}_i)$

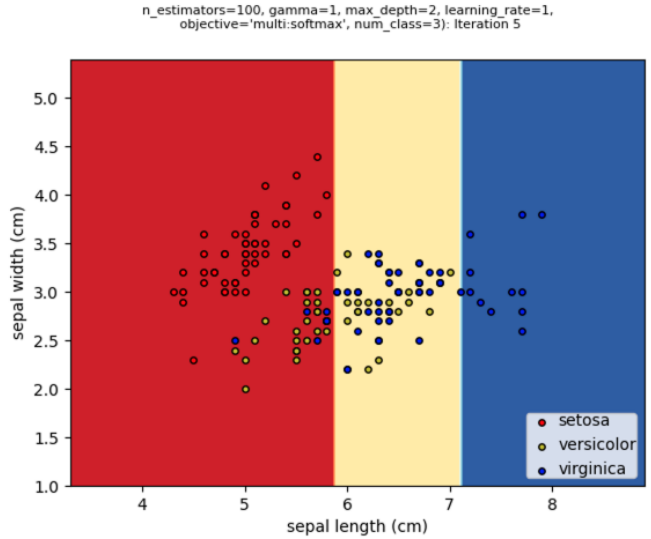
XGBoost

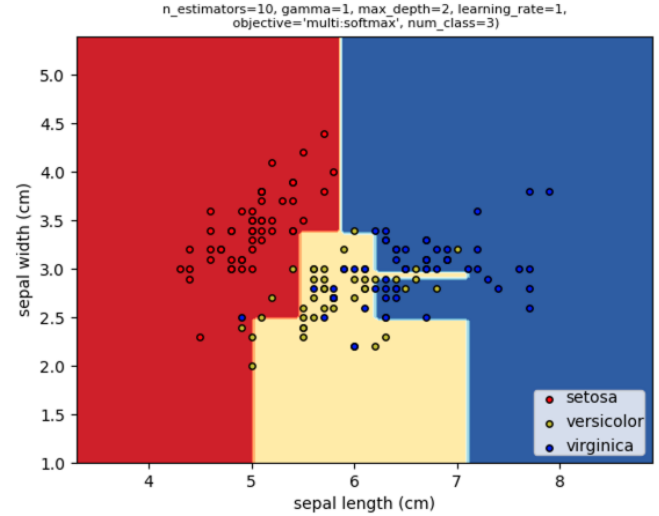

n_estimators=100, gamma=1, max_depth=2, learning_rate=1,
objective='multi:softmax', num_class=3): Iteration 1
sepal width (cm)
sepal length (cm)
setosa
versicolor
virginica


n_estimators=100, gamma=1, max_depth=2, learning_rate=1,
objective='multi:softmax', num_class=3): Iteration 2
sepal width (cm)
sepal length (cm)
setosa
versicolor
virginica


n_estimators=100, gamma=1, max_depth=2, learning_rate=1,
objective='multi:softmax', num_class=3): Iteration 3
sepal width (cm)
sepal length (cm)
setosa
versicolor
virginica


n_estimators=100, gamma=1, max_depth=2, learning_rate=1,
objective='multi:softmax', num_class=3): Iteration 4
sepal width (cm)
sepal length (cm)
setosa
versicolor
virginica


n_estimators=100, gamma=1, max_depth=2, learning_rate=1,
objective='multi:softmax', num_class=3): Iteration 5
sepal width (cm)
sepal length (cm)
setosa
versicolor
virginica


n_estimators=10, gamma=1, max_depth=2, learning_rate=1,
objective='multi:softmax', num_class=3)
sepal width (cm)
sepal length (cm)
setosa
versicolor
virginica

Markus Müller
SIB Swiss Institute of Bioinformatics

XGBoost

- How does XGBoost find the optimal tree?
- Let's assume the tree f_t is known appart from the leaf values $\boldsymbol{v}$
- Define $I_j = \{i | leaf(\boldsymbol{x}_i) = j\}$, i.e. all points $\boldsymbol{x}_i$ from the training set that belong to leaf j
- Then the optimal leaf values are:
  - $v_j = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$
  - The change in the loss function introduced by a split can then be calculated:

square loss: $v = -\frac{\sum_{i \in I}(\hat{y}_i - y_i)}{|I| + \lambda}$

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{\left(\sum_{i \in I_L} g_i\right)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{\left(\sum_{i \in I_R} g_i\right)^2}{\sum_{i \in R} h_i + \lambda} - \frac{\left(\sum_{i \in I} g_i\right)^2}{\sum_{i \in I} h_i + \lambda}\right] - \gamma$$

- Finding f_t that minimizes $\tilde{\mathcal{L}}^t(f_t)$ is difficult. The greedy *gbtree* algorithm finds the feature and the split $I = I_L \cup I_R$ with the largest $\mathcal{L}_{split} > 0$. If always $\mathcal{L}_{split} \leq 0$ then there is no further split.

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{\left(\sum_{i \in I_L}(\hat{y}_i - y_i)\right)^2}{|I_L| + \lambda} + \frac{\left(\sum_{i \in I_R}(\hat{y}_i - y_i)\right)^2}{|I_R| + \lambda} - \frac{\left(\sum_{i \in I}(\hat{y}_i - y_i)\right)^2}{|I| + \lambda}\right] - \gamma$$

Markus Müller
SIB Swiss Institute of Bioinformatics

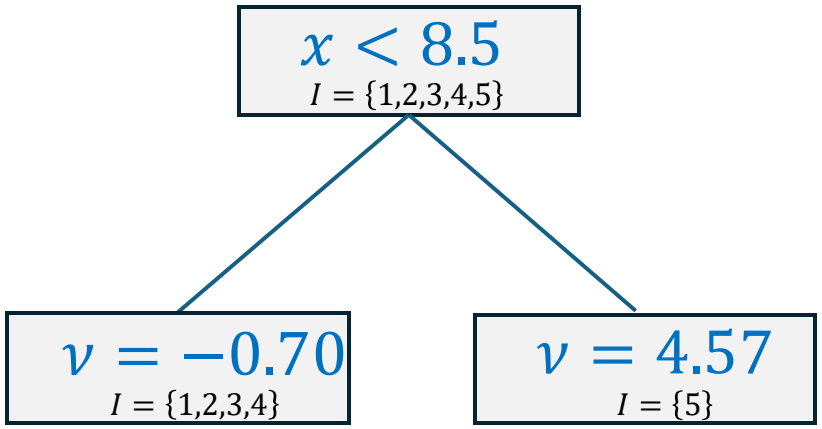
XGBoost

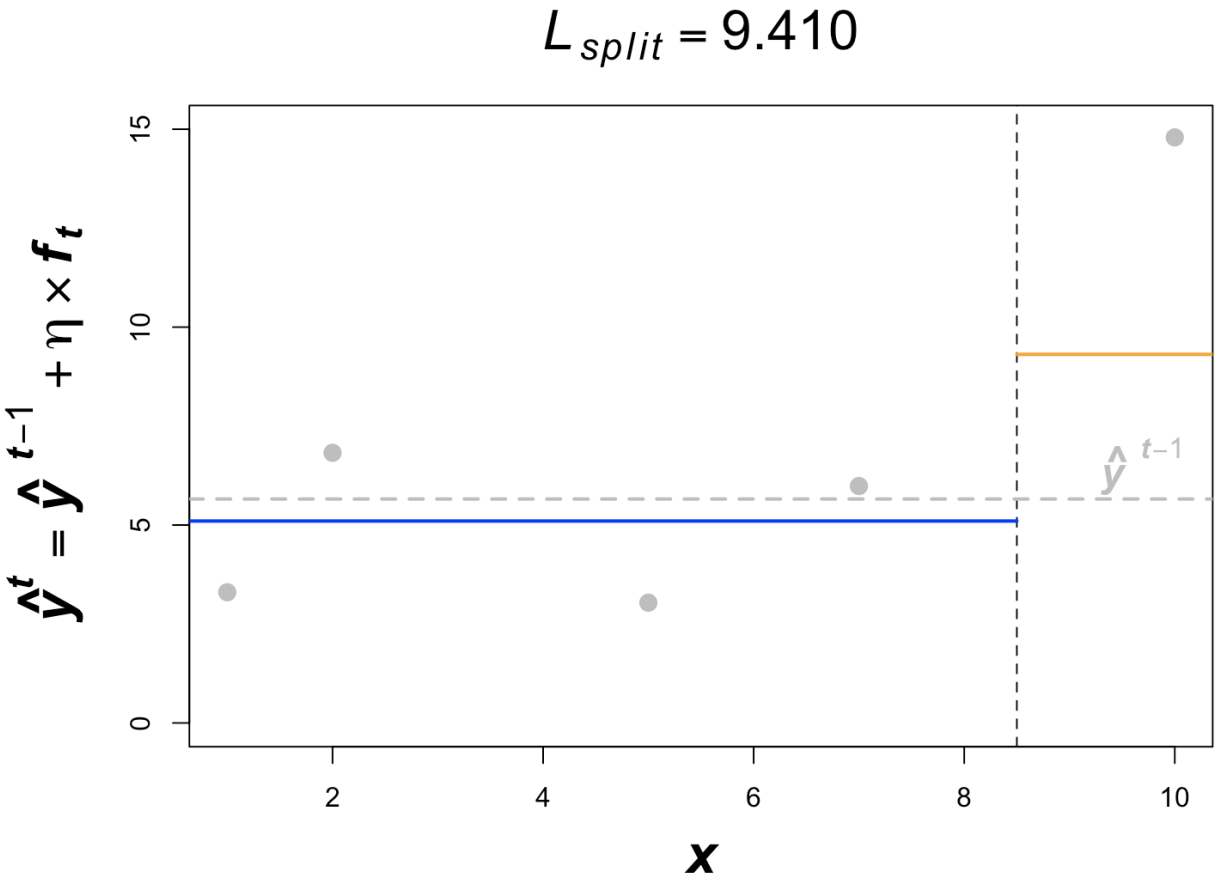
Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for j *in sorted*$(I,$ *by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

If the maximal score is larger than γ, we have a valid split

XGBoost





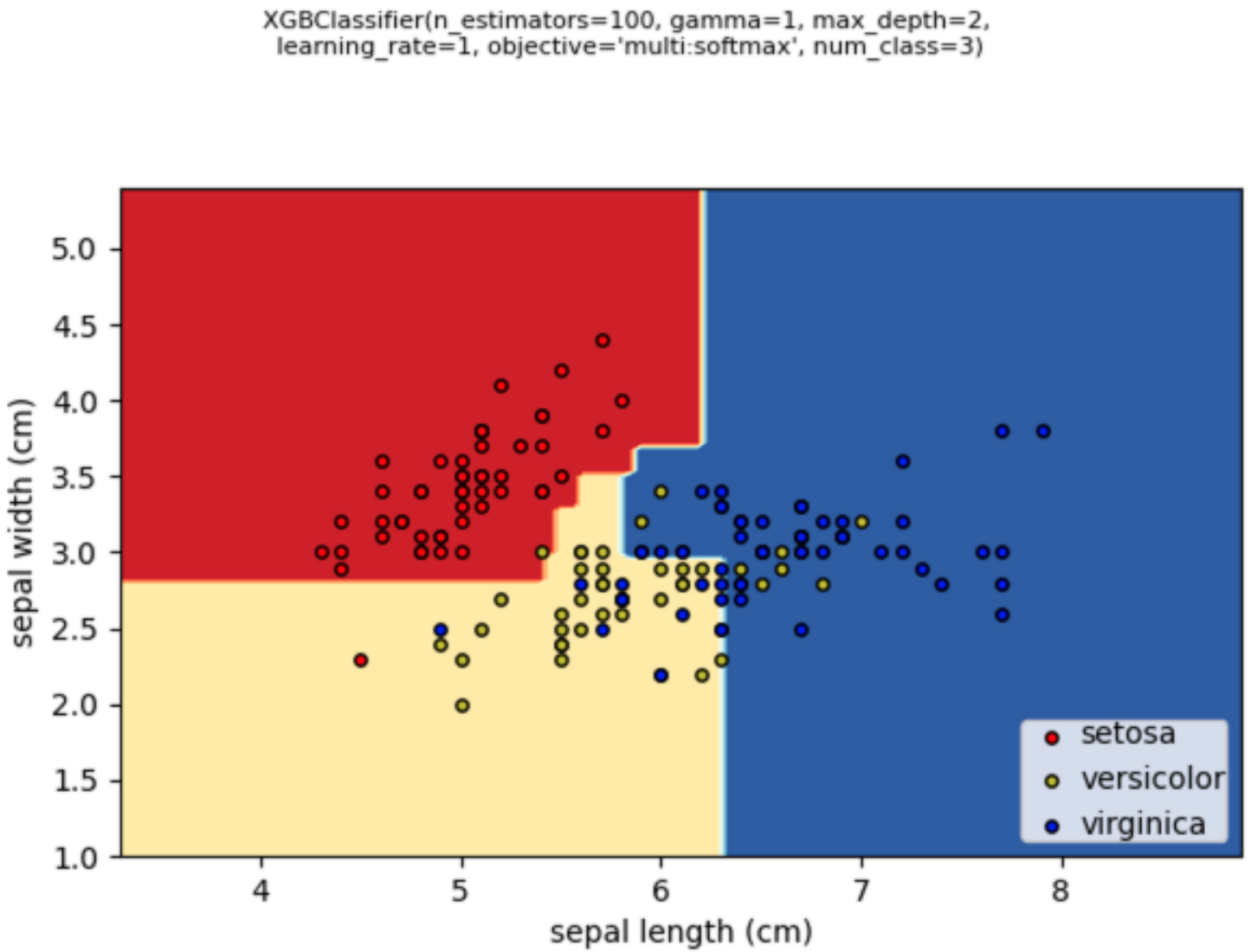
$$\eta = 0.8;\ \lambda = 1; \gamma = 10$$

$$v = -\frac{\sum_{i \in I}(\hat{y}_i^{t-1} - y_i)}{|I| + \lambda}$$

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{\left(\sum_{i \in I_L}(\hat{y}_i^{t-1} - y_i)\right)^2}{|I_L| + \lambda} + \frac{\left(\sum_{i \in I_R}(\hat{y}_i^{t-1} - y_i)\right)^2}{|I_R| + \lambda} - \frac{\left(\sum_{i \in I}(\hat{y}_i^{t-1} - y_i)\right)^2}{|I| + \lambda}\right] - \gamma$$

Markus Müller
SIB
Swiss Institute of Bioinformatics

XGBoost



XGBoost – Approximate algo and missing values

- Instead of testing each value of a numeric feature as a potential threshold, XGBoost has an approximate *histogram*-based algorithm, which only allows splits at certain percentiles.
- In case of missing values, XGBoost calculates a default direction for each node. The default direction gives the largest gain $\mathcal{L}_{split}$ if all missing values are assigned to this direction.
- For categorical features, XGBoost supports one-hot-encoding and target encoding.

XGBoost – approximate histogram splits

Algorithm 2: Approximate Algorithm for Split Finding

for $k = 1$ ***to*** m **do**

  Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature k.

  Proposal can be done per tree (global), or per split(local).

end

for $k = 1$ ***to*** m **do**

  $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$

  $H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$

end

Follow same step as in previous section to find max score only among proposed splits.

XGBoost – missing values

Data

Example	Age	Gender
X1	?	male
X2	15	?
X3	25	female

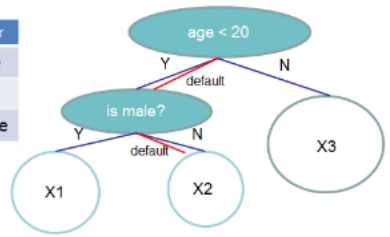


Figure 4: Tree structure with default directions. An example will be classified into the default direction when the feature needed for the split is missing.

Algorithm 3: Sparsity-aware Split Finding

Input: I, instance set of current node
Input: $I_k = \{i \in I | x_{ik} \neq \text{missing}\}$
Input: d, feature dimension
Also applies to the approximate setting, only collect statistics of non-missing entries into buckets
$gain \leftarrow 0$
$G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ ***to*** m **do**
  // enumerate missing value goto right
  $G_L \leftarrow 0,\ H_L \leftarrow 0$
  for j *in sorted*$(I_k,$ *ascent order by* $\mathbf{x}_{jk})$ **do**
    $G_L \leftarrow G_L + g_j,\ H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L,\ H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
  // enumerate missing value goto left
  $G_R \leftarrow 0,\ H_R \leftarrow 0$
  for j *in sorted*$(I_k,$ *descent order by* $\mathbf{x}_{jk})$ **do**
    $G_R \leftarrow G_R + g_j,\ H_R \leftarrow H_R + h_j$
    $G_L \leftarrow G - G_R,\ H_L \leftarrow H - H_R$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split and default directions with max gain

Markus Müller
Swiss Institute of Bioinformatics

XGBoost Hyperparameters

- *booster*: base model (gbtree, dart or gblinear)
- *device*: GPU or CPU
- *eta*: step size shrinkage (η)
- *gamma*: minimum loss reduction required to make a further partition on a leaf node (γ)
- *max_depth*: maximum depth of a tree
- *min_child_weight*: minimum sum of hessians (h_i) needed in a child
- *subsample*: subsample ratio of the training instances
- *colsample_bytree*, *colsample_bylevel*, *colsample_bynode*: parameters for subsampling of columns
- *lambda*: L2 regularization term on node values (λ)
- *alpha*: L1 regularization term on node values
- *tree_method*: The tree construction algorithm (exact, approx, and hist)
- *scale_pos_weight*: useful for unbalanced classes
- *process_type*: used to update existing model (transfer learning)
- *max_leaves*: maximum number of leaves
- *monotone_constraints*: constraint of variable monotonicity (regression)
- *interaction_constraints*: constraints for interaction representing permitted interactions
- *objective*: loss function
- *eval_metric*: evaluation metrics for validation data
- *max_cat_to_onehot*: how to deal with categorical features

Markus Müller
SIB Swiss Institute of Bioinformatics

XGBoost +/-

- When to use XGBoost
  - Classification and regression problems
  - Categorical variables
  - Missing values
  - Tabular data with heterogeneous features
  - Very flexible and comprehensive API
  - Applicable to large data
  - …
- When not to use XGBoost
  - $n > N$, i.e. more features than observations
  - Regression tasks, where output needs to be continuous
  - Extrapolation tasks
  - For image and language processing NN are preferred

- Other excellent gradient boosting tools are LightGBM, CatBoost,..

Hyperparameters

Markus Müller
SIB Swiss Institute of Bioinformatics

- In the classification task, we would like to predict the outcome $\boldsymbol{y}$ from the features $\boldsymbol{X}$.
- We choose a classifier $\mathcal{C}_{\boldsymbol{\phi}}$ with *hyper parameters* $\boldsymbol{\phi}$ and *internal parameters* $\boldsymbol{\theta}$.
- Hyper parameters (e.g k in $k-NN$, kernel in SVM, learning rate in LR) have to be specified by the user or learned in CV loop outside the classifier.
- The internal parameters $\boldsymbol{\theta}$ are learned by the classifier (e.g. the coefficients in LR, or support vector coefficients in SVM).
- The classifier learns those *internal parameters* $\boldsymbol{\theta}$ that are able to predict $\boldsymbol{y}$ from the features $\boldsymbol{X}$ with the lowest error:

$$\boldsymbol{\theta} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathcal{L}\left(\mathcal{C}_{\boldsymbol{\phi}}.\mathbf{predict}(\boldsymbol{X}, \boldsymbol{\theta}), \boldsymbol{y}\right)$$

where the loss function $\mathcal{L}$ measures the error between predicted ($\mathcal{C}_{\boldsymbol{\phi}}.\mathbf{predict}(\boldsymbol{X}, \boldsymbol{\theta})$) and known ($\boldsymbol{y}$) outcomes.

Markus Müller
Swiss Institute of Bioinformatics

Hyperparameter optimization

- ML models have internal parameters, which are learned by the training algorithm, and hyperparameters, which have to be set before training.
- Both internal - and hyperparameters have a strong impact on the performance of the ML model.
- There is a tradeoff between CPU/GPU cycles spent on hyperparameter optimization and training. Allocating more cycles for hyperparameter optimization can be more rewarding.
- Hyperparameter optimization is implemented as an outer loop over the CV loop

Markus Müller
Swiss Institute of Bioinformatics

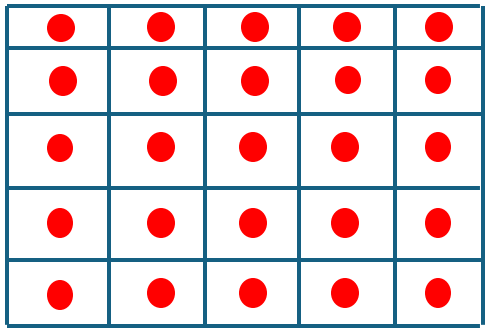
Hyperparameter optimization and CV

Hyperparameter space Φ
Dataset $\boldsymbol{X}$, Labels $\boldsymbol{y}$
Number of iterations M
Number of CV folds F

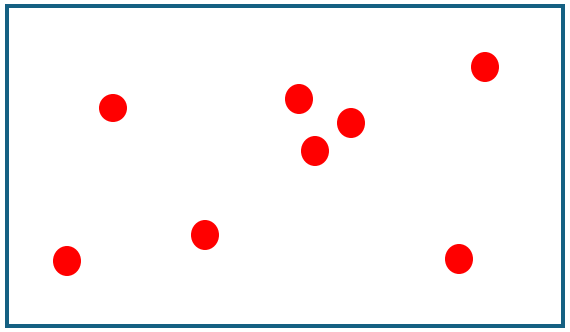
$S_{max} = -\infty, \boldsymbol{\phi}_{max} = None$
Hyperparameter loop → for $i = 1, .., M$:
    $\boldsymbol{\phi} = choose(\Phi)$
CV loop → for $j = 1, .., F$:
        $\boldsymbol{X}_{train}, \boldsymbol{X}_{test}, \boldsymbol{y}_{train}, \boldsymbol{y}_{test} = split(\boldsymbol{X}, j)$
        $C_{\boldsymbol{\phi},\boldsymbol{\theta}} = \mathcal{C}_{\boldsymbol{\phi}}.\text{train}(\boldsymbol{X}_{train})$
        $S_j = S\left(C_{\boldsymbol{\phi},\boldsymbol{\theta}}.\textbf{predict}(\boldsymbol{X}_{test}), \boldsymbol{y}_{test}\right)$
    $\bar{S} = \frac{1}{F}\sum_{j=1}^{F} S_j$
    if $\bar{S} > S_{max}$: $S_{max} = \bar{S}, \boldsymbol{\phi}_{max} = \boldsymbol{\phi}$

Markus Müller
Swiss Institute of Bioinformatics

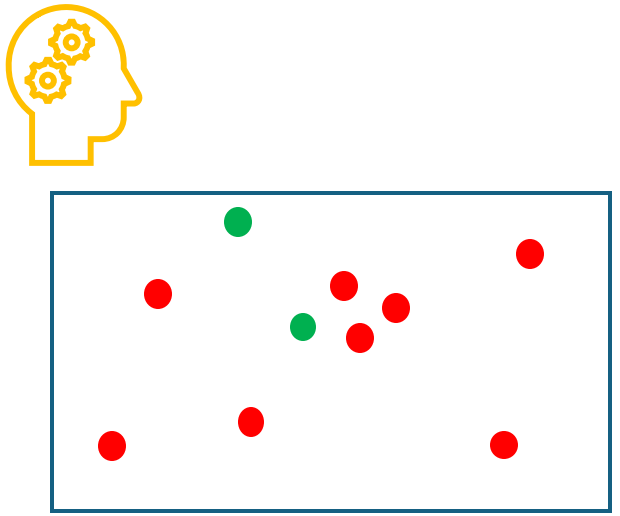
HP searches



Grid search:
Parse all HP grid cells.
Slow if many features
Resolution of grid limited by CPU time



Random search:
Random sampling based on prior knowledge.
Difficult for many features (curse of dimensionality)



Directed searches:
Define new HPs based on prior and acquired knowledge.
Often more efficient than grid or random search for many features

Hyperparameter optimization

- Hyperparameters have grid or tree-structured configuration spaces of discrete, ordinal, and continuous variables.
- Complete tree or grid searches are often too slow for large search spaces:
  - Random searches
  - Gaussian process (GP) and tree-parzen estimator (TPE) algorithms suggest which parameter combination to test in the next step based on previous results.
- Hyperopt (Bergstra et al. Neurips 2011) implements both strategies. It is open-source and available on https://github.com/hyperopt/hyperopt.
- Other tools are Hypermax (tree parzen estimator), Scikit-Optimizer (skopt) (Bayesian), Spearmint (Bayesian), Hyperband (fast random armed bandit), Optuna (dynamic search space, pruning, easy API), Optunity (grid-, random-, evolutionary-, swarm-, Bayesian search)

Hyperopt

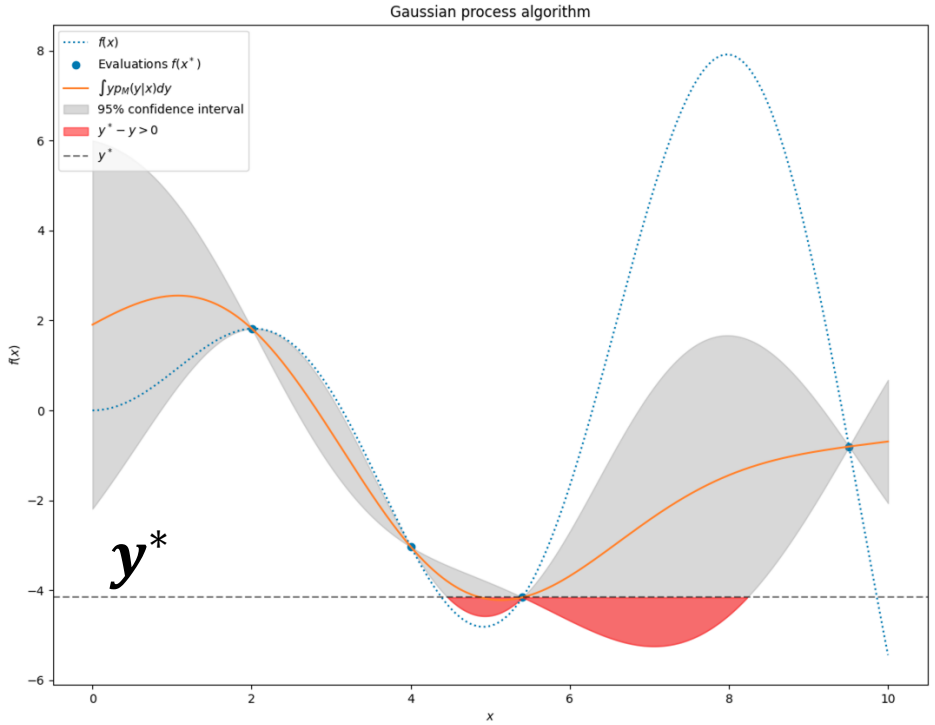
- Random algorithms find optimal solutions faster and more accurately than grid searches (Bergstra & Bengio, J. Mach. Learn. Res., 2012)
- Therefore Hyperopt implements a random search algorithm.
- Hyperopt also offers two other model-based search algorithms:
  - Gaussian process approach (GP) (not implemented in python)
  - Tree-structured Parzen window approach (TPE)
- Hyperopt supports nested parameter spaces
- Hyperopt supports warm start using previous results

Markus Müller
Swiss Institute of Bioinformatics

Hyperopt's GP algorithm

$$S(x, M) = EI_{y^*}(x) = \int_{-\infty}^{+\infty} \max(y^* - y, 0) p_M(y|x) dy \,; y^* = \min\{f(\boldsymbol{x}_i), 1 \leq i \leq n\}$$

$p_M(y|x)$: estimated by gaussian process (GP)

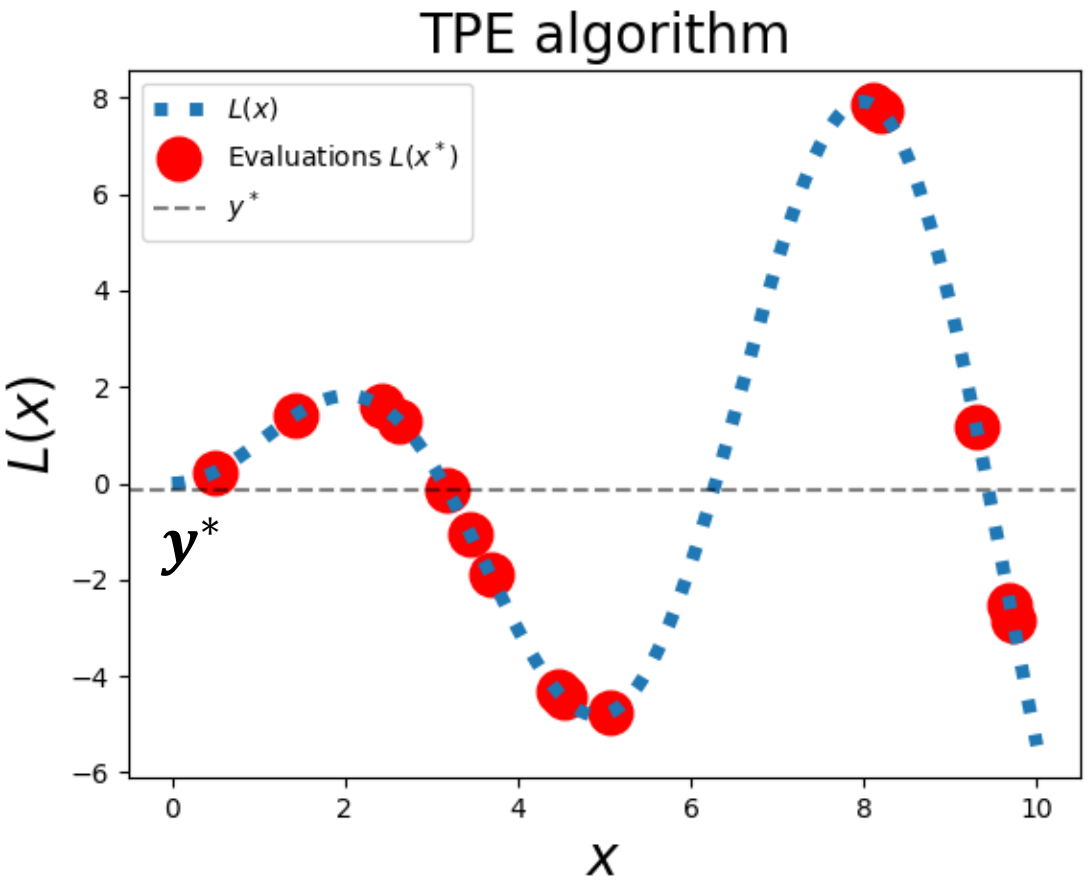


The GP algorithms will give a large *EI* value for hyperscore values x if a large probability mass is below y^*. This is the case for

- values of x close to the minimum
- values of x in not yet explored regions, which have a large uncertainty

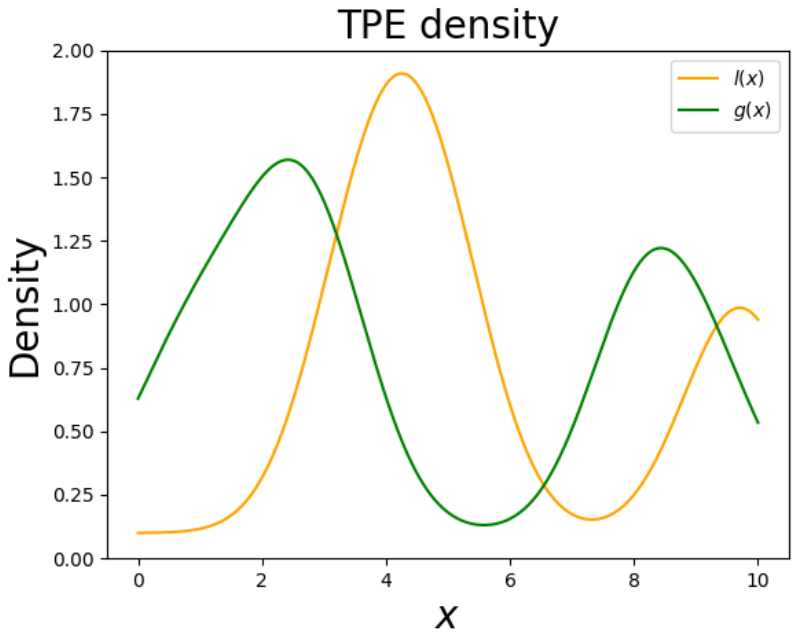
Markus Müller
Swiss Institute of Bioinformatics

Hyperopt's TPE algorithm



$g(x)$

$l(x)$



$$p_M(x|y) = \begin{cases} l(x) \text{ if } y \leq y^* \\ g(x) \text{ if } y > y^* \end{cases}$$

$$p_M(x) = \gamma l(x) + (1-\gamma) g(x)$$

$$p_M(y|x) = \frac{p_M(x|y)p(y)}{p(x)}$$

TPE: Tree structured Parzen estimator

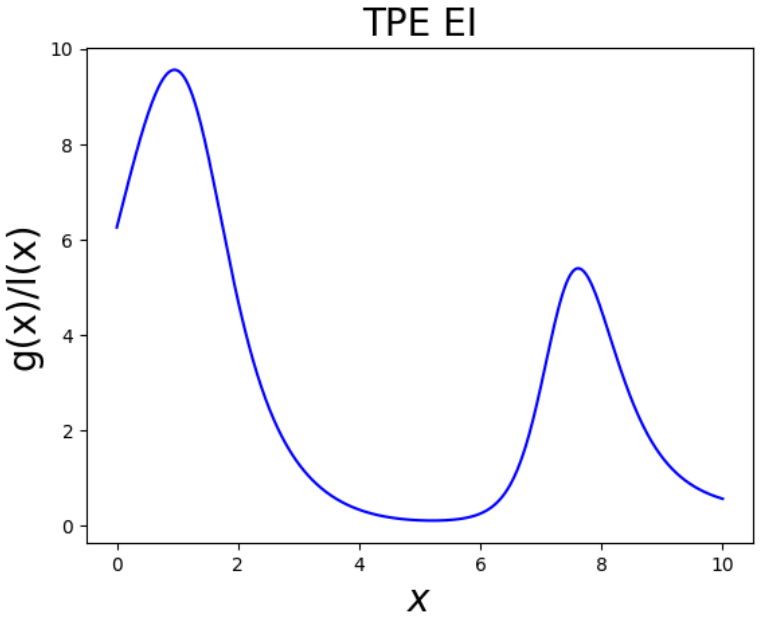
$$y^* = \gamma - \text{quantile}: p(y < y^*) = \gamma$$

Markus Müller
Swiss Institute of Bioinformatics

Hyperopt's TPE algorithm

- $EI_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y) \frac{p_M(x|y)p(y)}{p(x)} dy$

$$\propto 1/\left(\gamma + \frac{g(x)}{l(x)}(1-\gamma)\right)$$



- The TPE algorithm will sample hyperscores from $l(x)$ and return the x^* with the smallest $g(x)/l(x)$ value.
- $g(x)$ and $l(x)$ are hierarchical as defined in the HP space variable.
- $g(x)$ and $l(x)$ are initialized by the uniform or gaussian prior distributions specified in HP space variable. After each iteration, these distributions are updated with the newly sampled x^*values.

Markus Müller
SIB Swiss Institute of Bioinformatics

Hyperopt's model-based algorithm

SMBO(f, M_0, T, S)

1 $\mathcal{H} \leftarrow \emptyset$,
2 For $t \leftarrow 1$ **to** T,
3 $x^* \leftarrow \text{argmin}_x \; S(x, M_{t-1})$,
4 Evaluate $f(x^*)$, ▷ *Expensive step*
5 $\mathcal{H} \leftarrow \mathcal{H} \cup (x^*, f(x^*))$,
6 Fit a new model M_t to $\mathcal{H}$.
7 **return** $\mathcal{H}$

Figure 1: The pseudo-code of generic Sequential Model-Based Optimization.

$$S(x, M) = EI_{y^*}(x) = \int_{-\infty}^{+\infty} \max(y^* - y, 0) p_M(y|x) dy$$

EI: Expected improvement for score values x

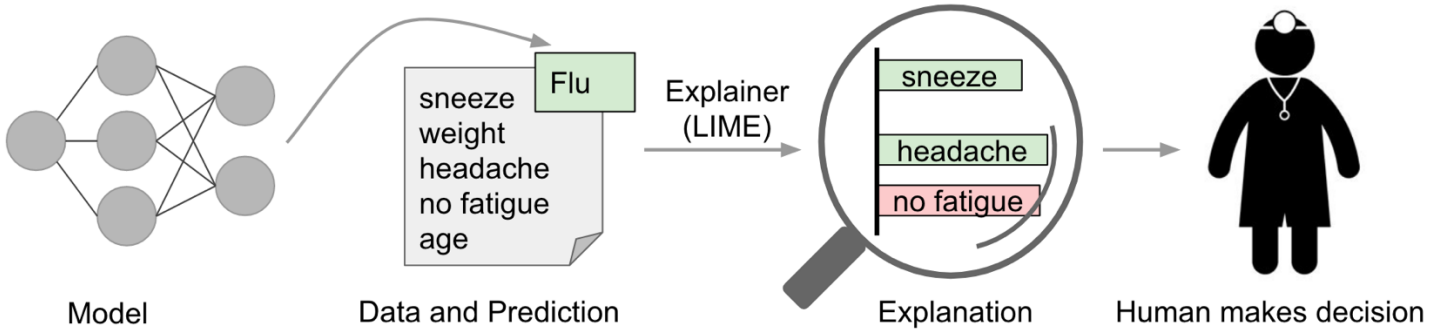
Markus Müller
Swiss Institute of Bioinformatics

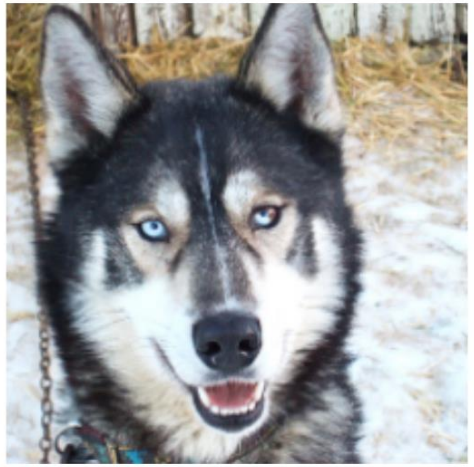
Interpretable Machine Learning (IML)

- ML tools are optimized for metrics such as accuracy, which can lead to accurate, but wrong models.
- Many models are black boxes, i.e. they do not reveal in a human readable form how they make their decision.
- In order to understand a model, we need to approximate the model's decision with a simple human readable model.
- If this simple approximation corresponds to human intuition and is faithful (i.e. truly reflects the underlying ML model), we can better trust the ML model.

Markus Müller
Swiss Institute of Bioinformatics

Interpretable Machine Learning (IML)





(a) Husky classified as wolf

(b) Explanation

[Ribeiro et al., ACM SIGKDD, 2016]

Markus Müller
Swiss Institute of Bioinformatics

Interpretable Machine Learning (IML)

- What does «*interpretable*» mean in the context of ML?
- A good IML model has to deliver an interpretation of an ML model prediction that is human-readable, plausible, and faithful.
  - Plausible: how convincing is the interpretation to humans
  - Faithful:
    - the IML interpretation approximates the true reasoning process of the ML model
    - the interpretation should not change drastically if different datasets are used for the ML model training or prediction.
    - it should be similar for different ML models, which make similar predictions.
- https://christophm.github.io/interpretable-ml-book/
- Chen et al, Nature Methods, 2024

Markus Müller
Swiss Institute of Bioinformatics

Interpretable Machine Learning (IML)

- Some ML methods such as linear regression, logistic regression, small decision trees, GLM's, GAM's are interpretable by humans, i.e. they tell you explicitly how features contribute to the outcome.
- Other ML methods (such as NN, SVM, and boosting methods) can perform very well but are not interpretable by humans.
- However, understanding prediction by ML model is important to trust the model, study biases, and eventually improve it.
- Either, these black-box methods are customized for interpretability or one has to use *post hoc* tools such as Shap, Lime, DeepLIFT, …
- These ad-hoc tools provide human interpretable simple models, which approximate the underlying complex ML model.
- Such simple models may be feature importance scores (linear approximation), simple trees, or decision rules.

Markus Müller
Swiss Institute of Bioinformatics

Interpretable Machine Learning (IML)

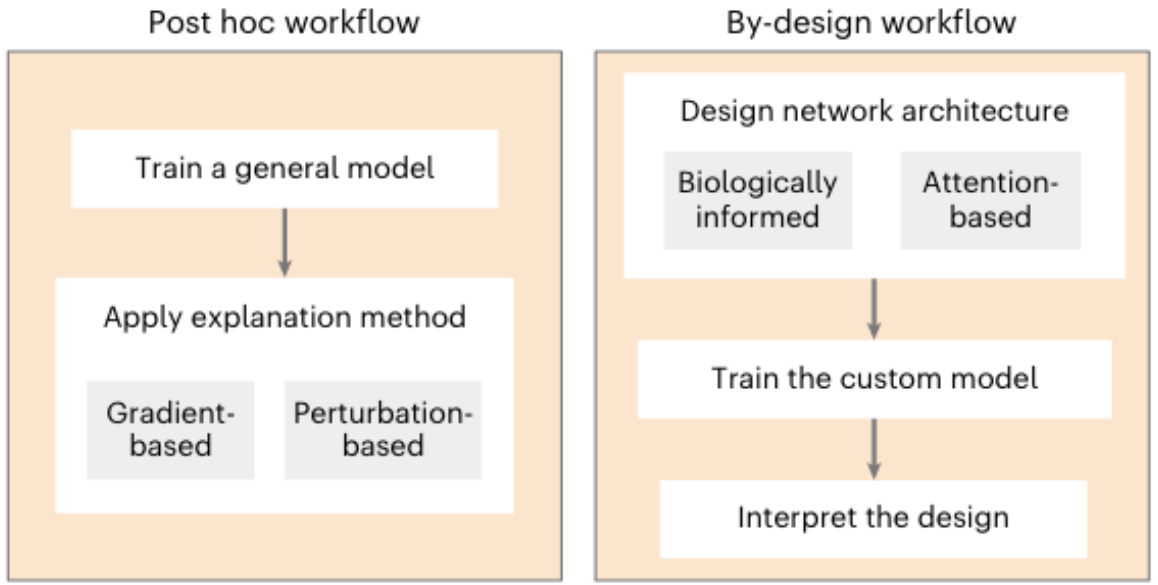


Fig. 1 | The two main IML approaches used to explain prediction models are post hoc explanations and by-design explanations. Each approach has its canonical workflows and popular types of IML methods: post hoc explanations are model agnostic and are applied after a model is trained, while by-design explanations are typically built into or inherent to the model architecture.

Chen et al, Nature Methods, 2024

Markus Müller
SIB Swiss Institute of Bioinformatics

Interpretable Machine Learning (IML)

- However, there is no consensus, which IML method works best (Krishna et al, Mach. Learning Research, 2024; Agarwal et al. NeurIPS, 2022)
- Need to evaluate the IML interpretation scores by these criteria:
  - Plausibility: is the interpretation plausible? Do the IML interpretations agree with prior knowledge?
  - Faithfulness: Does it give the correct answers on synthetic datasets (https://open-xai.github.io/)? Do they reflect the model's reasoning process?
  - Stability: are the scores robust for different hyperparameter choices for the underlying model? Are they robust with regard to data sampling and different datasets? Do different IML methods give similar results?

Markus Müller
SIB Swiss Institute of Bioinformatics

IML Guidelines

- Use several IML methods and compare the results
- Use several datasets if possible, or randomly sample from your data, to show that your results are robust.
- Use different hyperparameter choices for the underlying model. Use more than one model.
- Evaluate the plausibility and faithfulness of the IML interpretation:
  - Do they agree with prior or expert knowledge?
  - Do they represent the model's reasoning process?
- Avoid cherry-picking nice examples, but always report full results.

- Chen et al, Nature Methods, 2024
- Alvarez-Melis & Jaakkola, 2018

IML

- Local interpretation:
  - An interpretation is calculated for a single data vector $\boldsymbol{x}$.
  - A local interpretation tells you, how a specific data vector $\boldsymbol{x}$ was predicted by the ML model. It tells you which features are predominantly involved in the prediction.

- Global interpretation:
  - An interpretation is calculated on a whole data set $\{\boldsymbol{x}_k\}, k = 1, .., N$
  - The global interpretation tells you, which features are predominantly involved in the prediction of a whole dataset.
  - A global interpretation can be obtained by *averaging* over local interpretations.

LIME

- LIME: Local Interpretable Model-Agnostic Explanations
- Approximates any ML model by a simple local model
- Provides control over the complexity of the local model
- Provides global interpretation of the ML model
- https://github.com/marcotcr/lime
- Watch: https://www.youtube.com/watch?v=hUnRCxnydCc

- Ribeiro et al., ACM SIGKDD, 2016

LIME: local interpretation

Markus Müller
SIB Swiss Institute of Bioinformatics

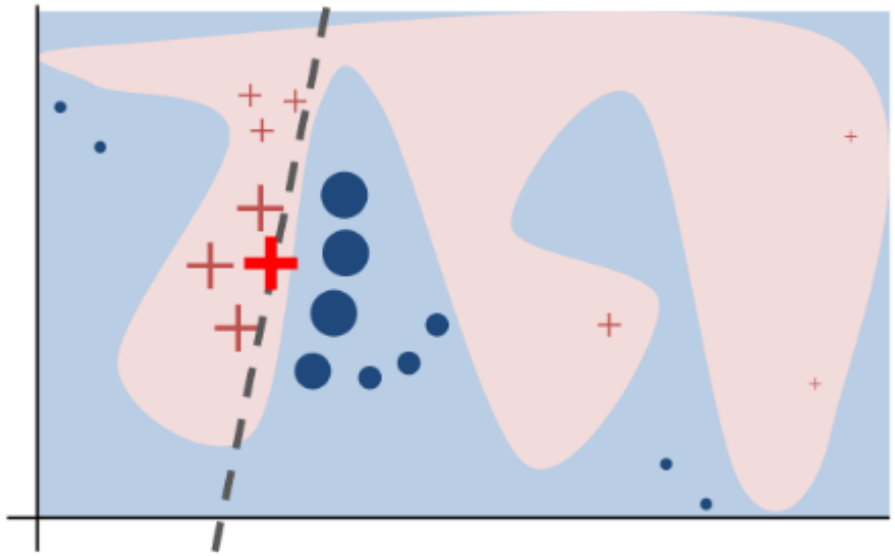


Figure 3: Toy example to present intuition for LIME. The black-box model's complex decision function f (unknown to LIME) is represented by the blue/pink background, which cannot be approximated well by a linear model. The bold red cross is the instance being explained. LIME samples instances, gets predictions using f, and weighs them by the proximity to the instance being explained (represented here by size). The dashed line is the learned explanation that is locally (but not globally) faithful.

$$\xi(\boldsymbol{x}) = \underset{g \in G}{\operatorname{argmin}}\, \mathcal{L}(f, g, \pi_{\boldsymbol{x}}) + \Omega(g)$$

Markus Müller
SIB
Swiss Institute of Bioinformatics

LIME : local interpretation

Algorithm 1 Sparse Linear Explanations using LIME

Require: Classifier f, Number of samples N
Require: Instance x, and its interpretable version x'
Require: Similarity kernel π_x, Length of explanation K

$\mathcal{Z} \leftarrow \{\}$
for $i \in \{1, 2, 3, ..., N\}$ **do**
  $z'_i \leftarrow sample\_around(x')$
  $\mathcal{Z} \leftarrow \mathcal{Z} \cup \langle z'_i, f(z_i), \pi_x(z_i) \rangle$
end for
$w \leftarrow$ K-Lasso$(\mathcal{Z}, K)$ ▷ with z'_i as features, $f(z)$ as target
return w

Markus Müller
SIB Swiss Institute of Bioinformatics

LIME : global interpretation

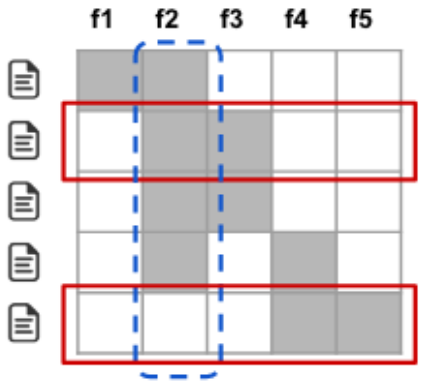


Figure 5: Toy example $\mathcal{W}$. Rows represent instances (documents) and columns represent features (words). Feature f2 (dotted blue) has the highest importance. Rows 2 and 5 (in red) would be selected by the pick procedure, covering all but feature f1.

Algorithm 2 Submodular pick (SP) algorithm

Require: Instances X, Budget B
  for all $x_i \in X$ **do**
    $\mathcal{W}_i \leftarrow$ **explain**(x_i, x_i') ▷ Using Algorithm 1
  end for
  for $j \in \{1 \dots d'\}$ **do**
    $I_j \leftarrow \sqrt{\sum_{i=1}^{n} |\mathcal{W}_{ij}|}$ ▷ Compute feature importances
  end for
  $V \leftarrow \{\}$
  while $|V| < B$ **do** ▷ Greedy optimization of Eq (4)
    $V \leftarrow V \cup \text{argmax}_i \, c(V \cup \{i\}, \mathcal{W}, I)$
  end while
  return V

Markus Müller
SIB Swiss Institute of Bioinformatics

Shapley Values

- Developed by economics Noble prize winner Loyd Shapley in 1951
  How do you evaluate the importance of a single player in a team of n players $\mathcal{N}$?
- Let's assume the outcome of the game is measured by a score f.
  E.g. f might be the percentage of games won, or the probability of a classifier.
- If only a subset $S \subset \mathcal{N}$ of all players takes part in the game, the score is $f(S)$
- Idea: the importance of a player is evaluated relative to all subsets of players S.

Markus Müller
Swiss Institute of Bioinformatics

Shapley Values

- The Shapley value φ_i of player i is defined by:

$$\varphi_i(f) = \underbrace{\frac{1}{n!}\sum_{P}}_{\text{Sum over all permutations } P \text{ of } \{1,\dots,n\}} \underbrace{\left(f\left(S_i^P \cup \{i\}\right) - f\left(S_i^P\right)\right)}_{\text{Gain in } f \text{ by including player } i \text{ in team } S_i^P = \{P_j \mid j \prec P(i)\}}$$

Markus Müller
SIB Swiss Institute of Bioinformatics

Shapley Values

- Shapley values have several desirable properties.
  - Efficiency: $\sum_{i \in \mathcal{N}} \varphi_i(f) = \varphi(\mathcal{N}) - \varphi(\emptyset)$
  - Monotonicity: $f(S \cup \{i\}) - f(S) \geq f'(S \cup \{i\}) - f'(S)\ \forall S \subset \mathcal{N} \Rightarrow\ \varphi_i(f) \geq \varphi_i(f')$
  - Null player: If $f(S \cup \{i\}) = f(S)\ \forall S \subset \mathcal{N}, S \cap \{i\} = \emptyset$, then $\varphi_i(f) = 0$
  - Symmetry: If $f(S \cup \{i\}) = f(S \cup \{j\})\ \forall S \subset \mathcal{N}, S \cap \{i, j\} = \emptyset$, then $\varphi_i(f) = \varphi_j(f)$
  - Linearity: $\varphi_i(f + u) = \varphi_i(f) + \varphi_i(u)$
- The Shapley formula is the only formula that has all these properties
- There are extensions of Shapley values for feature groups and feature interactions.

- BUT: how can we calculate the Shapley values efficiently for large n?

Markus Müller
Swiss Institute of Bioinformatics

Shapley Values for ML

- Let's assume we have a trained ML model, which predicts the output $f(\boldsymbol{x}_j)$ of a vector $\boldsymbol{x}_j = (x_{j1}, .., x_{jn}) \in \mathbb{R}^n$ from our dataset $\mathcal{D} = \{\boldsymbol{x}_j\}, 1 \leq j \leq N$
- $S \subset \mathcal{N} = \{1, .., n\}$. How do we define $f(\boldsymbol{x}, S \cup \{i\}) - f(\boldsymbol{x}, S)$?

- Define mask: $\boldsymbol{x}_S(\boldsymbol{x}, \boldsymbol{y}) \equiv \begin{cases} x_i \text{ for } i \in S \\ y_i \text{ for } i \notin S \end{cases}$

$$f(\boldsymbol{x}, S) = E_{\boldsymbol{x}_{\mathcal{N}\setminus S}}(f(\boldsymbol{x})) = \sum_{\boldsymbol{y}\in\mathcal{D}} \left[f(\boldsymbol{x}_S(\boldsymbol{x}, \boldsymbol{y})) - f(\boldsymbol{y})\right]$$

$$\varphi_i(\boldsymbol{x}) = \sum_{S\subset\mathcal{N}/\{i\}} \frac{|S|!\,(n - |S| - 1)!}{n!} \left(f(\boldsymbol{x}, S \cup \{i\}) - f(\boldsymbol{x}, S)\right)$$

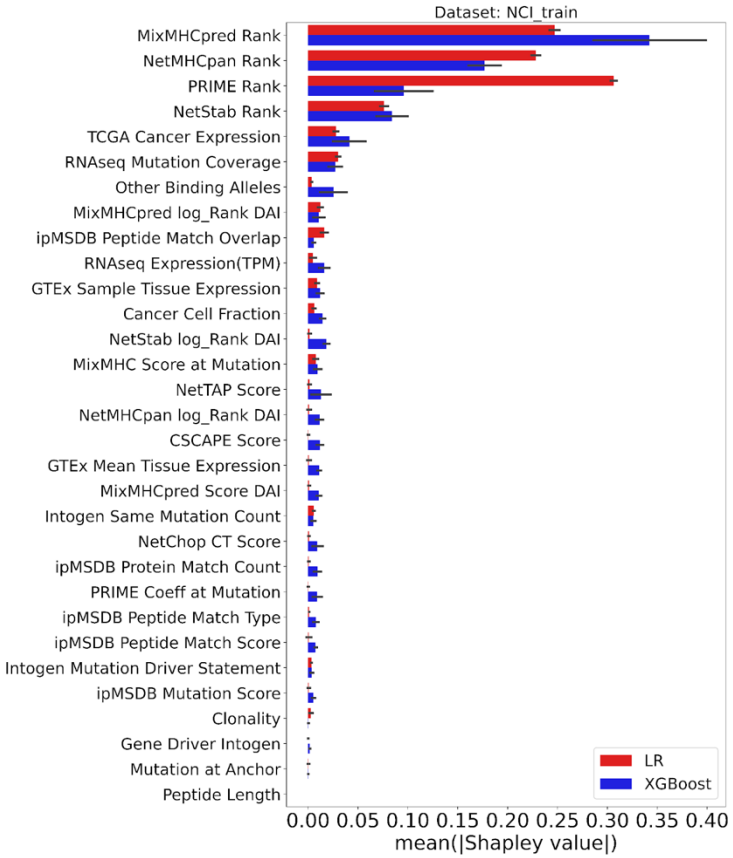
- $\sum_{i=1}^{n} \varphi_i(\boldsymbol{x}) = f(\boldsymbol{x}) - E_{\boldsymbol{x}}\left(f(\boldsymbol{x})\right)$

Shapley Values for ML

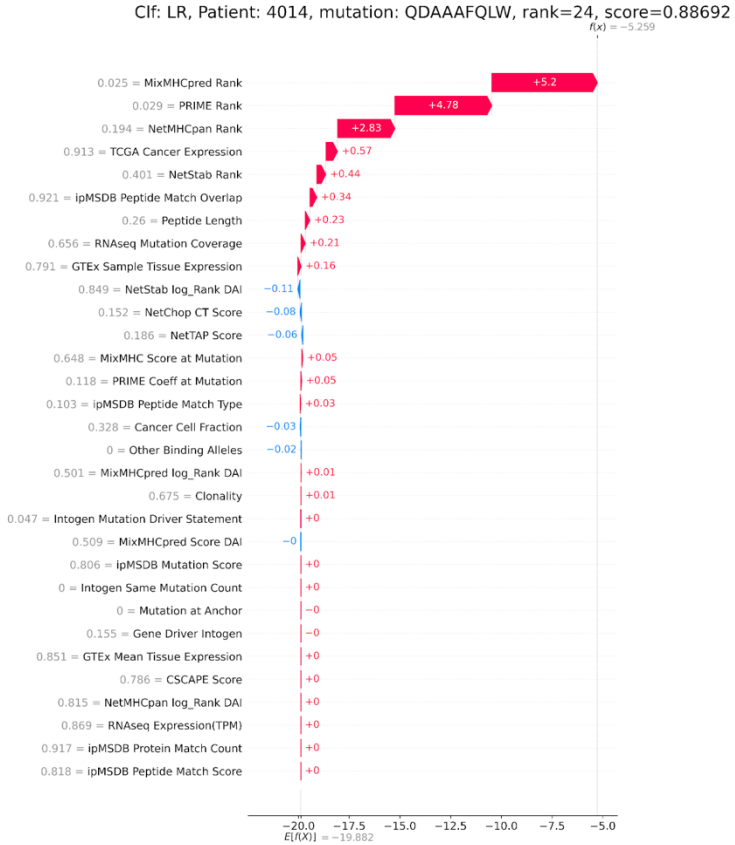
- Shapley values are too time-consuming to calculate exactly for large n
- Strumbelj & Kononenko, LNCS, 2011 provide a general sampling method
- Lundberg et al., ICML, 2017 provide an efficient sampling method called *Tree SHAP* to calculate Shapley values for assemblies of trees (included in XGBoost)
- Lundberg & Lee, NIPS, 2017 provide a unified view of LIME, DeepLIFT, and Shapley values. They also provide Kernel SHAP, a variant of LIME, which outputs Shapley values.
- Python package SHAP (SHapley Additive exPlanations) available on github: https://github.com/slundberg/shap . It provides efficient calculation of Shapley values and many diagnostic plots.

Markus Müller
Swiss Institute of Bioinformatics

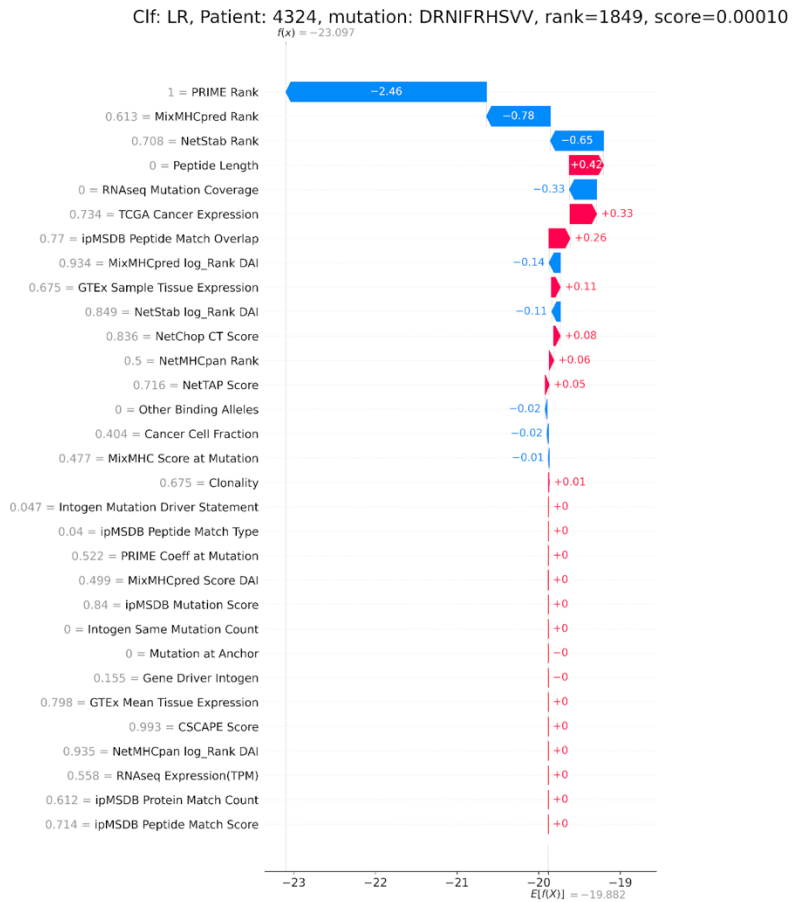
IML with Shapley values



Good prediction



Bad prediction



Müller et al. Immunity, 2024