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

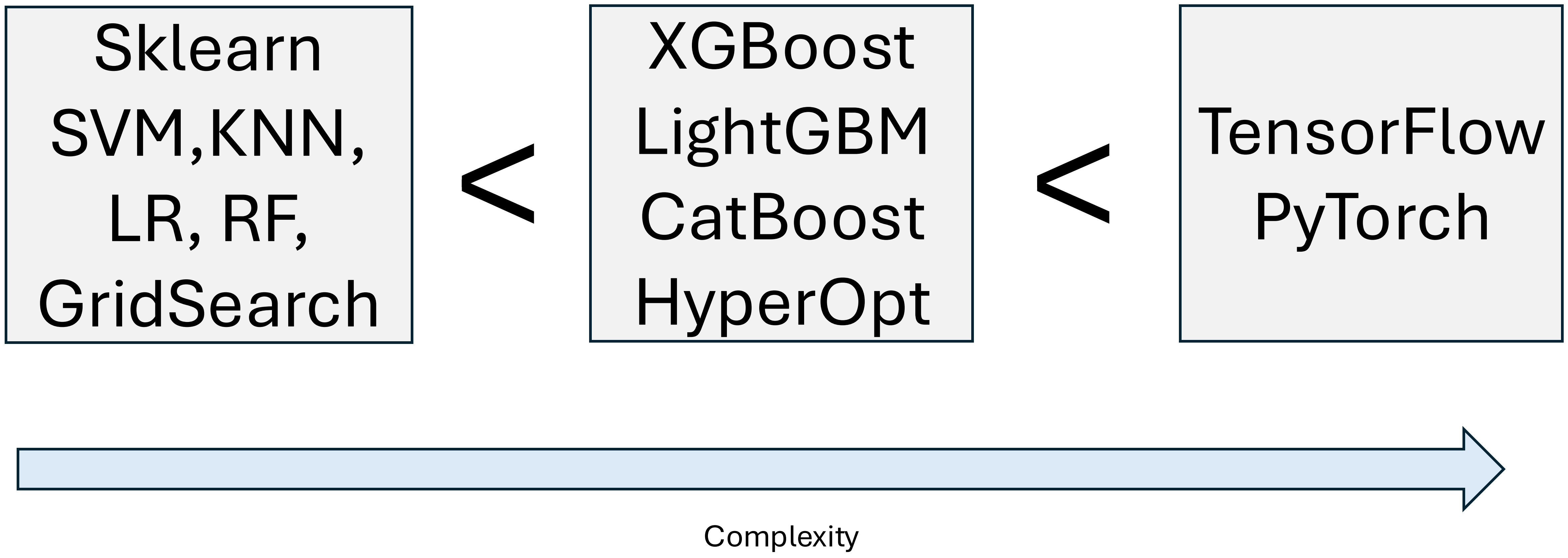
Markus Müller

Swiss Institute of Bioinformatics

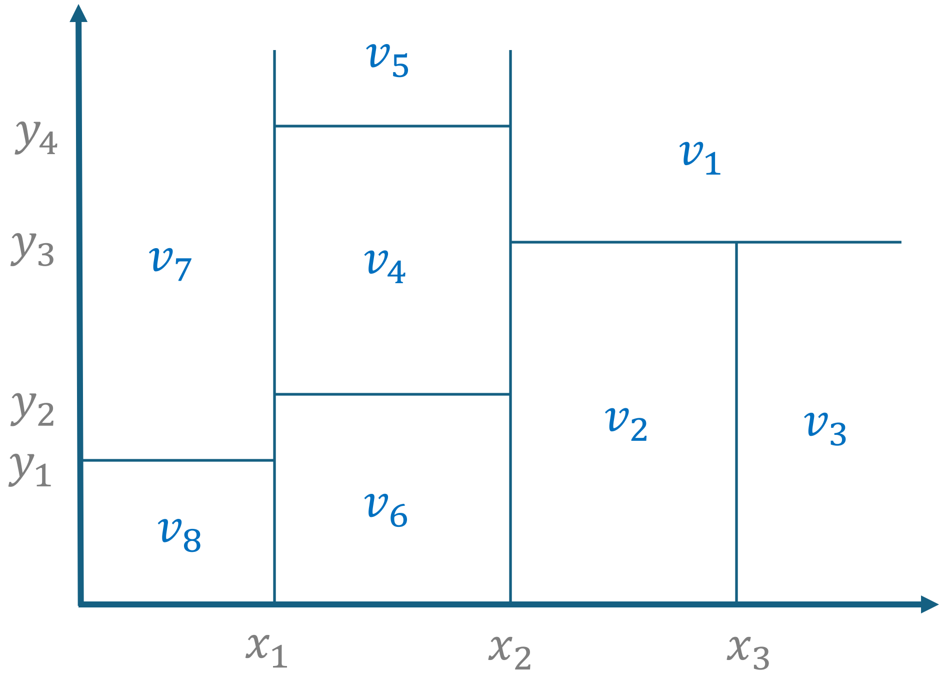
Outline

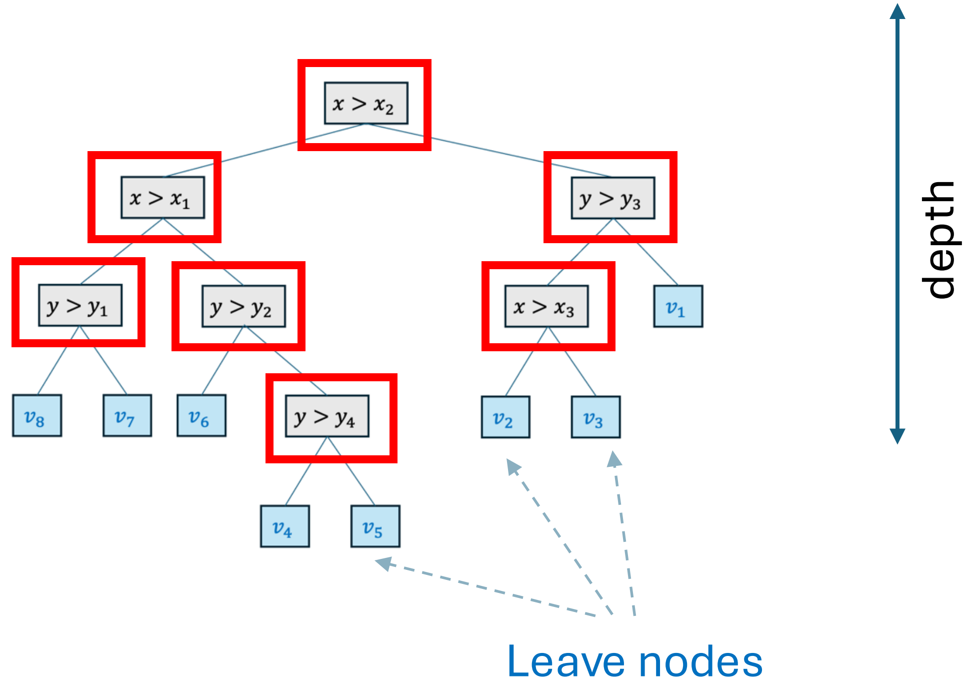
- Intermediate machine learning (ML)
- Trees
- Bagging and boosting
- Classification and regression with XGBoost
- Hyperparameter optimization with Hyperopt
- Nested cross-validation
- Interpretable machine learning with LIME and SHAP

Intermediate ML



Trees


v_5
v_1
v_7
v_4
v_2
v_3
v_6
v_8
y_4
y_3
y_2
y_1
x_1
x_2
x_3


$x > x_2$
$x > x_1$
$y > y_3$
$y > y_1$
$y > y_2$
$x > x_3$
v_1
v_8
v_7
v_6
$y > y_4$
v_2
v_3
v_4
v_5
Leave nodes
depth

Markus Müller
Swiss Institute of Bioinformatics

Trees

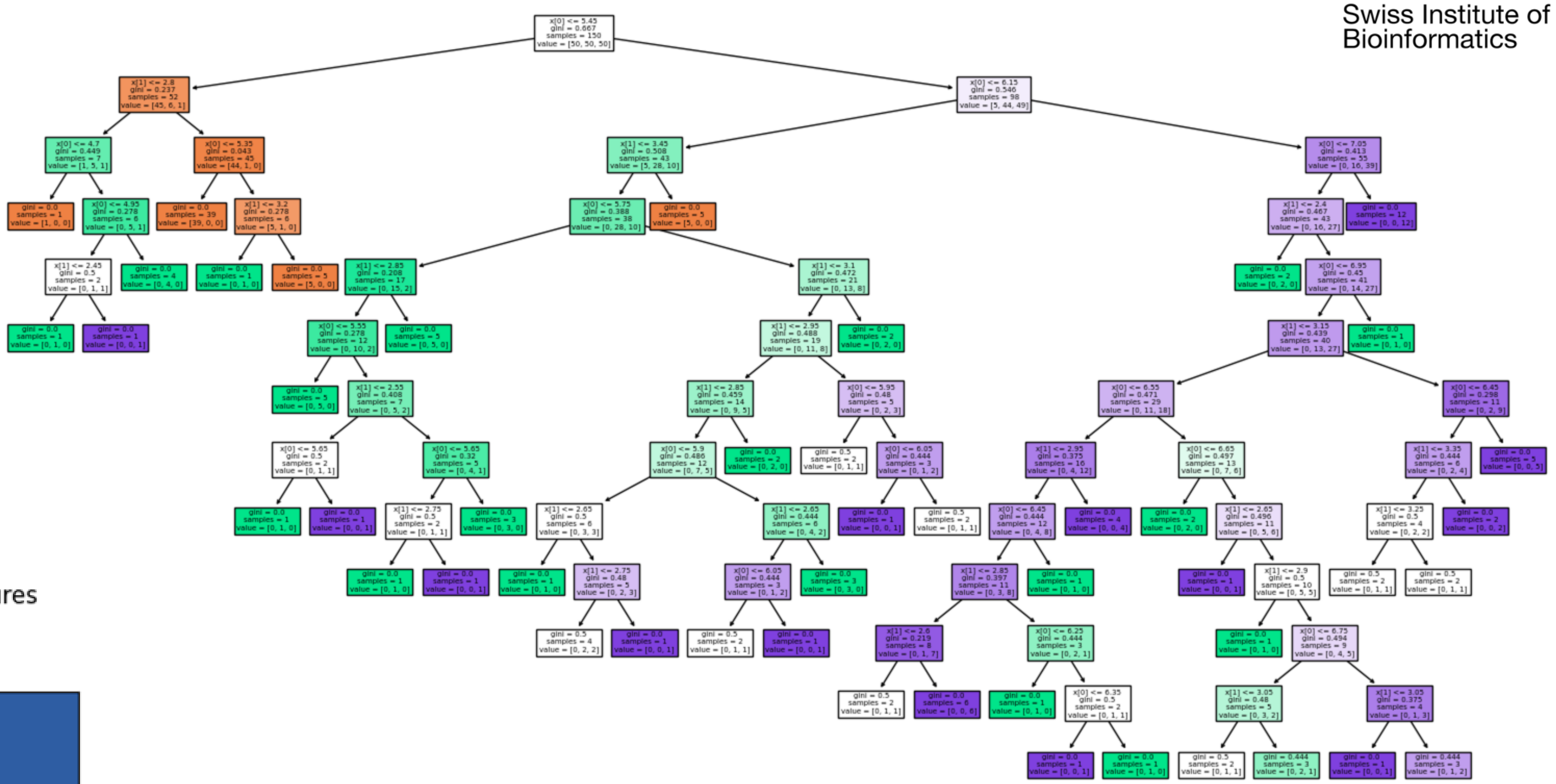
- Regression and classification trees
- Trees are universal approximators (no bias) if the size is not restricted
- Tree's are naturally able to deal with categorical features, and are not sensitive to the distribution of features values
- Parameters:
  - For each node k we have a feature F_k and threshold θ_k
  - For each leaf node l we have a weight v_l for regression, and a class label c_l for classification
- 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

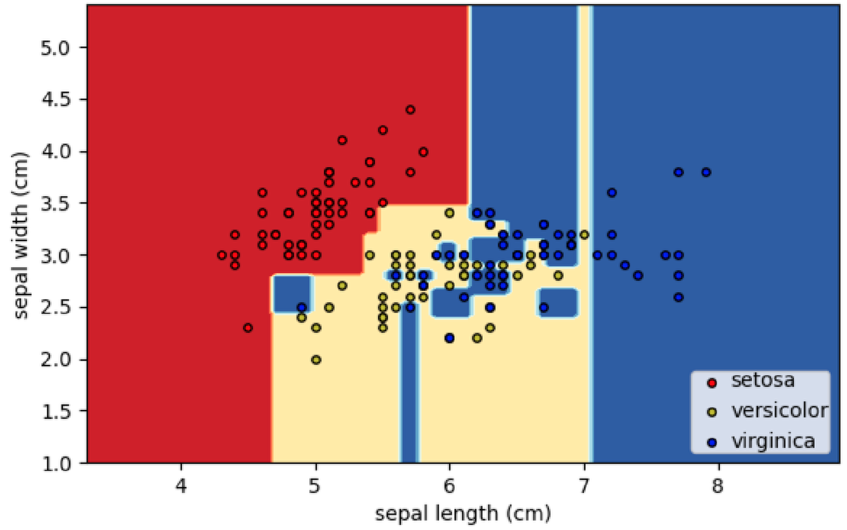
Swiss Institute of Bioinformatics

Trees

Decision tree trained on all the iris sepal length and width



Decision surface of decision trees trained on pairs of features



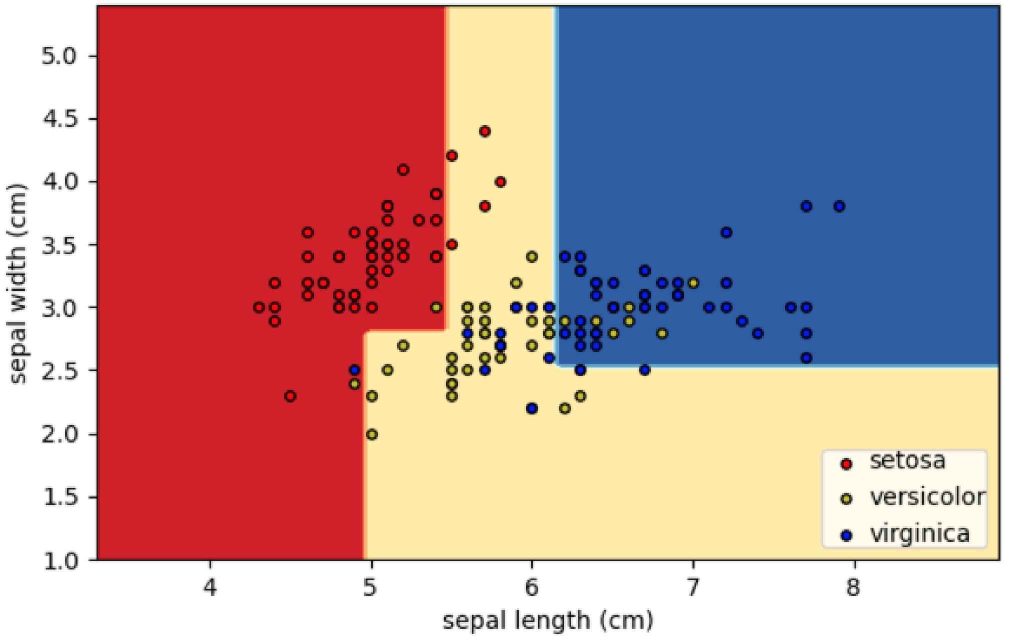
Code on sklearn page !

Markus Müller

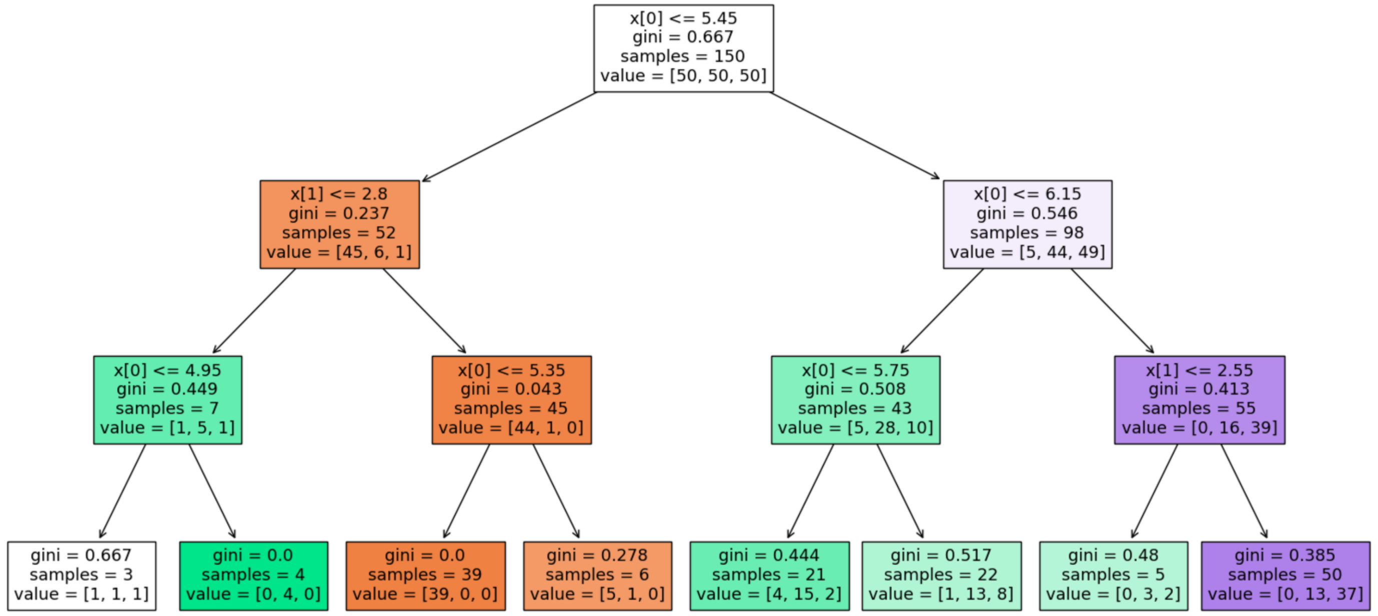
Swiss Institute of Bioinformatics

Tree hyperparameters

DecisionTreeClassifier(min_samples_leaf=3, max_depth=3, max_features=1)



DecisionTreeClassifier(min_samples_leaf=3, max_depth=3, max_features=1)



x[0] <= 5.45 gini = 0.667 samples = 150 value = [50, 50, 50]
x[1] <= 2.8 gini = 0.237 samples = 52 value = [45, 6, 1]
x[0] <= 6.15 gini = 0.546 samples = 98 value = [5, 44, 49]
x[0] <= 4.95 gini = 0.449 samples = 7 value = [1, 5, 1]
x[0] <= 5.35 gini = 0.043 samples = 45 value = [44, 1, 0]
x[0] <= 5.75 gini = 0.508 samples = 43 value = [5, 28, 10]
x[1] <= 2.55 gini = 0.413 samples = 55 value = [0, 16, 39]
gini = 0.667 samples = 3 value = [1, 1, 1]
gini = 0.0 samples = 4 value = [0, 4, 0]
gini = 0.0 samples = 39 value = [39, 0, 0]
gini = 0.278 samples = 6 value = [5, 1, 0]
gini = 0.444 samples = 21 value = [4, 15, 2]
gini = 0.517 samples = 22 value = [1, 13, 8]
gini = 0.48 samples = 5 value = [0, 3, 2]
gini = 0.385 samples = 50 value = [0, 13, 37]

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 noise in the training data (overfitting). The variance becomes smaller if we increase the size of your training data.

Bias and Variance

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

More flexible models, boosting

More data, bagging

Bagging and Boosting

- Obtain a good classifier by combining many basic, weak classifiers with limited amount of data
- Weak classifiers have either a high bias and/or high variance
- Weak classifiers are for example linear classifiers (high bias for non-linear problems) or tree classifiers (high variance without pruning)
- Weak classifiers have to be better than random labels

Bagging

- Bootstrap data items, train a base model on bootstrapped data, repeat N-times. Average predictions for regression or take a majority vote for classification.
- Reduces variance but not bias
- Bootstrapped models are correlated, which is not optimal. In the Random Forest method, features and data items are sampled in each iteration, which reduces the correlation of base models.

Boosting

- In each iteration, the boosting method tries to correct the error of the previous iteration by focusing on the wrongly predicted data items.
- The current base model corrects those misclassified data points and is added to the previous models with a weight that is inversely proportional to its error.
- The boosting method can correct the bias of the base classifier.
- AdaBoost implements the boosting algorithm for classification and regression.
- Gradient boosting is a more general framework that includes boosting algorithms.

XGBoost: a scalable and flexible gradient boosting system

- Extreme gradient boosting system
- Introduced by Chen & Guestrin in 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, and pruning, …
- XGBoost provides robust and complementary results when compared to neural network models on tabular data

XGBoost

- Output after t iterations: $\hat{y}_i^t = \sum_{k=1}^{t} f_k(\boldsymbol{x}_i)$;
- $f_k(\boldsymbol{x}_i) =$ base classifier at iteration k (here we assume it's a tree)

- Boosting algo: fit $f_t(\boldsymbol{x}_i)$ 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 $l(\hat{y}_i, y_i)$
  - regularization term $\Omega(f)$. For trees $\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{i=1}^{T} v_i^2$
    - T: number of leaves; $\boldsymbol{v}$: values at leaves

XGBoost

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

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

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

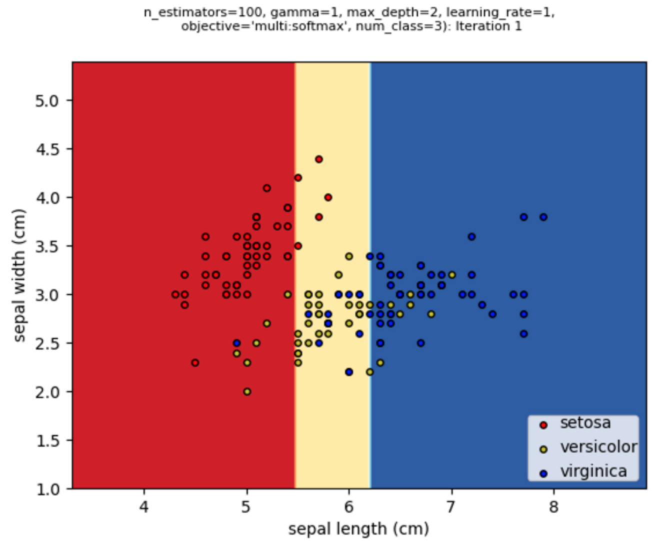
XGBoost - square loss

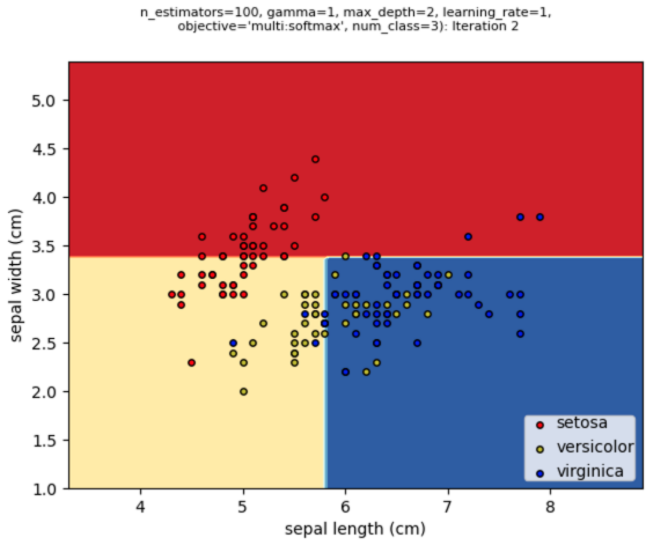
- $l(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$

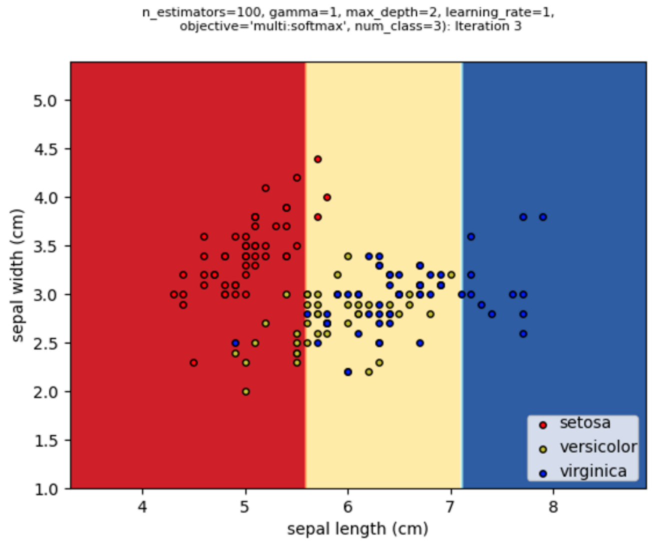
- Gradient boosting: $g_i = \left(\hat{y}_i^{t-1} - y_i\right); h_i = 1$

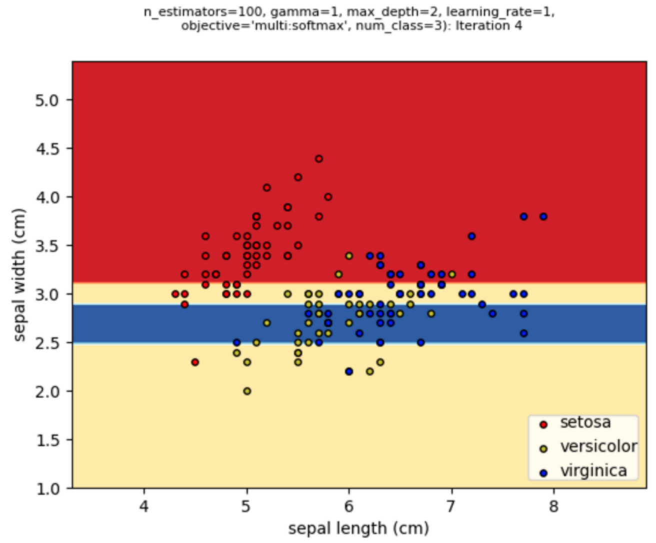
  - $$\tilde{\mathcal{L}}^t\left(\hat{y}_i^{t-1}, f_t\right) = \sum_{i=1}^{N} \Big\{ \underbrace{\left(\hat{y}_i^{t-1} - y_i\right) f_t(\boldsymbol{x}_i) +}_{\begin{array}{l} \left(\hat{y}_i^{t-1} - y_i\right) > 0 : f_t(\boldsymbol{x}_i) < 0 \\ \left(\hat{y}_i^{t-1} - y_i\right) \sim 0 : f_t(\boldsymbol{x}_i) \sim 0 \\ \left(\hat{y}_i^{t-1} - y_i\right) < 0 : f_t(\boldsymbol{x}_i) > 0 \end{array}} \underbrace{\frac{1}{2} f_t^{\,2}(\boldsymbol{x}_i) \Big\} + \Omega(f_t)}_{\text{Regularization term}}$$

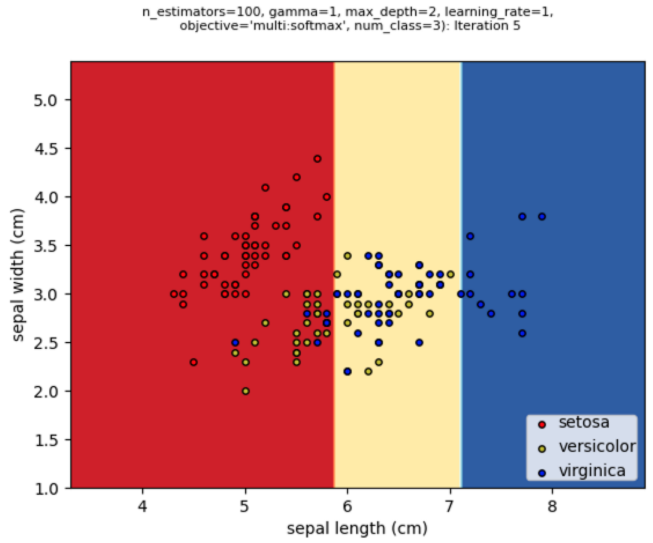
XGBoost

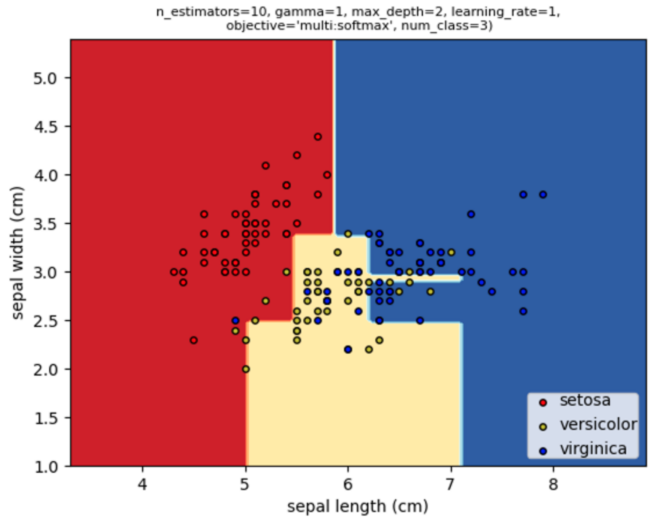

n_estimators=100, gamma=1, max_depth=2, learning_rate=1, objective='multi:softmax', num_class=3): Iteration 1
sepal width (cm)
sepal length (cm)
setosa
versicolor
virginica


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

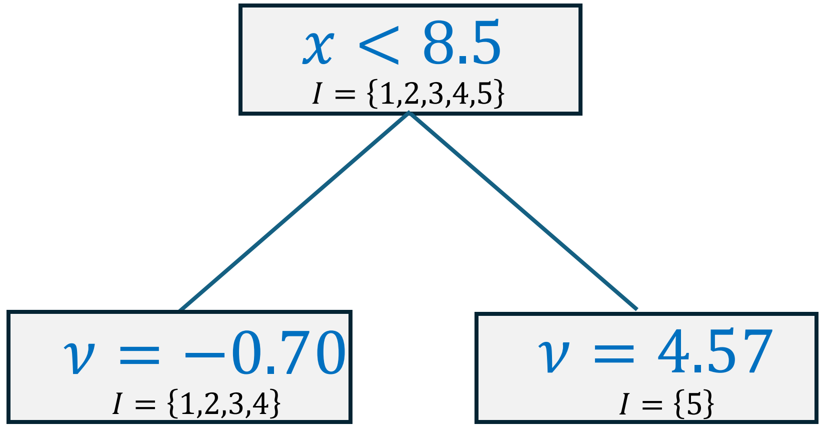
XGBoost

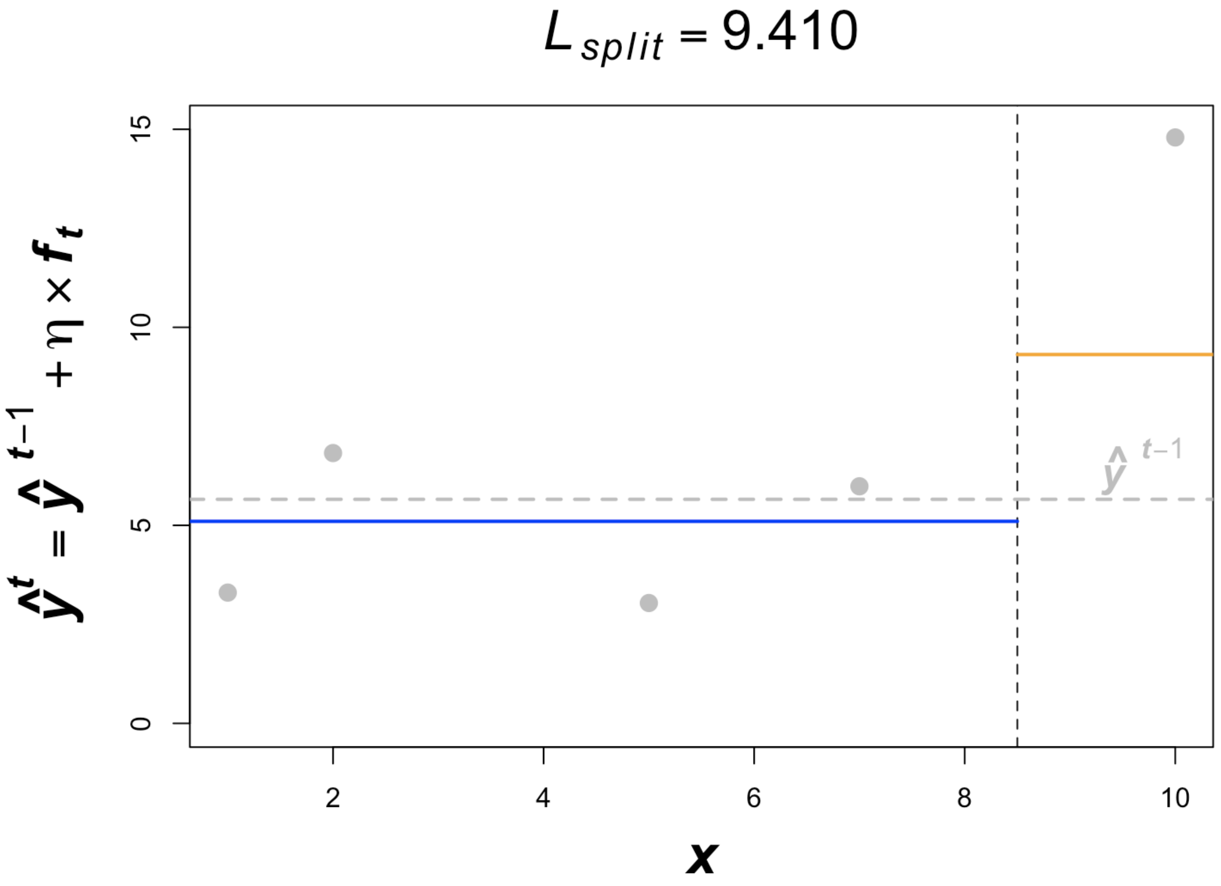
- How does XGBoost find the optimal tree?
- Let's assume the tree f_t is known appart from the leaf values $\boldsymbol{v}$
- Define $I_j = \{i|leaf(\boldsymbol{x}_i) = j\}$, i.e. all points $\boldsymbol{x}_i$ from the training set that belong to leaf j
- Then the optimal leaf values are (for square loss):
  - $v_j = -\frac{\sum_{i\in I_j}(\hat{y}_i^{t-1}-y_i)}{|I_j|+\lambda}$
  - Then the loss reduction introduced by a split can then be calculated exactly:

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

- 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.

XGBoost gbtree algo





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

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.

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 instance hessians 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 weights
- *alpha*: L1 regularization term on weights
- *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
- *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

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, ..

Hyperparameter (HP) 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 HP optimization and training. Allocating more cycles for HP optimization can be more rewarding.
- HP optimization is implemented as an outer loop over the CV loop

Hyperparameter optimization and CV

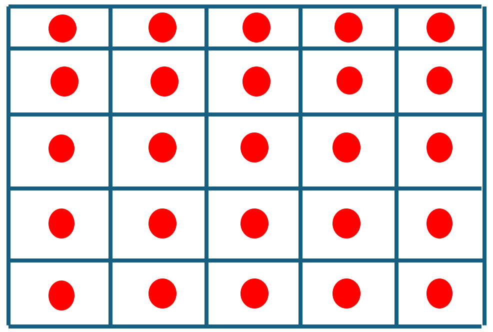
Hyperparameter space Φ
Dataset $\boldsymbol{X}$, Labels $\boldsymbol{y}$
Number of iterations M
Number of CV folds F

Hyperparameter search loop

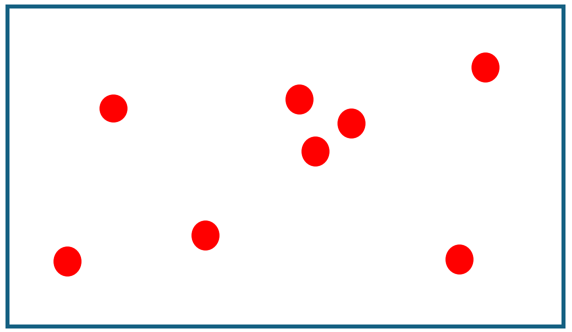
CV loop

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

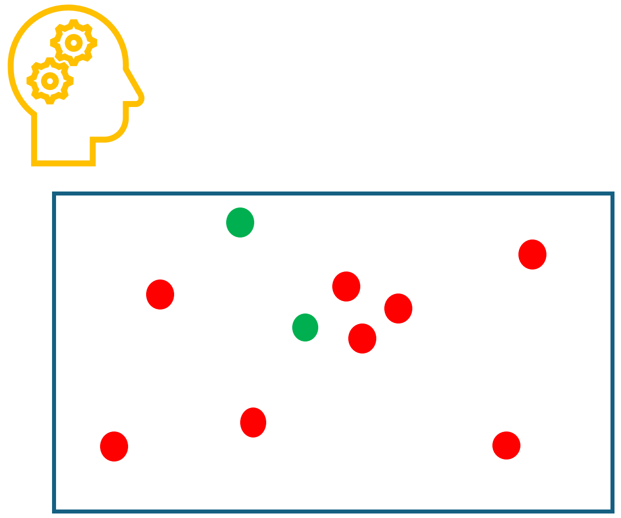
HP searches



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
Resolution limited by
CPU time



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

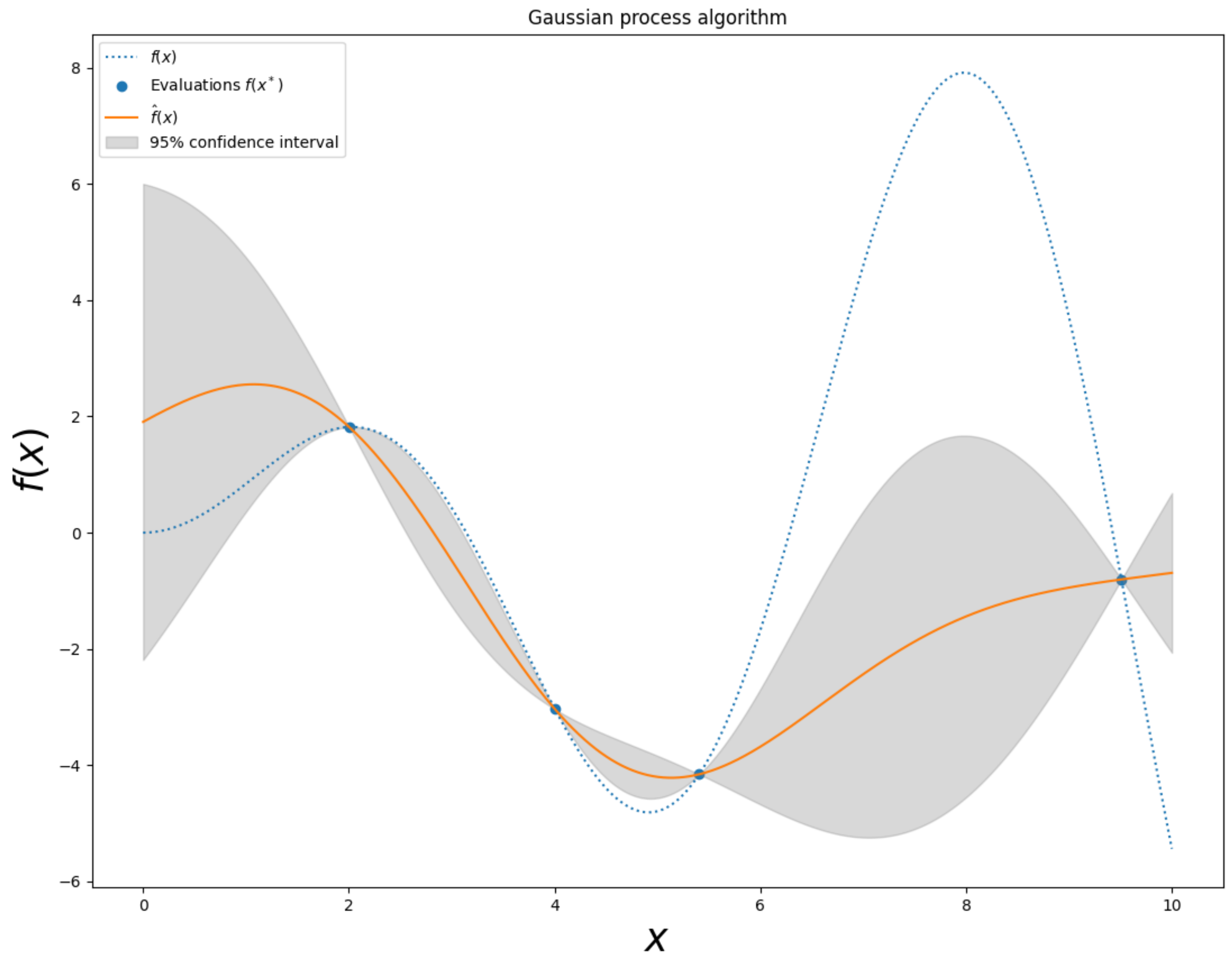
Markus Müller

Swiss Institute of Bioinformatics

Hyperopt

- Random algorithms find optimal solutions faster and more accurately than grid searches
- Hyperopt also offers two other model-based search algorithms for python:
  - Random search
  - Tree-structured Parzen window approach (TPE)
- Hyperopt supports nested parameter spaces
  - Hyperopt samples HPs from defined probability distributions or from predefined values
  - The HP space can have a hierarchical structure reflecting dependenies between HPs
- Hyperopt supports warm start using previous results

Hyperopt's bayesian HP exploration



Markus Müller

Swiss Institute of Bioinformatics

Hyperopt's optimization score: expected improvement

EI: Expected improvement for hyperparameter values x
$y = f(\boldsymbol{x})$: objective function to minimize (e.g. loss function L)
y^*: lower percentile of y

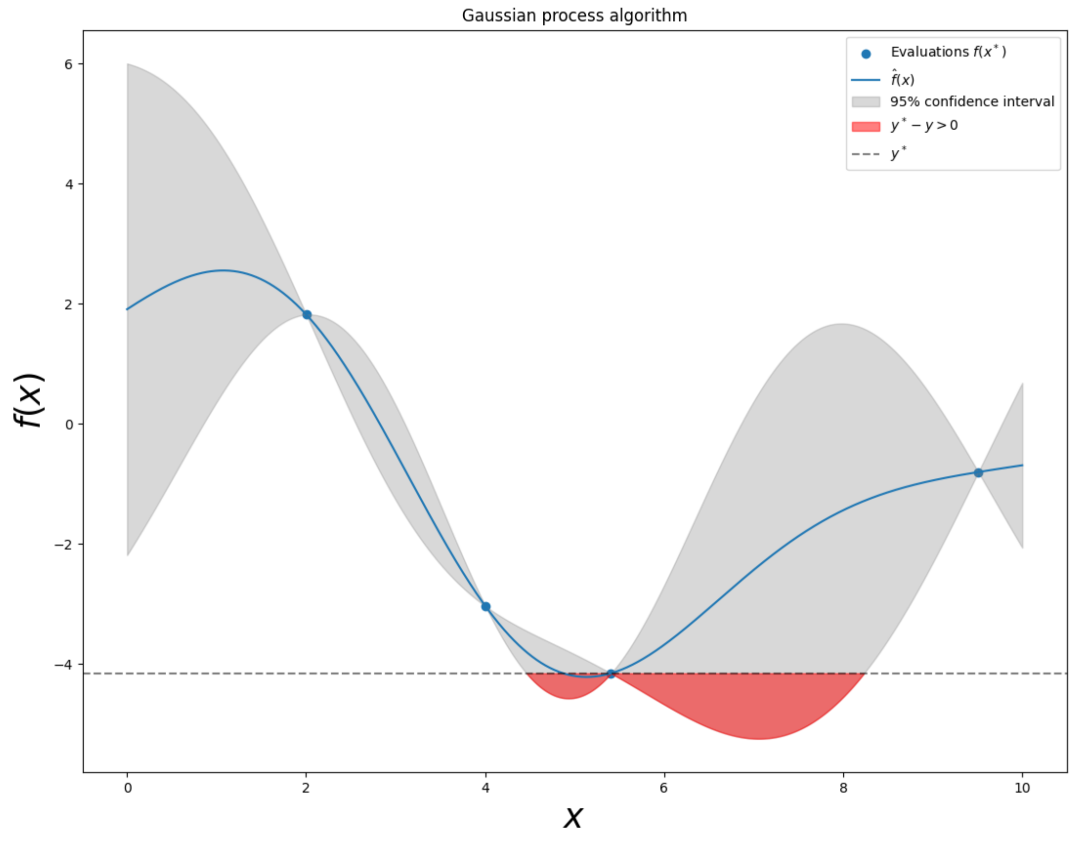
$$EI_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y)\, p_M(y|x) dy$$

EI is high where $\hat{f}(x) = \int_{-\infty}^{\infty} y\, p_M(y|x) dy$ is low and/or

where the variance $\sigma^2(x) = \int_{-\infty}^{\infty} \left(y - \hat{f}(x)\right)^2 p_M(y|x) dy$

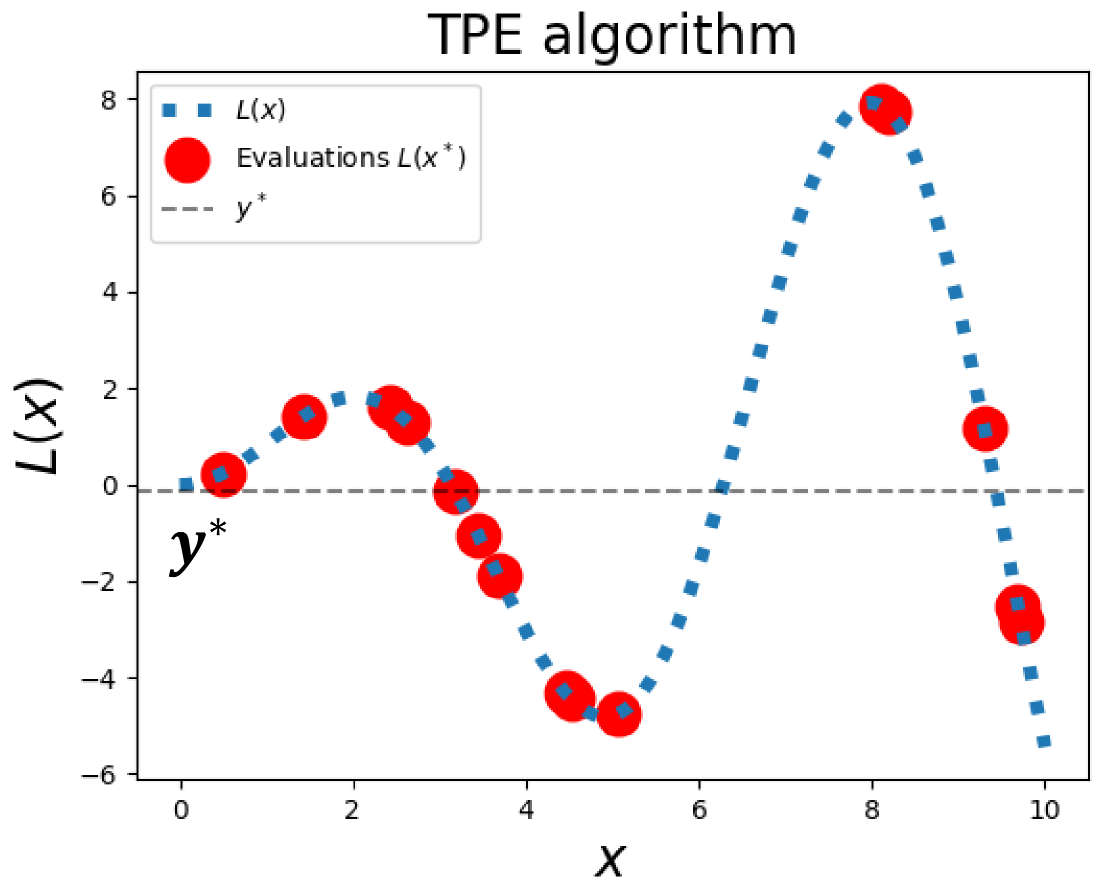
is high.

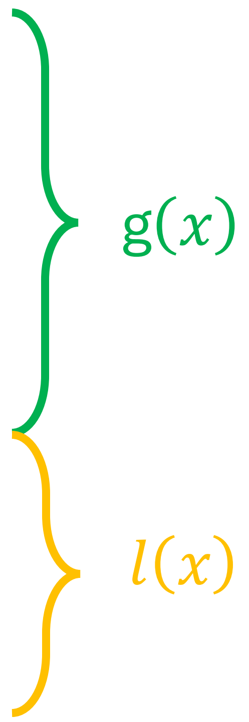
=> We need to estimate $p_M(y|x)$

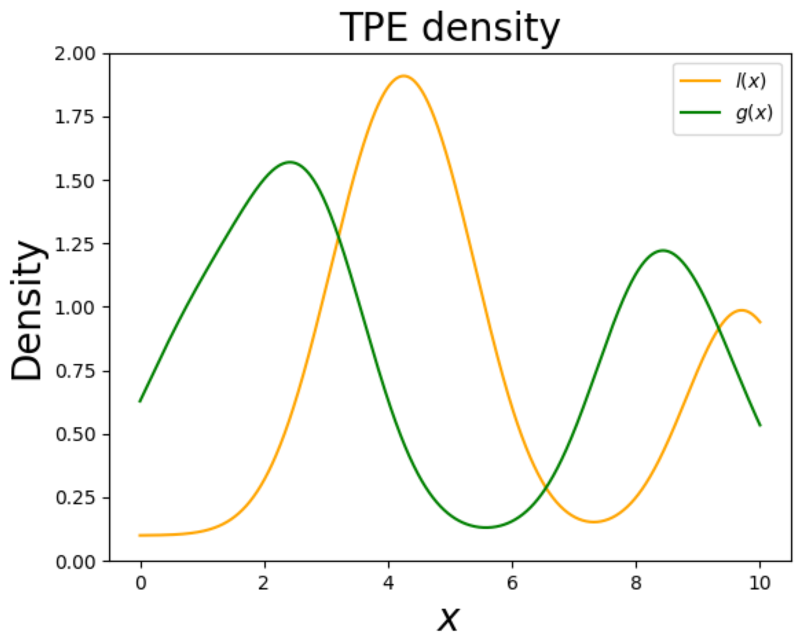


Markus Müller
Swiss Institute of Bioinformatics

Hyperopt's TPE algorithm







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

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

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

TPE: Tree structured Parzen estimator

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

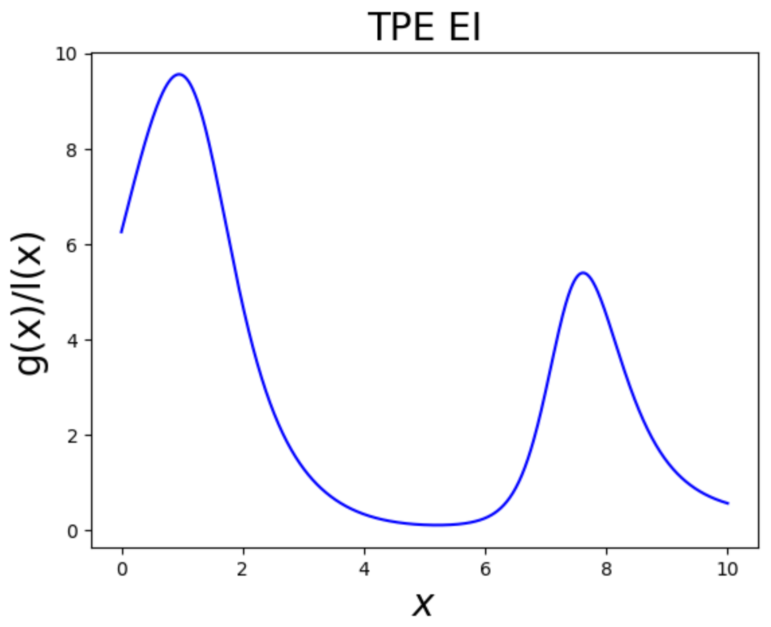
Markus Müller
Swiss Institute of Bioinformatics

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.

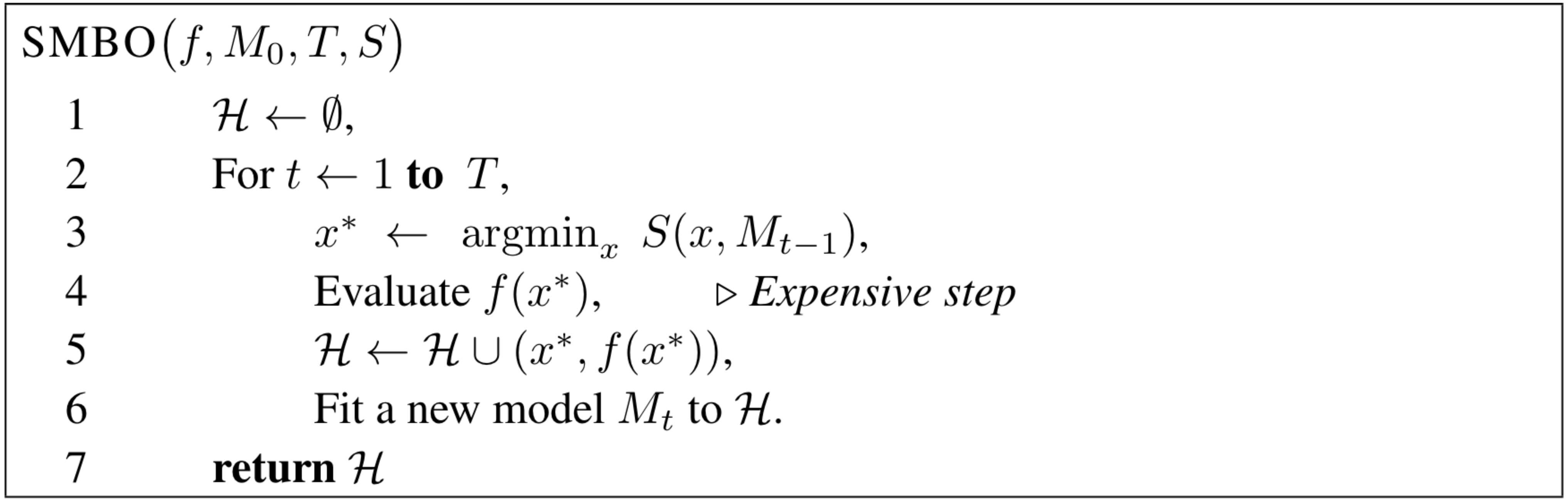


Markus Müller
Swiss Institute of Bioinformatics

Hyperopt’s model-based search

$M_t(\boldsymbol{x}) \Rightarrow p_M(y|x)$

$S(x, M) = -EI_{y^*}(x)$



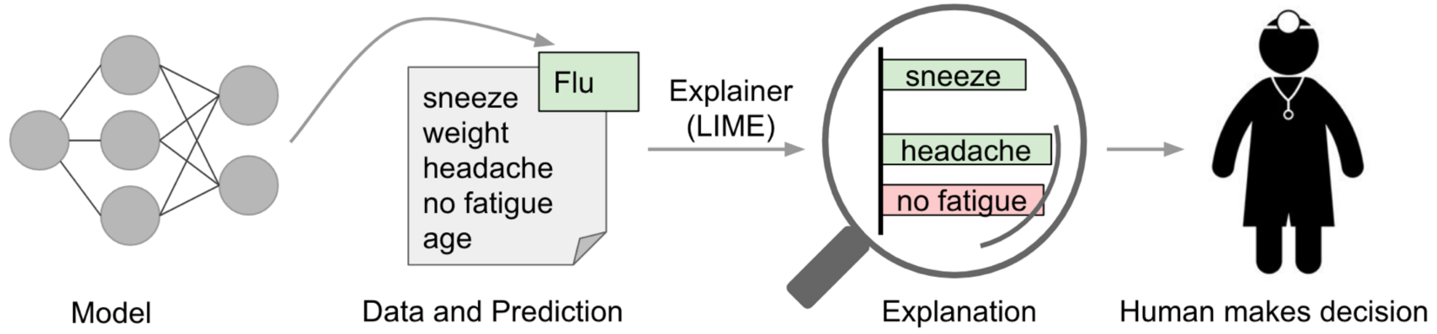
```
SMBO$(f, M_0, T, S)$
1   $\mathcal{H} \leftarrow \emptyset$,
2   For $t \leftarrow 1$ **to** $T$,
3       $x^* \leftarrow \text{argmin}_x \; S(x, M_{t-1})$,
4       Evaluate $f(x^*)$,          $\triangleright$ *Expensive step*
5       $\mathcal{H} \leftarrow \mathcal{H} \cup (x^*, f(x^*))$,
6       Fit a new model $M_t$ to $\mathcal{H}$.
7   **return** $\mathcal{H}$
```

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

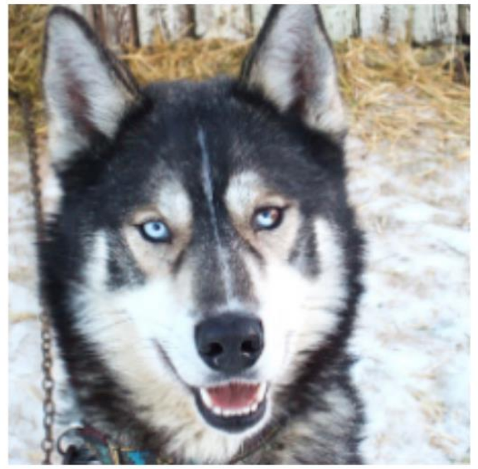
Interpretable Machine Learning (IML)

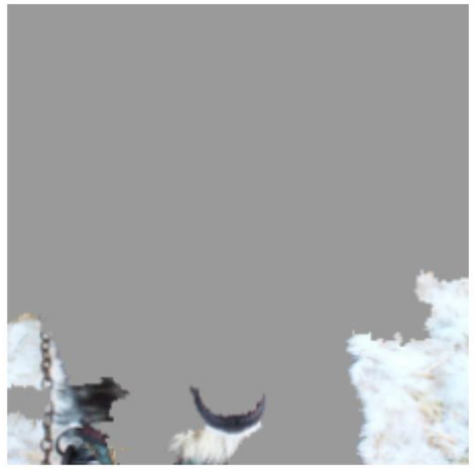
- ML tools are optimized for metrics such as accuracy, which can lead to accurate, but wrong models.
- Many models are black boxes, i.e. they do not reveal in a human readable form how they make their decision.
- In order to understand a model, we need to approximate the model's decision with a simple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

Interpretable Machine Learning (IML)



Ribeiro et al., ACM SIGKDD, 2016


(a) Husky classified as wolf


(b) Explanation

Interpretable Machine Learning (IML)

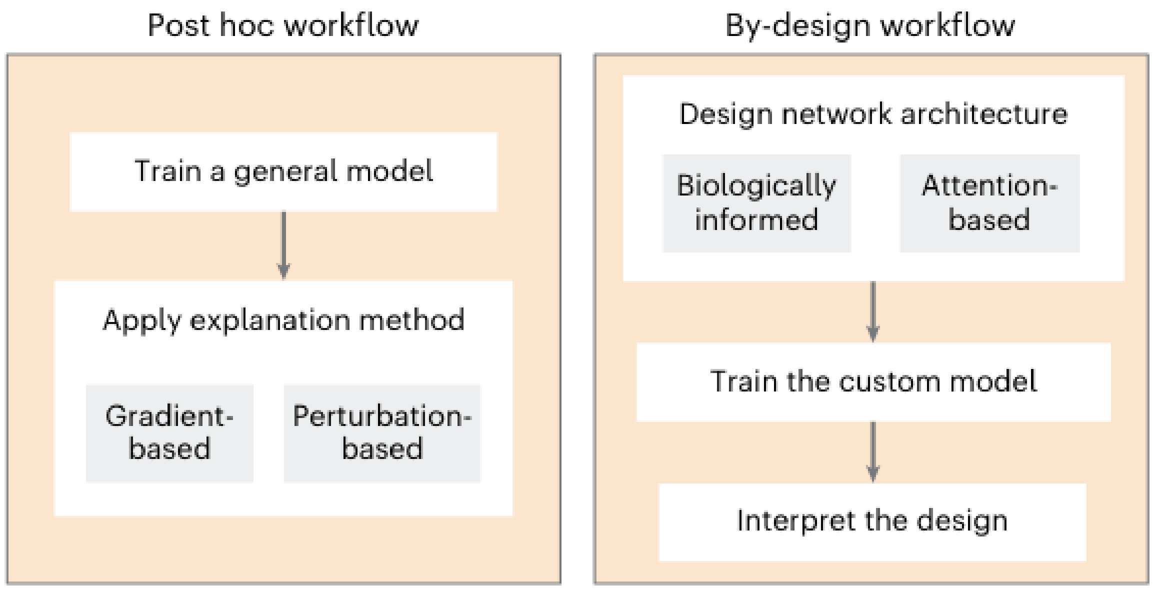


Fig. 1 | The two main IML approaches used to explain prediction models are post hoc explanations and by-design explanations. Each approach has its canonical workflows and popular types of IML methods: post hoc explanations are model agnostic and are applied after a model is trained, while by-design explanations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

Chen et al, Nature Methods, 2024

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.

IML

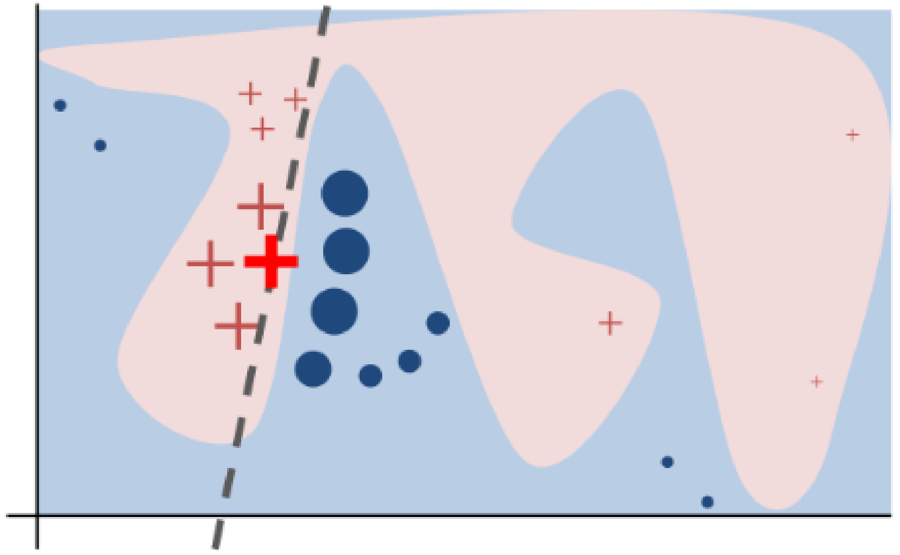
- Local interpretation:
  - An interpretation is calculated for a single data vector $\boldsymbol{x}$.
  - A local interpretation tells you, how a specific data vector $\boldsymbol{x}$ was predicted by the ML model. It tells you which features are predominantly involved in the prediction.

- 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

LIME: local interpretation



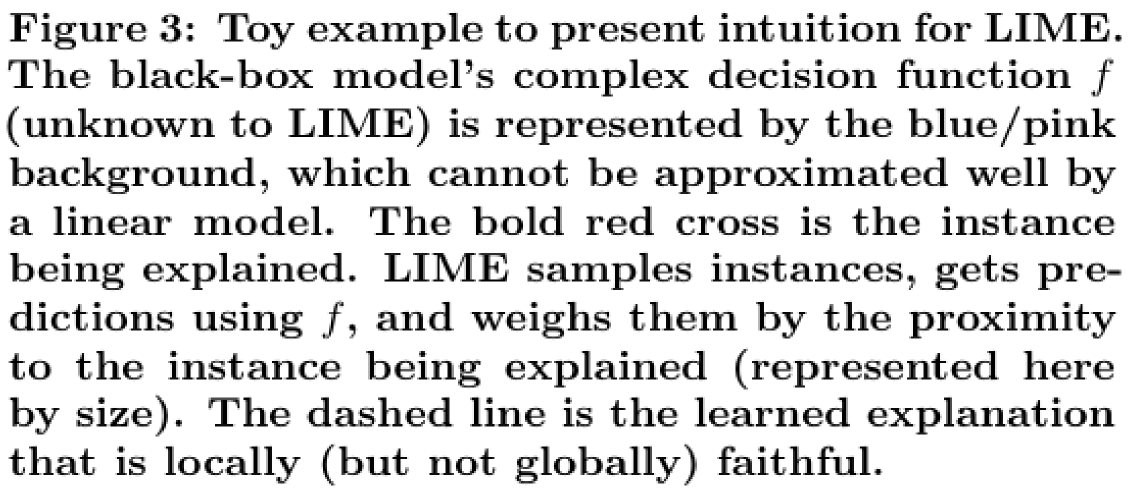


Figure 3: Toy example to present intuition for LIME. The black-box model's complex decision function f (unknown to LIME) is represented by the blue/pink background, which cannot be approximated well by a linear model. The bold red cross is the instance being explained. LIME samples instances, gets predictions 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.

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

Shapley Values

- Loyd Shapley (1951): How do you evaluate the importance of a single player in a team of n players $\mathcal{N}$?
- 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:

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

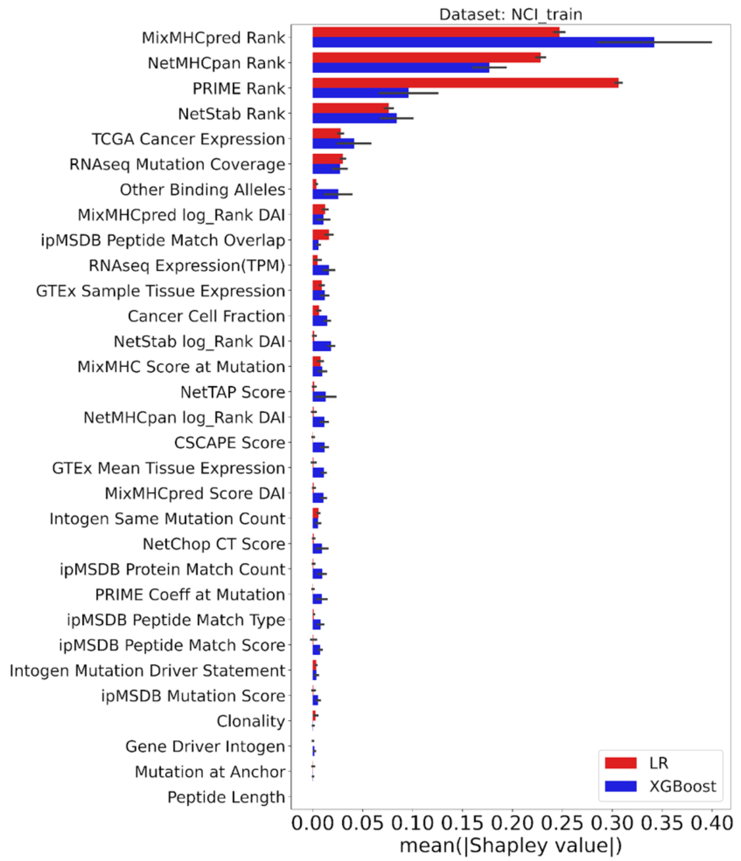
- $E_x\left(f(\boldsymbol{x})\right) + \sum_{i=1}^{n} \varphi_i(\boldsymbol{x}) = f(\boldsymbol{x})$

Shapley Values for ML

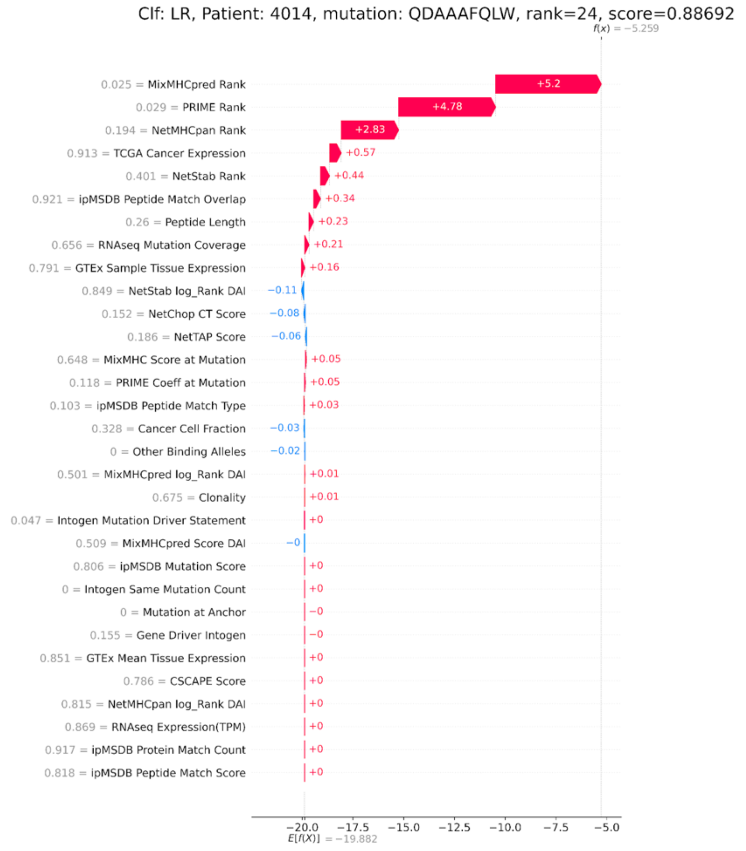
- Let's assume we have a trained ML model, which predicts the output $f(\boldsymbol{x}_j)$ of a vector $\boldsymbol{x}_j = (x_{j1},..,x_{jn}) \in \mathbb{R}^n$ from a dataset $\mathcal{D} = \{\boldsymbol{x}_j\}, 1 \leq j \leq N$
- $S \subset \mathcal{N} = \{1,..,n\}$. How do we define $f(\boldsymbol{x}, S)$?
  - $f(\boldsymbol{x}, S)$: keeps values x_i for $i \in S$ and replaces values x_i for $i \notin S$ by some backgound values
  - In the Shap package, you can define this background with the *masker* parameter.
  - You can also group the features before masking (e.g. if features are strongly dependent)

Markus Müller
SIB Swiss Institute of Bioinformatics

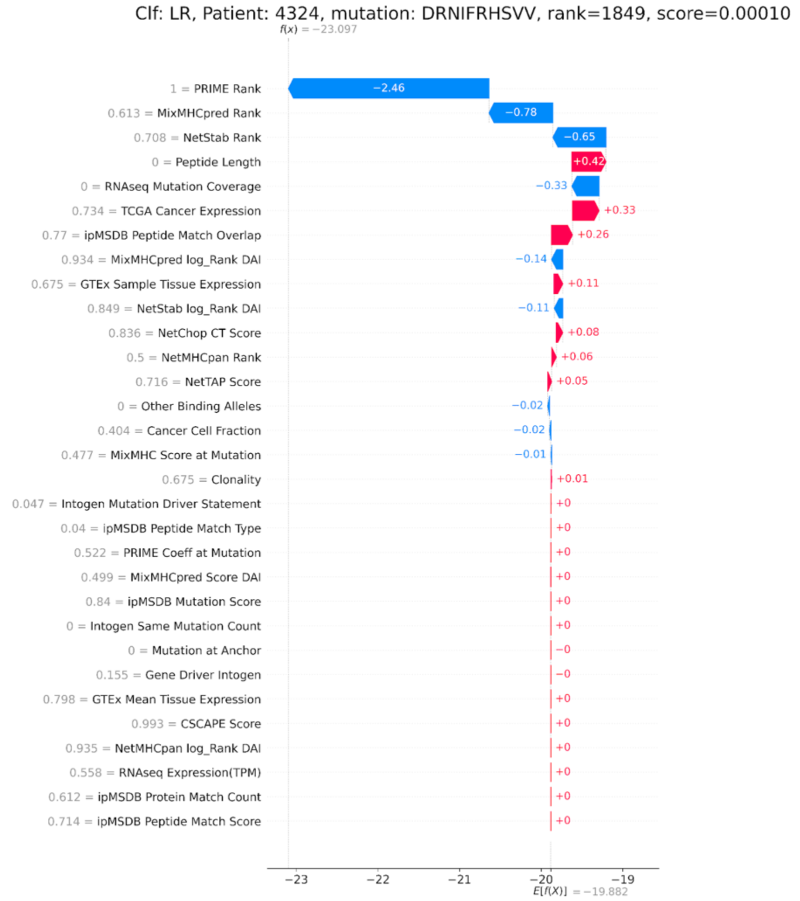
IML with Shapley values



Good prediction



Bad prediction


Clf: LR, Patient: 4324, mutation: DRNIFRHSVV, rank=1849, score=0.00010
f(x) = −23.097
1 = PRIME Rank −2.46
0.613 = MixMHCpred Rank −0.78
0.708 = NetStab Rank −0.65
0 = Peptide Length +0.42
0 = RNAseq Mutation Coverage −0.33
0.734 = TCGA Cancer Expression +0.33
0.77 = ipMSDB Peptide Match Overlap +0.26
0.934 = MixMHCpred log_Rank DAI −0.14
0.675 = GTEx Sample Tissue Expression +0.11
0.849 = NetStab log_Rank DAI −0.11
0.836 = NetChop CT Score +0.08
0.5 = NetMHCpan Rank +0.06
0.716 = NetTAP Score +0.05
0 = Other Binding Alleles −0.02
0.404 = Cancer Cell Fraction −0.02
0.477 = MixMHC Score at Mutation −0.01
0.675 = Clonality +0.01
0.047 = Intogen Mutation Driver Statement +0
0.04 = ipMSDB Peptide Match Type +0
0.522 = PRIME Coeff at Mutation +0
0.499 = MixMHCpred Score DAI +0
0.84 = ipMSDB Mutation Score +0
0 = Intogen Same Mutation Count +0
0 = Mutation at Anchor −0
0.155 = Gene Driver Intogen −0
0.798 = GTEx Mean Tissue Expression +0
0.993 = CSCAPE Score +0
0.935 = NetMHCpan log_Rank DAI +0
0.558 = RNAseq Expression(TPM) +0
0.612 = ipMSDB Protein Match Count +0
0.714 = ipMSDB Peptide Match Score +0
E[f(X)] = −19.882