A Brief Introduction to Random Forests

E. Renner, C. Bracco, F.M Velotti

ML-coffee, 19-11-21

Overview and Outline

A Random Forest (RF) is an **ensemble classifier/regressor based on binary decision trees.**

Decision trees **exhibit low bias but high variance** (overfit to training data, noise). **RF** use a large number of decorrelated trees to reduce prediction variance

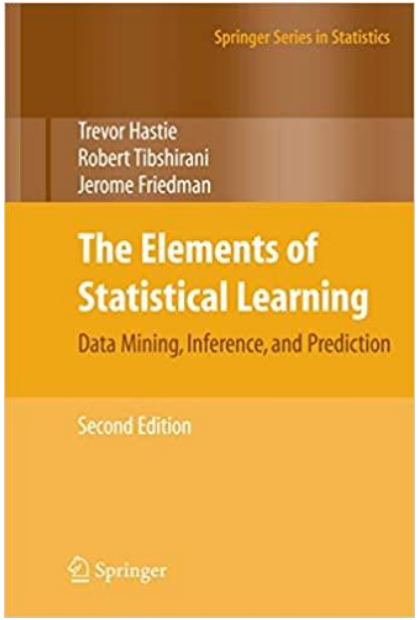
1. Decision Trees
2. From single Decision Trees to Ensemble Methods & RF

RF can be used for **classification** and **regression** & are very **robust to data pre-processing and hyper parameter tuning**

3. RF Regression and Classification: Hyperparameters, Pros & Cons
4. Example PSB Injection Surrogate Model
5. Outlook: Other ensemble predictors

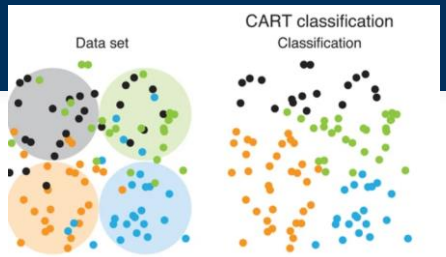
Recommended Literature

https://web.stanford.edu/~hastie/ElemStatLearn/printings/ESLII_print12_toc.pdf



Reminder: Decision Trees

Decision Trees



- Belongs to the simplest classifiers
- Can be used both for classification and for modelling
- Method generates a binary tree (two possibilities per decision)
- Variables are selected by maximizing the information which can be obtained by division of the training data

- Different algorithms, e.g.:
  - **CART** (**C**lassification **a**nd **R**egression **T**rees) (discussed here)
  - ID3 (Iterative Dichotomiser 3)
  - C4.5 (successor of ID3)
  - MARS (Multivariate adaptive regression splines)

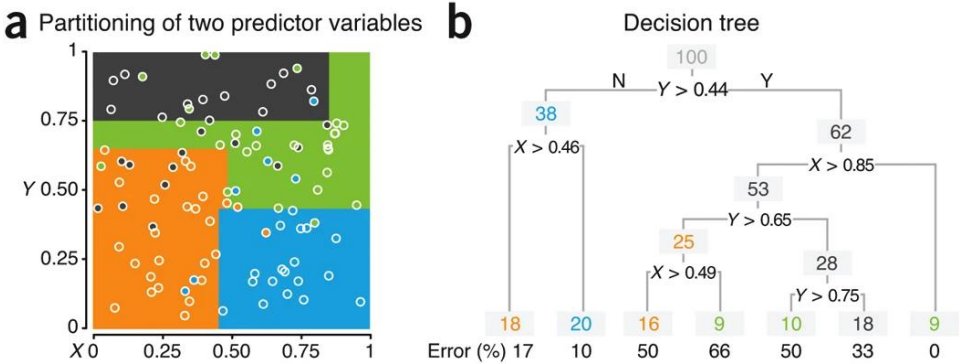
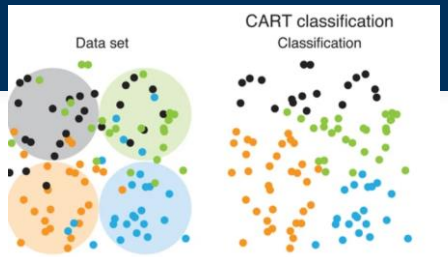


Fig adapted from Nature Methods: Classification and regression trees (08/2017)

CART Example (Classification)



1. Find in the given (partial) dataset the **variable** and its corresponding **decision threshold** which exhibits the **best discrimination**
2. Separate the (partial) dataset based on the selected variable + threshold into two groups
3. For each partial dataset, go to step 1 until a **predefined stop criterion** is met

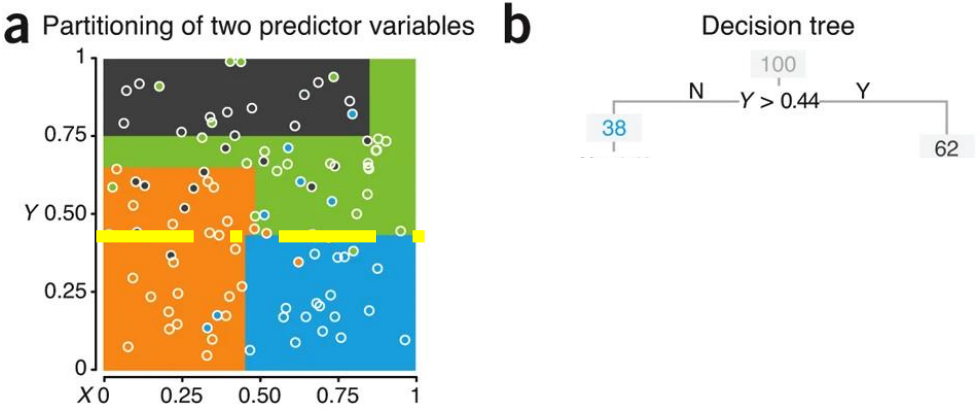
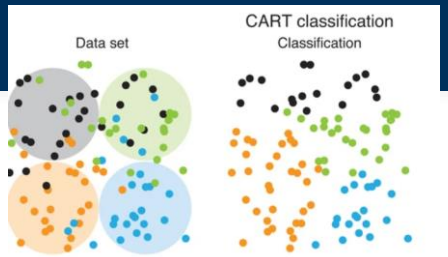


Fig adapted from Nature Methods: Classification and regression trees (08/2017)

CART Example (Classification)



1. Find in the given (partial) dataset the **variable** and its corresponding **decision threshold** which exhibits the **best discrimination**
2. Separate the (partial) dataset based on the selected variable + threshold into two groups
3. For each partial dataset, go to step 1 until a **predefined stop criterion** is met

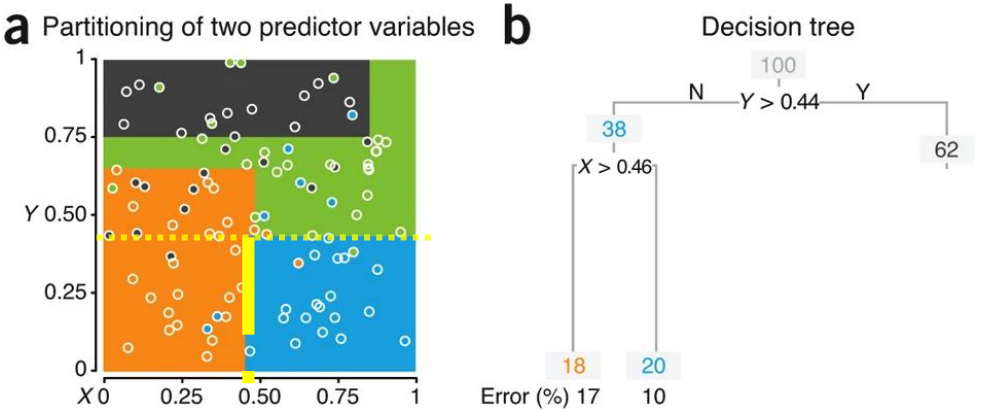
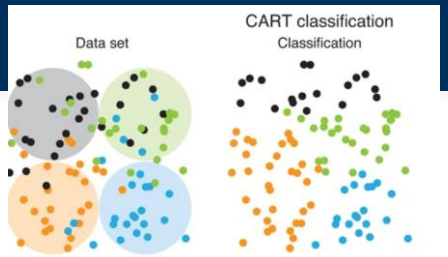


Fig adapted from Nature Methods: Classification and regression trees (08/2017)

CART Example (Classification)



1. Find in the given (partial) dataset the **variable** and its corresponding **decision threshold** which exhibits the **best discrimination**
2. Separate the (partial) dataset based on the selected variable + threshold into two groups
3. For each partial dataset, go to step 1 until a **predefined stop criterion** is met

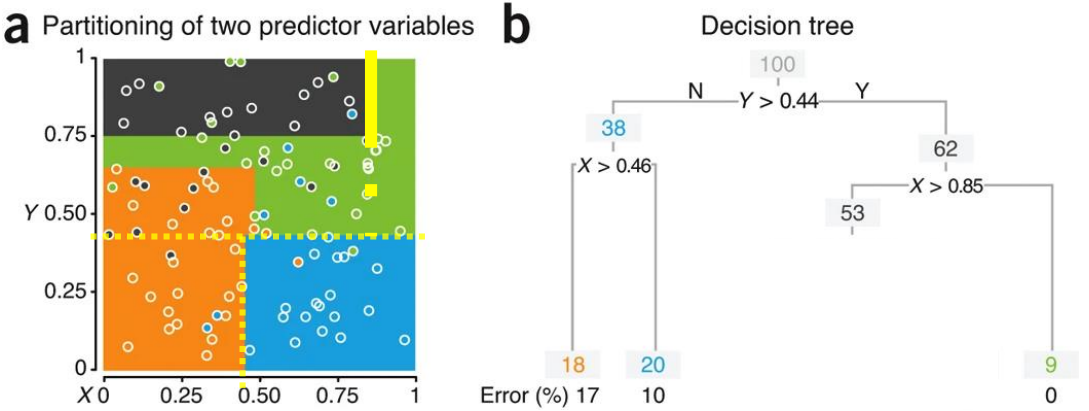
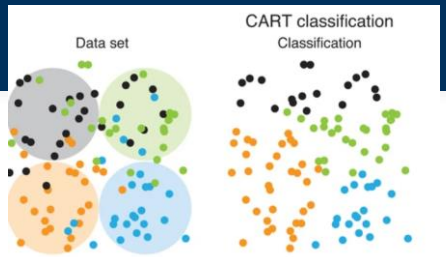


Fig adapted from Nature Methods: Classification and regression trees (08/2017)

CART Example (Classification)



1. Find in the given (partial) dataset the **variable** and its corresponding **decision threshold** which exhibits the **best discrimination**
2. Separate the (partial) dataset based on the selected variable + threshold into two groups
3. For each partial dataset, go to step 1 until a **predefined stop criterion** is met

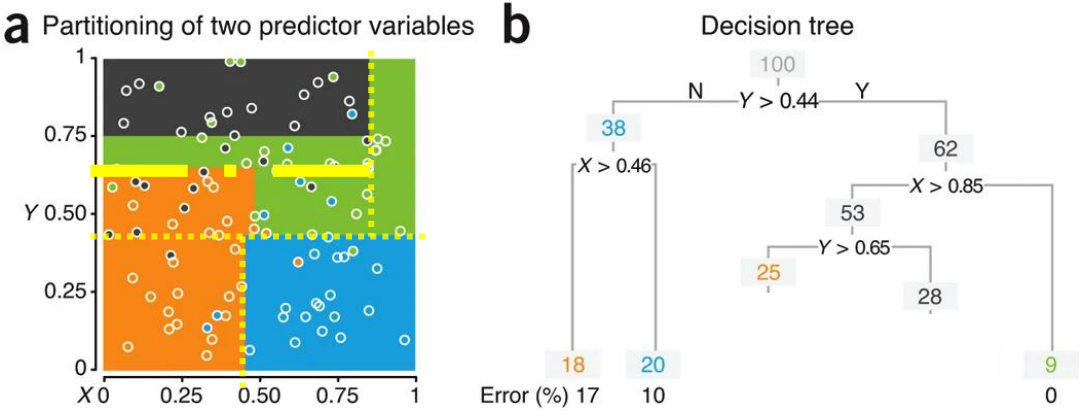
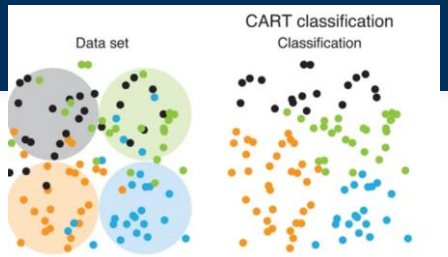


Fig adapted from Nature Methods: Classification and regression trees (08/2017)

CART Example (Classification)



1. Find in the given (partial) dataset the **variable** and its corresponding **decision threshold** which exhibits the **best discrimination**
2. Separate the (partial) dataset based on the selected variable + threshold into two groups
3. For each partial dataset, go to step 1 until a **predefined stop criterion** is met

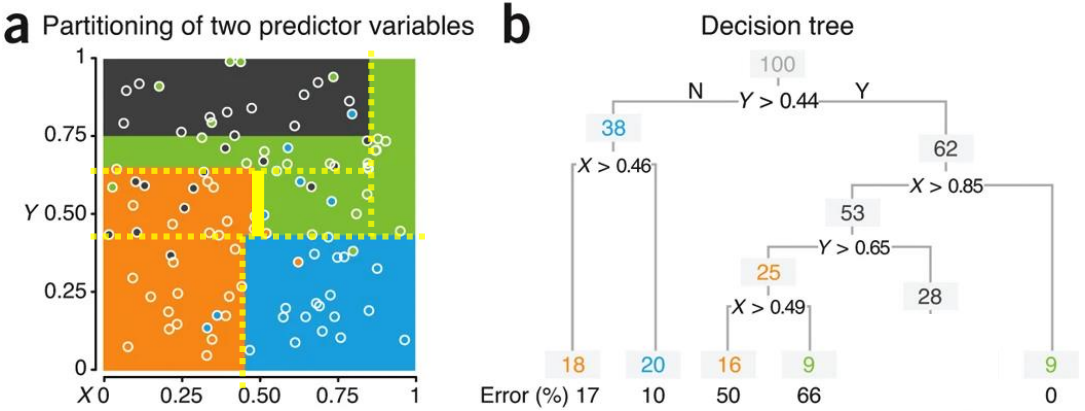
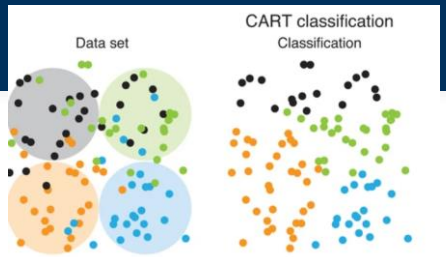


Fig adapted from [Nature Methods: Classification and regression trees](#) (08/2017)

CART Example (Classification)



1. Find in the given (partial) dataset the **variable** and its corresponding **decision threshold** which exhibits the **best discrimination**
2. Separate the (partial) dataset based on the selected variable + threshold into two groups
3. For each partial dataset, go to step 1 until a **predefined stop criterion** is met

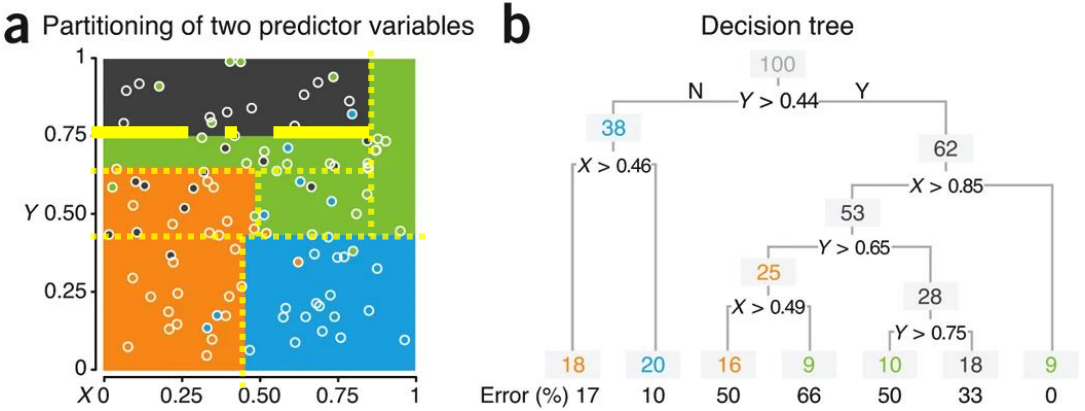
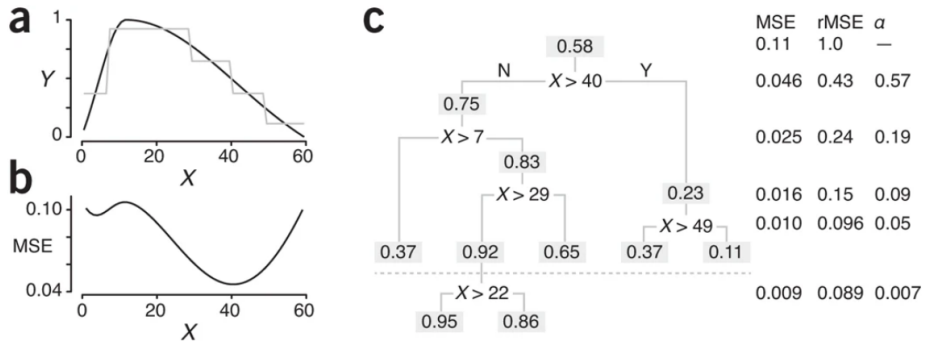


Fig adapted from Nature Methods: Classification and regression trees (08/2017)

CART Regression

- Same as decision tree, **but divide predictor space $f = f(x_1, x_2, .., x_n)$ into distinct, non overlapping regions**.
- Every sample falls into a feature region defined by $(\boldsymbol{x_1, x_2, .., x_n})$
- make **prediction** which is the **mean response in training set in that region**



(**a**) A nonlinear function (black) with its prediction (gray) based on a regression tree. (**b**) Splits in the regression tree minimize mean square error (MSE), which is shown here for all possible positions of the first split. (**c**) The full regression tree for the prediction shown in **a**. For each split, the absolute MSE and relative (to the first node) rMSE is shown along with the difference in successive rMSE values, a. The tree was built with cutoff of a = 0.01, which terminates the growth of the tree at the dashed line.

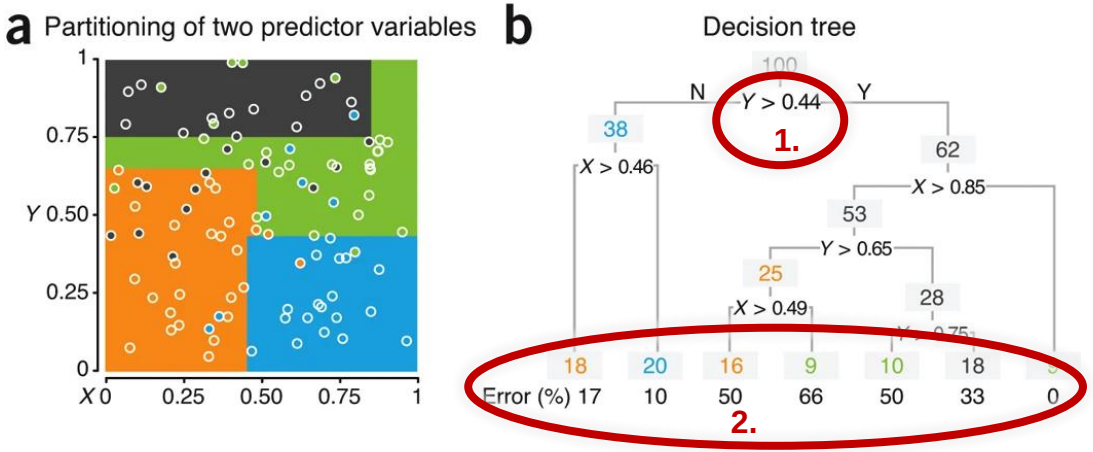
Fig from Nature Methods: Classification and regression trees (08/2017)

CART Model Parameters

Main design option of the CART Tree (and hence also for tree ensemble methods such as RF)

1) **Quantification of the optimum partition**

2) **Termination criterion**



Splitting Criterion

- **Quantification of the partition performance**, different for classification or Regression Trees
- Select **partition with maximised information gain**

Classification:

- False classification rate
- Entropy/Information Gain
- Gini Impurity (computationally much faster than entropy)

$$Gini = 1 - \sum_{i=1}^{n} p^2(c_i)$$

$$Entropy = \sum_{i=1}^{n} -p(c_i) log_2(p(c_i))$$

where $p(c_i)$ is the probability/percentage of class c_i in a node.

criterion : *{"gini", "entropy"}, default="gini"*

Regression

- Squared error (converges faster than Absolute Error)
- Absolute Error
- Poisson

criterion : *{"squared_error", "absolute_error", "poisson"}, default="squared_error"*

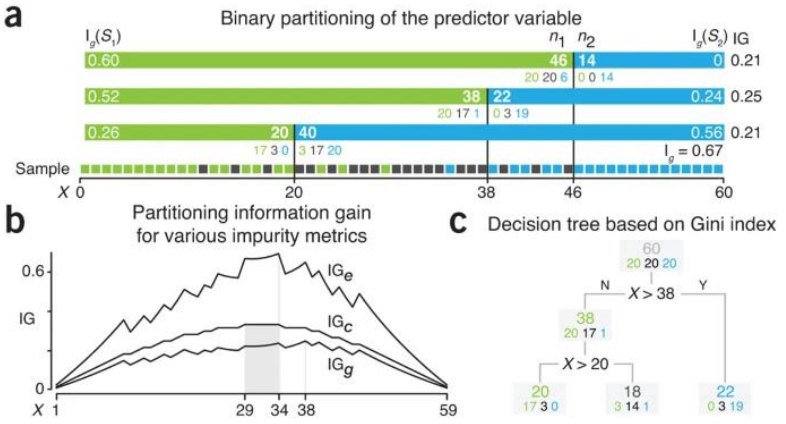


Fig from [Nature Methods: Classification and regression trees](#) (08/2017)

Termination Criterion

When do we stop splitting a node any further? Many options, e.g.

- Number of data points in node below threshold

  min_samples_leaf : *int or float, default=1*

- Potential additional splits do not lead to improvement in splitting criterion

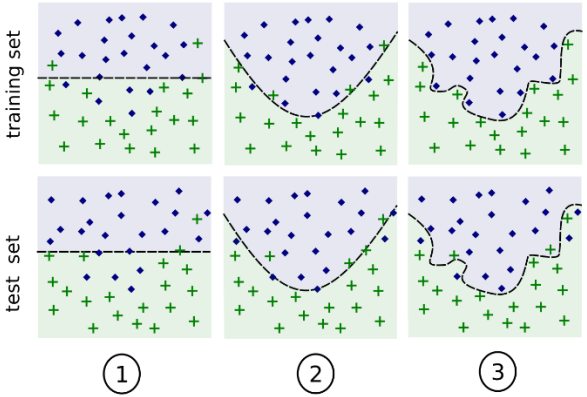
  min_impurity_decrease

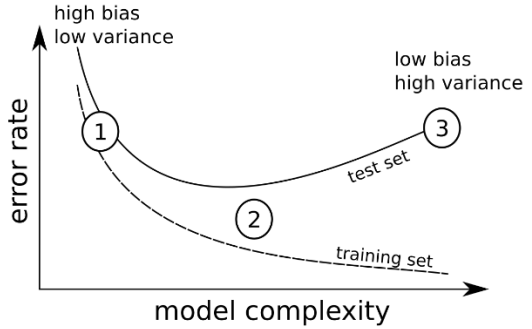
- Split which improves entropy results in a leaf with very small number of samples
- Max. depth of tree

Theoretically, a single CART can grow arbitrarily deep and can fully overfit the training data set

Generally, for a single CART when we

- **1. Stop splitting early: high bias, low variance**
- **3. Grow deep tree: low bias, high variance**





Ensemble Methods

Many “weak classifiers” make a powerful committee

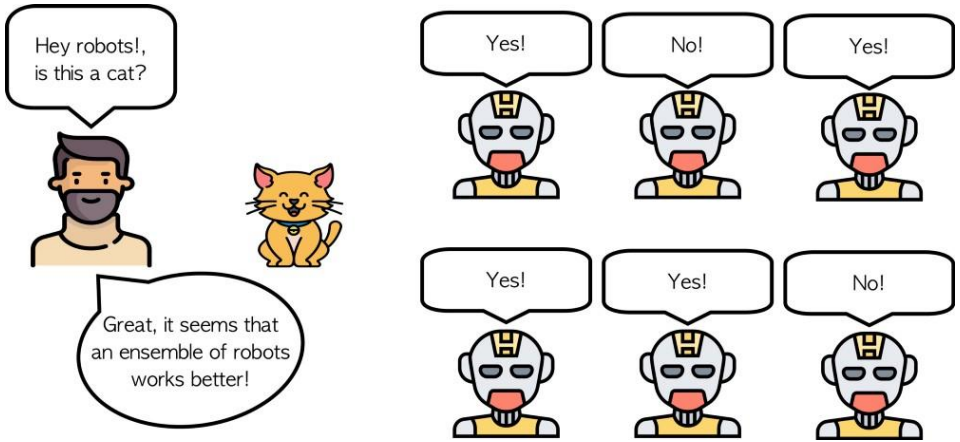
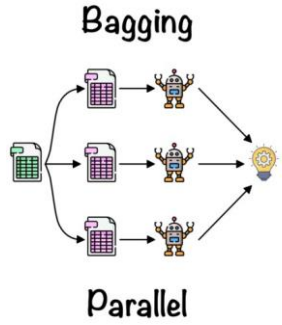


Fig from https://towardsdatascience.com/ensemble-learning-stacking-blending-voting-b37737c4f483

Ensemble Methods

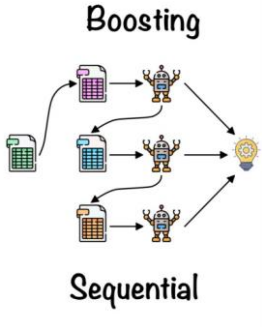
= combine several base models in order to create an improved model



Bootstrap sampling is a concept applied in many ensemble methods **= sample, which is drawn with replacement**. E.g. each tree is grown with $Ntrain$ samples, but within this training set some samples can occur multiple times while others are not represented → All week models are trained with different training data

Several ways to combine base models to an ensemble, e.g.

- **Bagging** = "**B**ootstrap **Agg**regation": Create each weak model with a **bootstrap sample**
- **Boosting:**
  - **Sequential application of weak classifier** to modified version of dataset/model & all decision trees are trained with different training data
  - Adapt sample weights (e.g. AdaBoost) or model (e.g. Stochastic Gradient Boosting) in every step to improve predictive power



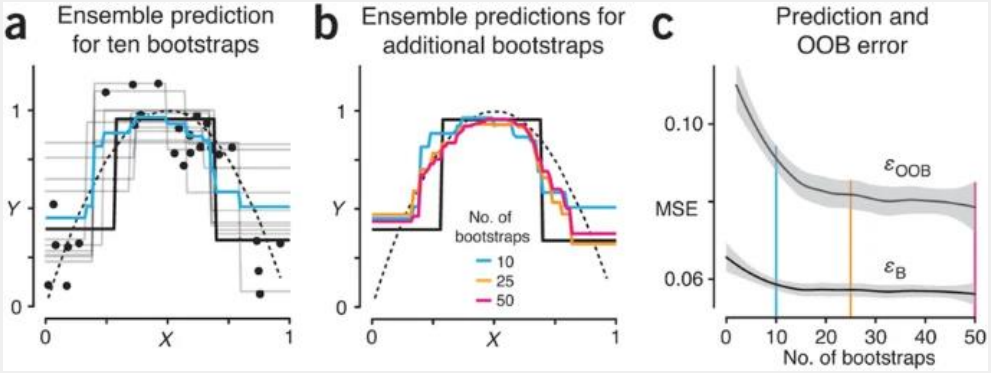
Boosting vs. Bagging? Depends on the problem & dataset!

- Bagging does not improve bias but avoids overfitting
- Boosting improves bias but is prone to overfitting

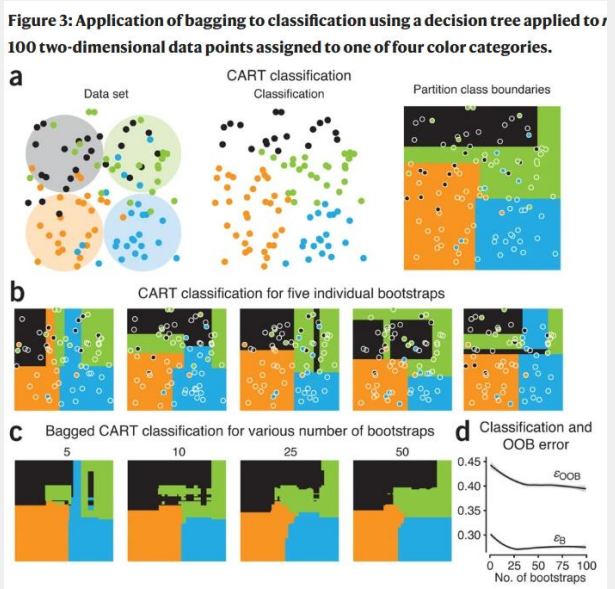
Fig from https://towardsdatascience.com/ensemble-learning-bagging-boosting-3098079e5422

Bagged (="bootstrap aggregation") Trees

- **build a large numb er NT trees (ensemble),** each of them exhibiting low bias but high variance.



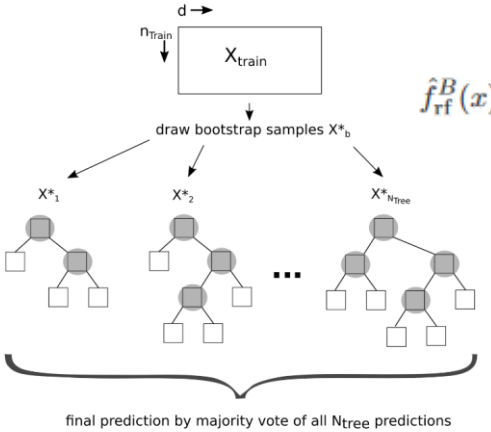
(**a**) The consensus regression (blue line) for ten bootstrap iterations (gray lines) for data in Figure 1. (**b**) Ensemble regressions for 10, 25 and 50 bootstrap iterations. (**c**) The bagged and OOB errors (ε_B, ε_{OOB}) as a function of the number of bootstraps. The curve is fit to ten simulations at each bootstrap level using locally weighted smoothing. The gray band is the fit's 95% confidence interval.



https://www.nature.com/articles/nmeth.4438

Bagged (="bootstrap aggregation") Trees

- **build a large numb er NT trees (ensemble),** each of them exhibiting low bias but high variance.
- by averaging their results **variance is reduced while bias stays the same**
- each of the N_T trees is grown with a **bootstrap sample** of size N_{train}, which is drawn from the training data (with replacement – **different data points seen by different trees**!)



$$\hat{f}_{\mathrm{rf}}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T(x;\Theta_b).$$

The variance of each tree is assumed to be σ^2 and the positive pairwise correlation[5] is $\rho\sigma^2$. The variance of the mean of N_T trees with variance σ^2 is

$$\begin{aligned}
var\left[\frac{1}{N_T}\sum_{b=1}^{N_T}\hat{\mathbf{f}}(\mathbf{X}^*{}_b)\right] &= \frac{1}{N_T^2}var\left[\sum_{b=1}^{N_T}\hat{\mathbf{f}}(\mathbf{X}^*{}_b)\right] = \\
&= \frac{1}{N_T^2}\left(\sum_{b=1}^{N_T} var\left(\hat{\mathbf{f}}(\mathbf{X}^*{}_b),\hat{\mathbf{f}}(\mathbf{X}^*{}_b)\right)\right) + \\
&\quad \frac{1}{N_T^2}\left(\sum_{b1=1}^{N_T}\sum_{b2\neq b1;b2=1}^{N_T} var\left(\hat{\mathbf{f}}(\mathbf{X}^*{}_{b1}),\hat{\mathbf{f}}(\mathbf{X}^*{}_{b2})\right)\right) \qquad (1.10) \\
&= \frac{1}{N_T^2}\left(N_T\cdot\sigma^2 + N_T\cdot(N_T-1)\rho\sigma^2\right) \\
&= \sigma^2\rho + \frac{1-\rho}{N_T}\sigma^2
\end{aligned}$$

- **If single models (trees) are correlated we do not improve the variance of the ensemble model! Max. Variance reduction compared to single tree when $\rho = 0$**

Random Forests

- **Tweaked version of bagged trees**
- Bagged trees, while **reducing correlation between individual trees**
- First introduced by Leo Breimann (1928-2005) in 2001
  doi:10.1023/A:1010933404324

The variance of each tree is assumed to be σ^2 and the positive pairwise correlation[5] is $\rho\sigma^2$. The variance of the mean of N_T trees with variance σ^2 is

$$
\begin{aligned}
var\left[\frac{1}{N_T}\sum_{b=1}^{N_T}\hat{\mathbf{f}}(\mathbf{X}^*{}_b)\right] &= \frac{1}{N_T^2}var\left[\sum_{b=1}^{N_T}\hat{\mathbf{f}}(\mathbf{X}^*{}_b)\right] = \\
&= \frac{1}{N_T^2}\left(\sum_{b=1}^{N_T}var\left(\hat{\mathbf{f}}(\mathbf{X}^*{}_b),\hat{\mathbf{f}}(\mathbf{X}^*{}_b)\right)\right) + \\
&\quad \frac{1}{N_T^2}\left(\sum_{b1=1}^{N_T}\sum_{b2\neq b1;b2=1}^{N_T}var\left(\hat{\mathbf{f}}(\mathbf{X}^*{}_{b1}),\hat{\mathbf{f}}(\mathbf{X}^*{}_{b2})\right)\right) \qquad (1.10) \\
&= \frac{1}{N_T^2}\left(N_T\cdot\sigma^2 + N_T\cdot(N_T-1)\rho\sigma^2\right) \\
&= \sigma^2\rho + \frac{1-\rho}{N_T}\sigma^2
\end{aligned}
$$

reduce ρ

Random Forest

- Eq. (1.10) shows how the **variance decreases with increasing N_T**
- crucial that the positive **pairwise correlation ρ** of the decision trees is **as low as possible** to profit of variance reduction.

The variance of each tree is assumed to be σ^2 and the positive pairwise correlation[5] is $\rho\sigma^2$. The variance of the mean of N_T trees with variance σ^2 is

$$
\begin{aligned}
var\left[\frac{1}{N_T}\sum_{b=1}^{N_T}\hat{\mathbf{f}}(\mathbf{X}^*{}_b)\right] &= \frac{1}{N_T^2}var\left[\sum_{b=1}^{N_T}\hat{\mathbf{f}}(\mathbf{X}^*{}_b)\right] = \\
&= \frac{1}{N_T^2}\left(\sum_{b=1}^{N_T}var\left(\hat{\mathbf{f}}(\mathbf{X}^*{}_b),\hat{\mathbf{f}}(\mathbf{X}^*{}_b)\right)\right) + \\
&\quad \frac{1}{N_T^2}\left(\sum_{b1=1}^{N_T}\sum_{b2\neq b1;b2=1}^{N_T}var\left(\hat{\mathbf{f}}(\mathbf{X}^*{}_{b1}),\hat{\mathbf{f}}(\mathbf{X}^*{}_{b2})\right)\right) \\
&= \frac{1}{N_T^2}\left(N_T\cdot\sigma^2 + N_T\cdot(N_T-1)\rho\sigma^2\right) \\
&= \sigma^2\rho + \frac{1-\rho}{N_T}\sigma^2
\end{aligned}
\tag{1.10}
$$

- **RF: build a large numb er NT of de-correlated trees (ensemble),** each of them exhibiting low bias but high variance **(reduce ρ)**
- **ensured by considering a only a subset $m < d$ of features** (rather than all d descriptors) for the defining the splitting condition on **each node.**
- different splitting conditions are found for each node, resulting in differing trees and minimised pairwise correlation

Important: In the **ensemble we only reduce the variance and not the bias**! → each individual **model needs to have a small bias, i.e. be a deep, (overfitted) tree**

Pseudocode Bagged Trees

Algorithm 3: Random Forest [9, 22]

Training Phase: Growing and storing N_T decc
binary decision trees. For each tree

1. a bootstrap sample $\mathbf{X}^*$ of size N_{train} is drawn of the training data,
2. with which a decision tree is grown. The splitting criteria for each node is found by

   b) estimating the best splitting decision for splitting the data resulting in two new daughter nodes.

   The growing process is continued until all terminal nodes reach a specified level of purity.

Classification Phase:

1. The test samples are classified by each of the N_T trees.
2. The final class is assigned by majority voting.

Bagging

Example for binary classification → transferrable for regression

Pseudocode Random Forest

Algorithm 3: Random Forest [9, 22]

Training Phase: Growing and storing N_T decc
binary decision trees. For each tree

1. a bootstrap sample $\mathbf{X}^*$ of size N_{train} is drawn of the training data,
2. with which a decision tree is grown. The splitting criteria for each node is found by
   a) randomly selecting a subset $m \leq d$ variables of the descriptors,
   b) estimating the best splitting decision for splitting the data resulting in two new daughter nodes.

   The growing process is continued until all terminal nodes reach a specified level of purity.

Classification Phase:

1. The test samples are classified by each of the N_T trees.
2. The final class is assigned by majority voting.

Example for binary classification → transferrable for regression

Bagging

RF: Random Feature Selection

Implementation in Sci-Kit Learn, Parameters and Attributes

sklearn.ensemble.RandomForestRegressor

class sklearn.ensemble.RandomForestRegressor(*n_estimators=100, *, criterion='squared_error', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, ccp_alpha=0.0, max_samples=None*) [source]

RF: Implementaton & Parameters

sklearn.ensemble.RandomForestRegressor

class sklearn.ensemble.RandomForestRegressor(*n_estimators=100, *, criterion='squared_error', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, ccp_alpha=0.0, max_samples=None*) [source]

```python
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier

# Fit RF classifier
forest = RandomForestClassifier()
forest.fit(X_train, y_train)
```

- RF are rather robust to parameter tuning,
- **Default parameters are normally good enough** and not much tuning is required! Attractiveness of RF!

- **Parameters covering main properties from previous slides:**
  - **Main hyperparameter is number of trees (n_estimators):** See next slide
  - **Max_features:** default normally ok (different in classification and regression)
    - Number of features used for splitting decision at each node, Can improve performance depending on dataset (difference from RF to Bagged Trees)
  - For performance evaluation: set **oob_score = True**

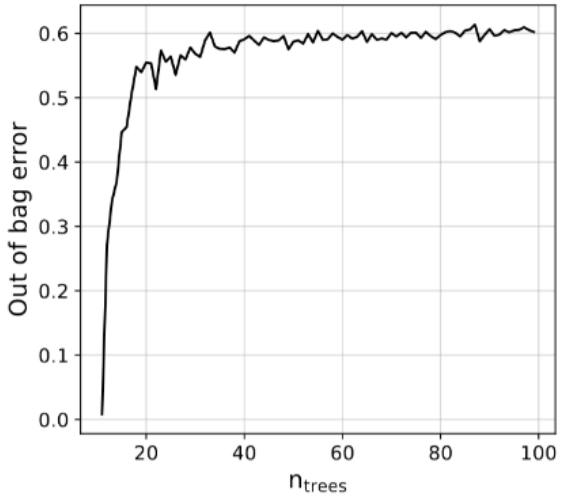
  - **Termination Criteria** *(Max Depth / min_samples_splot, min_samples_leaf, …)* stop growing of the tree. We want a deep tree to have small bias as supported by default parameters (no max. depth, etc)
  - **Bootstrap=True:** Use bootstrap sample during training
  - **Criterion for node splitting:** *criterion{"squared_error", "absolute_error", "poisson"}, default="squared_error"*

RF – Hyperparameter: Number of Trees

- **Larger number of trees reduces variance**

$$\sigma^2\rho + \frac{1-\rho}{N_T}\sigma^2$$

- For large N_T, different trees start being correlated, and we do not decrease σ by adding more trees
- But we **don't degrade predictive power of model** if N_T is unnecessarily large
- Still, larger N_T = larger training / evaluation time



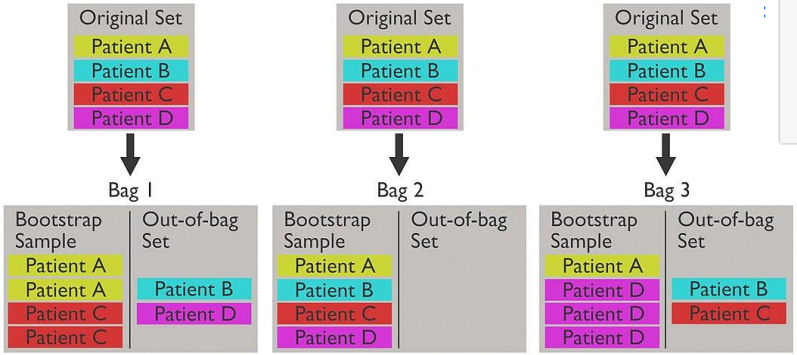
- For most applications $N_T = 75 - 100$ is sufficient
- Compute performance of model with different N_T (e.g., oob estimate as shown next slides or k-fold cross validation) and select N_T for which performance does not improve anymore

```
oob = list()
for i in np.arange(5,100,2):
    regr = RandomForestRegressor(n_estimators = i, oob_score=True)
    pred = regr.fit(xtrain,ytrain)
    oob.append(regr.oob_score_)
```

RF: Out of Bag Error

As we draw a bootstrap sample to train each decision tree, the trees have not seen all samples during training.

Out of Bag Error: Compute the mean prediction error for each sample when using only the trees which did not see this sample during training for prediction




```
oob = list()
for i in np.arange(5,100,2):
    regr = RandomForestRegressor(n_estimators = i, oob_score=True)
    pred = regr.fit(xtrain,ytrain)
    oob.append(regr.oob_score_)
```

Fig from https://en.wikipedia.org/wiki/Out-of-bag_error

RF: Feature Importance

- Identifying which **variables were important for the prediction** provides some interpretability of the model
- When training each tree, we can assess how **much each feature contributed in reducing the impurity of a node** (in RF: average decrease in impurity over all trees)
- **Impurity – based feature importance** comes “for free” and is quick to retrieve but can be biased to high cardinality features & different feature scales

→ Good first approach, but be careful (as always).

Forest paper "We show that random forest variable importance measures are a sensible means for variable selection in many applications, but are not reliable in situations where potential predictor variables vary in their scale of measurement or their number of categories."

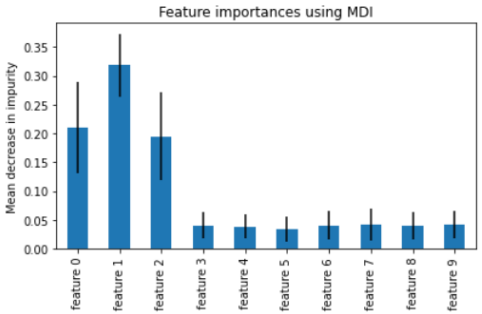
Example fom scikit-learn

RF classifier on data set with 3 informative features and 7 “useless” features

```
# Fit RF classifier
forest = RandomForestClassifier(random_state=0)
forest.fit(X_train, y_train)

# A: Impurity based feature importance ("for free")
importances = forest.feature_importances_
forest_importances = pd.Series(importances, index=feature_names)
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axis=0)

# Plot feature importances
fig, ax = plt.subplots()
forest_importances.plot.bar(yerr=std, ax=ax)
ax.set_title("Feature importances using MDI")
ax.set_ylabel("Mean decrease in impurity")
fig.tight_layout()
```



Side Note: Comparison to Permutation Importance

Often listed as alternative to the “free” feature importance is the **permutation importance**

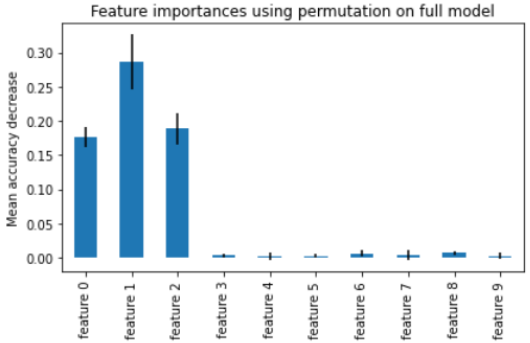
- Take test data set, shuffle values of one feature & estimate difference in prediction on original test data set & shuffled dataset
- Only reliable for non-correlated features

applicable to any predictive model, not specific to RF!

Example fom scikit-learn

RF classifier on data set with 3 informative features and 7 “useless” features

```
:
1  from sklearn.inspection import permutation_importance
2
3  # B: Get permutation based feature importance with test set
4  result = permutation_importance(
5      forest, X_test, y_test, n_repeats=10, random_state=42, n_jobs=2
6  )
7
8  forest_importances = pd.Series(result.importances_mean, index=feature_names)
9  fig, ax = plt.subplots()
10 forest_importances.plot.bar(yerr=result.importances_std, ax=ax)
11 ax.set_title("Feature importances using permutation on full model")
12 ax.set_ylabel("Mean accuracy decrease")
13 fig.tight_layout()
14 plt.show()
```



Random Forest Regressor: Pros & Cons

Advantages:

+ Handles **non-linear data**
+ Intrinsic **dimensionality reduction**
+ **Avoid overfitting**, less impacted by noise
+ **Robust regarding hyper-parameter settings** → often usable out of the box with default parameters
+ Can provide interpretability with **feature importance** (how important was each feature for splitting decision?)
+ With **out-of-bag error** (=prediction error of samples, which were not used for training that tree) intrinsic cross validaton.
+ Handles **missing values** well
+ **No feature scaling** required

Disadvantages:

- **Cannot extrapolate** beyond unseen values
- For many data samples & many trees **high training/evaluation time** (need deep trees to keep bias low)
- Poor performance on **imbalanced / skewed data**

Example: Surrogate Model of PSB Injection using RF

Dummy Example (simplified example for illustrative purposes):

https://gitlab.cern.ch/erenner/rf_intro_example

Full project can be found here:
https://gitlab.cern.ch/erenner/psb_inj_surrogate
The surrogate model is trained in `create_model_rf.py` and applied in `run_pybobyqa_hyperparamstuning.py`

Example: Surrogate Model for PSB Injection Painting I

1. Load the Data

- here saved in a csv file
- I filter the data before training the model, i.e. remove samples with very large losses (>5%)

```
df0 = pd.read_csv('data/optimise_isolde_dataspace/data_collection_90turns.csv')

df = df0[df0["loss1"]<0.05].copy().reset_index()
keys = df.keys().to_list()

df.head()
```

	index	bump	A1	v2	t1	offset	loss1
0	0	30.850	25.0	-0.08910	18.0	0.0025	0.031160
1	1	32.925	25.0	-0.08910	18.0	0.0025	0.030780
2	2	30.850	29.5	-0.08910	18.0	0.0025	0.030415
3	3	30.850	25.0	-0.08910	24.0	0.0025	0.029440
4	4	30.850	25.0	-0.04455	18.0	0.0025	0.026432

2. Split into features & target variables and training & Test Dataset

```
target = df[['loss1']].copy()
print('The shape of our output (y) is:', target.shape)

data = df[['index', 'bump', 'A1', 'v2', 't1', 'offset']].copy()
print('The shape of our features (x1,...,xn) is:', data.shape)

xtrain, xtest,  ytrain, ytest = train_test_split(data, target, train_size = 0.8)
```

```
The shape of our output (y) is: (2963, 1)
The shape of our features (x1,...,xn) is: (2963, 6)
```

3. Train RF model for different numbers of Trees and plot OOB Error to estimate suitable number of trees

```
n_trees = np.arange(10,100,2)
oob = list()

for nt in n_trees:
    regr = RandomForestRegressor(n_estimators=nt, max_depth=None, max_features='auto', oob_score=True)
    pred = regr.fit(xtrain, ytrain.values.ravel())
    oob.append(regr.oob_score_)
```

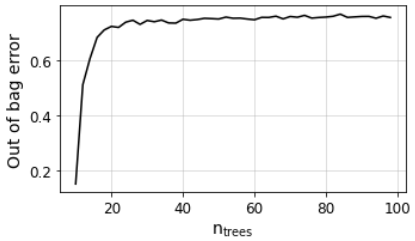
4. Train Model with 75 Trees and predict output for test & training data

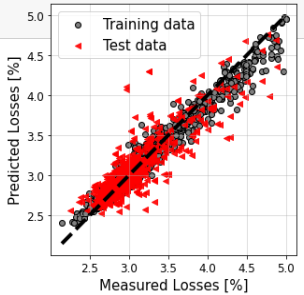
Remeber, this is just an example. Consider using e.g. cross validation in a real application

```
regr = RandomForestRegressor(n_estimators=75, max_depth=None, max_features='auto', oob_score=True)

pred = regr.fit(xtrain, ytrain.values.ravel())
#predicted = cross_val_predict(regr, xtrain, ytrain.values.ravel(), cv=10)

predicted = regr.predict(xtrain)
predicted_test = regr.predict(xtest)
```



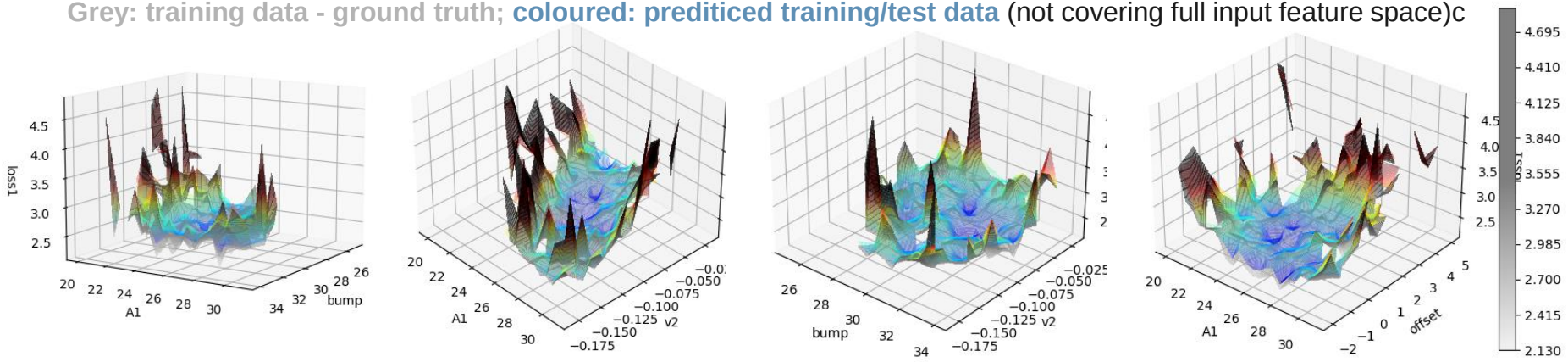


5. Use the model when calling your optimisation environment

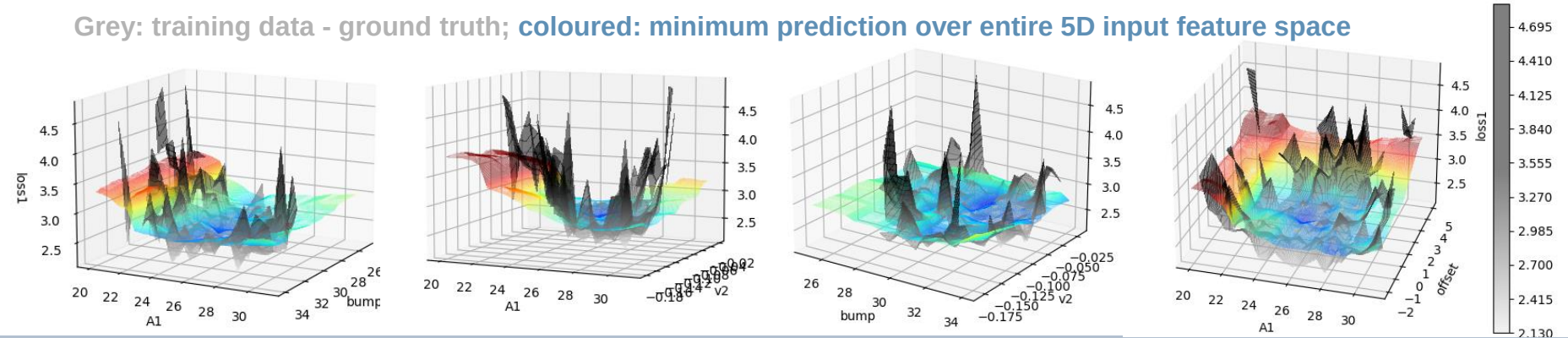
- call `regr.predict(x)` is every step in your optimisation environment (instead of getting observables on the machine
- add random noise to model prediction if appropriate for the model

Example: Surrogate Model for PSB Injection Painting II

Grey: training data - ground truth; **coloured: prediticed training/test data** (not covering full input feature space)c

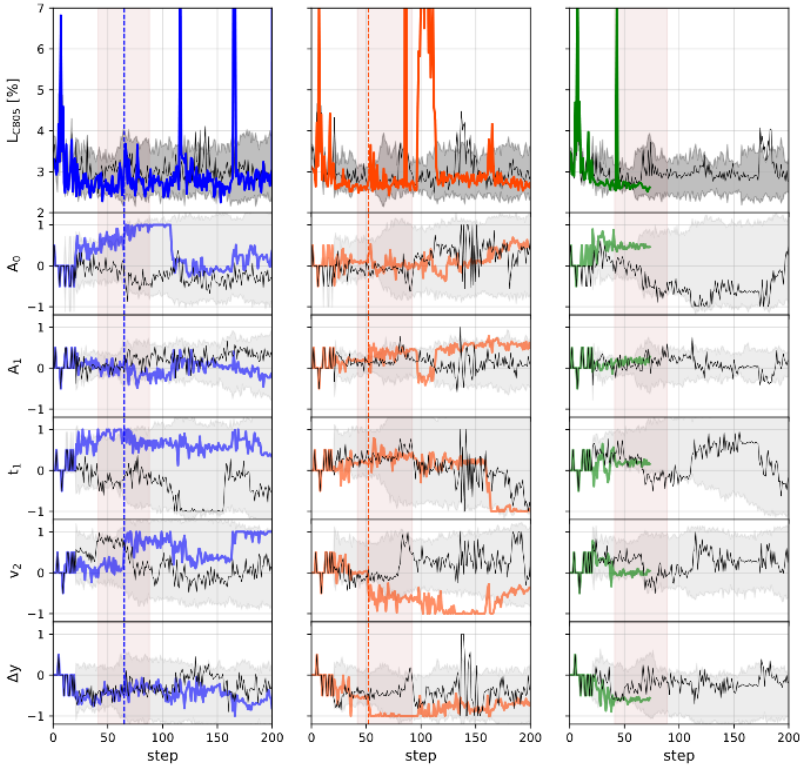


Grey: training data - ground truth; **coloured: minimum prediction over entire 5D input feature space**



Example: Surrogate Model for PSB Injection Painting III

- Fig: **Qualitative comparison** when applying pyBobyqa on the **machine** (colored) vs the **surrogate model** (greyscale)
- obtained
  - **loss evolutions** (top)
  - **set feature combinations** (bottom)
- **different number of samples** averaged per step:
  - Blue: n=1
  - Red: n=5
  - Green: n=20



Outlook & Summary

Applications & Related Methods

Ensemble methods have been **widely applied in many fields** (medicine, finance, remote sensing of landscapes e.g. satellite images, …) and there are many aspects which have not been covered in this presentation!

Just to state a few related methos (adaptions to RF & other ensemble methods)

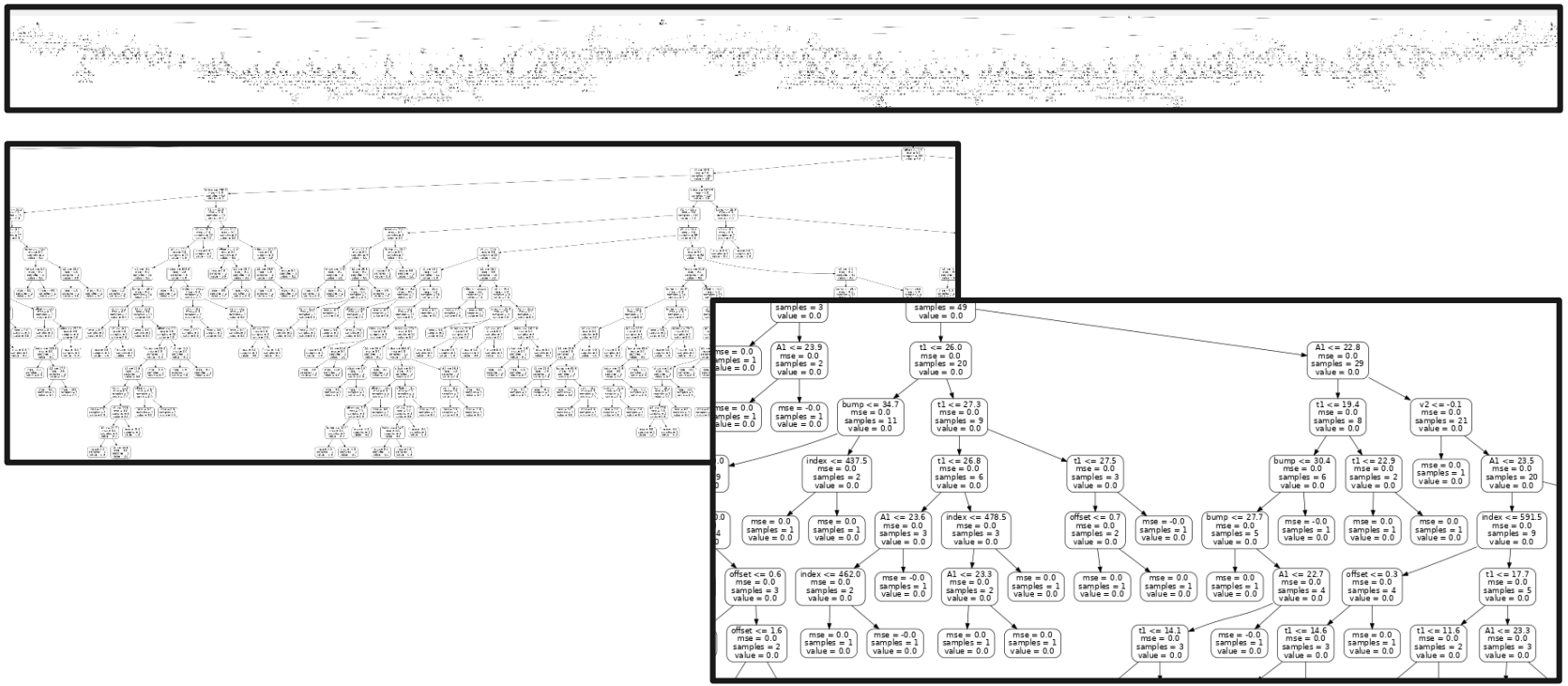
- Quantile regression forest
- Isolation Forest (Anomaly detection)
- Ensemble of other “weak models”, such as kNNs
- Deep learning:
  - Create an ensemble of deep networks in the classical sense is computationally expensive, but ensemble idea is also included in deep learning methods in different ways
  - Ensemble deep learning: A review (04/2021)
  - E.g. dropout layers in NN are often considered an ensemble method

Summary

- A Random Forest (RF) is an **ensemble classifier/regressor based on binary decision trees.**
- **Deep Decision trees** exhibit **low bias but high variance** (overfit to training data, noise).
- **Ensemble methods** use a **large number of trees trained on different datasets or with modified models**
- **RF are a tweaked version of Bagged Tree Ensembles**, which use a large number of decorrelated trees to reduce prediction variance
- RF can be used for **classification** and **regression** & are very **robust to data pre-processing and hyper parameter tuning** (difficult to mess it up!)
- Recommend to add it to the repertoire of considered methods when solving a problem!
- Surrogate model for the PSB injection painting optimisation has been created using a RF → very fast & easy implementation.
- Would it make sense to implement an extension to GeOFF which allows to save actor/objective data in a uniform way and then - after several runs - allows to load datafiles & compute RF model?

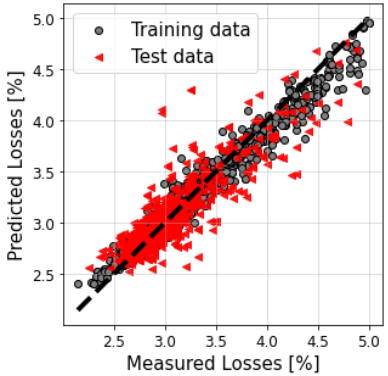
Thanks ☺

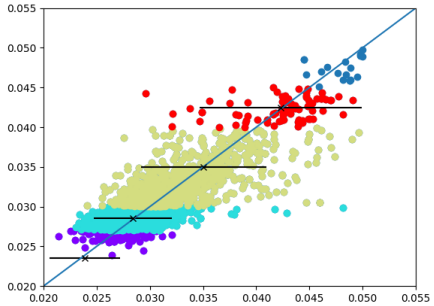
A single tree is large!

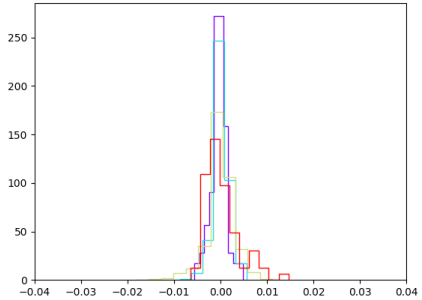


Example: Surrogate Model for PSB Injection Painting II

- **Captures data structure well**
- Obvious **skewed distribution of rediduals**, depending on value of objective function
- Identified as not critical as accurat modeling of smaller objectives (i.e. closer to minimum) more relevant.








```
regr = RandomForestRegressor(n_estimators = 75, max_depth = None, oob_score=True)
```