Tree models

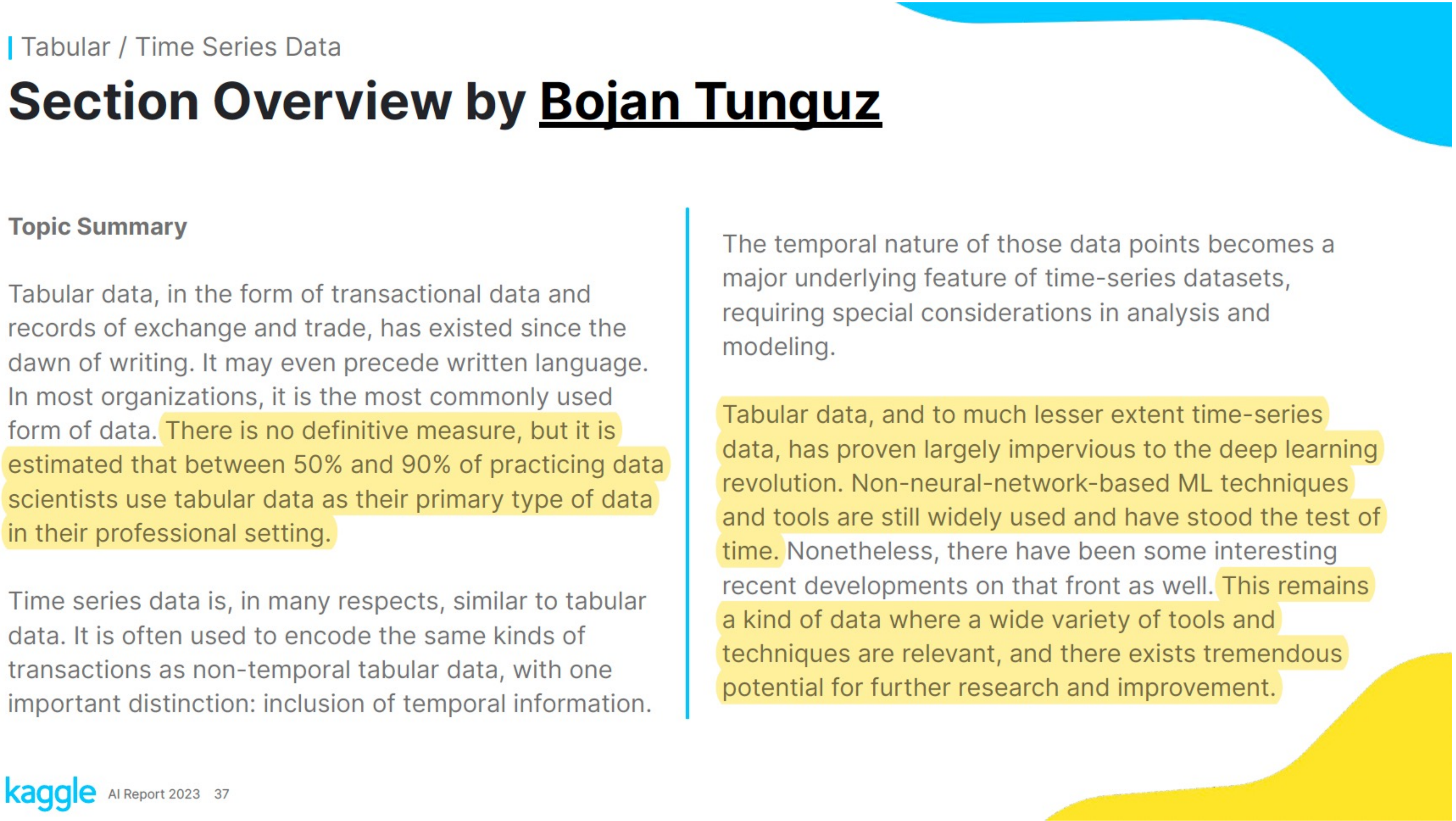
or - How to make the most of your tabular data

Mihai Alexe

ECMWF Bonn

first.lastname@ecmwf.int

Kaggle – AI report, 2023



| Tabular / Time Series Data

Section Overview by Bojan Tunguz

Topic Summary

Tabular data, in the form of transactional data and records of exchange and trade, has existed since the dawn of writing. It may even precede written language. In most organizations, it is the most commonly used form of data. There is no definitive measure, but it is estimated that between 50% and 90% of practicing data scientists use tabular data as their primary type of data in their professional setting.

Time series data is, in many respects, similar to tabular data. It is often used to encode the same kinds of transactions as non-temporal tabular data, with one important distinction: inclusion of temporal information. The temporal nature of those data points becomes a major underlying feature of time-series datasets, requiring special considerations in analysis and modeling.

Tabular data, and to much lesser extent time-series data, has proven largely impervious to the deep learning revolution. Non-neural-network-based ML techniques and tools are still widely used and have stood the test of time. Nonetheless, there have been some interesting recent developments on that front as well. This remains a kind of data where a wide variety of tools and techniques are relevant, and there exists tremendous potential for further research and improvement.

kaggle AI Report 2023 37

https://storage.googleapis.com/kaggle-media/reports/2023_Kaggle_AI_Report.pdf

What is tabular data?

- Data that can be well presented in a table!

- Rows are examples to train on.

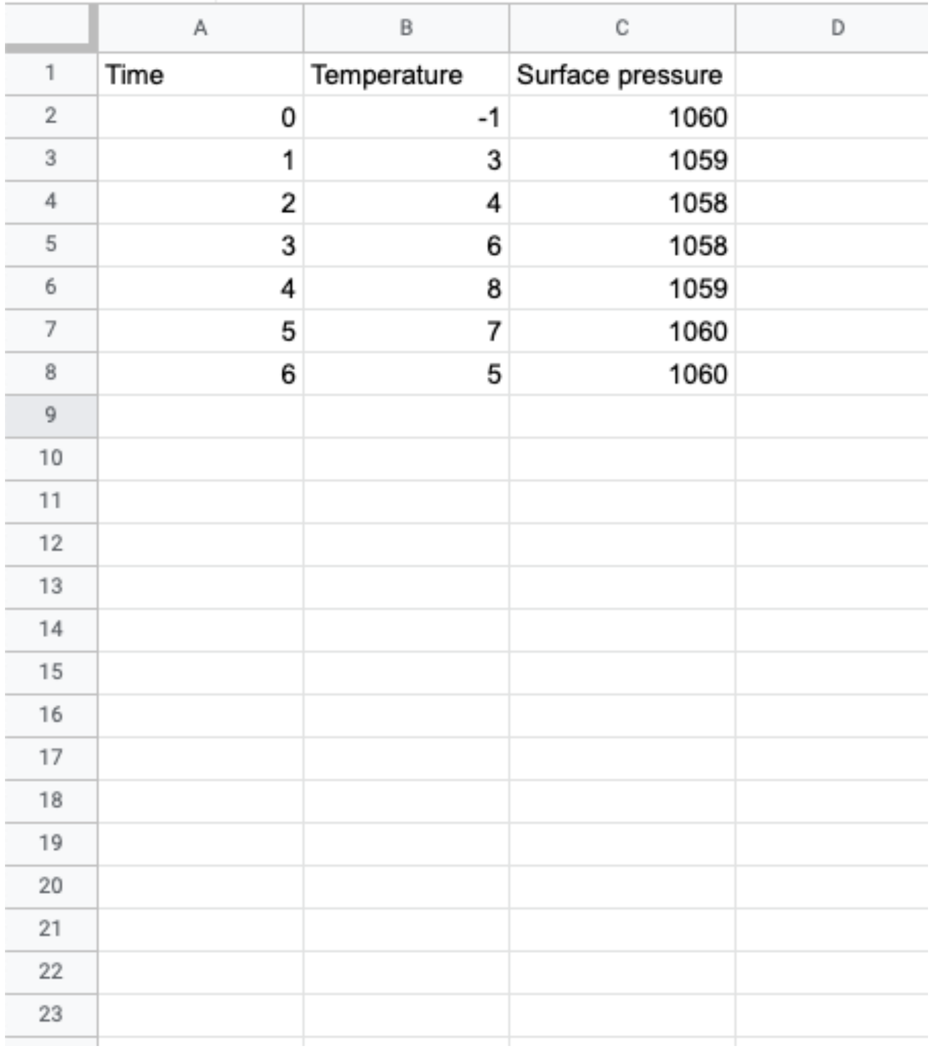
- Columns are different variables to be used in prediction or to be predicted.

- When is earth-system data tabular?
  - When the temporal and spatial components of the problem are not important.
  - e.g. correcting the weather forecast for your house.

- On tabular, the methods we will explore today are very strong.

Predictors/predictands

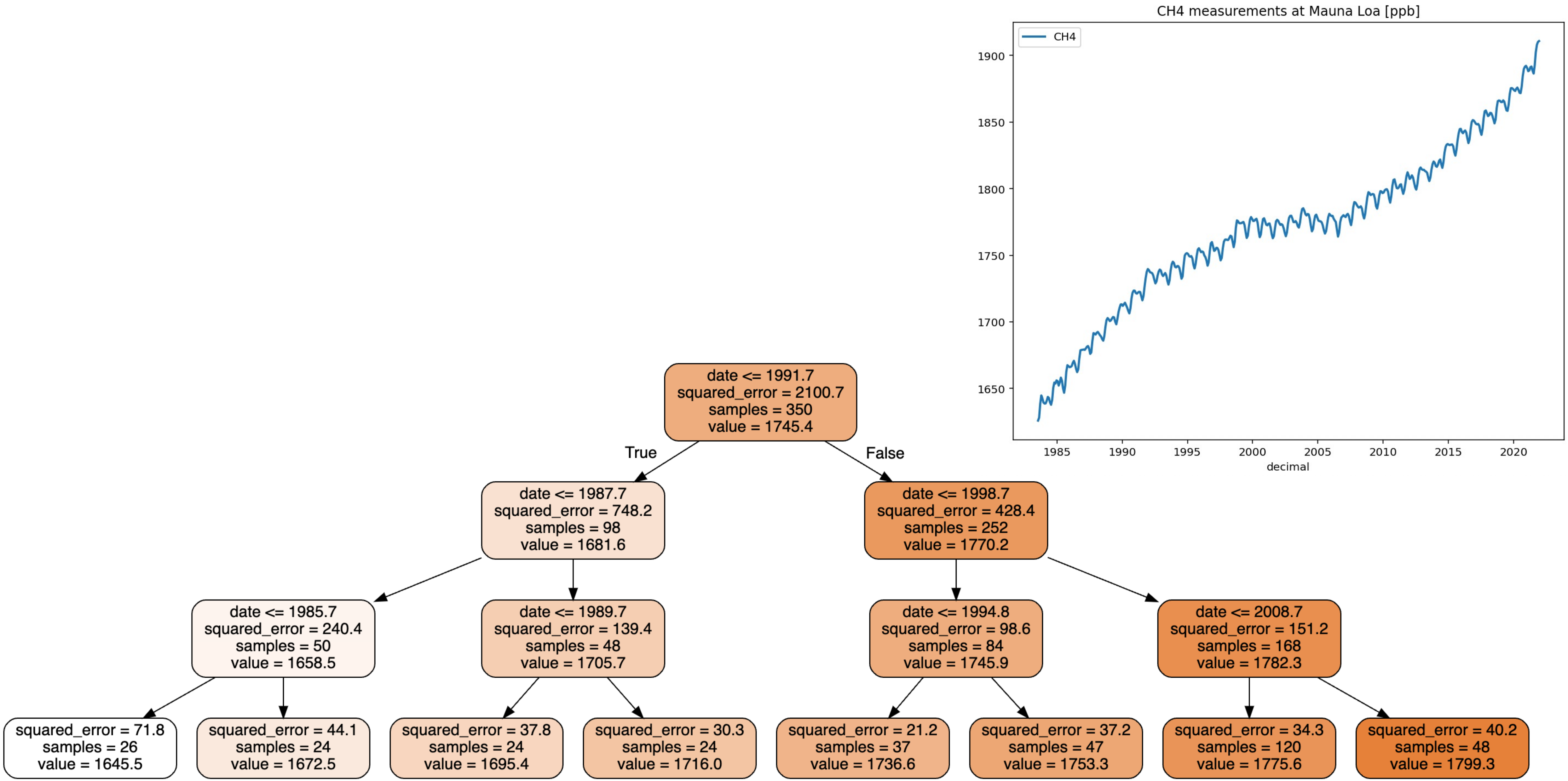
Examples



	A	B	C	D
1	Time	Temperature	Surface pressure	
2	0	-1	1060	
3	1	3	1059	
4	2	4	1058	
5	3	6	1058	
6	4	8	1059	
7	5	7	1060	
8	6	5	1060	
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				

Decision trees

DECISION TREE
SO I MADE THIS ELABORATE DECISION TREE,
FXNOW



date <= 1991.7
squared_error = 2100.7
samples = 350
value = 1745.4

date <= 1987.7
squared_error = 748.2
samples = 98
value = 1681.6

date <= 1998.7
squared_error = 428.4
samples = 252
value = 1770.2

date <= 1985.7
squared_error = 240.4
samples = 50
value = 1658.5

date <= 1989.7
squared_error = 139.4
samples = 48
value = 1705.7

date <= 1994.8
squared_error = 98.6
samples = 84
value = 1745.9

date <= 2008.7
squared_error = 151.2
samples = 168
value = 1782.3

squared_error = 71.8
samples = 26
value = 1645.5

squared_error = 44.1
samples = 24
value = 1672.5

squared_error = 37.8
samples = 24
value = 1695.4

squared_error = 30.3
samples = 24
value = 1716.0

squared_error = 21.2
samples = 37
value = 1736.6

squared_error = 37.2
samples = 47
value = 1753.3

squared_error = 34.3
samples = 120
value = 1775.6

squared_error = 40.2
samples = 48
value = 1799.3

How do you pick the right decisions?

- Search over a set of possible splits in the data (e.g. dates in the previous slide).
  - For each split, calculate an **impurity/loss**.
  - Choose the split that **minimises** the loss value.

- For **classification**, calculate probability of being a class for the samples in the branch.
  - Gini impurity
  - Log loss/entropy

$$H(Q_m) = -\sum_k p_{mk} \log(p_{mk})$$

- For **regression**:
  - Mean-squared error
    - of each value in the tree against the branch-average value.
  - Mean absolute error
  - Half Poisson deviance

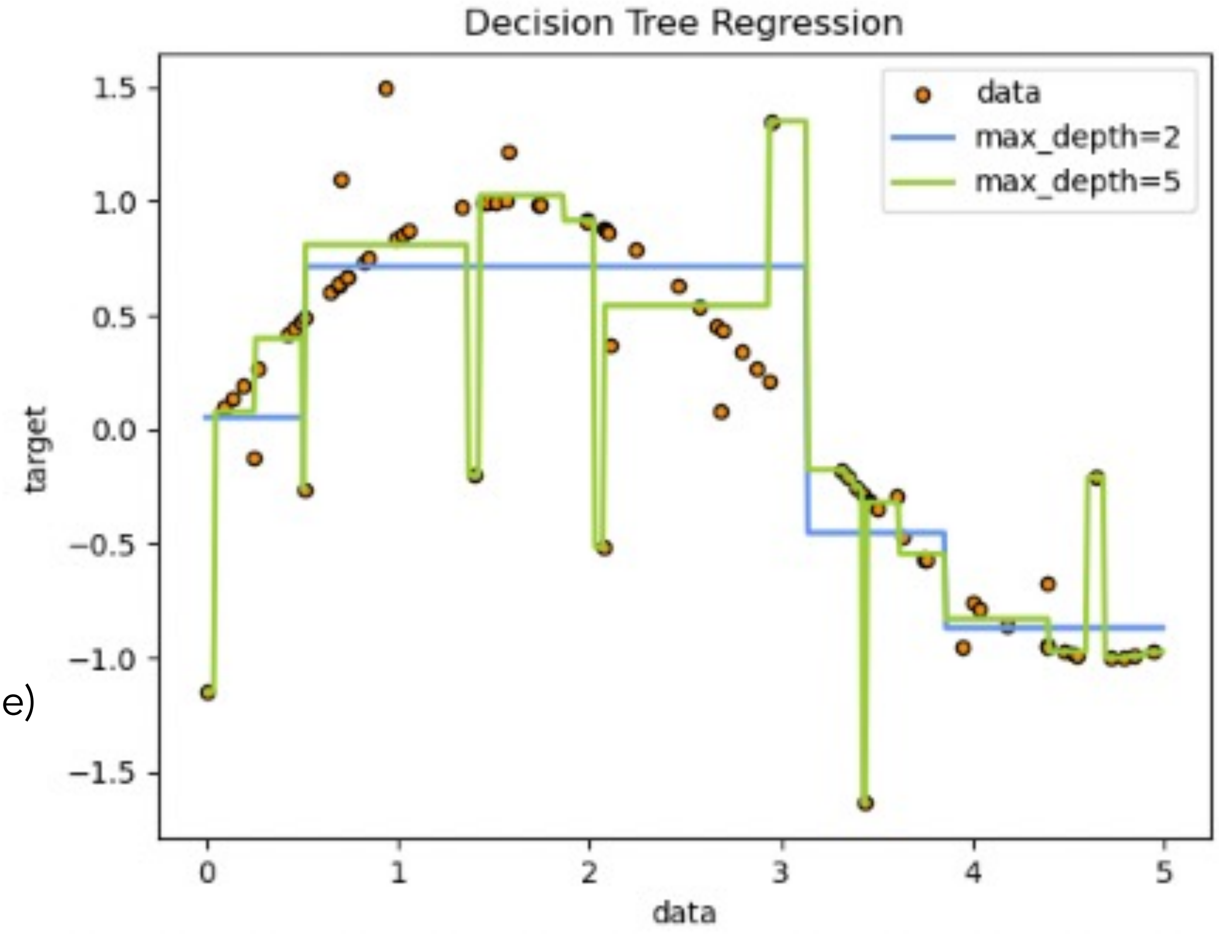
$$\bar{y}_m = \frac{1}{n_m} \sum_{y \in Q_m} y$$

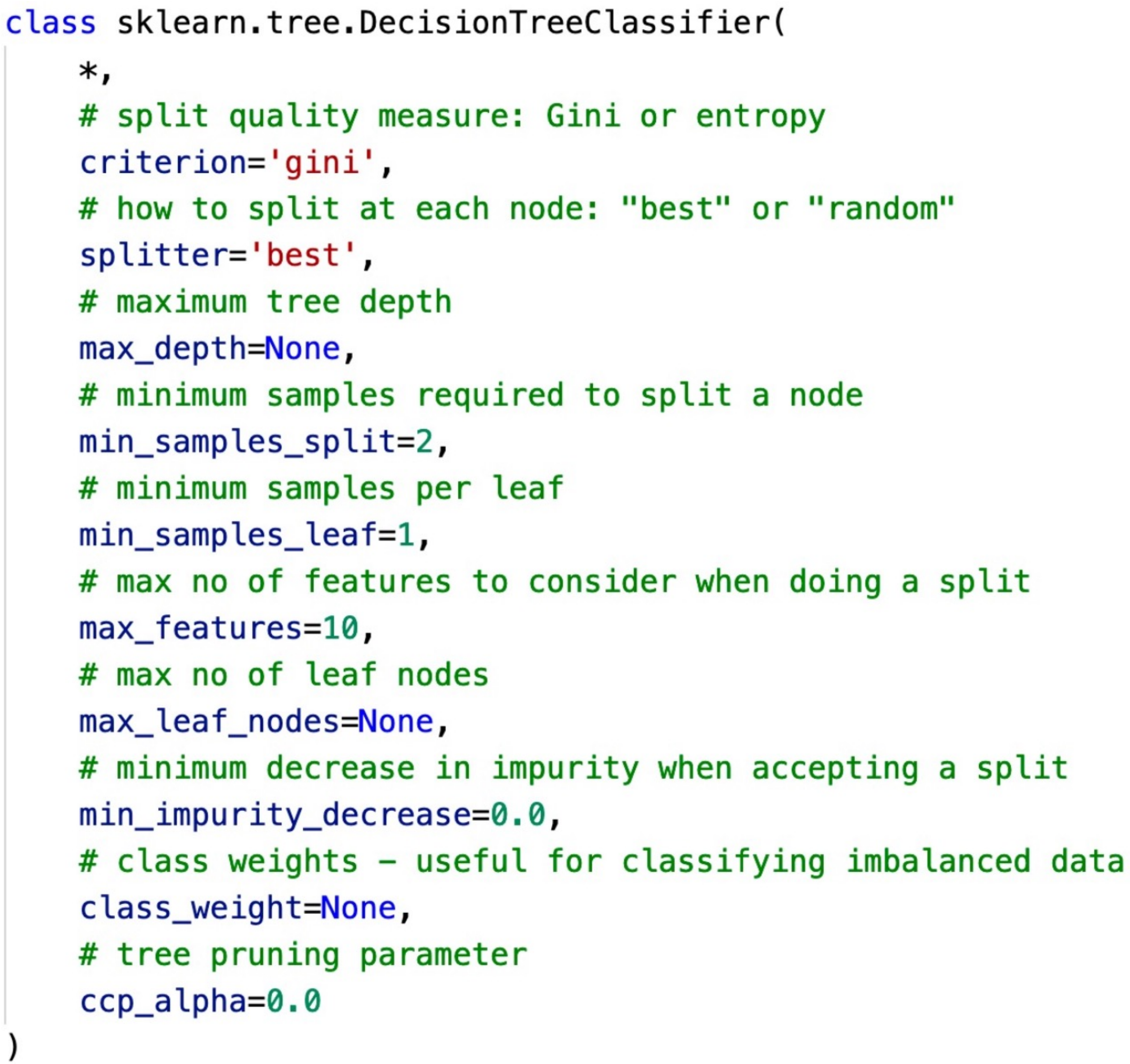
$$H(Q_m) = \frac{1}{n_m} \sum_{y \in Q_m} (y - \bar{y}_m)^2$$

Decision tree

- No need to normalise data.
- Easy to interpret
- Can handle numerical and categorical data

- High variance
- Easy to overfit (needs regularization)
- Scales poorly with large data or decision spaces.
- Not differentiable (can't optimize jointly with something else)
- Produce piecewise constant approximations, so can't extrapolate (can be strength or weakness).
- Can be biased if your (categorical) data are imbalanced





```
class sklearn.tree.DecisionTreeClassifier(
    *,
    # split quality measure: Gini or entropy
    criterion='gini',
    # how to split at each node: "best" or "random"
    splitter='best',
    # maximum tree depth
    max_depth=None,
    # minimum samples required to split a node
    min_samples_split=2,
    # minimum samples per leaf
    min_samples_leaf=1,
    # max no of features to consider when doing a split
    max_features=10,
    # max no of leaf nodes
    max_leaf_nodes=None,
    # minimum decrease in impurity when accepting a split
    min_impurity_decrease=0.0,
    # class weights - useful for classifying imbalanced data
    class_weight=None,
    # tree pruning parameter
    ccp_alpha=0.0
)
```

regularization

7

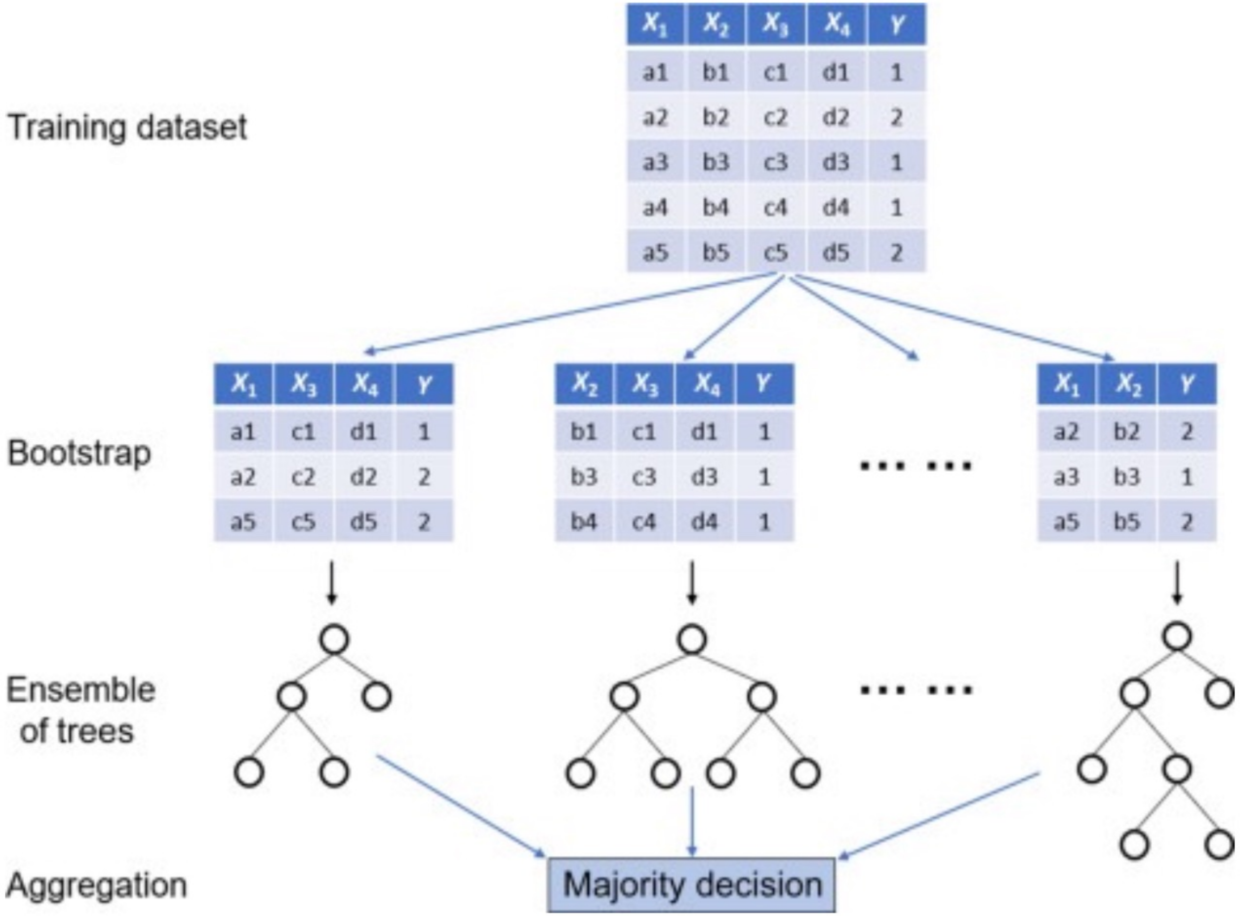
Random forests

Random forests

- Decision trees are very sensitive to small variations in training data.

- How to fix this? Build many trees.
  - On subsets of the inputs and subsets of examples.

- When predicting:
  - Average the solution of each tree to get answer.



X_1	X_2	X_3	X_4	Y
a1	b1	c1	d1	1
a2	b2	c2	d2	2
a3	b3	c3	d3	1
a4	b4	c4	d4	1
a5	b5	c5	d5	2

X_1	X_3	X_4	Y
a1	c1	d1	1
a2	c2	d2	2
a5	c5	d5	2

X_2	X_3	X_4	Y
b1	c1	d1	1
b3	c3	d3	1
b4	c4	d4	1

X_1	X_2	Y
a2	b2	2
a3	b3	1
a5	b5	2

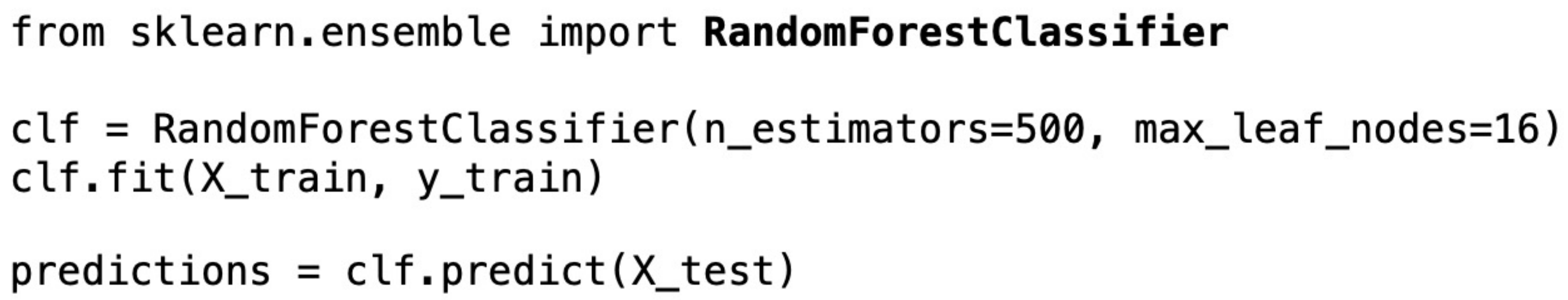
Bootstrapping + aggregation == ***bagging***

Figure from Wikipedia

How to train your tree model?

"My job is to make decisions.
Your job is to make them good decisions."

Building a Random Forest



```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16)
clf.fit(X_train, y_train)

predictions = clf.predict(X_test)
```

- *We're trading bias for variance (a common paradigm in machine learning!)*
- n_estimators == number of trees
- Many of the same choices as a decision tree.
- How can we optimise these parameters?

Hyperparameter optimization: randomized search with CV

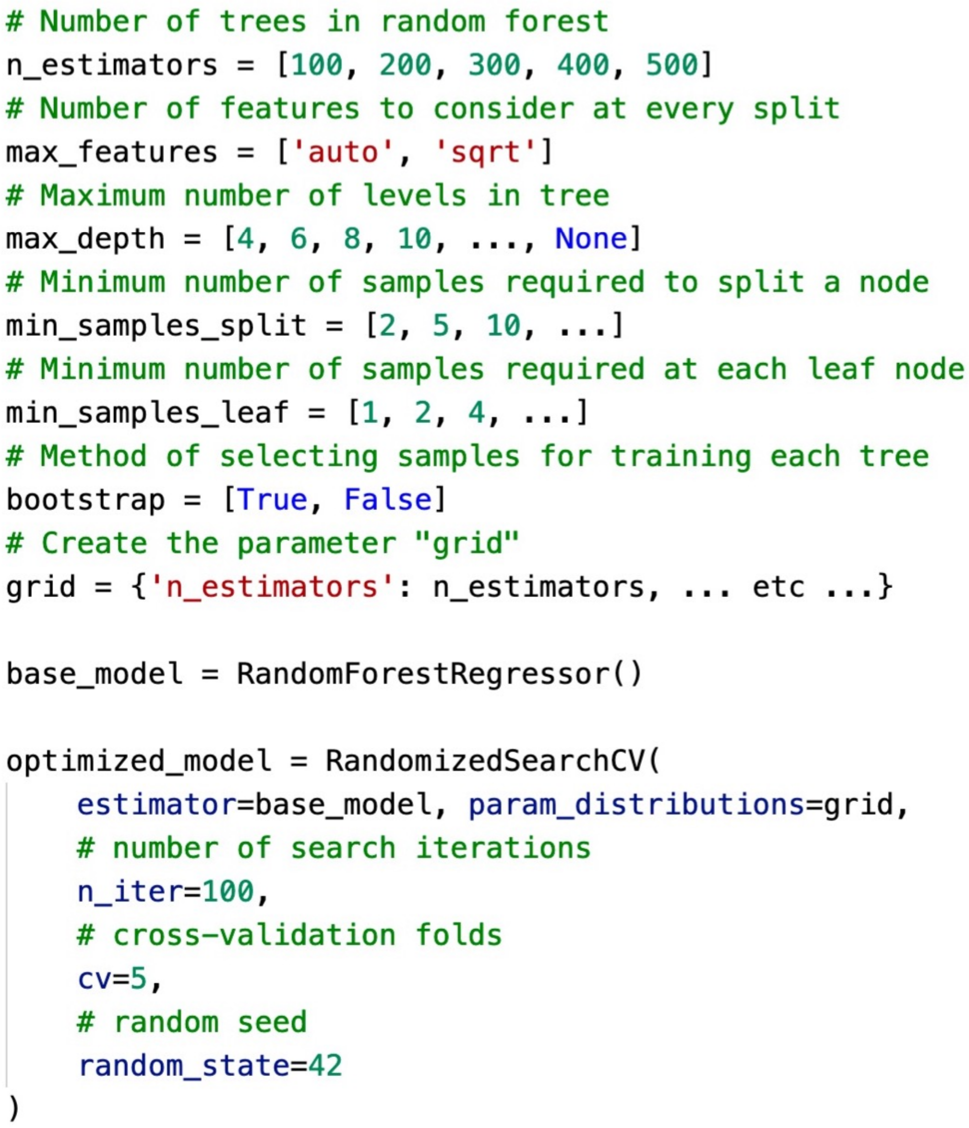
`sklearn.model_selection.`**`RandomizedSearchCV`**

- Random search over a grid of hyperparameters.

- *Cross-validation* is used to create subsets of the training data for training and evaluation.

- A separate **test set** should be generated **first**.

- *GridSearchCV* could be used to completely explore the space.

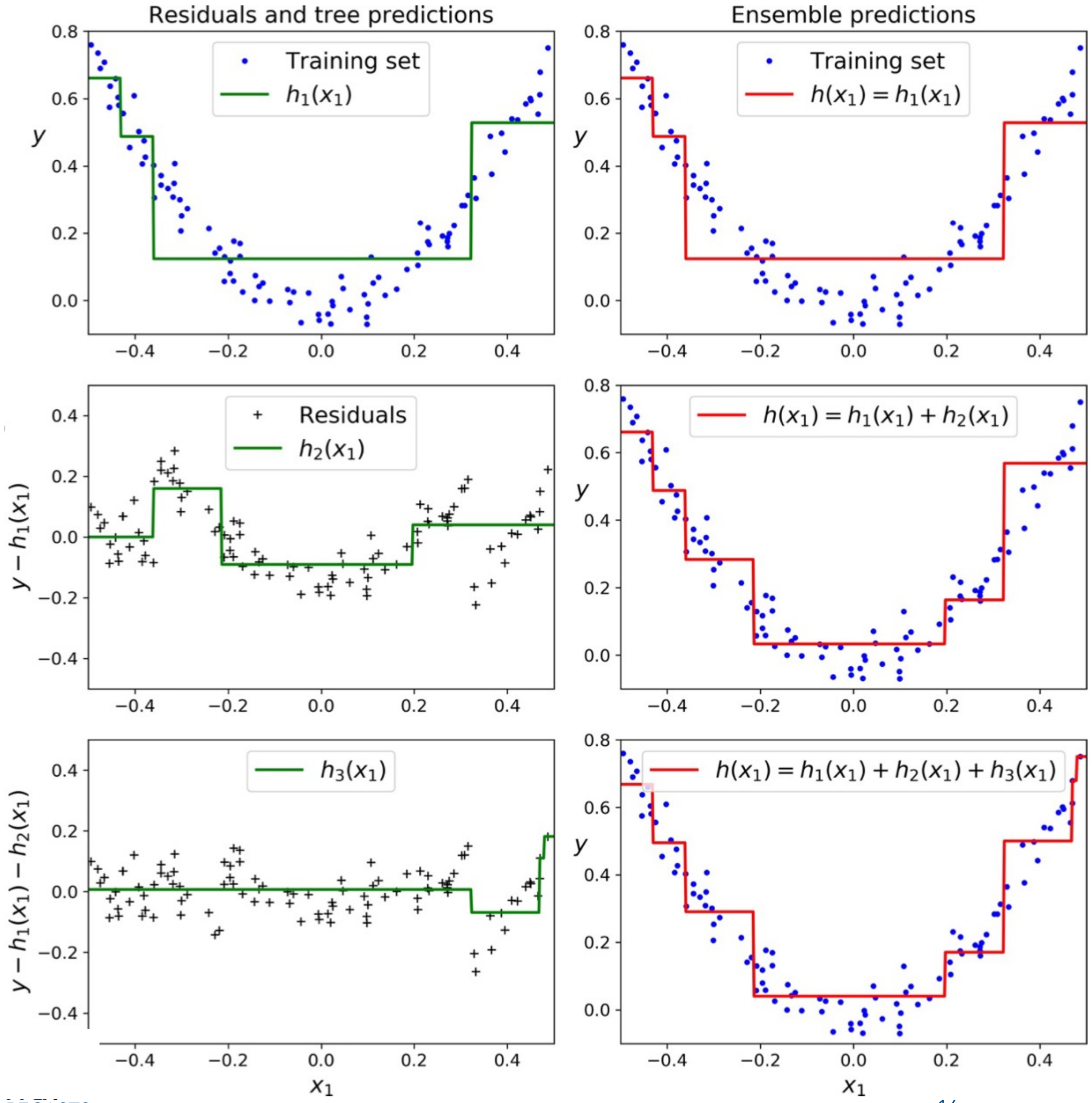


```
# Number of trees in random forest
n_estimators = [100, 200, 300, 400, 500]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [4, 6, 8, 10, ..., None]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, ...]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4, ...]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the parameter "grid"
grid = {'n_estimators': n_estimators, ... etc ...}

base_model = RandomForestRegressor()

optimized_model = RandomizedSearchCV(
    estimator=base_model, param_distributions=grid,
    # number of search iterations
    n_iter=100,
    # cross-validation folds
    cv=5,
    # random seed
    random_state=42
)
```

Gradient boosted trees

Gradient boosted trees

- Sequentially add trees to an ensemble.
  - Each correcting its predecessor.
  - The next tree fits the residual of the prior one.
- Random subsets of the training data are drawn for training each tree to regularise.
- Need care not to overfit.
- Important to have validation and test datasets.
  - Stop training when validation scores no longer improve.



From Geron 2019, chapter 7
Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

Gradient boosted trees

An extremely powerful technique!
Just look at the Kaggle challenges ☺

How to avoid overfitting:

- Early stopping (OOB performance)
- Tree regularization, e.g. max-depth, leaf count, …
- Adjust learning rate (control the contribution of each tree to the ensemble) – LR shrinkage
- Stochastic boosting: randomly subsample the fraction of training instances to be used when training each tree (again, we're trading bias against variance)

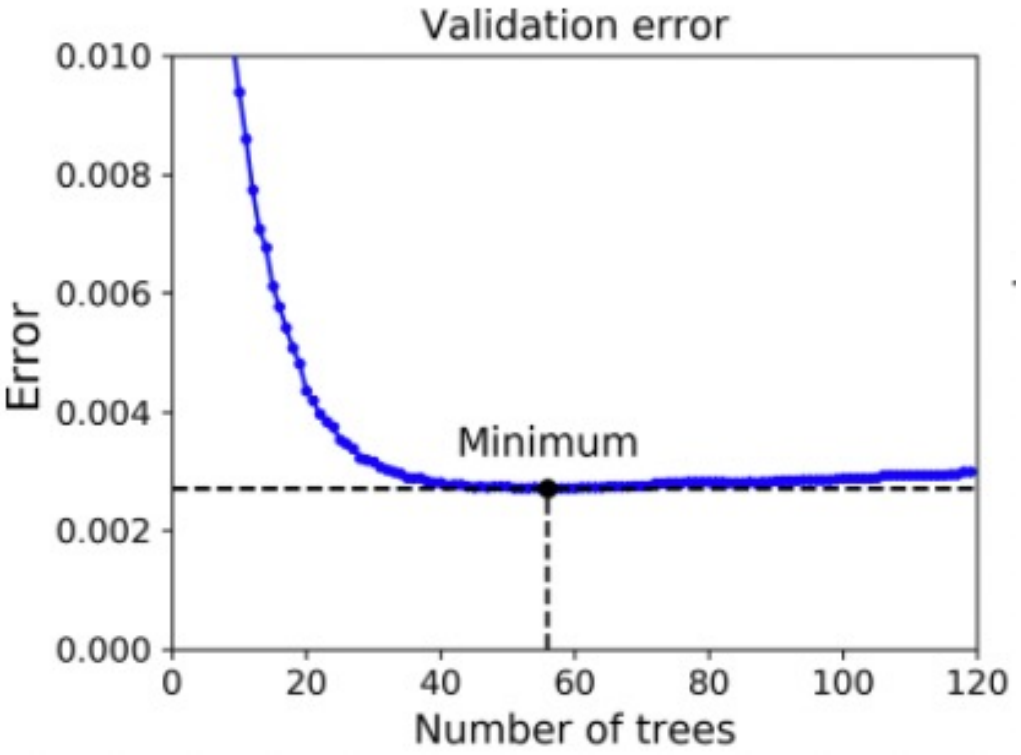
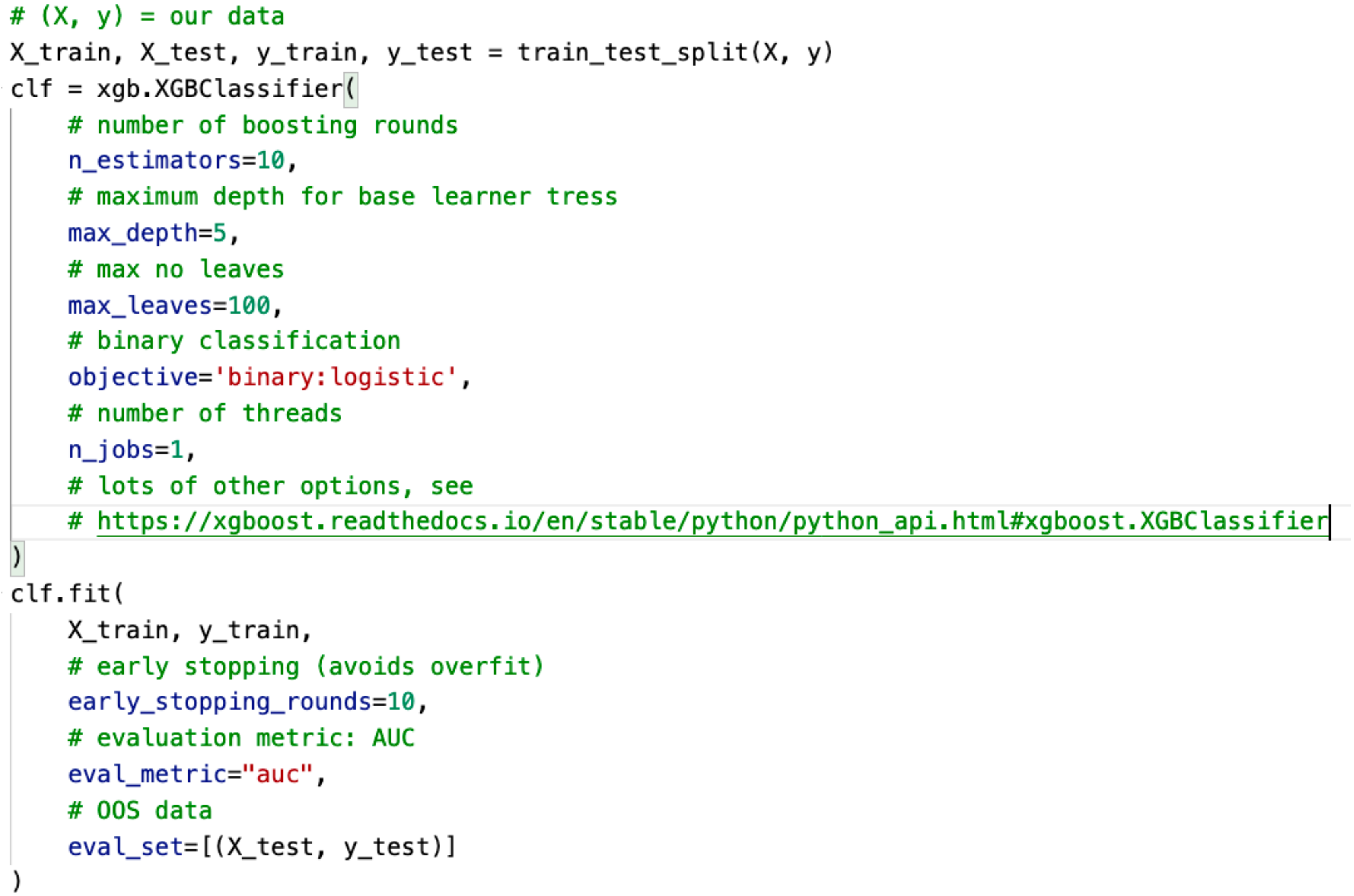


Figure from chapter 7 of (Geron, 2019)

LightGBM dmlc XGBoost scikit learn CatBoost

ECMWF

XGBoost

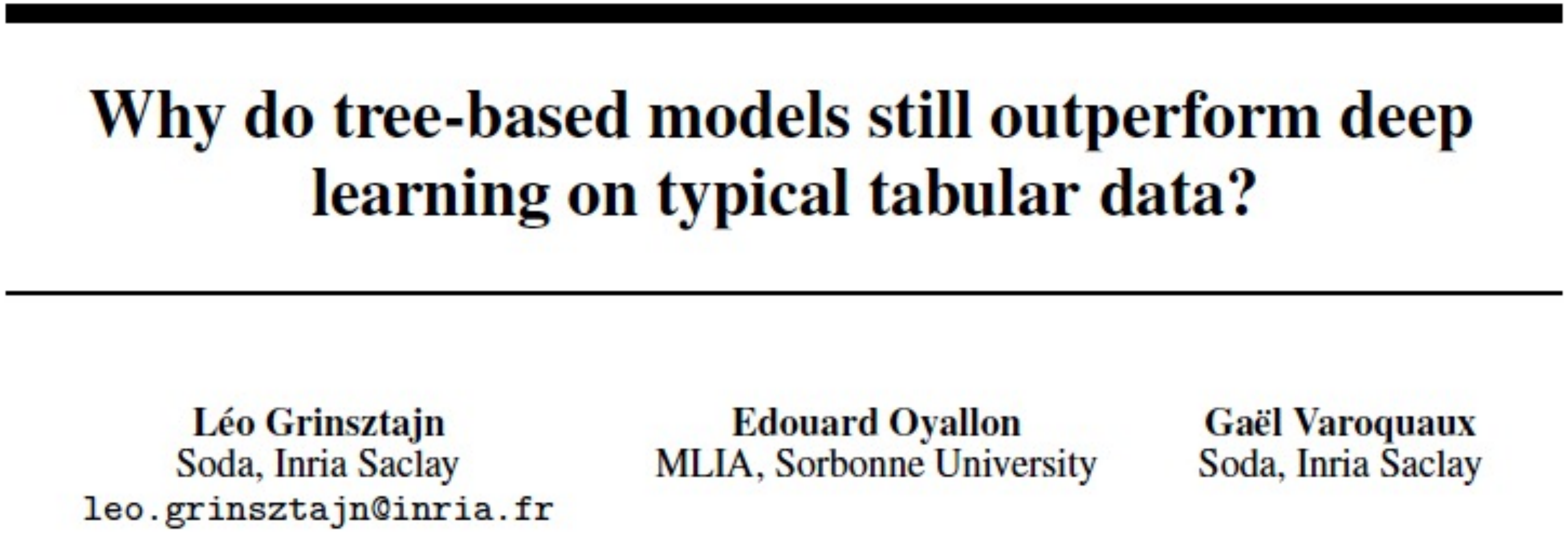


```
# (X, y) = our data
X_train, X_test, y_train, y_test = train_test_split(X, y)
clf = xgb.XGBClassifier(
    # number of boosting rounds
    n_estimators=10,
    # maximum depth for base learner tress
    max_depth=5,
    # max no leaves
    max_leaves=100,
    # binary classification
    objective='binary:logistic',
    # number of threads
    n_jobs=1,
    # lots of other options, see
    # https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.XGBClassifier
)
clf.fit(
    X_train, y_train,
    # early stopping (avoids overfit)
    early_stopping_rounds=10,
    # evaluation metric: AUC
    eval_metric="auc",
    # OOS data
    eval_set=[(X_test, y_test)]
)
```

https://github.com/dmlc/xgboost

https://xgboost.readthedocs.io/en/stable/tutorials/model.html

https://arxiv.org/abs/2207.08815



Why do tree-based models still outperform deep learning on typical tabular data?

Léo Grinsztajn
Soda, Inria Saclay
leo.grinsztajn@inria.fr

Edouard Oyallon
MLIA, Sorbonne University

Gaël Varoquaux
Soda, Inria Saclay

Best methods on tabular data: ensembles of decision trees (bagging or boosting)

Why?

Inductive biases of trees appear better suited to tabular data

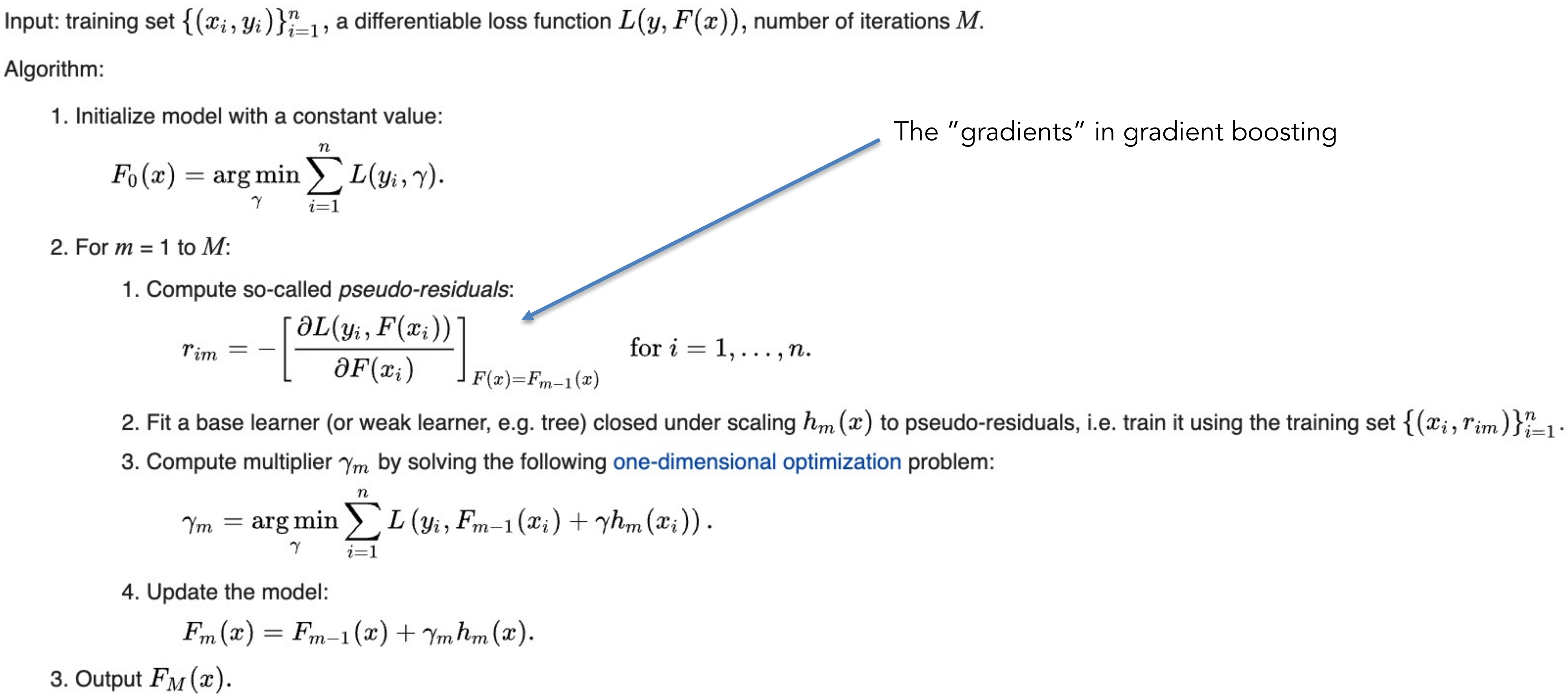
- NNs biased to overly smooth solutions
- NNs less robust to uninformative features
- Tree models are not rotationally invariant (unlike MLPs), as they attend to each feature separately

Summary

- Tree algos don’t (usually) make the headlines.
  - But on tabular data they should always be tested, most often beat neural networks.
- Interfaces are easy to use.
  - Scikit-learn has a standard interface for Regression/Decision tree/Random forest.
  - XGBoost, CatBoost, LightGBM: GPU support
- Robust models can be built on small datasets.
- Decision trees/random forest/gradient-boosted trees are all very capable of overfitting.
  - Vital to have good data hygiene.
  - Truly independent training/validation/test sets.

Extra slides

Gradient boosting: pseudocode



```
Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations $M$.

Algorithm:

  1. Initialize model with a constant value:
       $F_0(x) = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L(y_i, \gamma).$
  2. For $m$ = 1 to $M$:
       1. Compute so-called *pseudo-residuals*:
            $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$
       2. Fit a base learner (or weak learner, e.g. tree) closed under scaling $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
       3. Compute multiplier $\gamma_m$ by solving the following one-dimensional optimization problem:
            $\gamma_m = \underset{\gamma}{\arg\min} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$
       4. Update the model:
            $F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$
  3. Output $F_M(x)$.
```

The "gradients" in gradient boosting

https://arxiv.org/abs/2207.08815

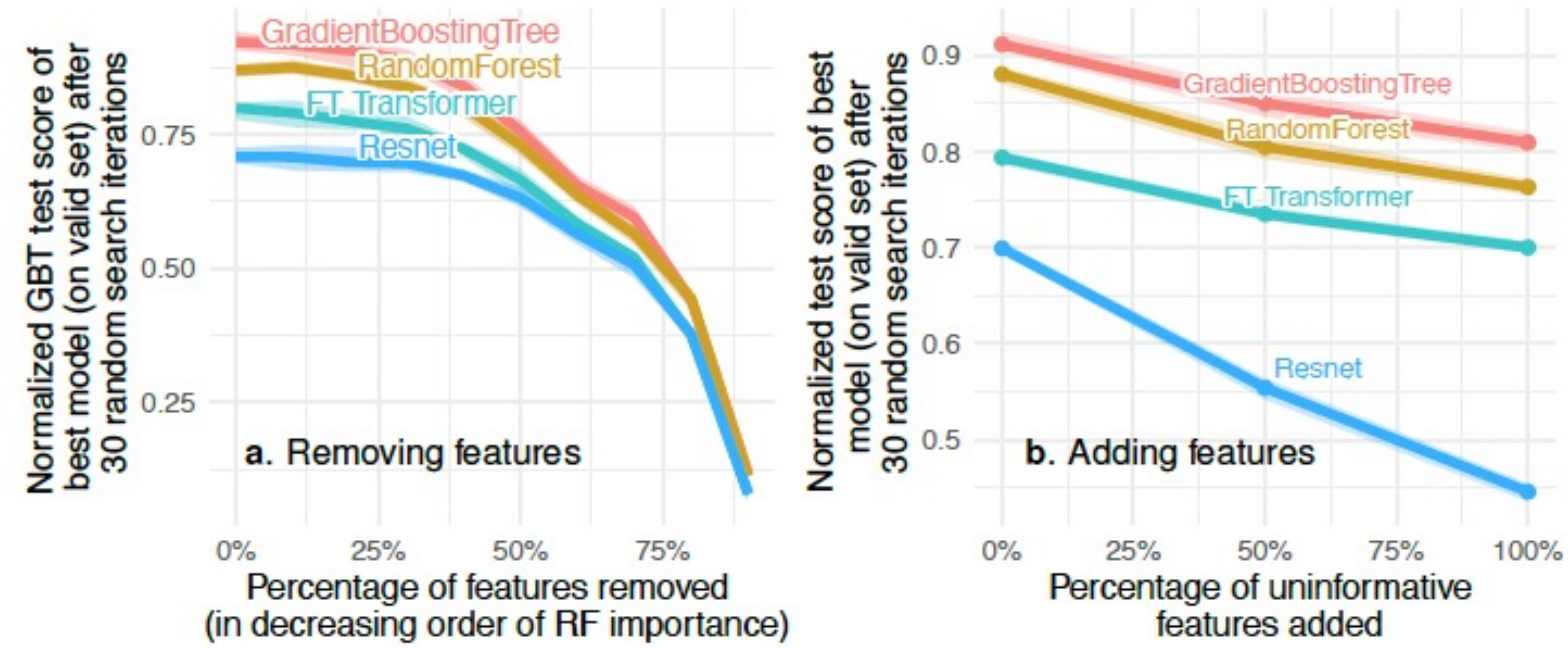


Figure 4: **Test accuracy changes when removing (a) or adding (b) uninformative features.** Features are removed in increasing order of feature importance (computed with a Random Forest). Added features are sampled from standard Gaussians uncorrelated with the target and with other features. Scores are averaged across datasets, and the ribbons correspond to the minimum and maximum score among the 30 different random search reorders (starting with the default models).

https://arxiv.org/abs/2207.08815

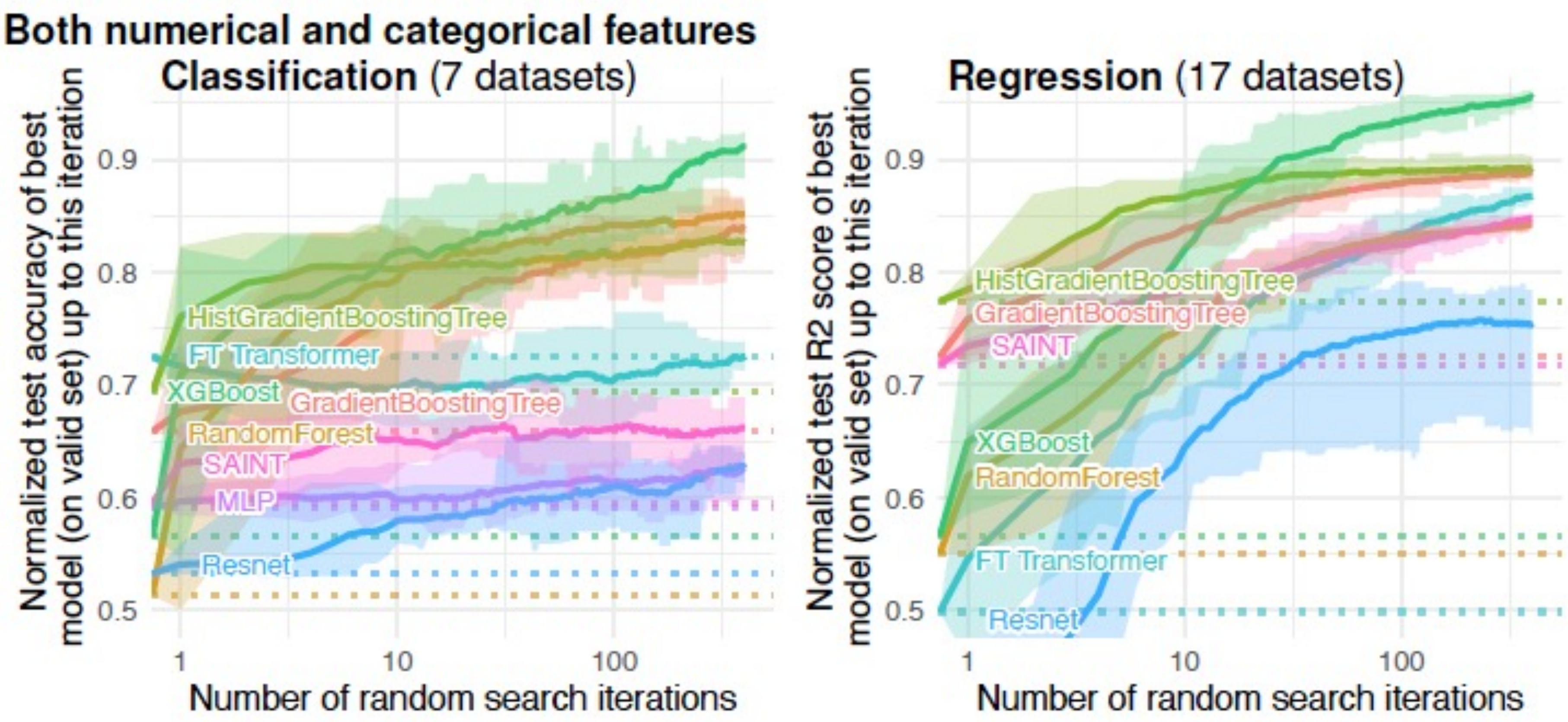
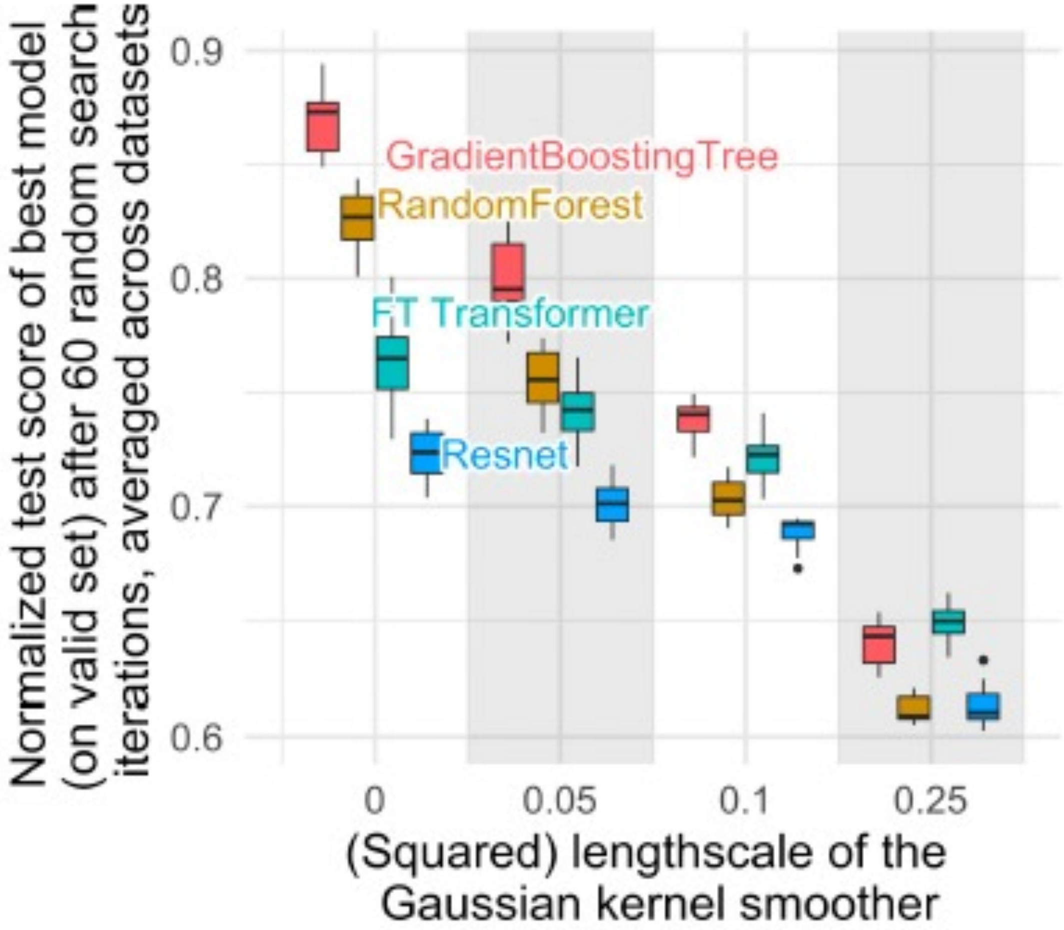


Figure 1: **Benchmark on medium-sized datasets**, top only numerical features; bottom: all features. Dotted lines correspond to the score of the default hyperparameters, which is also the first random search iteration. Each value corresponds to the test score of the best model (on the validation set) after a specific number of random search iterations, averaged on 15 shuffles of the random search order. The ribbon corresponds to minimum and maximum scores on these 15 shuffles.

https://arxiv.org/abs/2207.08815



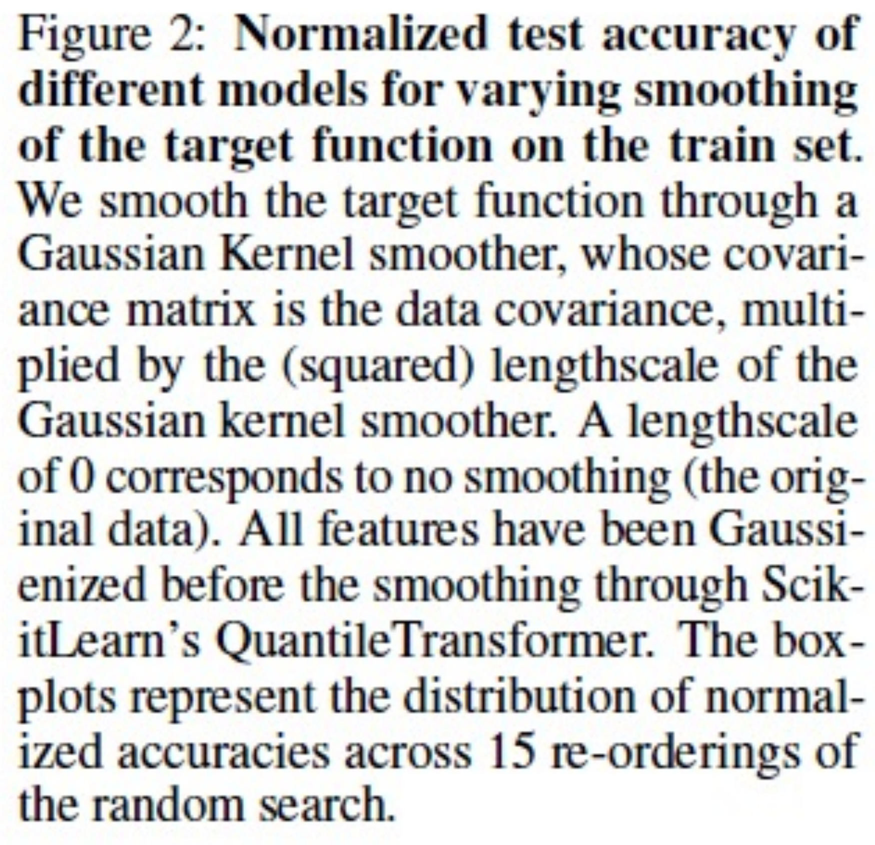


Figure 2: **Normalized test accuracy of different models for varying smoothing of the target function on the train set.** We smooth the target function through a Gaussian Kernel smoother, whose covariance matrix is the data covariance, multiplied by the (squared) lengthscale of the Gaussian kernel smoother. A lengthscale of 0 corresponds to no smoothing (the original data). All features have been Gaussienized before the smoothing through ScikitLearn's QuantileTransformer. The boxplots represent the distribution of normalized accuracies across 15 re-orderings of the random search.