Tree models

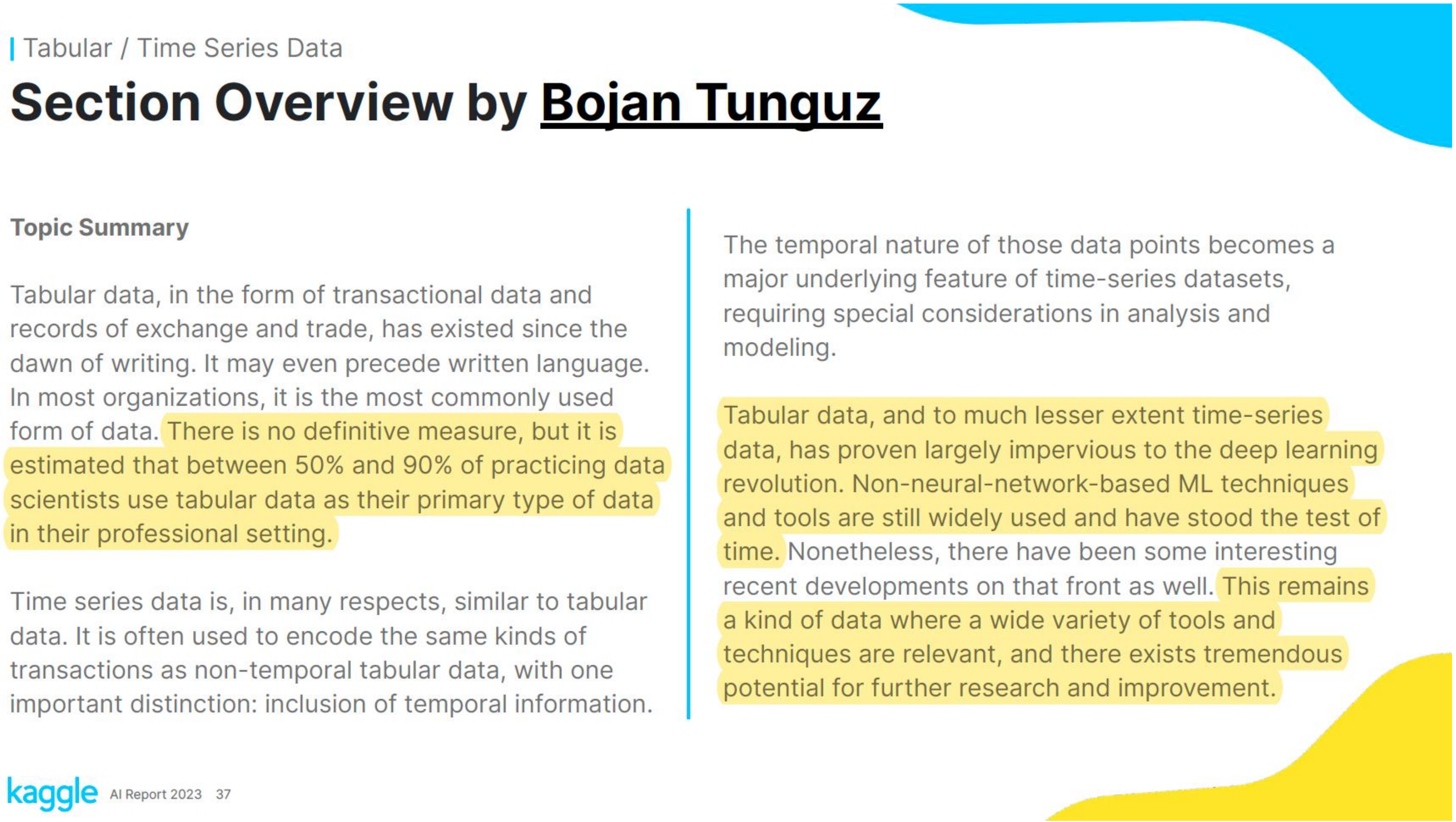
Random forests and boosting

Dr. William Becker

Machine Learning training coordinator, ECMWF

Thanks to Mihai, Mat and others…

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://www.kaggle.com/AI-Report-2023

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.

What’s **not** tabular data?

- Spatiotemporal data (where ordering, correlations important)
- Images, video, etc.

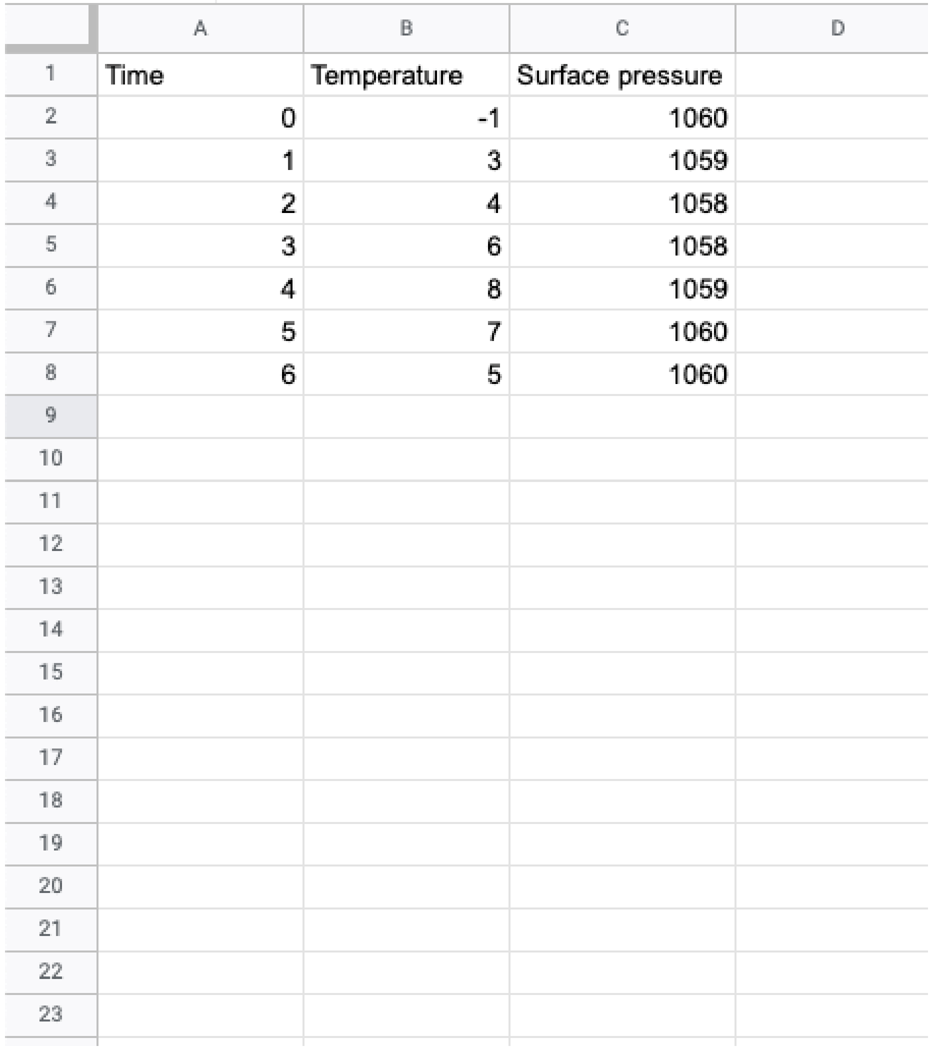
When is earth-system data tabular?

- When the temporal and spatial components of the problem are not important.
- e.g. correcting the weather forecast for your house.

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

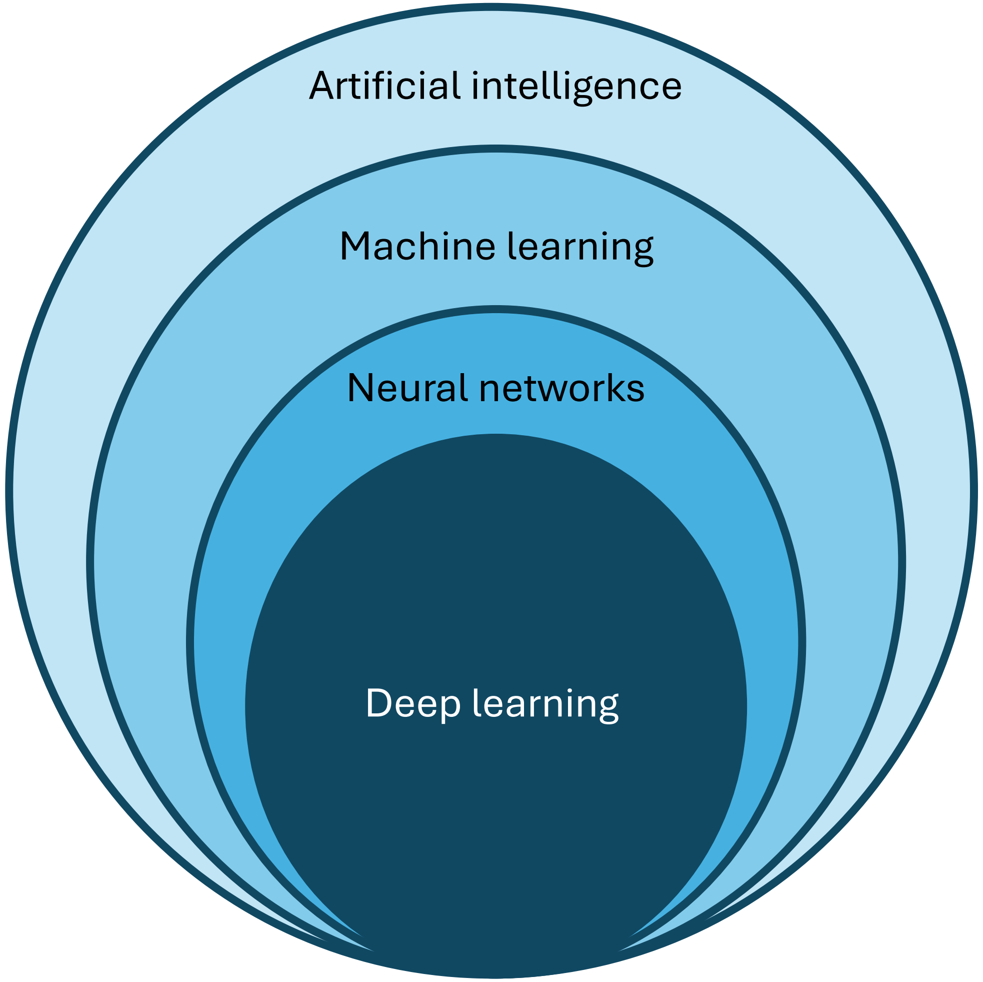
Predictors/predictands

Examples

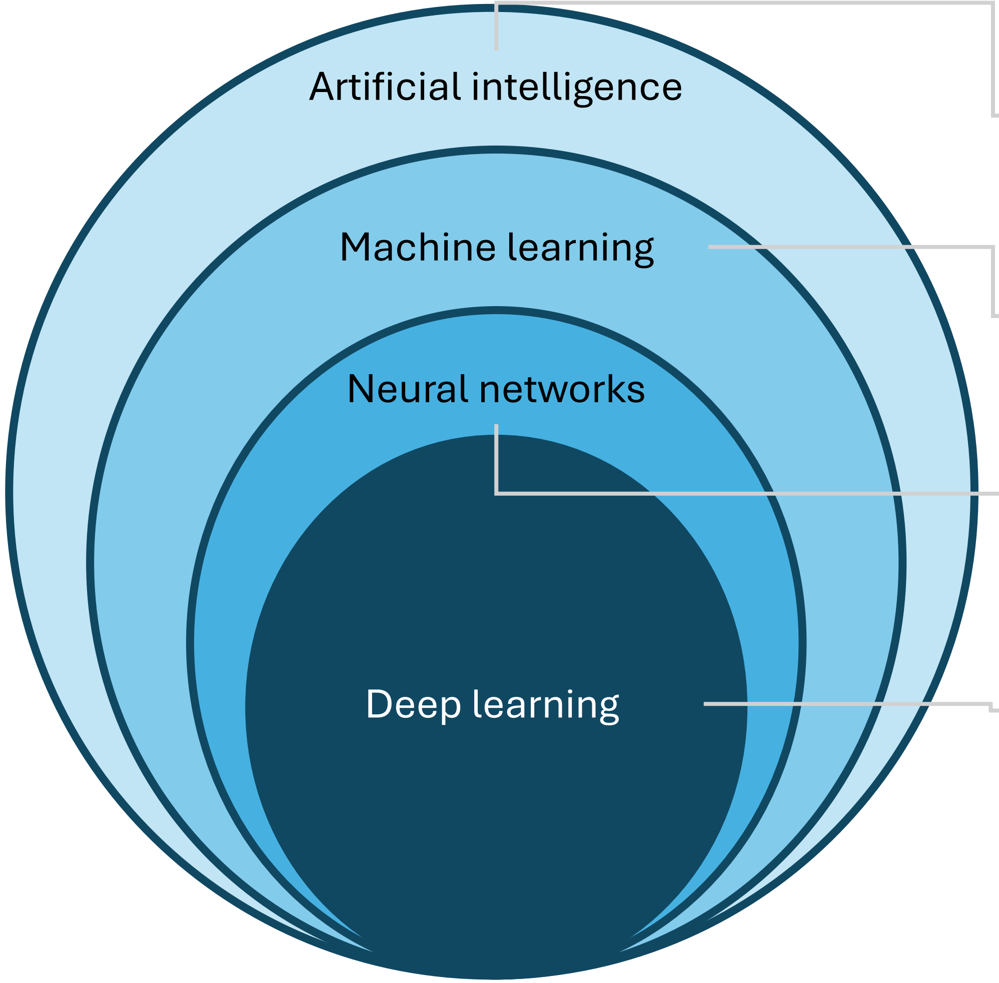


	A	B	C	D
1	Time	Temperature	Surface pressure	
2	0	-1	1060	
3	1	3	1059	
4	2	4	1058	
5	3	6	1058	
6	4	8	1059	
7	5	7	1060	
8	6	5	1060	
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				

Decision trees: where are we?



Decision trees: where are we?



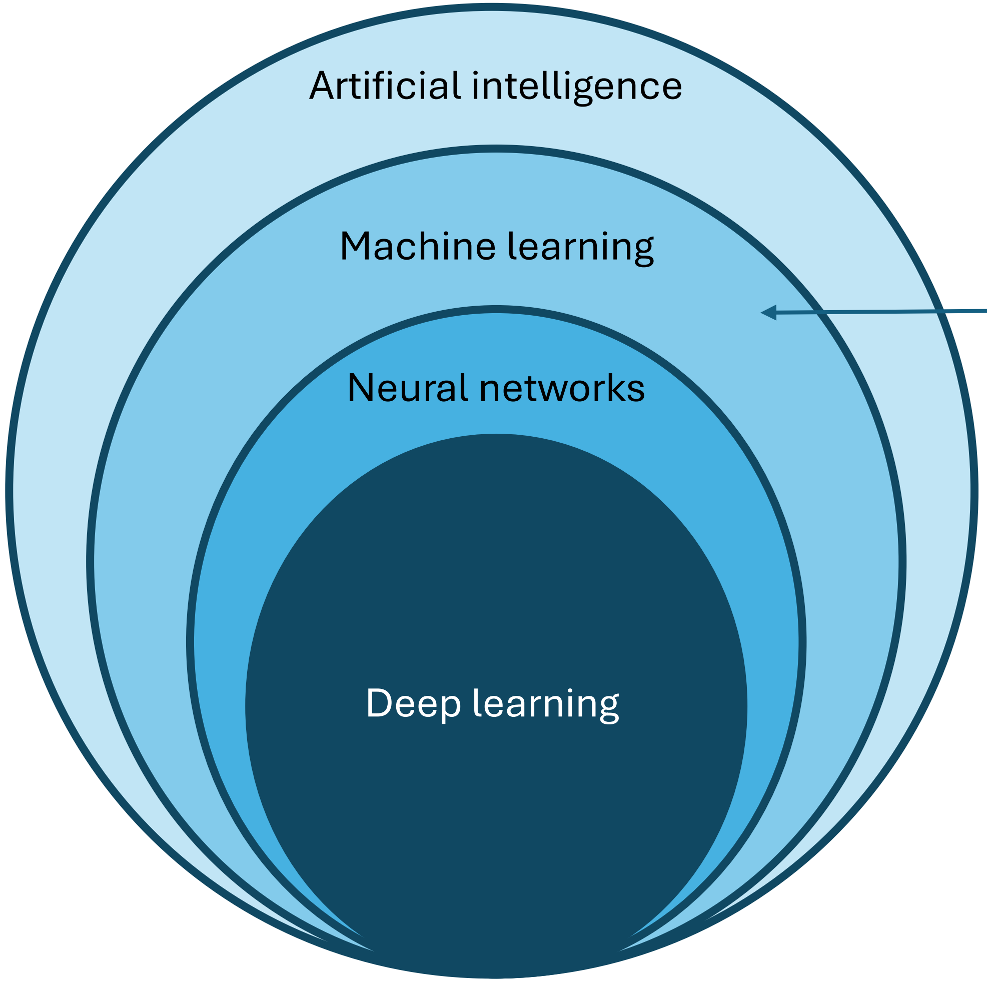
Performing tasks typically associated with human intelligence, such as learning, reasoning, problem-solving, perception, and decision-making

Algorithms that can learn from data and generalise to unseen data, and thus perform tasks without explicit instructions.

Computational models inspired by the structure and functions of biological neural networks

The use of neural networks with multiple hidden layers of neurons (up to possibly thousands).

Where are we?

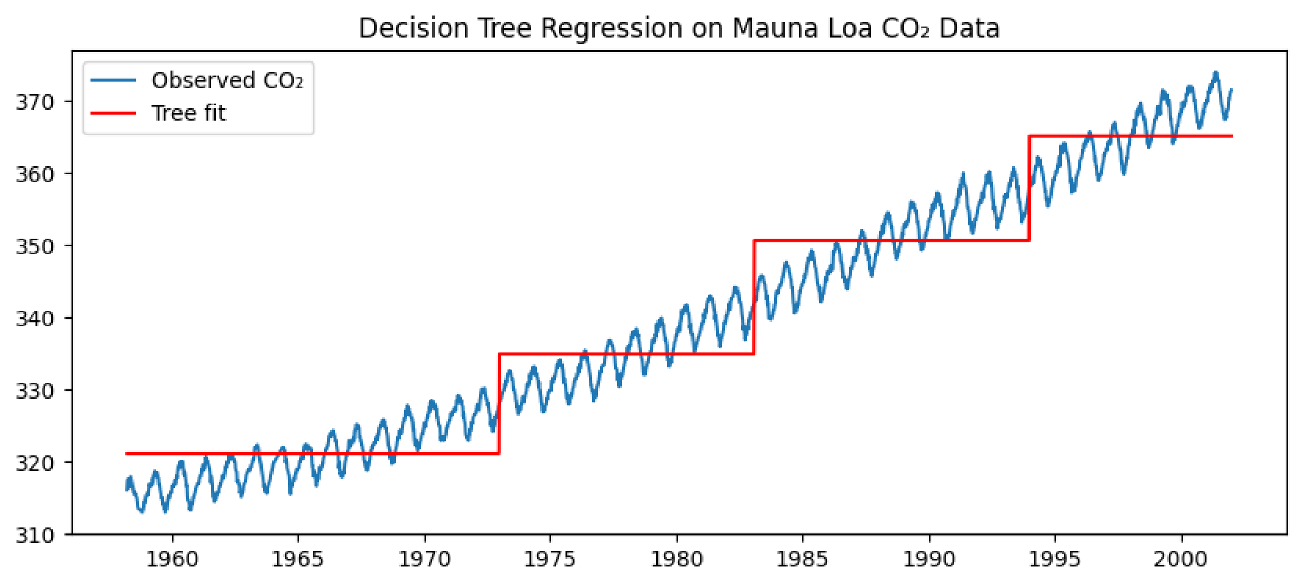


Tree models are:

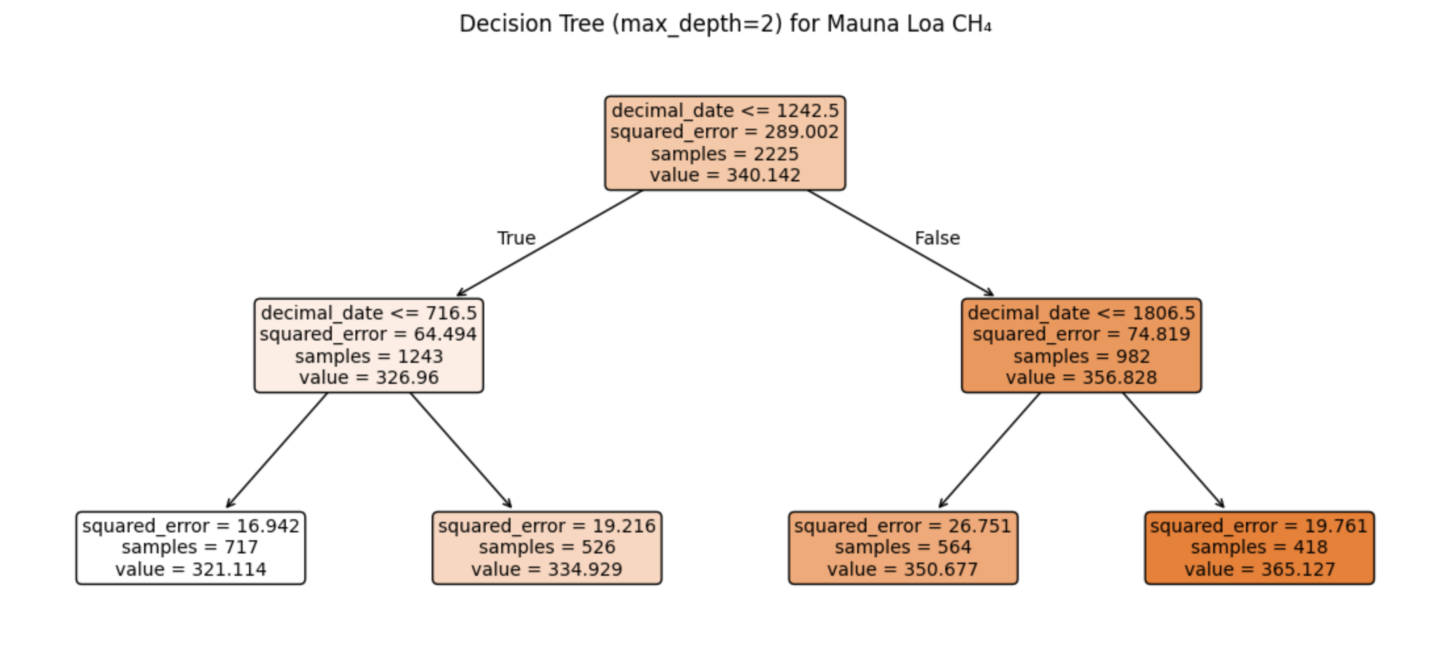
- ML methods, but not neural networks
- Supervised learning method (learning relationships between features and target variable(s))

Decision trees

The building blocks



Decision Tree Regression on Mauna Loa CO_2 Data



Decision Tree (max_depth=2) for Mauna Loa CH_4

- Feature space is recursively partitioned.
- Each “leaf” is assigned a constant value.
- On multidimensional problems it is partitioned one variable/feature at a time.

Where to split?

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 (over the two new regions)
  - Greedy algorithm!

- For **classification**, calculate probability of being a class for the samples in the branch.
  - Gini impurity $H(Q_m) = \sum_k p_{mk}(1 - p_{mk})$
  - Log loss/entropy $H(Q_m) = -\sum_k p_{mk}\log(p_{mk})$

$$p_{mk} = \frac{1}{n_m} \sum_{y \in Q_m} I(y = k)$$

Proportion of observations in class *k* at node *m*.

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

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

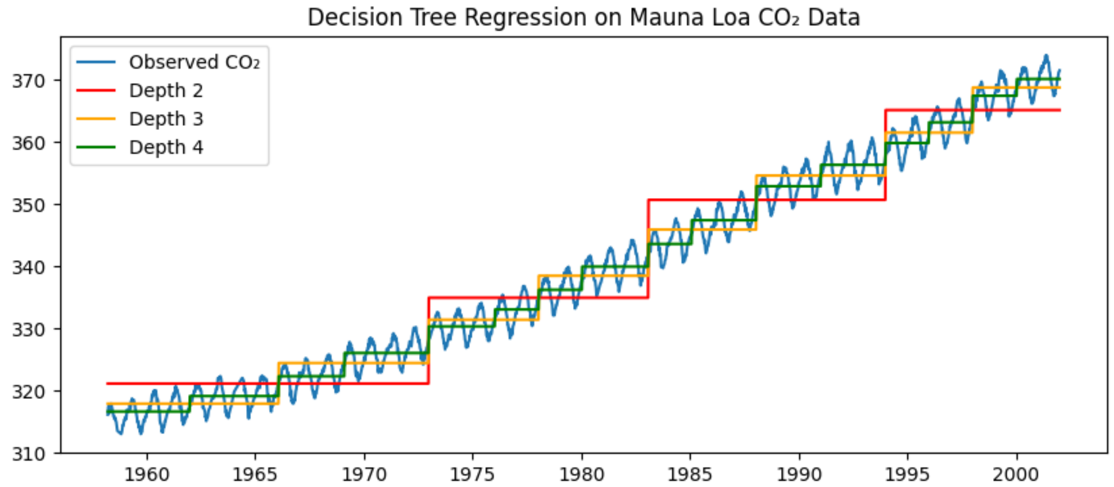
$$H(Q_m) = \frac{1}{n_m} \sum_{y \in Q_m} (y - \bar{y}_m)^2$$

Implementation

https://scikit-learn.org/stable/#

Very straightforward in scikit-learn.

```
# Fit decision tree
tree =
DecisionTreeRegressor(max_depth=2,
random_state=0)
tree.fit(X, y)
```



Hyperparameter	Meaning	Effect / Use
max_depth	Maximum depth of the tree (number of splits from root to leaf).	Controls **model complexity**. Smaller = smoother fit (less overfitting).
min_samples_split	Minimum number of samples required to split an internal node.	Prevents overly fine splits on small sample subsets.
min_samples_leaf	Minimum number of samples required to be in a leaf node.	Ensures leaves have enough data; helps **smooth predictions**.
max_leaf_nodes	Limits the total number of leaf nodes.	Another way to constrain tree size and overfitting.

Reminder:

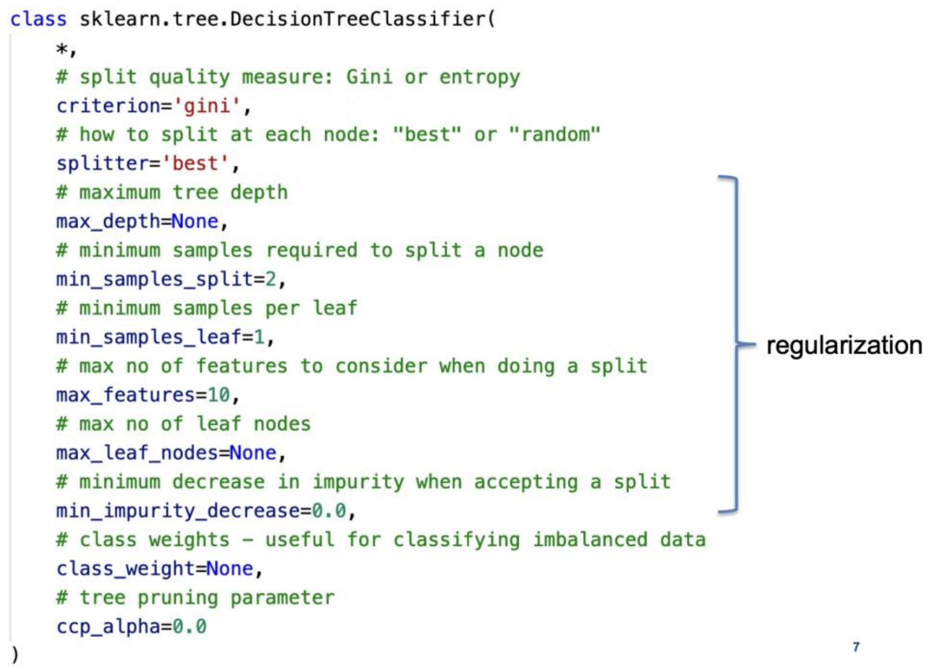
- *Hyperparameters* are model specifications that we set.
- *Parameters* are learned from the training data.

https://github.com/ecmwf-training/2025-ml-training/blob/main/1-ml-basics/decision_trees_mauna_loa.ipynb

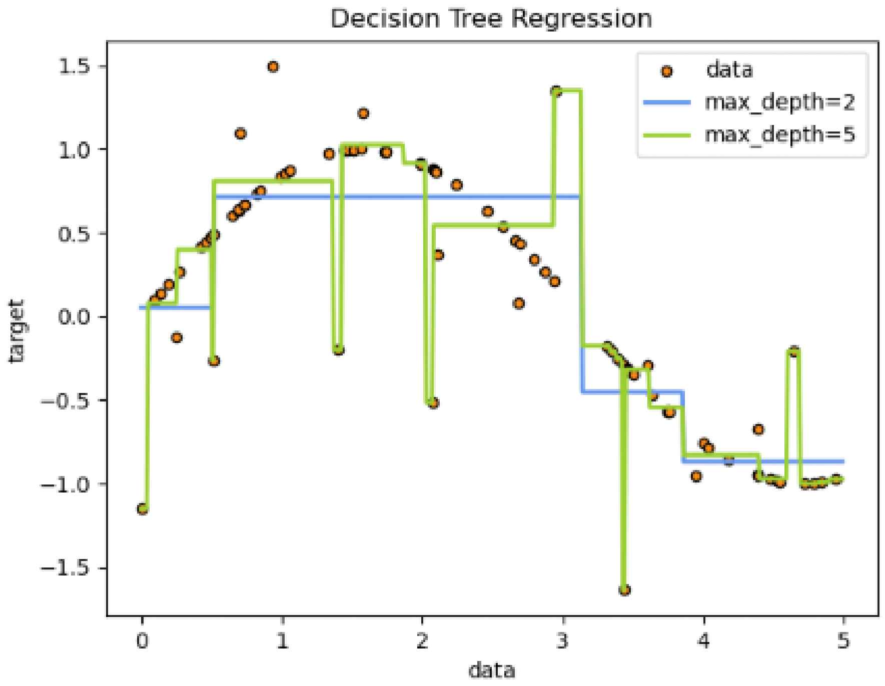
Pros/cons of decision trees

The Good

- No need to normalise data.
- Easy to interpret
- Can handle nonlinear and interacting features
- Can handle numerical and categorical data
- Fairly quick to train on small/medium datasets



```
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
)
```



The Bad

- High variance, easy to overfit (needs regularization)
- Small changes in data lead to different trees. Outliers/noise not handled well.
- Scales poorly with large data or decision spaces.
- Produce piecewise constant approximations, so can't extrapolate (can be strength or weakness).

Random forests

Strength in numbers

Random forests

Machine Learning, 45, 5–32, 2001
© 2001 Kluwer Academic Publishers. Manufactured in The Netherlands.

Random Forests

LEO BREIMAN
Statistics Department, University of California, Berkeley, CA 94720

Editor: Robert E. Schapire

165k citations!

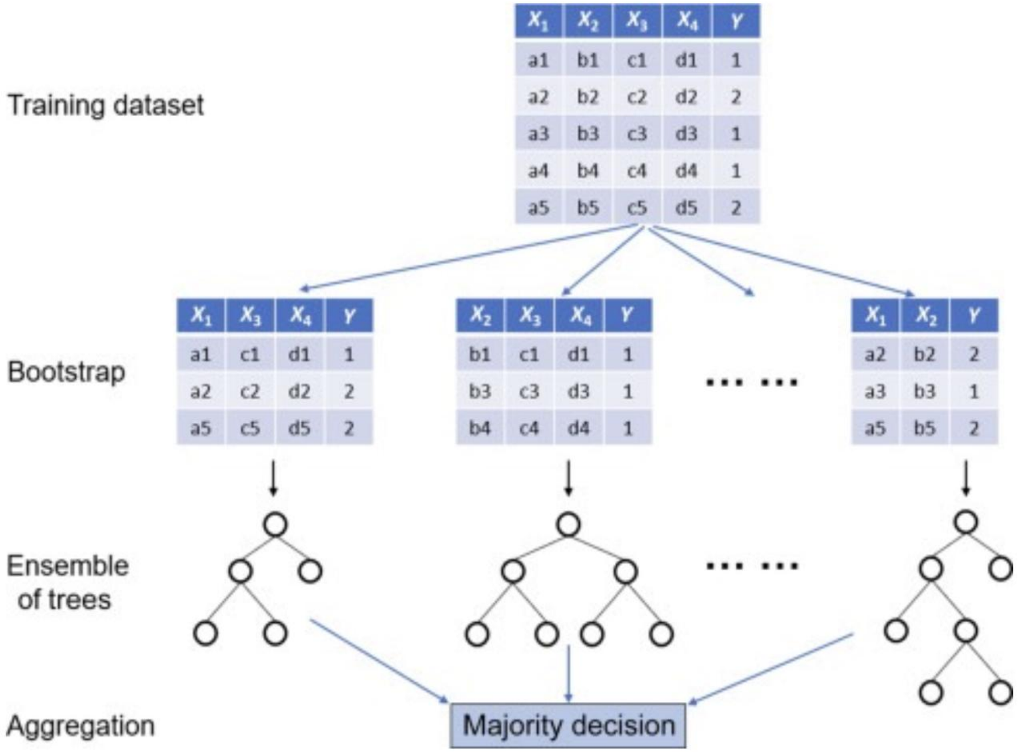
Decision trees intuitive and flexible, but are very sensitive to small variations in training data – they are “high variance learners”.

How to fix this? Build many trees, using:

- Subsets of the training data (bootstrapping)
- Subsets of the features at each split

When predicting:

- **Average the solution** of each tree to get answer – reduces bias
- Bootstrapping + aggregation = *bagging*
- Conceptually similar to ensemble weather forecasting
- “Wisdom of the crowd”



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

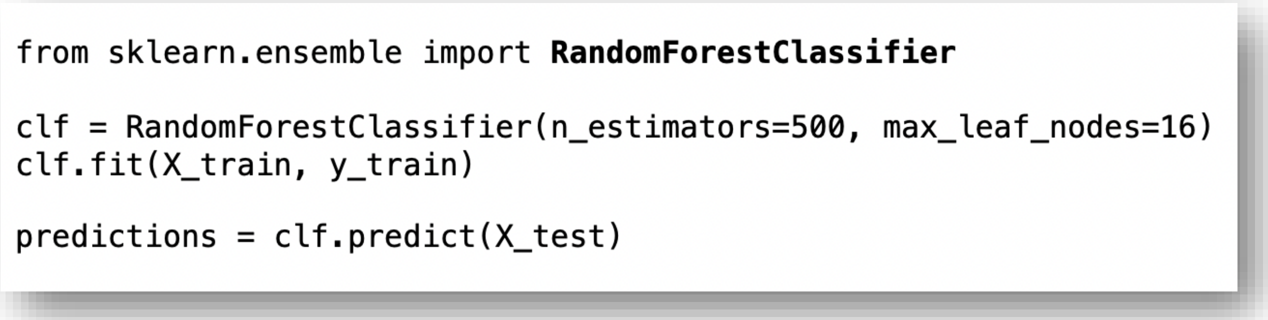
Bootstrapping + aggregation == ***bagging***

Figure from Wikipedia

Implementation

Random Forests are also implemented in scikit-learn, with a number of options (hyperparameters).

```
class sklearn.ensemble.RandomForestClassifier(
      n_estimators=100, *,
      criterion='gini’,
      max_depth=None,
      min_samples_split=2,
      min_samples_leaf=1,
      min_weight_fraction_leaf=0.0,
      max_features='sqrt’,
      max_leaf_nodes=None,
      min_impurity_decrease=0.0,
…)
```



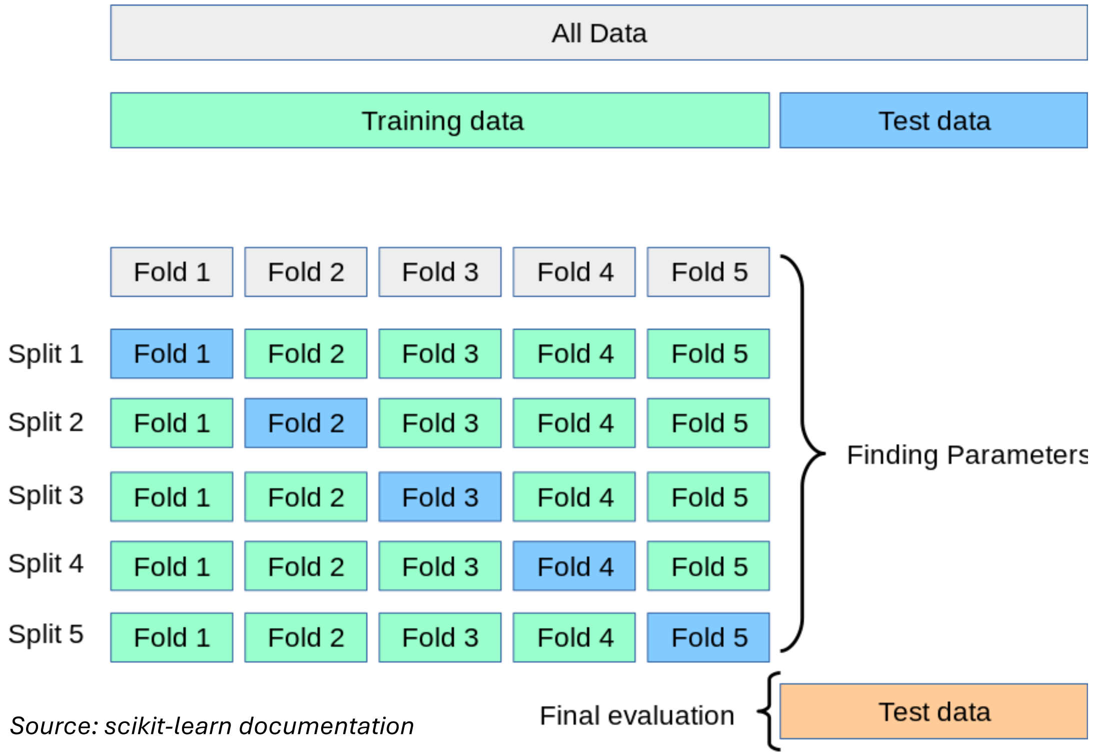
```
from sklearn.ensemble import RandomForestClassifier

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

predictions = clf.predict(X_test)
```

Can we optimise these hyperparameters?

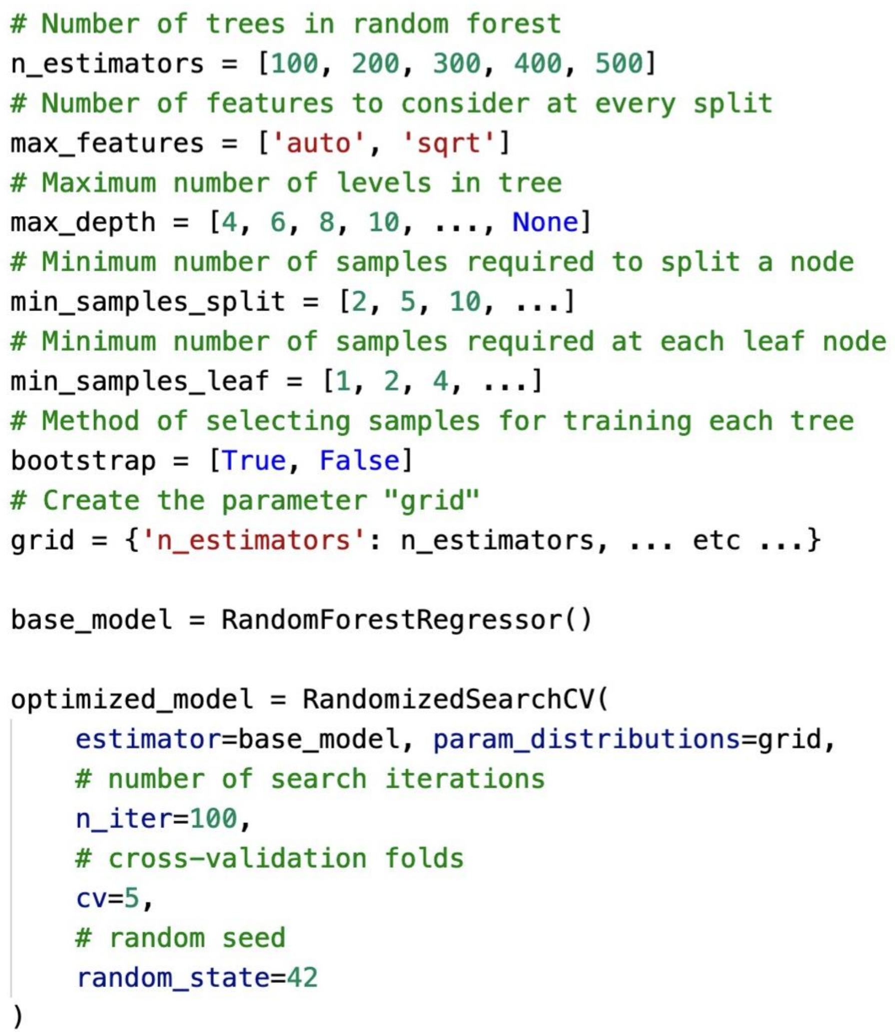
Hyperparameter optimization

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

Performs a randomized search across a specified hyperparameter grid, using k-fold cross validation at each iteration to generate average evaluation metrics. Use test set for final evaluation.



Source: scikit-learn documentation



```
# 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
)
```

For exhaustive exploration of the hyperparameter space, *GridSearchCV* may be employed.

Pros/cons of random forests

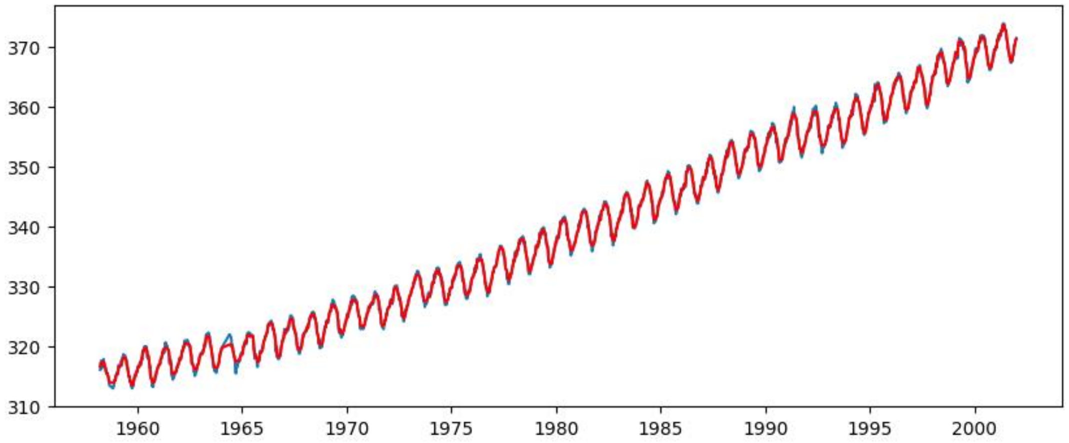
Advantages

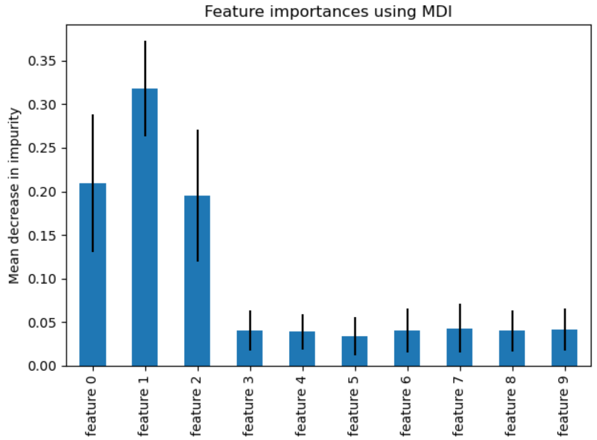
- Automatically manages nonlinear relationships and feature interactions.
- Demonstrates strong resilience to noise and overfitting.
- Performs effectively with diverse types of features.
- Offers estimates of feature importance (mean decrease in impurity)
- Supports parallel processing and is highly scalable.

Limitations

- Less transparent and harder to interpret than individual decision trees.
- Requires a considerable amount of memory for large ensembles.
- Prediction speed is slower compared to linear models.
- Struggles to extrapolate beyond the scope of the training data.
- Feature importance measures may be biased when features vary in scale or type.

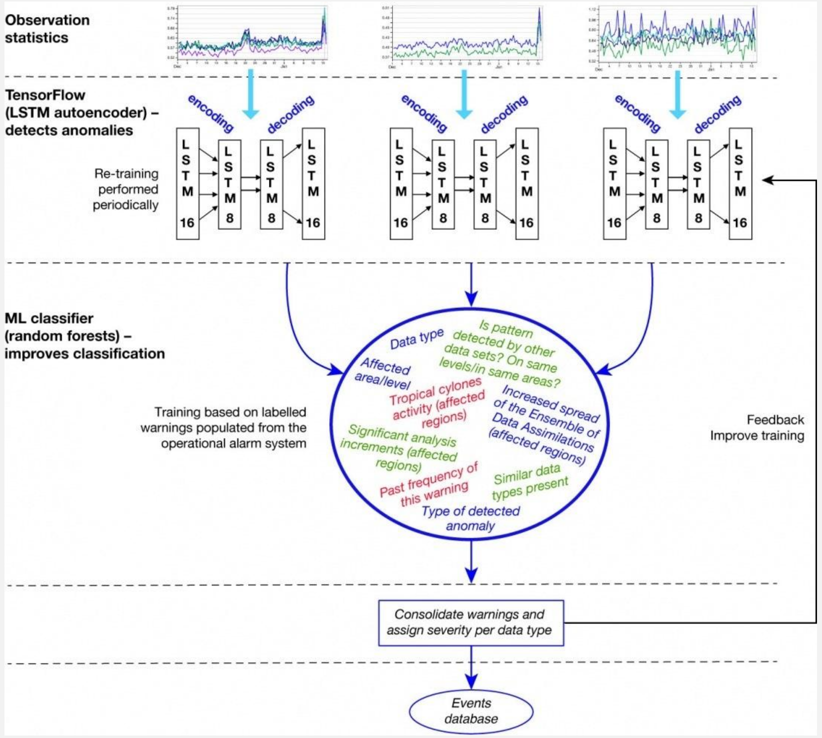
```
rf = RandomForestRegressor(n_estimators=100, max_depth=10,
random_state=42)
rf.fit(X, y)
```





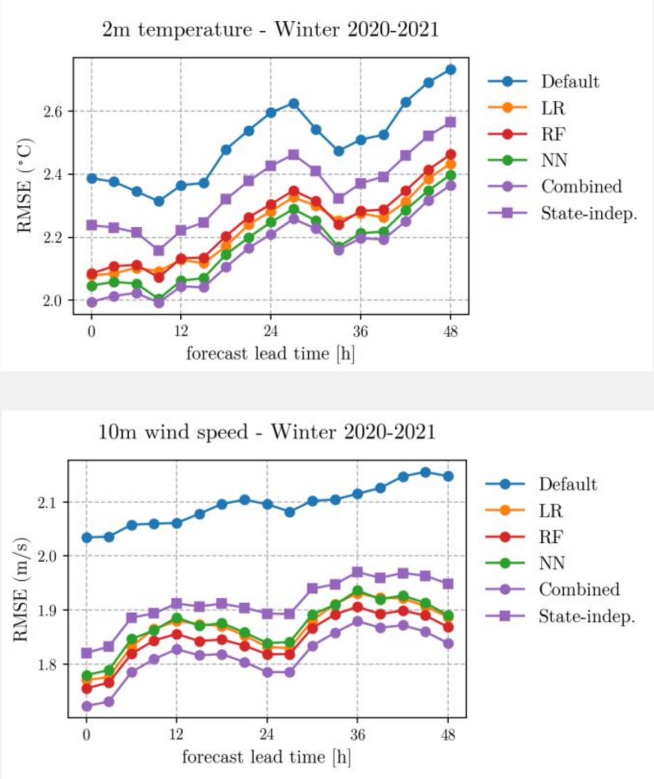
Some applications

Detection and classification of observation anomalies


Observation statistics
TensorFlow (LSTM autoencoder) – detects anomalies
encoding
decoding
LSTM 16
LSTM 8
LSTM 8
LSTM 16
Re-training performed periodically
ML classifier (random forests) – improves classification
Training based on labelled warnings populated from the operational alarm system
Data type
Is pattern detected by other data sets? On same levels/in same areas?
Affected area/level
Tropical cyclones activity (affected regions)
Increased spread of the Ensemble of Data Assimilations (affected regions)
Significant analysis increments (affected regions)
Past frequency of this warning
Similar data types present
Type of detected anomaly
Feedback Improve training
Consolidate warnings and assign severity per data type
Events database

https://www.ecmwf.int/en/newsletter/174/earth-system-science/use-machine-learning-detection-and-classification

Post-processing forecast errors



Bouallègue, Z. B., Cooper, F., Chantry, M., Düben, P., Bechtold, P., & Sandu, I. (2023). Statistical modeling of 2-m temperature and 10-m wind speed forecast errors. *Monthly Weather Review*, *151*(4), 897-911.

Gradient-boosted tree ensembles

Learning from mistakes

Gradient boosted trees

Sequentially add trees to an ensemble.

- Each correcting its predecessor.
- The next tree fits the residual of the prior one.
- It's all a bit meta

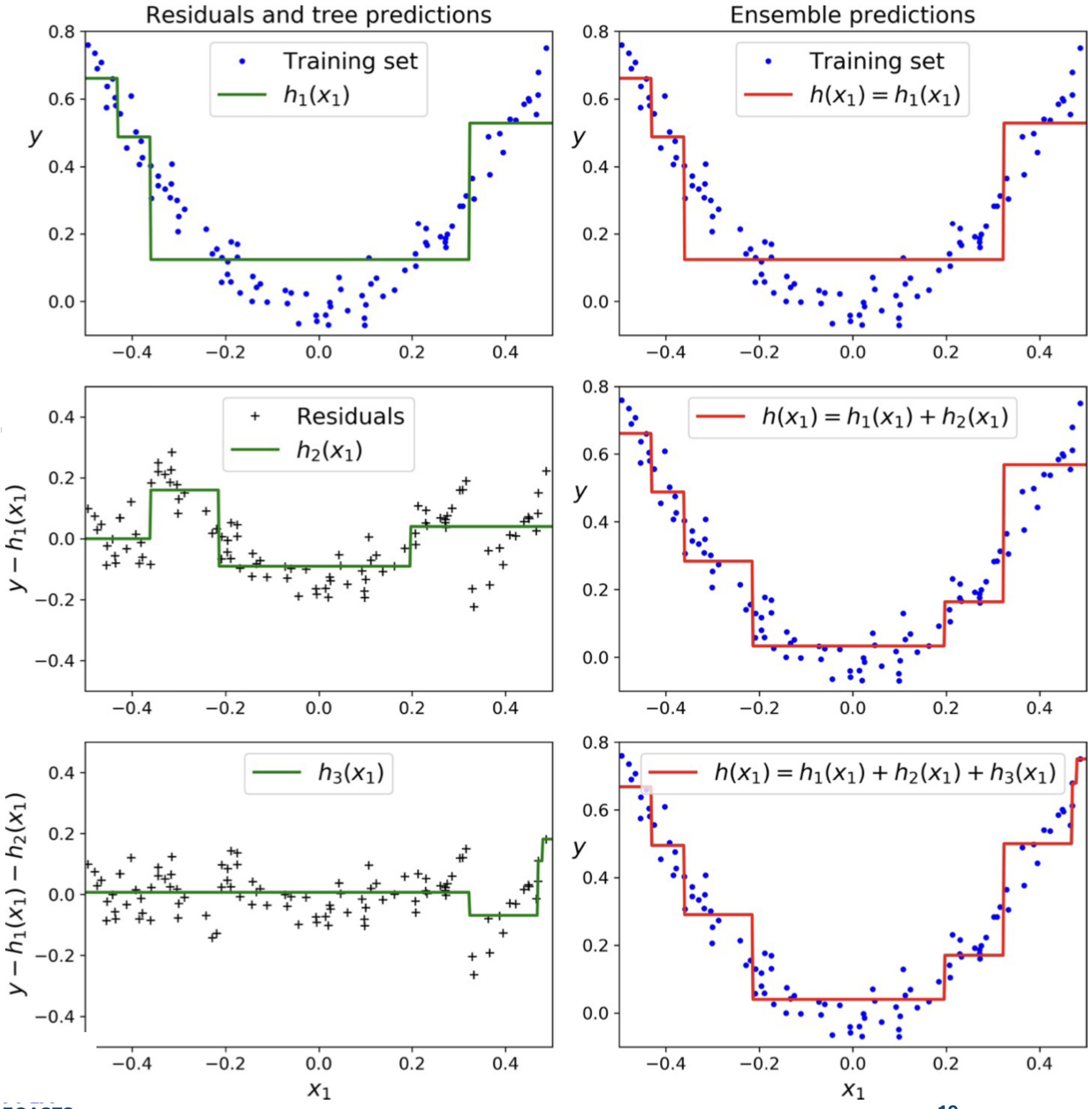
Each decision tree is relatively simple – a so-called "weak learner"

Regularisation is implemented by (depending on specific algorithm):

- Model complexity term in the objective function
- Bootstrapping training data, subsetting features
- Hyperparameters: max depth, max leaves, etc

Important to have validation and test datasets.

- Stop training when validation scores no longer improve. Test set is final check of performance.



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

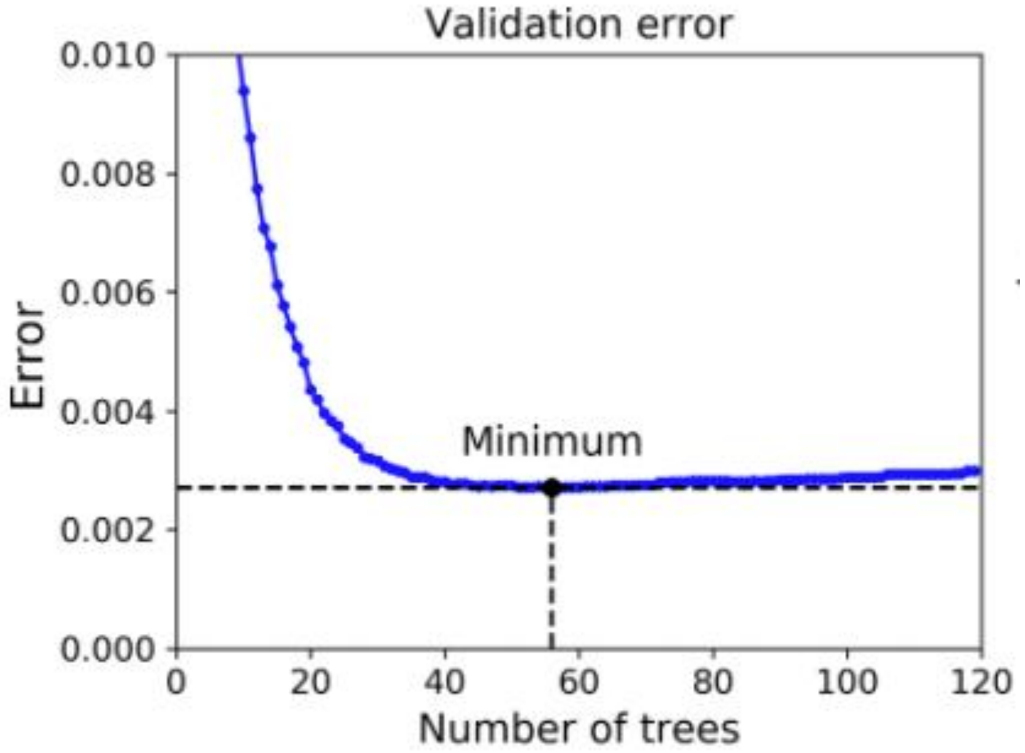
Gradient boosted trees

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

How to avoid overfitting:

- Early stopping (see figure)
- 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 training data when training each tree (leads to better generalisation)

GBTs can be optimized for GPUs = fast over big datasets.

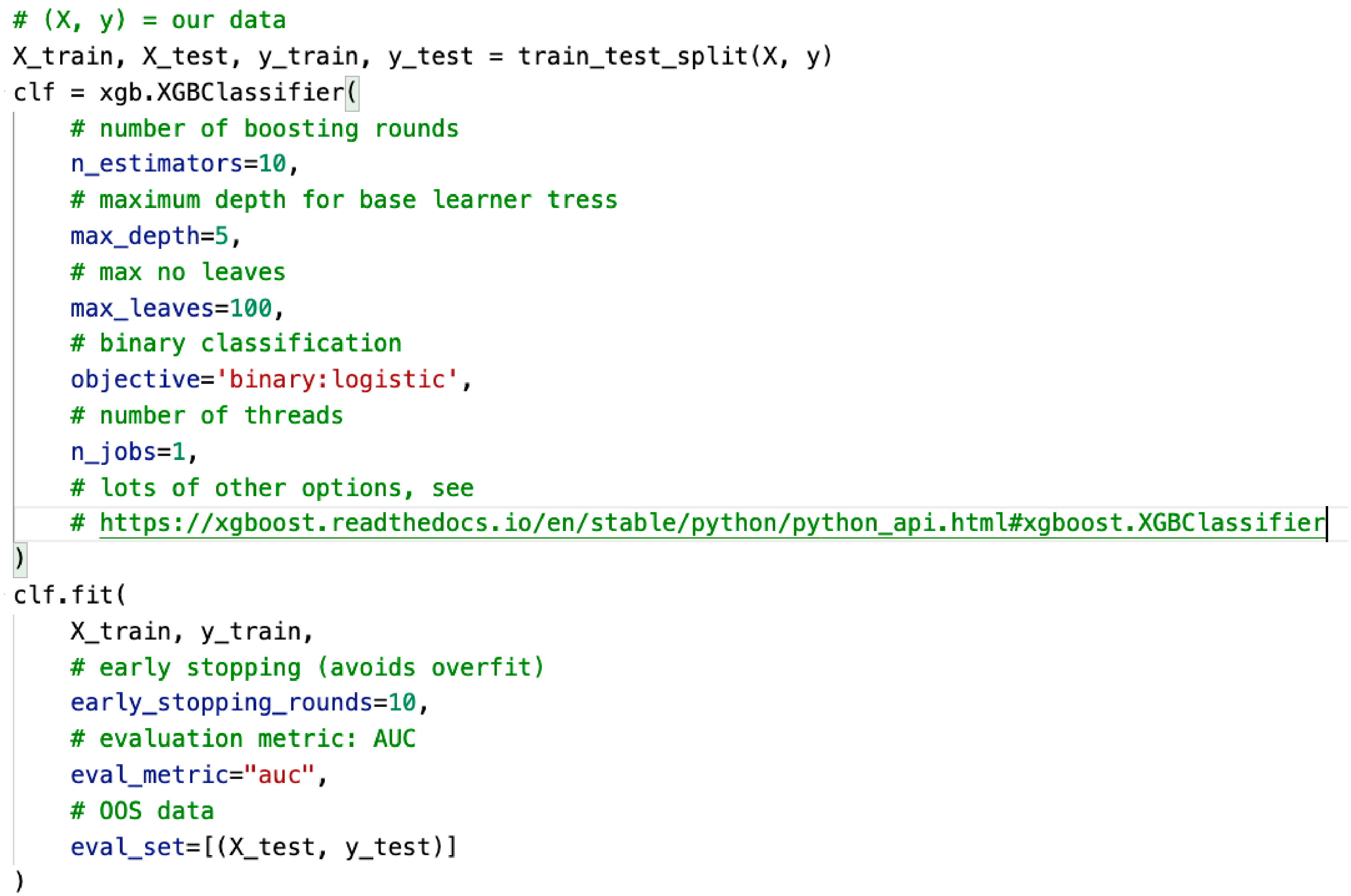


LightGBM dmlc XGBoost scikit learn CatBoost

ECMWF

Figure from chapter 7 of (Geron, 2019)

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

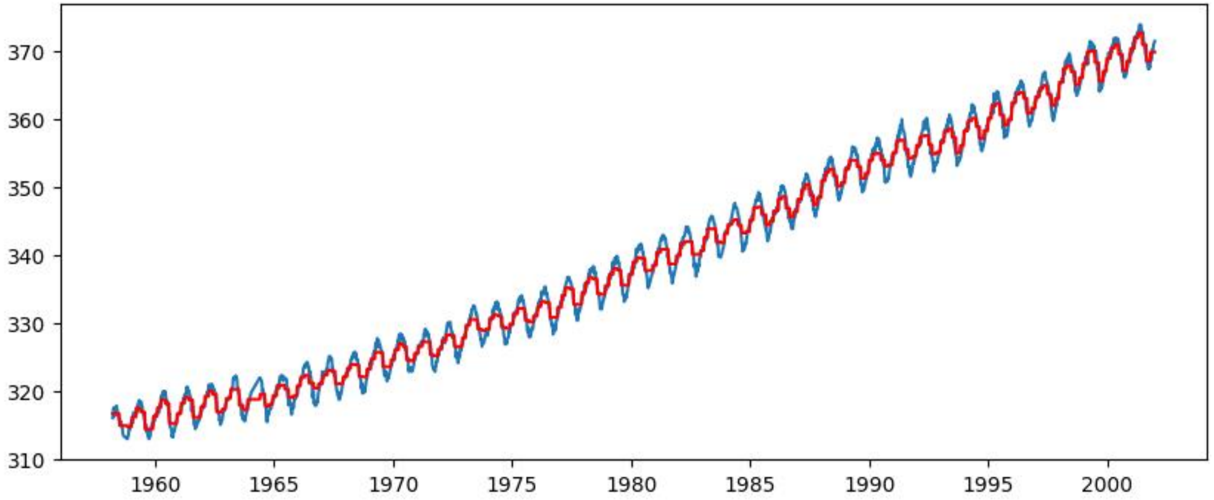
Fitted model

Very flexible – most basic example fitted here.

Feature importance is built in with plotting functions using `plot_importance` method.

- **gain:** Average improvement in model performance when a feature is used
- **weight:** number of times a feature is used to split
- **cover:** av. number of samples affected by splits on that feature

Even better – use SHAP values from the shap package (global measure, accounts for correlations, etc).



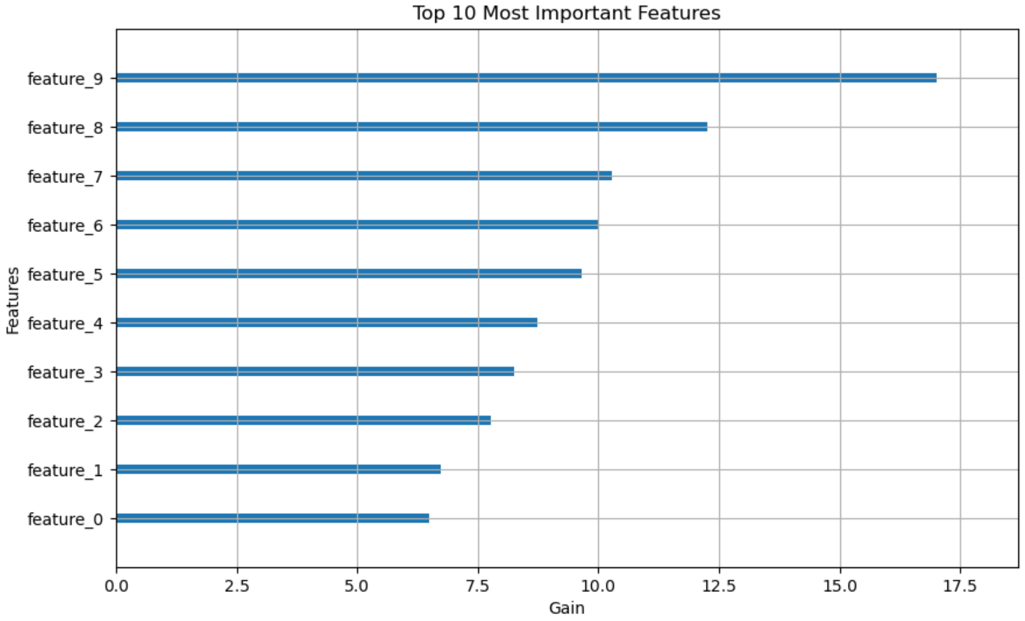


Figure from chapter 7 of (Geron, 2019)

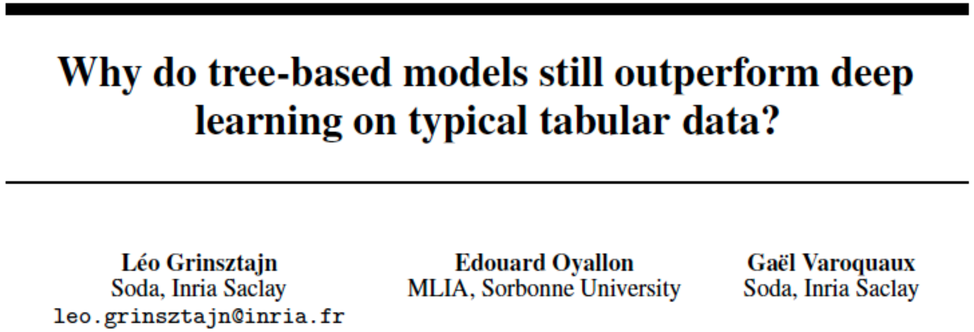
Comparison with NNs

Often, they are the 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



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

https://arxiv.org/abs/2207.08815

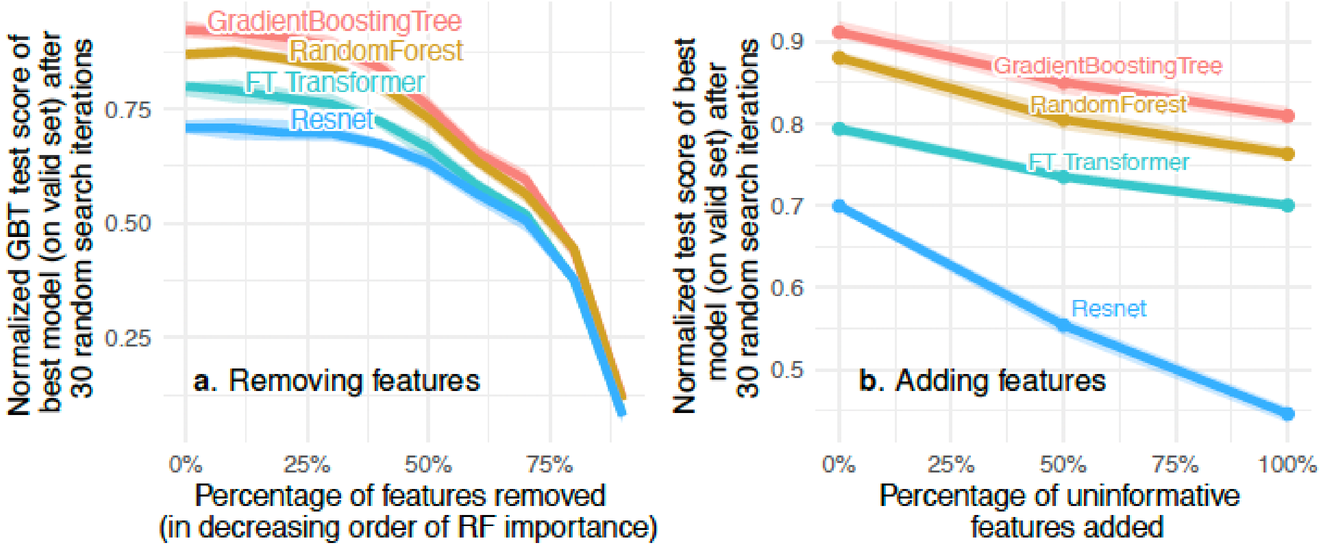


Figure 4: **Test accuracy changes when removing (a) or adding (b) uninformative features.** Features are removed in increasing order of feature importance (computed with a Random Forest). Added features are sampled from standard Gaussians uncorrelated with the target and with other features. Scores are 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).

When to use what?

Tabular data

Pure decision trees not usually useful, but important to understand as a building block of RFs and GBTs

Random forests (ensembles of decision trees):

- Work well out of the box, not especially sensitive to hyperparameters
- Easily parallelised
- Natural uncertainty estimation
- Good with noisy data

Gradient boosted trees (sequential correction of residuals):

- Can squeeze a bit more performance than a RF, but
- More sensitive to tuning parameters
- Can be more sensitive to noise

Neural networks can also work well! But, usually take more time to train, less explainable, etc.

Non-tabular data

Quite often, neural networks will be a good choice, if sufficient training data available:

- Can handle very complex relationships
- Scalable to very large datasets
- Can take very high dimensional inputs through e.g. convolutions, etc.

See next lectures!

Summary

Tree algorithms don’t (usually) make the headlines (any more). But on tabular data they should always be tested, most often beat neural networks. Not generally suitable for data where space/time order is important.

Good things

✓ 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.

✓ May be useful to combine with PCA for correlated features

Be careful with

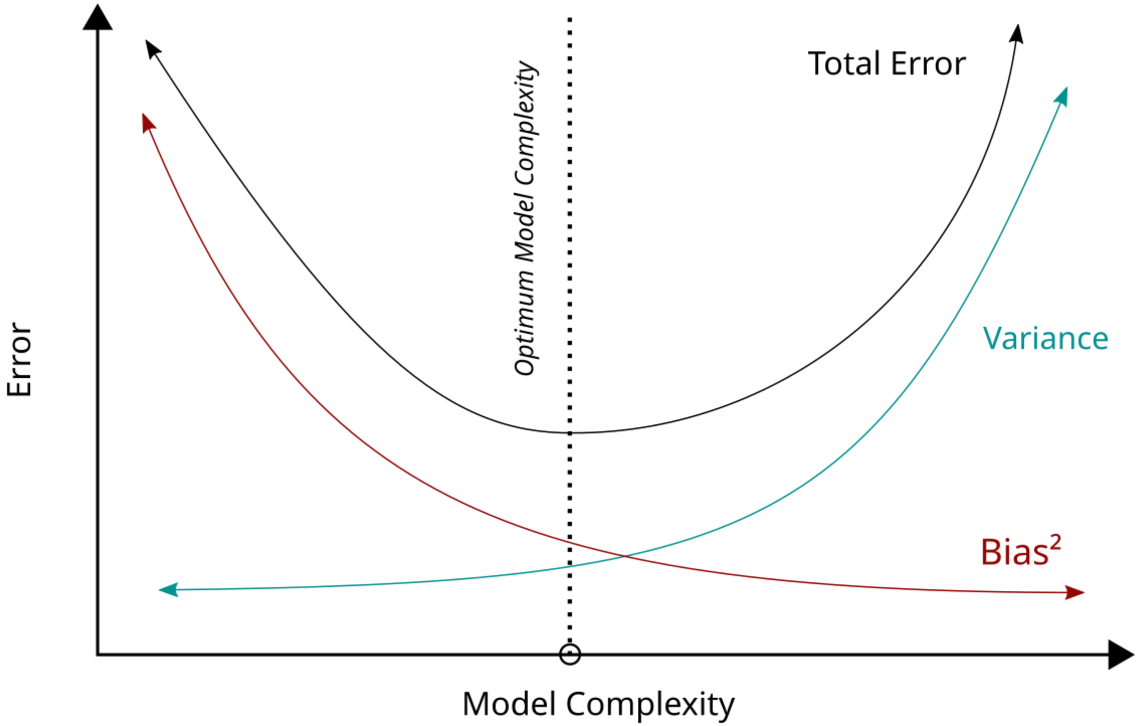
○ Decision trees/random forest/gradient-boosted trees are all very capable of overfitting.

- Various tools/parameters for regularisation exist
- Vital to have good data hygiene.
- Truly independent training/validation/test sets.

○ Usually need to deal with missing values prior to using tree-based methods, though some libraries have tools built in.

ECMWF | EUROPEAN CENTRE FOR MEDIUM-RANGE WEATHER FORECASTS

Extra: bias-variance



Bias is the deviation of the model from the true underlying process.

Model variance represents sensitivity to the training data (overfitting).

There is usually some irreducible error, due to noisy measurements and/or unaccounted features.

$$y = f(x) + \epsilon$$

$$\mathbb{E}[(y - \hat{f}(x))^2] = \underbrace{(\mathbb{E}[\hat{f}(x)] - f(x))^2}_{\text{Bias}^2} + \underbrace{\mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2]}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Irreducible error}}$$