

Introduction to Machine Learning for Environmental Science

Stefan Siegert

This 2-day workshop introduces core concepts of machine learning including common algorithms, standard workflows and key software libraries. The focus will be on supervised machine learning for regression and classification using a range of methods such as random forests and deep convolutional neural networks. The material will be taught through a mix of theory and practical examples across different environmental sciences such as meteorology and ecology.

Introduction

- Supervised ML
- Model training
- Examples, model evaluation

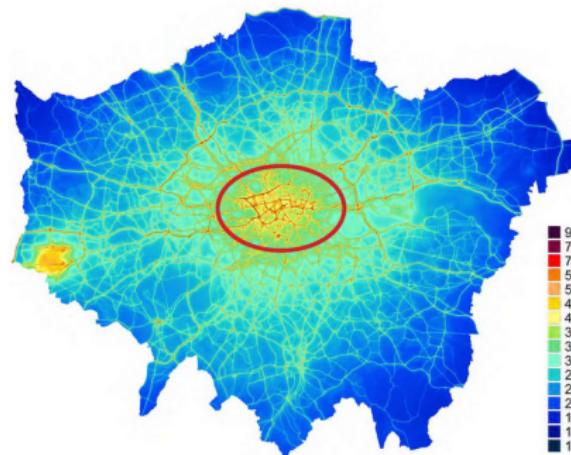
Tree-based methods

- Decision trees
- Random Forest, XGBoost

Neural Networks

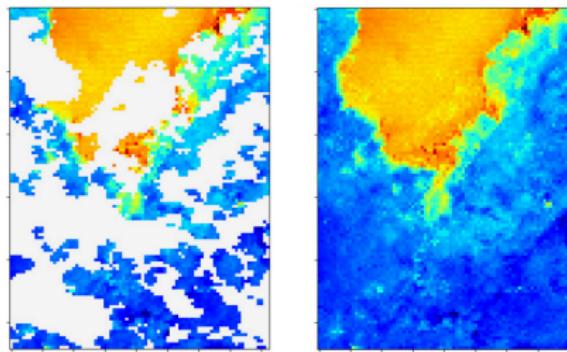
Some examples

Air quality prediction: Predict high-resolution spatial pattern of next day PM10 concentration based on learned relationships between pollution, weather and traffic.



Examples of ML applications

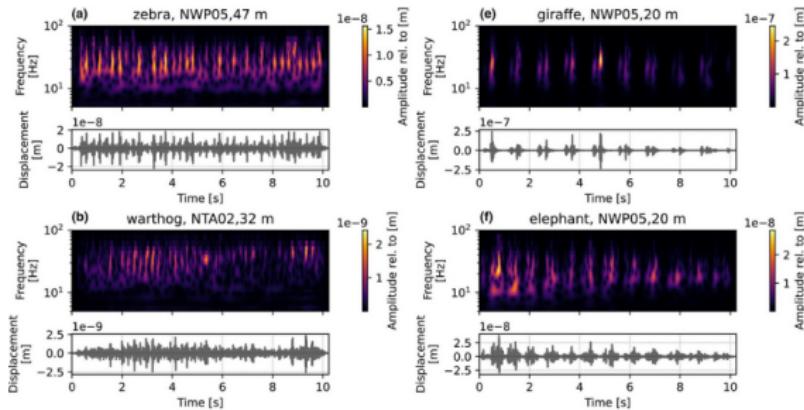
Gap filling: Fill in missing data due to clouds in satellite images based on learned spatial patterns.



<https://doi.org/10.3390/rs12233865>

Examples of ML applications

Monitoring wildlife: Learning patterns in seismic signals to detect and monitor animal species.

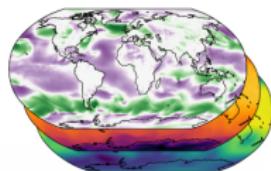


<https://doi.org/10.1111/2041-210X.70021>

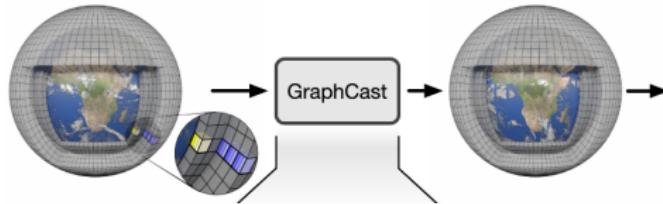
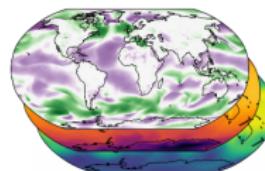
Examples of ML applications

Weather prediction: Learning spatial, temporal and intervariable relationships between weather quantities to make skilful global weather predictions.

a) Input weather state



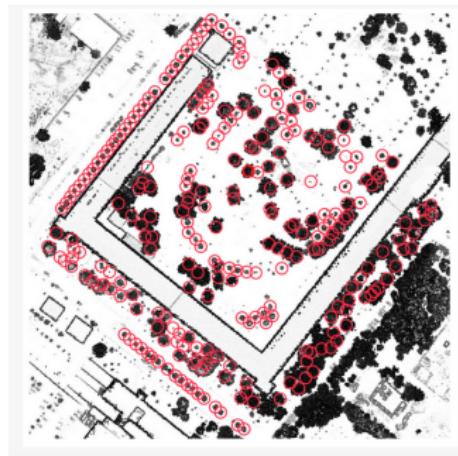
b) Predict the next state



<https://doi.org/10.48550/arXiv.2212.12794>

Examples of ML applications

Image segmentation: Detecting and annotating trees on urban maps based on learned image patterns.



<https://doi.org/10.3390/ijgi11040226>

Examples of ML applications

What do these examples have in common?

- High-dimensional data (space, time, multivariable)
- Need to learn complex interactions between variables
- Data-rich but often limited mechanistic understanding
- Clear framing as an input-output relationship
- Focus on skilful prediction of observables

Supervised machine learning: General setup

- We have an observable **target outcome** we wish to predict.
- The prediction can be based on a set of **input features**.
- For example
 - predict future temperature based on current temperature, pressure, wind
 - predict occurrence of wildfire based on measures of heat, humidity, wind
 - predict crop yield based on agricultural practice, soil nutrients, pests
- The goal of supervised ML is to **build a model** that predicts outcomes based on available features.
- A "good" ML model is one that produces accurate predictions.

Regression and classification

- The type of target variable determines the type of ML problem.
- **Regression:** Predicting quantitative outcomes.
 - For example: temperature in degrees, crop yield in tonnes, number of sharks, rain amount in mm
- **Classification:** Predicting categorical outcomes.
 - For example: wildfire yes/no, temperature freezing/nonfreezing, precipitation type rain/snow/hail, air quality low/medium/high
- Regression vs classification determines various details of the ML model and analysis.

WIP: Regression and classification

TODO: example outputs from a regression model and a classification model used later in the course to illustrate the distinction

Function approximation theory

- A ML model is a **prediction rule** that translates one or more features x_1, x_2, \dots into an estimate \hat{y} of the target y .
- The prediction rule is calculated by a **mathematical function**

$$\hat{y} = f(x_1, x_2, \dots)$$

- The function $f(\dots)$ usually includes **model parameters** $\theta_1, \theta_2, \dots$ that control how inputs are translated to outputs.
- Defining the feature vector $\mathbf{x} = (x_1, x_2, \dots)$ and parameter vector $\boldsymbol{\theta} = (\theta_1, \theta_2, \dots)$ a trainable ML model is often compactly written as

$$\hat{y} = f(\mathbf{x}; \boldsymbol{\theta})$$

Training data

- Supervised ML works by "learning from examples".
- **Training data S** is a set of input-output examples

$$S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$$

where

- $\mathbf{x}_i = (x_{i,1}, x_{i,2}, \dots, x_{i,k})$ is the **feature vector** (containing k features) in the i -th example
- y_i is the **target outcome** observed in the i -th example
- n is the **sample size**
- The data set S is used to train our ML model $f(\mathbf{x}; \theta)$ that can predict the outcome y^* for arbitrary feature vector \mathbf{x}^* .

Loss function

- A loss function $L(\hat{y}, y)$ is a function that quantifies **how different** the prediction \hat{y} is to the target outcome y .
- Loss functions are typically **negatively oriented**, where lower values indicate better predictions.
- The loss function must be appropriate for the ML task at hand (regression vs classification) and can usually be set when initialising the ML model.
- Otherwise the choice of loss function is **often ambiguous** and difficult to fully justify a particular choice.

Loss function

- **Regression** (where \hat{y} and y are real numbers):
 - Squared-error loss $L(\hat{y}, y) = (\hat{y} - y)^2$
 - Absolute-error loss $L(\hat{y}, y) = |\hat{y} - y|$
- **Binary classification** (where y is 0 or 1 and \hat{y} is the probability for outcome "1"):
 - Cross-entropy loss $L(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$
 - Quadratic loss $L(\hat{y}, y) = (\hat{y} - y)^2$
- **Multivariate classification** (where $y = (0, \dots, 1, \dots, 0)$ is a binary vector indicating which of k classes occurs and \hat{y} is a vector of k class probabilities)
 - Cross-entropy loss $L(\hat{y}, y) = -\sum_{i=1}^k y_k \log \hat{y}_k$
 - Focal loss $L(\hat{y}, y) = -\sum_{i=1}^k y_k (1 - \hat{y})^\gamma \log \hat{y}_k$

Empirical loss minimisation

- We can now **formalise** training of ML model $f(\mathbf{x}; \boldsymbol{\theta})$ as
 - trying to find model parameters $\boldsymbol{\theta}$ such that
 - the model outputs $\hat{y}_1, \dots, \hat{y}_n$ calculated from features $\mathbf{x}_1, \dots, \mathbf{x}_n$
 - achieve minimal average loss when compared to the targets y_1, \dots, y_n in the training data set
 - using an appropriate loss function $L(\hat{y}, y)$.
- **The Empirical Loss Minimisation paradigm:**

$$\hat{\boldsymbol{\theta}} = \operatorname{argmin}_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^n L[f(\mathbf{x}_i; \boldsymbol{\theta}), y_i]$$

- The trained model with parameters $\hat{\boldsymbol{\theta}}$ can then be used to make predictions \hat{y}^* based on new feature vectors \mathbf{x}^*

$$\hat{y}^* = f(\mathbf{x}^*; \hat{\boldsymbol{\theta}})$$

In summary ...

To train a ML model by empirical loss minimisation we need:

- a training data set S containing n examples of inputs and corresponding outcomes
- a model function $f(\mathbf{x}; \boldsymbol{\theta})$ with trainable parameters $\boldsymbol{\theta}$ that translates inputs \mathbf{x} into an output \hat{y}
- a loss function $L(\hat{y}, y)$ that quantifies how close model output \hat{y} is to the target outcome y
- a search algorithm to find parameters that minimise the loss

The combination of training data set, search algorithm, model function $f(\mathbf{x}; \boldsymbol{\theta})$ and loss function $L(\hat{y}, y)$ determine the ML approach and **these details should always be reported when documenting ML results.**

Code example: Linear regression

- **Features** x_1, \dots, x_n : temperatures at different times t_1, \dots, t_n
- **Targets** y_1, \dots, y_n : temperatures a short time Δ later $t_1 + \Delta, \dots, t_n + \Delta$
- **Model**: Linear function with intercept θ_1 and slope θ_2 , such that

$$\hat{y} = f(x; \theta) = \theta_1 + \theta_2 \cdot x$$

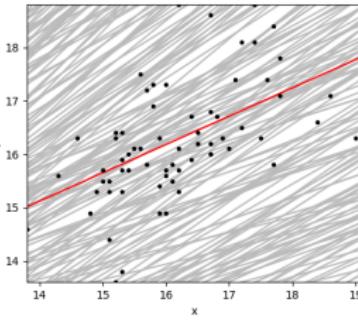
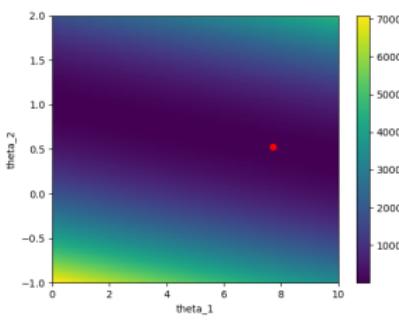
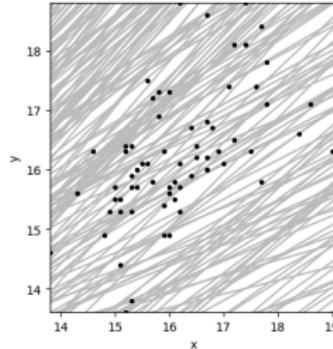
- **Loss**: Squared-error loss

$$L(\hat{y}, y) = (\hat{y} - y)^2$$

- **Search**: Exaustive search over a grid of parameter values
(don't do this in practice!)

Code example: Linear regression

```
In [129]: np.stack([x,y]).T
Out[129]:
array([[16. , 15.6],
       [15.9, 14.9],
       [15.9, 16.3],
       [15.2, 16.4],
       [13.8, 14.6],
       [16.7, 18.6],
       [15.3, 13.8],
       [16.1, 15.8],
       [15.1, 15.5],
       [16.7, 16.8],
       [14.9, 15.3],
       [15.3, 15.9],
       [15.1, 15.3],
```



```
In [164]: np.stack([x,y,y_hat]).T
Out[164]:
array([[16. , 15.6 , 16.18 ],
       [15.9 , 14.9 , 16.127],
       [15.9 , 16.3 , 16.127],
       [15.2 , 16.4 , 15.756],
       [13.8 , 14.6 , 15.014],
       [16.7 , 18.6 , 16.551],
       [15.3 , 13.8 , 15.809],
       [16.1 , 15.8 , 16.233],
       [15.1 , 15.5 , 15.703],
       [16.7 , 16.8 , 16.551],
       [14.9 , 15.3 , 15.597],
       [15.3 , 15.9 , 15.809],
       [15.1 , 15.3 , 15.703],
       [15.1 , 14.4 , 15.703],
       [15.9 , 15.4 , 16.127],
       [16. , 15.7 , 16.18 ],
       [15.4 , 16. , 15.862],
```

Machine Learning in python with scikit-learn

<https://scikit-learn.org>

The screenshot shows the official scikit-learn website at https://scikit-learn.org. The top navigation bar includes links for 'Install', 'User Guide', 'API', 'Examples', 'Community', and 'More'. A search bar and a version dropdown ('1.7.2 (stable)') are also present. The main content area features three large sections: 'Classification' (with a grid of 2x5 scatter plots), 'Regression' (with a line plot titled 'Predicted average energy transfer during the week'), and 'Clustering' (with a scatter plot titled 'K-means clustering on the digits dataset (PCA-reduced data)'). Each section has a 'Examples' button at the bottom.

Classification
Identifying which category an object belongs to.
Applications: Spam detection, image recognition.
Algorithms: Gradient boosting, nearest neighbors, random forest, logistic regression, and more...

Regression
Predicting a continuous-valued attribute associated with an object.
Applications: Drug response, stock prices.
Algorithms: Gradient boosting, nearest neighbors, random forest, ridge, and more...

Clustering
Automatic grouping of similar objects into sets.
Applications: Customer segmentation, grouping experiment outcomes.
Algorithms: k-Means, HDBSCAN, hierarchical clustering, and more...

Linear Regression in Scikit-Learn

```
import numpy as np
from sklearn.linear_model import LinearRegression

# load file
t7110 = np.loadtxt("t7110.dat", comments="#")

# extract features (july temperatures) and targets (august temperatures)
x = t7110[:, 7]
y = t7110[:, 8]

# sklearn expects k features to be stored in a 2d array with k columns
x = x.reshape(-1, 1)

# initialise a linear regression model
model = LinearRegression()

# fit the model to data
model.fit(x, y)

# extract parameter estimates from fitted model
theta_hat = np.array([model.intercept_, model.coef_[0] ])

# calculate predictions in training data
y_hat = model.predict(x)

# calculate predictions on new data
x_new = np.linspace(15, 20, 100).reshape(-1,1)
y_new = model.predict(x_new)
```

How good is my model?

- The model outputs $\hat{y}_1, \dots, \hat{y}_n$ be compared to the targets y_1, \dots, y_n using a suitable error metric.
- For example root mean squared error (RMSE)

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2}$$

- Our model has $RMSE \approx 0.9$ – Is that "good"?
- Getting $RMSE > 0$ means our model is not perfect, but how bad is it?
- To judge whether the model is useful we should compare it to a suitable benchmark.

Benchmarking

- A benchmark model (or reference model) for performance evaluation is usually one or several of
 - An alternative (competing) model for the same target.
 - Climatology (constant mean):

$$\hat{y}_i^{(clim)} = \frac{1}{n} \sum_{i=1}^n y_i$$

- Persistence (last available observation of the target); here:

$$\hat{y}_i^{(pers)} = x_i$$

- Any other simple estimate of the target that could be calculated with reasonable effort given the same inputs as our model.
- In fact, linear regression itself is often a good reference to benchmark more complicated ML model.

Skill scores

- Given a suitable error metric, we calculate
 - S : the error of our model (averaged over a test data set)
 - S_{ref} : the mean error of our chosen reference model
 - S_{perf} : the mean error of a hypothetical perfect model that outputs $\hat{y}_i = y_i$ each time; (usually $S_{perf} = 0$)
- The skill score of our model relative to the benchmark is then defined as

$$Skill = \frac{S_{ref} - S}{S_{ref} - S_{perf}}$$

- $Skill \leq 0$: Our model is not better than the benchmark.
- $Skill \in (0, 1)$: Our model improves over the benchmark.
- $Skill = 1$: Our model is perfect.

Skill scores

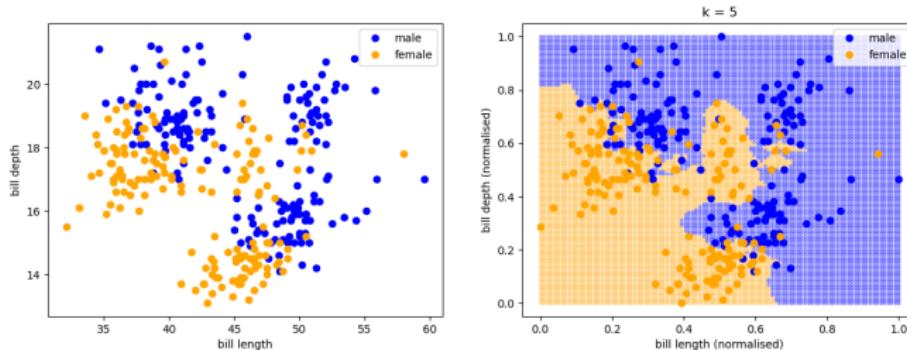
MORE HERE Skill scores of linear regression model

Example: k-nearest-neighbor classification

- given an input \mathbf{x}^* find the k closest features $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(k)}$ in the training data, and the corresponding outcomes $y^{(1)}, \dots, y^{(k)}$
- "closeness" between the feature vectors $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ is defined in terms of the Euclidean distance $\sqrt{\sum_m (x_m^{(i)} - x_m^{(j)})^2}$
- predict \hat{y}^* as the majority vote over $y^{(1)}, \dots, y^{(k)}$
- ties are resolved by picking the outcome class lowest value or alphabetical rank

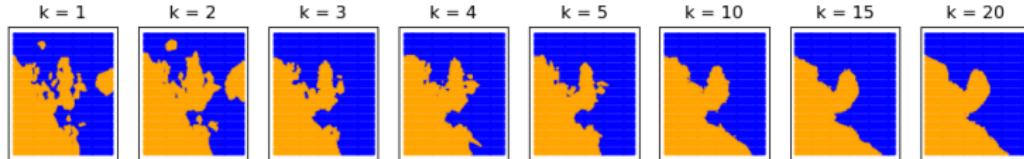
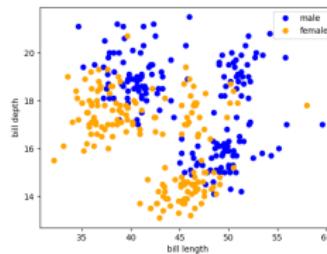
Example: k-nearest-neighbor classification

- For illustration we use data from the "Palmer Penguin" data.
- Features are penguins' bill length (x_1) and bill depth (x_2) and outcome (y) is the penguin's sex (male or female).
- The kNN classifier ($k = 5$) separates the x_1/x_2 plane into regions for "female" and "male", separated by a decision boundary.



Example: k-nearest-neighbor classification

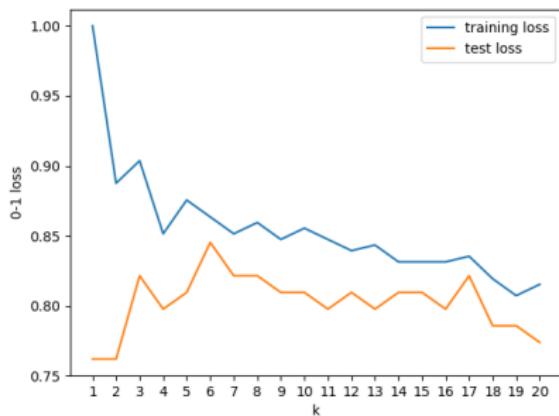
- The parameter k controls the "roughness" of the classifier.
- Larger values of k produce smoother decision boundaries.



- Is k a trainable parameter that can be selected by empirical loss minimisation?

Example: k-nearest-neighbor classification

- Split the data randomly into training and test data set:
 - Only the training data set is used by the kNN classifier.
 - Training and test data are used separately to calculate the 0-1 loss (aka accuracy, aka proportion correct) of the classifier.



- The training accuracy is perfect for $k = 1$ (Why?) and goes down as k increases.
- The test accuracy is best for $k \approx 6$.

- The neighborhood parameter k can be optimised by loss minimisation, but should use a different data set than the one used to define the kNN classifier.

In-sample vs Out-of-sample error

- Our goal is to train a model that generalises well to new data.
- Loss on training data (in-sample error) is too optimistic because the model has seen the data.
- Overfitting: A model achieves very low training loss by "regurgitating" the training data, including any noise and accidental patterns. Overfitted models generalise poorly to new data.
- Loss on previously unseen data (out-of sample error) is what we are really interested in.
- It is good practice to remove a fraction of data (e.g. 20%) during training and use only for model testing.

Further reading

- Linear algebra for least squares regression.
- Alternative loss functions
- Multiple linear regression for multiple inputs.
- Logistic regression for classification.
- Radius-neighborhood classifier and other neighborhood methods.
- Neighborhood methods for missing data imputation.
- Uncertainty estimation: Resampling methods, bootstrapping.

Introduction

- Supervised ML
- Model training
- Examples, model evaluation

Tree-based methods

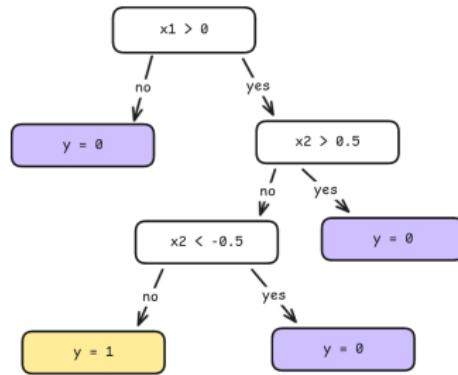
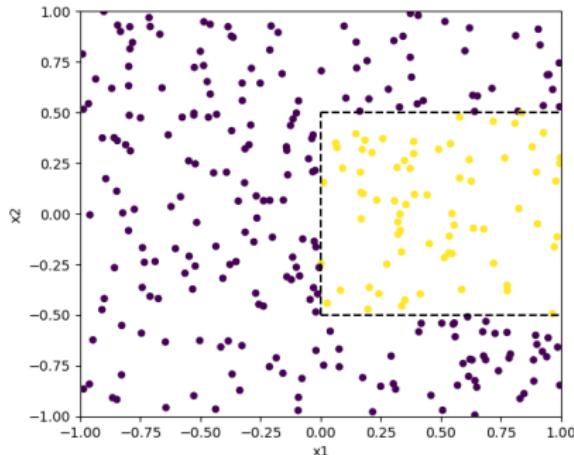
- Decision trees
- Random Forest, XGBoost

Neural Networks

Background

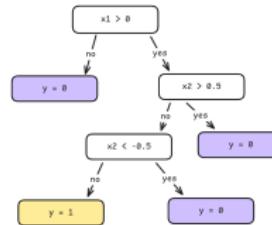
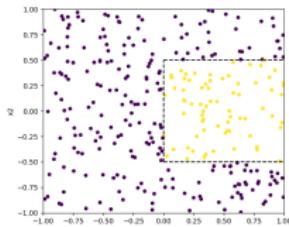
- Tree-based methods are simple and powerful function approximation methods.
- During training, the feature space is split up into rectangular regions.
- The model's prediction within each rectangular region is a constant.

Decision tree example



- Left: Suppose (normalised) precipitation x_1 and temperature x_2 determine survival/death ($y = 1$ or 0) of a plant species by a rectangular "habitable zone" in the $x_1 - x_2$ plane (left).
- Right: The survival/death classification can be represented by a decision tree (right).

Decision tree example



- Classification function for feature vector $\mathbf{x} = (x_1, x_2)$

$$f(\mathbf{x}; \theta) = \begin{cases} 0 & \text{if } x_1 < 0 \\ 0 & \text{if } x_1 > 0, x_2 > 0.5 \\ 0 & \text{if } x_1 > 0, x_2 < -0.5 \\ 1 & \text{otherwise} \end{cases}$$

- If decision thresholds are trainable parameters, a decision tree becomes an ML model.

Decision tree terminology

- **Split:** Partitioning data based on a feature and threshold
- **Node:** a point in the tree where a decision is made
- **Root node:** first split in the tree
- **Internal node:** A node that performs a split
- **Leaf node:** Final node that outputs a prediction
- **Depth:** Number of decision levels from Root node to leaves
- **Impurity:** Measure how mixed the targets are within a split
- **Information gain:** Reduction in impurity after a split

Training a decision tree for binary classification

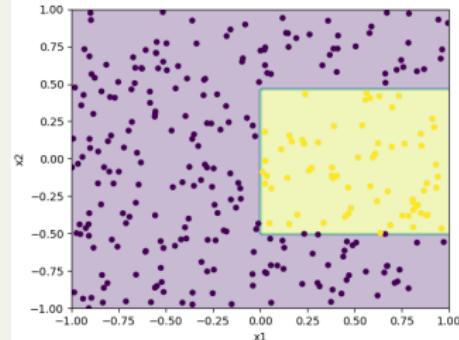
```
from sklearn.tree import DecisionTreeClassifier

# synthetic data
# X.shape = (300, 2) -- x1,x2 as columns
# y.shape = (300,) -- targets

# initialise decision tree with max. depth 3
clf = DecisionTreeClassifier(max_depth=3)

# train decision tree
clf.fit(X, y)

# predict
clf.predict(np.array([0.5,0]).reshape(-1,2))
# returns: array([1])
```



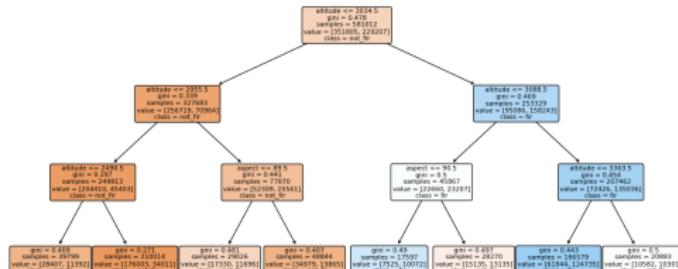
The trained decision tree recovers the correct classification boundary.

WIP: Code example: Tree cover classification

TODO: Data set (plot), no code required it's on previous slide,
decision surface

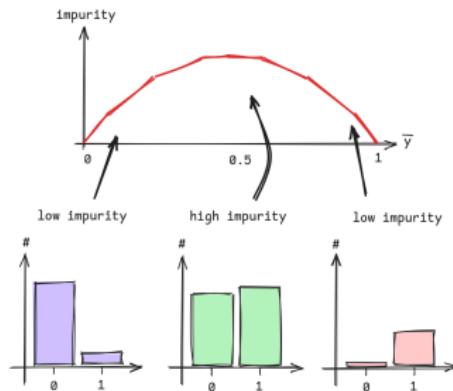
Visualising trained decision trees

```
from sklearn import tree
clf = DecisionTreeClassifier(
    max_depth=3)
clf.fit(X, y)
plt.figure(figsize=(10, 6))
tree.plot_tree(clf,
    feature_names=["altitude",
                   "aspect"],
    class_names=["not_fir",
                 "fir"],
    filled=True,
    rounded=True,
    fontsize=8)
plt.show()
```



Finding features and thresholds for splitting

- Gini impurity and entropy are measures of impurity, disorder, indeterminism.
- Binary data:
 $\mathbf{y} = (y_1, \dots, y_n) = (0, 0, 1, 0, 1, \dots)$.
- Summarise $\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$.



- Gini impurity: $G(\mathbf{y}) = 1 - \bar{y}^2 - (1 - \bar{y})^2$
- Entropy: $H(\mathbf{y}) = \bar{y} \log \bar{y} + (1 - \bar{y}) \log(1 - \bar{y})$

Maximising Information Gain

- We apply threshold-splitting of the data wrt one feature.
- This separates the data \mathbf{y} into \mathbf{y}_{left} and \mathbf{y}_{right} of sizes n_{left} and n_{right} .
- The impurity of the split data set is calculated by

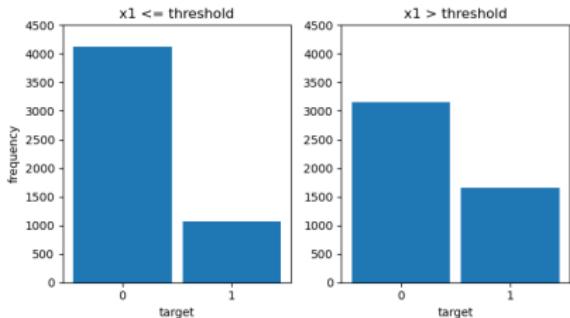
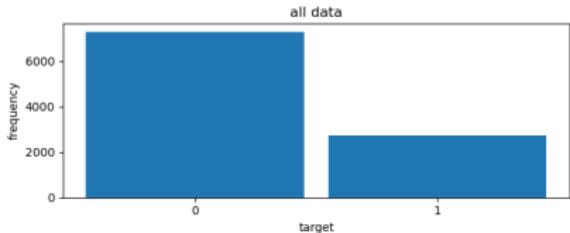
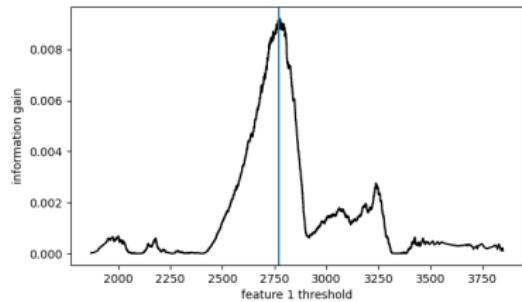
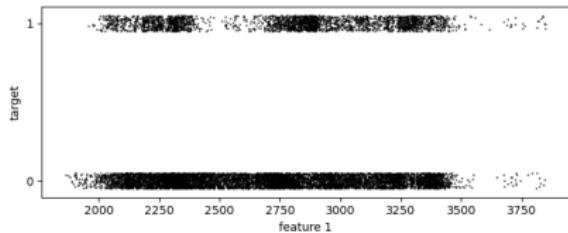
$$G(\mathbf{y}_{left}, \mathbf{y}_{right}) = \frac{n_{left}}{n} G(\mathbf{y}_{left}) + \frac{n_{right}}{n} G(\mathbf{y}_{right})$$

- The impurity reduction (or "information gain" IG) caused by the split is given by

$$IG = G(\mathbf{y}) - G(\mathbf{y}_{left}, \mathbf{y}_{right})$$

- The feature/threshold combination that yields the highest information gain is selected.

Maximising Information Gain



Random Forest

- A RF is an ensemble of decision trees.
- Each tree is trained on random subsets of training data and input features.
- Each tree makes a prediction and the forest aggregates them.
 - Classification: Majority vote
 - Regression: Average
- Improves robustness and avoids overfitting compared to single decision tree.

Example

```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(
    n_estimators=100, # number of trees
    max_depth=20,      # maximum tree depth
    random_state=42   # reproducibility
)
model.fit(X, y)
y_pred = model.predict(X)
```

MORE HERE: example data, single tree vs RF decision surfaces

Boosting

- In ensemble methods (such as RF), each classifier (tree) is independently trained and optimised for maximum accuracy.
- A boosting algorithm trains a chain of weak classifiers sequentially.
- Core algorithm:
 - train a simple model
 - calculate its errors over the training data
 - train the next model to take the same inputs but predict the previous model's errors
 - repeat, then aggregate all models (often with weights)
- Throughout the sequence the model is improved by gradually refining the prediction.

WIP: Gradient Boosting, XGBoost

- Gradients of the loss function $g_i = \frac{\partial L(\hat{y}_i, y_i)}{\partial \hat{y}_i}$ approximate errors.
- Changing \hat{y}_i proportional to $-g_i$ will decrease the loss.
- In gradient boosting, each tree predicts the negative gradient $\frac{\partial L(\hat{y}, y)}{\partial \hat{y}}$ of the previous tree
- XGBoost is a popular library for Gradient Boosting
- MORE HERE

WIP: Forest fire prediction

WIP: Evaluation metrics in scikit-learn

- accuracy
- confusion matrix
- ROC/AUC analysis
- classification report
- many more for further reading

WIP: Variable importance

- quick and dirty: `clf.feature_importances_` (mean impurity decrease)
- more informative: `sklearn.inspection.permutation_importance` (performance drop when a feature is shuffled)
- "explainability" /reporting: SHAP values (global importance measure, standard)
- many more for further reading

Introduction

- Supervised ML
- Model training
- Examples, model evaluation

Tree-based methods

- Decision trees
- Random Forest, XGBoost

Neural Networks

Artificial neuron (Perceptron)

- An artificial neuron (perceptron) is a function

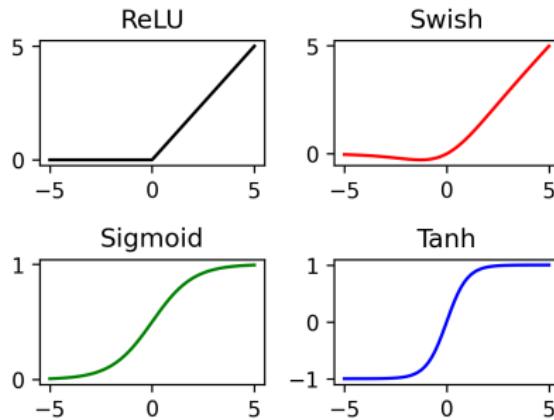
$$y = f(\mathbf{x}; \mathbf{w}, b) = \varphi(w_1 \cdot x_1 + \cdots + w_k \cdot x_k + b)$$

- where
 - $\mathbf{x} = (x_1, \dots, x_k)$ is the k -dimensional input vector
 - y is the 1-dimensional output of the neuron
 - $\mathbf{w} = (w_1, \dots, w_k)$ is the k -dimensional vector of weights
 - b is the bias
 - $\varphi(\cdot)$ is the activation function

Activation functions

- Typical activation functions:

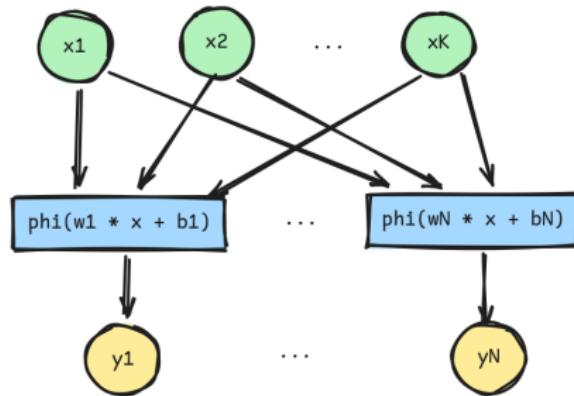
- ReLU (Rectifier): $\varphi(x) = \max(0, x)$
- Sigmoid: $\varphi(x) = 1/(1 + e^{-x})$
- Swish: $\varphi(x) = x/(1 + e^{-x})$
- Tanh: $\varphi(x) = (e^x - e^{-x})/(e^x + e^{-x})$



Single-layer perceptron

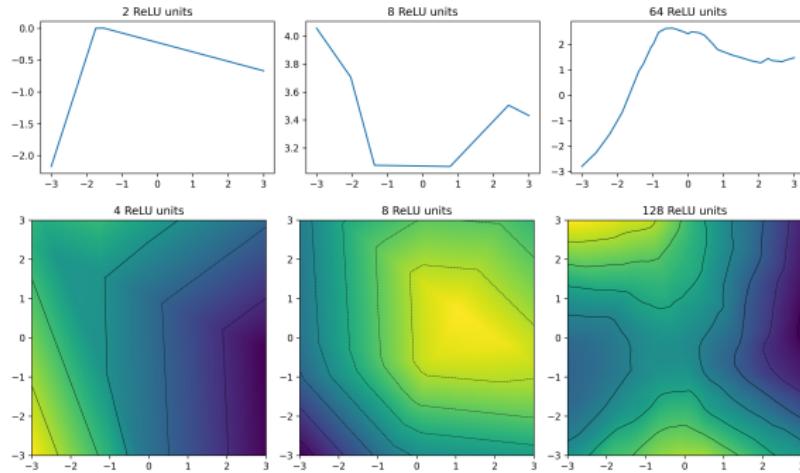
- multiple perceptrons (each with different weights and biases) calculating their outputs in parallel

$$y_i = \varphi(\mathbf{w}_i^T \mathbf{x} + b_i)$$



Examples

Examples of randomly initialised 1d and 2d single layer perceptrons with increasing numbers of neurons, illustrating increasing flexibility and "expressive power".



WIP - Terminology

- Neuron, node
- Weight
- Bias
- Hidden layer
- Output layer

Multilayer perceptron

The Keras API

Simple data example

Model summary

Loss functions

crossentropy-loss for classification, squared-error loss for regression

Gradient descent

Backpropagation

Stochastic gradient descent

Optimiser hyperparameters

Monitoring performance

learning curves, restarting training runs

Spatial data: 2d convolution

The channel dimension

Convolutional neural network

Encoder-Decoder architecture

Example: Weather image classification

Cross-validation, early stopping

Dropout

Further Reading

- Sensitivity analysis, interpretability
- Fine-tuning for new outputs/objectives
- Speeding up NNs: Quantisation, layer-wise optimisation
- MORE HERE