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

Supervised machine learning

- We usually have an target (outcome) measurement that we wish to predict.
- The prediction should be based on a set of features (inputs).
- For example
  - predict future temperature based on current temperature, pressure and wind speed
  - predict occurrence of wildfire based on measures of heat, drought and wind
  - predict crop yield based on precipitation, temperature, soil nutrients, pests.
- The goal of supervised ML is to build a model that predicts outcomes based on available features in new, previously unseen situations.
- A "good" model is one that produces accurate predictions.

Regression and classification

- The nature of the target variable determines the type of ML problem.
- Regression: Predicting quantitative outcomes.
  - For example: temperature in degrees, crop yield in tonnes, number of sharks, precipitation amount in mm
- Classification: Predicting categorical outcomes.
  - For example: wildfire occurrence, temperature below zero, precipitation type (rain/snow/hail), precipitation amount none/low/medium/high

Function approximation

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

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

- The function $f(\ldots)$ usually includes one or many parameters $\theta_1, \theta_2, \ldots$ that control how the function translates inputs to outputs.
- Defining the feature vector $\boldsymbol{x} = (x_1, x_2, \ldots)$ and parameter vector $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots)$ a ML model is often compactly written as

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

Training data

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

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

where

  - $\boldsymbol{x}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,k})$ is the feature vector (containing k features) in the i-th example
  - y_i is the outcome in the i-th example
  - n is the sample size
- The training data set S is used to build our prediction model to predict a new outcome y^* based on a feature vector $\boldsymbol{x}^*$.

Empirical loss minimisation

- Model training is achieved by
  - trying to find model parameters $\boldsymbol{\theta}$ such that
  - the model outputs $\hat{y}_1, ..., \hat{y}_n$ calculated from $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$
  - are as "close" as possible to the training targets $y_1, \ldots, y_n$
  - where closeness is measured by the loss function $L(\hat{y}, y)$

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

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

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

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 with trainable parameters $\boldsymbol{\theta}$ that translates inputs $\boldsymbol{x}$ into an output $\hat{y}$
- a loss function that quantifies how close model output $\hat{y}$ is to the target outcome y
- a mechanism to find parameters that minimise the loss

Example: Linear regression

- Training features $x_1, \ldots, x_n$ are temperatures at different times $t_1, \ldots, t_n$
- Training targets $y_1, \ldots, y_n$ are temperatures a short time Δ later $t_1 + \Delta, \ldots, t_n + \Delta$
- Model function is a linear function with intercept θ_1 and slope θ_2, such that

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

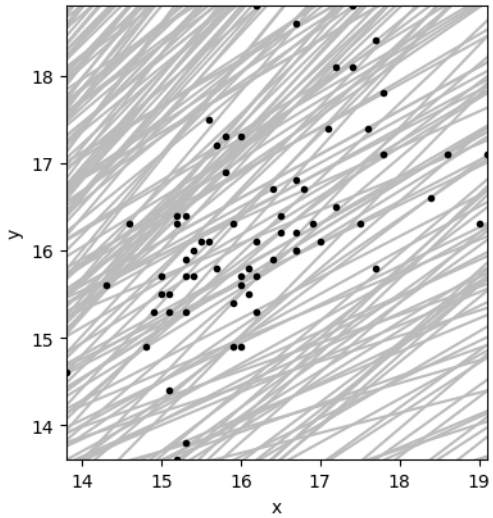
- Our loss function is the squared-error loss

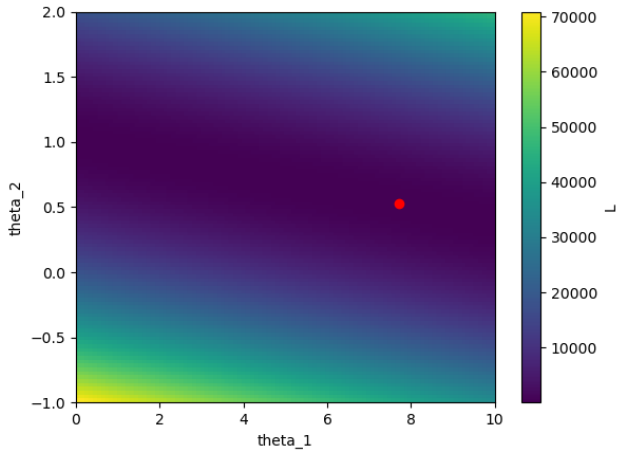
$$L(\hat{y}, y) = (\hat{y} - y)^2$$

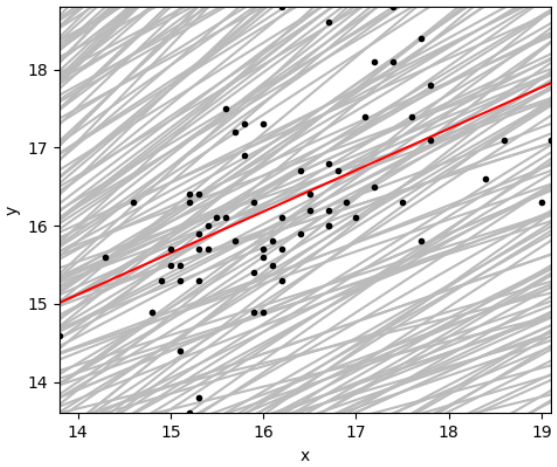
- Optimisation: exaustive search over a grid of parameter values (don't do this in practice!)

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],
```



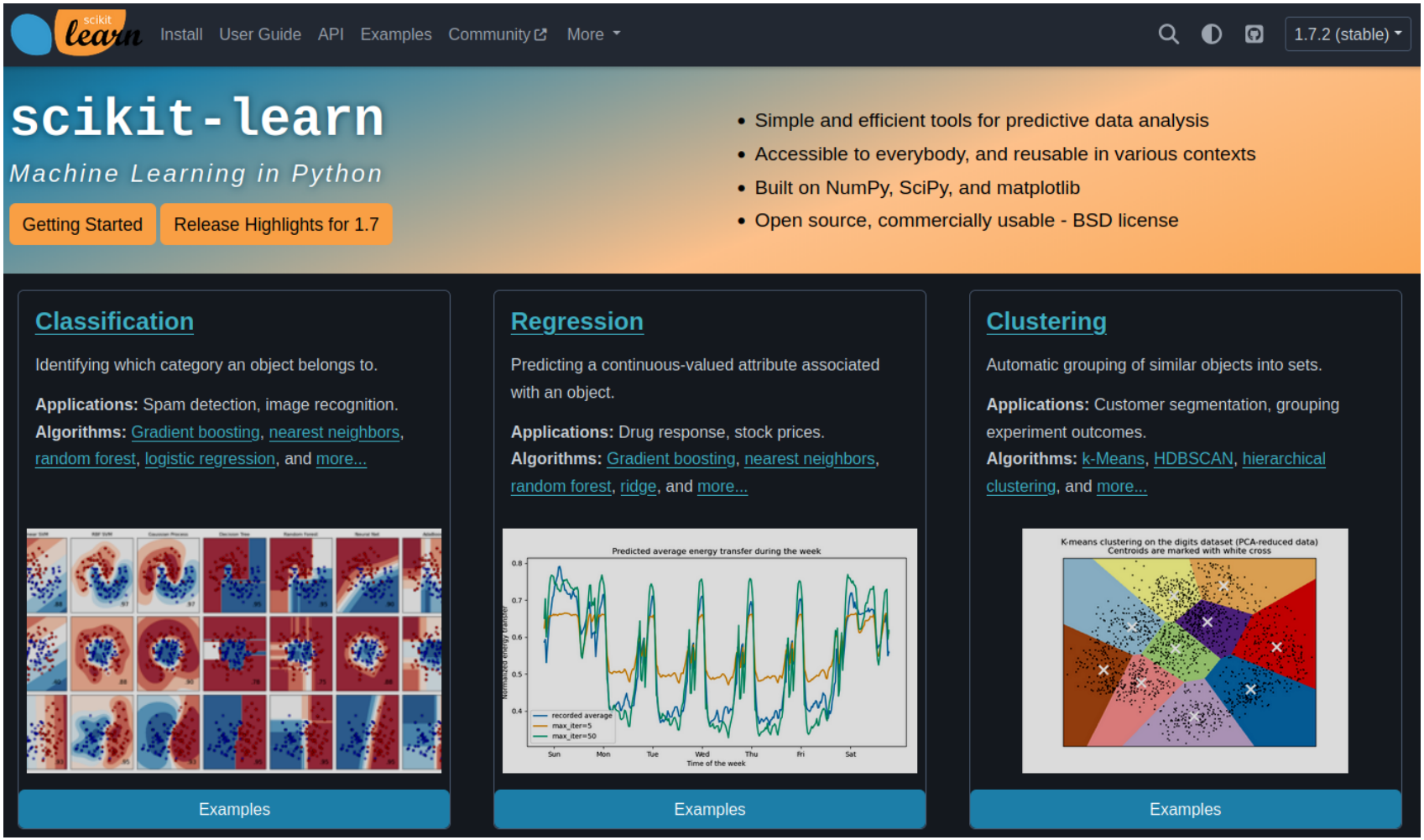





```
In [164]: np.stack([x,y,y_hat]).T
Out[164]:
array([[16.   , 15.6  , 16.18 ],
       [15.9  , 14.9  , 16.127],
       [15.9  , 16.3  , 16.127],
       [15.2  , 16.4  , 15.756],
       [13.8  , 14.6  , 15.014],
       [16.7  , 18.6  , 16.551],
       [15.3  , 13.8  , 15.809],
       [16.1  , 15.8  , 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`



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 $\hat{y}_1, \ldots, \hat{y}_n$ be compared to the targets $y_1, \ldots, y_n$ 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 or 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

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

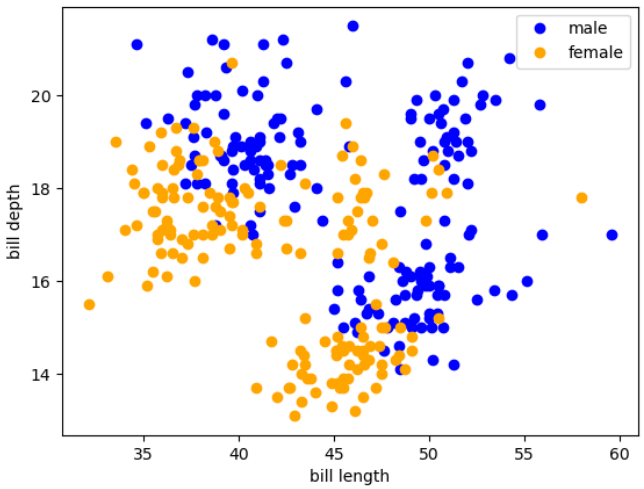
- $Skill \leq 0$: Our model is not better than the benchmark.
- $Skill \in (0, 1)$: Our model improves over the benchmark.
- $Skill = 1$: Our model is perfect.

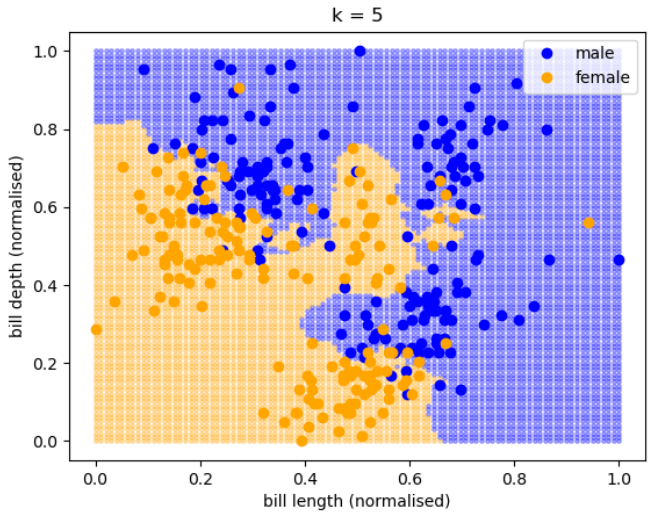
Example: k-nearest-neighbor classification

- given an input $\boldsymbol{x}^*$ find the k closest features $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(k)}$ in the training data, and the corresponding outcomes $y^{(1)}, \ldots, y^{(k)}$
- "closeness" between the feature vectors $\boldsymbol{x}^{(i)}$ and $\boldsymbol{x}^{(j)}$ 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 $y^{(1)}, \ldots, y^{(k)}$
- ties are resolved by picking the outcome class lowest value or alphabetical rank

Example: k-nearest-neighbor classification

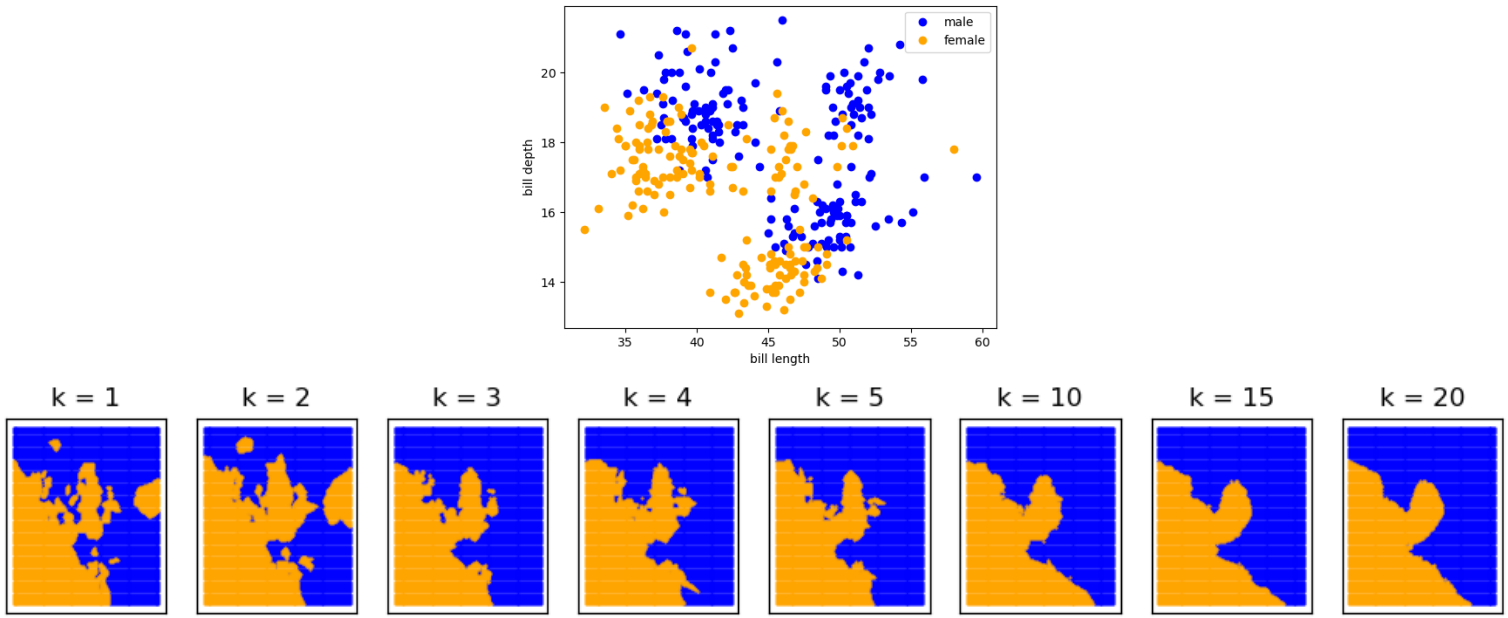
- For illustration we use data from the "Palmer Penguin" data.
- Features are penguins' bill length (x_1) and bill depth (x_2) and outcome (y) is the penguin's sex (male or female).
- The kNN classifier ($k = 5$) separates the x_1/x_2 plane into regions for "female" and "male", separated by a decision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





Example: k-nearest-neighbor classification

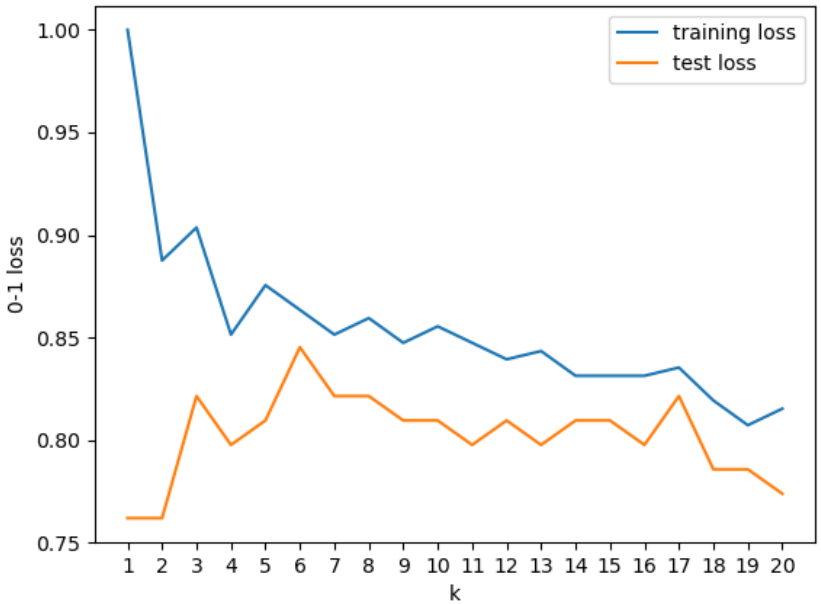
- The parameter k controls the "roughness" of the classifier.
- Larger values of k produce smoother decision boundaries.



- 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 "regurtitating" 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

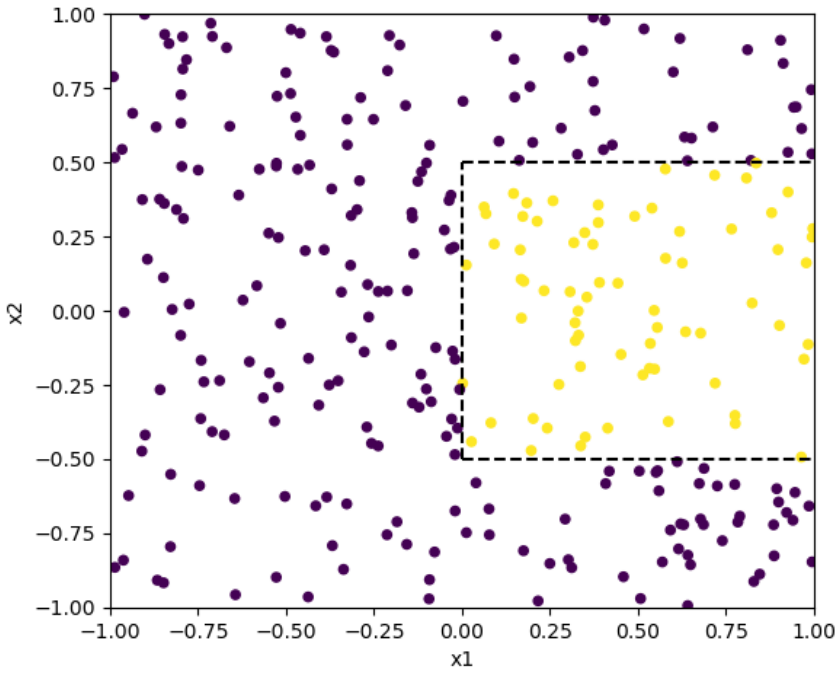
- Decision trees
- Random Forest, XGBoost

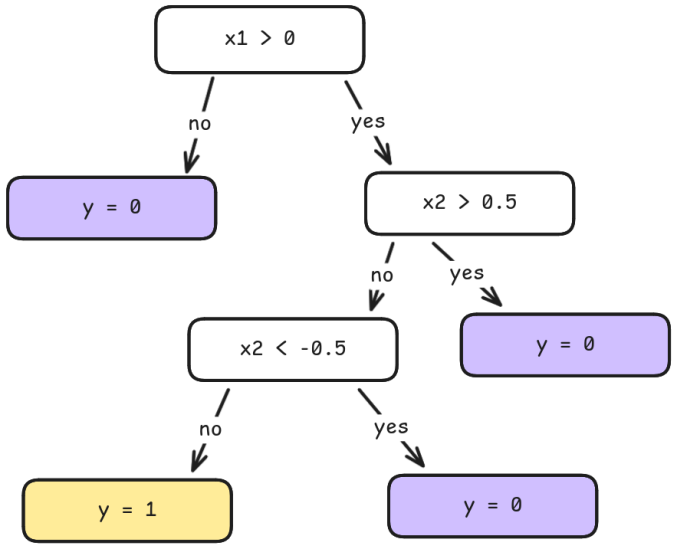
Neural Networks

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





Decision tree terminology

- Split: Partitioning data based on a feature and threshold value
- Node: a point in the tree where a decision is made
- Root node: first split in the tree
- Internal/decision node: Any 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 classes are
- Information gain: Reduction in impurity after a split

Training a decision tree

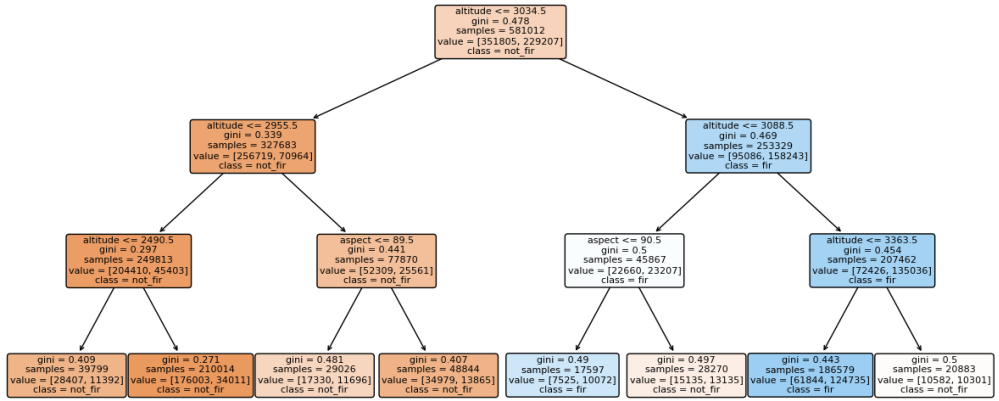
```
from sklearn.tree import DecisionTreeClassifier

# initialise decision tree with set maximum depth
clf = DecisionTreeClassifier(max_depth=10)

# X: shape (n_samples, n_features)
# y: shape (n_samples,) with 0/1 encoded class label

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

Decision tree for tree cover classification

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()
```



How to find node features and thresholds?

MORE HERE: at each node one of the available features needs to be compared to a threshold - how to select feature and threshold?, impurity measure, information gain, worked example on a decision stump and comparison to code

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

MORE HERE: show code, 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)
- Throughtout the sequence the model is improved by gradually refining the prediction.

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

OLD Background: Measuring surprise

- Shannon and Weaver (1948) developed a mathematical theory of communication
- As part of this they needed a mathematical measure of "surprise" after observing the outcome of a random variable, which has n possible outcomes with probabilities $p_1, \ldots, p_n$
- Based on simple considerations (additivity, continuity) they showed that if outcome k is observed, the only reasonable measure of surprise is

$$-\log_2 p_k$$

- A high-probability outcome is less surprising than a low-probability outcome.

OLD Background: Entropy

- Entropy is the negative expected surprise

$$H(p_1, \ldots, p_n) = \sum_{i=1}^{n} p_i \log_2 p_i$$

- Entropy is a summary measure of uncertainty or information content of a probability distribution.
- The higher entropy, the more "deterministic" a distribution is.
- TODO: A few bar plots of distributions with their entropy values.

OLD Background: Empirical entropy

- Given a set of values $S = (y_1, \ldots, y_n)$ where each y_i is either 0 or 1
- From S we estimate the probability that $y_i = 1$ by

$$\hat{p} = \frac{1}{n} \sum_{i=1}^{n} y_i$$

- The probability that $y_i = 0$ is then estimated by $1 - \hat{p}$.
- We calculate the empirical binary entropy of the sample S as the entropy of the distribution $(\hat{p}, 1 - \hat{p})$

$$H(S) = \hat{p} \log_2 \hat{p} + (1 - \hat{p}) \log_2 (1 - \hat{p})$$

OLD Decision stump

- We have a data set of input/output pairs $S = \{(x_i, y_i)\}_{i=1}^{n}$ where
- $x_i \in \mathbb{R}$ are continuous inputs (eg temperature) and
- $y_i \in \{0, 1\}$ are binary outputs (eg rain occurrence)
- We want to find a threshold τ to "optimally separate" the data set S into S_l and S_r by

$$S_l = \{(x_i, y_i) : x_i < \tau\} \quad S_r = \{(x_i, y_i) : x_i > \tau\}$$

- How to choose the threshold? What does "optimally separate" mean here?

OLD Decision stump

- After setting the threshold τ a fraction $q_l = \frac{|S_l|}{|S|}$ of data ends up left of the threshold and $q_r = \frac{|S_r|}{|S|}$ on the right
- Each set has its individual empirical entropy $H(S_l)$ and $H(S_r)$
-

Introduction

- Supervised ML
- Model training
- Examples, model evaluation

Tree-based methods

- Decision trees
- Random Forest, XGBoost

Neural Networks