Machine Learning

Data Science tools & Decision Trees

Dr Guillermo Hamity

University of Edinburgh

September 27, 2021

European Research Council

Established by the European Commission

Welcome to Machine Learning

About me

- Postdoctoral Research Associate working in ATLAS EXPERIMENT experiment on searches for long-lived particles.

Contact

- JCMB Room 3404
- Office hours
  - Mostly working from **home**
  - Try to be on site Thursdays
- Email: hamity.daml@pm.me

Credit

- Previous lecturers and slides: S. Palazzo (2020), A. Sogaard(2019)
- Book: Intro to ML in Python (Library online access)
- Yandex ML summer school 2018 (link on github)

Lectures outline

The Machine learning part of the course is divided in 4
Each unit comprises: 1 lecture + 1 workshop

1. ☒ **Lecture 2** + Workshop [1] (Sep 27):
   Data Science tools & Decision Trees
2. ☐ Lecture 3 + Checkpoint 1 (Oct 4):
   Intro to Neural Networks & Deep Learning
3. ☐ Lecture 5 + Checkpoint 2 (Oct 18):
   Intro to Convoluted Neural Networks
4. ☐ Lecture 6 + Checkpoint 3 (Oct 25):
   Intro to Generative & Adversarial Neural Networks

Conclude with a project utilising the skills learned during the lectures and workshops.

[1] checkpoint **not** assessed

Lectures and workshops

Lectures (1h) are Mondays 9:00—11:00 (JCMB LTB)

- Introduce machine learning techniques
- Total four hours of lectures (just scratching the surface)

Workshops (3h) are Wednesdays 10:00—13:00 (JCMB 4325D)

- Exercises provided in Jupyter notebook 

- Solve tasks provided in notebooks —TAs and Lecturer will offer some assistance
- Notebooks are submitted for assessment
  - can continue to work on them after workshop
- Lecture slides, exercises and example notebooks will be uploaded on Learn

Deadline for checkpoints

- **10 AM on the Friday** following the CP workshop
- Solutions uploaded to learn after deadline

Installing Anaconda with DAML packages

Students can work on their own laptops or the CP labs.
We will be using the Anaconda platform (installation instructions)
After installation need to install packages for DAML

Local Anaconda Setup (GUI)

1. Open up Anaconda Navigator
2. Click on `Environments` in the left menu
3. Pick `Import` from the buttons that have now appeared near the bottom.
4. Fill in the form as follows:
   - Name: `daml`
   - Specification File: Select the `daml-environments.yml` attached file (**download from Learn**)
5. Click `Import` (and wait till it downloads and installs the missing packages)

Alternatively from the terminal

```
conda env create -n daml -f /path/to/daml-environments.yml
```

Setup environment

Local Anaconda Setup (2/2)

6. To confirm that everything went smoothly, open a terminal and type `conda info -e`.
   You should see both the "base" and the "daml" environments, e.g.

```
(base) guillermo@pop-os:~$ conda info -e
# conda environments:
#
base                  *  /home/guillermo/miniconda
daml                     /home/guillermo/miniconda/envs/daml
```

7. To enable the daml environment, type:

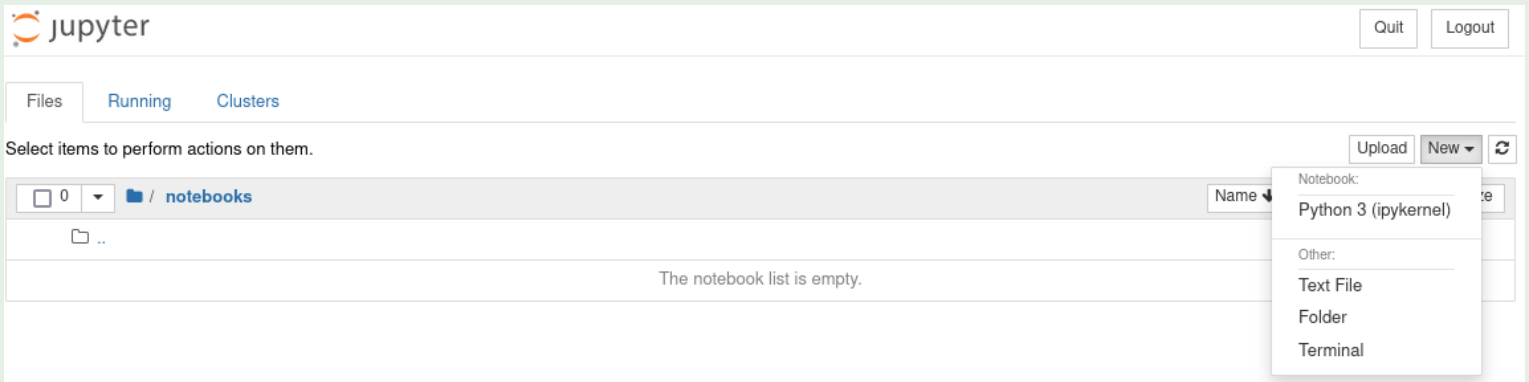
```
conda activate daml
```

8. To start a jupyter notebook, type:

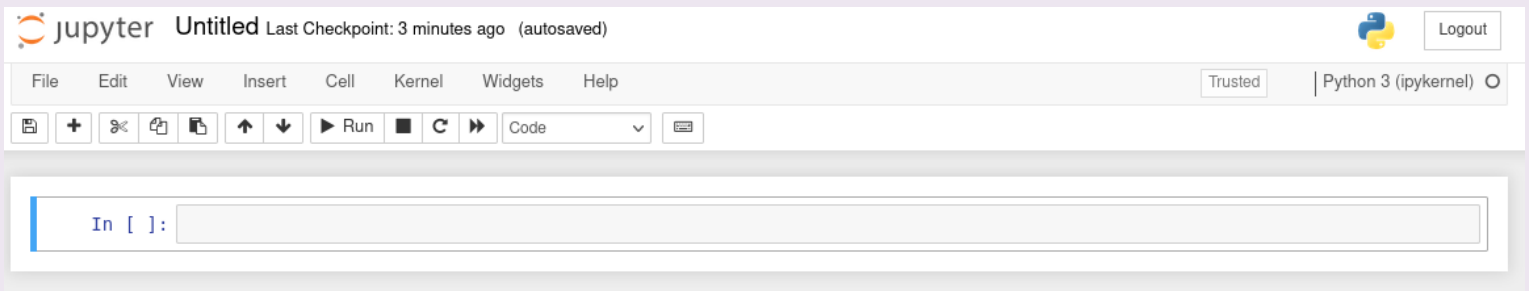
```
jupyter notebook
```

Intro to Jupyter

Example (New Jupyter notebook, choose Python3)

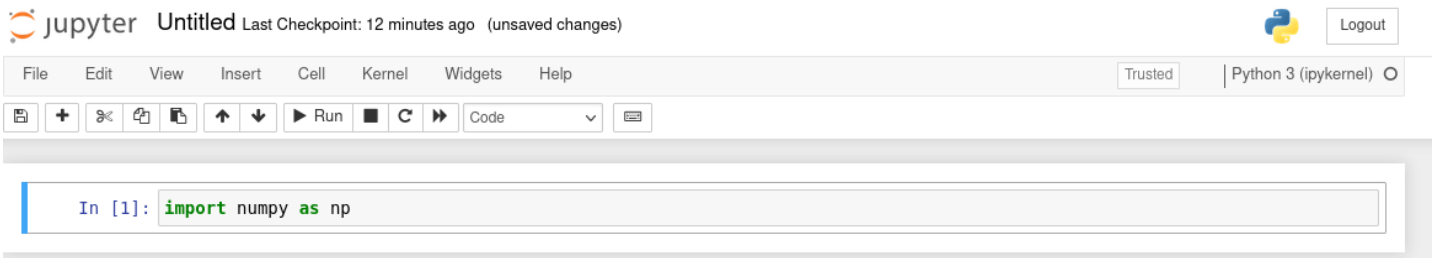


Notebook is ready with empty cell to start coding

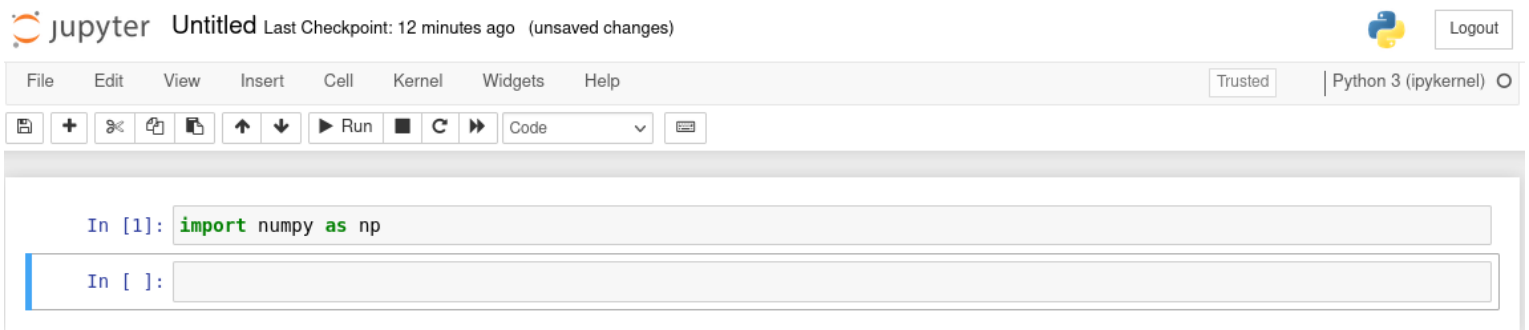


Execute a cell

- Write lines of code in an empty cell

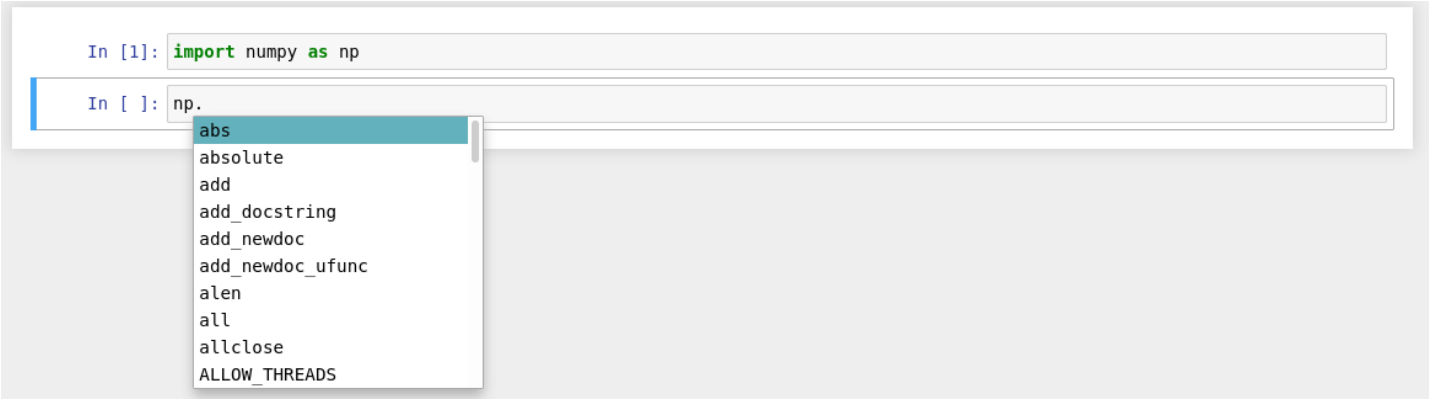


- `[Shift + Enter]` to execute code in the cell and make a new cell

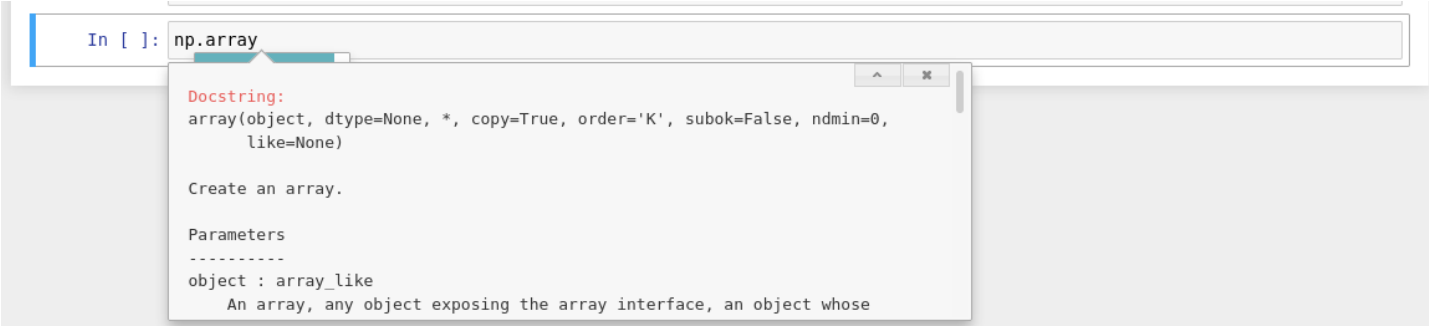


Methods and documentation

- Auto-complete by hitting [Tab]



- Documentation of a method with [Shift + Tab]



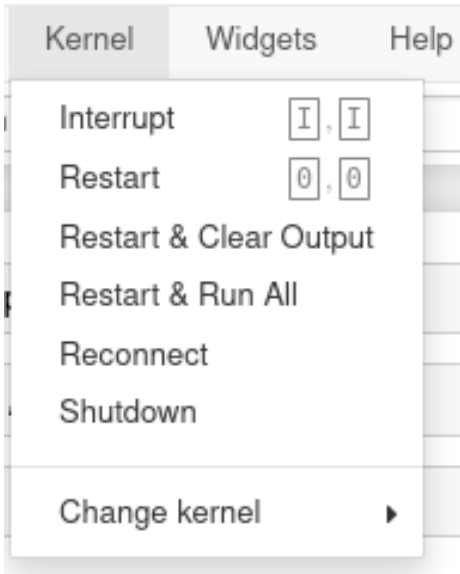
- or execute method with question mark: np.array?

Python kernel

- Note that cells can be executed out of order

```
In [1]: import numpy as np
In [2]: a = np.array([5,3,65])
In [5]: print(a)
        5
In [4]: a=5
```

- If this gets too messy or there is a crash, restart the python Kernel →



Important

Don't submit code that can't be re-run sequentially, ie: running 

Data science libraries

We will use core pakages of the SciPy python-based ecosystem.

- `NumPy` 

Base N-dimensional array package (docs)

```
import numpy as np
```

- `Pandas` 

powerful tool for data structures, analysis & manipulation (pandas in 10 min)

```
import pandas as pd
```

- `Matplotlib` 

Comprehensive plotting library with interface module `pyplot` (docs)

```
import matplotlib.pyplot as plt
```

- A notebook with useful commands is provided as reference `data-science-tools.ipynb`

Machine Learning libraries

Machine learning (ML) python-based libraries.

- **Scikit-learn** 

common ML library for python. It supports various ML algorithms. It is focused on data modelling.

```
import sklearn
```

- **Tensorflow** 

ML open source library for python. It supports model building for multi-dimensional data e.g. Neural-Networks (NNs).

```
import tensorflow as tf
```

- **Keras** 

acts as is a high-level API for tensorflow. It supports several tools to facilitate creating NNs.

```
import tensorflow.keras
```

Why Machine Learning?

Consider expert knowledge and physical models VS machine learned models

1. Would you apply machine learning for. . . .
   - . . . Newtonian mechanics?
   - . . . recommending shopping items to online customers?
   - . . . sorting integers?
   - . . . sorting strawberries?
2. Machine Learned models are powerful when
   - little prior knowledge exist
   - systems are too complex for manual examination
   - sample data of sufficient size is available

Consider point 1 in light of point 2

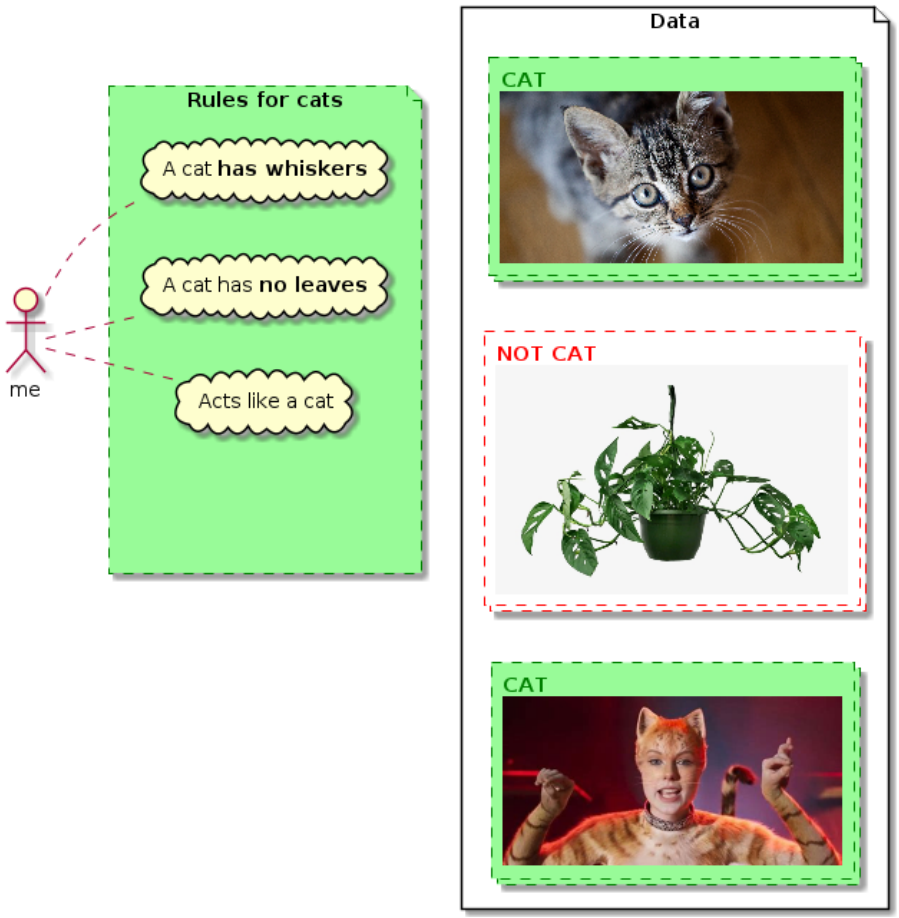
Why Machine Learning?

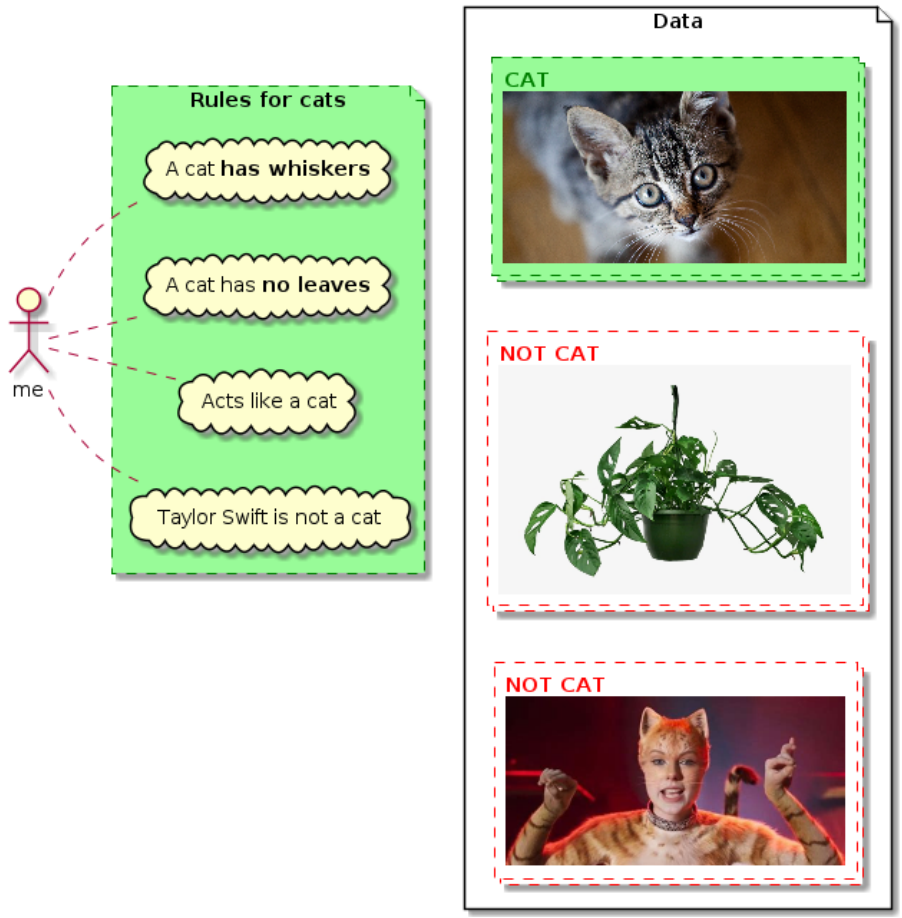
Designing models for complex systems is a difficult job for humans

- Designing rules requires a deep understanding of how a decision should be made
- Many problems are easily solved with hard-coded rules
  - e.g spam filters based on freq. of certain words
- but not others
  - e.g. identifying images with cats in them

Why Machine Learning?

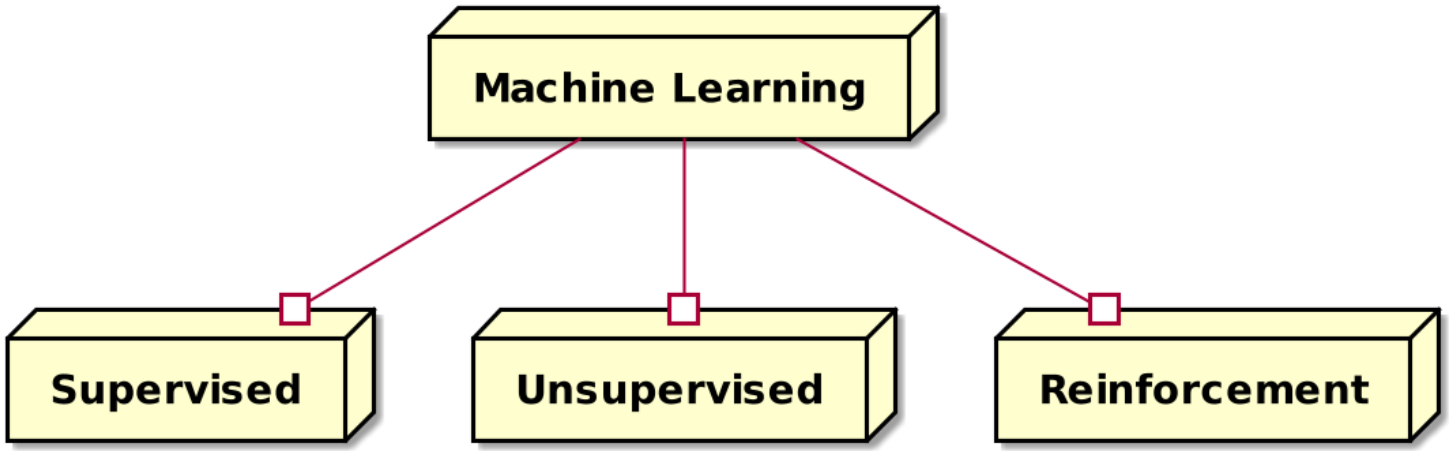
Changing problem domain even slightly might require a rewrite of the whole system





Paradigms of ML

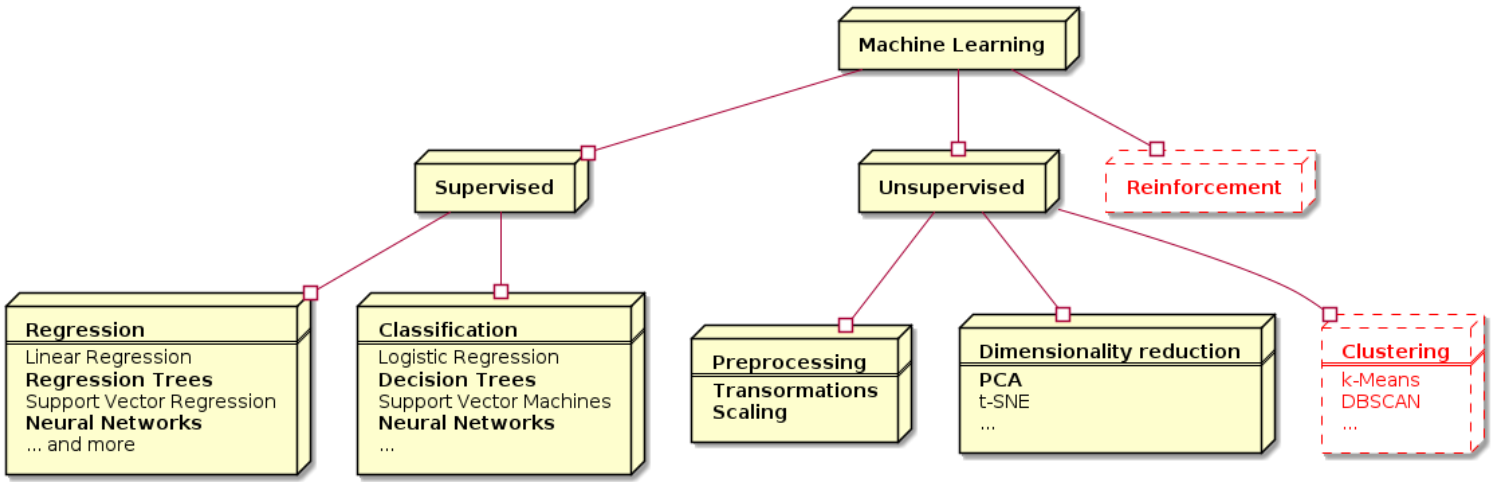
ML algorithms can be classified under three main topics



- Supervised
  - Used when you know the desired output (e.g. **labelled** data)
  - Find model h to map **features** X to **dependent variables** y: $y = h(X)$
- Unsupervised
  - **Only** the **input data** is known, and no known output data is given to the algorithm
  - Used to **extract features/information** from data
- Reinforcement
  - Algorithms learn in long term reward
  - Trains by "playing" with the enviroment

Algorithms in ML

ML is a huge field, we will only be scratching the surface



- Will focus on **Supervised Learning**
  - Either **Regression or Classification**
  - Focus on Trees and Neural Networks (DNN, CNN, VAEs, GANs)
- Will supplement with some **Unsupervised Learning**

Supervised Learning

- Consider dataset with several instances.
- Each instance i has **features** ($\boldsymbol{x}_i$): attributes associated with the data (e.g. height, length, wight,...)
- An unknown function $f : \mathbb{X} \to \mathbb{Y}$ maps to dependent features y_1, y_2.

$$y_i = f(\boldsymbol{x}_i) \text{ for } i = 1, 2, ...$$

- The machine leaning problem: find a plausible **hypothesis** function $h : \mathbb{X} \to \mathbb{Y}$ from hypothesis space $\mathbb{H}$
- We have to quantify how good our hypothesis is.
  - Error of of hypothesis h is deviation from true f
  - Known as **loss function**, an example:

$$Q(h, X^\ell) = \frac{1}{\ell} \sum_{i=1}^{\ell} (f(\boldsymbol{x}_i) - h(\boldsymbol{x}_i))^2$$

Regression vs Classification

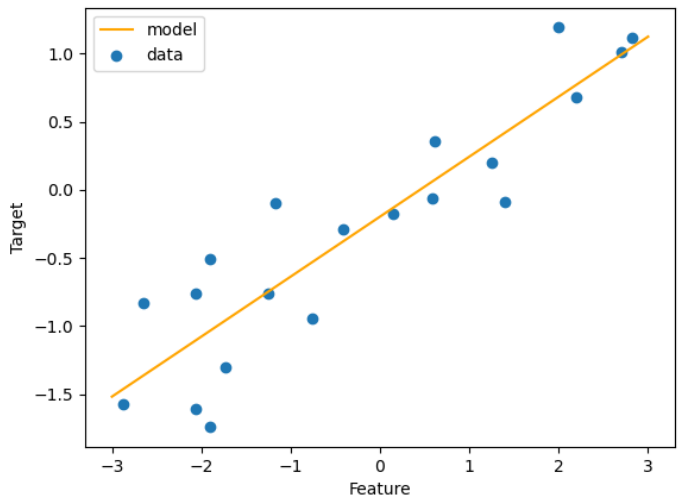
If $f(\boldsymbol{x})$ outputs real values call this **regression**

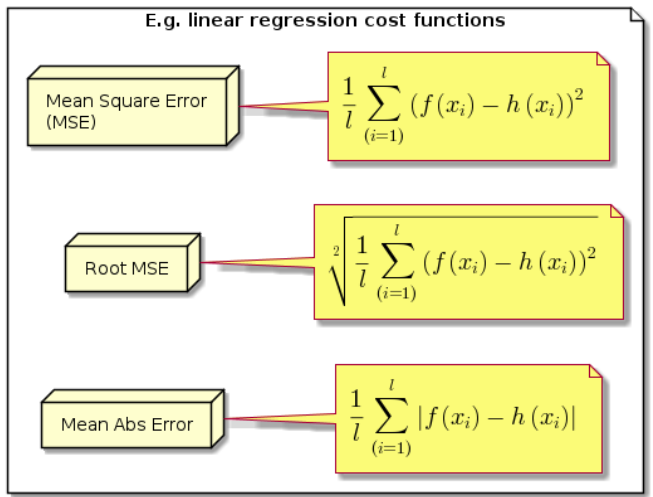
Example (Linear regression)

- Single feature x
- Single dependent target y
- Data $X^\ell = \{(x_i, y_i)\}_{i=1}^{20}$
- Linear model:

$$y_i = h(x_i, \boldsymbol{w}) = w_1 x_i + w_0$$

- Find constants $\boldsymbol{w} = (w_0, w_1)$ given X^ℓ
- Done by **minimising loss function**
  - Numerical derivative (e.g. gradient decent)
- Choice in **loss function** and **minimisation** algorithm





Regression vs Classification

If $f(\boldsymbol{x})$ outputs integer values call this **classification**

Example (Linear classification)

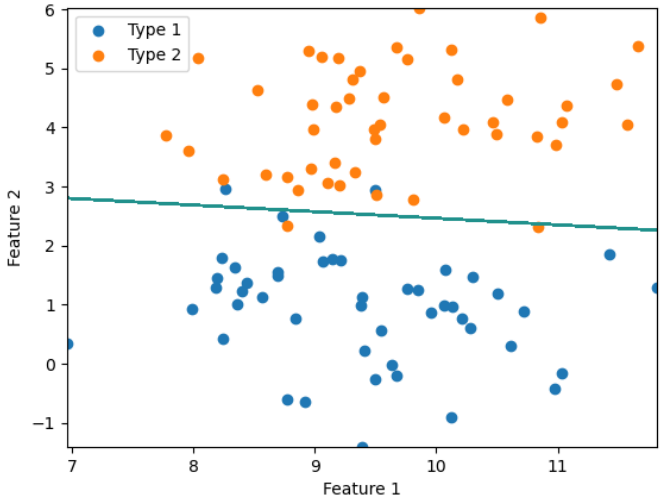
- Target is labels $yi \in \{-1, 1\}$
- **Remember** Linear regression:

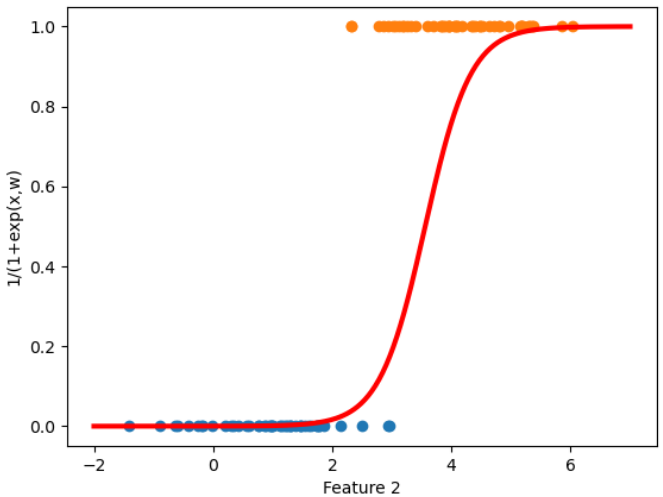
$$y_i = h(\boldsymbol{x}_i, \boldsymbol{w})$$

- **Now** Linear classification:

$$y_i = \text{sign}(\boldsymbol{w}^T \boldsymbol{x}_i)$$

- Choice of sigmoid function
  - Fit $h(x) = \frac{1}{1+\exp(\boldsymbol{x},\boldsymbol{w})}$
  - **Minimisation** $\Sigma_{i=1}^{\ell} \frac{1}{1+\exp(\boldsymbol{x}_i,\boldsymbol{w})} \rightarrow \min$





Regression vs Classification

- Doing this with **scikit-learn** is just a few lines of code

```
#Get regression data
X ,Y = make_wave()

from sklearn.linear_model import LinearRegression
linear_model = LinearRegression().fit(X, Y)
#get accuracy score: acc=1 is perfect prediction
acc = linear_model.score(X,Y)
#get constants from the fit
w0 = linear_model.intercept_
w1 = linear_model.coef_
print("y = {0:.2g}*x + {1:.2g}, accuracy {2:.2g}".format(w1[0],w0,acc))
```

```
y = 0.44*x + -0.2, accuracy 0.82
```

```
#Make a prediction for 10th value of x
y_pred = linear_model.predict(X[10].reshape(-1,1))
print("X {0:.2g}, Y {1:.2g}, y_pred =
    {2:.2g}".format(X[10,0],Y[10],y_pred[0]))
```

```
X -2.9, Y -1.6, y_pred = -1.5
```

Our first application: Classification of Iris species

- Iris dataset used for ML classification problems is a popular starting point

Features length and width of petals, length and width of sepals

Labels species of Iris, either setosa, versicolor, virginica

Iris Versicolor | Iris Setosa | Iris Virginica

Goal buld a ML algorithm that ca learn from the measurements, whose species is known, and predict the species for a new Iris.

The full example is given in `lecture2.ipynp`

Data visualisation: Tables

- `pandas.DataFrame` are tables of `pandas.Series` (columns)
  - Viewing first 5 rows (indexes)

```
df.head(5)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	type	target
0	5.1	3.5	1.4	0.2	setosa	0
1	4.9	3.0	1.4	0.2	setosa	0
2	4.7	3.2	1.3	0.2	setosa	0
3	4.6	3.1	1.5	0.2	setosa	0
4	5.0	3.6	1.4	0.2	setosa	0

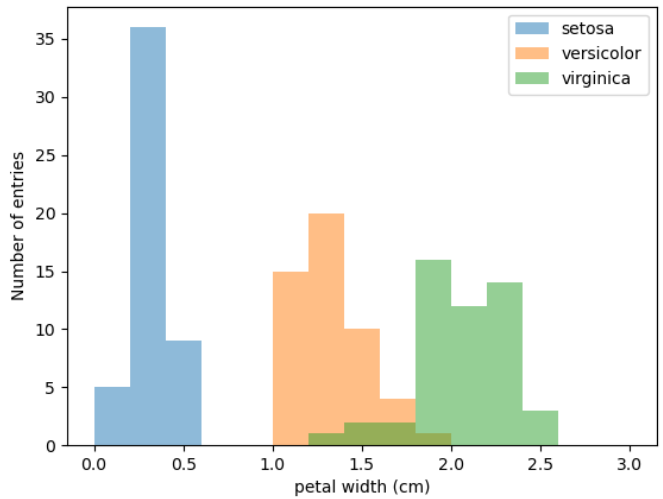
```
df.describe()
```

  - Viewing statistical summary of columns

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

Data visualisation: Histograms

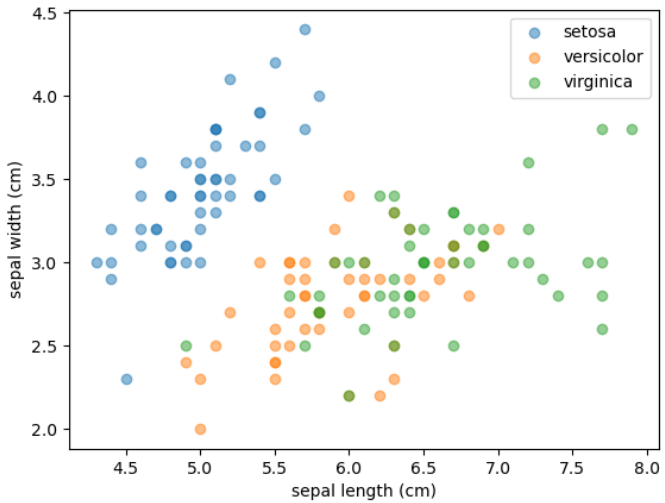
```
# Import[s]
import matplotlib.pyplot as plt

# Feature to plot
feat = 'petal width (cm)'

# Bin range
bins = np.linspace(0, 3, 15 + 1,
    endpoint=True)

# Create figure and axis objects.
fig, ax = plt.subplots()

for t in iris.target_names:
    # Boolean mask
    mask = df['type'] == t

    # Make histogram for current type
    ax.hist(df[mask][feat], bins=bins,
        alpha=0.5, label=t)
    pass
# Decorations
ax.legend()
ax.set_xlabel(feat)
ax.set_ylabel("Number of entries")
fig.show()
```



- Histograms help in visualizing the data that we have.
- Each histogram represents the number of entries of the feature (e.g. petal width), for each species.

Data visualisation: Scatter plot

```
# Features to plot
featx = 'sepal length (cm)'
featy = 'sepal width (cm)'

# Create figure and axis objects.
fig, ax = plt.subplots()

for t in iris['target_names']:
    # Boolean mask
    mask = df['type'] == t

    # Scatter plot for current type
    ax.scatter(df[mask][featx],
               df[mask][featy],
               label=t, alpha=0.5)
    pass

# Draw legend
ax.legend()
ax.set_xlabel(featx)
ax.set_ylabel(featy)
fig.show()
```

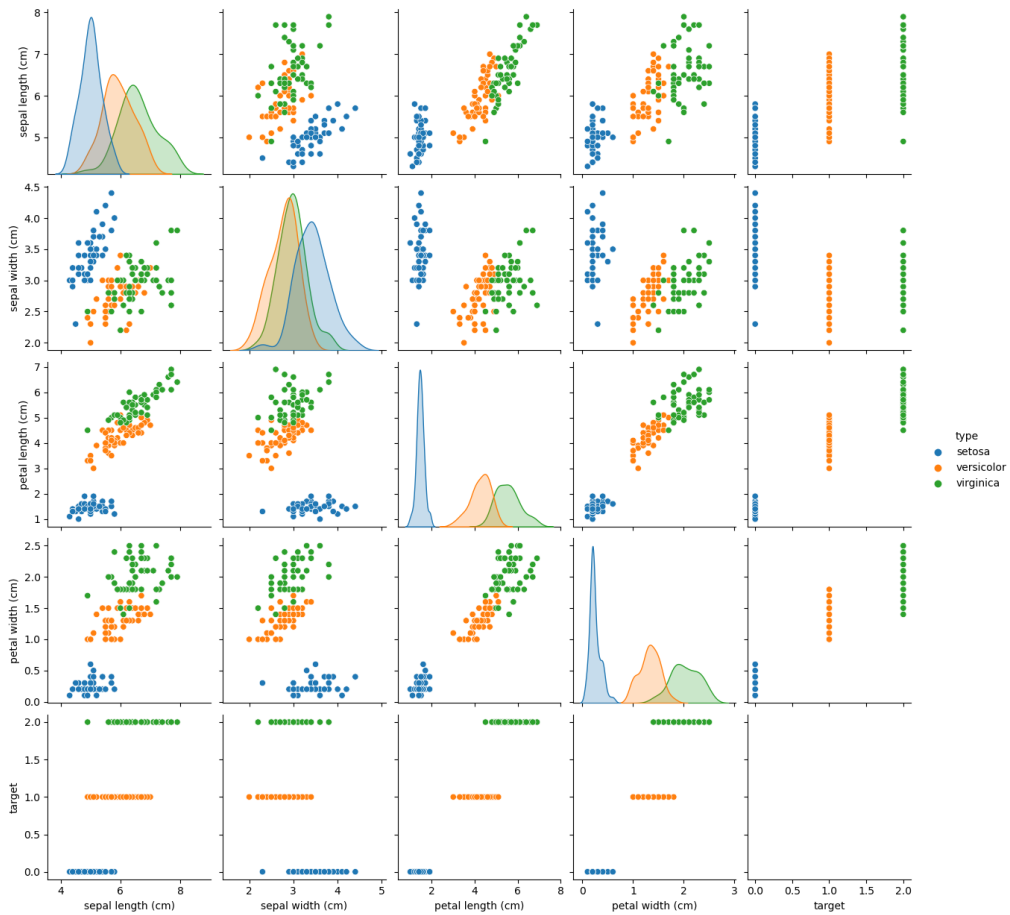


- The scatter plot shows 2 features for the 3 species of the Irises.
- The x axis has the sepal lenght while the y axis has the sepal width.

Data visualisation: Pair plot

- `seaborn.pairplot` method to facilitate visualisation of features

```
fig = sns.pairplot(df, hue = 'type');
```



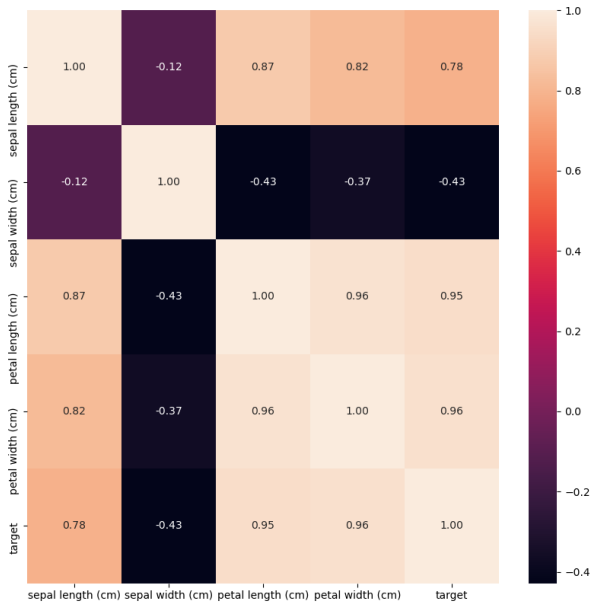
Data visualisation: Correlations

- Correlation

$$r_{xy} = \frac{\Sigma(x_i - \bar{x}) \times (y_i - \bar{y})}{\sqrt{\Sigma(x_i - \bar{x})^2}\sqrt{\Sigma(y_i - \bar{y})^2}}$$

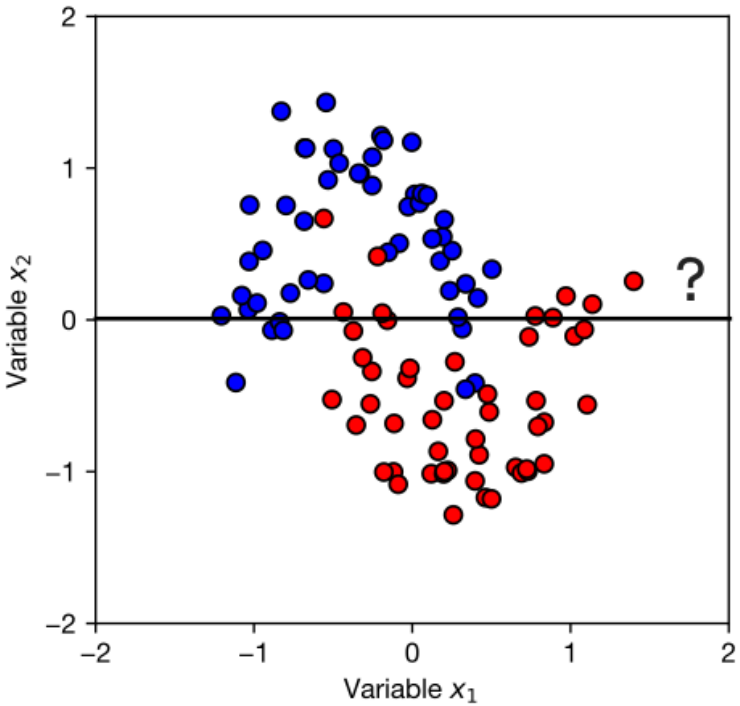
- Shows relationship between different features
  - $r_{xy} \to 1$: one feature increases, the other increases
  - $r_{xy} \to -1$: one feature increases, the other decreases
- Highly correlated features may encode similar (duplicate) information, and could consider dropping one from models

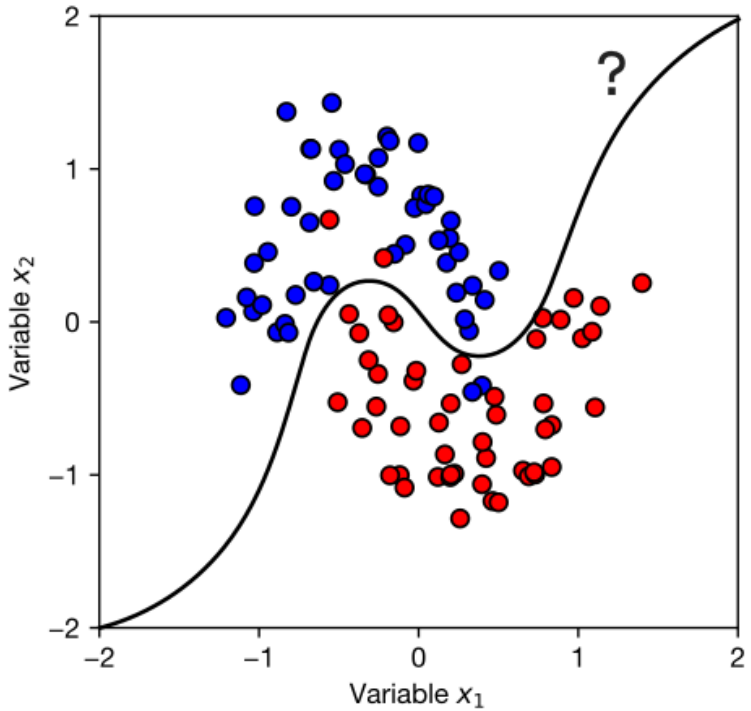
```
#Get the correlation matrix
corr_matrix = df.corr()
fig, ax = plt.subplots(figsize=(10,
    10))
# Generate Heat Map, allow annotations
    and place floats in map
sns.heatmap(corr_matrix, annot=True,
    fmt=".2f")
plt.show()
```



Classification problem

- Consider classification problem where simple linear classification is not sufficient.
- How do we find decision boundaries in such a case? Ever more complex polynomials?



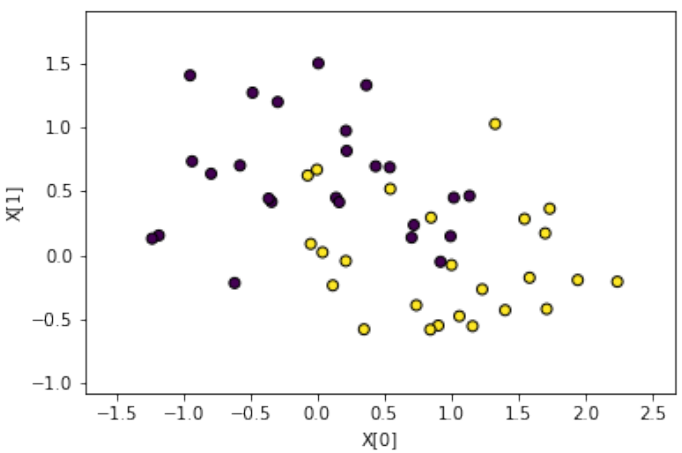


- Non trivial to choose functional form for higher dimensions and non-trivial correlations
- Decision trees provide a type of general solution to the problem

Decision trees (Depth 0)

- Decision trees build simple rules for partitioning the feature space based on selection in individual features $x_i < \theta_i$
- The first such decision is called the **root node**

- Consider case with two features



- Need to find boundary which gives **best separation** of our targets

Classification

Choices for split quality criterion, both based on class probability in node

- Gini

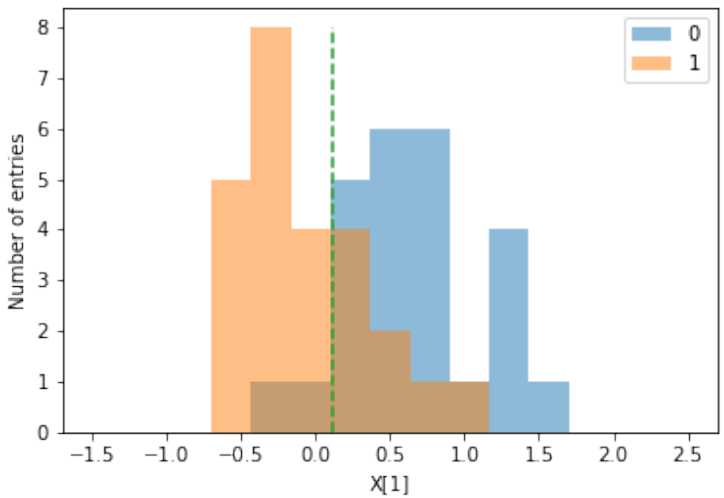
$$1 - \Sigma_{\text{class}} p_c^2$$

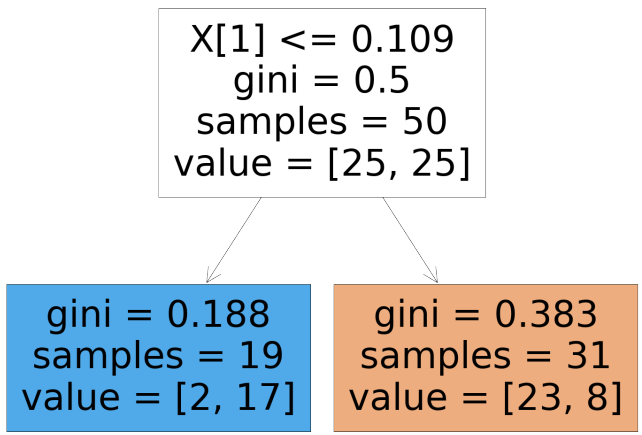
- Entropy

$$-\Sigma_{\text{class}} p_c \log(p_c)$$

Decision trees (Depth 1)

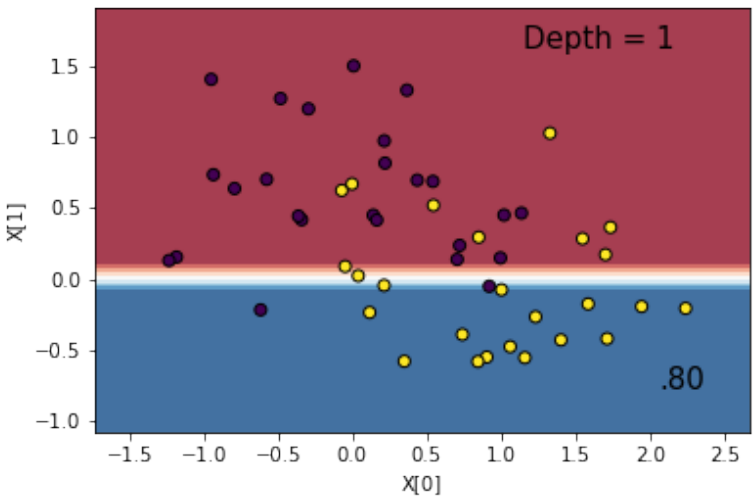
- Algorithm searches each intermediate value in each feature (`X[0]` and `X[1]`)

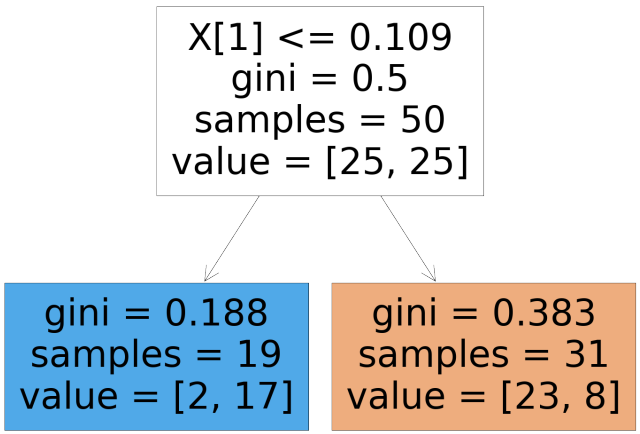




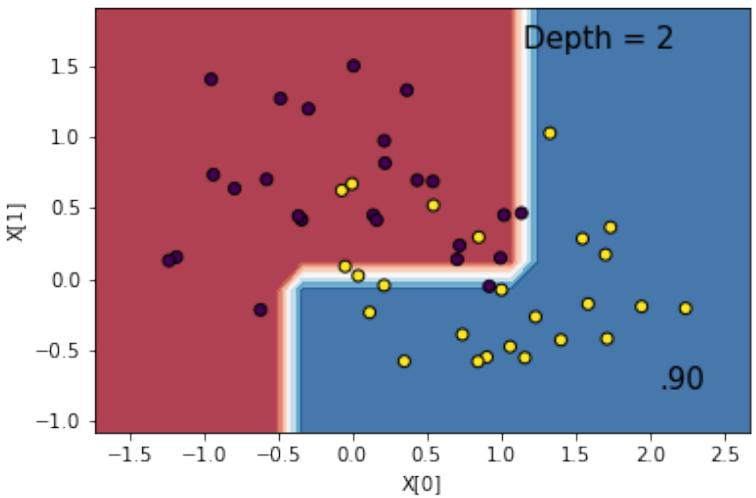
- **Root node** is chosen as threshold with lowest `Gini` score
  - Left: $1 - \Sigma_{\text{class}} p_c^2 = 1 - \frac{2}{19} - \frac{17}{19} = 0.188$
  - Right: $1 - \Sigma_{\text{class}} p_c^2 = 1 - \frac{23}{31} - \frac{8}{31} = 0.383$
  - Weighted average: $\frac{19}{50} \times 0.188 + \frac{31}{50} \times 0.383 = 0.309$

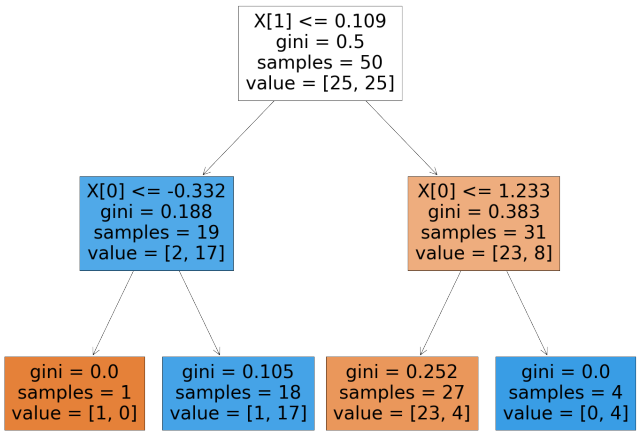
Decision trees (Depth 1)





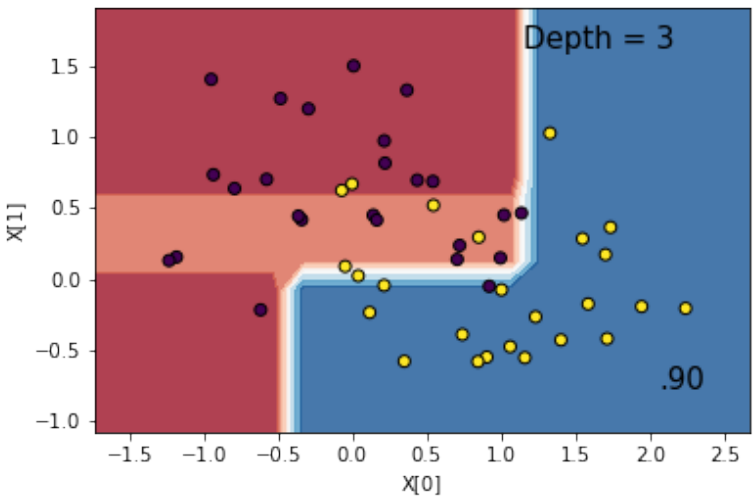
Decision trees (Depth 2)

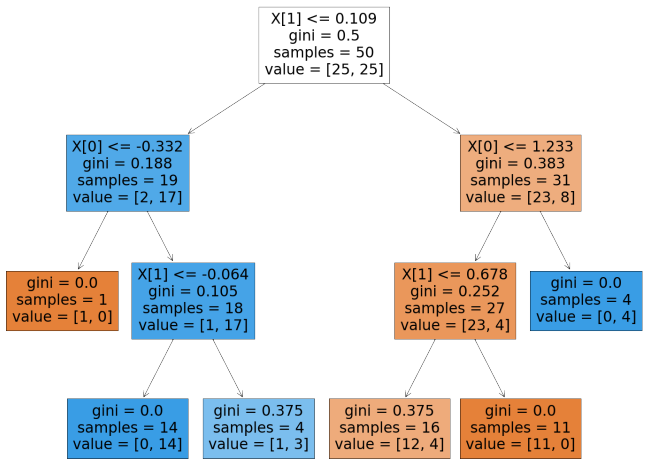




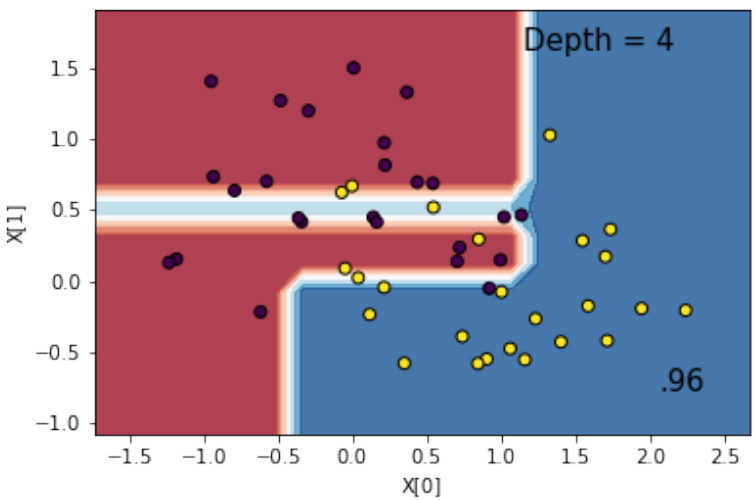
- Two of the 2nd layer nodes are **pure** (only have 1 type) and are no longer split.
- Internal nodes are called **branches** and final nodes are **leaves** of the trees

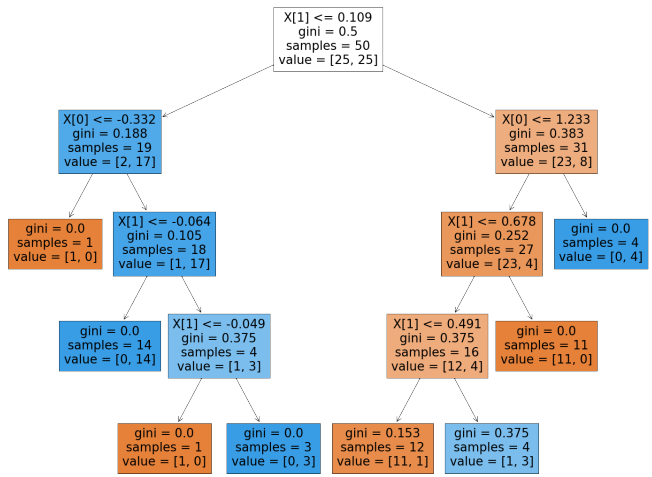
Decision trees (Depth 3)



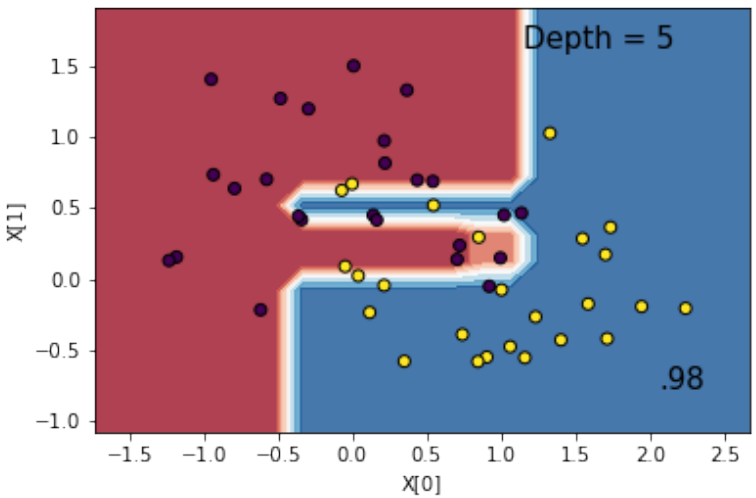


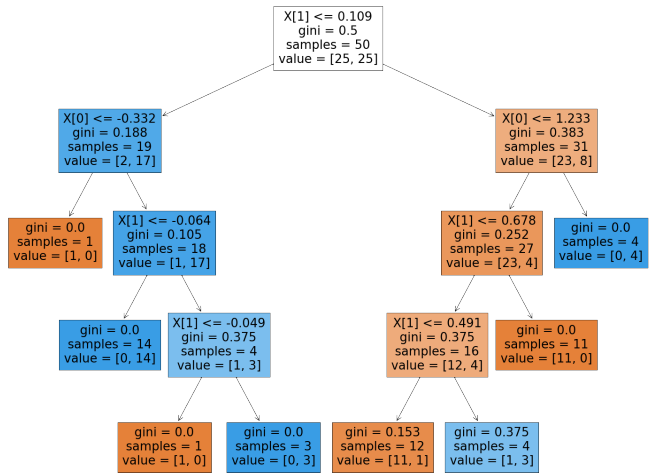
Decision trees (Depth 4)



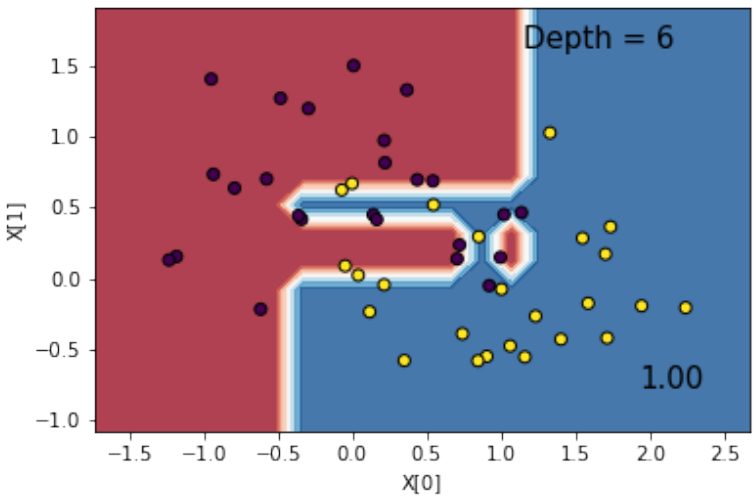


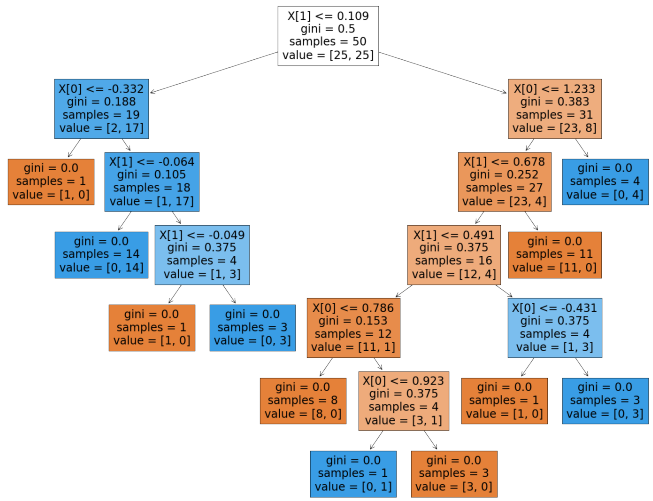
Decision trees (Depth 5)





Decision trees (Depth 6)





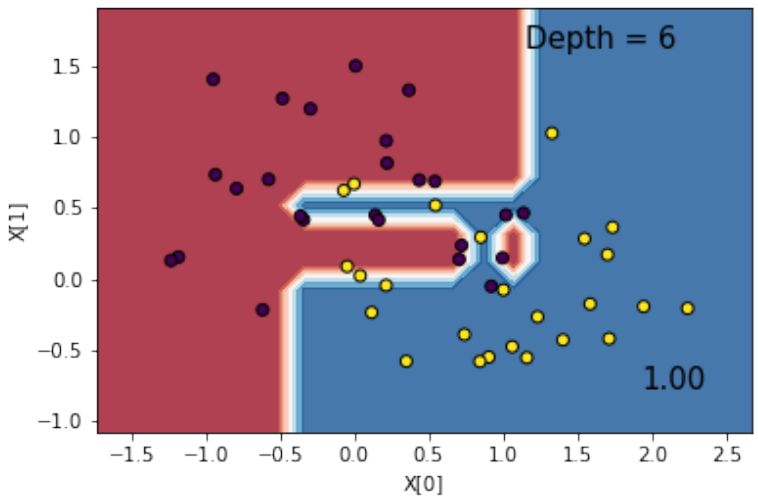
- Accuracy of the model has increased to 100%!
- Is this a good thing?

Decision trees (Depth 6)

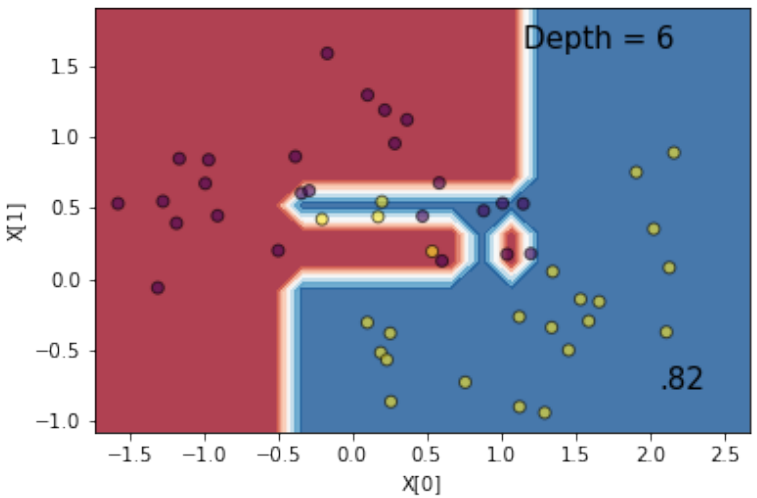
Over-fitting

- Decision trees allowed to achieve high purity will tend to overfit the training data
- Checking on a test sample shows much lower accuracy

Training Sample



Testing Sample



Decision trees

Remedy for over-training is to prune our trees (early stopping before we over-fit)

Stopping criteria

- Maximum tree depth
- Minimum number of objects on leaf
- Maximum number of leafs on tree
- Minimum improvement

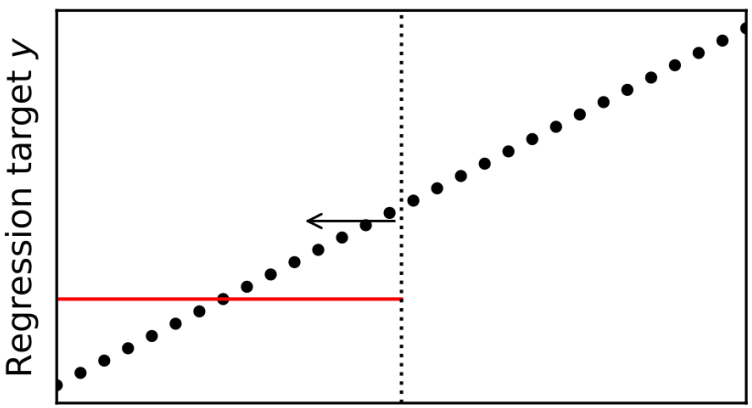
```
Init signature:
DecisionTreeClassifier(
    *,
    criterion='gini',
    splitter='best',
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    max_features=None,
    random_state=None,
    max_leaf_nodes=None,
    min_impurity_decrease=0.0,
    min_impurity_split=None,
    class_weight=None,
    ccp_alpha=0.0,
)
```

Usually choosing one such criteria is enough, but ideally requires optimisation.

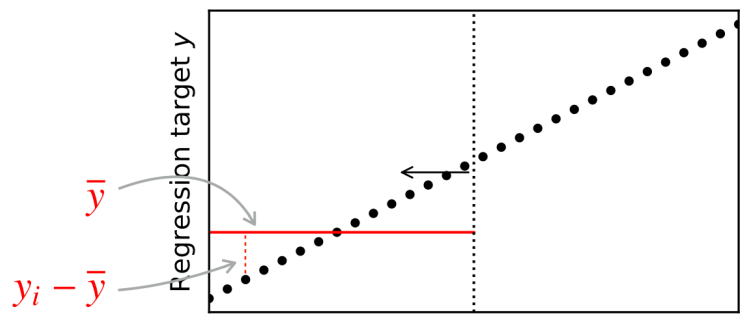
What about Regression?

Regression trees follow in the same manner, but in this case we **predict a real value**

- Choose a boundary in feature space



- The prediction y_i is the average of the target value in the node



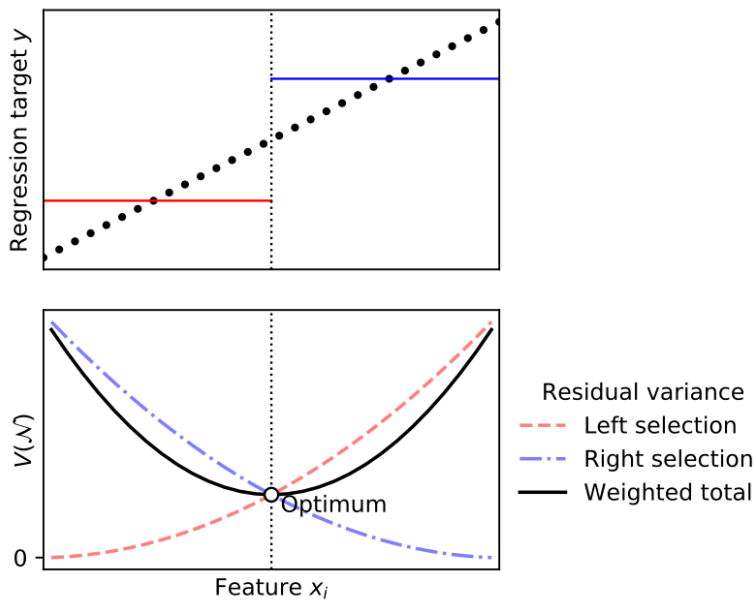
- Similar pitfalls as DTs: splitting to only single values where $\bar{y} = y_i$ would lead to over-fitting.

Regression Trees

- The boundary is selected such that we minimise the **residuals** of the predictions (like in linear regression)

$$V(N) = \frac{1}{N} \sum_{i \in N} (y_i - \bar{y})^2$$

- Compute **average V(N) weighted** by number of examples in each node

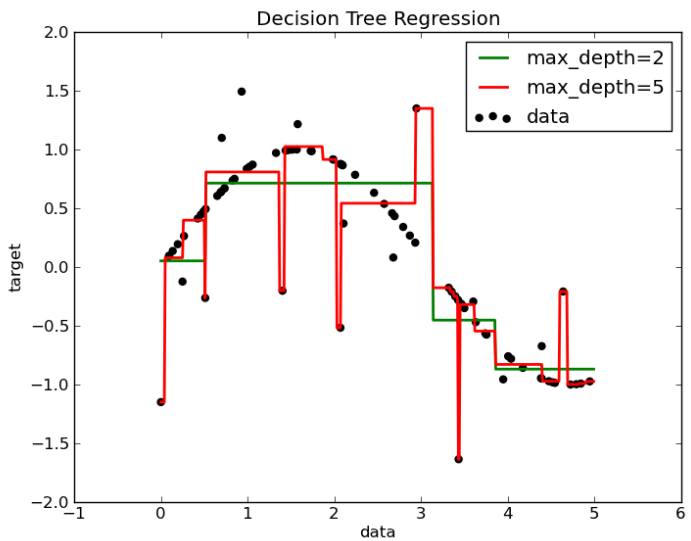


Regression Trees

- The regressor prediction are averages in different nodes
- Looking at the example here, you may think the regression tree is not suitable
  - Depth 2 predicts quite badly, depth 5 clearly overfits!
  - So yes you are right! Fitting a function would serve better!

But consider multidimensional problems

- Target: max value of loan to approve
- Features: income, debt, employment, marital status, home owner, ...
- Here a regression tree will shine!



Summary of Decision/Regression trees

Pros

- Intuitive
- Transparent/Interpretable
- Suitable for variable selection.
- Can easily capture Non-linear patterns
- Minimal preprocessing,e.g.
  - no feature normalisation needed
  - small datasets

Cons

- Prone to overfitting
- Discontinuous (regression predictions)
- Restrictive compared to other methods (e.g. NNs)
- Susceptible to variance in data, can result in the different decision tree.
- Are biased with imbalance dataset, so recommended to balance out the dataset before training.
- Regression DTs don't extrapolate from training dataset

Ensemble of Decision Trees

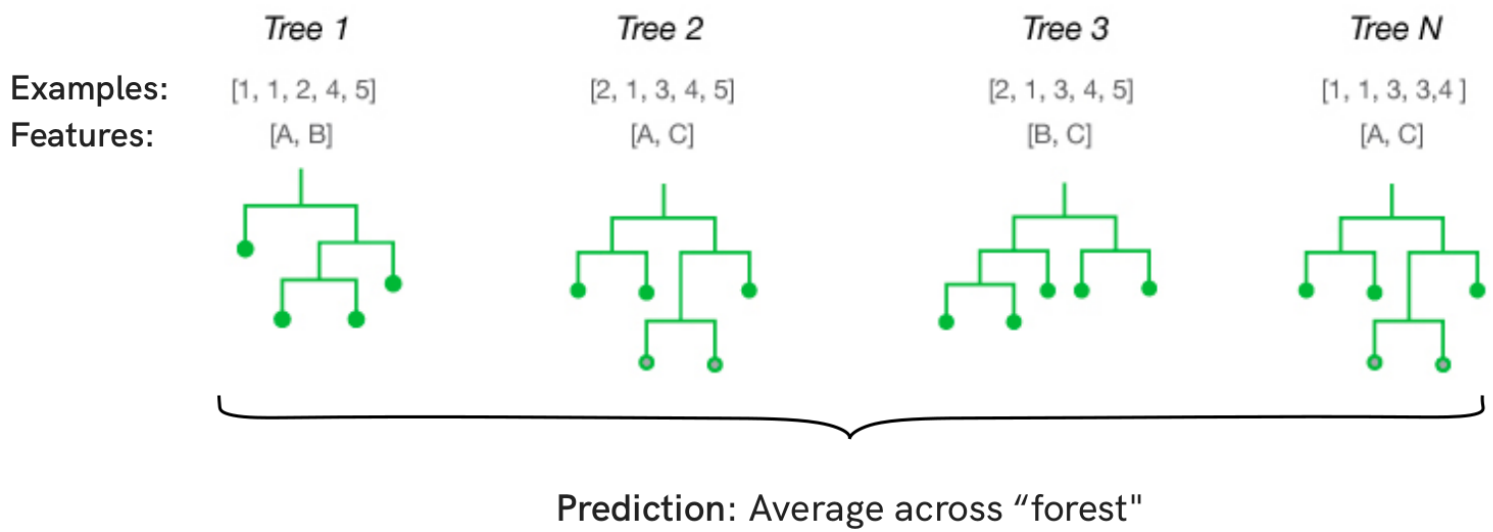
- Individual DTs are simplistic, prone to overfitting, etc.
- However, possible to construct ensemble of DTs which improves performance.
- Two methods discussed here:
  - **Bagging**: train several DTs independently from each other and combines them in averaging process
  - **Boosting**: For a sequence of DTs, make each DT learn from the mistakes of the previous one.

Bagging: Random Forests

- Bagging stands for (B)ootstrap (agg)regation
- **Bootstrapping**: Uniformly, randomly sampling dataset (X^*Y^*) with replacement. e.g.:
  - Dataset: `[A,B,C,D]`
  - Random bootstrapping:
    - `[B,B,C,D]`
    - `[D,A,D,A]`
    - etc.
- **Bagging**
  - Generate N bootstrapped pseudo-datasets
  - Fit N DTs, fitted on each pseudo-dataset
  - Ensemble prediction is average of predictions
    - **Classification**: Each DT gives a probability for classification
    - **Regression**: Average of predictions
- **Randon Forest**
  - Bagging + use random subset of features

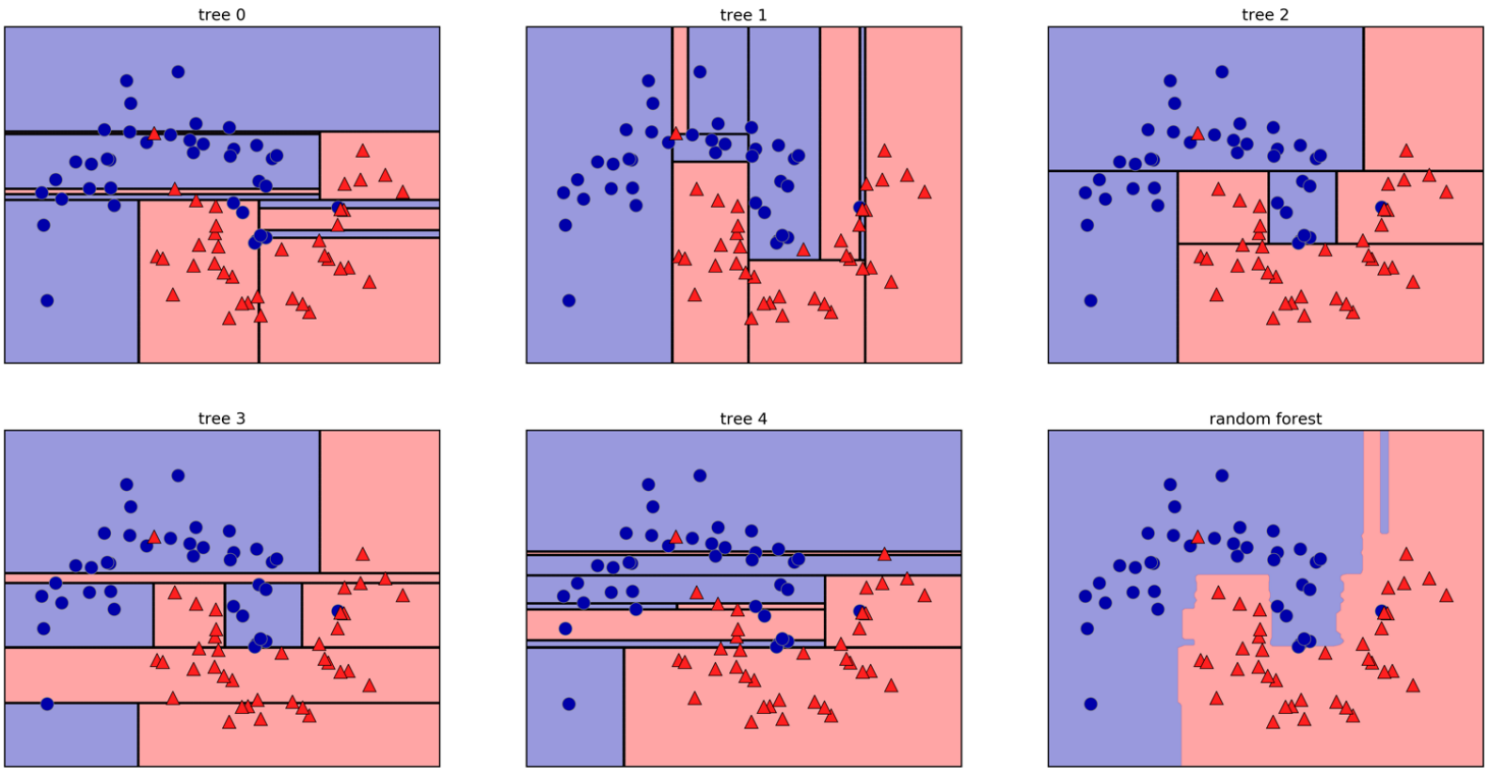
Random forest

- Trees will be built completely independently from each other the algorithm will make different random choices for each tree to make sure the trees are distinct.
- Each node in a tree can make a decision using a different subset of the features.
  - each split in each tree operates on a different subset of features.
  - this ensure that all trees in the random forest are different.



Random forest

- Random forest will train a number of trees to build (the `n_estimators` parameter of `RandomForestRegressor` or `RandomForestClassifier`).
- The random forest overfits less than any of the trees individually, and provides a much more intuitive decision boundary.



Gradient Boosting

- One of the most famous enseble boosting method is the Gradient Boosted Regression/Classification Tree
- Gradient boosting works by building trees in a serial manner:
  - Not random (unlike random forests)
  - Trees tend to be shallower in depth (1-5 layers) which makes the model smaller in terms of memory and makes predictions faster.
  - Add up many weak learners, each learning some part of the data better, while poor in others.
  - Each tree tries to correct the mistakes of the previous one.
- Apart the number of trees and pre pruning in the ensemble, another important parameter of gradient boosting is the learning rate
  - controls how strongly each tree tries to correct the mistakes of the previous trees.
- BDTs often need finer tuning of parameters to get good results

Time to practice

Three jupyter notebooks to play with

1. `data-science-tools.ipynb`
   - Summary of `numpy`, `pandas`, and other general python things
2. `lecture2.ipynb`
   - Example for Iris dataset: processing, visualisation, decision trees (with solutions)
3. `workshop_ml_problems.ipynb`
   - Example for wine quality dataset: processing, visualisation, decision trees (without solutions)
   - Not marked, so use it a testing ground

Note

- Several exercises will have several ways to achieve a solution (numpy, pandas, user functions)
- You are generally free to find your own solution, but encouraged to use/learn from examples you see here.