Day 1 - Introduction

May 29, 2024

Machine learning in practice

28.05.2024

Machine learning (ML)

Make a Machine learn by experiences without explicit instructions or "self-teaching computers".

Learn a function f that maps from input data X to produce an output y without specifying relationship between X and y.
Term ml was coined by Arthur Samuel at IBM in 1959.

Machine learning (ML)

Imperfect definitions

- Linear regression does not fit the definition of ml.
  Find best β such that $y = X\beta + \epsilon$, where ϵ is random noise that follows a normal distribution.
- Similar is the case for most other models from Statistics, *e.g.* logistic regression etc.
- They are considered ml algorithms in Machine Learning:
  A Probabilistic Perspective by Kevin P. Murphy,
  Pattern Recognition and Machine Learning by Christopher Bishop.

Taxonomy of machine learning algorithms

1. Supervised learning
2. Unsupervised learning
3. Semi-supervised learning
4. Reinforcement learning

1. Supervised learning

***Given: Labelled data (*X*,* y*). Task: See* X*, predict* y

Examples:

- X: Tissue gene expression samples, y: tissue name, *e.g.* blood, liver, etc.
- X: Tissue DNA methylation samples, y: Binary cancer class
- X: Bone X-ray Images, y: Binary fracture Class

Classification and regression

- All previous examples are for a classification task
- Label y can be continuous, *e.g.* it can be proportion of cell types within a tissue gene expression samples or it can be cancer severity scores $\rightarrow$ Task of Regression
- In classification, there is no order between classes (*e.g.* either Dogs $>$ cat or cat $>$ dog is wrong).
- Fuzzy boundaries between classification and regression: In some cases, classification is a special case of regression. *e.g.* Binary cancer classification of cancer and no cancer of a tissue sample depends on the amount of cancer or specific cell states.

2. Unsupervised learning

Given: Unlabelled data (X). Task: Find clusters of X, learn representation of X, generate more X.

Examples:

- X: Tissue gene expression samples
- X: Tissue DNA methylation samples
- X: Bone X-ray Images

3. Semi-supervised learning

Given: Labelled data (X_1, y_1) + Unlabelled data (X_2). See X_1, X_2 and predict y_1 with better generalization, performance or with limited labels.

Examples:

- X_1: Tissue gene expression samples, y_1: tissue name, *e.g.* blood, liver, etc.
  X_2: some tissue gene expression samples
- X_1: Tissue DNA methylation samples, y_1: Binary cancer class
  X_2: some Tissue DNA methylation samples
- X_1: Bone X-ray images, y_1: Binary fracture class
  X_2: some bone X-ray images X_2
- Generally, $X_1 \cap X_2 = \emptyset$, *i.e.* No sample in X_1 and X_2 are same.

3.1 Self-supervised learning

Given: Unlabelled data (X), learn y.

- X: Given a sequence, y: predict next word. *e.g.* generative pretrained transformers (GPTs).

SSL is useful in data where there is a structure i.e. language, protein sequences, gene expression programs etc.

4. Reinforcement learning

> ***Training an agent to achieve an objective through iterative feedbacks***
>
> Find the best policy that achieves an objective by setting appropriate rewards (think of, training a dog with food.)

Course organization

Day 1: Supervised learning

- Talk (30-45 minutes)
  1. Data setup and filtering
  2. Machine learning classifiers
     a. Random Forest
     b. Support vector machines
     c. Multi-layered perceptrons (MLPs)
- Hands-on tutorials
  1. Benchmarking data transformations and machine learning algorithms for single-cell RNA sequencing data

Course organization

Day 2: Unsupervised learning (Representations)

- Talk (45 minutes)
  1. Autoencoders (AEs)
  2. Bias and variance
- Hands-on tutorials
  1. Learning compact representations of single-cell data using AEs
  2. Using autoencoder representations

Course organization

Day 3: GNNs and transformers

Given by Darius Schaub

Tentative schedule:

- Talk (1 hour)
  1. GNNs
  2. Transformers
- Hands-on tutorials
  1. Applications on imaging / gene expression data

Data preparation

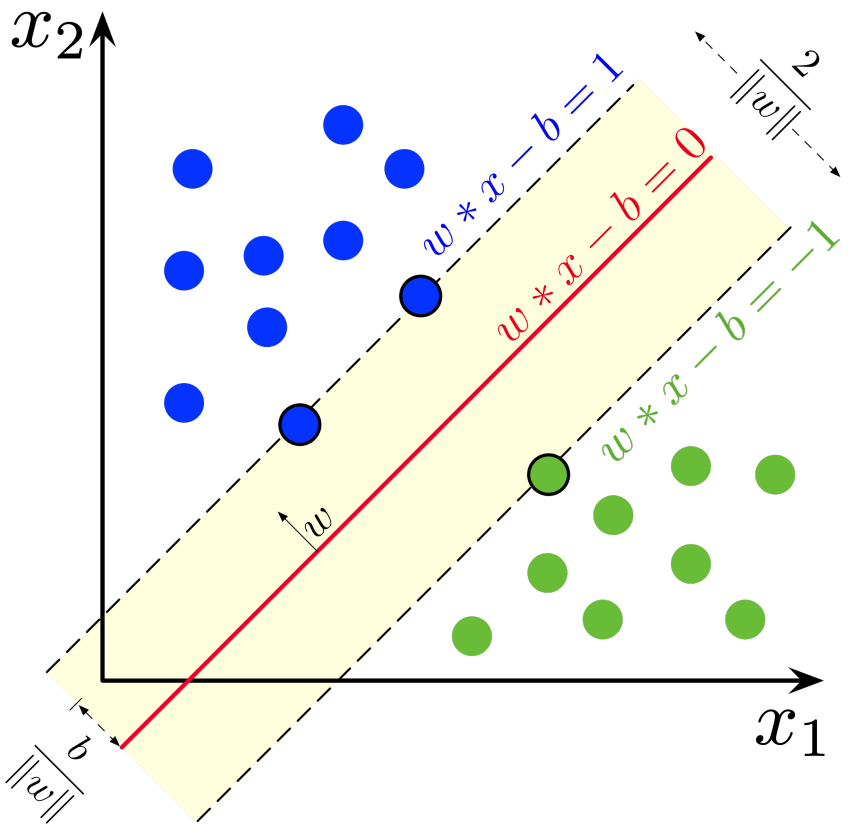
1. Identify predictors (X) and target variable(s) (y)
2. If there is no separate test set, divide X and y into two parts: call them (*train_X*, *train_y*) & (*test_X*, *test_y*)
    a. No data leakage
    b. Sufficient test set size
    c. No manipulation of test set during learning or figuring things out
3. Fine-tune hyper-parameters (most ml models have a set of these)
    a. Generally using cross-validation on the training set.

Support vector classifiers (SVC)

- Invented by Vladimir N. Vapnik and Alexey Ya. Chervonenkis in 1964.
- Simple idea:
  1. There are planes (or hyperplanes in case of high dimensional data) where the maximial separated boundaries between two or more classes can be identified,
  2. Data may not be linearly separable in the original space but some transformation of it may be in "some other space". (This is called Kernel trick and leads to kernelized SVMs)

Support vector classifiers (SVC): Hyperplanes

The algorithm identifies the hyperplane that maximizes the margin, which is the distance between the hyperplane and the nearest data points of each class. These nearest data points are called support vectors.



Source: Wikipedia.

Support vector classifiers (SVC): Kernels

When data is not linearly separable, SVC can transform it into a higher-dimensional space using a technique called the kernel trick. This transformation makes it possible to find a linear separating hyperplane in the new space.

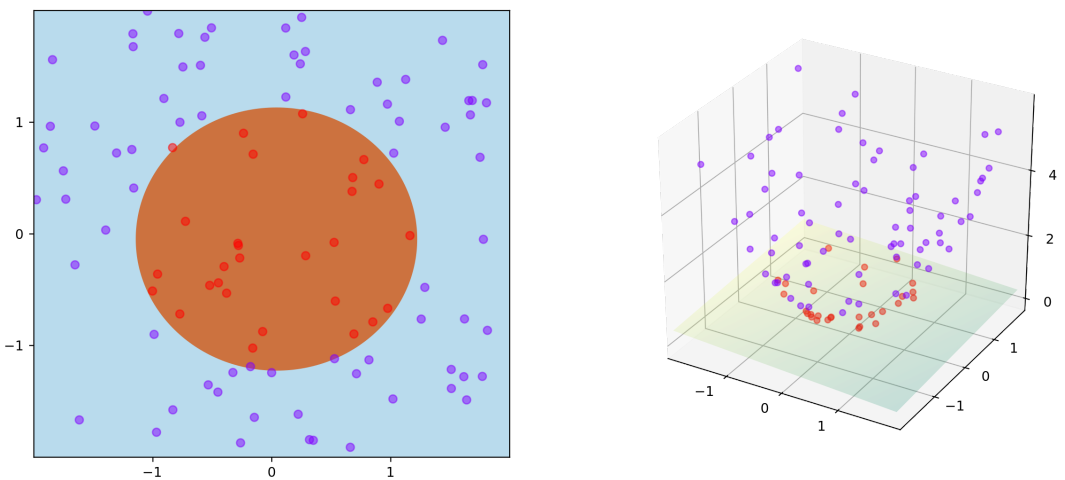


Figure: Source: Wikipedia.

Support vector classifiers (SVC): Hyperparameters - I

- **Kernel**: Specifies the kernel type to be used in the algorithm. Common options are.
    1. **linear**: No transformation, just a linear hyperplane.
    2. **poly**: Polynomial kernel, allows polynomial separation of data.
    3. **rbf** (Radial Basis Function): Also known as the Gaussian kernel, allows more complex separation.
    4. **sigmoid**: Uses a sigmoid function for separation.

Support vector classifiers (SVC): Hyperparameters - II

- **C (Regularization Parameter)**: Controls the trade-off between achieving a low error on the training data and minimizing the norm of the weights (*i.e.*, the margin maximization). A small C value encourages a wider margin, even if some training points are misclassified. A large C value aims for all training points to be classified correctly, potentially leading to a narrower margin
- **Gamma**: Applies to the rbf, poly, and sigmoid kernels, defining how far the influence of a single training example reaches. A small gamma value means far influence (smooth decision boundary), and a large gamma value means close influence (more complex decision boundary).
- **Degree**: Only relevant for the poly kernel. It specifies the degree of the polynomial used to find the decision boundary.

Random Forest

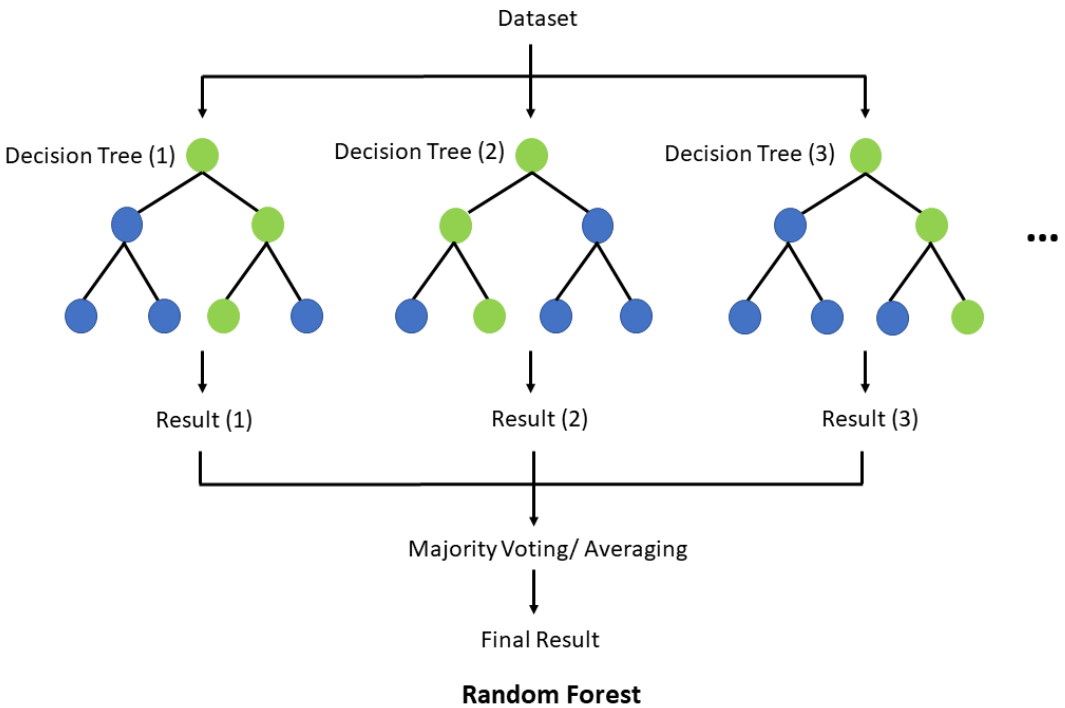


Figure: Source: Wikipedia

Random Forest: Hyperparameters - I

- **n_estimators**: The number of trees in the forest. More trees can lead to better performance but also increase computational cost.
- **max_depth**: The maximum depth of each tree. Limiting depth can prevent overfitting by controlling the complexity of the model.
- **min_samples_split**: The minimum number of samples required to split an internal node. Higher values can prevent overfitting.
- **min_samples_leaf**: The minimum number of samples required to be at a leaf node. Higher values can smooth the model, reducing variance.

Random Forest: Hyperparameters - II

- **max_features**: The number of features to consider when looking for the best split. Options include:
  - **auto** or **sqrt**: Uses the square root of the number of features.
  - **log2**: Uses the base-2 logarithm of the number of features.
  - A specific integer or float: Uses that number of features or percentage of features.
- **bootstrap**: Whether bootstrap samples are used when building trees. If **False**, the whole dataset is used to build each tree.
- **criterion**: The function to measure the quality of a split. Common options are:
  - **gini**: Gini impurity.
  - **entropy**: Information gain.

Multi-layered perceptrons (MLPs)

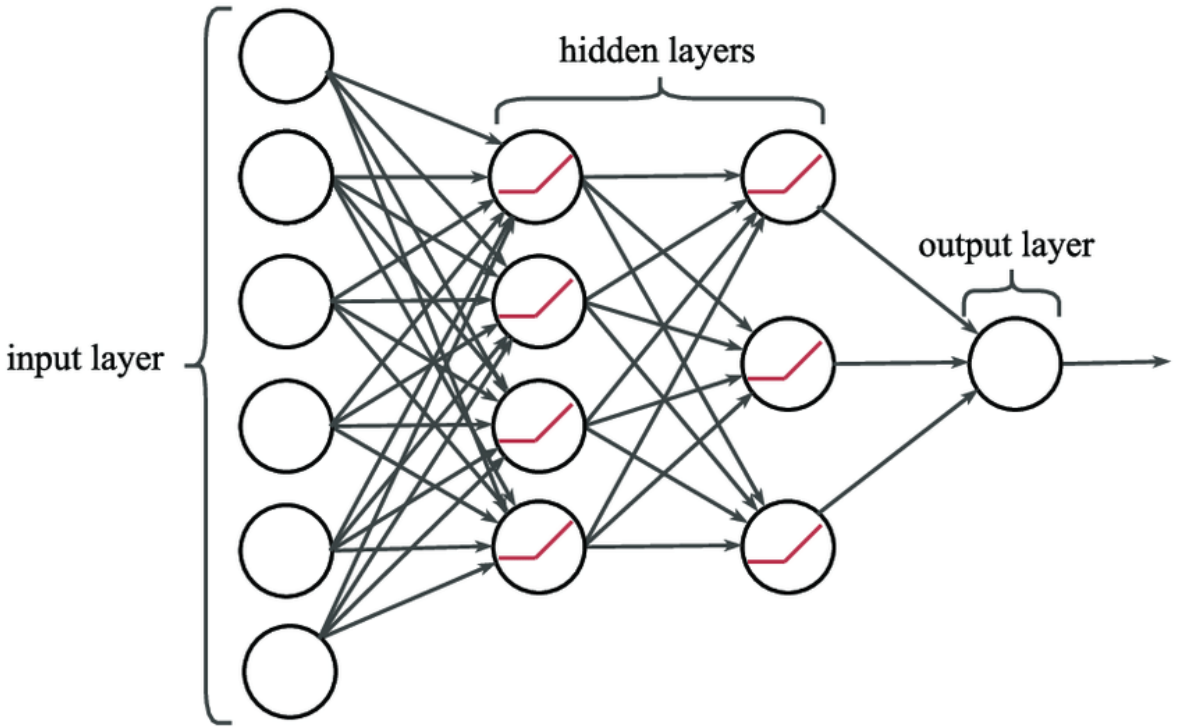


Figure: Source: Cenek et al., 2018

Summary-I

Support vector classifiers

- Effective in high-dimensional spaces
- Can be kernelized
- Highly sensitive to hyper-parameter choice
- Not a great choice for noisy data or mixed true kernels

Summary-II

Random Forest classifiers

- More robust with respect to data transformations
- Easily explainable
- Less prone to overfitting

Summary-III

MLPs

- Can approximate any function (in theory, thanks to Universal approximation theorem)
- Better generalization than support vector machines and random forests.
- Less prone to overfitting
- Inherently a black box (not explainable without post-hoc methods)
- Needs large amount of data
- Benefits from GPU.

Link to lab: https://learn.baiome.org/