Overview of Machine Learning in Python

Adrian Schmitt

Agenda

Introduction

Introduction

Types of Machine Learning

ML is broad term, but there are these common sub-disciplines:

- **Unsupervised Learning**
  - unlabelled data, usually no information given
  - *Goal*: Identify data structures (Clustering, Data mining)
- **Supervised Learning**
  - labelled data (ie. supervised)
  - *Goal*: Predict label (Regression & Classification)
- **Reinforcement Learning**
  - agent in an environment learns interactions
  - *Goal*: Learn most rewarding sequence of actions
- **Semi-supervised Learning**
  - training on dataset with labelled and unlabelled data
  - *Goal*: Predict label (Regression & Classification)

Introduction

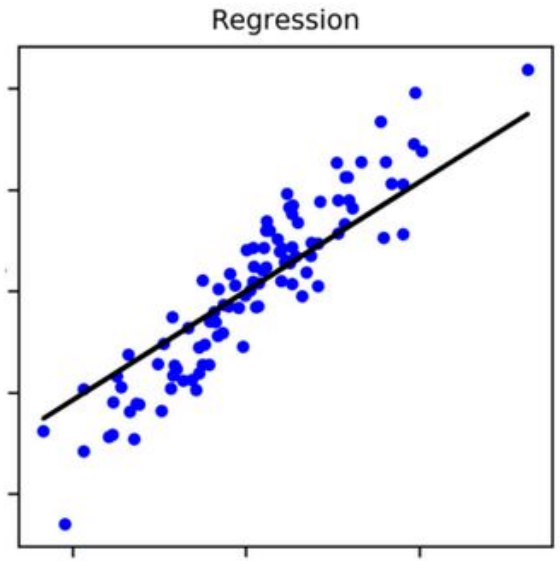
Supervised Learning

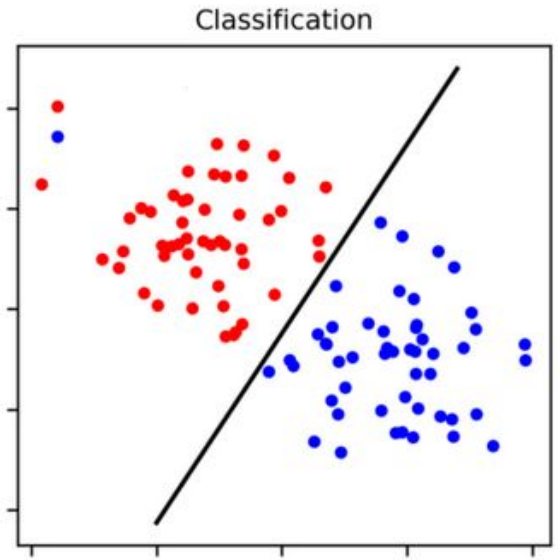
Regression:

➔ Predict numeric value (continuous variable)

Classification:

➔ Predict class value (fixed set)



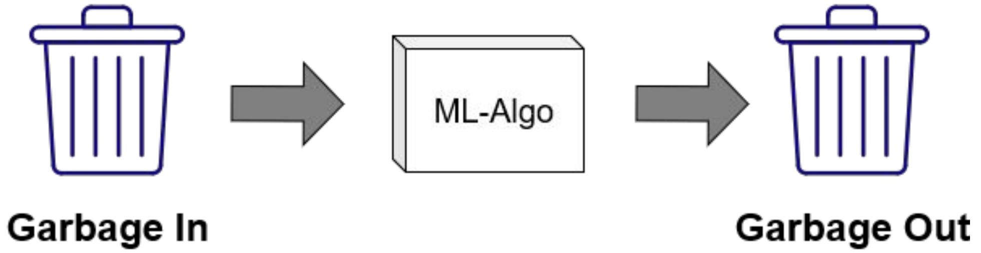


Data Preparation

Data preparation

Motivation

In supervised learning we need to analyse the data to create a feasible ML-Algorithm:



Usually the most demanding task in ML is to prepare and refine the data before applying any algorithm. However, there are many different aspects to consider.

Data preparation

Issues

Let’s focus on tabular data.

There are syntactic aspects:

- Numerical vs categorical values → Use encoding
- Different value ranges → Use scaling / normalisation / standardisation if distances matter
- Missing values → Define strategy

And also semantic aspects:

- Are the target classes equally represented (oversampling needed)?
- Is the data set biased?
- Does every attribute matter?
- Are there enough attributes collected?

Data preparation

Encoding

Non-numerical data must be transformed into numbers:

- **Label encoding**
  - Replace nominal value by ID
- **Ordinal encoding**
  - Same as label, but uses ranks (eg: tiny < medium translates to 1 < 2)
- **One-hot encoding**
  - Replace categorical values with binary representation as new columns
    (ie. n values are expressed with n-bits containing exactly one “1”; this is then used as n new columns replacing the original column)

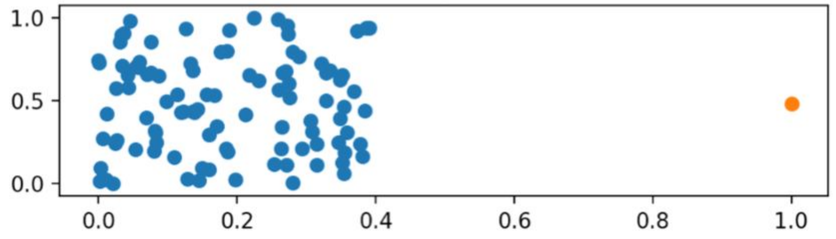
If the ML-Algo is sensitive to distances, label encoding tempers the model!

Data preparation

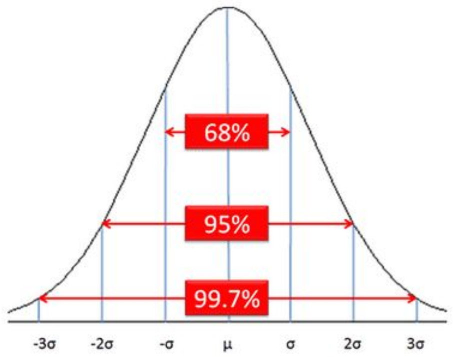
Scaling

Scaling is recommend for algorithm that use distances. Otherwise the specific value range impacts the model (especially outliers).

- **Min-Max scaling**
  - scale all variables to the same range
  - however outliers define spectrum



- **Standard Scaling (Z-Score Normalization)**
  - scale in standard deviation
  - outliers are handled
  - often used



Data preparation

Missing values

Some ML-Algorithm are affected by missing values. In general you should not manipulate your data, thus you should decided your strategy before analysing the data.

- **Deletion**
  - Rows or columns can be removed completely
    → only works if you have very few missing values
- **Imputation**
  - Substitute missing values
    → use “N/A” or “-1” as special value
    → randomly take the value from another entry
    → use mean / most frequent value of that attribute
    → …

Data preparation

Overview

Before deciding any preparation steps, it’s good to visualize the data and evaluate it.
The applied ML-Algorithm also affects the preprocessing.

Complex data types like images have further preprocessing step and they often use feature extraction to gain new attributes to learn.

Any preprocessing fitted on the whole dataset (eg. imputation, scaling) must be done after the split in training and testing data!
⟶ Otherwise the train- and test-data impact each other and this is a **Data leakage**

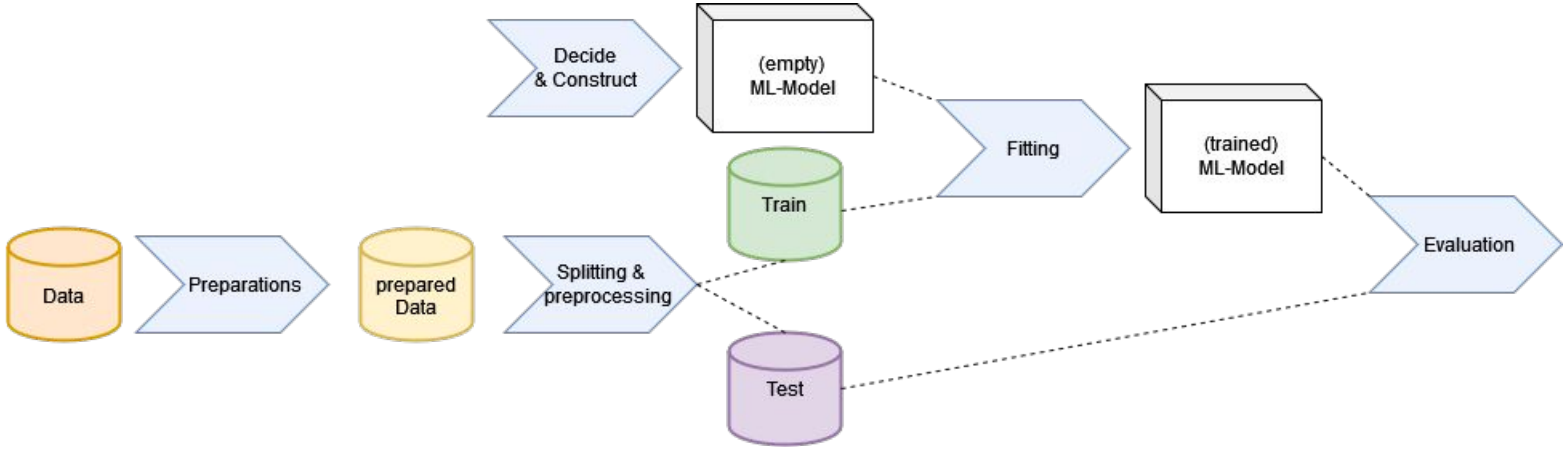
Sources of data for experimenting:

https://www.kaggle.com/
https://archive.ics.uci.edu/

Training & Evaluation

Training & Evaluation

Overview



Often the Splitting & Fitting is repeated to stabilize evaluation metrics.
Model-training depends on the concrete ML-Algorithm.

Evaluation

Splitting and training

We need labelled data to train the ML-Algo and then to test it, there are different methods:

- **Holdout**
  - split the data in two partitions of given size (eg. 75% training data and 25% test data)
    $\longrightarrow$ should be repeated to yield more consistent metrics
- **Cross-validation**
  - Create k folds of equal size and use exactly 1 fold for the test set, repeat for all possibilities
    $\longrightarrow$ ensures that every entry was part of training and of testing
- **Bootstrapping**
  - Random subset is used for test set

Evaluation

Splitting and training

There are more things to consider:

- **Use of validation set**
  - We want to estimate the generalisation error instead of test set error
  - Thus a third set is needed to be excluded from the training
- **Randomisation of the data**
  - Removes sorting
- **Stratification**
  - Ensures that each class is represented equally in training and test sets

Evaluation

Metrics

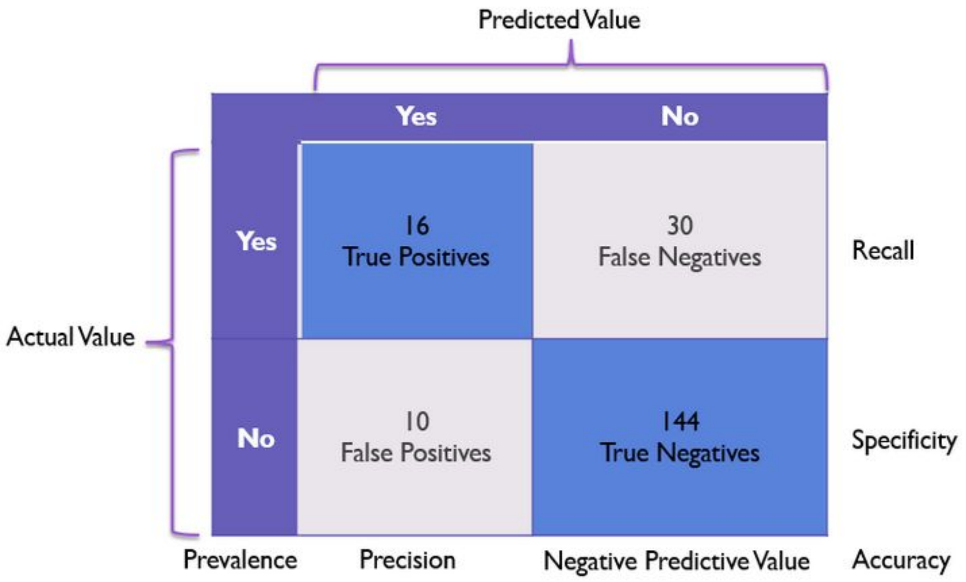
The metrics differ between classification and regression:

Classification:

- Accuracy
- Precision
- Recall
- F1-Score

Regression:

- Mean absolute error
- Mean squared error
- R2-Score



The metrics can also be computed per class (*macro-averaged*)

Evaluation

Interpretation

How much training should be done?

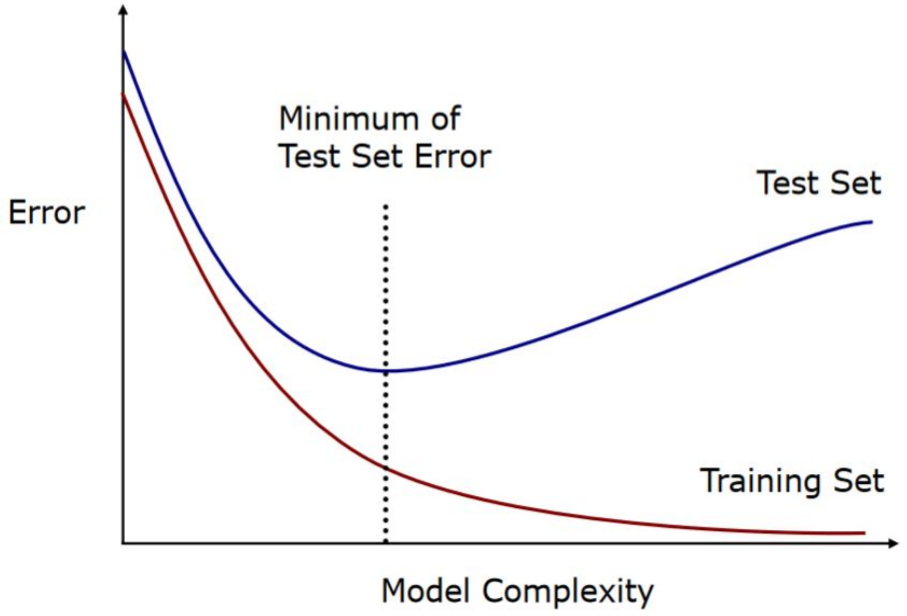
- too little → underfitting
- too much → overfitting

This is due to **Bias & Variance**:

- Bias is the tendency to learn assumptions
- Variance is model's sensitivity to the training data

→ Low bias & low variance needs complex models

→ Often we cannot calculate the bias and variance, since the "real" prediction function is unknown



Evaluation

Hyperparameter optimization

Which parameters of your algorithm yield best results?

- **Grid search**
  - exhaustive search to try out parameters in specified intervals
    $\longrightarrow$ computationally expensive
- **Randomized search**
  - random configuration
    $\longrightarrow$ faster than grid search
- **Bayesian optimization**
  - uses a probability distribution to model the target function
  - iteratively selects evaluation points to improve the distribution
    $\longrightarrow$ efficient and often used for optimizations of expensive algorithms

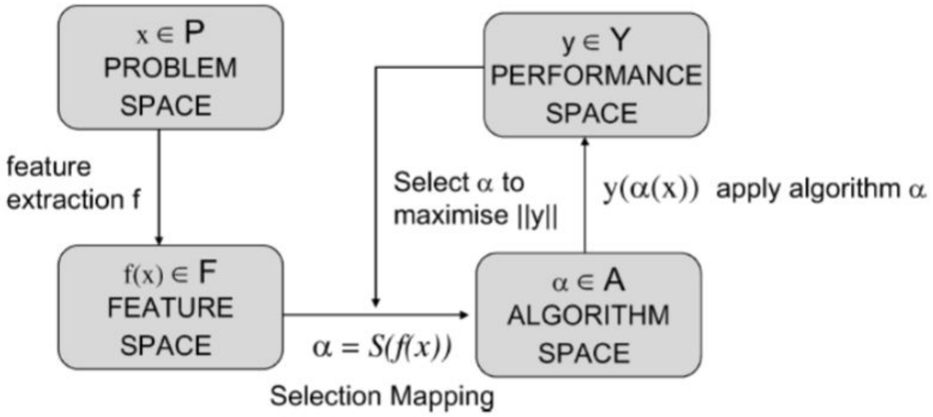
Evaluation

Algorithm selection

Which algorithm should you choose?

→ **No Free Lunch theorem**: *there is no universal best choice for all problems*

So we need to find out
→ **Meta-Learning:**



But for many applications good algorithms are known (eg. Neural Networks for visual detection)

Decision Tree

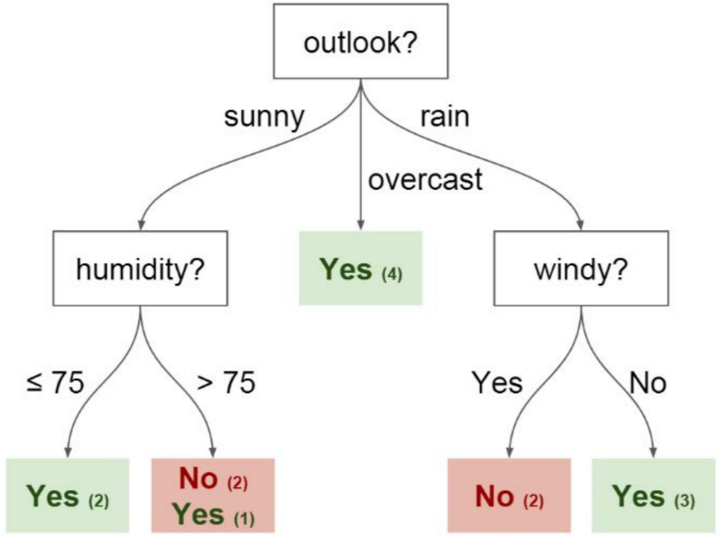
Decision Tree

Rule-based model

DTs are graphs that split the dataset by attribute value:

The motivation is to put most important attributes closer to the root. The selection of the current decision-feature and the best spit values can be computed:

- **Error rate**
  - Try out every split and pick the one with least errors
    ⟶ often uses heuristic
- **Information gain**
  - based on Shannon's theory of entropy
  - the best split reduces entropy the most
    ⟶ biased towards features with many values



The leaves contain the predicted value, depending on the training they are not homogenous data-sets.

Decision Tree

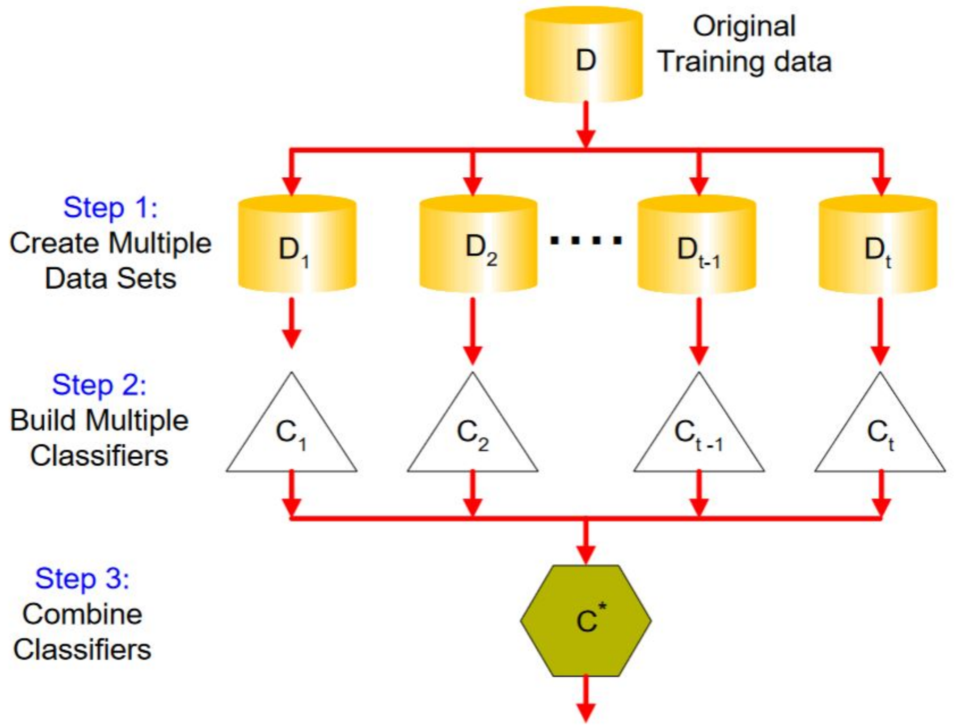
Random Forest

RFs combine many DTs into a forest:

Usually each tree receives a random subset of entries and features to be trained on.

The prediction is a majority voting of the forest.

RFs outperform single DTs significantly, but there is no general rule on how large the RF should be.



Decision Tree

Pitfalls

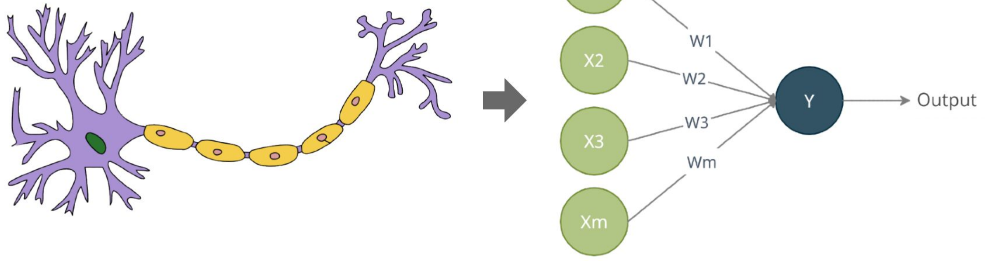
Overfitting	⟶ Use pruning / random forest
High variance	⟶ Use random forest
Low expressiveness	⟶ Use random forest

Neural Network

Neural Network

Perceptron

NNs are inspired by brain activities, to learn complex target functions.



A neuron is being represented by a perceptron, both have:

- Several inputs, each with its own weight
- A core to sum up all scaled inputs
- A specific activation in that core, producing an output

In NNs, the inputs and activation function has to be defined, while the weights are trained.

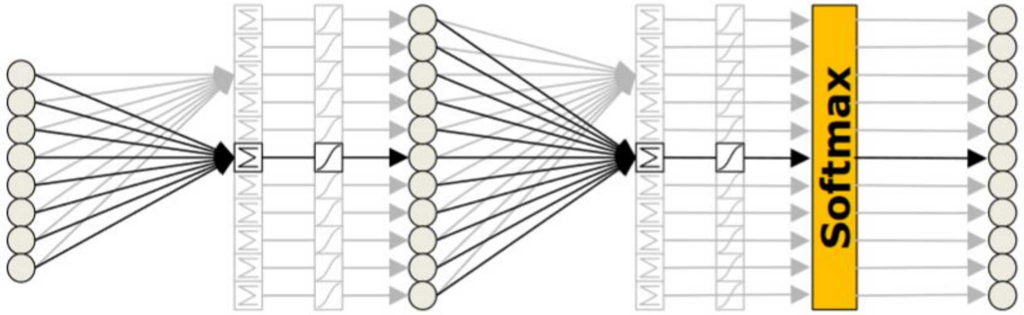
A single perceptron can only perform linear separation for classification.

Neural Network

Multi-Layer Perceptron

MLPs combine perceptrons into a network:

They are fully connected between layers, feed-forward, have at least one hidden layer and no cycles.



The training-algorithm is a back propagation with gradient descent.

But how many nodes per layer and how many layers are needed?
⟶ depends the input
⟶ **Universal approximation theorem**:

A MLP can approximate any continuous function with a single hidden layer and a finite set of nodes

Neural Network

Deep NN

“deep” = MLP with more than one hidden layer (also called **feedforward NN “FNN”**)

This NN are often specialised and complex, some examples:

- **Convolutional NN (CNN)**
  - Has special layers for image operations (convolution / kernel, subsampling)
    ⟶ utilizes common image extractions directly into the NN-model
- **Recurrent NN (RNN)**
  - Has cycles and is used for learning of sequential data (eg. language)
  - Simulates memory
  - Training is unrolled version of back-propagation
    ⟶ longer sequences may scale gradients in extremes
- **Long Short-Term Memory NN (LSTM-NN)**
  - is RNN with memory cells and gates
    ⟶ usually outperforms RNN

Neural Network

Deep NN in python

The Keras-framework can be used to customize deep NNs (see https://www.tensorflow.org/guide/keras)

Example:

```
model = keras.Sequential()
model.add(keras.Input(shape=(250, 250, 3)))                     # 250x250 RGB images
model.add(layers.Conv2D(32, 5, strides=2, activation="relu"))   # convolution
model.add(layers.Conv2D(32, 3, activation="relu"))
model.add(layers.MaxPooling2D(3))                               # subsampling
model.add(layers.Conv2D(32, 3, activation="relu"))
model.add(layers.Conv2D(32, 3, activation="relu"))
model.add(layers.MaxPooling2D(3))
model.add(layers.Conv2D(32, 3, activation="relu"))
model.add(layers.Conv2D(32, 3, activation="relu"))
model.add(layers.MaxPooling2D(2))

# Now that we have 4x4 feature maps, time to apply global max pooling:
model.add(layers.GlobalMaxPooling2D())

# Finally, we add a classification layer:
model.add(layers.Dense(10))
```

Neural Network

Pitfalls

• *Overfitting*	$\longrightarrow$ dropout, regularization, early stopping,
• *Computational cost*	$\longrightarrow$ federated learning, pruning
• *Hyperparameter tuning*	$\longrightarrow$ experiments, hyperparameter optimization
• *Vanishing or explosion of gradients*	$\longrightarrow$ gradient clipping, regularization
• *Black box*	$\longrightarrow$ Feature importance analysis, LIME, dependency plots, …

Advanced Topics

Advanced Topics

- Boosting (= combination of classifiers)
- Efficiency
  - federated ML
  - transfer learning
- Explainability
  - models
  - feature importance
- Feature extraction / selection
  - SIFT
  - Tokenization
  - PCA
- Security (privacy, robustness)
  - K-anonymity
  - Adversarial Training
- Ethics

Summary

Summary

- Machine Learning very often refers to classification / regression, ie. automated prediction
- In most applications of ML the data collection, labelling and preprocessing is the most important and most human-time consuming task
- The choice of the model and hyperparameters depends on the data and should be explored
- Evaluation data must not be influenced by the training data
- Neural networks are directed graphs, the values flowing through network are updated based on gradient descent
- Deep NNs are often used as a black box, but there is current research on increasing the transparency

References

References

Papers:

Meta-Learning:
https://www.researchgate.net/publication/220565856_Cross-Disciplinary_Perspectives_on_Meta-Learning_for_Algorithm_Selection/link/57e1f8d208ae1f0b4d93fa7d/download?_tp=eyJjb250ZXh0Ijp7ImZpcnN0UGFnZSI6InB1YmxpY2F0aW9uIiwicGFnZSI6InB1YmxpY2F0aW9uIn19

Language Models:
https://arxiv.org/pdf/1906.03591.pdf

Overviews:

https://www.engineersgarage.com/machine-learning-algorithms-classification/

https://towardsdatascience.com/supervised-learning-algorithms-cheat-sheet-40009e7f29f5

https://www.kaggle.com/code/faressayah/decision-trees-random-forest-for-beginners

References

Scikit:
https://scikit-learn.org/stable/user_guide.html

Keras:
https://www.tensorflow.org/guide/keras

Data-Sources for experiments:
https://www.kaggle.com/
https://archive.ics.uci.edu/

Github-Examples:

https://github.com/Gnosling/ML_Python_Scikit_Example

https://github.com/pbloem/language-models

https://github.com/microsoft/LMChallenge

https://github.com/lualeperez/coursera-introduction-to-deep-learning-with-keras