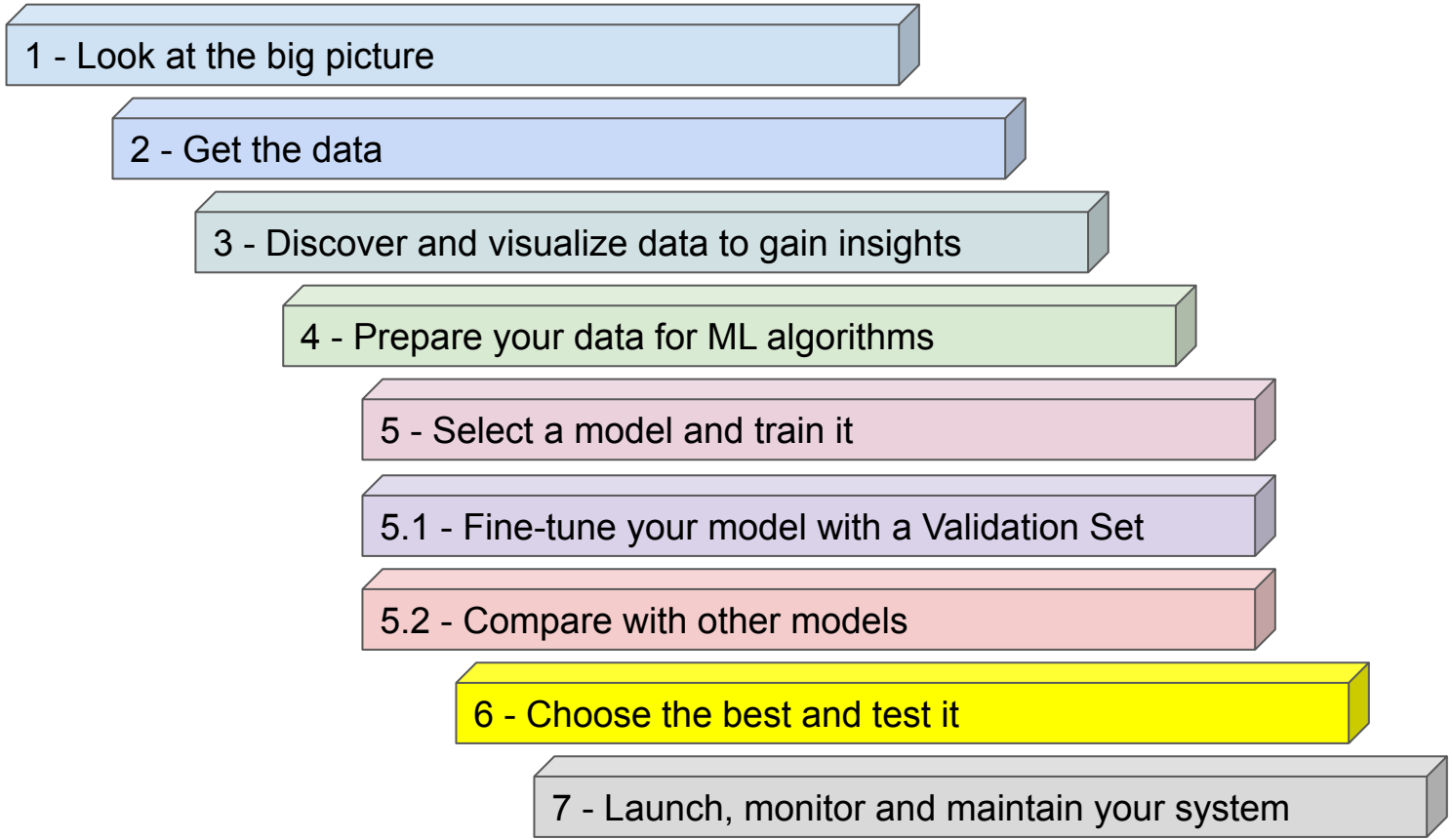
UNIVERSITÀ
DI PARMA
DIPARTIMENTO DI INGEGNERIA E ARCHITETTURA

Design principles for ML

Gianfranco Lombardo, Ph.D
gianfranco.lombardo@unipr.it

Project Management in Machine Learning



Gianfranco Lombardo, Ph.D (gianfranco.lombardo@unipr.it)

1 Look at the big picture

1. Define the objective in business terms

2. How will your solution be used?

3. How should you frame this problem? Supervised/Unsupervised learning?
   a. Is it a regression or a classification task?

4. How should performance be measured? and is the performance measure aligned with the business objective? What would be the minimum performance need to reach the business objective?

2 Get the data

1. List the data you need and how much you need
   a. If classification task: try to collect enough example for each class you need to teach to the classifier

2. Convert the data to a format you can easily manipulate without changing the data itself

3. Ensure sensitive information is deleted or protected (e.g., anonymized)

4. Check the size and type of data (Time series, sample, geographical etc..)

5. Sample an evaluation set (test set), put it aside and NEVER look at it until the end when you have to evaluate the generalization error of your model

3 Explore the data

1. Study each attribute of your dataset and its characteristics:
   a. Type (categorical, numerical)
   b. Bounded/unbounded
   c. Text or structured
   d. % missing values
   e. Noisiness and type of noise (random, outliers, rounding errors etc..)
   f. Evaluate if the attribute can be useful for the task
   g. Data distribution (Gaussian, uniform, logarithmic)

2. For supervised tasks, identify the target attribute (y)

3. Visualize the data

4. Study the correlations between attributes

4 Prepare your data for the ML algorithms

1. **Data cleaning**
   a. Fix or remove the outliers (optional): but outliers can have an impact on learning and on the performance measure
   b. Fill in missing values (e.g., with zero, mean of the attribute, median..) or drop their rows (or columns)

2. **Feature selection** (optional)
   a. Drop the attributes that provide no useful information for the task

3. **Feature engineering**
   a. Discretize continuous features
   b. Add promising transformations of features (e.g., log(x), x^2, sqrt(x)
   c. Aggregate features into promising new ones

4. **Feature scaling**: normalize or standardize features

Feature scaling

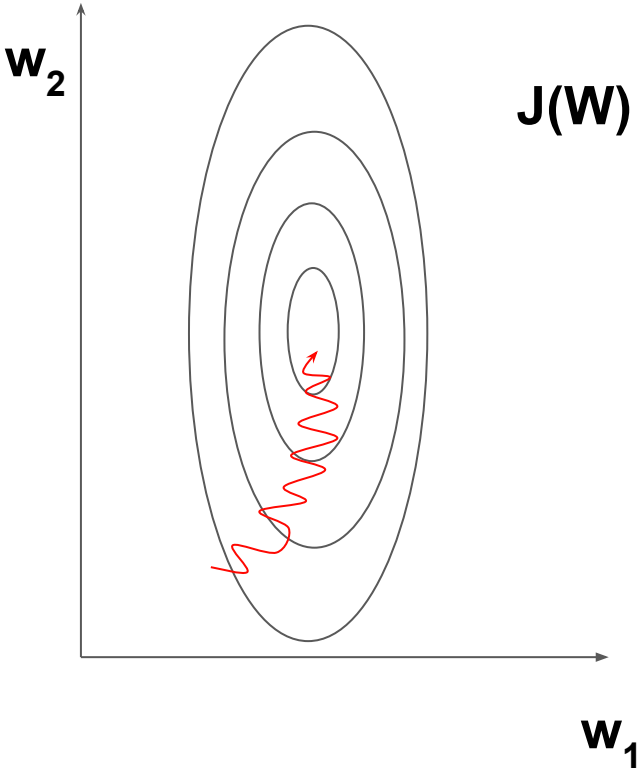
- One of the most important pre-processing is represented by feature scaling in order to help gradient descent to converge quickly towards the global minimum

- Main idea: Make sure features are on a similar scale (range of values)
- Let's see an example:
  - Price of the house dataset
  - One feature x_0 (m^2): range between 50 to 600
  - Another feature is n° rooms: range between 1 to 10

- When we want to perform a simple linear regression (or classification) and look at the parameter space we can see that these different ranges introduces a distortion of the space over (several) axes

- Make the hypothesis we want to minimize a cost function $J(w_0,w_1,w_2)$

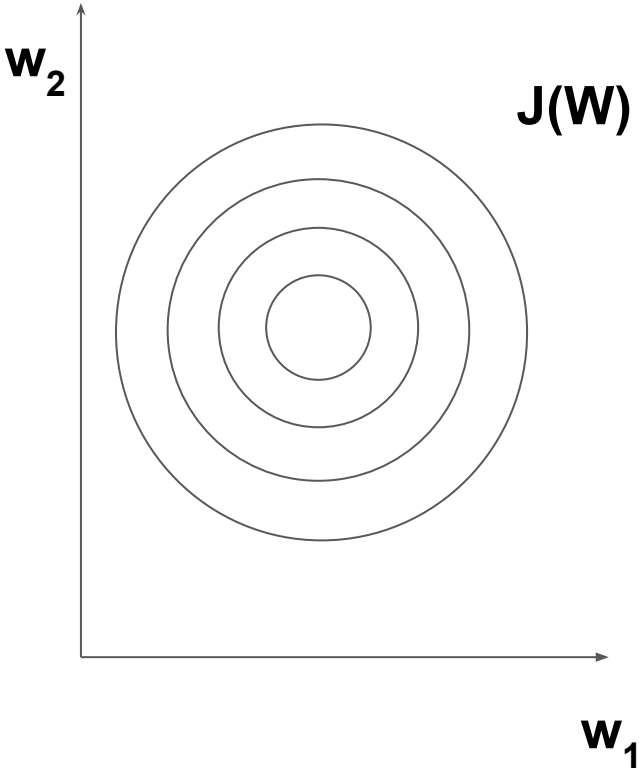
Feature scaling: main idea



- Distortion of the space along the axes
- Moreover, different algorithms exploit the euclidean distance among the training examples (e.g., K-nearest neighbors).

- If one of the features has a broad range of values, the distance will be governed by this particular feature

Feature scaling: main idea



- If we scale feature in order to have the same range, our J(W) will be less skinny and more regular
- Gradient descent will require less time to converge towards the minimum

Feature scaling types

Min-max normalization

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

The feature will range between
0 and 1

Mean normalization

$$x' = \frac{x - \text{average}(x)}{\max(x) - \min(x)}$$

Standardization (or Z-Score)

$$x' = \frac{x - \bar{x}}{\sigma} \qquad \text{with} \qquad \bar{x} = \text{average}(x)$$

The feature will have zero-mean
and unit variance

Feature scaling

- Pay attention: To compute min, max, mean or std to perform feature scaling you have always refer to the training-set! Never the test set, because you should always think that the test set could be something unknown during the training!

- Store the values you need from the training and then apply the pre-processing over the test-set before using it in your predictor

Feature scaling

Mq	N°rooms	Price
50	2	80,000 $
75	3	100,000 $
125	4	130,000 $
35	1	50,000 $
200	5	220,000 $
60	2	90,000 $
100	3	110,000 $

Training-set

avg(Mq) = 97 std(Mq) = 59,88
avg(Rooms) = 3 std(Rooms) = 1,41

Example of standardization

Mq	N°rooms	Price
-0,78	-0,70	80,000 $
-0,36	0	100,000 $
0,46	0,70	130,000 $
-1,04	-1,41	50,000 $
1,72	1,141	220,000 $

Training-set

avg(Mq) = 97 std(Mq) = 59,88
avg(Rooms) = 3 std(Rooms) = 1,41

zero mean zero mean

I will use later these values (computed on the training-set) to pre-process also the test set !

Feature scaling with Python

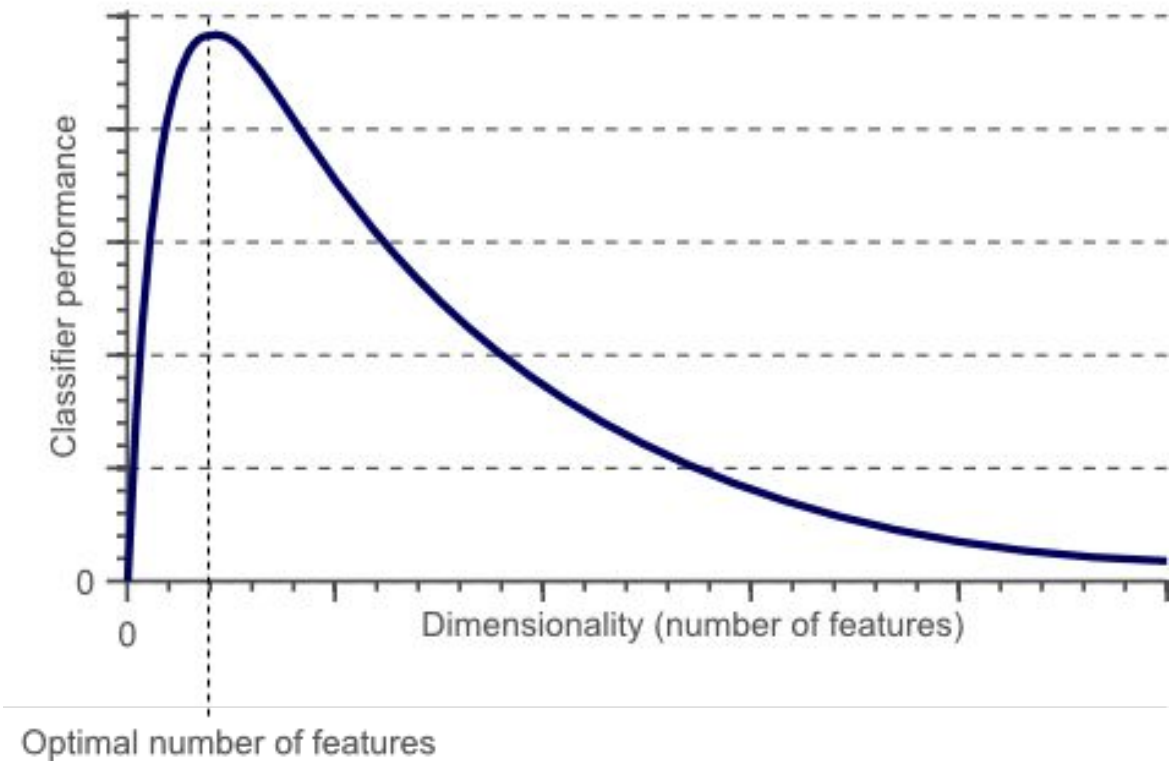
```
avg = np.mean(X_train, axis=0)
std = np.std(X_train,axis=0)

X_train = (X_train-avg)/std
X_test= (X_test-avg)/std
```

Alternatives present directly in Sklearn
https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing

Feature Selection

- Big data are not always the answer
- Some features can lead to worst results
- Some features are just unuseful
- Reduces Overfitting: Less redundant data means less opportunity to make decisions based on noise.
- Improves results: Less misleading data·
- Reduces Training Time: fewer data points reduce algorithm complexity and algorithms train faster.

Feature Selection



This phenomenon is due to the Curse of Dimensionality

Example

Number of Instances
150 (50 in each of three classes)

Number of Attributes
4 numeric, predictive features and the class

Features Information

- sepal length in cm
- sepal width in cm
- petal length in cm
- petal width in cm
- **class:**
  - Iris-Setosa
  - Iris-Versicolour
  - Iris-Virginica

●Using matplotlib and the iris dataset.csv:

○ Read the dataset with pandas

○Plot iris data using sepal length (x axis) and sepal width (y axis), representing different classes with different colors.

○Plot iris data using petal length (x axis) and petal width (y axis), representing different classes with different colors.

●Compare the two representations

Univariate feature selection with chi^2 test

- Select the best features based on univariate statistical tests, for example the chi² test
    - This score can be used to select the n_features with the highest values for the test chi-squared statistic from X, **which must contain only non-negative features** such as booleans or frequencies, relative to the classes.
- The chi-square test measures dependence between stochastic variables
- This function avoid features that are the most likely to be independent of class and therefore irrelevant for classification.

Univariate feature selection with chi^2 test

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
df = pd.read_csv("iris_dataset.csv")
y = df.pop('class')
X = df


#apply SelectKBest class to extract the k best features
bestfeatures = SelectKBest(score_func=chi2, k=2)
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe columns
print(featureScores)
```

Feature selection with Mutual information

- Estimate mutual information for a target variable.
- Mutual information (MI) between two random variables is a non-negative value, which measures the dependency between the variables. It is equal to zero if and only if two random variables are independent, and higher values mean higher dependency
- The methods based on F-test estimate the degree of linear dependency between two random variables (like chi^2). On the other hand, mutual information methods can capture any kind of statistical dependency, but being nonparametric, they require more samples for accurate estimation.

Feature selection with Mutual information

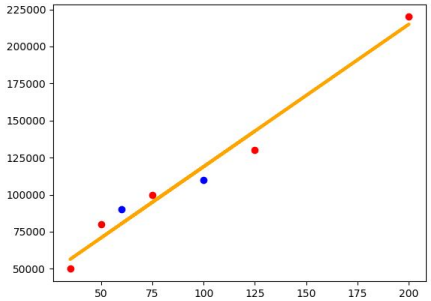
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_classif #mutual_info_regression
df = pd.read_csv("iris_dataset.csv")
y = df.pop('class')
X = df


#apply SelectKBest class to extract the k best features
bestfeatures = SelectKBest(score_func=mutual_info_classif, k=2)
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe columns
print(featureScores)
```

5 Assumptions on data

- From (A. Geron book)

- A model is a simplified version of the observations, where “simplification” means that the model discards the superfluous details that are unlikely to generalize to new instances. However, to decide what data to discard and what data to keep, you must make ***assumptions***

- For example, a linear model makes the assumption that the data is fundamentally linear and that the distance between instances and the straight line is just noise, which can be be, thus, ignored.



5 The No free Lunch Theorem (NFL, D.Wolpert 1996)

- If you make absolutely no assumption about the data, then there is no reason to prefer one model over any other

IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 1, NO. 1, APRIL 1997

No Free Lunch Theorems for Optimization

David H. Wolpert and William G. Macready

***Abstract*—A framework is developed to explore the connection between effective optimization algorithms and the problems they are solving. A number of "no free lunch" (NFL) theorems are presented which establish that for any algorithm, any elevated performance over one class of problems is offset by performance over another class. These theorems result in a geometric interpretation of what it means for an algorithm to be well suited to an optimization problem. Applications of the NFL theorems to information-theoretic aspects of optimization and benchmark measures of performance are also presented. Other issues addressed include time-varying optimization problems and *a priori* "head-to-head" minimax distinctions between optimization algorithms, distinctions that result despite the NFL theorems' enforcing of a type of uniformity over all algorithms.**

***Index Terms*— Evolutionary algorithms, information theory, optimization.**

I. Introduction

THE past few decades have seen an increased interest in general-purpose "black-box" optimization algorithms that exploit limited knowledge concerning the optimization

information theory and Bayesian analysis contribute to an understanding of these issues? How *a priori* generalizable are the performance results of a certain algorithm on a certain class of problems to its performance on other classes of problems? How should we even measure such generalization? How should we assess the performance of algorithms on problems so that we may programmatically compare those algorithms?

Broadly speaking, we take two approaches to these questions. First, we investigate what *a priori* restrictions there are on the performance of one or more algorithms as one runs over the set of all optimization problems. Our second approach is to instead focus on a particular problem and consider the effects of running over all algorithms. In the current paper we present results from both types of analyses but concentrate largely on the first approach. The reader is referred to the companion paper [5] for more types of analysis involving the second approach.

We begin in Section II by introducing the necessary notation. Also discussed in this section is the model of computation

5 The No free Lunch Theorem (NFL, D.Wolpert 1996)

In light of NFL, for some datasets the best model is a linear model, for others Gradient boosting (or others) and for more other datasets is a Neural Network:

- There is no model that is a priori guaranteed to work better
- The only way to know for sure which is the best model for your data and task is to evaluate them all
- An alternative is to make some reasonable assumptions to evaluate less models
  - For simple tasks you may evaluate linear models with various levels of regularization or ensemble learning
  - For complex tasks you may evaluate a Neural Network
- Another thing to be taken into account is the number of parameters you should learn with respect of your dataset dimension (in terms of features and in terms of n° examples)

5 Model comparison

- A general idea can be to compare all the models using the same training and validation set with the default parameters and then choose the best one
    - First problem: Results are not always the same because of:
        - Stochasticity of learning models (usually parameters initialization)
        - Random division between training and validation
        - Solution: Evaluate each model for several different runs and then consider the average result
            - At least 10 runs is suggested
    - Once you found the best learning algorithm you should **fine-tune** the model hyper-parameters (n° of tree, regularization methods etc..)

How to fine-tune hyper-parameters ?

Grid-search

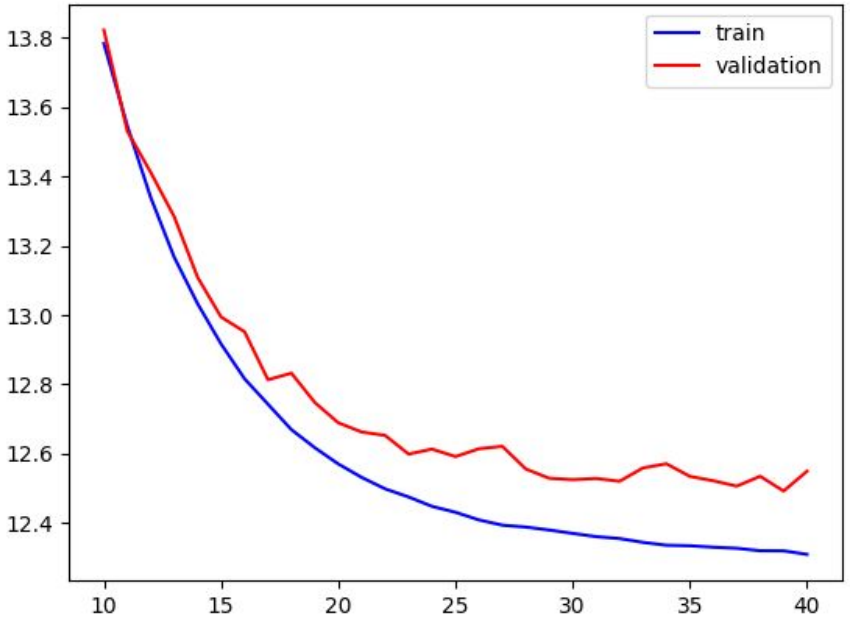
- We try every combination of a preset list of values of the hyper-parameters and evaluate the model for each combination.
- Each set of parameters is taken into consideration and the result is noted.
- Once all the combinations are evaluated, the model with the set of parameters which give the best result on the VALIDATION SET is considered to be the best.
- The number of evaluations required for this strategy increases exponentially with each additional parameter

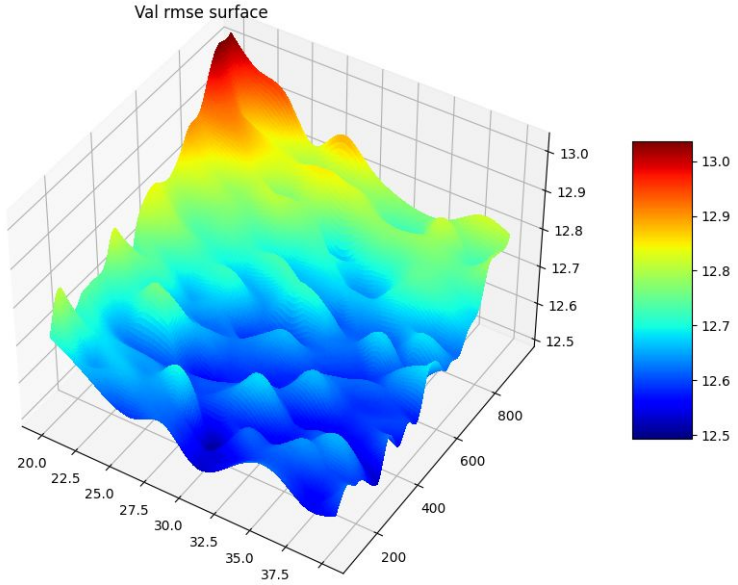
Random search

- Random combinations of the hyperparameters are used to find the best solution
- It works well for lower dimensional data since the time taken to find the right set is less with less number of iterations
- In Random Search for Hyper-Parameter Optimization by Bergstra and Bengio, the authors show empirically and theoretically that random search is more efficient for parameter optimization than grid search
- When dimension is high it is better to combine with a Grid-search

5 Analyze the best model and its errors during fine-tuning

- To better analyze the best configuration of the hyper-parameters it is suggested to evaluate for each combination what is the error on the training-set and on the validation-set to identify if overfitting or underfitting occurs and when because otherwise the model will not work on test-set at the end!

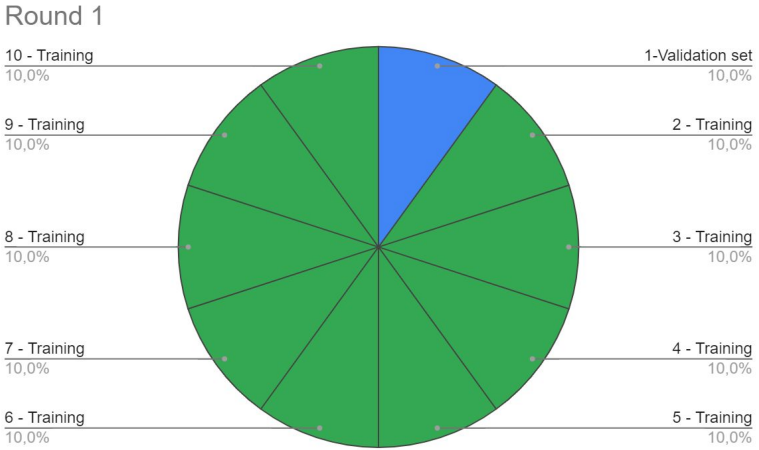


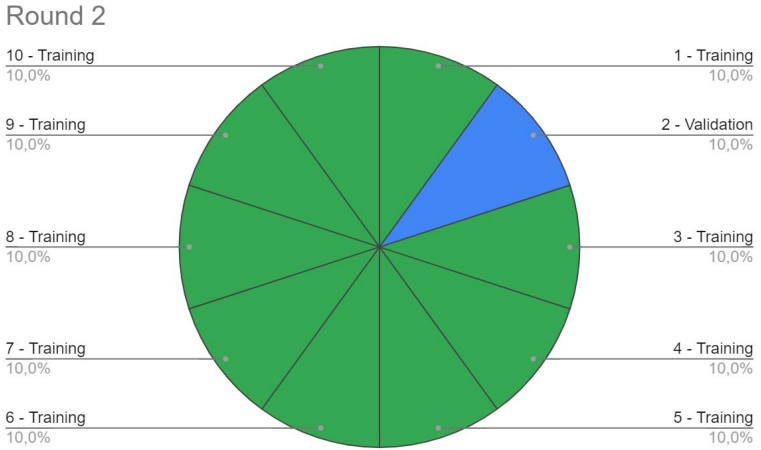


5 K-Fold Cross-Validation

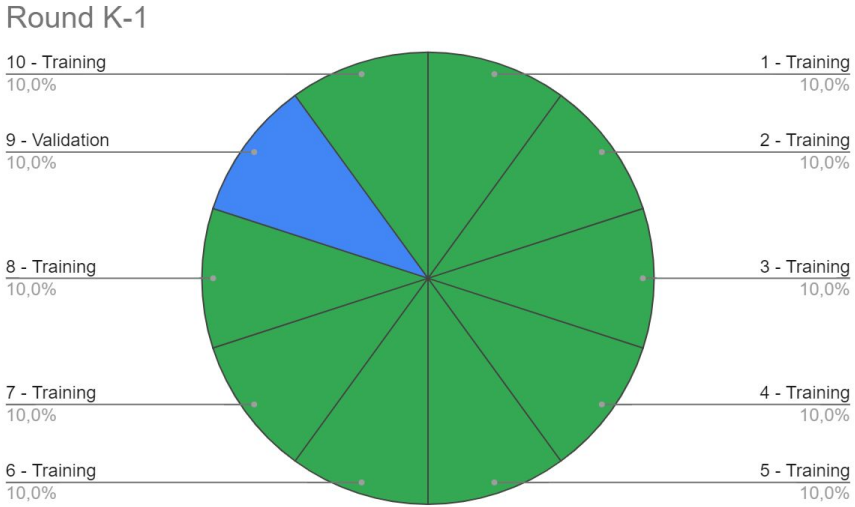
- When the dataset is not huge the validation set can be small
  - This means that you risk to evaluate your model on lucky or unlucky cases

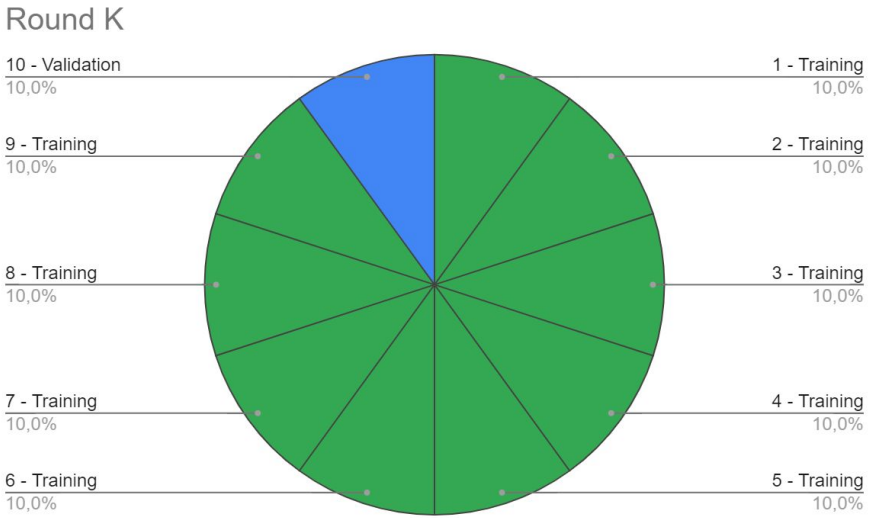
- You can always random split dataset in training and validation at each run but the dimension of your validation can always be a source of bias
  - One suggested possibility is to cross-validate your model





5 K-Fold Cross-Validation





Gianfranco Lombardo, Ph.D (gianfranco.lombardo@unipr.it)

5 K-Fold Cross-Validation

- K-Fold because you divide your dataset in K folds
- Perform K runs and for each run the algorithm select K-1 fold for training and one as Validation-set
- Once finished you get the average performance on the entire dataset so you have less variance rather than using a single small validation set
- Moreover the cross-validation allows you to get not only an estimate of the performance of your model, but also a measure of how precise the estimate is (e.g., the standard deviation)
- Pay attention: If you want to compute the accuracy you should use a different version of k-fold cross-validation to get balanced folds (see for example stratified k-fold if your dataset is already balanced)

5 K-Fold Cross-Validation

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import KFold

df = pd.read_csv("iris_dataset.csv")
y = df.pop('class')
X = df

kf =KFold(n_splits=10, random_state=None, shuffle=False) #initialize KFold

for train_index, validation_index in kf.split(X):
        print("TRAIN:", train_index, "VALIDATION:", validation_index)
        X_train = X.iloc[train_index]
        X_validation = X.iloc[validation_index]
        y_train= y[train_index]
        y_validation = y[validation_index]
```

5 Let's summarize the pipeline

1. Divide data in 80 % training and 20% test

2. Apply feature scaling

3. **From the training extract** the 10% as **a validation set** and use only this set to compare, select and fine-tune your model (In alternative use the Cross-validation)

4. Make a grid or random search over the possible parameters and evaluate your models with the validation

5. When you got good results, evaluate your model over the test set to have good guarantees about your model capacity of generalization!
   a. <u>If you use the test-set before that point you will lose the possibility of evaluate if your model is really generalizing or not</u>