What is Statistical Machine-Learning?

STATISTICS
Data analysis

ARTIFICIAL INTELLIGENCE

OPTIMIZATION

Machine Learning

Real-world examples of SML applications

- Handwritten characters recognition
  \[
  \begin{array}{ccc}
  3 & \rightarrow & 3 \\
  3 & \rightarrow & 6 \\
  \vdots & \rightarrow & \vdots \\
  \end{array}
  \rightarrow \text{Handwritten digits recognition system}
  \]

- Object category visual recognition
  \[
  \text{Pedestrians} \quad \text{« non-pedestrians »} \\
  \rightarrow \text{Pedestrian recognition system}
  \]

- Speech recognition
  \[
  \text{Feature Extraction} \quad \text{Neural Network}
  \]
  - Multi-factorial forecasting
  - Natural Language understanding
  - Playing GO!
  - \ldots

Statistical Machine-Learning

- One of many sub-fields of Artificial Intelligence
- Application of optimization methods to statistical modelling
- Data-driven mathematical modelling, for automated classification, regression, partitioning/clustering, or decision.behavior rule

Clustering
Types of Machine-learning

- Availability of target output data?
  → **Supervised** learning vs. **Unsupervised** learning
  or **Reinforcement** Learning

- Permanent adaptability?
  → **offline** learning vs. **online (life-long)** learning

- What kind of (mathematical) model?
  → polynom/spline, decision tree, neural net, kernel machine, ...

- Which objective function?
  → cost function (quadratic error, ...), implicit criterium, ...

- **How to find the best-fitting model?**
  → algorithm type (exact solving, gradient descent, quadratic optimization, heuristics, ...)

---

Most simple ML example: Least Squares Linear Regression

- Model: (straight) line $y=ax+b$ (2 parameters $a$ and $b$)
- Data: $n$ points with target value $(x_i,y_i) \in \mathbb{R}^2$
- Cost function: sum of squares of deviation from line
  $\kappa=\sum (y_i-a.x_i-b)^2$
- Algorithm: direct (or iterative) solving of linear system

$$
\begin{bmatrix}
\sum_{i=1}^{n}x_i^2 & \sum_{i=1}^{n}x_i \\
\sum_{i=1}^{n}x_i & n
\end{bmatrix}
\begin{bmatrix}
a \\
b
\end{bmatrix}
=
\begin{bmatrix}
\sum_{i=1}^{n}x_i y_i \\
\sum_{i=1}^{n}y_i
\end{bmatrix}
$$

[Question: Where does this equation come from?]
Supervised vs Unsupervised learning

Learning is called "supervised" when there are "target" values for every example in training dataset:

\[ \text{examples} = (\text{input-output}) = (x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \]

The goal is to build a (generally non-linear) approximate model for interpolation, in order to be able to generalize to input values other than those in training set.

"Unsupervised" = when there are NO target values:

\[ \text{dataset} = \{ x_1, x_2, \ldots, x_n \} \]

The goal is typically either to do datamining (unveil structure in the distribution of examples in input space), or to find an output maximizing a given evaluation function.
Reinforcement Learning (RL)

Goal: find a “policy” \( a_t = \pi(s_t) \) that maximizes

\[
R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \quad \gamma \in [0, 1]
\]

Typical use of RL: learn a BEHAVIOR

SUPERVISED LEARNING: regression or classification

Regression

- Continuous output(s)

points = examples \rightarrow curve = regression

Classification

- Discrete output(s)

Input = point position

target Output = class label (\( \square = -1, + = +1 \))

Function label = f(x) (and separation boundary)
**Supervised learning**

Examples (input-output) 
\((x_1,y_1), (x_2,y_2), \ldots, (x_n, y_n)\)

\(H\) (parameterized) family of mathematical models

Hyper-parameters for training algorithm

**LEARNING ALGORITHM**
(usually based on optimization technique)

\(h^* \in H\) so that \(h^*(x_i) \approx y_i\)

In most cases, \(h^* = \text{argMin}_{h \in H} K(h, \{(x_i,y_i)\})\) where \(K=\text{cost} \ K = \sum_i \text{loss}( h(x_i), y_i ) \ [+ \text{regularization-term}] \) and \(\text{loss}=||h(x_i) - y_i||^2\)

---

**Cost function and loss function**

Most supervised Machine-Learning algorithms work by minimizing a "cost function"

- The cost function is generally the average over all training examples of a "loss function"

\[ K = \sum_i \text{loss}( h(x_i), y_i ) \]
\(+ \text{sometimes an additional « regularization » term}\)

- The loss function is usually some measure of the difference between target value and prediction by the output of the learnt model
Many different supervised ML approaches & algorithms

- Linear regressions
- Decision trees (ID3 or CART algorithms)
- Bayesian (probabilistic) methods
- ...
- **Multi-layer neural networks** trained with gradient backpropagation
- **Support Vector Machines**
- **Boosting** of "weak" classifiers
- **Random forests**
- **Deep Learning** (Convolutional Neural Networks, ...)
- ...

Usual two distinct phases of supervised Machine-Learning

**Training**

- Pedestrians
- "non-pedestrians"
- cars
- "non-cars"

**Recognition**

Input

CLASSIFIER

Category (class)
Typology of classification methods

- By similarity ➔ Nearest Neighbors (kNN)
- By succession of elementary tests ➔ Decision Trees
- By probabilistic computations (using hypothesis on distribution of classes) ➔ Bayesian methods
- By error minimization (gradient descent, etc...) ➔ Neural Networks, etc...
  Idem + "margin" maximization ➔ Support Vector Machines (SVM)
- By voting committee (ensemble methods):
  - using trees ➔ Random Forests
  - using successive weightings of examples ➔ Boosting

Nearest Neighbors algorithm

Principle of Nearest Neighbors (kNN) for classification

[What are the main drawbacks of this method??]
# Linear Multivariate Regression

### Linear Regression, Mean Square Loss:
- decision rule: \( y = W'X \)
- loss function: \( L(W, y^i, X^i) = \frac{1}{2}(y^i - W'X^i)^2 \)
- gradient of loss: \( \frac{\partial L(W, y^i, X^i)}{\partial W}' = -(y^i - W(t)'X^i)X^i \)
- update rule: \( W(t + 1) = W(t) + \eta(t)(y^i - W(t)'X^i)X^i \)
- direct solution: solve linear system \([\sum_{i=1}^{P}X^iX^i]'W = \sum_{i=1}^{P}y^iX^i\)

[From slide by Y. LeCun: Machine Learning and Pattern Recognition ]

---

# Logistic Multivariate Regression

If target output is binary (classification)

### Logistic Regression, Negative Log-Likelihood Loss function:
- decision rule: \( y = F(W'X) \), with \( F(a) = \tanh(a) = \frac{1-\exp(a)}{1+\exp(a)} \) (sigmoid function).
- loss function: \( L(W, y^i, X^i) = 2 \log(1 + \exp(-y^iW'X^i)) \)
- gradient of loss: \( \frac{\partial L(W, y^i, X^i)}{\partial W}' = -(y^i - F(W'X)))X^i \)
- update rule: \( W(t + 1) = W(t) + \eta(t)(y^i - F(W(t)'X^i))X^i \)

[From slide by Y. LeCun: Machine Learning and Pattern Recognition ]
Different types of classification errors

<table>
<thead>
<tr>
<th>Reference</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>FN</td>
<td>TP</td>
</tr>
<tr>
<td></td>
<td>FP</td>
</tr>
<tr>
<td>TN</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>predicted as positive</th>
<th>predicted as negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>positive</td>
<td>TP</td>
<td>FN</td>
</tr>
<tr>
<td>negative</td>
<td>FP</td>
<td>TN</td>
</tr>
</tbody>
</table>

Error rate = \[
\frac{(FP+FN)}{(TP+TN+FP+FN)}
\]

BUT: False Negatives ("missed") ≠ False Positives!

Recall: percentage of relevant examples successfully predicted/retrieved

Precision: percentage of actually relevant examples among all those returned by the classifier

Accuracy, recall & precision formulas

<table>
<thead>
<tr>
<th></th>
<th>predicted as positive</th>
<th>predicted as negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>positive</td>
<td>TP</td>
<td>FN</td>
</tr>
<tr>
<td>negative</td>
<td>FP</td>
<td>TN</td>
</tr>
</tbody>
</table>

Accuracy ("correctness") [en français, exactitude] = \[
\frac{\text{# of correct predictions}}{\text{Total # of examples}} = \frac{TP + TN}{TP + TN + FP + FN}
\]

Recall (sensitivity) = \[
\frac{\text{# of correct positive predictions}}{\text{# of real positives}} = \frac{TP}{TP + FN}
\]

Precision (specificity) = \[
\frac{\text{# of correct positive predictions}}{\text{# of positive predictions}} = \frac{TP}{TP + FP}
\]
Classification performance metrics

- **Accuracy** = proportion of correct
- **Recall (sensitivity)** \(\approx\) proportion of “not missed” 
  \(\approx\) “completeness” level [exhaustivité]
- **Precision (specificity)** \(\approx\) reliability of predicted labels
- **Confusion matrix**: predicted label v.s. true label

<table>
<thead>
<tr>
<th>C. Matrix</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>ACTUAL</th>
<th>RECALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>339</td>
<td>15</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>359</td>
<td>94.43%</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>305</td>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>334</td>
<td>91.32%</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>10</td>
<td>242</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>258</td>
<td>93.80%</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>302</td>
<td>30</td>
<td>0</td>
<td>332</td>
<td>90.96%</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>15</td>
<td>368</td>
<td>0</td>
<td>383</td>
<td>98.08%</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>394</td>
<td>394</td>
<td>100.00%</td>
</tr>
<tr>
<td>PREDICTED</td>
<td>360</td>
<td>330</td>
<td>261</td>
<td>317</td>
<td>398</td>
<td>394</td>
<td>2060</td>
<td>94.43%</td>
</tr>
</tbody>
</table>

**Precision - recall trade-off and curve**

Classifier C1 predicts better than C2 iff C1 has better recall *and* precision

+ Trade-off between recall and precision

\[ \Rightarrow \text{Compare precision-recall curves!} \]

For numeric comparison (or if curves cross each other),
**Area Under Curve (AUC)**
Quality measures of learnt model: loss function and error types

- Quality measure for a learnt model h:
  \[ Q(h) = E \left( L(h(x), y) \right) \]
  where \( L(h(x), y) \) is the «LOSS function»
  generally = \( \|h(x) - y\|^2 \)

- What optimum for h?
  \( h^* \) absolute optimum = \( \text{argMin}_h (E(h)) \)
  \( h^*_H \) optimum within \( H \) family = \( \text{argMin}_{h \in H} (E(h)) \)
  \( h^*_H,n \) optimum in \( H \) from finite set of examples = \( \text{argMin}_{h \in H} (E_n(h)) \)

where \( E_n(h) = (1/N) \sum_i (L(h(x_i), y_i)) \)

\[ E(h^*_H,n) - E(h^*) = [E(h^*_H,n) - E(h^*_H)] + [E(h^*_H) - E(h^*)] \]

ESTIMATION error \quad MODEL error

Formal definition of SUPERVISED LEARNING

"LEARNING = APPROXIMATE + GENERALIZE"

Given a FINITE set of examples \((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\)
where \( x_i \in \mathbb{R}^d = \) input vectors, and \( y_i \in \mathbb{R}^s = \) target values
(given by the "teacher"), find a function \( h \) which
"approximates AND GENERALIZES as best as possible"
the underlying function such that \( y_i = f(x_i) + \text{noise} \)

\( \Rightarrow \) goal = to minimize the GENERALIZATION error

\[ E_{\text{gen}} = \int \|h(x) - f(x)\|^2 p(x) dx \]

(where \( p(x) = \) probability distribution of \( x \))
About over-fitting

The generalization error cannot be directly measured, only **empirical error** on examples can be estimated:

$$E_{\text{emp}} = \left( \sum_i \| h(x_i) - y_i \|^2 \right) / n$$

Fitting a data set to different orders of polynomials [from Bishop, "Pattern Recognition and Machine Learning"]

Detection of over-fitting for an iterative algorithm

Machine-Learning methodology: importance of validation set

To **avoid over-fitting** and maximize generalization, absolutely **essential** to use some **validation estimation**, for optimizing training hyper-parameters (and stopping criterion):

- either use a **separate validation dataset** (random split of data into Training-set + Validation-set)
- or use **CROSS-VALIDATION**:
  - Repeat k times: train on (k-1)/k proportion of data + estimate error on remaining 1/k portion
  - Average the k error estimations

3-fold cross-validation:
- Train on S1∪S2 then estimate errS3 error on S3
- Train on S1∪S3 then estimate errS2 error on S2
- Train on S2∪S3 then estimate errS1 error on S1
- Average validation error: (errS1+errS2+errS3)/3
Empirical error and VC-dimension

• In practice, the only error that can be estimated and minimized is the empirical error computed on a finite set of examples:

\[ E_{\text{emp}} = \left( \frac{\sum_i \| h(x_i) - y_i \|^2}{n} \right) \]

• According to « regularization theory » and theoretical result by Vapnik, minimizing \( E_{\text{emp}}(h) \) within \( h \in H \) shall also minimize \( E_{\text{gen}} \) if \( H \) has a finite VC-dimension

**VC-dimension**: maximum cardinal \( \forall \) so that for any set \( S \) of \( \forall \) points, all dichotomies of \( S \) can be performed by one \( h \in H \) (VC-dim \( \approx \) complexity of \( H \))

\[ [\text{VC-dimension \{hyperplanes of } \mathbb{R}^n \}] \]

Regularization by adding penalty to the cost function

Vapnik has shown that:

\[ \text{Proba}(\max_{h \in H} |E_{\text{gen}}(h) - E_{\text{emp}}(h)| \geq \varepsilon) < G(n, \delta, \varepsilon) \]

where \( n \) = # of examples and \( \delta = \text{VC-dim} \) and \( G \) decreases with \( \delta/n \)

⇒ to be sure that \( E_{\text{gen}} \) en decreases when minimizing \( E_{\text{emp}} \), the smaller \( n \) is, the smaller the VC-dim \( \delta \) needs to be

A possible way to automatically reduce VC-dim is to modify the cost function into: \( C = E_{\text{emp}} + \Omega(h) \)

where \( \Omega(h) \) penalizes « complexity » of \( h \)

\( \Rightarrow \) reduction of « effective » VC-dim

NB: \( \approx \) application of "Occam's razor"!!

\( \approx \) "why do complicated if it can be done simpler?"

Usual for of regularization penalty: $L_1$ norm

In many cases, the complexity (in VC-dim sense) increases with maximum value of its parameters $w_i \Rightarrow$ interesting to penalize large values of $w_i$

Usually done by modifying cost function into

$$C = E_{emp} + \lambda \sum_i (||w_i||)$$

Example: LASSO = regularized linear regression

$$\min_w (\sum_j ||y_j - w \cdot x_j||_2^2 + \lambda ||w||_1)$$

[L$_1$-norm penalization of regressor]

NB: if using $L_0$ (# of NON-ZERO components) penalization (instead of $L_1$), we can obtain SPARSE model

Synthesis on various algorithms for SUPERVISED Machine-Learning
Supervised learning

Examples (input-output)
\((x_1,y_1), (x_2,y_2), \ldots, (x_n, y_n)\)

\(H\) (parameterized) family of mathematical models

Hyper-parameters for training algorithm

LEARNING ALGORITHM
(usually based on optimization technique)

\(h \in H\)
so that
\(h(x_i) \approx y_i\)

Summary of main shallow SUPERVISED learning algorithms

- **Decision trees:** naturally adapted to symbolic input, very fast, good scaling for very high number of classes, *white* box;
  BUT noise sensitive

- **Multi-layer neural networks:** universal approximators, good generalization, easy handling of multi-class;
  BUT optimum model NOT guaranteed, many critical hyper-parameters (# hidden neurons, weight init., learning rate, # training epochs,…)

- **Support Vector Machines:** maths-guaranteed optimal separation, possible handling of structured input (graphs, etc…) via kernel;
  BUT not very efficient for multi-class (n times 1-vs-all SVMs), training computation rises quickly with input dim and # of examples \(O(\max(n,d) \min(n,d)^2)\)

- **Boosting of « weak » classifiers:** simple algo, can build strong classifier from any weak classifier, can select features during training;
  BUT not very efficient for multi-class (n times 1-vs-all SVMs)

- **Random forests:** OK for symbolic input, robustness to noise, very fast to compute, efficient for large # of classes and high input dim;
  BUT training sometimes long
### Model type choice criteria for SUPERVISED learning

<table>
<thead>
<tr>
<th></th>
<th>MLP Neural Network</th>
<th>Conv. Net (Deep Learning)</th>
<th>SVM</th>
<th>Boosting</th>
<th>Decision Tree</th>
<th>Random Forest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robustness to noise and erroneous labels</td>
<td>+</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>--</td>
<td>++</td>
</tr>
<tr>
<td>Ease/speed of training</td>
<td>-</td>
<td>---</td>
<td>+</td>
<td>++</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Execution time</td>
<td>-</td>
<td>--</td>
<td>--</td>
<td>++</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Many classes</td>
<td>+</td>
<td>+</td>
<td>--</td>
<td>--</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>High dimension of input</td>
<td>REQUIRED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Many examples</td>
<td>REQUIRED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interpretable (« white » box)</td>
<td>YES</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data OTHER than vectors of values</td>
<td>Structured (string, graph)</td>
<td>symbolic</td>
<td>symbolic</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Handling of features</td>
<td>Learn them</td>
<td>Automated selection</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Some REFERENCE TEXTBOOKS on Statistical Machine-Learning

- **Introduction au machine learning**  
  C. Azencott, Dunod (2018).  

- **The Elements of Statistical Learning (2nd edition)**  

- **Deep Learning**  

- **Pattern recognition and Machine-Learning**  

- **Introduction to Data Mining**  

- **Apprentissage artificiel : concepts et algorithmes**  