Machine Learning: Algorithms and Applications

Floriano Zini

Free University of Bozen-Bolzano
Faculty of Computer Science

Academic Year 2011-2012

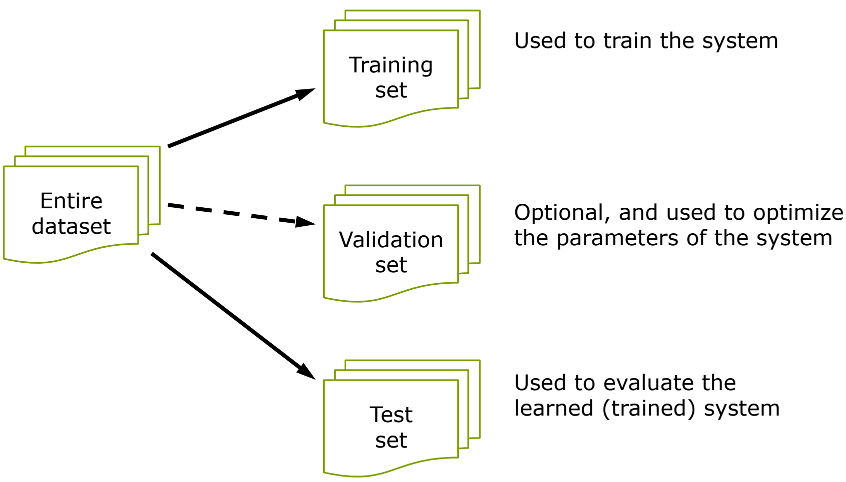
Lecture 6: 2nd April 2012

Evaluation of the ML system's performance

Evaluation of ML system performance

- The evaluation of a ML system's performance is typically conducted **experimentally**, rather than analytically
  - Analytical evaluation aims at proving a system is correct and complete (e.g., theorem prover in logics)
  - *Unable to build a formal specification (definition) of the problem* that a ML system is trying to solve (i.e., what correctness and completeness are)
- We focus on the system performance evaluation that
  - is *automatically done* using a set of instances (i.e., a test set)
  - does not involve real users
- Evaluation **methods**
  - → How to obtain a reliable evaluation of the system's performance?
- Evaluation **metrics**
  - → How to measure the system's performance?

Evaluation methods (1)



Evaluation methods (2)

- How to obtain a reliable estimate of the system's performance?
  - Generally, the larger the training set the better the learned system
  - The larger the test set the more accurate the error estimate
  - Problem: (very) large datasets are not always available
- The performance of the learned system *depends not only on the learning algorithm* but also on some other factors
  - Class distribution
  - Cost of misclassification
  - Size of the training set
  - Size of the test set

Evaluation methods (3)

- Hold-out
- Stratified sampling
- Repeated hold-out
- Cross-validation
  - *k*-fold
  - Leave-one-out
- Bootstrap sampling

Hold-out (Splitting)

- The entire dataset X is divided into two **disjoint** subsets
  - The training set X_train – for training the system
  - The test set X_test – for evaluating the trained system
  - $\rightarrow X = X\_train \cup X\_test$, and typically $|X\_train| >> |X\_test|$
- Motivation
  - Every instance included in the test set *X_test* should not be used in the training phase
  - Every instance used in the training phase (i.e., included in *X_train*) should not be exploited in the test phase
  - The unseen test instances in *X_test* provides an unbiased estimate of the system's predictive accuracy
- A common splitting choice:
  - $|X\_train|=(2/3)*|X|$, $|X\_test|=(1/3)*|X|$
- Suitable when the entire dataset X is large

Stratified sampling

- For small or unbalanced datasets, samples in the training and test sets might not be representative
  - For instance, (very) few, or no, instances of some classes
- Goal: The class distribution in the training and test sets is approximately the same as that in the entire dataset X
- Stratified sampling
  - A way of balancing the data
  - To ensure that each class is represented with approximately equal proportions in the training and test sets
- Stratification does not make sense for numeric prediction systems (i.e., the system's output is a real value, not a class label)

Repeated hold-out

- The hold-out evaluation method is applied repeatedly (i.e., several times) to produce different <X_train, X_test> pairs
  - In each iteration, a certain proportion (e.g. 2/3) of the entire dataset *X* is ***randomly selected*** to construct the training set X_train (possibly with stratification)
  - The error rates made by the system in these iterations on the test set X_test are *averaged* to produce the overall error rate
- Still not perfect
  - There is overlap (i.e., the same instances) among the different test sets used in the iterations

Cross-validation

- To avoid overlapping test sets
- *k*-fold cross-validation
  - The entire dataset *X* is partitioned into *k* **disjoint** subsets (i.e., called *folds*) of (approximately) equal size
  - Each fold in turn is used as the test set and the remainder (i.e., (*k*-1) folds) as the training set
  - The *k* error rates (i.e., each corresponds to a fold used) are averaged to produce the overall error estimate
- Common choice of *k*: 10 or 5
- Often the *k* folds are stratified before the cross-validation evaluation is performed
- Suitable when the entire dataset *X* is not large

Leave-one-out cross-validation

- A special form of cross-validation
  - The number of folds is equal to the size of the dataset (i.e., $k=|X|$)
  - Each fold contains only one instance
- Make the best use (i.e., the highest exploitation) of the dataset
- Involve no random sub-sampling
- Stratification does not make sense
  - → Because there is only one instance in the test set
- Very computationally expensive
  - Suitable when the entire dataset X is very small

Bootstrap sampling (1)

- Cross-validation uses sampling without replacement
  - → An instance, *once selected, cannot be selected again* for including in the training set
- Bootstrap uses ***sampling with replacement*** to form the training set
  - Assume that the entire dataset X consists of n instances
  - Sample the dataset X n times with replacement to form the training set X_train of n instances
    - From the set X, take one instance x randomly (but **not remove** x from the set X)
    - Put the instance x in the training set: $X\_train = X\_train \cup \{x\}$
    - Repeat this process n times
  - Use X_train as the training set
  - Use the instances in X that are **not included** in X_train to form the test set: $X\_test = \{z \in X;\ z \notin X\_train\}$

Bootstrap sampling (2)

- In each step, an instance has a probability of $\left(1-\frac{1}{n}\right)$ not being put in the training set
- Hence, the probability that an instance is (after the bootstrap sampling process) not included in the test set is

$$\left(1-\frac{1}{n}\right)^n \approx e^{-1} \approx 0.368$$

- This means that
  - The training set (i.e., size of n) will contain approximately 63.2% of the instances in X (Note: an instance in X may have **more than one occurrence** in *X_train*)
  - The test set (i.e., size $<n$) will contain approximately 36.8% of the instances in X (Note: an instance in X may have **at most one occurrence** in *X_test*)
- Bootstrap sampling is suitable for (very) small datasets

Validation set

- The instances in the test set cannot be used in any way in the training (learning) of the system
- In some learning problems, the training phase consists of two stages
  - In the first stage, build the learned system (i.e., learn the model)
  - In the second stage, optimize the parameter settings
- The test set cannot be used for parameter tuning
- In this case, the entire dataset X is divided into three subsets: a *training* set, a *validation* set, and a *test* set
- The validation set is used to optimize the parameters used in the learning algorithm
  → For a parameter, the value that produces *the best accuracy on the validation set* is used as the final value of that parameter

Evaluation metrics

- **Predictive accuracy**
  - → How accurate the learned system makes predictions on the test instances
- Efficiency
  - → Time and (memory) resources needed for the training and test phases
- Robustness
  - → How much the system is capable of handling noisy and value-missing instances
- Scalability
  - → How much the system's performance (e.g., speed) is sensitive to the size of the data
- Interpretability
  - → How easily the system's output and operation are understandable to human
- Complexity
  - → How compact (simple) the learned model is

Predictive accuracy

- For classification task
  - → i.e., the system's output is a nominal value

$$Accuracy = \frac{1}{|X\_test|} \sum_{x \in X\_test} Identical(o(x), c(x)); \qquad Identical(a,b) = \begin{cases} 1 & \text{if } (a=b) \\ 0 & \text{otherwise} \end{cases}$$

  - *x*: A test instance in the test set *X_test*
  - *o(x)*: The system's output (i.e. predicted class) for *x*
  - *c(x)*: The desired (true/actual) class for *x*

- For regression task
  - →i.e., the system's output is a real value

$$Error = \sum_{x \in X\_test} Error(x); \qquad Error(x) = |d(x) - o(x)|$$

  - *o(x)*: The system's output (i.e. predicted real value) for *x*
  - *d(x)*: The desired (true/actual) output for x

Confusion matrix

- Also called Contingency Table
- Can be used ***only for classification problems***

- ***TP_i***: The number of c_i-class instances correctly classified
- ***FP_i***: The number of non c_i-class instances misclassified in c_i
- ***TN_i***: The number of non c_i-class instances not classified in c_i
- ***FN_i***: The number of c_i-class instances not classified in c_i

Class c_i		Classified **by the system**	
		Positive	Negative
Desired (**true**) output	Positive	**TP_i**	**FN_i**
	Negative	**FP_i**	**TN_i**

Precision and Recall (1)

- Very often used in the evaluation of text classification/categorization systems
- **Precision** w.r.t. class c_i
  - → The number of c_i-class instances *correctly* classified divided by the number of instances classified in c_i

$$Precision(c_i) = \frac{TP_i}{TP_i + FP_i}$$

- **Recall** w.r.t. class c_i
  - → The number of c_i-class instances *correctly* classified divided by the number of c_i-class instances

$$Recall(c_i) = \frac{TP_i}{TP_i + FN_i}$$

Precision and Recall (2)

- How to compute the overall precision and recall for the entire set of classes $C=\{c_i\}$?
- Macro-averaging

$$Precision = \frac{\sum_{i=1}^{|C|} Precision(c_i)}{|C|} \qquad Recall = \frac{\sum_{i=1}^{|C|} Recall(c_i)}{|C|}$$

- Micro-averaging

$$Precision = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|} (TP_i + FP_i)} \qquad Recall = \frac{\sum_{i=1}^{|C|} TP_i}{\sum_{i=1}^{|C|} (TP_i + FN_i)}$$

- Macro-averaging gives equal weight to every class, while micro-averaging gives equal weight to every instance

F_1-measure

- A measure that combines the precision and recall estimates

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

- F_1-measure is the ***harmonic mean*** of the precision and recall estimates
  - F_1-measure tends to be closer to the smaller one between the precision and recall estimates
  - F_1-measure is high if both precision and recall is high

Model selection

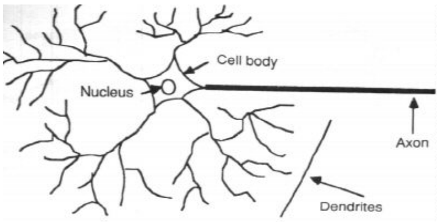
- Model selection criteria attempt to find a good compromise between
  - The complexity of the learned system (model)
  - The learned system's predictive accuracy on the training set
- *Occam's razor*. A good model is a **simple** model that achieves **high accuracy** on the given data
- Example
  - Classifier *Sys1*: (very) simple, fits the training set relatively well
  - Classifier *Sys2*: significantly complex, fits the training set perfectly

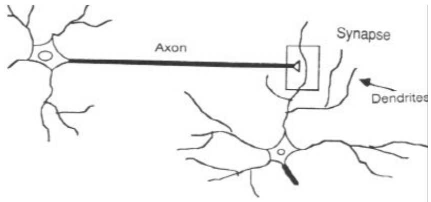
  → Classifier *Sys1* is preferred to classifier *Sys2*

Neural Networks

Introduction (1)

- Human brain
  - Densely interconnected network of 10^{11} neurons each connected to 10^{4} others (neuron switching time : approx. 10^{-3} sec.)
- Artificial neural network (ANN)
  - Mimics the highly parallel information processing of human brain

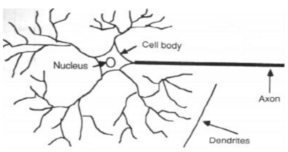


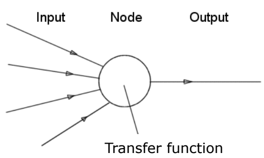


Introduction (2)

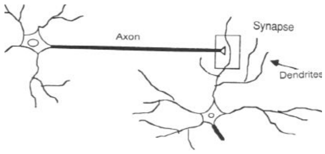
- ANNs incorporate the two fundamental components of biological neural nets

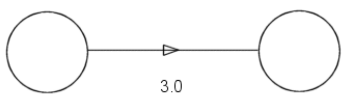
1. Neurons (nodes)





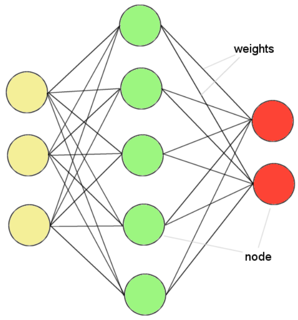
2. Synapses (weights)





Introduction (3)

- ANN is a structure (network) composed of many interconnected units (artificial neurons)
- ANN has the ability to learn, recall, and generalize from training data by assigning and adjusting the interconnection weights
- Each unit (neuron)
  - Has an input/output (I/O) transfer function
  - Implements a local computation (i.e., local function)
- The output of a unit is determined by
  - Its (possibly external) inputs
  - Its I/O transfer function
- The overall function is determined by
  - The network topology
  - The individual neuron characteristic
  - The learning (training) strategy
  - The training data



Artificial neural networks – When?

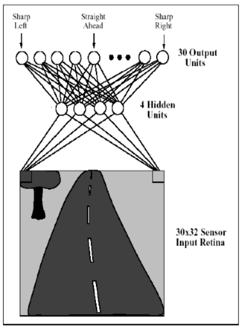
- Input is high-dimensional discrete or real-valued (e.g., raw sensor input)
- Output is real-valued, discrete-valued or vector-valued
- Possibly noisy data
- Long training time is accepted
- Short classification/prediction time is required
- Human readability of result is not (very) important

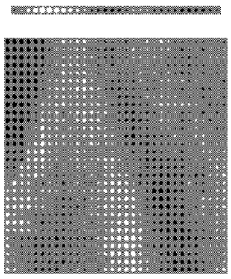
Application examples

- Image processing
  - E.g., image matching, classification, or segmentation
- Financial systems
  - E.g., stock market analysis, credit card authorization, and securities trading
- Pattern recognition
  - E.g., speech recognition and understanding, character (letter or number) recognition, face recognition, and handwriting analysis
- Medicine
  - E.g., electrocardiographic signal analysis and understanding, diagnosis of various diseases, and medical image processing

ALVINN

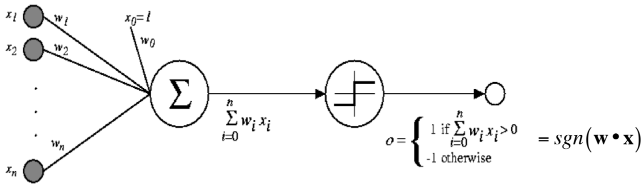
- ANN learned to drive at up to 112 Km/h for 144 Km on the highway





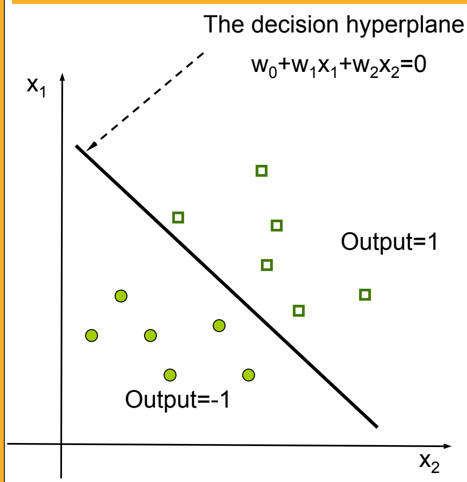
Perceptron

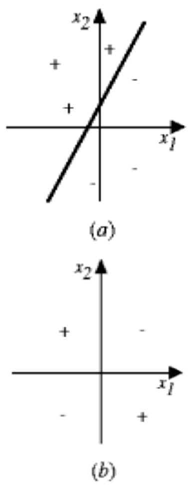
- A perceptron is the simplest type of ANN



- $x_1,\dots,x_n$: inputs
- $w_1,\dots,w_n$: weights
- w_0: threshold
- $x_0=1$: additional constant input
- Learning a perceptron means choosing values for $w_0,\dots,w_n$
- The hypothesis space is $H = \{\mathbf{w} \mid \mathbf{w} \in \Re^{n+1}\}$

Perceptron – Illustration





Linearly separable case like (a): Possible to classify by hyperplane
Linearly inseparable case like (b): Impossible to classify

AND function

<Training examples>

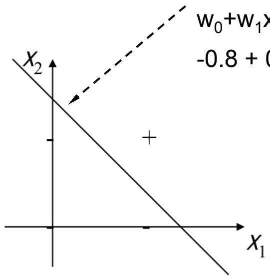
x_1	x_2	output
0	0	–1
0	1	–1
1	0	–1
1	1	1

The AND function is implemented by a perceptron where w_0=-0.8, w_1=w_2=0.5

The decision hyperplane

$w_0+w_1x_1+w_2x_2= 0$

$-0.8 + 0.5\ x_1+ 0.5\ x_2= 0$



<Test Results>

x_1	x_2	$\Sigma\, w_i x_i$	output
0	0	–0.8	–1
0	1	–0.3	–1
1	0	–0.3	–1
1	1	0.2	1

OR function

<Training examples>

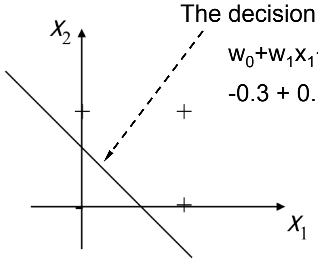
x_1	x_2	output
0	0	–1
0	1	1
1	0	1
1	1	1

The OR function is implemented by a perceptron where w_0=-0.3, w_1=w_2=0.5

The decision hyperplane

$w_0+w_1x_1+w_2x_2= 0$

$-0.3 + 0.5\ x_1+ 0.5\ x_2= 0$



<Test Results>

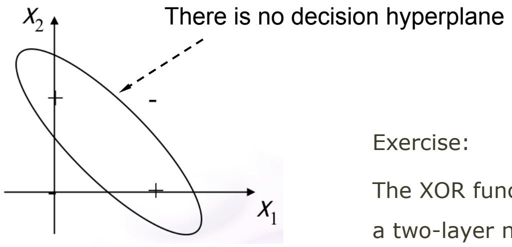
x_1	x_2	$\Sigma w_i{}^*x_i$	output
0	0	-0.3	-1
0	1	0.2	1
1	0	0.2	1
1	1	0.7	1

XOR function

<Training examples>

x_1	x_2	output
0	0	–1
0	1	1
1	0	1
1	1	–1

The XOR function is not implementable by a perceptron because positive and negative instances are not linearly separable



Exercise:

The XOR function can be implementable by a two-layer network of perceptrons

$$x_1 \oplus x_2 = x_1 \cdot \overline{x_2} + \overline{x_1} \cdot x_2$$