Machine Learning Notes

by Mattia Orlandi

2. Classification

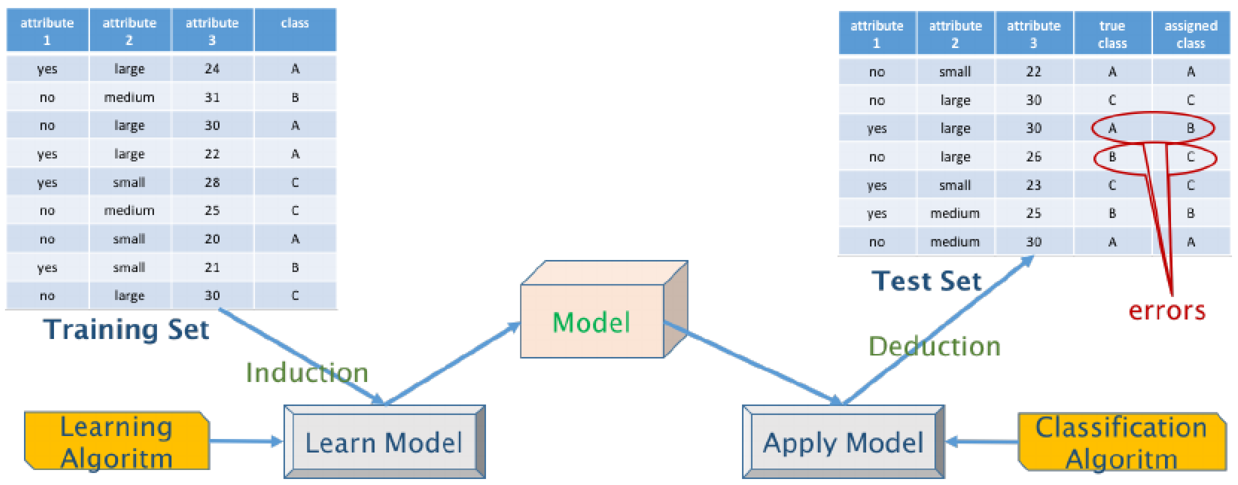
2.1. Introduction

Classification Model

- An algorithm which computes the class of an individual whose class is unknown.
- The algorithm is *parametrized* in order to optimize the results for the specific problem at hand.
- Developing a classification model requires:
    - choice of the learning algorithm;
    - letting the algorithm learn its parametrization;
    - assessing the quality of the classification model.
- The classification model is used by a run-time classification algorithm with the developed parametrization.

Workflow

1. Learning the model for the given set of classes:
    - a training set is available, containing a number of individuals;
    - for each individual the value of the class label is available;
    - the training set should be *representative* as much as possible, and therefore should be obtained by a random process.
2. Estimate the *accuracy* of the model:
    - a *test set* is available, for which the class labels are known;
    - the model is run by a *classification algorithm* to assign labels to individuals;
    - labels assigned are compared with true ones, to estimate accuracy.
3. The model is used to label new individuals:
    - if true labels become available, they can be used to compare true accuracy with the estimated one.



attribute 1	attribute 2	attribute 3	class
yes	large	24	A
no	medium	31	B
no	large	30	A
yes	large	22	A
yes	small	28	C
no	medium	25	C
no	small	20	A
yes	small	21	B
no	large	30	C

attribute 1	attribute 2	attribute 3	true class	assigned class
no	small	22	A	A
no	large	30	C	C
yes	large	30	A	B
no	large	26	B	C
yes	small	23	C	C
yes	medium	25	B	B
no	medium	30	A	A

Approaches for classification:

- **Crisp** ⇒ the classifier assigns to each individuals *one label* (necessary when an immediate action is required).
- **Probabilistic** / **Soft** ⇒ the classifier assigns to each individuals the *probability* of each label (more appropriate when classification is part of a process including several evaluation/action steps).

2.2. Classification with Decision Trees

Classification Algorithm

Run-time classifier with a tree-shaped set of tests:

- The decision tree has *inner nodes* and *leaf nodes*.
- Starting from the root, tests are performed and, based on the result, branch are explored.
- Once a leaf is reached, the corresponding label is assigned.

Model Generation

Given a set $\mathcal{E}$ of elements for which the class is known, grow a decision tree as follows:

- if all the elements belong to class c or $\mathcal{E}$ is *small*, generate a leaf node with label c;
- otherwise:
  - choose a test based on a single attribute with two or more outcomes;
  - make this test the root of a tree with one branch for each of the outcomes of the test;
  - partition $\mathcal{E}$ into subsets corresponding to the outcomes and apply recursively the procedure to the subsets.

Problems:

- choice of the attribute to be tested;

- choice of the kind of test;
- meaning of *small* for set $\mathcal{E}$.

Obs.: contingency tables provide an insight on correlations between attribute values.

Entropy and Information Gain

Pattern evaluation: based on information theory and on the concept of **entropy**.

Given a source X with V possible values, with probability distribution:

$$P(v_1) = p_1, \ldots, P(v_V) = p_V$$

the best coding allows the transmission with an average number of bits given by

$$H(X) = -\sum_j p_j \log_2(p_j)$$

where $H(X)$ is the **entropy** of the information source X.

- **High entropy** $\Rightarrow$ probabilities mostly similar (**flat** histogram).
- **Low entropy** $\Rightarrow$ some symbols have much higher probability (histogram with **peaks**).
- Higher number of allowed symbols $\Rightarrow$ higher entropy.

E.g.: in a binary source with symbol probabilities p and $1-p$, entropy is zero when $p=0$ or $p=1$.

Conditional Specific Entropy: the entropy of Y considering only the rows s.t. $X=v$

$$H(Y|X=v)$$

Conditional Entropy: the weighted average of conditional specific entropy of Y w.r.t. the values of X

$$H(Y|X) = \sum_j P(X=v_j) \cdot H(Y|X=v_j)$$

Information Gain: the amount of insight that X provides to forecast the values of Y

$$IG(Y|X) = H(Y) - H(Y|X)$$

Decision Tree - Learning

1. Choose the attribute giving the highest information gain.
2. Partition the dataset according to the chosen attribute.
3. Choose as class label of each partition the majority.

4. Build recursively a new tree starting from each subset where the minority is non-empty.

It uses a **greedy** strategy: it chooses the local optimum and never backtrack.

Errors and Overfitting

The **training set error** is computed by executing the generated decision tree on the training set, and by counting the number of discordances between the true classes and the predicted ones (it can be non-zero, therefore the Decision Tree can fail).

The labelled (supervised) data is partitioned into a *training set*, used to generate the model, and in a *test set* used with the generated model to compute the **test set error**, which is more indicative of the expected behaviour with new data.

Overfitting: when a learning algorithm is affected by noise, the performance on the test set is much worse than that on the training set.

- More formally, a decision tree is a *hypothesis* of the relationship between the predictor attributes and the class:
    - h : hypothesis
    - $error_{train}(h)$: error of hypothesis on training set
    - $error_{\mathcal{E}}(h)$: error of hypothesis on entire dataset
- h overfits the training set if there is an alternative hypothesis h' s.t.

$$error_{train}(h) < error_{train}(h')$$
$$error_{\mathcal{E}}(h) > error_{\mathcal{E}}(h')$$

Causes for overfitting:

- Noise: individuals in test set can have bad values in the predicting attributes/in the class label.
- Lack of representative instances: some real-world situations can be underrepresented/not represented at all in test set.

Good hypotheses have low *generalization* error, thus they work well with examples different from the ones used in training.

Decision Tree - Pruning

From Occam’s Razor, it follows that a long hypothesis which fits data is more likely to be a coincidence than a short hypothesis ⇒ **pruning** a decision tree is a way to simplify it.

Possibilities:

- Pre-pruning: early stop of tree-growth before it perfectly classify training set.
- Post-pruning: pruning, according to some criterion, portions of the tree after it has been completely built ⇒ preferred, since it’s difficult to estimate when to stop the

tree growth.

Post-pruning criteria:

- **Validation set**:
  - the supervised dataset is spitted in three parts:
    - **training set**: basis for building the classification model;
    - **validation set**: the model is *tuned* (e.g. pruned) to minimize the error;
    - **test set**: assess the final expected error;
  - the three sets must be independent.
- **Statistical pruning**: based on error estimation and significance testing (compatibility of a node with a random effect, cfr. 2.3).
- **Minimum Description Length - MDL**:
  - learning process produces a *theory*, which can be encoded;
  - errors can be encoded as *exceptions* to the theory;
  - the length of the theory description is given by the sum of theory encoding and exceptions encoding $\Rightarrow$ theory with shortest description is preferred;
  - given theory T and training set $\mathcal{E}$, define:
    - $L(T)$: lenght of encoding of T
    - $L(\mathcal{E}|T)$: length of encoding of exceptions
  - most probable theory is given by Bayes theorem:

    $$P(T|\mathcal{E}) = P(\mathcal{E}|T) \cdot P(T)/P(\mathcal{E})$$

    which is equivalent to

    $$-\log(P(T|\mathcal{E})) = -\log(P(\mathcal{E}|T)) - \log(P(T)) + \log(P(\mathcal{E}))$$

  - maximizing probability is equivalent to minimizing the absolute value of logarithm

    $$\min(-\log(P(\mathcal{E}|T)) - \log(P(T)))$$

    which is related to minimization of encoding (from Information Theory)

    $$\min(-L(\mathcal{E}|T) - L(T))$$

  - only partly comparable with performance analysis of a classifier, since *test set errors* should also be taken into account.

Specification of the test for an attribute

- Discrete:
  - it depends on value domain of attributes;
  - if domain is discrete with V values, the split generates V nodes.
- Continuous:
  - it depends on value domain of attributes;
  - if domain is continuous with V values, the split into V nodes is **not feasible** (high number of branches $\Rightarrow$ small subsets $\Rightarrow$ small significance);
  - **discretization**: the continuous domain of attributes is converted into a discrete set of values (intervals are chosen according to the characteristics of value

domain);
- **binarization**: discretization with only two values (split point is chosen taking into account the relationship between attribute values and class values);
- **threshold**:
  - let $d \in \mathcal{D}$ be a real-valued attribute, let t be a value of the domain of d, let c be the class attribute
  - entropy of c w.r.t. d with threshold t:

$$H(c|d:t) = H(c|d<t) \cdot P(d<t) + H(c|d \geq t) \cdot P(d \geq t)$$

  - information gain provided by "d exceeds threshold t" to forecast class c:

$$IG(c|d:t) = H(c) - H(c|d:t)$$

  - information gain provided by d to forecast c:

$$IG(c|d) = \max_t IG(c|d:t)$$

  - the complexity of $IG(c|d)$ is $N \cdot \log N + 2 \cdot N \cdot V_d$, where
    - N is the number of individuals in the node under consideration;
    - V_d is the number of distinct values of the attribute d.

Impurity functions

The split to be performed must be the one generating maximum **purity** $\Rightarrow$ necessity of a measure for the purity (or impurity) of a node.

Impurity functions:

- **Information Gain**
- **Gini Index**:
  - consider a node p with C_p classes
  - for class j:
    - $f_{p,j}$: frequency of wrong classification given by random assignment
    - $1 - f_{p,j}$: frequency of other classes
    - $f_{p,j} \cdot (1 - f_{p,j})$: probabilityof wrong assignment
  - Gini index is the total probability of wrong classification:

$$\sum_j f_{p,j} \cdot (1 - f_{p,j}) = \sum_j f_{p,j} - \sum_j f_{p,j}^2 = 1 - \sum_j f_{p,j}^2$$

    it has maximum value $1 - 1/C_p$ when all records are uniformly distributed over all classes, and it has minimum value 0 when all records belong to the same class.
- **Misclassification Error**:
  - if a node is a leaf, it’s possible to find the highest label frequency, which is the *accuracy* of the node;
  - the misclassification error is the complement to 1 of the accuracy:

$$ME(p) = 1 - \max_j f_{p,j}$$

    - the choice of the split is done in the samw way as for the Gini index;
    - ME is linear, therefore is less robust to errors in the frequency.

Final remarks

Algorithms for building DTs:

- several variants depending on
    - tree construction strategy;
    - partitioning strategy;
    - pruning strategy;
- tests based on linear combinations for numeric attributes;
- multivariate tests.

Complexity of DT induction:

- N instances and D attributes in $\mathcal{E} \Rightarrow$ tree height is $\mathcal{O}(\log(N))$;
- each level of the tree requires consideration of all dataset, and each node requires consideration of all attributes $\Rightarrow$ overall cost is $\mathcal{O}(D \cdot N \log(N))$;
- binary split of numeric attributes costs $\mathcal{O}(N \log(N))$, without increment of complexity;
- pruning requires to consider globally all instances at each level $\Rightarrow$ additional $\mathcal{O}(N \log(N))$ which does not increase complexity.

Characteristics of DT induction:

- Non-parametric approach to build classification model.
- Finding *best* DT is NP-complete $\Rightarrow$ heuristics used to find sub-optimal solutions in reasonable time.
- Extremely efficient at runtime to classify new instances: $\mathcal{O}(h)$, h height of the tree.
- Robust w.r.t. noise in training set if overfitting is avoided with appropriate pruning.
- Redundant attributes do not cause any difficulty (if two attributes are strongly correlated and the first one is chosen, the second one probably will never be chosen since it will never provide a good increment of node purity).
- Nodes at high depth easily irrelevant and thus pruned.
- Impurity measure has low impact on final result.
- Pruning strategy has high impact on final result.
- Portions of tree easily replicated, due to *one attribute at a time* strategy.

2.3. Model Selection: Evaluation of a Classifier

- In supervised learning the training set performance is overoptimistic $\Rightarrow$ necessity of a lower bound for performance obtained by independent tests.
- Supervised data are usually scarce $\Rightarrow$ necessity to balance the use of them between train, validation (possibly omitted) and test.
- One must evaluate how much the theory fits the data and the cost generated by prediction errors.
- The more training data are used, the more performing is the classifier.

- The evaluation is independent from the algorithm used to generate the model.

Error Estimation

Meaning of *test error*:

- suppose that test set is a good representation *on the average* of the entire dataset $\mathcal{E}$
- the relationship between training set and $\mathcal{E}$ will be subject to probabilistic variability
- evaluation can be done at two levels:
  - general: performance of the whole classifier
  - local: performance of single components of the model (e.g. nodes)
- if test set error ratio is x, runtime error will be $x \pm \epsilon \Rightarrow$ **confidence interval**.

Confidence intervals in error estimation:

- let p be the probability of error, N the number of binary random events of the same type, and f the empirical frequency of error
- increasing N, with constant f, results in a narrowing of the uncertainty for p

Statistical pruning of DTs

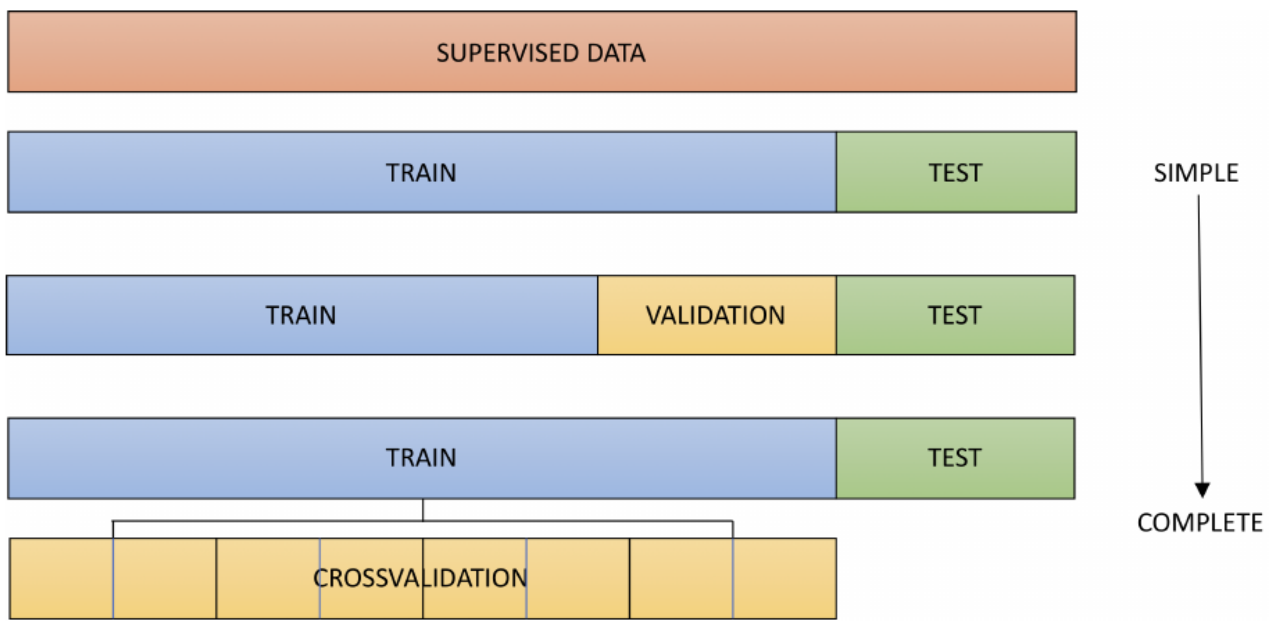
- Consider a subtree near leaves.
- Compute its maximum error e_l as a weighted sum of the maximum error of the leaves.
- Compute the maximum error e_r of the root of the subtree transformed into a leaf.
- Prune if $e_r \leq e_l$.
- With pruning the error frequency increases, but the same does the number of records in node $\Rightarrow$ maximum error can decrease.

Testing a classifier

- Error frequency is the simplest indicator of the quality of a classifier (maximum error is more used than empirical error).
- Accuracy and other more sophisticated indicators are used to:
  - compare different classifiers/parameter settings;
  - estimate runtime performance and thus cost of errors.

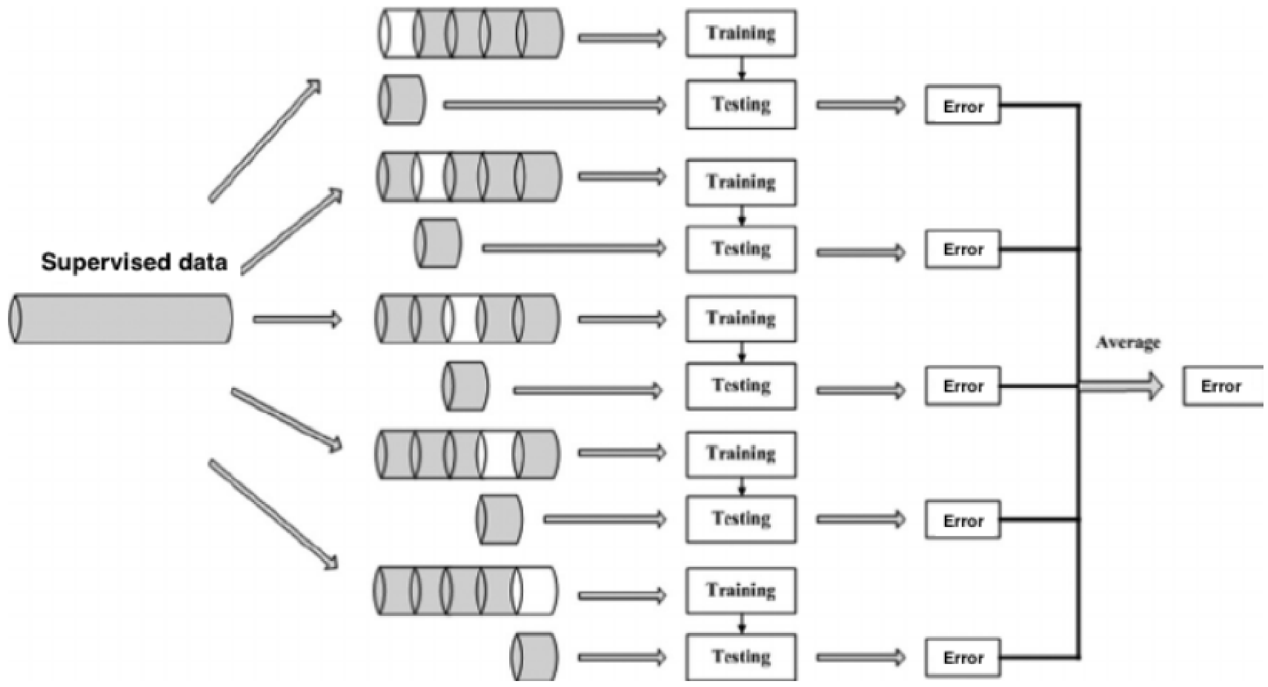
Hyperparameters:

- Parameters of every machine learning algorithms.
- Several train/test loops are necessary to find best values for hyperparameters.



Testing strategies:

- Holdout:
    - splitting data into *training set* and *test set*
        - training/test ratio: $2/1$
        - split as random as possible
        - test set is used to estimate performance with new data
    - splitting data into *training set*, *validation set* and *test set*
        - validation set is used to tune hyperparameters
- Crossvalidation (k-fold):
    - training set randomly partitioned (possibly stratified) into k subsets
    - k iterations using one of the subsets for test and the others for training
    - combine tests results
    - generate final model using entire training set
    - optimal use of supervised data (each record used $k-1$ times for training and once for testing, tipical value $k=10$)
    - estimate is more reliable, but process is slower
    - in the extreme case $k=N$, there's no random partitioning and it's intrinsically non-stratified



- Bootstrap:
  - sampling of N records with **replacement**
  - the records that are not selected are used for test

Performance measures of a classifier

For binary prediction:

- $\text{success rate/accuracy} = \frac{TP+TN}{N_{test}}$, with TP : true positives and TN : true negatives
- $\text{error rate} = 1 - \text{success rate}$

		Predicted class	
		P	N
True class	P	TP	FN
	N	FP	TN

Accuracy is not the only indicator:

- velocity
- robustness w.r.t. noise
- scalability
- interpretability

Measures:

- Precision : $TP/(TP+FP) \Rightarrow$ rate of true positives among positive classifications
- Recall : $TP/(TP+FN) \Rightarrow$ also called **Sensitivity**, it's the rate of positives that can be caught
- Specificity : $TN/(TN+FP) \Rightarrow$ rate of negatives that can be caught
- Accuracy : $\text{Sensitivity} \cdot \frac{\text{pos}}{N} + \text{Specificity} \cdot \frac{\text{neg}}{N} \Rightarrow$ weighted sum of sensitivity and specificity
- F-measure : $2\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision}+\text{Recall}} \Rightarrow$ also called **F1 score** or **balanced F-score** armonic mean

In the multidimensional case, the 2D table for binary predictions is extended and is called **confusion matrix**:

- each cell contains the number of test records of class i and predicted as class j;
- the numbers on the main diagonal are the true predictions.

Accuracy does not take into account the a-priori information.

κ statistic

- Let $\bar{C}$ be a classifier whose confusion matrix has a certain proportion of predicted classes, and suppose it makes $x_{\bar{C}}$ correct predictions.
- Let $R_{\bar{C}}$ be a random classifier producing the same proportion of classes as $\bar{C}$, and suppose it makes $x_{R_{\bar{C}}}$ correct predictions **by chance**.
- Suppose the perfect classifier makes x_P correct predictions.
- The improvement of $\bar{C}$ over $R_{\bar{C}}$ is $\mu_{\bar{C}} = x_{\bar{C}} - x_{R_{\bar{C}}}$.
- The improvement of $\bar{C}$ over $R_{\bar{C}}$ is $\mu_P = x_P - x_{R_{\bar{C}}}$.
- $\kappa(\bar{C})$ is the improvement of classifier $\bar{C}$ over the perfect classifier

$$\kappa(\bar{C}) = \frac{\mu_{\bar{C}}}{\mu_P}$$

κ-statistic evaluates the concordance between two classifications:

- the probability of concordance is $P(c) = \frac{\sum_i^{n_{classes}} TP_i}{N}$
- $P(r)$ is the probability of random concordance
- κ is the ratio between concordance exceeding the random component and maximum surplus possible

$$-1 \leq \kappa = \frac{P(c) - P(r)}{1 - P(r)} \leq 1$$

- 1 means perfect agreement

$$\sum_i^{n_{classes}} TP_i = N$$

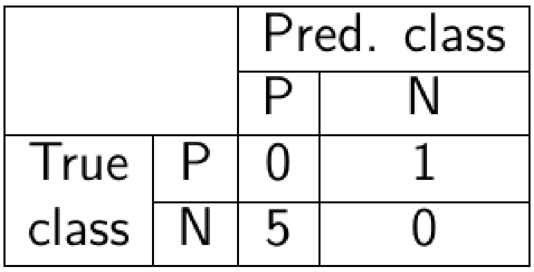
- -1 means total disagreement $\Rightarrow$ perfect swap between predictions and true labels

$$\sum_{i}^{n_{classes}} TP_i = 0$$

- 0 means random agreement

Cost of errors

Bad predictions imply a cost: a **cost matrix** is used to compare two classification models and take into account the different cost of the different errors (correct classifications do not imply any cost).



		Pred. class	
		P	N
True	P	0	1
class	N	5	0

Cost sensitive learning:

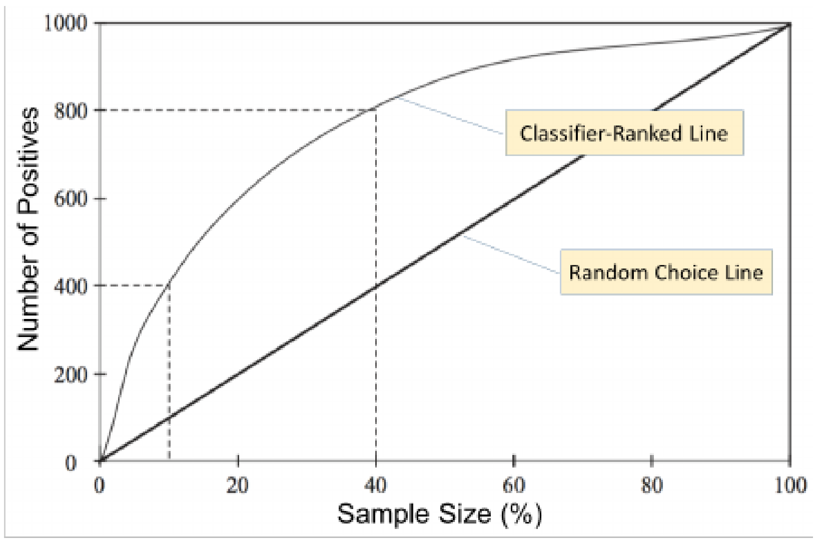
- Weighted cost of errors must be computed.
- Alternatives:
  - alterate proportion of classes in supervised data, duplicating examples with higher classification error, s.t. the classifier becomes more able to predict these classes;
  - add weights to instances (only in some learning schemes).

2.4. Evaluation of a probabilistic classifier

- Crisp values sometimes hide probabilities (e.g. leaves with non-sero minority counts).
- Probabilities can be converted to a crisp value with the following techniques:
  - set a threshold for the positive class (binary classification);
  - output class with maximum probability (multi-class classification).

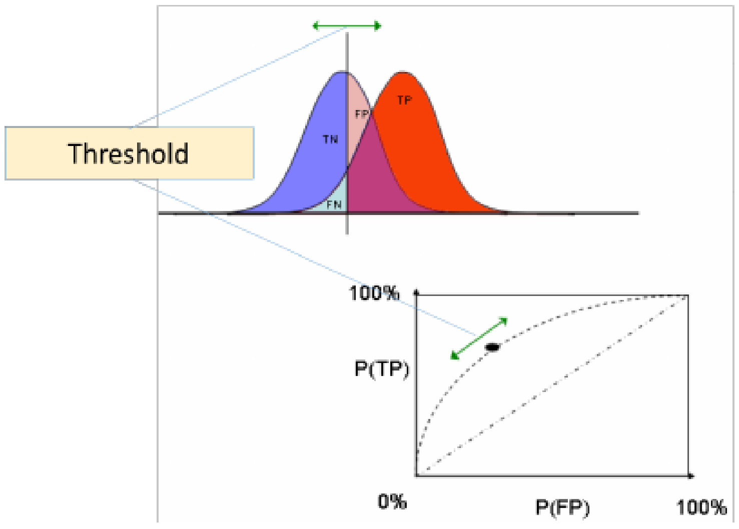
Lift Chart

1. Apply a probabilitic classification scheme to a dataset.
2. Sort classified elements by decreasing probability of positive class.
3. Make bi-dimensional chart with axes:
   - $x:$ sample size;
   - $y:$ number of positives in sample.
4. Straight line plots number of positives obtained with a random choice of a sample of test data.
5. Curve plots number of positives obtained drawing a fraction of test data with decreasing probability.
6. The larger the area between the two curves, the best the classification model.



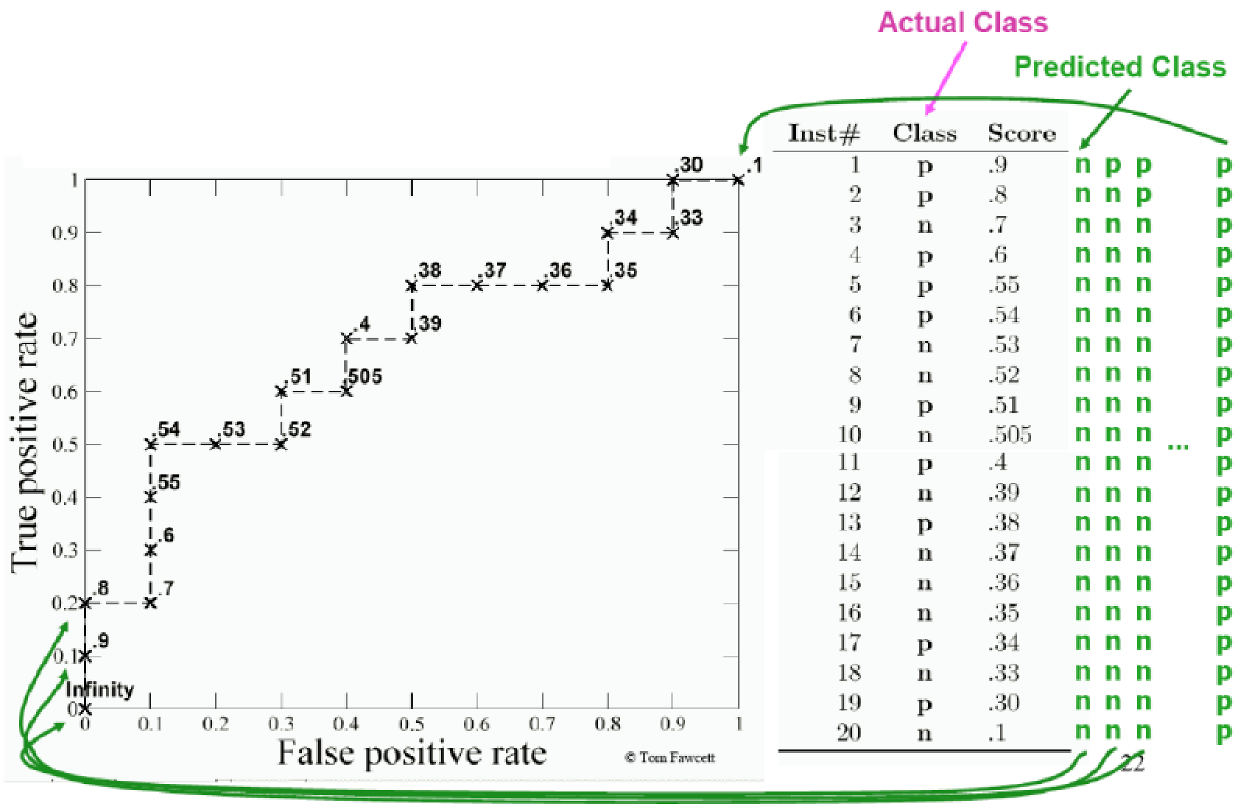
ROC (Receiver-Operator Characteristic) Curve

- Tradeoff between true positives and false positives.
- Noise alters the two levels according to a gaussian distribution.
- The positive/negative threshold must be chosen s.t. it maximizes the above tradeoff.
- Moving threshold towards right increases both the rate of TP and FP caught.
- The area between ROC curve and non-discrimination line is a quality of the classification model.



For a soft classifier:

- it can be converted into a crisp by choosing a threshold, which affects ratio of TP and FP;
- threshold steps allow to draw ROC curve:
  - sort test elements by decreasing positive probability:
  - set threshold to highest probability. set TP and FP to zero;
  - repeat the following:
    - update number of TP and FP with probability from threshold to 1;
    - draw a point in the curve;
    - move to next top probability of positive.



Inst#	Class	Score					
1	p	.9	n	p	p		p
2	p	.8	n	n	p		p
3	n	.7	n	n	n		p
4	p	.6	n	n	n		p
5	p	.55	n	n	n		p
6	p	.54	n	n	n		p
7	n	.53	n	n	n		p
8	n	.52	n	n	n		p
9	p	.51	n	n	n		p
10	n	.505	n	n	n	...	p
11	p	.4	n	n	n		p
12	n	.39	n	n	n		p
13	p	.38	n	n	n		p
14	n	.37	n	n	n		p
15	n	.36	n	n	n		p
16	n	.35	n	n	n		p
17	p	.34	n	n	n		p
18	n	.33	n	n	n		p
19	p	.30	n	n	n		p
20	n	.1	n	n	n		p

2.5. From binary classifiers to multi-class classification

Several classifiers generate a binary classification model $\Rightarrow$ two solutions:

- transform training algorithm and model (increases the size of the problem);
- combine results of a set of binary classifiers: **one-vs-one** and **one-vs-all**/one-vs-rest strategies.

One-vs-one (OVO):

- Consider all possible pairs of classes and generate a binary classifier for each one: $C \cdot (C-1)/2$ pairs.
- Each binary problem considers only the examples of the two selected classes.
- At prediction time a voting scheme is applied:
  - an unseen example is submitted to the $C \cdot (C-1)/2$ binary classifiers;
  - each winning class receives a $+1$;
  - class with highest number of $+1$ wins.

One-vs-all (OVA):

- Consider C binary problems where class c is a positive examples whereas all the others are negative.
- Build C binary classifiers.
- At prediction time a voting scheme is applied:
  - an unseen example is submitted to the C binary classifiers, obtaining a confidence score;
  - the confidences are combined and the class with highest global score is chosen.

Obs.: - OVO requires solving a higher number of problems (even if they're smaller);
- OVA tends to be intrinsically unbalanced (if classes are evenly distributed in examples, each classifier has a proportion positive to negative of 1 to $C-1$).

2.6. Ensemble methods

A set of **base classifiers** is trained, and the final prediction is obtained taking the votes of each of them.
Ensemble methods tend to perform better than a single classifier since, if the errors of each of them are uncorrelated, then the ensemble will be wrong only when the majority of them is wrong:

$$\epsilon_{ensemble} = \sum_{i=majority}^{total} \binom{total}{i} \epsilon^i (1-\epsilon)^{total-i} < \epsilon_{single}$$

Ensemble methods are useful if:

- base classifiers are independent;
- performance of base classifiers is better than random choice.

Methods for ensemble classifiers

By **manipulating training set data** are resampled according to sampling strategy:

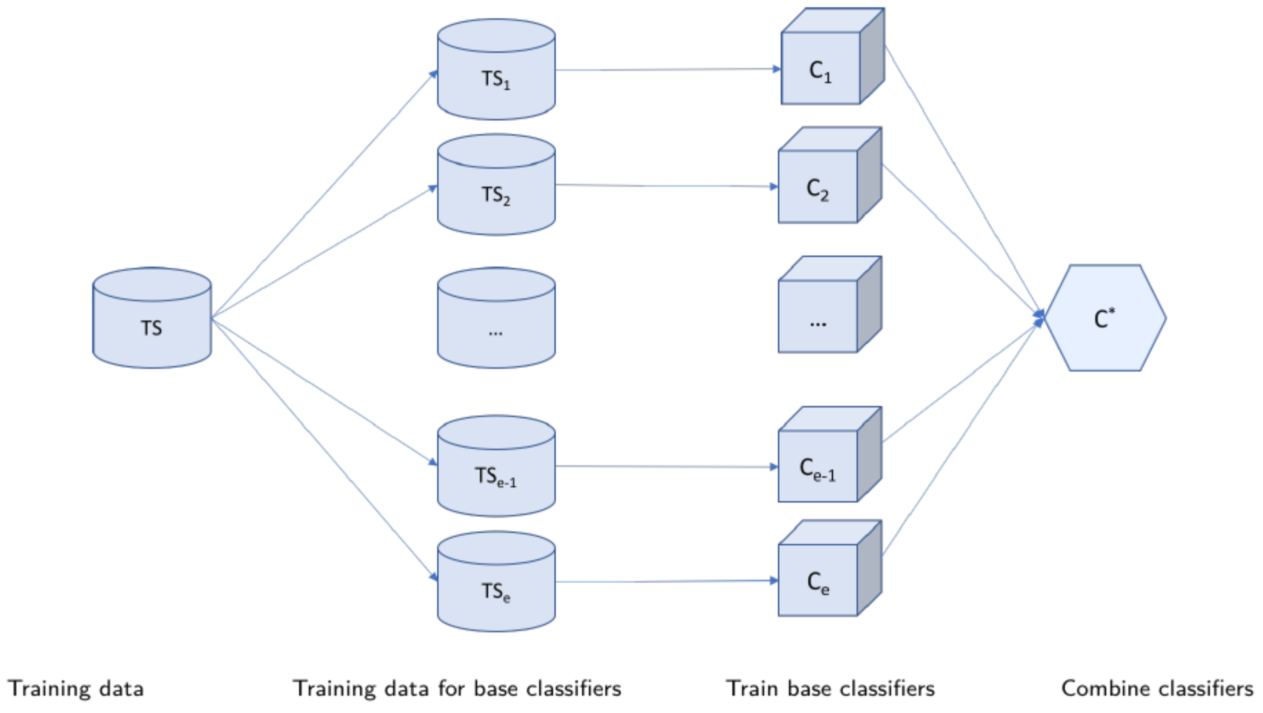
- **Bagging**: repeatedly samples with replacement according to uniform probability distribution.
- **Boosting**: iteratively changes distribution of training examples s.t. base classifiers focus on hard-to-classify examples.
- **Adaboost**: importance of each base classifier depends on its error rate.

By **manipulating input features** a subset of input features can be chosen either randomly or according to domain:

- **Random forest**: uses decision trees as base classifiers and frequently produces good results.

Manipulating class labels is useful when the number of classes is high:

- for each base classifier, randomly partition class labels into two subsets, A_1 and A_2, and re-label the dataset;
- train binary classifiers w.r.t. the two classes;
- at testing time when a subset is selected all the classes included receive a vote, and the class with the top score wins.



2.7. Statistical modeling - Naive Bayes Classifier

- Based on statistics, particularly on the Bayes' theorem.
- Considers the contribution of all the attributes.
- Assumes that each attribute is independent from the others, given the class.
- Estimates the probabilities with the frequencies.

The Bayes method

Given an hypothesis H and an evidence E that bears on that hypothesis:

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)} \qquad \text{(Bayes Theorem)}$$

- The hypothesis is the class c, the evidence is the tuple of values of the elements to be classified.
- The evidence can be splitted into pieces, one per attribute, and, if the attributes are independent inside each class, then:

$$P(c|E) = \frac{P(E_1|c) \cdot \ldots \cdot P(E_n|c) \cdot P(c)}{P(E)}$$

Naive Bayes Method:

- compute conditional probabilities from examples;
- apply theorem (denominator is the same for all classes, and thus it can be eliminated by normalization step).

Obs.: - the assumption of independence between attributes is simplistic (*naive*);
- if the value v of attribute d never appears in the elements of class c, then the

probability of the class for that evidence drops to zero: $P(d = v|c) = 0 \Rightarrow$ alternative solution needed.

Laplace smoothing:

- α : smoothing parameter (tipically 1)
- $af_{d=v_i,c}$: absolute frequency of value v_i in attribute d over class c
- V : number of distinct values in attribute v_i over the dataset
- af_c : absolute frequency of class c in the dataset

the smoothed frequency is:

$$sf_{d=v_i,c} = \frac{af_{d=v_i,c} + \alpha}{af_c + \alpha V}$$

With $\alpha = 0$ the formula gives the standard, unsmoothed result.

Obs.: - Higher values of α give more importance to the prior probabilities for the values of d w.r.t. the evidence given by the examples.

Missing values

They do not affect the model, thus it is not necessary to discard an instance with missing values:

- Test instance:
  - calculation of likelihood simply omits this attribute;
  - likelihood will be higher for all classes, but it is compensed by normalization.
- Train instance:
  - the record is simply not included in the frequency counts for that attribute;
  - descriptive statistics are based on the number of values that occur, rather than on the number of instances.

Numeric values

The method based on frequencies is inapplicable:

- it is assumed that values have a **Gaussian distribution**;
- mean μ and variance σ of the values of each numeric attribute **inside each class** are computed from examples (instead of fractions of counts);
- thus, for given attribute and class, the distribution is:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

In Naive Bayes:

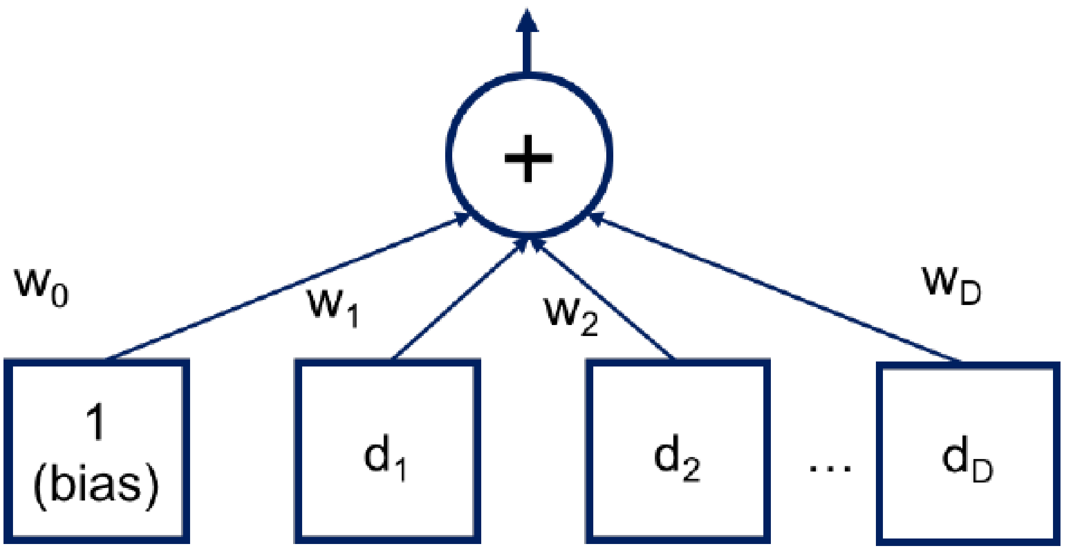
- on a continuous domain, probability of a variable assuming *exactly* a single real value is 0;
- a value of density function is the probability that the variable lies in a small interval around that value;
- values are rounded at some precision factor, which is the same for all classes;
- if numeric values are missing, mean and standard deviation are based only on those that are present.

Summary

- Clear semantics for learning probabilistic knowledge.
- Excellent results in many cases.
- Dramatic degradation when simplistic conditions are not met:
    - violation of *independence* between attributes;
    - violation of *gaussian distribution*.

2.8. Linear Classification with the Perceptron

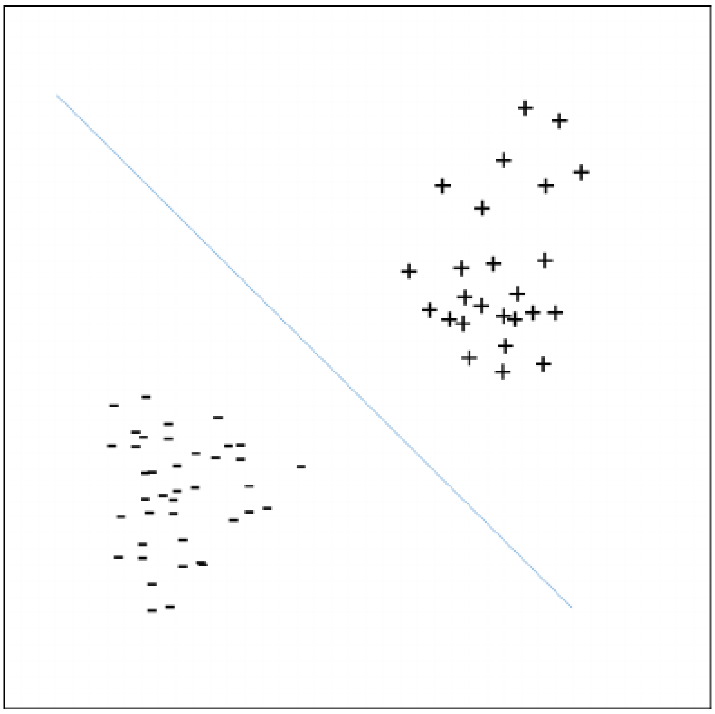
Often called also artificial neuron, it's a linear combination of weighted inputs.



Hyperplane:

- for a dataset with numeric attributes, the hyperplane divides the positive from the negatives;
- described by a set of weights $w_0, \dots, w_D$ in a linear equation on data attributes $e_0, \dots, e_D$ (fictious attribute $e_0 = 1$ is added s.t. hyperplane does not pass through origin);
- either **none** or **infinite** such hyperplanes:

$$w_0 e_0 + w_1 e_1 + \ldots + w_D e_D \begin{cases} > 0 \Rightarrow \text{positive} \\ < 0 \Rightarrow \text{negative} \end{cases}$$



Training the perceptron

1. Set all weights to zero.
2. For each training instance e incorrectly classified:
    - if its class is positive, add e data vector to vector of weights;
    - if its class is negative, subtract e data vector from vector of weights.
3. Repeat step 2 until all examples are correctly classified.

Convergence

Each change of weights moves the hyperplane towards misclassified instance: consider the equation after the weight change for a positive instance e which was classified as negative:

$$(w_0 + e_0) \cdot e_0 + \ldots + (w_D + e_D) \cdot e_D$$

the result of the equation is increased by a positive amount $e_0^2 + \ldots + e_D^2 \Rightarrow$ result will be **less negative**/positive (analogously for a negative instance classified as positive).

Obs.: corrections are incremental and can interfere with previous updates.

The algorithm converges **if the dataset is lineraly separable**, otherwise it does not terminate $\Rightarrow$ necessity to set an upper bound to iterations.

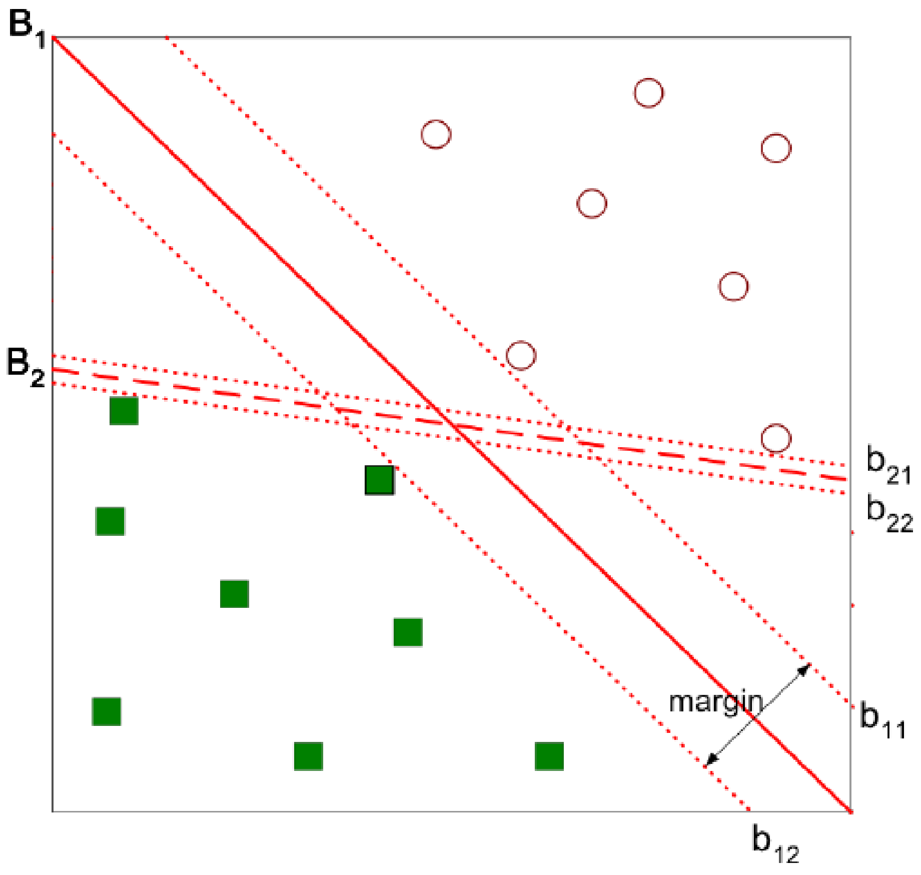
2.9. Support Vector Machines

If data are not linearly separable, one possibility would be to use non-linear models, but the method would soon become intractable and prone to overfitting.

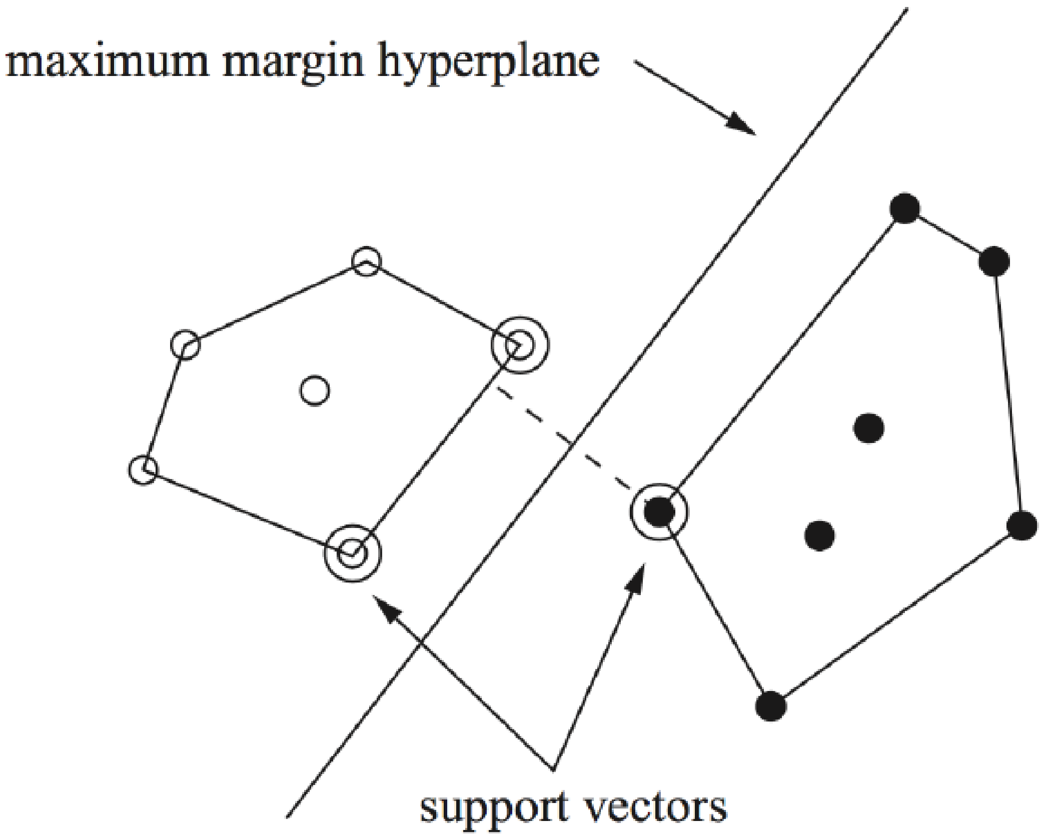
Key ideas:

- Computational learning theory.
- Efficient separability of non-linear functions that use **kernel functions**.
- Optimization rather than greedy search.
- Statistical learning: search modeled as a **function estimation** problem.

The Maximum Margin hyperplane



The linear perceptron accepts any hyperplane able to separate the classes of the training examples; the **maximum margin hyperplane** gives the **greatest separation** between classes.



- The *convex hull* of a set of points is the tighest enclosing convex polygon $\Rightarrow$ if the dataset is linearly separable the convex hulls of the classes do not intersect.
- The maximum margin hyperplane is as far as possible from both the hulls $\Rightarrow$ it is the perpendicular bisector of the shortest line connecting the hulls.
- In general a subset of the points, called **support vectors**, is sufficient to define the hull; they are the elements of the training set which would change.
- Finding support vectors and maximum margin hyperplane is a *constrained quadratic optimization* problem:

$$\max_{w_0,\ldots,w_D} M$$

$$\sum_{j=1}^{D} w_j^2 = 1$$

$$c_i(w_0 + w_1 x_{i1} + \ldots + w_D x_{iD}) > M, \forall i = 1, \ldots, N$$

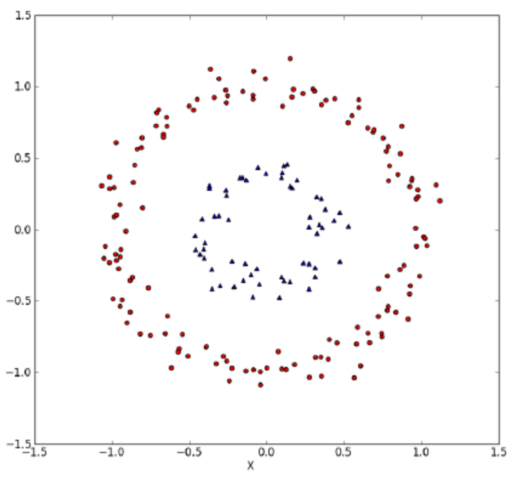
  where the class of the example i is either -1 or 1 and M is the margin.

Soft margin:

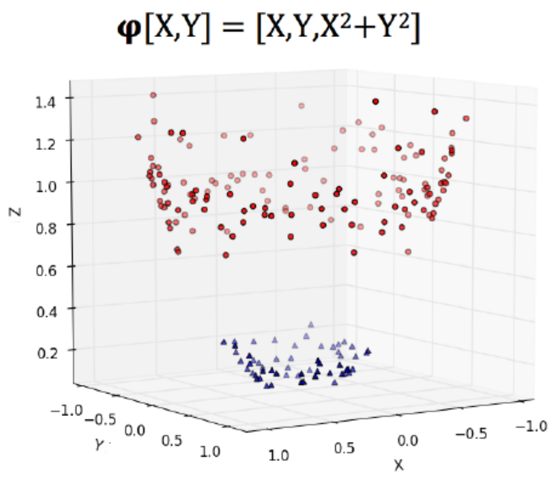
- it is quite common that a separating hyperplane does not exist, thus an hyperplane that *almost* separates the classes is found, and the examples generating very narrow margins are disregarded;
- greater robustness to individual observations;
- better classification of most of training observations;
- obtained by adding a constraint to the optimization problem expressed by a single numeric parameter C:
  - C controls the amount of overfitting;
  - C tuning is critical.

Non-linear class boundaries

- The support vector method avoids the problem of overfitting.
- The non-linearity of boundaries can be overcome with a **non-linear** mapping:
  - data are mapped into a new space, a *feature space*, s.t. a linear boundary in the feature space can correspond to a non-linear boundary in the original space;
  - feature space can have a number of dimension higher than the original.



Input space



Feature space

Kernel trick:

- Computation to separate hyperplane requires a series of dot products among training data vectors.
- Defining the mapping on the basis of a family of functions, the **kernel functions** (or **kernels**), the mapping does not need to be explicitly computed, and computation is done in input space $\Rightarrow$ it avoids an increase in complexity.
- Some kernel functions:

$$\begin{gathered}\text{linear} : \langle x, x' \rangle \\ \text{polynomial} : (\gamma \cdot \langle x, x' \rangle + r)^{dg} \\ \text{rbf} : \exp(-\gamma \cdot ||x - x'||^2) \\ \text{sigmoid} : \tanh(\langle x, x' \rangle + r)\end{gathered}$$

  where γ, dg, r are parameters specified in the documentation of learning tools.

SVM time complexity is mainly influenced by the efficiency of the optimization library: `libSVM` scales from $\mathcal{O}(D \cdot N^2)$ to $\mathcal{O}(D \cdot N^3)$, depending on effectiveness of data caching (which is **data dependent**, and reduced in case of sparse data).

Final remarks:

- Learning is slower w.r.t. simpler methods.
- Tuning necessary to set parameters.
- Results can be very accurate.

- Not affected by local minima.
- Do not use any notion of distance ⇒ not affected by the curse of dimensionality.
- SVM do not directly provide probability estimates, but a confidence score related to the distance from the hyperplane.

2.10. Neural Networks

It’s another method to overcome the limit of linear decision boundary:

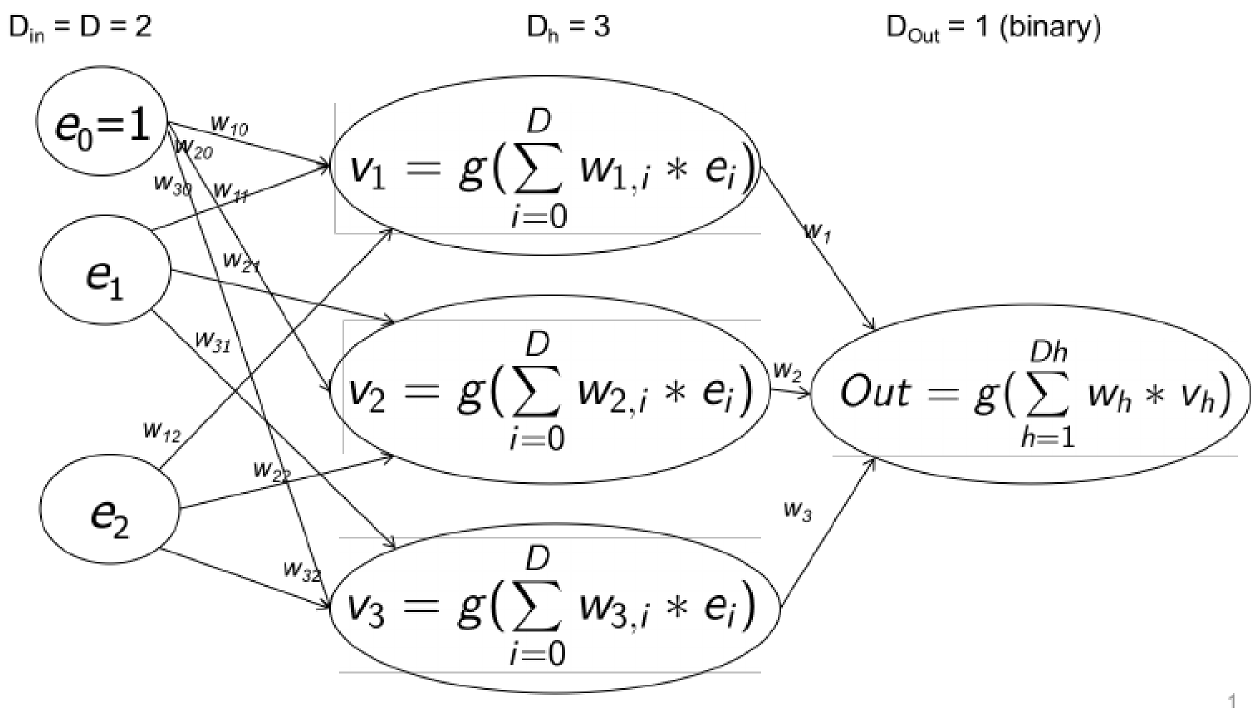
- arrange many perceptron-like elements in a hierarchical structure;
- inspired by complex interconnections of neurons in animal brain;
- a neuron is a signal processor with a **threshold**;
- signal transmission from one neuron to another is weighted (weights change over time due to learning).

Multi-layer perceptron

- Signals transmitted are modeled as real numbers.
- Threshold of biological systems is modeled as a mathematical function:
  - non-linear, continuous and differentiable, superiorly and inferiorly limited;
  - derivative can be expressed in terms of the function itself.
- Several functions available:
  - Sigmoid (*squashing function*): $\frac{1}{1+e^{-x}} \in]0,1[$
  - Arctangent
- The shape of the function can influence learning speed.

Feed-forward multi-layered network:

- inputs feed an **input layer**: one input node for each dimension in training set;
- input layer feeds (with weights) a **hidden layer**: number of nodes is a parameter of network;
- hidden layer feeds (with weights) an **ounput layer**:
  - one node for binary classification;
  - one node per class in case of multi-class classification.



$D_{in} = D = 2$
$D_h = 3$
$D_{Out} = 1$ (binary)
$e_0=1$
e_1
e_2
$v_1 = g(\sum_{i=0}^{D} w_{1,i} * e_i)$
$v_2 = g(\sum_{i=0}^{D} w_{2,i} * e_i)$
$v_3 = g(\sum_{i=0}^{D} w_{3,i} * e_i)$
$Out = g(\sum_{h=1}^{Dh} w_h * v_h)$
w_{10}
w_{20}
w_{30}
w_{11}
w_{21}
w_{31}
w_{12}
w_{22}
w_{32}
w_1
w_2
w_3
1

- $g(\cdot)$ is the transfer function of the node (e.g. sigmoid).
- Unitary input e_0 is added for dealing with the bias, as in the case of linear perceptron.
- Weights are for each edge connecting two nodes.
- Feed-forward:
  - edges connect a node in a layer only to a node in the following one (input to hidden and hidden to output);
  - each node of one layer is connected to every node of the following one;
  - signal flows from input to output without loops.

Training the Neural Network

1. Set all weights to random values.
2. For each training instance e:
   - feed network with e and compute output $nn(e)$;
   - compute weight corrections for $nn(e) - e_{out}$;
   - propagate back weight corrections;
3. Repeat until termination condition is satisfied.

The weights **encode** knowledge given by supervised examples (it’s not easily understandable since it’s a structured set of real values).

Obs.: convergence is not guaranteed.

Computing weight corrections

- Let $\mathbf{x}$ and y be the input vector and the desired output of a node.
- Let $\mathbf{w}$ be the input weight vector of a node.
- The error is:

$$E(\mathbf{w}) = \frac{1}{2}(y - \text{Transfer}(\mathbf{w}, \mathbf{x}))^2$$

- Move towards a local minimum of error:
  - follow the gradient;
  - compute partial derivatives of error as function of weights;
  - weight is changed by subtracting the partial derivative multiplied by a **learning rate** constant (it influences convergence speed and precision);
  - subtraction moves towards smaller errors;
  - derivatives of input weights of the nodes of a layer can be computed if the ones for the following layer are known.

Training algorithm

Learning modes:

- **Stochastic**:
  - each forward propagation is immediately followed by a weight update;
  - introduces some noise in gradient descent since gradient is computed from a single data point;
  - avoids local minimum;
  - good for online learning.
- **Batch**:
  - many propagations occur before updating weights, accumulating errors over samples within a batch;
  - faster and more stable descent towards local minimum since update is performed in direction of average error.

Repetitions:

- A learning round over all samples is called **epoch**.
- After each epoch the classification capability will be slightly improved and starting weights will be different.
- Several epochs are necessary.

Design choices:

- Structure of input and output layers is determined by domain.
- Number of nodes in input layer can be changed.
- Learning rate can be changed in different epochs:
  - in the beginning epochs a higher learning rate pushes faster towards desired directions;
  - in later epochs a lower learning rate pushes more precisely towards minimum.

Stop criteria:

- All weights update in current epoch have been small.
- Classification error rate goes below predefined target.
- Timeout condition reached.

Risks:

- Local minima.
- Overfitting (if network is too complex w.r.t. complexity of decision problem).

Regularization:

- Used to improve generalization capability of a model by modifying performance function (normally it is the sum of squared errors).
- In essence:
  - improvement of performance obtained by a **loss function** (i.e. sum of squared errors);
  - regularization corrects the loss function in order to **smooth** fitting to data;
  - amounts of regularization must be tuned.