Introduction to Machine Learning

Isabelle Guyon

Notes written by: Johann Leithon.

Introduction

The process of Machine Learning consist of having a big training data base, which is the input to some learning algorithm and comes out in a learning machine, then in the utilization phase you can input real data to the trained machine for having an answer. This is a very general framework (see figure 1):

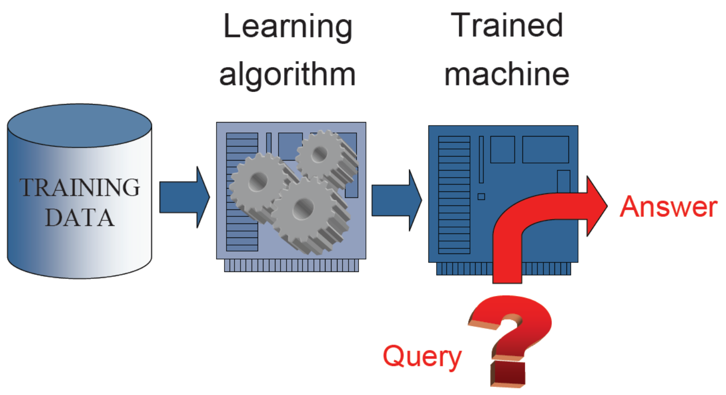


Figure 1: Machine Learning Framework

Machine Learning is useful to solve a large number of problems related to: classification (when you need to predict the outcome of a variable which is binary or categorical), Time Series Prediction (continous outcomes to predict), Regression (continous outcomes to predict) and Clustering (find groups in data).

Learning machines include: Linear Methods, Kernel Methods, Neural Networks and Decision Trees. These are the most used but there are people working in new ones.

Applications can be map alog two axis, the x axis is the number of inputs that you have and the y axis is the number of training examples that you have (see figure 2). In this two dimensional space you can have, map the complexity of problems, the few the number of training examples, the harder it is going to be to generate a learning machine, this is getting worst as the number of inputs increases compared to the number of training examples.

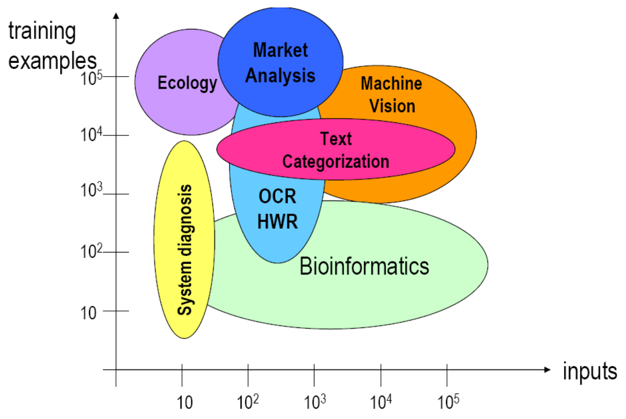


Figure 2: Application Map

Sometimes problems become complex because you have to much data, it is now a problem of computational complexity and not a problem of evaluating the parameters of the learning machine. For example: having a lot of example and a lot of inputs means a huge compute power.

Typical applications of Machine Learning include: Banking, Telecom and Retail, you are interested about identifying prospective costumers, dissatisfied costumers, good costumers, bad payers, in order to obtain: more effective advertising, less credit risk, fewer fraud and decrease the churn rate (churn means for an employee to switch to one company to another). Biomedical, Biometrics areas, you need to screen people for risk of certain desease, to make a diagnosis or to predict the outcome of a treatment (prognosis), you are interested also about discovering new drugs. Security Area: Machine Learning is needed for face recognition, iris verification, fingerprinting, signature, and also DNA fingerprinting. In computation and Internet, people have been using Machine Learning for designing computer interfaces (troubleshooting wizards, handwriting and speech recognition interfaces, and brain waves - for handing computer machines through your brain). Internet applications include: hit ranking, spam filtering, text categorization, text translation and recommendation.

Guyon describes challenges for NIPS 2003 and WCCI 2006 which consist of perfoming some experiments (classification tasks) with ten datasets corresponding to different areas for having a wide spectrum of difficulty in terms of inputs and the number of training examples. Results showed how different learning machines perform in different datasets (see figure 3)

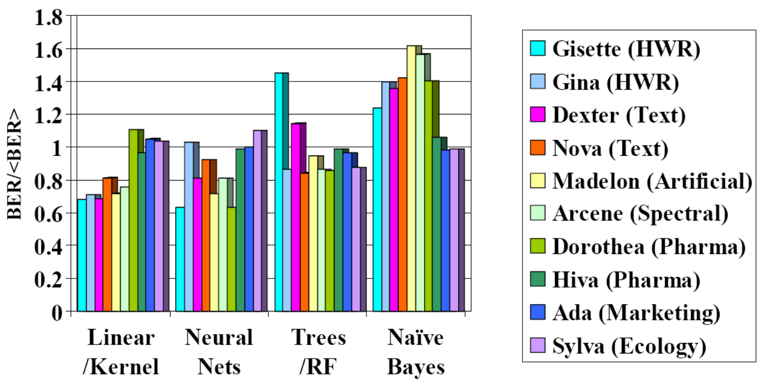


Figure 3: Challenge Winning Methods

Figure (3) groups learning machines into four categories: Linear and Kernel methods, Neural Nets, Decision Trees and Näive Bayes. As it can be seen, in terms of relative Balance Error Rate which is the Balance Error Rate over the average Balance Error Rate of the participants, the best performers were doing very well, the Linear and Kernel methods for the first two datasets but not better than other methods for the last four datasets. Neural networks can do very well in some datasets but they have problems in others. Decision trees perform almost the same as Neural Networks, very well in some datasets but not al all. Näive Bayes, which is a very simple method making independents assumptions about the inputs can do well in some datasets but generally, they performed worst. One of the tasks in ML is to predict, to have on time which learning machines is going to perform well before you have seen data.

Convention

Data are going to be represented as a matrix, the lines represent examples and the columns represent feautures or variables. For example, it could be patients (in lines) and for each patient you will have recorded the age, the weight, the number of children, etc.

In terms of learning machines, the weights which are associated with the columns, are used to construct the decision function. When those weights are associated with the lines, we called them the alphas. There is a special column separated from the others, which is the target, quantity that you want to predict (see figure 4)

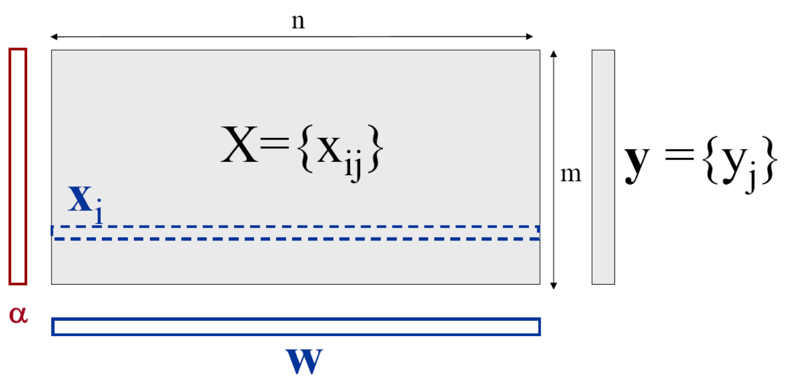


Figure 4: Conventions

Unsupervised Learning Problems

You are dealing with finding a structure in data. If you want to see any structures in data you have to perform clustering, then randomize data and finally clustering again in order to realize a structure.

Supervised Learning Problems

We want to predict an extra column y. This class is about supervised learning and building functions of x, x, being the inputs that allow us to make predictions about the target y, so f(x) should be as close as posible to the y that we want to predict.

Linear Models

The simplest way to build f(x) is to make a dot product between x, which is a vector (one line of your matrix, for example), and the weight vector w. This is equivalent to making a weighted sum of the input features weighted by some coefficients, to build the output.The decision is going to be built as some kind of voting, the weights are the voting power of each input feature, see equation (1):

$$f(x) = w \cdot x + b = \sum_{j=1:n} w_j x_j + b \tag{1}$$

When the inputs and the parameters of the function are not linear, you obtain a family of functions which is called Perceptron, see equation (2).

$$f(x) = w \cdot \phi(x) + b = \sum_{j=1:n} w_j \phi_j(x) + b \tag{2}$$

$\phi(x)$is a transform of the input in a feature space. $\phi(x)$ are features in a transformed space, for example the product of two features. Suppose it is easier to have the product of the age and weight, you have another feature which can be $\phi(x)$. You can have many much complex operations (pieces of images, lines, feature extractions).

In the same familiar methods, there are kernel methods, in this case you replace the weights w by alphas because now instead of weighting the features of the matrix (columns), you are weighting the lines of the matrix. Replace the $\phi(x)$ with some kind of basic functions. The difference is that we are using the training examples and copying the training examples with the new examples, that is under study, K is a similarity measure between the training example and the new unknown example. There is also a weighted sum but the features are of special kind, the features are similarities with the original example.

Artificial Neurons

In the 80's, people were trying to imitate the brain (the way people understand how the brain functions) to get better predictions (by the machine). The basic processing unit of the brain is the neuron, it has been model coarsy by 1943, this is the model most used in ML, even though there have been a lot of refinments. This is also a linear model, the only difference is that you have a so called activation function at the output so the unit makes a weighted sum of the inputs and then this wighted sum goes to what the people call activation function, so the output is bounded by two values (for example, for making a decision). See figure ().

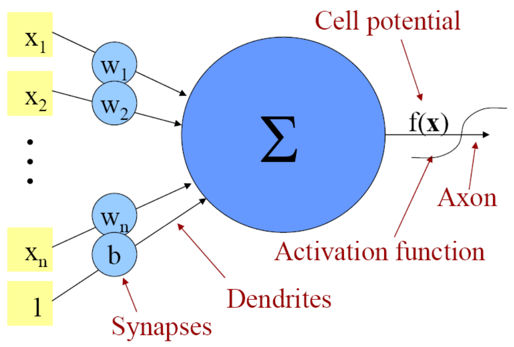


Figure 5: Neural Network

Linear Decision Boundary

Consist of making a decision in order to separate two classes using a linear decision boundary. 2 Classes, 2D it is a line, in 3D it is a plane, in general for higher dimensions it is a hyperplane which dimension is the dimension of the features minus one, see figure (6).

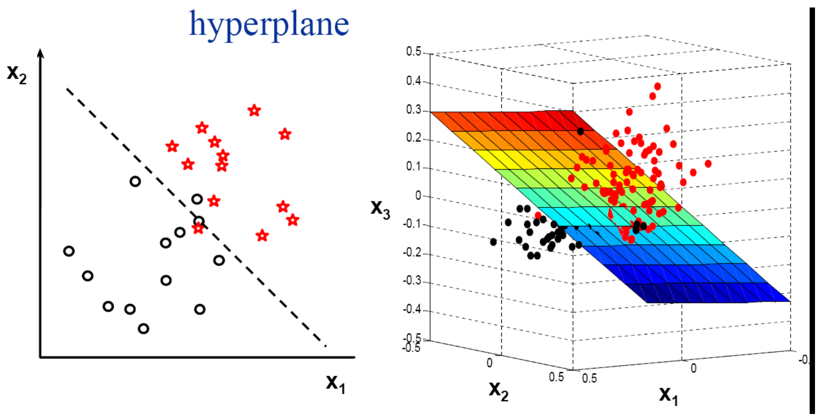


Figure 6: Linear Decision Boundary

Perceptron

The second simplest learning machine you can imagine. It was invented by Rosenblatt in 1977. You replace the inputs by the transformed inputs, the $\phi(x)$, which is a function computed eventually from the other original inputs, you obtain a new vector of dimension N, when the original number of features was smaller. f(x) is going to be a continous value to predict your outcome See figure (7). The advantage of Perceptron is that it has a non linear decision boundary and lets you to separate classes which are not easy separated using a simple linear decision boundary. See figure ()

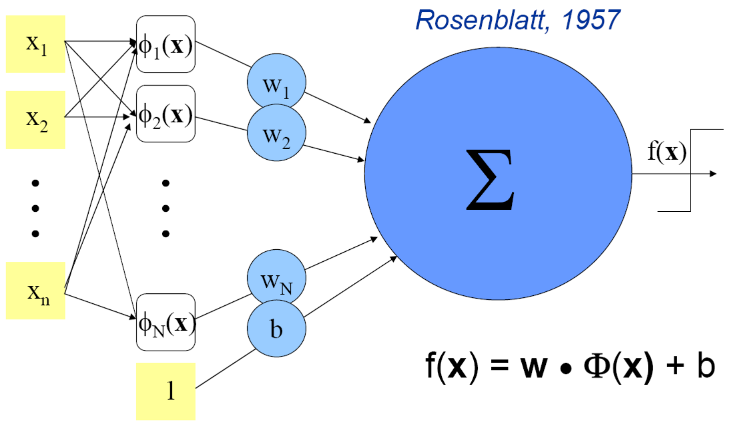


Figure 7: The Perceptron

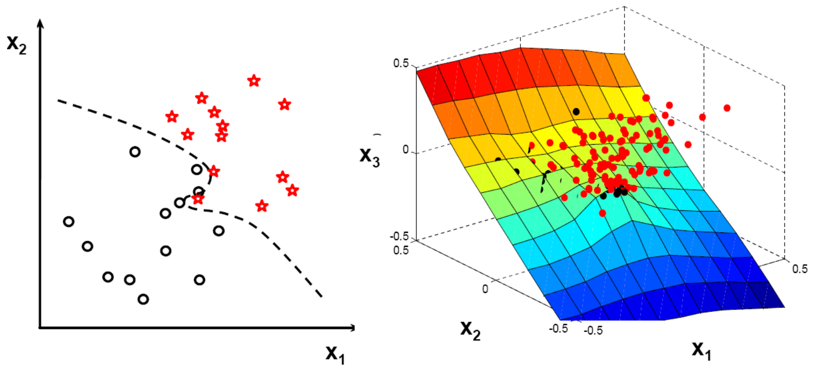


Figure 8: Non Linear Decision Boundary, 2D, 3D.

Kernel Methods

Special case of linear methods, they are similar to perceptron except that $\phi(x)$ functions are replaced by $k(x_i, x)$ functions. Graphically, all the inputs of (new patient), are being compared with the inputs your training examples, x_1 represent the vector of the features of the first patient, in the simplest case we compute the dot product between vector x (unknown patient) with x_1(representing the patient 1). This method is similar to the Nearest Neighbor Method. See figure ().

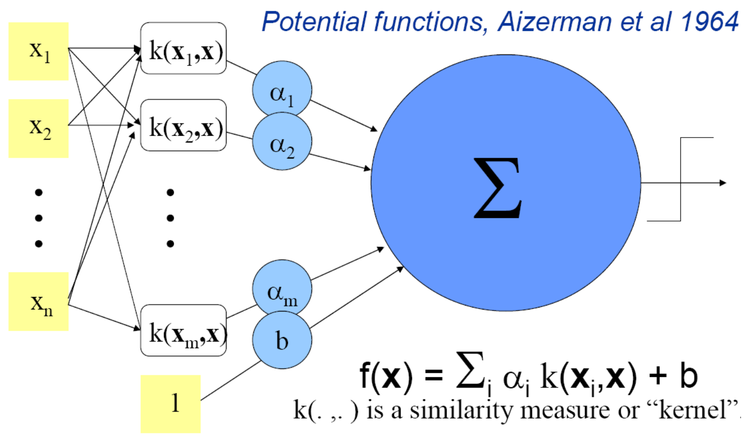

Potential functions, Aizerman et al 1964
x_1
x_2
x_n
$k(\mathbf{x}_1,\mathbf{x})$
$k(\mathbf{x}_2,\mathbf{x})$
$k(\mathbf{x}_m,\mathbf{x})$
1
α_1
α_2
α_m
b
Σ
$f(\mathbf{x}) = \sum_i \alpha_i \, k(\mathbf{x}_i,\mathbf{x}) + b$
k(. ,.) is a similarity measure or "kernel".

Figure 9: Kernel Method

Hebb's Rule

If there is activity between one neuron and the next one, this reinforce the synapse, while if there is not activity between two consecutive neurons, this decrease the synapse, see equation (3).

$$w_j \leftarrow w_j + y_i x_{ij} \tag{3}$$

If one uses ± 1outputs, the term $y_i x_{ij}$ in (3) will be positive only if input and output are both positive or negative.

Kernel “Trick”

Hebb’s rule is useful to understand the Kernel “Trick”. This is a basis for a lot of kernel methods algorithms. It establishes that there is a correspondece between the perceptron representation and the kernel based representation. A perceptron and a kernel can represent the same decision function. First, consider the Hebb’s rule applied to the perceptron, the weight vector is going to be a weighted sum of the $y_i \phi(x_i)$:

$$w = \sum_i y_i \phi(x_i)$$

$$f(x) = w \cdot \phi(x) = \sum_i y_i \phi(x_i) \cdot \phi(x)$$

For the kernel based expression, define the product:

$$k(x_i, x) = \phi(x_i) \cdot \phi(x)$$

We finally find an equivalent expression for the kernel based method:

$$f(x) = \sum_i y_i k(x_i, x)$$

In general expressions:

$$f(x) = \sum_i \alpha_i k(x_i, x)$$

$$k(x_i, x) = \phi(x_i) \cdot \phi(x)$$

Which has a dual form in a Perceptron based representation:

$$f(x) = w \cdot \phi(x)$$

$$w = \sum_i \alpha_i \phi(x_i)$$

Clearly, it can be shown through simple algebra.

What is a Kernel?

A kernel is a similarity measure, it is a dot product in some feature space and sometimes we don't know what the space is. Examples:

$$k(s,t) = e^{-\frac{||s-t||^2}{\sigma^2}}$$

$$k(s,t) = (s \cdot t)^q$$

The first example is called the Gaussian Kernel and just computes the euclidean distance between two vectors. The second example is called the Polynomial Kernel and allows you to perform a polinomial function.

Multi-Layer Perceptron

You combine many neurons and stack them in a structure where one layer of neurons is the input to another layer, see figure (10). Originally, this was design to solve the so called chessboard problem which establishes that when there is two classes to be arranged in a chessboard, you can not classified them using a linear decision boundary, but if you have two, then you can make a perfect separation, which is related to stack several layers to produce several decision boundaries, see figure ().

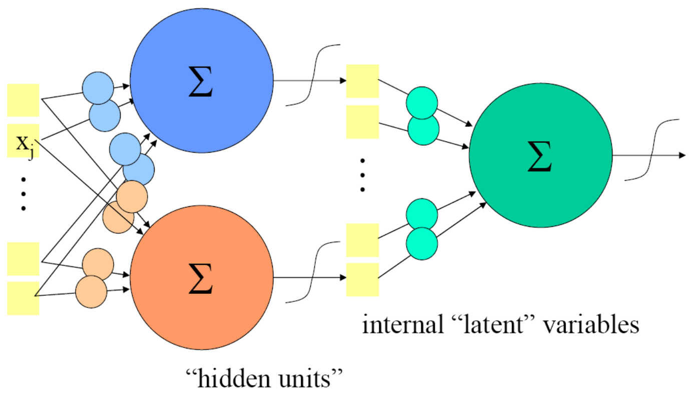


Figure 10: Multi-Layer Perceptron

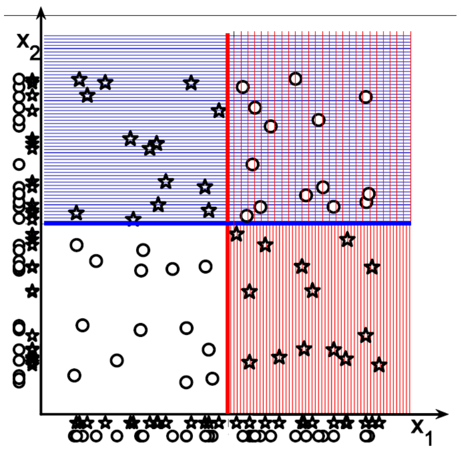


Figure 11: Chessboard Problem

Tree Classifiers

They are going to try progresively to separate examples along axes of the feature space, one feature of one input and in the other side the other. Objective: to separate two classes, they could be related to two populations. You will still have quiet bit of error. “At each step, choose the feature that “reduces entropy”most. Work towards “node purity”.” See figure (12).

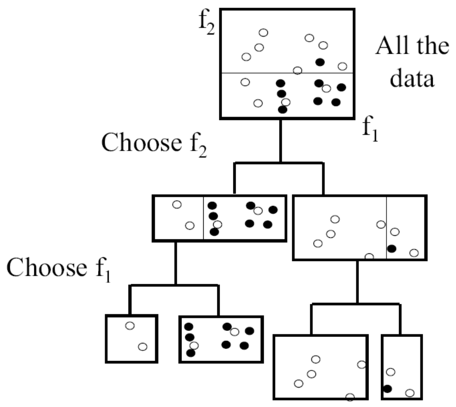


Figure 12: Decision Trees

Fit/Robustness Tradeoff

If you have a complex decision boundary, new data probably will not fit as training examples, while, if your decision boundary is simple, the probability of new data to fit correctly is bigger. Figure () shows how new data fit better when a simple line is used as a decision boundary in comparison with new data in a complex decision boundary.

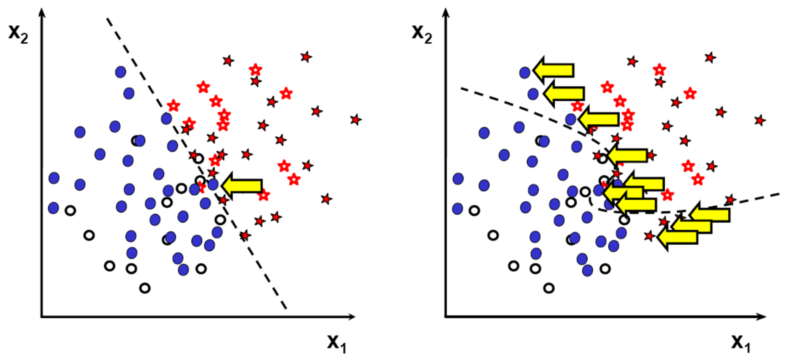


Figure 13: Fitting and Decision Boundaries

Performance Evaluation

We can put more weight on the error of one class than in the other. For example, false negatives are more serious than false positives when regarding to diseases. This discrimination can be perfoming simple by sliding the decision boundary toward the side which represents worst error in choosing it, the idea is to reduce its area in order to reduce higher weighted errors in classification. Otherwise one may shift the decision boundary in the other direction, it depends on the class which represent the higher weighted errors in choosing, see figure (14).

In general, you can vary the bias value and monitor the tradeoff between the error making in the positive class and the error making in the negative class, this is the so called ROC curve.

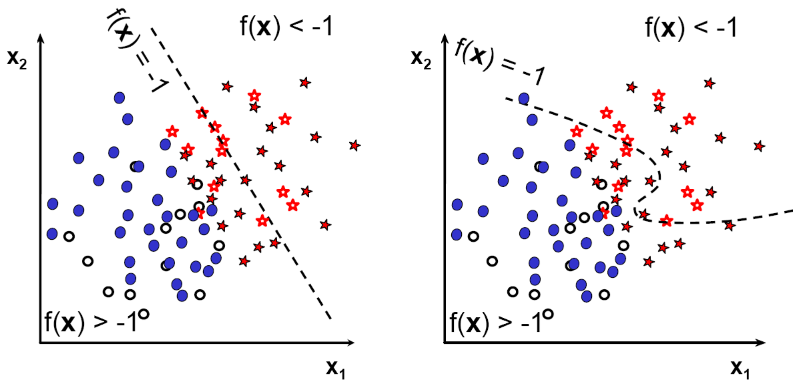


Figure 14: Bias Displacement

ROC Curve

The ROC curve plots the positive class success rate (also called the hit rate or sensitivity) versus the one minus the negative class success rate (also called false alarm rate, or one minus the specificity). For a given threshold on f(x), you get a point on the ROC curve, see figure (15).

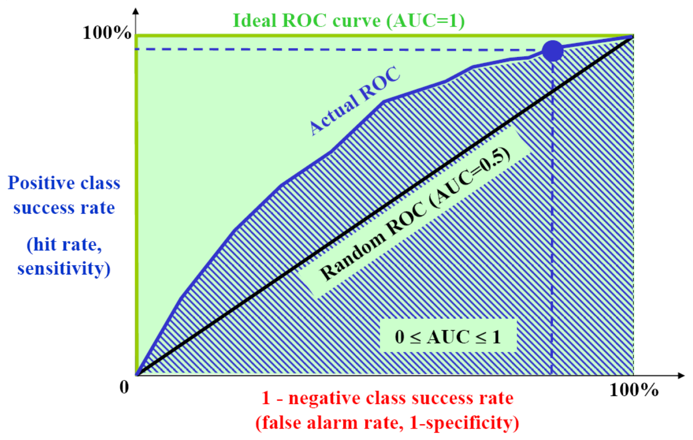


Figure 15: ROC Curve

The area under that curve is the better from a point of view of the classification accuracy, this is one way people meassure classification accuracy which also has to be independent on the particular choice of the bias that you make. The idea of ROC curve is this one: it has an area of one it's making more error on the negative class and you move the bias towards the positive class. The random case (when you're making decision at random for the positive and negative classes) gives an area under the curve of 0.5.

Sometimes people meassure using the Lift Curve. The Lift Curve plots the fraction of good customers versus the fraction of customers selected, see figure (16).

If you have two classes and making prediction, then you'll have several types of errors: the false positives (when you classify an item as positive and this is actually negative) and the false negatives (when you classify an item as negative being positive). Then you have the correct classification the true negative and true positive. True negative plus false positive is the total number of negative examples, and the sum of the false negative and the true positive is the total number of positive examples. Then you have the rejected examples which are the total number of negative examples classified (true negative plus false negative). You also have the selected examples, the ones that you have classified as positive (false and true). Using these quantities you can compute virtually all the measurements that people used to assess performance of classifiers, see figure (17).

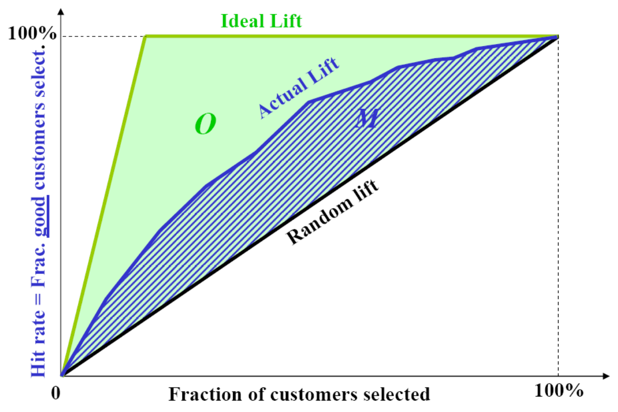
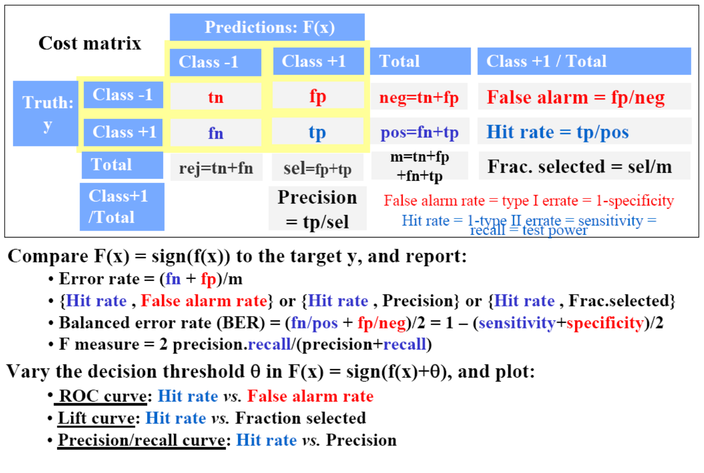


Figure 16: Lift Curve



Cost matrix		Predictions: F(x)			
		Class -1	Class +1	Total	Class +1 / Total
Truth: y	Class -1	tn	fp	neg=tn+fp	False alarm = fp/neg
	Class +1	fn	tp	pos=fn+tp	Hit rate = tp/pos
	Total	rej=tn+fn	sel=fp+tp	m=tn+fp +fn+tp	Frac. selected = sel/m
	Class+1 /Total		Precision = tp/sel	False alarm rate = type I errate = 1-specificity Hit rate = 1-type II errate = sensitivity = recall = test power	

Compare F(x) = sign(f(x)) to the target y, and report:

- **Error rate = (fn + fp)/m**
- **{Hit rate , False alarm rate} or {Hit rate , Precision} or {Hit rate , Frac.selected}**
- **Balanced error rate (BER) = (fn/pos + fp/neg)/2 = 1 – (sensitivity+specificity)/2**
- **F measure = 2 precision.recall/(precision+recall)**

Vary the decision threshold θ in F(x) = sign(f(x)+θ), and plot:

- **ROC curve: Hit rate *vs.* False alarm rate**
- **Lift curve: Hit rate *vs.* Fraction selected**
- **Precision/recall curve: Hit rate *vs.* Precision**

Figure 17: Performance Assessment

Risk Function

Risk function is a function of the parameters of the learning machine assessing how much it is expected to fail in a given tasks. For examples: for classification problems, the error rate (or 1 minus the area under the ROC curve) is going to be a function of risk. For regression, people use most the mean square error, see equation (4). In this case you are taking the output of the learning machine and the desired output making the difference square and averaging it.

$$\frac{1}{m} \sum_{i=1:m} (f(x_i) - y_i)^2 \tag{4}$$

Training

A risk function is needed for training. When you are training you have to define how are you going to measure and how your learning machine is going to perform, then you want to optimize it. You want to minimize the risk function and that can be done with a variety of methods including gradient descent, mathematical programming, simulated annealing, genetic algorithms etc.

Summary

With linear threshold units ("neurons") we can build many different kinds of Learning Machines including:

- Linear discriminant (including Naïve Bayes -the special case-)
- Kernel methods (which are linear in the parameters, not necessarily in the input components)
- Neural networks which are nonlinear both in the parameteres and the input components.
- Decision trees that have also elementary nodes that make a simple linear decision.

The architectural of the learning machines, also called hyper-parameters may include:

- The choice of basis functions ϕ (features), which ϕ function you are using in the perceptron or which kernel you're using in the kernel.
- The kernel
- The number of units in the case of a neural network.

Learning means fitting the parameters that are the weights and also the hyper parameters. One has to be aware of the fit vs.robustness tradeoff, it is not necessary best to obtain a decision boundary that learns exactly well the training example but what one needs to care about is how well it is going to be well on the feature examples that we haven't seen in the training data. In that respect, using a linear decision boundary sometimes is better that using a complex decision boundary.

All figures from [1].

Bibliography

[1] Guyon, Isabelle, Introduction to Machine Learning, Slides and Videolecture. Available at: videolectures.net.