Machine Learning notes

Luigi Tarasi

October 8, 2025

Contents

0.1 ML purposes

The ML studies and proposes methods to build (infer) dependencies/functions/hypothesis from examples of observed data:

- that fits the known examples;
- able to generalize, with reasonable accuracy for new data
- considering the expressiveness and algorithmic compLexity of the models and learning algorithms

Chapter 1

Data

1.1 Encoding

Flat case:

- Numerical encoding for categories: e.g. 0/1 or -1/+1 for 2 classes (or for multi-classes the one-hot-encoding);
- Structured: sequences (lists), trees, graphs, tables...

1.2 Further terminologies

- Noise: addition of external factors to the stream of (target) information (signal); due to randomness in the measurements, not due to the underlying law. (E.g. Gaussian noise)
- Outliers: unusual data values that are not consistent with most observations (e.g. due to abnormal measurements errors). Removing in preprocessing.
- Feature selection: selections of a small number of informative features. It can provide an optimal input representation for a learning problem.

1.3 Tasks

The task defines the purpose of the application:

- Predictive: function approximation (classification, regression).
- Descriptive: find subsets or groups of unclassified data (clusyer analysis, association rules)

1.3.1 Supervised Learning

Given: Training examples as $< input, output >=< \vec{x}, d >$ (labelled examples) for an unkown function f known only at the given points of examples

Find: a *good* approximation of f (a hypothesis h that can be used for prediction on unseen data $\vec{x}'$, i.e. that is able to generalize)

Target: d (or t or y) a categorical or numerical label;

- Classification: discrete value outputs:

$$f(\vec{x}) \in \{1, 2, ..., K\} \text{ classes}$$

- Regression: real continuous output values (approximate a real-valued target function, in R or R^k)

1.3.2 Unsupervised Learning

No teacher!

- TR (training set) = set of unlabelled data $< \vec{x} >$
- Finding natural groupings in a set of data (clustering, dimensionality reduction / visualization / preprocessing, modelling the data density)

1.3.3 Classification

Patterns (feature vectors) are seen as members of a class and the goal is to assing the patterns observed classes (labels)

- $f(\vec{x})$ returns the correct class for $\vec{x}$
- 2 classes: $f(\vec{x})$ is a Boolean function: binary classification, concept learning (T/F or 0/1 or -1/+1 or neg/pos)
- > 2: multi-class problem $(C_1, C_2, C_3, ..., C_K)$

It can be also viewed as the allocation of the input space in decision regions 0/1 (e.g. linear separators)

1.3.4 Regression

Process of estimating of a real-valued function on the basis of finite set of noisy samples.

Chapter 2

Models

The model defines the class of functions that the learning machine can implement: it's needed to capture/describe relationships among the data (on basis of the task) by a "language". This language is related to the representation used to get knowledge.

- Linear models: representation of H defines a continuously parametrized space of potential hypothesis, each assignment of w is a different one, e.g.:

  $$\text{Binary classifier: } h(\vec{x}) = sign(\vec{w}^T\vec{x} + w_0)$$

  $$\text{Linear regression: } h_w(\vec{x}) = \vec{w}^T\vec{x}$$

- Symbolic rules: hypothesis space is based on discrete representations, different rules are possible, eg.

  $$\text{Binary classifier: if } (x_1 = 0) \text{ and } (x_2 = 1) \text{ then } h(\vec{x}) = 1 \text{ else } h(\vec{x}) = 0$$

- Probabilistic models: estimate $p(x, y)$
- K-nearest neighbor regression: predict mean y value of nearest neighbors (memory-based)

Chapter 3

Learning Algorithm

Basing on data, task and model, search through the hypothesis space H of the best hypothesis. H may not concide with the set of all possible functions and the search can not be exhaustive: we need to make assumptions $\rightarrow$ inductive bias.

3.1 The role of the inductive bias

In order to set up a model and a learning algorithm we can make assumptions (about the nature of the target function) concerning either

- constraints in the model (in the hypothesis space H, due to the set of hypothesis that we can express or consider) (LANGUAGE BIAS)
- constraints or preferences in learning algorithms/search strategy (SEARCH BIAS)
- both

Such assumptions are strictly needed to obtain a useful model for the ML aims (model with generalization capabilities)

3.2 Find the Version Space (VS)

We call the version space $VS_{H,TR}$ with respect to hypothesis space H and training set TR, the subset of hypothesis from H consistent with all training examples.

- h is consistent with the TR if $h(\vec{x}) = d(\vec{x})$ for each training example $< \vec{x}, d(\vec{x}) >$ in TR.

3.3 Unbiased Learner

An unbiased learner is unable to generalize. A learner that makes no prior assumptions regarding the identity of the target function/concept has no rational

basis for classifying any unseen instances. Bias not only is assumed for efficiency, it is also needed for generalization capability. However, it does not tell us which one is the best solution for generalization yet.

3.4 Tasks and Loss

Using a "inner" loss function/measure to judge the "goodness" of approximation. High value of loss means poor approximation.

$$L(h_{\vec{w}}(\vec{x}), d)$$

The Error (or Risk or Loss) is an expected value of this L over the set of samples:

$$Loss(h_{\vec{w}}) = E(\vec{w}) = \frac{1}{l}\sum_{p=1}^{l} L(h_{\vec{w}}(\vec{x}_p), d_p)$$

L can change for different tasks.

3.4.1 Regression

Predicting a numerical value.

- Output: $d_p = f(\vec{x}_p) + e$ (real value function + random error);
- H: a set of real-valued functions;
- Loss function L: the squared error

$$L(h_{\vec{w}}(\vec{x}_p), d_p) = (d_p - h_{\vec{w}}(\vec{x}_p))^2$$

- The mean over the dataset provide the MSE mean square error.

3.4.2 Classification

Classification of data into discrete classes.

- Output: e.g. $0, 1$;
- H: a set of indicator functions;
- Loss function L: the 0/1 Loss

$$L(h_{\vec{w}}(\vec{x}_p), d_p) = \begin{cases} 0 & if \quad h_{\vec{w}}(\vec{x}_p) = d_p \\ 1 & otherwise \end{cases}$$

- The mean over the dataset provide the number/percentage of misclassified patterns and so the accuracy.

3.5 Generalization

Generalization error measures how accurately the model predicts over novel samples data. Error/loss computed over new data.

- Learning phase (training, fitting): build the model from known data - TR and bias.
- Predictive or test phase: apply the model to new examples, making evaluation of the generalization capability of our predictive hypothesis.
- Theory: e.g. statistical learning theory (SLT, Vapnik) answers under what mathematical conditions a model is able to generalize.

3.6 Validation

Evaluation of performances for ML systems = generalization/predictive accuracy evaluation. Validation techniques to:

- evaluate (model assessment).
- manage the generalization capability (model selection).

3.7 Generalization issues

Overfitting: a learner overfits the data if it outpus a hypothesis $h(\cdot) \in H$ having true/generalization error (risk) R and empirical (training) error E, but there is another $h'(\cdot) \in H$ having $E' > E$ and $R' < R$ so that h' is a better choice, despite a worst fitting.

Critical aspect: accuracy/performance estimation both theoretical and empirical.

Chapter 4

Complexity control

4.1 Complexity on a case of study

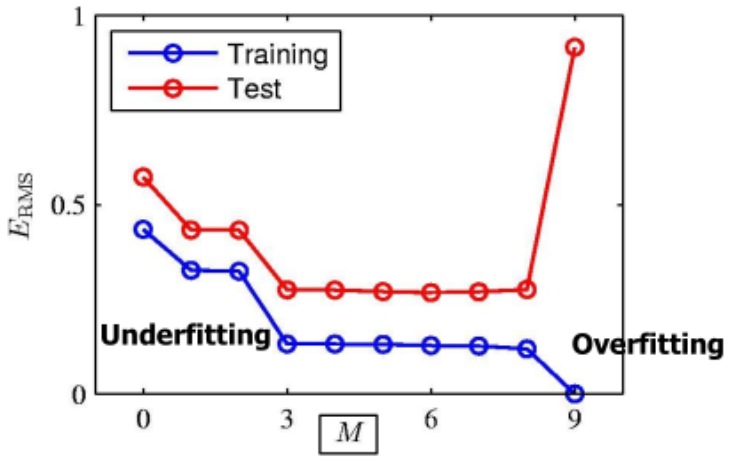
In an example of parametric model for regression the complexity of the hypothesis increases with the degree M of the polynomials of the set of functions.

l = number of examples (patterns).

$$h_{\vec{w}}(x) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

$$E(\vec{w}) = \sum_{p=1}^{l} (y_p - h_{\vec{w}}(x_p))^2$$

Minimize $E(\vec{w})$ and find this way the best $\vec{w}*$



Root-Mean-Square (RMS) Error: $E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^\star)/l}$

Where $E(\mathbf{w}^*)$ is the error for the trained model

4.2 Formal setting

Approximate unkown $f(\vec{x})$, d is the target = true f + noise. Given value of d, the probability distribution $P(\vec{x}, d)$ and a loss or cost function e.g.

$$L(h(\vec{x}, d)) = (d - h(\vec{x}))^2$$

We ask to search in H to minimize risk function

$$R = \int L(d, h(\vec{x}))dP(\vec{x}, d)$$

But we have only the finite data set TR = $(\vec{x}_p, d_p)$ $p = 1, \ldots, l$. To seaerch h: minimize empirical risk (training error E), finding the best values for the model free parameters

$$R_{emp}(h, TR) = \frac{1}{l}\sum_{p=1}^{l}(d_p - h(\vec{x}_p))^2$$

4.3 Vapnik-Chervonenkis-dim and Statistical Learning Theory (SLT)

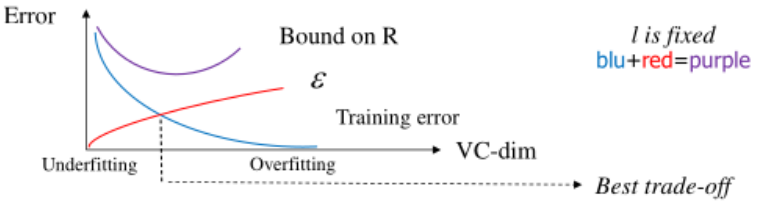
A general theory for the problem concerning all these topics. Given a measure of complexity of the H space (VC-dim, shortly VC) we ask if we can use R_{emp} to approximate R.

- It holds with probability $1 - \delta$ that the guaranteed risk:

  $$R \leq R_{emp} + \epsilon\,(1/l, VC, 1/\delta)$$

  where ϵ is the VC-confidence.

- *Structural risk minimization*: minimize the bound !



Chapter 5

Validation

After models training on the TR set → evaluation of the performances.
Generalization/predictive accuracy evaluation.

5.1 Two aims

- Model selection: estimating the performance of different learning models in order to choose the best one to generalize (generalization error). This includes searching the best hyperparameteres of the model (e.g.polynomial order,...). It returns a model;
- Model assessment: having chosen a final model, estimating/evaluating its prediction error/risk on new test data. It returns an estimation.

Golden rule: keep separation between goals and use separate datasets.

5.2 Validation: ideal world

- A large TR to find the best hypothesis;
- a large VS validaition set for model selection;
- a very large external unseen data TS test set.

Basic techniques to estimate the generalization performance:

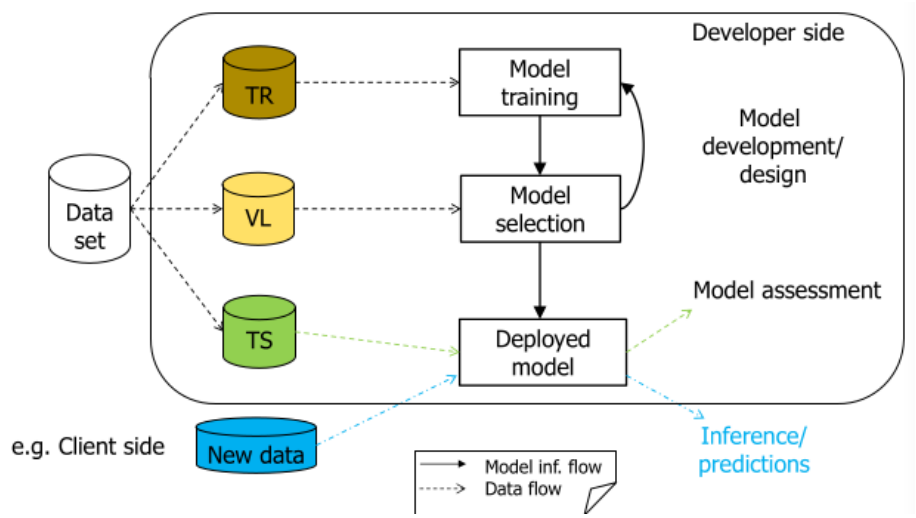
- Simple hold-out (basic setting)
- K-fold Cross Validation

5.3 Hold out cross validation

Hold out: basic setting. Partition dataset D into TR, VL and TS disjoint sets.

- TR used to training the algorithm;
- VL can be used to select the best model (e.g. hyperparams tuning);

- TS (result) is not to be used for any tuning/selection but it's only for model assessment.



5.4 K-fold cross validation

Hold out can make insufficient use of data. K-fold cross validation consists in:

- splitting dataset D into K mutually exclusive subsets $D_1, \ldots, D_K$;
- training the learning algorithm on $D \setminus D_i$ and test it on D_i;
- Can be applied for both VL or TS splitting.

5.5 Classification accuracy

5.5.1 Confusion matrix

Actual vs Predicted results.

Actual \ Predicted	Positive	Negative
Positive	TP	FN
Negative	FP	TN

We can define:

- Specifity (true negative rate)

$$Sp = \frac{TN}{FP + TN}$$

- Sensitivity (true positive rate)

$$Sens = \frac{TP}{TP + FN}$$

- Precision

$$Pr = \frac{TP}{TP + FP}$$

- Accuracy

$$Acc = \frac{TP + TN}{Tot}$$

Chapter 6

Linear Models

6.1 Regression

$$h_{\vec{w}}(\vec{x}) = out = w_1 x + w_0$$

$$(\vec{x}_p, y_p) \quad p = 1, \ldots, l$$

LMS least mean square of errors:

$$Loss(h_{\vec{w}}) = E(\vec{w}) = \sum_{p=1}^{l} (y_p - h_{\vec{w}}(x_p))^2 = \sum_{p=1}^{l} (y_p - w_1 x_p - w_0)^2$$

Note: to obtain the mean divide by l

6.1.1 Solve it

$$\frac{\partial E(\vec{w})}{\partial w_i} = 0$$

Same solutions for w_0 and w_1 of the LMS problem.

6.1.2 Notation for multidimensional inputs

$$\vec{w}^T \vec{x} + w_0 = w_0 + w_1 x_1 + \cdots + w_n x_n = w_0 + \sum_{i=1}^{n} w_i x_i$$

w_0: intercept, threshold, bias, offset..... If we put $x_0 = 1$ we can state an inner product:

$$\begin{cases} \vec{x}^T = [1, x_1, \ldots, x_n] \\ \vec{w}^T = [w_0, w_1, \ldots, w_n] \end{cases} \implies \vec{w}^T \vec{x} = \vec{x}^T \vec{w}$$

$$h(\vec{x}_p) = \vec{x}_p^T \vec{w}$$

6.2 Classification

We reuse the linear model. In this case categorical targets e.g. 0/1 or -1/+1.

$$\vec{w}^T\vec{x} + w_0 = 0$$

It defines a hyperplane through which we can descern among negative or positive values and classificate.

$$h(\vec{x}) = \begin{cases} 1 & if \quad \vec{w}^T\vec{x} + w_0 \geq 0 \\ 0 & otherwise \end{cases} \iff [0,1] \text{ output range}$$

$$h(\vec{x}) = sign(\vec{w}^T\vec{x} + w_0) \iff [-1,+1] \text{ output range}$$

The classification may be viewed as the allocation of the input space in decision regions (Linear threshold unit LTU in the form $\vec{w}^T\vec{x} = -w_0$).

6.3 Learning algorithms

Two algorithms both based on LMS:

- Normal equation solution (direct approach);
- Gradient descent (iterative approach).

6.3.1 Normal equation algorithm

Finding the $\vec{w}*$ that minimizes the expected loss on TR data

$$R_{emp} = \frac{1}{l}\sum_{p=1}^{l} L(h(\vec{x}_p), y_p) = \frac{1}{l}\sum_{p=1}^{l}(y_p - \vec{w}^T\vec{x})^2$$

- $$\frac{\partial E(\vec{w})}{\partial w_j} = 2\sum_{p=1}^{l}(y_p - \vec{x}_p^T\vec{w})x_{p,j}$$

- We can get the normal (imponing derivative = 0):

  $$\left(X^TX\right)\vec{w} = X^T\vec{y} \implies \vec{w} = \left(X^TX\right)^{-1}X^T\vec{y} = X^+\vec{y}$$

  Where "+" denotes the Moore-Penrose pseudoinverse of the matrix (it always exists)

- The solution are infinite: we can choose the min norm($\vec{w}$) solution

Computing the pseudoinverse of X

We write X in its singular value decomposition (SVD) so computing the pseudoinverse is easy.

$$X = U\Sigma V^T \implies X^+ = V\Sigma^+U^T$$

Where Σ is diagonal and so its pseudoinverse is obtained by replacing every nonzero entry by its reciprocal

6.3.2 Gradient descent

The gradient shows the direction where the function grows more. Its negative shows the direction where the function decreases and becomes flat.

- $$\frac{\partial E(\vec{w})}{\partial w_j} = -2\sum_{p=1}^{l}(y_p - \vec{x}_p^T\vec{w})\vec{x}_{p,j}$$

- Delta rule:

  $$\vec{w}_{new} = \vec{w} + \eta\Delta\vec{w}$$

  It's an error correction rule, where the "error" of the weights is:

  $$\Delta\vec{w} = -\vec{\nabla}_{\vec{w}}E(\vec{w})$$

  and η is the learning rate parameter (step size) which rules the speed of the gradient descending.

Batch/On-line version of gradient descent

- For batch version the gradient is the sum over all the l patterns:

  $$\frac{\partial E(\vec{w})}{\partial w_j} = -2\sum_{p=1}^{l}(y_p - \vec{x}_p^T\vec{w})x_{p,j}$$

  And the weights are updated all together after evaluating this sum (after one "epoch")

- For the on-line/stochastic version we update weights pattern after pattern, so the error depents on it:

  $$\frac{\partial E_p(\vec{w})}{\partial w_j} = -2(y_p - \vec{x}_p^T\vec{w})x_{p,j} = -\Delta_p w_j$$

  It could be faster, but it needs smaller learning rate.

There exist also intermediate cases like mini-batch training.

6.4 Linear models: inductive bias

- Language bias (constraints of the model): the H is a set of linear functions (may be very restrictive and rigid)

- Search bias (preferences on the learning algorithm): ordered search guided by the Least Squares Minimization goal

6.5 Extending linear model: LBE

Basis transformations: linear basis expansion (LBE):

$$h_{\vec{w}}(\vec{x}) = \sum_{k=0}^{K} w_k \phi_k(\vec{x})$$

$$\phi_k \quad : \quad \mathcal{R}^n \to \mathcal{R}$$

The expansion is linear but the basis are transformations of $\vec{x}$ which can be non linear (log, polynomials, roots....) So the model is linear in the parameters and also in ϕ, but not in $\vec{x}$: we can use same learning algorithms as before. It can be applied to both regression and classification.

- PROS: can model more complicated relationships, it is more expressive;
- CONS: with large basis of functions, we easily risk overfitting, hence we require methods for controlling the complexity of the model. Whereas complexity is not referring at any computational cost, but it's just a measure of the flexibility of the model to fit the data (VC-dim)

6.6 Improvements: Tikhonov regularization

Smoothed model is possible, just add contraints to the sum of value of $|w_j|$ penalizing models with high values of $|\vec{w}|$

$$Loss(\vec{w}) = \sum_{p=1}^{l} (y_p - \vec{x}_p^T \vec{w})^2 + \lambda ||\vec{w}||^2$$

i.e. we are favoring this way "sparse" models using less terms due to weights $w_j = 0$ or close to 0 so basically a less complex model. λ is the regularization (hyper)parameter.

6.6.1 Solving it

- Direct approach:
$$\vec{w} = \left(X^T X + \lambda I\right)^{-1} X^T \vec{y}$$
- Gradient approach $\implies$ weight decay technique:
$$\vec{w}_{new} = \vec{w} + \eta \Delta \vec{w} - 2\lambda \vec{w}$$

6.6.2 Tikhonov regularization: trade-off

$$Loss(\vec{w}) = \sum_{p=1}^{l} \left(y_p - \vec{x}_p^T \vec{w}\right)^2 + \lambda ||\vec{w}||^2$$

The trade-off is of course ruled by the value of λ:

- small λ leads to the risk of overfitting;
- high λ leads to the risk of underfitting.

Chapter 7

KNN

K-nearest neighbor. Still simple but flexible and local.

7.1 1-nn

- Simply store the training data

$$< \vec{x}_p, y_p > \quad p = 1, \ldots, l$$

- Given an input $\vec{x}$ of dimension n
- Find the nearest training example $\vec{x}_i$ (i-th pattern) s.t.:

$$\vec{x}_i = \arg\min_p d(\vec{x}, \vec{x}_i)$$

  Where d measures a distance e.g. Euclidian distance:

$$d(\vec{x}, \vec{x_p}) = \sqrt{\sum_{t=1}^{n} (x_t - x_{p,t})^2} = ||\vec{x} - \vec{x}_p||$$

- Then output y_i

7.2 K-nn

A natural way to classify a new point is to have a look at its neighbors and take an average. Denoting $N_k(\vec{x})$ the neighborhood of $\vec{x}$ that contains exactly k neighbors (closest patterns according to distance d), we have:

$$avg_k(\vec{x}) = \frac{1}{k} \sum_{\vec{x}_i \in N_k(\vec{x})} y_i$$

Majority Voting: if there is a clear dominance of one of the classes in the neighborhood of the observation then it is likely that the observation itself would belong to that class aswell. So, for targets in $[0, 1]$:

$$h(\vec{x}) = \begin{cases} 1 & if \quad avg_k(\vec{x}) > 0.5 \\ 0 & otherwise \end{cases}$$

While for regression tasks we can use directly the average mean over K-nn.

7.3 Limits of K-nn

- The computational cost is deferred to the prediction phase
- Computationally intensive in time and space (all the training data)
- Offer little interpretation (subjectivity of interpretation, dependancies on the metrics)
- Curse of dimensionality: when there are lots of input variables it totally fails
- Always in high dim, there are some problems: hard to find nearby points; low sampling density; irrelevant features issues.

Chapter 8

Bias/Variance and ensembling

A training set is only one possible realization from the universe of data: different TR sets can provide different estimates

Decomposition of the expected error at a point $\vec{x}$:

- Bias: quantifies the discrepancy between true function and $h(\vec{x})$: the smaller is H, the higher is the bias
- Variance: quantifies the variability of the response of model h for different realizations of the training data: the higher is the flexibility, the higher the variance
- Noise: because the labels include random error (e.g. for a given $\vec{x}$ there are more than one possible d)

8.1 Bias-Var. Analysis

Assuming that the data points are drawn i.i.d. from a unique probability distribution P The goal of the analysis is to compute, for an arbitrary new point $\vec{x}$,

$$E_P\left[(y - h(x))^2\right]$$

Where y is the value for $\vec{x}$ that could be present in a dataset, and the expectation is over all training sets drawn according to P. We will decompose this expectation into three components: bias, variance and noise.

We now that

$$\begin{cases} Var[Z] = E_P[(Z - \bar{Z})^2] \\ Var[Z] = E[Z^2] - \bar{Z}^2 \implies E[Z^2] = \bar{Z}^2 + Var[Z] \end{cases}$$

$$E_P\left[(y - h(x))^2\right] = E_P[h(x)^2 - 2yh(x) + y^2] = E_P[h(x)^2] + E_P[y^2] - 2E_P[y]E_P[h(x)]$$

So using the variance lemma above

$$E_P[h(x)^2] = \bar{h}(x)^2 + E_P[(h(x) - \bar{h}(x))^2]$$

And for the second term

$$E_P[y^2] = f(x)^2 + E_P[(y - f(x))^2]$$

Note that we used $E_P[y] = E_P[f(x) + \epsilon] = f(x)$ cause the noise is gaussian with 0 mean.

Putting all together and doing some calculations we find:

$$E_P[(y - h(x))^2] = E_P[(h(x) - \bar{h}(x))^2] + (\bar{h}(x) - f(x))^2 + E_P[(y - f(x))^2]$$

And we can clearly see that first term is the variance of our model, the second is the (bias)^2 and the third is the (noise)^2.

8.2 Bias/Var and regularization

Recall regularization:

$$Loss(\vec{w}) = \sum_p (d - p - o(\vec{x}_p))^2 + \lambda ||\vec{w}||^2$$

Varying λ we can obtain more complex solutions (low λ) or more regularized and simple solutions (high λ), and of course there are consequences on bias and variance of our model.

- High λ: higher bias, lower variance
- Decreasing λ: we are more dependent on the specific training data
- Very low λ: low bias, high variance; on average the best situation.

8.3 Ensemble Learning

Take advantage of multiple models: simplest case "voting schema".

- Regression: Average the output:

$$\bar{h}(\vec{x}) = \frac{1}{K} \sum_{i=1}^{K} h_i(\vec{x})$$

- Classification: Take a vote over many classifier - we can proceed with some methods like bagging, boosting.

8.3.1 Bagging (bootstrap aggregating)

Bagging: train K classifiers on different subsets of training set and differentiate each training using bootstrap (resampling with replacement) The average of h reduces the variance.

8.3.2 Boosting (e.g. AdaBoost)

If models have the same errors there is not such advantage on ensembling. Boosting can solve this issue greatly improving performance of weak learners. It consists in:

1. Differentiate each training focusing on errors (more weight to difficult instances)
2. Combine results by output weights (weighted vote for classifiers, with more weight to low error classifiers)

But it can suffer for noisy data (those are accidentally weighted more)

Chapter 9

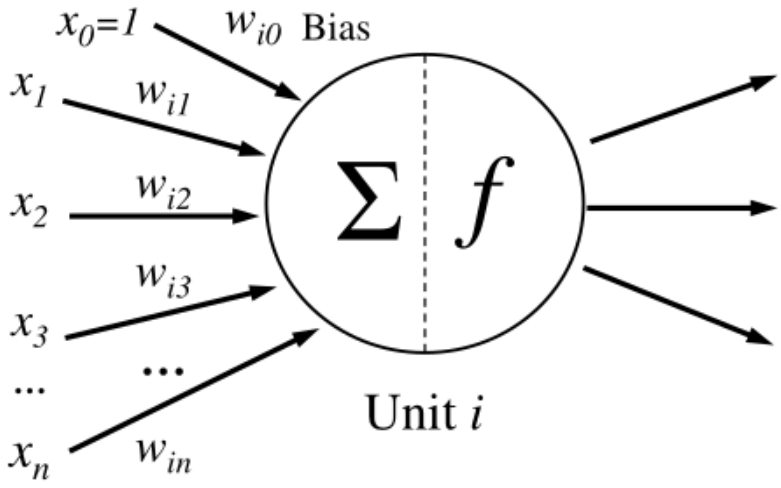
Networks models

9.1 Artificial neuron: processing unit

- Node, neuron or unit;
- Input: from extern source or other units
- Input connections: weights $\vec{w}$ free parameters; modified by learning.

$$\begin{cases} net_i(\vec{x}) = \sum_j w_{ij} x_j \\ o_i(\vec{x}) = f(net_i(\vec{x})) \end{cases}$$

The weighted sum net_i is called the net input to unit i and the function f is the unit's activation function (e.g.linear,LTU,...)



9.1.1 Perceptron

Single neuron with LTU as activation function. Linear treshold unit.

9.1.2 McCulloch and Pitts networks

Neurons in two possible states: firing (1) and not firing (0). All synapses are equivalent and characterized by a single real number (their strenght, $\vec{w}$) which is positive for excitatory connections and negative for inhibitory ones.

- a neuron i becomes active when the sum of those connections w_{ij} coming from neurons j connected to it which are active, plus a bias, is larger than zero. Binary inputs $\rightarrow$ binary output.
- "basically" the LTU/perceptron

9.1.3 Properties of Networks of perceptrons

- Perceptrons can represent any universal boolean functions like AND, OR, NOT (NAND, NOR)
- Only two levels deep is sufficient (more can be more efficient)
- Single layer cannot model all the possible functions due to the linear seaparable problems
- No provided (up to now) learning algorithm even for one unit!

9.1.4 Learning for one unit model - Overview

Two kinds of method for learning (also historical view)

- Adaline (Widrow, Hoff, 1959) = Adaptive Linear Neuron: linear unit during training; LMS direct solution and gradient descent solution
- Perceptron (Rosenblatt, 1957) = non-linear unit during training: with hard limiter or threshold activation function (ONLY CLASSIFICATION)

9.1.5 The perceptron learning algorithm

Rosenblatt (1958-1962). Based on minimizing the number of misclassified patterns:

$$\text{Find } \vec{w} \text{ s.t.} sign(\vec{w}^T \vec{x}) = d$$

It's an on-line algorithm: a step can be made for each input pattern.

1. Initialize the weights (either to zero or to a small random value)
2. pick a learning rate η (this is a number between 0 and 1)
3. until stopping condition is satisfied (e.g. weights don't change): for each training pattern $(\vec{x}, d)$ with d=+1/-1 let *out* be:

$$out = sign(\vec{w}^t \vec{x}) \in [-1, +1]$$

4. if $out = d$ don't change weights (i.e. "minimize only misclassifications"!)

5. if $out \neq d$ update the weights:

$$\vec{w}_{new} = \vec{w} + \eta d\vec{x} \text{ add } \begin{cases} +\eta\vec{x} \text{ if } \vec{w}^T\vec{x} \leq 0 \text{ and } d = +1 \\ -\eta\vec{x} \text{ if } \vec{w}^T\vec{x} > 0 \text{ and } d = -1 \end{cases}$$

Or in a unique different form:

$$\vec{w}_{new} = \vec{w} + \frac{\eta}{2}(d - out)\vec{x}$$

9.1.6 Perceptron's Convergence Theorem

The perceptron with the perceptron learning algorithm is always able to learn what it is able to represent: linear decision boundaries.

"The perceptron is guaranteed to converge (classifying correctly all the input patterns) in a finite number of steps if the problem is linearly separable".

This independently from the starting point, even if the final solution could depend on it. May be "unstable" if the problem is not separable.

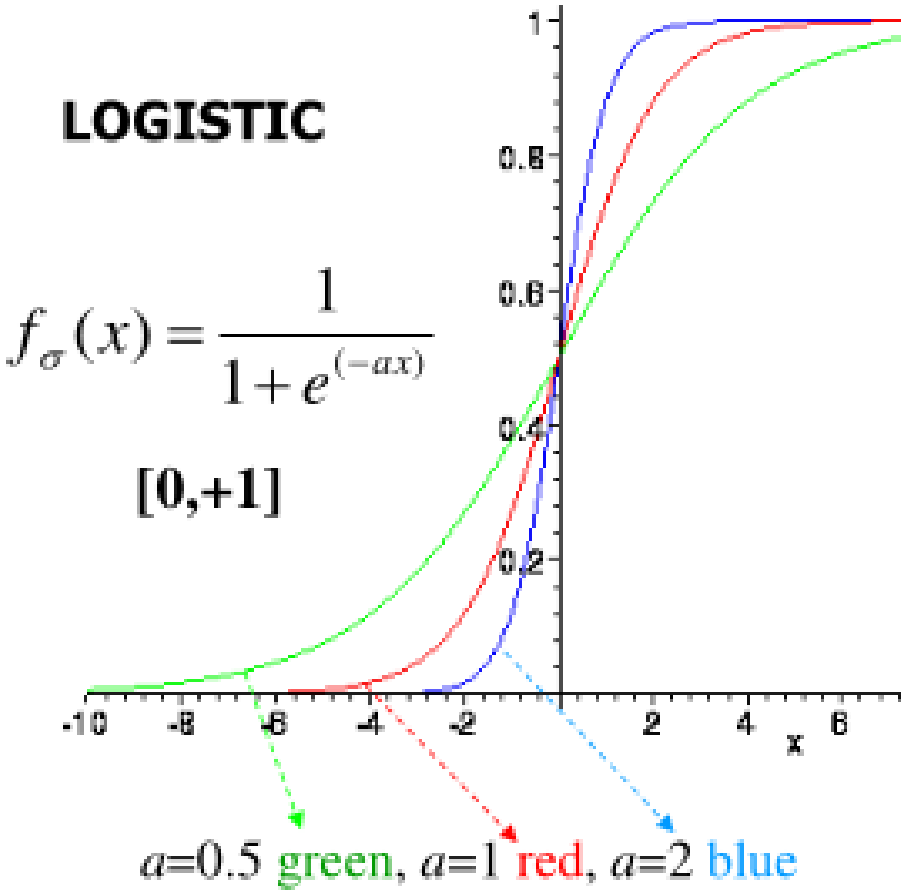
9.1.7 Perceptron Learn. Alg. Vs LMS alg.

- Min. misclassifications $out = sign(\vec{w})^T\vec{x}$ / Vs / Min. $E(\vec{w})$ with $out = \vec{w}^T\vec{x}$;
- Always converges for linear separable problems to a perfect classifier / Vs / Asymptotic convergence also for not linear separable problems;
- Difficult to be extended to networks of units (NN) / Vs / It can be extended to networks of units (NN), using the gradient based approach.

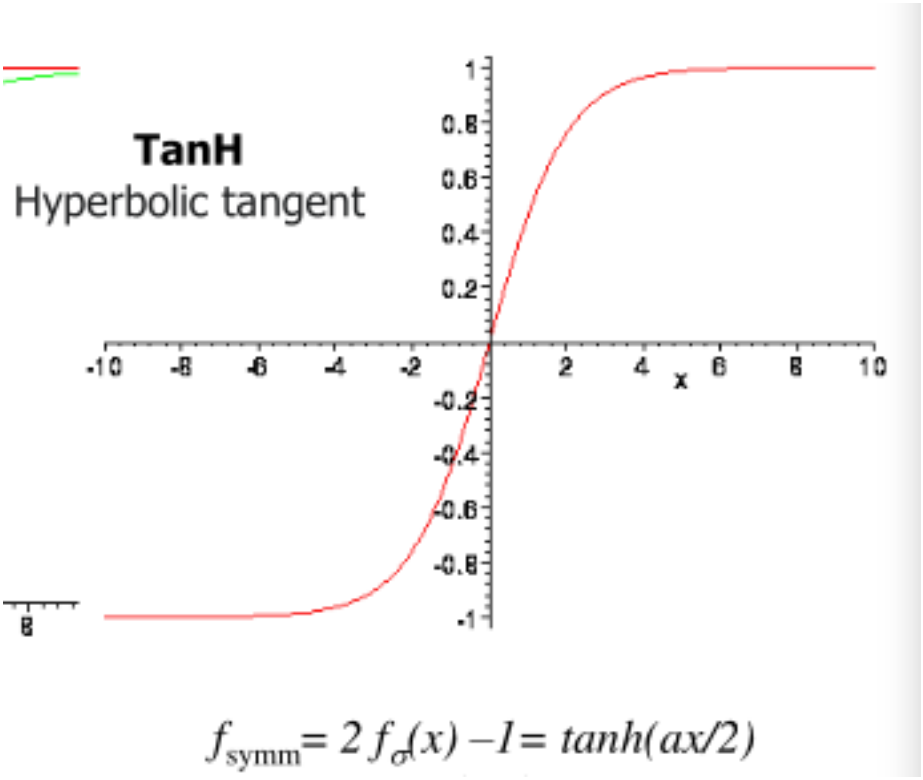
9.2 Activation functions

These functions are used to "activate" units. They can be linear as the case of perceptron/LTU,or non-linear as squashing functions like sigmoidal logistic function. Here some examples follow.

- Sigmoidal (logistic) function



- Hyperbolic tangent

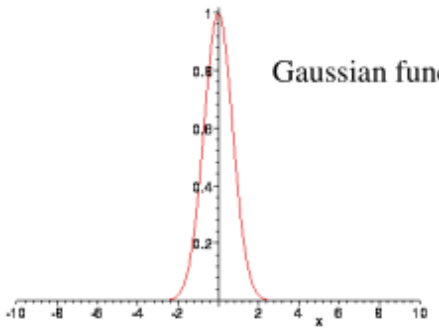


$$f_{\text{symm}} = 2 f_{\sigma}(x) - 1 = \tanh(ax/2)$$

- Radial Basis Function (RBF, gaussian)

$$f(x) = e^{(-ax^2)}$$

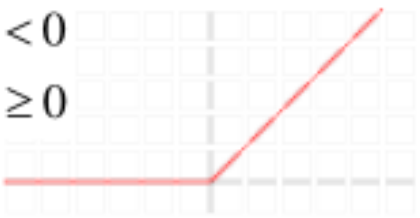
$$\|\boldsymbol{w} - \boldsymbol{x}_{input}\|$$



- Rectified Linear Unit (ReLu)

$$f(x) = \begin{cases} 0 & for \quad x < 0 \\ x & for \quad x \geq 0 \end{cases}$$

$$f(x) = \max(0, x)$$



- Softplus (smooth approximation of ReLu):

$$f(x) = ln(1 + e^x)$$

All these functions provide continuous outputs but it is always possible to assign e.g. for logistic function positive classes to all the values ≥ 0.5 and negative classes to all the others. It is always possible to change the threshold and even to consider a rejection zone in an interval around the threshold to avoid fragile decisions.

9.2.1 Activation functions: derivatives

These activation functions are useful to us also because their derivatives have an easy analytical form and computing them is so simple. (NOTE: except done for the step function because it's not continuous!)

Some examples, for sigmoid and tanh, for any values of a:

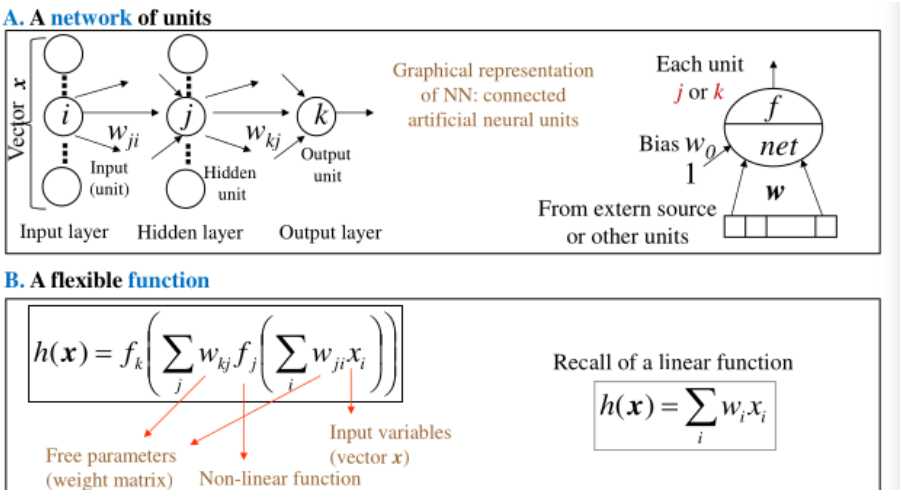
$$\frac{df_\sigma(x)}{dx} = f_\sigma(x)(1 - f_\sigma(x))$$

$$\frac{df_{tanh}(x)}{dx} = 1 - f_{tanh}(x)^2$$

9.3 Neural networks (NN)

NN models tradionally presented by the type of:

- UNIT: net, activation functions;
- ARCHITECTURE: number of units, topology, number of layers;
- LEARNING ALGORITHM.



The architecture of a NN defines the topology of the connections among the units.

9.3.1 The Multi Layer Perceptron (MLP)

A multilayer perceptron (MLP) is a name for a modern feedforward neural network consisting of fully connected neurons with nonlinear activation functions, organized in layers, notable for being able to distinguish data that is not linearly separable.

9.3.2 Feedforward processing

Input $\longrightarrow$ output:
For each input pattern $\vec{x}$:

1. The input pattern is loaded in the input layer;
2. We compute the output of all the units of the 1st hidden layer;
3. We compute the output of all the units of the 2st hidden layer and so on for all the hidden layers;
4. We compute the output of all the units of the output layer (NN output $h(\vec{x})$);
5. We can now compute the error (delta) at the output level.

9.3.3 NN flexibility: which tasks?

H space: continuous space of all the functions that can be represented by assigning the weight values of the given architecture. Note that, depending on the class of values produced by the network output units, discrete or continuous, the model can deal respectively with classification or regression tasks.

$$h(\vec{x}) = f_k\left(\sum_j w_{kj} f_j\left(\sum_i w_{ji} x_i\right)\right)$$

Also, if using multiple output units, we can deal with multi-regression or multi-classes classifications.

9.3.4 NN as a function

$$h(\vec{x}) = f_k\left(\sum_j w_{kj} f_j\left(\sum_i w_{ji} x_i\right)\right)$$

- This is the function computed by a two-layer feedforward neural network
- Units and architecture are just a graphical representation of the data flow processing
- Each f_j $(\sum_i w_{ji} x_i)$ can be seen as computed by an independent processing element (unit, hidden unit) or a special kind of ϕ of LBE
- also, NN is a function non linear in the parameters $\vec{w}$

NN compared to LBE

- LBE (recall): fixed linear basis functions

$$\text{Regression: } h(\vec{x}) = \sum_j w_j \phi_j(\vec{x}) \quad \text{Classification: } h(\vec{x}) = f\left(\sum_j w_j \phi_j(\vec{x})\right)$$

- Neural Network:

$$\begin{cases} h(\vec{x}) = f_k\left(\sum_j w_{kj} f_j\left(\sum_i w_{ji} x_i\right)\right) = f\left(\sum_j w_j \phi_k(\vec{x}, \vec{w})\right) \\ \phi_j(\vec{x}, \vec{w}) = f_j\left(\sum_i w_{ji} x_i\right) \end{cases}$$

So we can see that basically NN are adaptive and flexible types of LBE: now the ϕ depends on weights so they can change with data training. The basis functions themselves are adapted to data by fitting the $\vec{w}$.

So $h(\vec{x})$ is just a nonlinear function of weighted sums of nonlinearly transformed linear models + the important enhancement of adaptivity.

9.3.5 Universal approximation

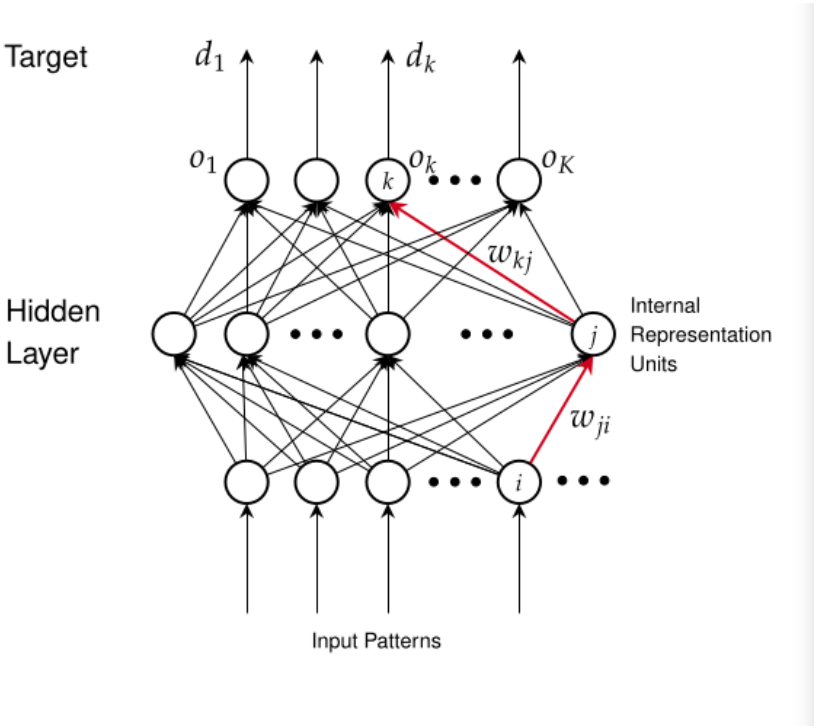
The flexibility of NNs is theoretically grounded.

"A single hidden-layer network with logistic activation functions can approximate arbitrarily well every continuous (on hypercubes) function provided enough units in the hidden layer".

This is an existence theorem. This not provide the algorithm nor the number of units required.

9.4 The learning algorithms: back-propagation

The basic architecture is a feed-forward fully connected neural network (MLP). We use different indices, k, j and i, as generic indices for the different layers of the MLP.



- The problem: to estimate the contribution of hidden units to the error at the ouput level. We do this by computing the Generalized Delta Rule.
- Training set (TR): $\left\{\left(\vec{x}^1, d^1\right), \ldots, \left(\vec{x}^l, d^l\right)\right\} \rightarrow$ supervised learning. Each input unit i is loaded with the input x_i (component i of the input vector $\vec{x}$), hence $o_i = x_i$ per each pattern $p = 1, \ldots, l$.
- Total Error:

$$E_{tot} = \sum_p E_p \quad E_P = \frac{1}{2} \sum_{k=1}^{K} \left(d_k - o_k\right)^2$$

  E_p is computed for the input pattern p for all the K units.
- Objective: find the values of parameters $\vec{w}$ that minimize E_{tot}
- Least Mean Square (LMS): the minimization of the above total error.
- Idea: perform a descent on the surface of E_{tot} on the basis of the gradient.

9.4.1 Iterative gradient descent training algorithm

Back-propagation (the same presented in general for the LMS)

Compute units outputs and E_{tot}

;

While $E_{tot} > \epsilon$ (desired value or other criteria) do:

$$\forall w \in \vec{w} : \Delta w = -\frac{\partial E_{tot}}{\partial w}$$
$$w^{new} \leftarrow w + \eta \Delta w + ...$$

Compute units outputs and E_{tot};

end.

Backpropagation focus mainly on step 1: computing the gradient.

$$\Delta \vec{w} = -\frac{\partial E_{tot}}{\partial \vec{w}} = -\sum_p \frac{\partial E_p}{\partial \vec{w}} = \sum_p \Delta_p \vec{w}$$

Whereas p is the p-th pattern to be fed as input to the neural network.

Then, for w_{tu} of a generic unit t, pattern p, and the input u from a generic unit u to the unit t, we have:

$$\Delta_p w_{tu} = -\frac{\partial E_p}{\partial net_t} \frac{\partial net_t}{\partial w_{tu}} = \delta_t o_u$$

$$\begin{cases} net_t = \sum_s w_{ts} o_s \\ o_t = f_t(net_t) \end{cases} \implies \frac{\partial net_t}{\partial w_{tu}} = \frac{\partial \sum_s w_{ts} o_s}{\partial w_{tu}} = o_u$$

Now we expand on δ_t (the delta of a unit t):

$$\delta_t = -\frac{\partial E_p}{\partial net_t} = -\frac{\partial E_p}{\partial o_t} \frac{\partial o_t}{\partial net_t} = -\frac{\partial E_p}{\partial o_t} f_t'(net_t)$$

Now we recall that $E_p = \frac{1}{2} \sum_{k=1}^{K} (d_k - o_k)^2$ and we have to distinguish 2 cases, depending on whether o_t is an output unit k (case 1) or an hidden unit j (case 2).

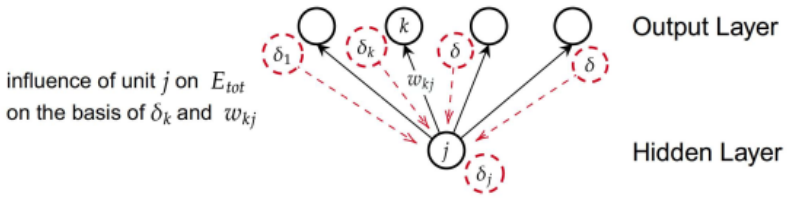
Case 1: Output Unit (t=k)

$$-\frac{\partial E_p}{\partial o_k} = -\frac{\partial \frac{1}{2} \sum_{r=1}^{K} (d_r - o_r)^2}{\partial o_k} = (d_k - o_k)$$
$$\delta_k = -\frac{\partial E_p}{\partial net_k} = (d_k - o_k) f_k'(net_k)$$

Case 2: Hidden unit (t=j)

$$-\frac{\partial E_p}{\partial o_j} = \sum_{k?1}^{K} -\frac{\partial E_p}{\partial net_k} \frac{\partial net_k}{\partial o_j} = \sum_{k=1}^{K} \delta_k w_{kj}$$
$$\text{since} \frac{\partial net_k}{\partial o_j} = \frac{\partial \sum_s w_{ks} o_s}{\partial o_j} = w_{kj}$$
$$\implies \delta_j = \left(\sum_{k=1}^{K} \delta_k w_{kj} \right) f_j'(net_j)$$

So we can see from this latter formula that we can back-propagate the signals δ_k that we had already obtained from the output to the hidden unit j.



9.5 Learning rate - improvements

Act on learning rate η for a better efficiency:

1. Nesterov momentum;
2. Variable learning rate;
3. Adaptive learning rates (change during training and for each $\vec{w}$);
4. Changing it depending on the layer (higher for deep layers) or higher for few input connections.

9.5.1 Nesterov Momentum

$$0 < \alpha : \text{momentum parameter} < 1$$

$$\Delta\vec{w}_{new} = -\eta\frac{\partial E(\vec{w})}{\partial w} + \alpha\Delta\vec{w}_{old}$$

Note: here η is included inside the correction for $\vec{w}_{new}$, differently from before.

$$\vec{w}_{new} = \vec{w} + \Delta\vec{w}_{new}$$

Chapter 10

Support Vector Machines (SVM)

Notation:

- N is the number of examples;
- m is the dimension of the input vector;
- b instead of w_0 as the intercept or bias.

10.1 SVM for binary classification

- Linear machine (LTU, Perceptron, ...)
- Maximization of the separation margin
- Structural risk minimization

Initially (hard margin SVM) we assume linear separable problems and no errors in data.

10.1.1 Linear separable patterns - Hard margin SVM

Given the training set $TR = \{(\vec{x}_{,i}, d_i)\}_{i=1}^{N}$

Find an hyperplane of equation $\vec{w}^T\vec{x} + b = 0$ to separate the examples:

$$\vec{w}^T\vec{x}_i + b \geq 0 \; for d_i = +1$$
$$\vec{w}^T\vec{x}_i + b < 0 \; for d_i = -1$$

- $g(\vec{x}) = \vec{w}^T\vec{x}_i + b$ is the discriminant function;
- $h(\vec{x}) = sign(g(\vec{x}))$ is the hypothesis.

Separation margin

The separation margin ρ is evaluated as (the double of) the distance between the linear hyperplane and the closest data point. We can think to a "safe zone".

Optimal hyperplane

The optimal hyperplane is the hyperplane which maximizes the margin ρ:

$$\vec{w}_o^T \vec{x} + b_o = 0$$

We will find:

$$\rho = \frac{2}{||\vec{w}||}$$

So maximize $\rho \iff$ minimize $||\vec{w}||$

Canonical representation of the hyperplane

$$\vec{w}^T \vec{x}_i + b \geq 0 \ for d_i = +1$$
$$\vec{w}^T \vec{x}_i + b < 0 \ for d_i = -1$$

Re-scaling $\vec{w}$ and b so that the closest points to the separating hyperplane satisfy

$$|g(\vec{x}_i)| = |\vec{w}^T \vec{x}_i + b| = 1$$

We can rewrite:

$$\vec{w}^T \vec{x}_i + b \geq 1 \ for d_i = +1$$
$$\vec{w}^T \vec{x}_i + b < -1 \ for d_i = -1$$

So in a compact form:

$$d_i(\vec{w}^T \vec{x}_i + b) \geq 1 \forall i = 1, \ldots, N$$

Support Vector

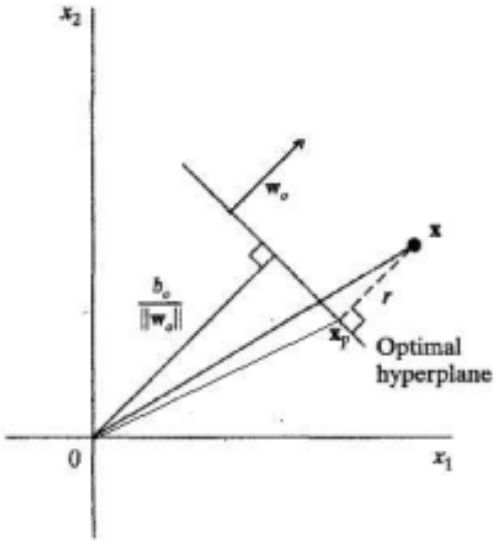
A support vector $\vec{x}^{(s)}$ satisfies the previous equation exactly:

$$d^{(s)}(\vec{w}^T \vec{x}^{(s)} + b) = 1$$

Computing the distance

The discriminant function is $g(\vec{x}) = \vec{w}^T \vec{x} + b$. Recall that $\vec{w}_o$ is a vector orthogonal to the hyperplane. Let's denote the distance between $\vec{x}$ and the optimal hyperplane with r:

$$\vec{x} = \vec{x}_p + r \frac{\vec{w_o}}{||\vec{w}_o||}$$



Evaluating it in $g(\vec{x})$ we obtain:

$$g(\vec{x}) = g(\vec{x}_p) + r\vec{w}_o^T \frac{\vec{w}_o}{||\vec{w}_o||}$$

$$\implies g(\vec{x}) = 0 + r\frac{||\vec{w}_o||^2}{||\vec{w}_o||} = r||\vec{w}_o|| \implies r = \frac{g(\vec{x})}{||\vec{w}_o||}$$

So for a positive support vector $\vec{x}^{(s)}$:

$$\vec{x}^{(s)} = \frac{g(\vec{x}^{(s)})}{||\vec{w}_o||} = \frac{1}{||\vec{w}_o||} = \frac{\rho}{2}$$

$$\implies \rho = \frac{2}{||\vec{w}_o||}$$

So the optimal hyperplane maximizes ρ and minimizes $||\vec{w}||$

Quadratic optimization problem

Finding the optimum values for $\vec{w}$ and b in order to maximize the margin yields to a quadratic optimization problem → Hard margin SVM

Optimization problem - Primal form

Given the training samples $T = \{(\vec{x}_i, d_i)\}_{i=1}^N$, find the optimum values of $\vec{w}$ and b which minimize:

$$\Psi(\vec{w}) = \frac{1}{2}\vec{w}^T\vec{w} \qquad (min(||\vec{w}||))$$

satisfying the constraints (zero classification errors)

$$d_i\left(\vec{w}^T\vec{x}_i + b\right) \geq 1 \forall i = 1, \ldots, N$$

- The objective function $\Psi(\vec{w})$ is quadratic and convex in $\vec{w}$;
- The constraints are linear in $\vec{w}$;
- Solving this problem scales with the size of the input space m.

Solving the quadratic problem

To solve this problem we use the Lagrangian multipliers method:

$$J(\vec{w}, b, \alpha) = \frac{1}{2}\vec{w}^T\vec{w} - \sum_{i=1}^{N} \alpha_i(d_i(\vec{w}^T\vec{x}_i + b) - 1)$$

- Minimize J with respect to $\vec{w} \implies \frac{\partial f}{\partial \vec{w}} = 0 \implies \vec{w}_o = \sum_{i=1}^{N} \alpha_i d_i \vec{x}_i$;
- Minimie J with respect to $b \implies \frac{\partial f}{\partial b} = 0 \implies \sum_{i=1}^{N} \alpha_i d_i = 0$;
- And we may substitute these in J to study the dual form and obtain the Lagrangian multipliers $\{\alpha_i\}_{i=1}^{N}$

Kuhn-Tucker conditions

From the KT conditions it follows that in the saddle point of J:

$$\alpha_i(d_i(\vec{w}^T\vec{x}_i + b) - 1) = 0 \forall i = 1, \ldots, N$$

- if $\alpha_i > 0$, then the $d_i(\vec{w}^T\vec{x}_i + b) = 1$ and $\vec{x}_i$ is a support vector;
- if $\vec{x}_i$ is not a support vector then $\alpha_i = 0$

Hence we can restrict the computation to N_s:

$$\vec{w}_o = \sum_{i=1}^{N_s} \alpha_{o,i} d_i \vec{x}_i$$

So the hyperplane depends only from support vectors.

Optimization problem - Dual form

Given the training samples $T = \{(\vec{x}_i, d_i)\}_{i=1}^{N}$, find the optimum values of the Lagrangian multipliers $\{\alpha_i\}_{i=1}^{N}$ which maximize:

$$Q(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j d_i d_j \vec{x}_i^T\vec{x}_j$$

satisfying the constraints

- $\alpha_i \geq 0 \forall i = 1, \ldots, N$;
- $\sum_{i=1}^{N} \alpha_i d_i = 0$

Solving dual problem

- Solve the dual problem obtaining $\{\alpha_{o,i}\}_{i=1}^{N}$
- Compute $\vec{w}_o = \sum_{i=1}^{N} \alpha_{o,i} d_i \vec{x}_i$
- Compute $b_o = 1 - \vec{w}_o^T \vec{x}^{(s)}$ corresponding to a positive support vector $\vec{x}^{(s)}$:

$$b_o = 1 - \sum_{i=1}^{N} \alpha_{o,i} d_i \vec{x}_i^T \vec{x}^{(s)}$$

- IT'S NOT NEEDED to explicitly know $\vec{w}_o$
- It's needed to know Lagrangian multipliers (by solving the dual problem) and then compute the optimal bias b_o

Remember:

$$\vec{w}_o = \sum_{i=1}^{N} \alpha_{o,i} d_i \vec{x}_i$$

And the decision surface is given by:

$$\vec{w}_o^T \vec{x} + b_o = 0 \iff \sum_{i=1}^{N} \alpha_{o,i} d_i \vec{x}_i^T \vec{x} + b_o = 0$$

So, given the input pattern $\vec{x}$:

1. Compute $g(\vec{x}) = \sum_{i=1}^{N} \alpha_{o,i} d_i \vec{x}_i^T \vec{x} + b_0$;
2. Classify $\vec{x}$ as the sign of $g(\vec{x})$.

Note that it is not necessary to compute $\vec{w}_o$ and the sum over N can be restricted to the support vectors N_s

10.1.2 Soft margin SVM

In the soft margin SVM paradigm, at least one point violate the exact-fit condition:

$$d_i(\vec{w}^T \vec{x}_i + b) \geq 1$$

Admitting points inside the margin allows us to have a larger margin.

So we introduce the non-negative scalar variables (slack variables):

$$\xi_i \geq 0 \forall i = 1, \ldots, N$$

$$d_i(\vec{w}^T \vec{x}_i + b) \geq 1 - \xi_i \forall i = 1, \ldots, N$$

Support vector (soft margin)

A support vector $\vec{x}_i$ satisfies the previous exactly:

$$d_i(\vec{w}^T \vec{x}_i + b) = 1 - \xi_i$$

Primal problem - soft margin

Given the training set $T = \{(\vec{x}_i, d_i)\}_{i=1}^N$ find the values of $\vec{w}$ and b which minimize the objective function:

$$\Psi(\vec{w}, \xi) = \frac{1}{2}\vec{w}^T\vec{w} + C\sum_{i=1}^{N}\xi_i$$

Under the constraints:

- $d_i(\vec{w}^T\vec{x}_i + b) \geq 1 - \xi_i \forall i = 1, \ldots, N$
- $\xi_i \geq 0 \forall i = 1, \ldots, N$

C as a regularization hyper-parameter, a trade-off between empirical risk minimization and capacity term (VC-confidence) minimization:

- Low C $\rightarrow$ many TR errors are allowed $\rightarrow$ possible underfitting;
- High C $\rightarrow$ no TR errors allowed $\rightarrow$ smaller margin, possible overfitting.

Dual problem (soft margin)

Given the training set $T = \{(\vec{x}_i, d_i)\}_{i=1}^N$ find the values of the Lagrangian multipliers $\{\alpha_i\}_{i=1}^N$ which maximize the objective function:

$$Q(\alpha) = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j d_i d_j \vec{x}_i^T x_j$$

Satisfying the constraints:

- $\sum_{i=1}^N \alpha_i d_i = 0$
- $0 \leq \alpha_i \leq C \forall i = 1, \ldots, N$

Note that C is a user defined parameter.

Kuhn-Tucker conditions (soft margin)

The KT conditions are now defined as:

- $\alpha_i(d_i(\vec{w}^T\vec{x}_i + b) + \xi_i - 1) = 0 \forall i = 1, \ldots, N;$
- $\mu_i\xi_i = 0 \forall i = 1, \ldots, N$

Where the μ_i are Lagrange multipliers introduced to enforce the non-negativity of the slack variables (in the Lagrange primal function)

$$0 < \alpha_i < C \rightarrow \xi_i = 0 (\text{on the edge margin})$$

$$\alpha_i = C \rightarrow \xi_i \geq 0 (\text{inside the margin})$$

Solving the problem

- Solve the dual problem and obtain $\{\alpha_i\}_{i=1}^N$
- Find $\vec{w}_o$ and b_o from $\{\alpha_{o,i}\}_{i=1}^N$

$$\vec{w}_o = \sum_{i=1}^{N} \alpha_{o,i} d_i \vec{x}_i$$

$$b_o = d_j - \sum_{i=1}^{N} \alpha_{o,i} d_i \vec{x}_i^T \vec{x}_j \text{for a pattern } j \text{ s.t. } 0 < \alpha_j < C$$

Again, the non-zero Lagrangian multipliers correspond to support vectors. We use this as before:

$$g(\vec{x}) = \sum_{i=1}^{N} \alpha_{o.i} d_i \vec{x}_i^T \vec{x} + b_o$$

$$h(\vec{x}) = sign(g(\vec{x}))$$

10.1.3 Mapping to High-Dimensional feature space

Idea: using a non-linear map for the data points in the input space leading to a high dimensional feature space, where they can be linearly separable.

1. Cover's Theorem: the pattern are linearly separable with high probability in the feature space under such conditions.
2. Finding the optimal hyperplane to separate the patterns in this new space.

However, high dimensional feature spaces (see large basis function expansion) require control on the complexity of the problem; sometimes can lead to overfitting or be computationally unfeasibile

$$\Longrightarrow \text{ Kernel approach}$$

Solving the problem in the Feature Space

Non linear function map

$$\phi : \mathcal{R}^{m_0} \longrightarrow \mathcal{R}^{m_1}$$

$$\vec{x} \longrightarrow \phi(\vec{x})$$

- The problem is formulated as before, but the training set is now $T = \{(\phi(\vec{x}_i), d_i)\}_{i=1}^N$
- The hyperplane is then $\vec{w}^T \phi(\vec{x}) + b = 0$ and we can incorporate the bias in the weight vector:

$$\begin{array}{r} w(0) = b \\ \phi_0(\vec{x}) = 1 \end{array} \Longrightarrow \phi(\vec{x}) = (\phi_0(\vec{x}) = 1, \phi_1(\vec{x}), \ldots, \phi_{m_1}(\vec{x}))^T$$

So the decision surface equation is: $\vec{w}^T \phi(\vec{x}) = 0$

So we can follow same step as before, but considering the input transformation $\phi(\vec{x})$.

$$\vec{w} = \sum_{i=1}^{N} \alpha_i d_i \phi_(\vec{x}_i)$$

Thus the hyperplane equation is:

$$\sum_{i=1}^{N} \alpha_i d_i \phi^T(\vec{x}_i)\phi(\vec{x}) = 0$$

Note that the dot product is now in the feature space! Evaluating $\phi(\vec{x})$ could be intractable $\implies$ Kernel trick.

Kernel trick

We can use the Inner product Kernel Function k to solve the problem without even evaluate $\phi(\vec{x})$ neither need to know the feature space itself!

$$k : \mathcal{R}^{m_0} \times \mathcal{R}^{m_0} \longrightarrow \mathcal{R}$$

$$k(\vec{x}_i, \vec{x}) = \phi^T(\vec{x}_i)\phi(\vec{x})$$

$$k(\vec{x}_i, \vec{x}) = k(\vec{x}, \vec{x}_i)$$

Crucial point: the dot product in feature space is evaluated without considering the feature mapping and the feature space itself. We evaluate the dot product in terms of the input patterns.

We can then define the Kernel Matrix:

$$K = \begin{pmatrix} k(\vec{x}_1, \vec{x}_1) & \dots & k(\vec{x}_1, \vec{x}_N) \\ k(\vec{x}_2, \vec{x}_1) & \dots & k(\vec{x}_2, \vec{x}_N) \\ \dots & & \\ k(\vec{x}_N, \vec{x}_1) & \dots & k(\vec{x}_N, \vec{x}_N) \end{pmatrix}$$

$$K = \{k(\vec{x}_i, \vec{x}_j)\}_{(i,j)=1}^{N}$$

Properties of Kernels

- Kernels gaining positive semi-definite kernel matrices (having non-negative eigenvalues of the kernel matrices) define a metric, so it is possible to compute the inner prdocut in a feature space through them.

- If k_1 and k_2 are kernels then the following are kernel functions aswell:

$$k_1(\vec{x}, \vec{y}) + k_2(\vec{x}, \vec{y})$$
$$\alpha k_1(\vec{x}, \vec{y}) \forall \alpha \in \mathcal{R}_+$$
$$k_1(\vec{x}, \vec{y}) k_2(\vec{x}, \vec{y})$$

Primal problem in feature space

Given the training set $T = \{(\phi(x_i), d_i)\}_{i=1}^N$ find the optimal value of $\vec{w}$ which minimizes the objective function:

$$\Psi(\vec{w}, \xi) = \frac{1}{2}\vec{w}^T\vec{w} + C\sum_{i=1}^{N}\xi_i$$

under the constraints:

- $d_i(\vec{w}^T\phi(x_i)) \geq 1 - \xi_i \forall i = 1, \ldots, N$;
- $\xi_i \geq 0 \forall i = 1, \ldots, N$

Note that $\vec{w}$ is now in the feature space and thus its dimensionality could lead to an intractable problem.

Dual problem in feature space

Given the training set $T = \{(\phi(x_i), d_i)\}_{i=1}^N$ find the optimal values of $\{\alpha_i\}_{i=1}^N$ which maximize the objective function:

$$Q(\alpha) = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{N}\alpha_i\alpha_j d_i d_j k(\vec{x}_i, \vec{x}_j)$$

satisfying the constraints:

- $\sum_{i=1}^N \alpha_i d_i = 0$;
- $0 \leq \alpha_i \leq C \forall i = 1, \ldots, N$

Remember C is a user specified non-negative parameter (regularization parameter).

Remember also that the optimization problem solution gains a sparse solution in $\{\alpha\}_{i=1}^N$, as all the multipliers corresponding to a non-support vector are zeros.

Some examples of Kernels

- Polynomial Learning Machine:

  $$k(\vec{x}, \vec{x}_i) = (\vec{x}^T\vec{x}_i + 1)^p$$

  where p is a user specified parameter

- Radial Basis Function Net:

  $$k(\vec{x}, \vec{x}_i) = e^{-\frac{1}{2\sigma^2}||\vec{x}-\vec{x}_i||^2}$$

  where σ^2 is a user specified parameter

- Two-layer perceptron:

  $$k(\vec{x}, \vec{x}_i) = tanh(\beta_0\vec{x}^T\vec{x}_i + \beta_1)$$

  where $\beta_0 > 0$ and $\beta_1 < 0$ are user specified parameters. However this last example computes an inner product only for some choices for β_0 and β_1 (Mercer's theorem)

10.2 SVM for non-linear regression

Recall the regression problem:

$$T = \{(\vec{x}_i, d_i)\}_{i=1}^{N} \qquad d = f(\vec{x}) + v$$

We estimate d using a linear exapnsion of non-linear functions $\{\phi_j(\vec{x})\}_{j=0}^{m_1}$

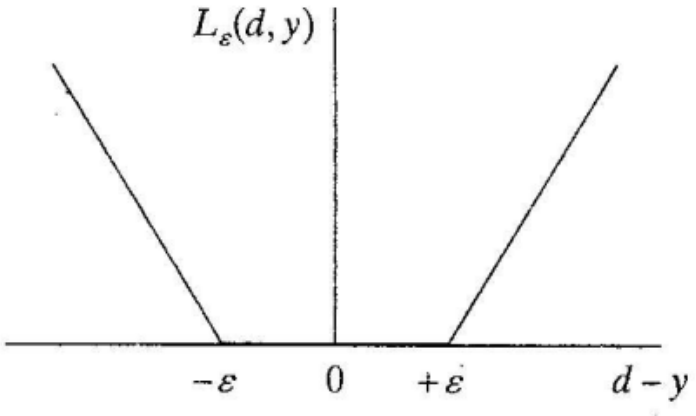
$$y = h(\vec{x}) = \vec{w}^T \phi(\vec{x})$$

Where:

$$\vec{w} = (w(0) = b, w(1), \ldots, w(m_1))^T$$
$$\phi(\vec{x}) = (\phi_0(\vec{x}) = 1, \phi_1(\vec{x}), \ldots, \phi_{m_1}(\vec{x}))^T$$

10.2.1 ϵ-insensitive (soft margin) loss

$$L_\epsilon(d, y) = \begin{cases} |d - y| - \epsilon & \text{if } |d - y| \geq \epsilon \\ 0 & \text{otherwise} \end{cases}$$



10.2.2 Formulating the optimization problem

Introduce the non-negative slack variables ξ_i' and ξ_i $\forall i = 1, \ldots, N$:

$$-\xi_i' - \epsilon \leq d_i - \vec{w}^T \phi(\vec{x}_i) \leq \epsilon + \xi_i \forall i = 1, \ldots, N$$

This lead to the following constraints:

$$\forall i = 1, \ldots, N \begin{cases} d_i - \vec{w}^T \phi(\vec{x}_i) \leq \epsilon + \xi_i \\ \vec{w}^T \phi(\vec{x}_i) - d_i \leq \epsilon + \xi_i' \\ \xi_i \geq 0 \\ \xi_i' \geq 0 \end{cases}$$

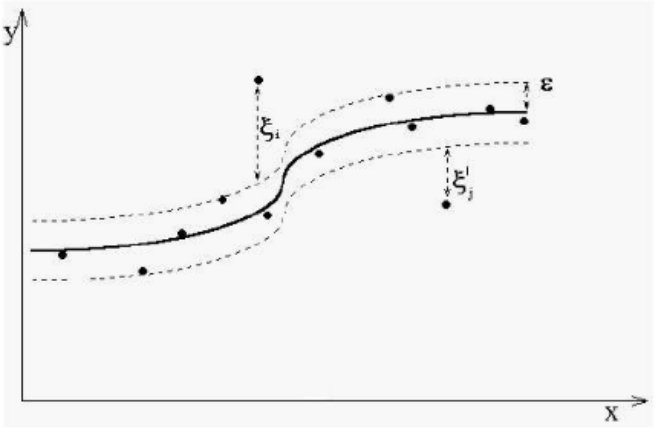


Figure: ϵ-tube and slack variables.

10.2.3 Primal problem

Given the training set $T = \{(\vec{x}_i, d_i)\}_{i=1}^N$ find the optimal values of $\vec{w}$ s.t. the following objective function is minimized:

$$\Psi(\vec{w}, \xi, \xi') = \frac{1}{2}\vec{w}^T\vec{w} + C\sum_{i=1}^{N}(\xi_i + \xi_i')$$

under the constraints

$$\forall i = 1, \ldots, N \begin{cases} d_i - \vec{w}^T\phi(\vec{x}_i) \leq \epsilon + \xi_i \\ \vec{w}^T\phi(\vec{x}_i) - d_i \leq \epsilon + \xi_i' \\ \xi_i \geq 0 \\ \xi_i' \geq 0 \end{cases}$$

10.2.4 Dual problem

Given the training set $T = \{(\vec{x}_i, d_i)\}_{i=1}^N$ find the optimal values of $\{\alpha_i\}_{i=1}^N$ and $\{\alpha_i'\}_{i=1}^N$ which maximize the following objective function:

$$Q(\alpha, \alpha') = \sum_{i=1}^{N} d_i\,(\alpha_i - \alpha_i') - \epsilon\sum_{i=1}^{N}(\alpha_i + \alpha_i') - \frac{1}{2}\sum_{(i,j)=1}^{N}(\alpha_i - \alpha_i')\left(\alpha_j - \alpha_j'\right)k(\vec{x}_i, \vec{x}_j)$$

under the constraints:

$$\forall i = 1, \ldots, N \begin{cases} d_i - \vec{w}^T\phi(\vec{x}_i) \leq \epsilon + \xi_i \\ \vec{w}^T\phi(\vec{x}_i) - d_i \leq \epsilon + \xi_i' \\ \xi_i \geq 0 \\ \xi_i' \geq 0 \end{cases}$$

Remember that C is a user specified parameter.

10.2.5 Computing the weight vector

Solving the dual problem we obtain the optimal values of the Lagrangian multipliers $\{\alpha_i\}_{i=1}^N$ and $\{\alpha_i'\}_{i=1}^N$

Then we can compute the optimal value of vector $\vec{w}$:

$$\vec{w} = \sum_{i=1}^{N} (\alpha_i - \alpha_i') \, \phi(\vec{x}_i) = \sum_{i=1}^{N} \gamma_i \phi(\vec{x}_i)$$

So in this case support vectors correspond to non-zero values of γ_i.

10.2.6 Computing the estimate

Substituting in the estimate function:

$$h(\vec{x}) = y = \vec{w}^T \phi(\vec{x}) = \sum_{i=1}^{N} \gamma_i \phi^T(\vec{x}_i) \phi(\vec{x}) = \sum_{i=1}^{N} \gamma_i k(\vec{x}_i, \vec{x})$$