

NB: Manca un po' di roba

Naive Bayes Classifiers

Operiamo nel campo della probabilità, per cui ricordiamo alcune delle proprietà di questo campo. Il **Bayes Theorem** in particolare è il fondamento teoretico di questo metodo. In quel contesto: Y = l'outcome che ci interessa; X = la feature esaminata.

Chiamiamo la probabilità di X dato Y **Likelihood**, la probabilità di Y **Probabilità a priori** e la probabilità di X **l'evidenza**. Con esse è possibile calcolare la probabilità **a posteriori** che accada Y dato X .

Il problema è che non abbiamo un'unica feature che descrive Y . Come faremo? Il classificatore si chiama **Naive** perché stima Y **assumendo che tutti gli attributi siano condizionalmente indipendenti**. Dovremmo quindi eliminare le probabilità con forte correlazione. **Se questo è vero, la probabilità di osservare tutte le X data Y dovrebbe essere uguale al prodotto di tutte le probabilità di osservare una singola X data Y .**

Si deve anche controllare l'indipendenza di Y . **Se Y è indipendente, possiamo applicare l'algoritmo senza che il risultato sia biased (Condizione d'Indipendenza).** Tuttavia, in situazioni reali è difficile che questo si verifichi.

A probabilistic framework for solving classification problems.
Let P be a probability function that assigns a number between 0 and 1 to events.
 $X = x$ an event is happening.
 $P(X = x)$ is the probability that event $X = x$.
Joint Probability $P(X = x, Y = y)$
Conditional Probability $P(Y = y | X = x)$
Relationship: $P(X, Y) = P(Y|X) P(X) = P(X|Y) P(Y)$
Bayes Theorem: $P(Y|X) = P(X|Y)P(Y) / P(X)$
Another Useful Property: $P(X = x) = P(X=x, Y=0) + P(X=x, Y=1)$

X denotes the attribute sets, $X = \{X_1, X_2, \dots, X_d\}$

Y denotes the class variable

We treat the relationship probabilistically using $P(Y|X)$

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

Learn the posterior $P(Y | X)$ for every combination of X and Y .

By knowing these probabilities, a test record X' can be classified by finding the class Y' that maximizes the posterior probability $P(Y'|X')$.

This is equivalent of choosing the value of Y' that maximizes $P(X'|Y')P(Y')$.

How to estimate it?

It estimates the class-conditional probability by **assuming that the attributes are conditionally independent** given the class label y .

The conditional independence is stated as:

$$P(X|Y = y) = \prod_{i=1}^d P(X_i|Y = y)$$

where each attribute set $X = \{X_1, X_2, \dots, X_d\}$

Given three variables Y, X_1, X_2 we can say that Y is independent from X_1 given X_2 if the following condition holds:

$$P(Y | X_1, X_2) = P(Y | X_2)$$

With the conditional independence assumption, instead of computing the class-conditional probability for every combination of X we only have to estimate the conditional probability of each X_i given Y .

Thus, to classify a record the naive Bayes classifier computes the posterior for each class Y and takes the maximum class as result

$$P(Y|X) = P(Y) \prod_{i=1}^d P(X_i|Y = y) / P(X)$$

How to estimate ?

Naïve Bayes Classifier

- Robust to isolated noise points
- Handle missing values by ignoring the instance during probability estimate calculations
- Robust to irrelevant attributes
- Independence assumption may not hold for some attributes
 - Use other techniques such as Bayesian Belief Networks (BBN, not treated in this course)

Linear Regression

Given a dataset containing N observations $X_i, Y_i, i = 1, 2, \dots, N$

Regression is the task of learning a target function f that maps each input attribute set X into a output Y .

The goal is to find the target function that can fit the input data with minimum error.

The error function can be expressed as

- Absolute Error = $\sum_i |y_i - f(x_i)|$
- Squared Error = $\sum_i (y_i - f(x_i))^2$



Linear regression is a linear approach to modeling the relationship between a **dependent variable** Y and one or more **independent** (explanatory) variables X .

The case of **one explanatory variable** is called **simple linear regression**.

For **more than one explanatory variable**, the process is called **multiple linear regression**.

For **multiple correlated dependent variables**, the process is called **multivariate linear regression**.

L'idea dietro la regressione è ***predire il valore di Y a un certo valore di X***. Ma

cosa significa di preciso? X può avere più valori. Ci riferiamo al valore medio di Y . Notare che la definizione formale di della funzione ricorda Bayes.

Nella **simple linear regression** ($Y = mX + b$) definiamo errori le deviazioni non osservate. Il nostro obiettivo è minimizzare gli errori la SSE. Inizialmente sono posti m e $b = 0$, poi si cercano i valori migliori.

Gli errori, noti anche come **residui**, sono le linee che collegano i puntini rossi alla linea blu, cioè la differenza fra il valore predetto (= la linea) e quello reale (= il puntino).

Sono spesso utilizzate delle tecniche per ridurre gli errori. Il **Ridge** (alias Tikhonov) risponde al problema della **multiple collinearity**, mentre il **Lasso** ha la funzione di *least absolute shrinkage and selection operator*. Il Ridge eleva gli errori alla seconda, il lasso considera il valore assoluto (λ è un parametro che indica quanto vogliamo penalizzare). Lo stesso tipo di regolarizzazione la troveremo dei NN.

Misure di valutazione. Se R^2 più è vicino a 1, meglio è. Può assumere valori negativi e se è 0 significa che non è presente alcuna relazione fra X e Y .

Coefficient of determination R^2

- is the proportion of the variance in the dependent variable that is predictable from the independent variable(s)

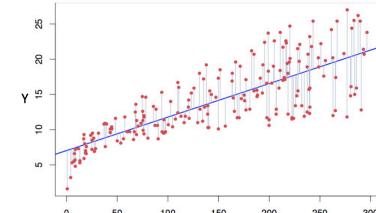
$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad \text{hat means predicted}$$

$\bar{y} = \frac{1}{n} \sum_{i=1}^n y_i$ and $\sum_{i=1}^n (y_i - \bar{y})^2 = \sum_{i=1}^n \epsilon_i^2$

Mean Squared/Absolute Error MSE/MAE

- a risk metric corresponding to the expected value of the squared (quadratic)/absolute error or loss

$$\text{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2 \quad \text{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i|$$



| | |
|----------|---|
| Simple | $\beta_0 + \beta_1 x - y$ |
| Multiple | $\beta_0 + \sum_i (y_i - \beta_i x_i)^2$ |
| Ridge | $\beta_0 + \sum_i (y_i - \beta_i x_i)^2 + \lambda \sum_j \beta_j^2$ |
| Lasso | $\beta_0 + \sum_i (y_i - \beta_i x_i)^2 + \lambda \sum_j \beta_j $ |

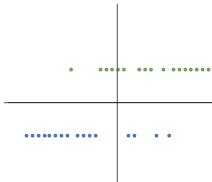
regularization

Logistic Regression

La rigressione logistica risponde ad alcuni problema della rigressione lineare.

Logistic Regression is used to fit a curve to data in which the dependent variable is binary, or dichotomous.

For example: predict the response to treatment, where we might code survivors as 1 and those who don't survive as 0, or pass/fail, win/lose, healthy/sick, etc.



In altre parole, è utilizzata per **risolvere problemi di classificazione non lineari**. Per comprenderla, introduciamo la nozione di *odd* e di *logit*.

$$\text{Odd} = p / (1-p)$$

$$\text{logit}(p) = \ln(\text{odds}) = \ln(p/(1-p))$$

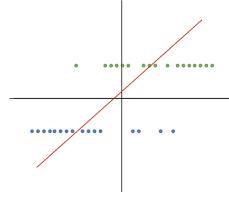
Drawing a line between the means for the two variable levels is problematic in two ways:

- the line seems to oversimplify the relationship,
- it gives predictions that cannot be observable values of Y for extreme values of X .

This is analogous to fitting a linear model to the probability of the event.

Probabilities can only take values in $[0, 1]$.

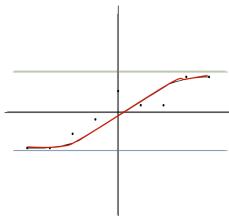
Hence, we need a different approach to ensure that our model is appropriate for the data.



As stated previously, we can model the nonlinear relationship between X and Y by transforming one of the variables.

A common transformation result in **sigmoid functions** is **logit** transformation.

Logit transformations impose a cumulative normal function on the data and are easy to work with because the function can be simplified to a linear equation.



Support Vector Machine

La computazione sta nel trovare il *support vector* per simulare l'imperlano che possa dividere due gruppi di dati.

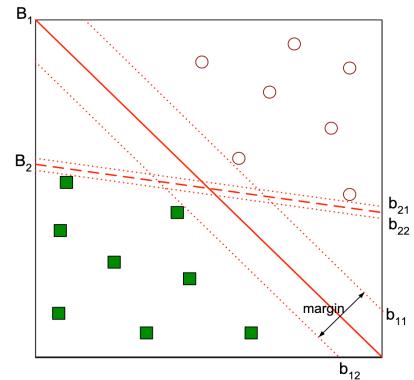
La soluzione migliore è quella che **massimizza il margine** fra i due piani separati (cioè la distanza fra i punti vicini al piano più è grande, meglio è). Nell'immagine a fianco B1 è una divisione migliore di B2. Questo per avere una divisione quanto più netta possibile per permettere poi la generalizzazione del modello.

Il decision boundary è modellato come un vettore w *moltiplicato* un vettore x + un intercept $b = 0$. I margini sono invece = 1 o -1.

La distanza fra un punto e l'imperlano è detta *decision boundary*. Usando la stessa formula (che è quella di distanza fra un punto e una retta), formalizziamo il margine come = $2/w$. Questo perché il margine è la distanza fra i due punti che hanno il ruolo di *support vectors* (i due più vicini appartenenti ad aree diverse). Per massimizzare il margine (cioè la distanza dei due punti SV), dobbiamo quindi minimizzare w .

Per trovare w e b adoperiamo il *Lagrange multiplier* λ , che considera le varie formule viste.

Lagrange introduce λ : il moltiplicatore di Lagrange. Se è 0 il punto non è support vector, altrimenti lo è.



$$f(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} + b \geq 1 \\ -1 & \text{if } \vec{w} \cdot \vec{x} + b \leq -1 \end{cases}$$

$$d(\mathbf{x}) = \frac{|\mathbf{x} \cdot \mathbf{w} + b|}{\sqrt{\|\mathbf{w}\|_2^2}} = \frac{|\mathbf{x} \cdot \mathbf{w} + b|}{\sqrt{\sum_{i=1}^d w_i^2}}$$

Minimize $\|\mathbf{w}\| = \langle \mathbf{w} \cdot \mathbf{w} \rangle$ subject to $y_i(\langle \mathbf{x}_i \cdot \mathbf{w} \rangle + b) \geq 1$ for all i
Lagrangian method : maximize $\inf_w L(w, b, \alpha)$, where

$$L(w, b, \alpha) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_i \alpha_i [(y_i(\mathbf{x}_i \cdot \mathbf{w}) + b) - 1]$$

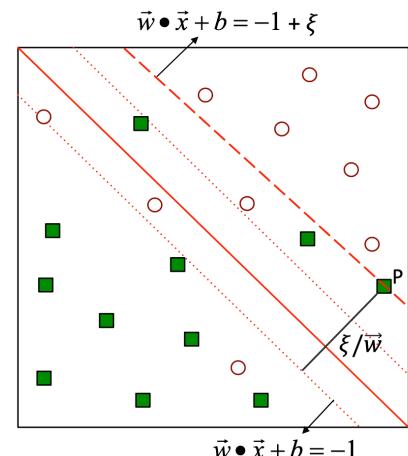
At the extremum, the partial derivative of L with respect both w and b must be 0. Taking the derivatives, setting them to 0, substituting back into L , and simplifying yields :

$$\text{Maximize } \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle$$

$$\text{subject to } \sum_i y_i \alpha_i = 0 \text{ and } \alpha_i \geq 0$$

Tuttavia il problema può non essere *linearmente separabile*. In questo caso sono introdotte delle *slack variables*, che aiutano a stimare gli errori del *decision boundary*. Per sapere quanto un punto è nell'area sbagliato, vediamo la sua distanza dal decision boundary dell'area a cui dovrebbe appartenere: *slack* / w è questa distanza.

La sostanza non cambia, vengono tenute in considerazione anche le *slack variable*, o più specificatamente C e K. Il moltiplicatore di Lagrange diventa qui limitato da C.



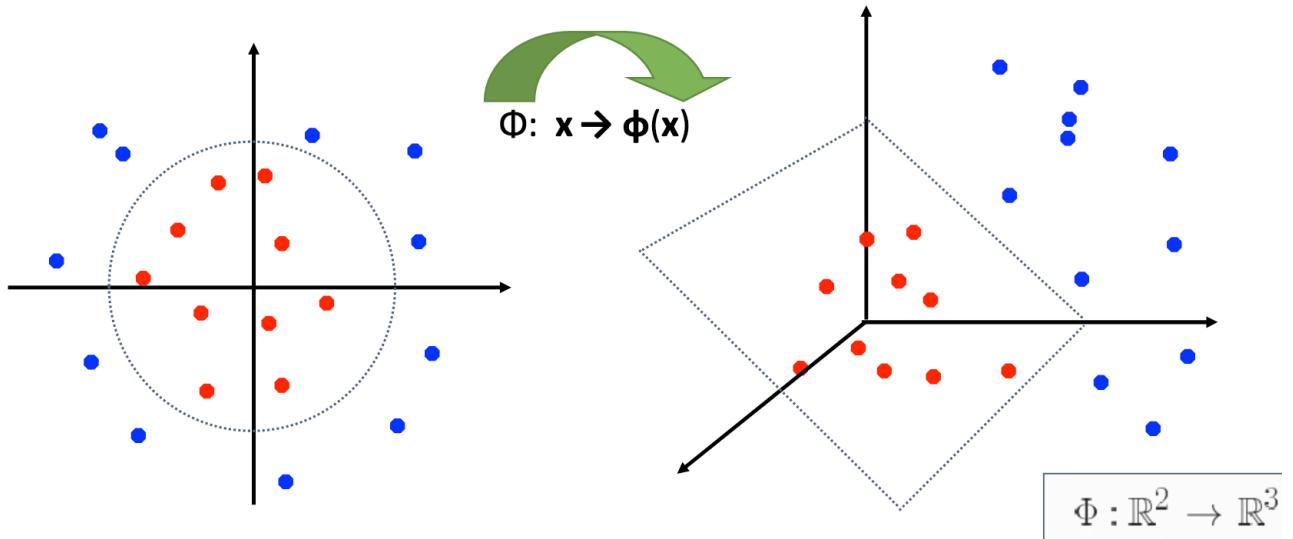
- Objective is to minimize
- Subject to the constraints
- where C and k are user-specified parameters representing the penalty of misclassifying the training instances
- Lagrangian multipliers are constrained to $0 \leq \lambda \leq C$.

$$L(w) = \frac{\|\vec{w}\|^2}{2} + C \left(\sum_{i=1}^N \xi_i^k \right)$$

$$y_i = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x}_i + b \geq 1 - \xi_i \\ -1 & \text{if } \vec{w} \cdot \vec{x}_i + b \leq -1 + \xi_i \end{cases}$$

In sostanza le slack servono a: stimare l'errore del decision boundary per esempi malclassificati e aiutare a individuare esempi malclassificati.

Come comportarsi con situazioni non linearmente separabili in due dimensioni? Si guardano in tre dimensioni e diventa così possibile porre un iperpiano che separa.



Alla formula del decision boundary è aggiunta Φ (*phi*).

$$\text{Decision boundary } \vec{w} \cdot \Phi(\vec{x}) + b = 0$$

Il resto rimane identico.

- Optimization problem
$$\min_{\mathbf{w}} \frac{\|\mathbf{w}\|^2}{2}$$

$$\text{subject to } y_i(\mathbf{w} \cdot \Phi(\mathbf{x}_i) + b) \geq 1, \forall \{(\mathbf{x}_i, y_i)\}$$

- Which leads to the same set of equations but involve $\Phi(x)$ instead of x .

$$f(\mathbf{z}) = \text{sign}(\mathbf{w} \cdot \Phi(\mathbf{z}) + b) = \text{sign}\left(\sum_{i=1}^n \lambda_i y_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{z}) + b\right).$$

In Lagrange avevamo $\mathbf{x}^T \mathbf{x}$. Qui abbiamo $\Phi(\mathbf{x})^T \Phi(\mathbf{x})$ (o per l'esattezza, $\Phi(\mathbf{x})^T \Phi(\mathbf{z})$). Per semplificare:

- $\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}) = K(\mathbf{x}_i, \mathbf{x}_j)$
- $K(\mathbf{x}_i, \mathbf{x}_j)$ is a kernel function
(expressed in terms of the coordinates in the original space)
- Examples:

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^p$$

$$K(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|^2/(2\sigma^2)}$$

$$K(\mathbf{x}, \mathbf{y}) = \tanh(k\mathbf{x} \cdot \mathbf{y} - \delta)$$

Si parla di *Kernel Trick*, perché definiamo la *Kernel Function* (K) che approssima il prodotto di $\Phi(x) * \Phi(x)$ in riferimento alle coordinate dello spazio originale, cioè in base alle dimensioni originali. In questo modo non è necessario mappare Φ e computare $\Phi(x) * \Phi(x)$ nello spazio originale evita la curse of dimensionality (perché rimaniamo nelle dimensioni originali senza aggiungerne una). E non si semplifica troppo l'iniziale formula di Lagrange.

Tuttavia, non tutte le funzioni possono essere kernel. Deve effettivamente esserci un Φ in qualche spazio multi-dimensionale. Questo è vero se si applica il Teorema di Mercer: la funzione deve essere "positivamente definita", cioè $\mathbf{x}^* \mathbf{x}$ deve essere sempre positiva e dunque il problema di ottimizzazione è risolvibile in tempo polinomiale.

Minimize $\|\mathbf{w}\| = \langle \mathbf{w} \cdot \mathbf{w} \rangle$ subject to $y_i(\langle \mathbf{x}_i \cdot \mathbf{w} \rangle + b) \geq 1$ for all i

Lagrangian method: maximize $\inf_{\mathbf{w}} L(\mathbf{w}, b, \alpha)$, where

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_i \alpha_i [y_i(\mathbf{x}_i \cdot \mathbf{w}) + b] - 1$$

At the extremum, the partial derivative of L with respect both \mathbf{w} and b must be 0. Taking the derivatives, setting them to 0, substituting back into L , and simplifying yields:

$$\text{Maximize } \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{subject to } \sum_i y_i \alpha_i = 0 \text{ and } \alpha_i \geq 0$$

- Suppose we have 5 one-dimensional data points

- $x_1=1, x_2=2, x_3=4, x_4=5, x_5=6$, with values 1, 2, 6 as class 1 and 4, 5 as class 2
- $\Rightarrow y_1=1, y_2=1, y_3=-1, y_4=-1, y_5=1$

- We use the polynomial kernel of degree 2

- $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z} + 1)^2$
- C is set to 100

- We first find α_i ($i=1, \dots, 5$) by

$$\begin{aligned} \max. \quad & \sum_{i=1}^5 \alpha_i - \frac{1}{2} \sum_{i=1}^5 \sum_{j=1}^5 \alpha_i \alpha_j y_i y_j (x_i x_j + 1)^2 \\ \text{subject to } & 100 \geq \alpha_i \geq 0, \sum_{i=1}^5 \alpha_i y_i = 0 \end{aligned}$$

Characteristics of SVM

- Since the learning problem is formulated as a convex optimization problem, efficient algorithms are available to find the **global** minima of the objective function (many of the other methods use greedy approaches and find **locally** optimal solutions).
- Overfitting is addressed by maximizing the margin of the decision boundary, but the user still needs to provide the type of kernel function and cost function.
- Difficult to handle missing values.
- Robust to noise.
- High computational complexity for building the model.

Perceptron

I Neural Network si basano su una metafora: i neuroni accettano informazioni da multipli input, e trasmettono l'informazione ad altri neuroni. Si parla spesso di "Black Box" perché il percorso che porta all'outcome non può essere seguito.

Ogni diverso input ha un diverso peso e viene fissata una *unità di bias* (sempre una sola). L'output può essere un valore, unipolare (0, 1) o bipolare (-1, 1). Il peso è molto importante: la procedura d'apprendimento si basa sul compiere errori e correggere gli errori per **minimizzare il costo**. Si inizia con dei pesi random, si prova a computare, si modifica il peso in base alla fase dell'apprendimento in cui siamo e ai risultati ottenuti. Questo finché non c'è più alcun errore.

A seconda del tipo della rete neurale si ha una diversa *activation function*, un modo diverso di calcolare l'output in base all'input.

Nell'esempio che segue vediamo anche l'uso dei pesi.

- Single layer network
 - Contains only input and output nodes
- Activation function: $f = \text{sign}(w \cdot x)$
- Applying model is straightforward

$$Y = \text{sign}(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

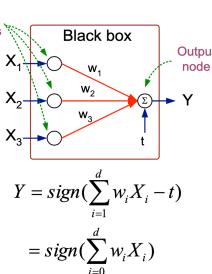
$$\text{where } \text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

$$X_1 = 1, X_2 = 0, X_3 = 1 \Rightarrow y = \text{sign}(0.2) = 1$$

- Model is an assembly of inter-connected nodes and weighted links

- Output node sums up each of its input value according to the weights of its links

- Compare output node against some threshold t (also named bias b)



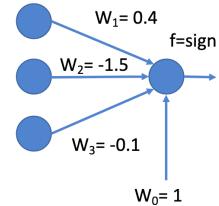
| id | X_1 | X_2 | X_3 | Y |
|----|-------|-------|-------|-----|
| 1 | 0 | 0 | 0 | 1 |
| 2 | 1 | 1 | 1 | -1 |
| 3 | 1 | 0 | 1 | 1 |
| 4 | 0 | 2 | 0 | -1 |

$$Y_1 = \text{sign}(1 + 0.4 * 0 + -1.5 * 0 + -0.1 * 0) = \text{sign}(1) = 1$$

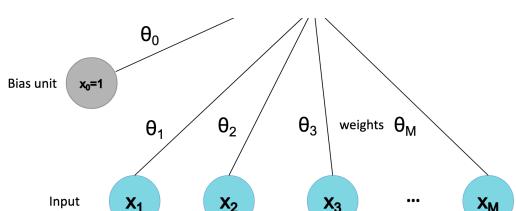
$$Y_2 = \text{sign}(1 + 0.4 * 1 + -1.5 * 1 + -0.1 * 1) = \text{sign}(-0.2) = -1$$

$$Y_3 = \text{sign}(1 + 0.4 * 1 + -1.5 * 0 + -0.1 * 0) = \text{sign}(1.3) = 1$$

$$Y_4 = \text{sign}(1 + 0.4 * 0 + -1.5 * 2 + -0.1 * 0) = \text{sign}(-2) = -1$$



Linear activat
function



- Weight update formula:

$$w^{(k+1)} = w^{(k)} + \lambda [y_i - f(w^{(k)}, x_i)] x_i ; \lambda : \text{learning rate}$$

- Intuition:

- Update weight based on error: $e = [y_i - f(w^{(k)}, x_i)]$
- If $y=f(x,w)$, $e=0$: no update needed
- If $y>f(x,w)$, $e=2$: weight must be increased so that $f(x,w)$ will increase
- If $y<f(x,w)$, $e=-2$: weight must be decreased so that $f(x,w)$ will decrease

Per quanto riguarda la costante di *learning rate*, assume valori da 0 a 1. Più è vicina a 0, più il nuovo peso è vicino ai valori del vecchio peso, più è vicina a 1, più il nuovo peso è influenzato dai *current adjustment*, i calcoli di cui sopra. La differenza fra vecchio e nuovo peso è definita anche *delta*.

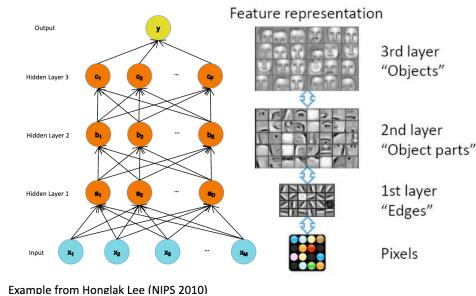
Deep Neural Network

Con il Percettrone alcuni problemi non si possono risolvere - spesso si hanno insieme di dati non linearmente separabili.

La soluzione è costituita da reti neurali multi-layer, che possono risolvere qualsiasi tipo di classificazione. Si può immaginare ciascun layer (nell'immagine n1-n4 ecc.) come singoli percettroni. Per questo motivo si parla anche di *Multilayer Perceptron*.

In ogni hidden layer abbiamo lo stesso funzionamento visto per il percettrone singolo.

Passare da layer a layer serve per andare sempre più nel dettaglio, "scomponendo" sempre più l'input originale, cercando i concetti discriminanti a diversi livelli d'astrazione. Ogni hidden layer corrisponde, in un certo senso, a un diverso livello di apprendimento.



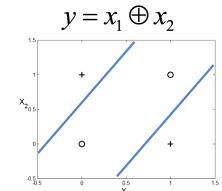
Per quanto riguarda le activation function, di recente si è diffusa la **tangente iperbolica** soprattutto nell'output layer, al posto del sigmoide. La tangente iperbolica ha come range del codominio da -1 a 1, il sigmoide da 0 a 1.

Molto diffuso è anche il RELU (REctified Linear Unit). Ha come output 0 per valori sotto < 0, l'input stesso per valori > 0 (come dire: passato).

Nonlinearly Separable Data

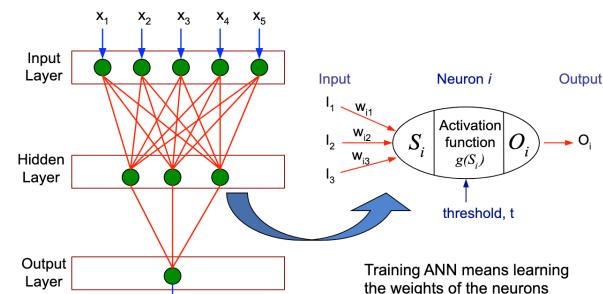
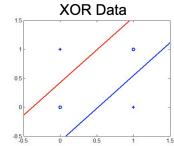
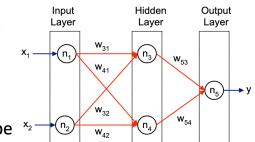
- Since $f(w, x)$ is a linear combination of input variables, decision boundary is linear.
- For nonlinearly separable problems, the perceptron fails because no linear hyperplane can separate the data perfectly.
- An example of nonlinearly separable data is the XOR function.

| XOR Data | | |
|----------------|----------------|----|
| x ₁ | x ₂ | y |
| 0 | 0 | -1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | -1 |

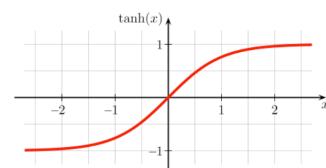


Multilayer Neural Network

- Hidden Layers:** intermediary layers between input and output layers.
- More general **activation functions** (sigmoid, linear, hyperbolic tangent, etc.).
- Multi-layer neural network can solve any type of classification task involving nonlinear decision surfaces.
- Perceptron is single layer.
- We can think to each hidden node as a perceptron that tries to construct one hyperplane, while the output node combines the results to return the decision boundary.

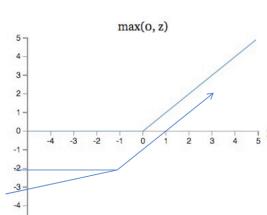
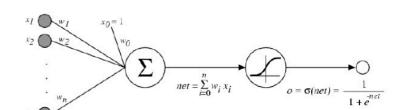


- A new change: modifying the nonlinearity
 - The logistic is not widely used in modern ANNs



Alternative 1:
tanh

Like logistic function but shifted to range [-1, +1]



Alternative 2: rectified linear unit

Linear with a cutoff at zero

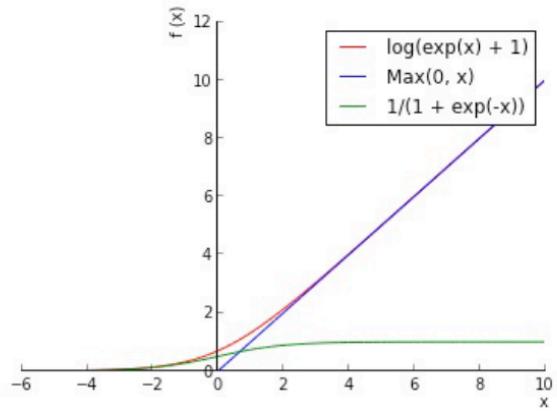
(Implementation: clip the gradient when you pass zero)

$$\max(0, w \cdot x + b).$$

Comparazione fra RELU (in blu), Sigmoid aka Logit (verde) e una versione soft del RELU

—

DA RICORDARE: Backpropagation perché non possiamo avere i reali valori negli Hidden. Come faccio? Stoc Grad Desc method, **passando e aggiornando ogni peso calcolando la derivata del loss rispetto a quel peso.**



Per l'output neurons tutto rimane come nel Percettrone, a cambiare sono gli Hidden.

Disappearing Gradient Descent: se è = 0.

La Loss Function riduce tutto a un singolo numero, è quindi importante scegliere quella giusta per i nostri scopi. Per una Classificazione sono buone la Quadratica (vista nelle slide precedenti) e la Cross-entropy (per classificazioni non binarie).

Regression: A problem where you predict a real-value quantity.

- Output Layer: One node with a linear activation unit.
- Loss Function: Quadratic Loss (Mean Squared Error (MSE))

Classification: Classify an example as belonging to one of K classes

- Output Layer:
 - One node with a sigmoid activation unit (K=2)
 - K output nodes in a softmax layer (K>2)
- Loss function: Cross-entropy (i.e. negative log likelihood)

| $J = E$ | Forward | Backward |
|---------------|---|---|
| Quadratic | $J = \frac{1}{2}(y - y^*)^2$ | $\frac{dJ}{dy} = y - y^*$ |
| Cross Entropy | $J = y^* \log(y) + (1 - y^*) \log(1 - y)$ | $\frac{dJ}{dy} = y^* \frac{1}{y} + (1 - y^*) \frac{1}{y - 1}$ |

Convolutional Neural Network: Usata per immagini e time series. La "convolution" è una moltiplicazione fra l'immagine che viene data in input e il kernel filter. Il filtro viene aggiornato durante l'esecuzione (al posto dei pesi).

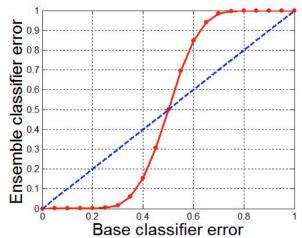
Recurrent Neural Network: Usati in NLP. Hanno in memoria i precedenti stati e influenzano sulla computazione.

Metodi Ensemble

È molto importante l'idea della *Saggezza della Massa (Wisdom of the Crowd)*: la media di tutte le opinioni corrisponde molto alla realtà. Si è notato che l'opinione degli esperti ha una distribuzione normale, laddove quella degli altri ha una distribuzione normale. Questo interessante perché può essere difficile trovare degli esperti (e ancor di più un insieme di esperti) e gli esperti possono essere *biased*. Non qualsiasi folla è tuttavia ideale.

Cosa rende la Crowd Wise? Diversity of Opinion, Independence, Specialized and Local Knowledge di tutti i membri, Aggregation.

Nel nostro caso useremo la Wisdom of the Crowd combinando le predizioni di diversi classificatori. Quando abbiamo visto si estende ai classificatori: la probabilità che un Ensemble sbagli è minore di quella che sbagli ciascun singolo classificatore. Dovremo scoprire come selezionare gli esperti dagli altri. **Tuttavia, se l'errore del singolo classificatore è > 0.5, è sconsigliato usare gli Ensemble: il risultato sarebbe even worse.**



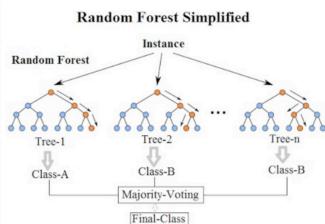
Esercizi a minuto 20 Lezione 1

Come calcolare la probabilità totale di errore, se abbiamo 8 possibili casi e 3 modelli? Controlliamo quali sono i modelli con almeno 2 classificatori che sbagliano (sono 4). Poi, poiché i classificatori sono indipendenti, si moltiplicano le singole probabilità d'errore per ogni modello. Infine, si sommano le 4 probabilità. Nell'esempio si ottiene un errore di 0.3515. Conviene usare l'Ensemble? No, perché il solo Modello 3 ha un errore minore (0.35). In questo caso Bagging non è preferibile all'Expert...

Random Forest

Is a class of ensemble methods specifically designed for **decision trees**.

It combines the predictions made by multiple decision trees and outputs the class that is the mode of the class's output by individual trees.



Each decision tree is built on a **bootstrap sample** based on the values of an **independent** set of random vectors.

- Unlike AdaBoost (see next slides), the random vector are generated from a fixed probability distribution.
- Bagging using decision trees is a special case of random forests where randomness is injected into the model-building process.

Each decision tree is evaluated among **m randomly chosen attributes** from the M available attributes

- $m \sim \sqrt{M}$, or
- $m \sim \log(M)$

Ogni DT lavora su un **bootstrap sample** del dataset originale. Ovvero, è mantenuta la cardinalità nei singoli alberi, ma copiando n volte tot record. Il numero di colonne è ridotto (radice del numero originale, o log del numero originale). Si svolge poi un majority voting. Poiché il DT non richiede normalizzazione, neppure la RF la richiede.

It is one of the most accurate learning algorithms available. For many data sets, it produces a high accurate classifier.

It runs efficiently on large databases.

It can handle thousands of input variables without variable deletion.

It gives estimates of what variables are important in the classification.

It generates an internal unbiased estimate of the generalization error as the forest building progresses.

Bootstrap

- It is a re-sampling statistical technique with re-entry to approximate the sample distribution of a statistic.
- We consider a dataset with n records $X = \{x_1, \dots, x_n\}$.
- From X we re-sample m datasets of constant size equal to n , say $\{X_1^*, \dots, X_m^*\}$.
- In each bootstrap extraction X_i^* , each record in the original dataset has $1/n$ probability of being extracted and can be extracted more than once or zero.

Deep Random Forest: più RF una dopo l'altra per migliorare l'accuracy.

Bagging aka Bootstrap AGGREGATING

Partendo da un dataset, si decidono tot bootstrap sample. Su ciascuna bootstrap sample si applica poi un classificatore (ad esempio il KNN). **Il RF fa la stessa cosa, ma già avendo built-in il DT. In più, effettua anche una dimensionality reduction selezionando un set di colonne. (NO! La fa anche il Bagging)**

Boosting

A differenza del Bagging, ogni iterazione influenza la successiva. All'inizio ogni record ha la stessa probabilità di essere selezionati, poi vengono selezionati solo quelli con i pesi più grandi. I pesi vengono assegnati a seconda di quanto male sono classificati. **Il voto di maggioranza è quindi influenzato dal peso dei singoli sample.**

Adaboost calcola l'*error rate* di ogni singolo classificatore considerando il peso. Una volta calcolato l'*error rate*, la usiamo per derivare l'importanza del singolo classificatore. L'importanza varia 0 a 5, mentre l'importanza da 0 a 1. Si noti che è una *sigmoid function*, solo al contrario (perché è l'input, l' x , che varia da 0 a 1).

Sampling with replacement

| Original Data | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|-------------------|---|---|----|----|---|---|----|----|---|----|
| Bagging (Round 1) | 7 | 8 | 10 | 8 | 2 | 5 | 10 | 10 | 5 | 9 |
| Bagging (Round 2) | 1 | 4 | 9 | 1 | 2 | 3 | 2 | 7 | 3 | 2 |
| Bagging (Round 3) | 1 | 8 | 5 | 10 | 5 | 5 | 9 | 6 | 3 | 7 |

Build classifier on each bootstrap sample

Each sample has probability $(1 - 1/n)n$ of being selected

Algorithm 5.6 Bagging Algorithm

```

1: Let  $k$  be the number of bootstrap samples.
2: for  $i = 1$  to  $k$  do
3:   Create a bootstrap sample of size  $n$ ,  $D_i$ .
4:   Train a base classifier  $C_i$  on the bootstrap sample  $D_i$ .
5: end for
6:  $C^*(x) = \arg \max_y \sum_i \delta(C_i(x) = y)$ ,  $\{\delta(\cdot) = 1 \text{ if its argument is true, and 0 otherwise}\}$ 

```

An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records.

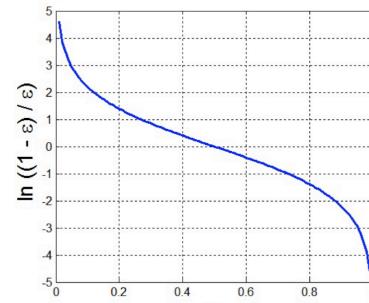
Initially, all the records are assigned equal weights.

Unlike bagging, weights may change at the end of each boosting round.

Records that are wrongly classified will have their weights increased.
Records that are classified correctly will have their weights decreased.

| Original Data | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|--------------------|---|---|---|----|---|---|---|----|---|----|
| Boosting (Round 1) | 7 | 3 | 2 | 8 | 7 | 9 | 4 | 10 | 6 | 3 |
| Boosting (Round 2) | 5 | 4 | 9 | 4 | 2 | 5 | 1 | 7 | 4 | 2 |
| Boosting (Round 3) | 4 | 4 | 8 | 10 | 4 | 5 | 4 | 6 | 3 | 4 |

- Example 4 is hard to classify
- Its weight is increased, therefore it is more likely to be chosen again in subsequent rounds



- Base classifiers: C_1, C_2, \dots, C_T
- Error rate:

$$\varepsilon_i = \frac{1}{N} \sum_{j=1}^N w_j \delta(C_i(x_j) \neq y_j)$$

- Importance of a classifier:

$$\alpha_i = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_i}{\varepsilon_i} \right)$$

High positive importance when error is close to 0,
High negative importance when error is close to 1

Ogni iterazione, AdaBoost aggiorna i pesi. I nuovi pesi sono poi normalizzati in modo tale che la loro somma sia 1 (\Rightarrow possano essere usati come probabilità).

AdaBoost uses **stumps**, i.e., decision tree with only one node and two leaves.

Thus, it uses a forest of stumps.

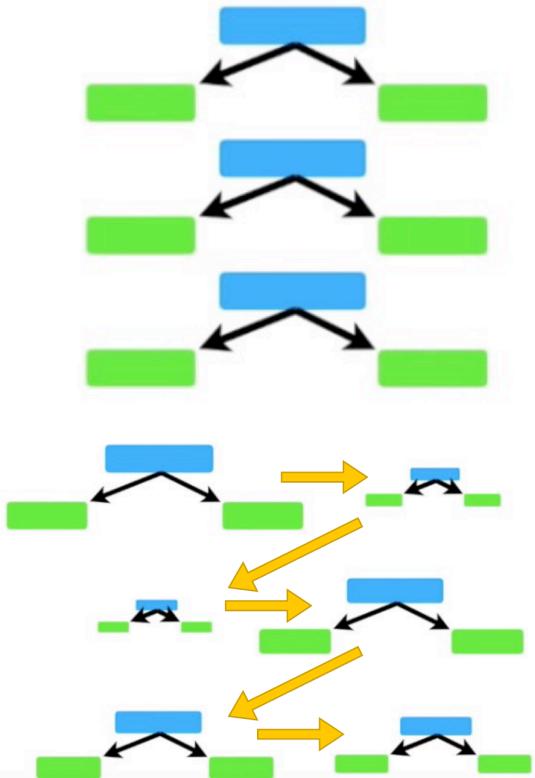
Stumps are not good at making accurate classifications.

Stumps are *weak learners*.

In forest of stumps some stumps have more importance in the final classification.

Each stump **is not** made independently from the others.

The error that the first stump makes influences the second stump and so on and so forth.



Rule based

Mutually exclusive rules

- Classifier contains mutually exclusive rules if the rules are independent of each other
- Every record is covered by at most one rule

Exhaustive rules

- Classifier has exhaustive coverage if it accounts for every possible combination of attribute values
- Each record is covered by at least one rule

- Coverage of a rule:

- Fraction of records that satisfy the antecedent of a rule

- Accuracy of a rule:

- Fraction of records that satisfy the antecedent that also satisfy the consequent of a rule

Rules are not mutually exclusive

- A record may trigger more than one rule
- Solution?
 - Ordered rule set
 - Unordered rule set – use voting schemes

Rules are not exhaustive

- A record may not trigger any rules
- Solution?
 - Use a default class

I Decision Tree sono Mutuali Esclusive (i dati non possono essere in più foglie) ed Esaustive (tutti i record sono classificati). A priori non è nessuno dei due.

Che fare se possiamo applicare più regole?

Ordiniamo le regole e andiamo in ordine. Come ultima chance c'è una *regola default* che applica la classe di maggioranza. L'ordine può essere deciso in base alla qualità della regola o per l'outcome (cioè la classe che viene affibbiata da quella regola: prima tutte le regole che *outputtano* risultato X, poi risultato Y)

Rule-based Ordering

(Refund=Yes) ==> No
(Refund=No, Marital Status={Single,Divorced},
Taxable Income<80K) ==> No
(Refund=No, Marital Status={Single,Divorced},
Taxable Income>80K) ==> Yes
(Refund=No, Marital Status={Married}) ==> No

Class-based Ordering

(Refund=Yes) ==> No
(Refund=No, Marital Status={Single,Divorced},
Taxable Income<80K) ==> No
(Refund=No, Marital Status={Married}) ==> No
(Refund=No, Marital Status={Single,Divorced},
Taxable Income>80K) ==> Yes

Le regole del RIPPER sono entrambe: sono esaustive perché nel peggiore dei casi "in tutti gli altri casi, è classe di maggioranza" e poiché i dati sono eliminati dal dataset sono mutualmente esclusive.

Le regole del RIPPER sono dal generale al particolare, ma poi sono fine-tunate al generale.

Tutto l'ordine

