Apprendimento automatico

Appunti delle lezioni di
Paolo Alfano

Note Legali

Liberatoria, aggiornamenti, segnalazione errori:

Quest'opera viene pubblicata in formato elettronico senza alcuna garanzia di correttezza del suo contenuto. Il documento, nella sua interezza, è opera di Paolo Didier Alfano
e viene mantenuto e aggiornato dallo stesso, a cui possono essere inviate eventuali segnalazioni all'indirizzo *paul15193@hotmail.it*
Ultimo aggiornamento: 23 gennaio 2018

Indice

1 Introduzione al Machine Learning **4**
1.1 Task . . . 5
1.2 Modelli . . . 7
1.3 Features . . . 9

2 Task **10**
2.1 Classificazione . . . 11
2.2 Scoring e Ranking . . . 15
2.3 Stima di probabilità . . . 19
2.4 Oltre la classificazione binaria . . . 22
2.5 Regressione . . . 25

3 Modelli lineari **27**
3.1 Regressione . . . 27
3.2 Support Vector Machine . . . 33
3.3 Errore di margine . . . 38
3.4 Kernel . . . 39

4 Ensemble Learning **42**
4.1 Begging . . . 43
4.2 AdaBoost . . . 44
4.3 Perché l'ensemble funziona . . . 47

27 / 09 /2017

1 Introduzione al Machine Learning

Il *machine learning* può essere grossolanamente rappresentato tramite tre componenti:

1. Task: una descrizione generale di quello che vogliamo fare
2. Modelli: il modo in cui risolviamo il problema
3. Features: come denotiamo la descrizione dell'oggetto stesso. Ad esempio, in un problema di classificazione, possiamo dare una descrizione numerica di quello che intendiamo classificare

Esempio 1 : un tipico esempio di problema nel machine learning è l'individuazione delle mail di spam. In questo caso il task è *individuare lo spam.*
Un noto strumento di eliminazione dello spam è SpamAssassin. Questo programma funziona come un filtro ed esegue una serie di test da cui ottiene una risposta booleana. Dopo aver eseguito tutti i test ne combiniamo i risultati usando una funzione lineare per ottenere una risposta. Consideriamo per comprendere meglio la Tabella 4

E-mail	x_1	x_2	Era Spam?	$4x_1 + 4x_2$
1	1	1	Sì	8
2	0	0	No	0
3	1	0	No	4
4	0	1	No	4

Tabella 1: La tabella relativa allo spam

Dove consideriamo quattro mail, una per ogni riga, sulle quali eseguiamo due test x_1 e x_2. Supponendo di star eseguendo una fase di apprendimento del programma, siamo a conoscenza anche se le mail ricevute fossero effettivamente mail di spam. Nell'ultima colonna riportiamo invece il risultato ottenuto dalla funzione lineare dei due test eseguiti.
Notiamo che per classificare una mail come spam dobbiamo stabilire un valore della funzione lineare oltre il quale la mail viene classificata come spam. In questo caso, configurando a cinque tale valore di soglia, otterremmo il risultato che vogliamo: la prima mail viene considerata spam ma le altre no.
Il compito del machine learning è quello di stabilire il valore da attribuire ai coefficienti della funzione lineare. Questo compito può essere definito come il *learning problem.*

Nel seguito vedremo che problemi di questo tipo sono detti di *classificazione lineare.* Li vedremo meglio nel seguito.

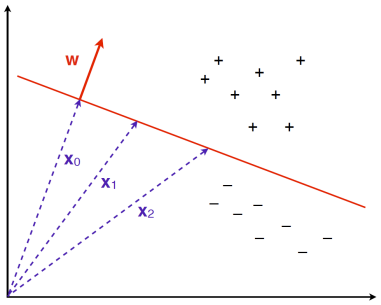

Figura 1: Rappresentazione di un modello predittivo lineare

Una prima definizione informale di machine learning che potremmo dare è la seguente: il machine learning è lo studio sistematico di quegli algoritmi che migliorano le loro prestazioni all'aumentare dell'esperienza.
Vediamo il significato dei termini *migliorano*, *prestazioni* ed *esperienza* riferendoci all'esempio di prima. Nel caso delle mail l'esperienza è l'insieme delle mail che ci vengono fornite. Le prestazioni ottenute riguardano la correttezza della classificazione eseguita e il miglioramento è relativo a tale prestazione.

Un fatto importante da ricordare è che i modelli rappresentano il modo di risolvere certi task, mentre gli algoritmi di learning risolvono dei problemi di learning. Inoltre gli algoritmi di learning istanziano i modelli.
Complessivamente diciamo che il machine learning riguarda l'utilizzo delle giuste feature per costruire i giusti modelli che raggiungano i giusti task.

1.1 Task

Esistono due tipi di task che possiamo cercare di risolvere:

- Predictive tasks: quando cerchiamo di predire il valore di una certa variabile. Se stiamo facendo una classificazione -binaria o multiclasse- diremo che la variabile è categorica. Nel caso si effettui un procedimento di regressione avremo una variabile numerica. Infine se effettuiamo un procedimento di clustering abbiamo una variabile nascosta.

- Descriptive tasks: quando vogliamo cercare di comprendere qualcosa riguardo i dati senza fare alcun tipo di predizione

Concentriamoci sui modelli predittivi. Un modello predittivo cerca di risolvere il problema di inferire alcune conoscenze riguardo un nuovo problema basandosi su istanze precedenti dello stesso problema.
Consideriamo la Figura 1

In questo caso una nuova istanza che si collochi in alto a destra nel grafico, verrà classificata come positiva in base al nostro modello lineare. Nel seguito

definiremo con maggiore precisione i modelli lineari.
Più importante è invece il concetto dell'*overfitting*. Supponiamo di memorizzare i risultati forniti dagli esempi su cui il programma ha fatto esperienza, detto *training set*. Su nuove istanze identiche alle precedenti sapremo alla perfezione la risposta. Costruire un modello che ottenga una prestazione perfetta sul training set ma che ottenga una pessima prestazione su nuovi esempi, è solitamente legato ad un problema di overfitting. Solitamente una tecnica di machine learning non memorizza i dati proprio per evitare l'overfitting. Comprendiamolo meglio tramite un esempio:

Esempio 1 : supponiamo di preparare un esame con un metodo di studio nuovo: imparare a memoria tutte le risposte dei precedenti esami senza studiare i contenuti del corso. In generale, se durante il nostro esame riceveremo delle domande incontrate precedentemente, sapremo perfettamente la risposta. D'altro canto, ricevendo domande differenti dalle precedenti ma inerenti gli stessi argomenti(scenario tipico di un esame) non sapremo cosa rispondere, ottenendo un risultato pessimo. In questo caso si è verificato un overfitting sugli esami passati che non ci ha permesso di generalizzare e comprendere le future domande.

Uno dei compiti del machine learning è quello di evitare l'overfitting. Riprenderemo in seguito il concetto di overfitting

Parliamo invece dei task descrittivi. Facciamolo tramite l'esempio che segue

Esempio 2 : vogliamo cercare di descrivere le preferenze dell'utenza riguardo i film. Le preferenze vengono mostrate con la Tabella 2

Utente	Le ali della libertà	I soliti sospetti	Il padrino	Il grande Lebowsky
U_1	1	0	1	0
U_2	0	2	2	2
U_3	0	0	0	1
U_4	1	2	3	2
U_5	1	0	1	1
U_6	0	2	2	3

Tabella 2: Gradimento degli utenti in base al film

In questo caso, ricavare uno schema da questa matrice potrebbe essere complicato. Potremmo invece cercare di descrivere il modello utilizzando le tre matrici che seguono

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 \\ 1 & 2 & 3 & 2 \\ 1 & 0 & 1 & 1 \\ 0 & 2 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Dove la prima matrice è il risultato già mostrato in Tabella 2, la seconda rappresenta una su ogni riga gli utenti e su ogni colonna il genere dei film, rispettivamente i generi sono drammatico, crimine e commedia. La seconda matrice è invece inferita dal programma mentre la terza indica per ogni riga i tre generi sopracitati e per ogni colonna uno dei quattro film. Ad esempio il grande Leboswky(ultima colonna della terza matrice) è un film di genere crimine e commedia, ma non drammatico. Supponendo di avere queste informazioni a disposizione il programma di learning può cercare di intuire i valori della seconda matrice, che ci permetteranno di comprendere meglio le preferenze degli utenti.
Solitamente però non capita di avere indizi riguardo le matrici che otteniamo dalla fattorizzazione. Questo rende più complesso l'intero processo.

Finora abbiamo visto che i problemi di learning possono essere predittivi o descrittivi. Un'altra suddivisione possibile è *supervised* o *unsupervised*:

- Supervised: abbiamo delle informazioni riguardo la natura del dominio che stiamo studiando. Nell'esempio della mail sapevamo prima se la mail era effettivamente spam.
- Unsupervised: non abbiamo informazioni riguardo il dominio

Per categorizzare in modo completo i problemi di learning possiamo osservare la Tabella 3

	Predictive Model	Descriptive Model
Supervised	classificazione, regressione	Scoperta di sottogruppi
Unsupervised	clustering predittivo	clustering descrittivo, scoperta di regole di associazione

Tabella 3: Categorizzazione del machine learning

1.2 Modelli

Attualmente i tre modelli più popolari per l'apprendimento automatico sono:

- Geometrico: cerchiamo di collocare i dati in un qualche spazio geometrico dimensionale e cerchiamo di inferire una proprietà geometrica comune ai dati.
- Probabilistico: consiste nel rappresentare i dati come punti e nell'assegnare un certo valore di probabilità ai dati e fare una qualche inferenza probabilistica.
- Logico: modelli costruiti in termini di formule logiche. Non abbiamo punti ma una serie di formule logiche che ci permettono di inferire informazioni.

Abbiamo parlato precedentemente riguardo i modelli geometrici, spendiamo qualche parola sui modelli probabilistici. Questi modelli cercano di stimare valori di probabilità dai dati. In base ad essi vengono fatte delle previsioni usando una qualche regola di decisione.
Un esempio di classificatore probabilistico potrebbe essere il seguente: supponendo di sapere che un evento Y dipenda da un certo evento X, potremmo chiederci quale sia la probabilità che si verifichi Y sapendo che si è verificato X. Ovvero:

$$Y_{MAP} = arg\ max_y P(Y|X)$$

Dalla legge di Bayes sappiamo che $P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$. Inoltre se dobbiamo massimizzare Y, possiamo sempre trascurare il valore di $P(X)$ e, sotto certe condizioni[1], possiamo anche trascurare la probabilità $P(Y)$.
Dunque un classificatore potrebbe essere:

$$Y_{ML} = arg\ max_y P(X|Y)$$

Ovviamente non è detto che lo stimatore si comporti sempre bene. Vediamo tutto questo tramite un esempio

Esempio 1 : supponiamo di voler stimare se una mail ricevuta sia spam. Sappiamo che quando tale mail contiene le parole "viagra" o "lotteria" potrebbe essere spam. Inoltre sappiamo che a priori il 30% delle mail sono di spam. Supponiamo di avere a disposizione i seguenti dati: Supponendo

X_1 =viagra	X_2 =lotteria	$P(X_1X_2\|spam)$	$P(X_1X_2\|\neg spam)$	$P(X_1, X_2)$
0	0	0.033	0.857	0.61
0	1	0.067	0.086	0.08
1	0	0.233	0.043	0.1
1	1	0.667	0.014	0.21

Tabella 4: Dati dell'esempio 1

che una mail viene reputata spam se la probabilità che lo sia è maggiore di 0.5, potremmo provare a chiederci $P(spam|X_1, X_2)$ utilizzando prima la legge di Bayes, poi lo stimatore. I risultati ottenuti sono i seguenti:

X_1	X_2	$P(spam\|X_1, X_2)$	Risultato
0	0	0.016	not spam
0	1	0.25	not spam
1	0	0.7	spam
1	1	0.95	spam

Tabella 5: Risultato utilizzando la legge di Bayes

[1] Quando abbiamo molti dati a disposizione

X_1	X_2	$P(spam\|X_1, X_2)$	Risultato
0	0	0.033	not spam
0	1	0.067	not spam
1	0	0.233	spam
1	1	0.667	spam

Tabella 6: Risultato utilizzando il classificatore probabilistico

Notiamo come i valori ottenuti utilizzando Bayes e il classificatore siano diversi, ma che comunque la classificazione sia invece identica.

28 / 09 /2017

Notiamo comunque che l'esempio precedente presenta un inconveniente: in presenza di n variabili dobbiamo avere a disposizione una tabella avente 2^n righe. Questo approccio risulta inefficace. Risulta più utile costruire una tabella più semplice sfruttando la *Naive Bayes Assumption*(letteralmente l'assunzione bayesiana ingenua). Con questa assunzione sosteniamo che le variabili X_1 e X_2 siano indipendenti, semplificando molto i conti.

Spendiamo solo qualche parola per i classificatori logici, riprendendo l'esempio di prima

Esempio 2 : un classificatore logico, nel caso in cui una mail contenga la parola viagra, stimerà la probabilità di spam come $4:1$, se la mail contiene la parola "pillola blu" stimerà la probabilità come $3:1$, altrimenti stima la probabilità come $1:6$

1.3 Features

Per rendere un sistema di apprendimento funzionale ai compiti che deve svolgere, dobbiamo lavorare sulle feature. Le feature possono essere pensate in due modi differenti: da una parte descrivono i dati e permettono di computare il modello lineare. Dall'altra le feature sono strumenti da usare per migliorare il nostro sistema. Cerchiamo di capirlo meglio tramite qualche esempio:

Esempio 1 : supponiamo di voler approssimare $y = cos(\pi x)$ sull'intervallo $-1 \leq x \leq 1$.
In questo caso un'approssimazione lineare non è di grande aiuto visto che otterremmo $y = 0$. È preferibile invece utilizzare due approssimazioni differenti: una per l'intervallo $-1 \leq x \leq 0$ e un'altra per l'intervallo $0 \leq x \leq 1$, cercando delle approssimazioni su ogni intervallo. Il risultato è mostrato in Figura 2

Esempio 2 : in certi casi è possibile che un classificatore lineare si comporti male su un certo insieme di dati. A volte può risultare conveniente mappare i dati in uno spazio di dimensione superiore(geometricamente parlando)

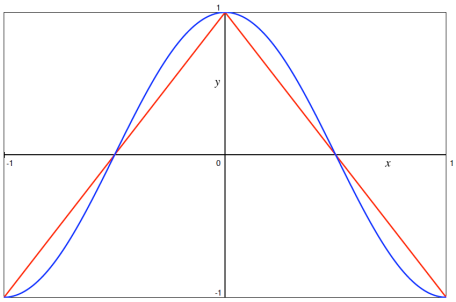

Figura 2: Approssimazione di $y = cos(\pi x)$ in due intervalli

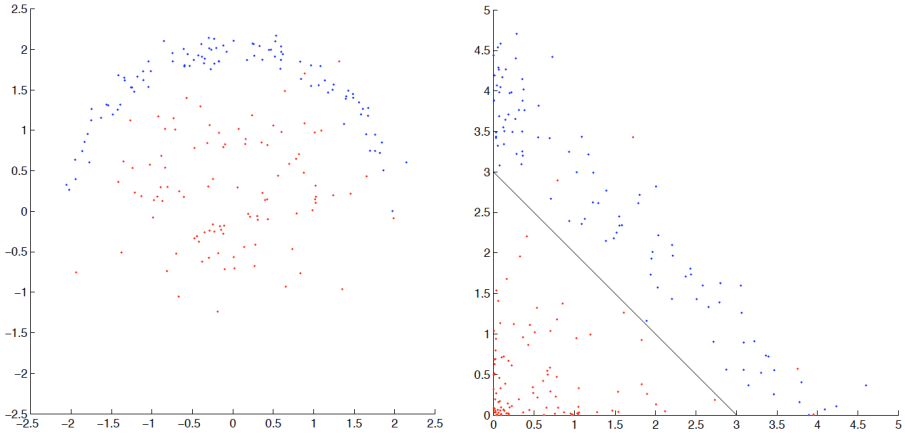

Figura 3: A sinistra un classificatore lineare per un set di dati della forma (x, y). A destra gli stessi dati elevati al quadrato

per ottenere una maggiore dispersione. Questa situazione viene mostrata in Figura 3

Notiamo come i dati nella forma (x, y) vengano separati tramite una circonferenza di raggio $\sqrt{3}$, mentre i dati della forma $(x', y') = (x^2, y^2)$ possano essere separati molto più comodamente mediante la retta

$$y = -x + 3$$

Questa tecnica, viene detta *kernel trick*

2 Task

Vediamo più nel dettaglio i task e le diverse tipologie di task. Abbiamo detto che i predictive task cercano di prevedere il valore di dati futuri basandosi sui

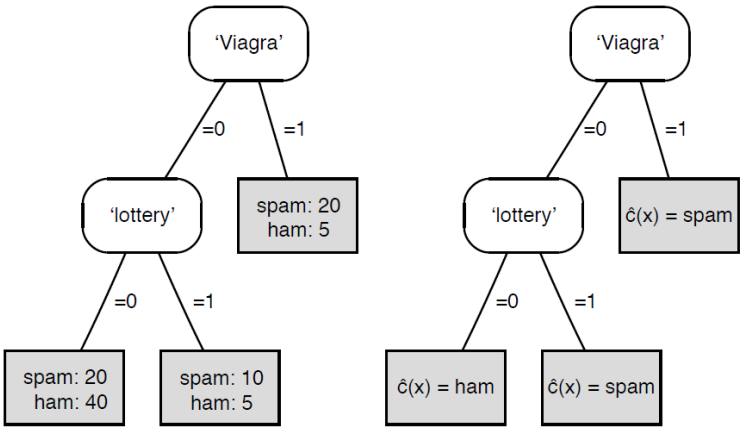

Figura 4: A sinistra un albero delle feature e a destra un albero di decisione. Entrambi per l'esempio del classificatore di spam

dati visti precedentemente. A seconda delle classi in cui è suddiviso il dataset e le proprietà che cerchiamo di predire, possiamo dividere i task in quattro tipi differenti: *classificazione*, *scoring and ranking*, *stima di probabilità* e infine *regressione*.

2.1 Classificazione

Un classificatore è una funzione approssimata del tipo

$$\hat{c} : \mathfrak{X} \to \mathfrak{C}$$

dove $\hat{c}$ indica un'approssimazione della reale funzione di classificazione c, $\mathfrak{X}$ è un'insieme di dati mentre $\mathfrak{C} = \{C_1, \ldots, C_k\}$ è un'insieme di classi in cui il dato può essere categorizzato.
Dunque affinché un classificatore apprenda, dobbiamo riuscire a costruire una funzione $\hat{c}$ che sia quanto più possibile simile a c.

L'esempio più semplice di classificazione è la *classificazione binaria*. In questo caso l'insieme $\mathfrak{C}$ è composto da due soli elementi. Dunque i dati sono classificabili solamente tra due categorie. Per poter creare un buon classificatore binario dobbiamo stabilire come valutarlo. Solitamente questo viene fatto andando a creare due alberi di decisione. Il primo viene detto proprio *albero di decisione*, il secondo invece viene detto *albero delle feature*.
L'albero delle feature è costruito a partire dalle variabili utilizzate per il problema. Un esempio di albero delle feature e di un albero di decisione viene mostrato in Figura 4

Notiamo come l'*accuratezza* del classificatore sia data dal numero di mail classificate correttamente. Se ad esempio 70 mail venissero classificate correttamente, allora avremmo $a = 70\%$. Per avere una visione più chiara di questo

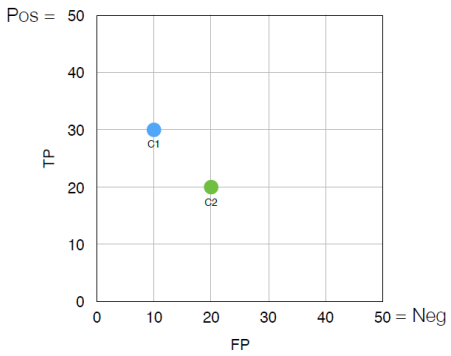

Figura 5: Un coverage plot che mostra i risultati ottenuti da due classificatori

può anche essere usata una *tabella di contingenza* in cui indichiamo sulle righe gli effettivi valori dei dati mentre sulle colonne le predizioni da parte del nostro classificatore. Mostriamo una tabella di contingenza in Tabella 7

	Predizione +	Predizione −
Effettivi +	30	20
Effettivi −	10	40

Tabella 7: Tabella di contingenza

Come dicevamo, l'accuratezza in questo caso è del 70%.
Assumendo che esistano un'insieme di dati effettivamente positivi e negativi, possiamo suddividere i dati in base alla predizione ottenuta utilizzando il nostro classificatore: *veri positivi*, *veri negativi*, *falsi negativi* e *falsi positivi*. In base al numero di questi possiamo effettivamente valutare le prestazioni di un classificatore. Ad esempio l'accuratezza è la somma dei veri positivi e dei veri negativi diviso il numero di elementi totali. Oppure la *precisione* o *confidenza* è data dal numero di veri positivi diviso la somma di veri positivi e falsi positivi...

02 / 10 /2017

Uno strumento che possiamo utilizzare per valutare l'efficacia di uno stimatore è quello dei *coverage plot*.
I coverage plot sono dei grafici che collocano sulle ascisse i falsi positivi mentre sulle ordinate i veri positivi. Un esempio di coverage plot viene mostrato in Figura 5

Notiamo che un buon classificatore si colloca nella parte alta a sinistra del grafico. Infatti in quel caso abbiamo un elevato numero di veri positivi e un basso valore di falsi positivi. Tale area del grafico viene detta *ROC heaven*.
Inoltre dobbiamo considerare che il peggior classificatore possibile si trova nell'angolo opposto. Ad ogni modo anche la parte centrale del grafico non è molto

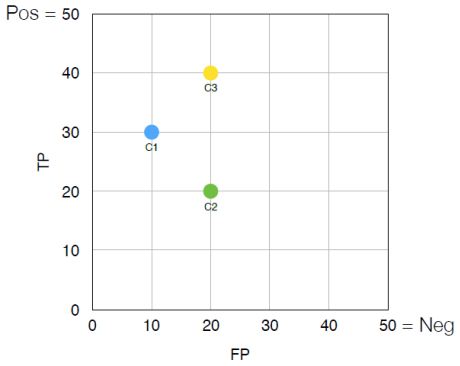

Figura 6: Un coverage plot in cui più di un classificatore può andar bene

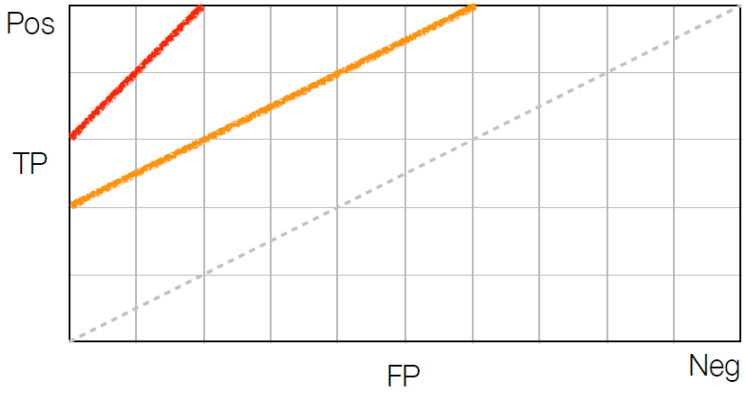

Figura 7: Un coverage plot "rettangolare"

buona visto che quando abbiamo un numero eguale di veri positivi e falsi positivi il nostro classificatore sbaglia la metà delle volte! Allora tanto vale usare una moneta...
Supponiamo invece di avere il caso mostrato in Figura 6

Potremmo chiederci quale sia il miglior classificatore. Certamente non C2 che possiede un eguale numero di veri positivi e falsi positivi. Come decidere tra C1 e C3? In generale, se non abbiamo ulteriori direttive possiamo usare indifferentemente uno o l'altro. Infatti in base alla definizione di *accuratezza* fornita precedentemente otteniamo lo stesso valore.
Un fatto importante da tenere presente è che sebbene gli assi indichino il numero di falsi positivi e veri positivi, sono limitati superiormente rispettivamente dal numero di negativi e dal numero di positivi. Dunque il grafico può avere una forma "rettangolare".
Questo fatto viene mostrato in Figura 7

Questo fatto è importante perché:

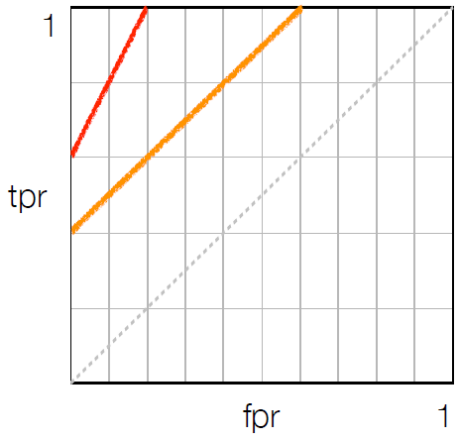

Figura 8: Il ROC plot ottenuto normalizzando i dati mostrati in Figura 7

- Classificatori che giacciono su una stessa retta avente $m = 1$ possiedono stessa *accuratezza* secondo la definizione fornita precedentemente.
- Classificatori che giacciono su una stessa retta parallela alla "diagonale" del coverage plot possiedono stessa *esaurienza media*. Dove l'esaurienza media è:

$$avgrecall = \frac{(recall + specificity)}{2} = \frac{\frac{TP}{POS} + \frac{TN}{NEG}}{2} \quad (1)$$

Esiste anche un altro tipo di coverage plot, che viene detto *ROC plot* (dove ROC sta per receiver operating characteristic). Di fatto è simile al precedente visto che semplicemente gli assi sono normalizzati. Mostriamo in Figura 8 un esempio di ROC plot

Un problema legato al ROC plot è la perdita di informazione. Supponendo di avere il grafico mostrato in Figura 8, non siamo a conoscenza del valore

$$\frac{NEG}{POS}$$

che è di fondamentale importanza per capire se un classificatore sia meglio dell'altro. Avendo a disposizione tale rapporto possiamo tracciare la retta passante per ogni classificatore e determinare quale sia la più vicina al ROC heaven.

Un discorso simile a quello appena fatto coinvolge le *funzioni di costo*. Infatti non tutte le situazioni di errore sono tollerabili. Vediamolo con il prossimo esempio.

Esempio 1 : supponiamo di effettuare una serie di esami per verificare la presenza di un tumore in un certo numero di soggetti. Nel caso di falsi positivi

non abbiamo grossi problemi. Infatti un utente segnalato come malato può essere sottoposto ad ulteriori controlli. Invece il caso dei falsi negativi è più preoccupante perché può portare alla morte del paziente.

In casi come quello appena mostrato vogliamo cercare di controllare il numero di falsi negativi. Supponendo di stabilire che il costo di un falso positivo sia $cost(FP) = 1$ mentre il costo di un falso negativo sia $cost(FN) = 100$ possiamo utilizzare una funzione di costo

$$c = \frac{cost(FN)}{cost(FP)} \tag{2}$$

Similmente a quanto detto prima possiamo tracciare per ogni possibile classificatore la retta avente coefficiente angolare $1/c$ e determinare quale retta sia più vicina al ROC heaven.

2.2 Scoring e Ranking

Partiamo dal sistema di scoring. È un sistema basato su punteggi in cui abbiamo una funzione

$$\hat{s} : \mathfrak{X} \to \mathbb{R}^k$$

il risultato ottenuto è dunque un vettore $\hat{s}(x) = (\hat{s}_1(x), \ldots, \hat{s}_k(x))$ a k componenti dove ogni componente rappresenta la valutazione ricevuta dal dato in input sulla classe relativa alla componente.
Il principale vantaggio che abbiamo utilizzando un classificatore a punteggio è che ci fornisce maggiori informazioni. Riprendiamo la Figura 1. Un semplice classificatore binario restituisce risposta positiva o negativa. Nel caso di un classificatore a punteggio invece possiamo effettuare una distinzione tra i vari punti. Ad esempio potremo essere maggiormente certi della "positività" dell'elemento in alto a destra rispetto agli elementi più vicini alla linea che separa le classi.
Anche in questo caso possiamo costruire un feature tree e uno scoring tree, come mostrato in Figura 9

Associato al vettore di scoring abbiamo il concetto di *margine*

$$z(x) = \begin{cases} +|\hat{s}(x)|, & \text{se } \hat{s} \text{ è corretto su x} \\ -|\hat{s}(x)|, & \text{altrimenti} \end{cases} \tag{3}$$

Il concetto di margine è facilmente comprensibile con l'esempio che segue

Esempio 1 : supponiamo di avere stabilito due diverse classificazioni per uno stesso insieme di dati. Mostriamo i due classificatori in Figura 10

Notiamo come entrambi i classificatori siano corretti ma come il primo sia peggiore poiché meno stabile. La minore stabilità deriva dalla minore ampiezza del margine.

Per stabilire quale margine sia migliore di un altro usiamo una *loss function* L della forma:

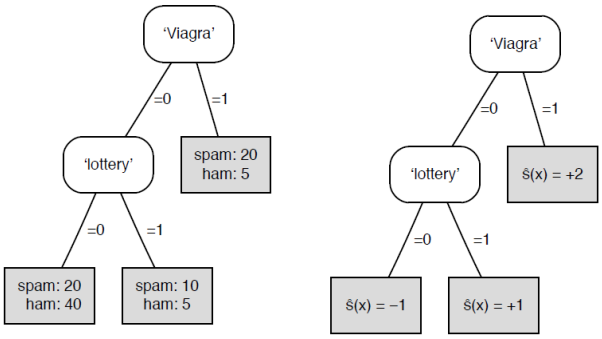

Figura 9: A sinistra l'albero delle feature, a destra l'albero dei punteggi

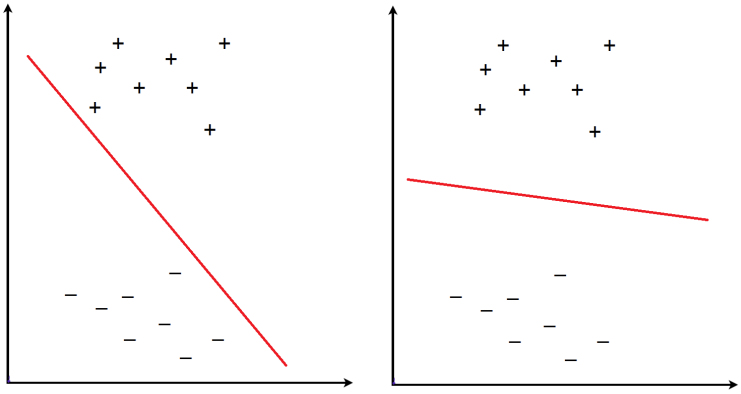

Figura 10: Due diversi classificatori. Quello a destra è migliore poiché ha un margine maggiore

$$L : \mathbb{R} \to \mathbb{N}^+$$

l'input ricevuto dalla funzione è la larghezza del margine. Questa funzione di loss è estremamente importante perché in molti algoritmi di learning permette di valutare la bontà della classificazione. Il margine può anche essere considerato negativo nel caso di errata classificazione. Nel complesso, il grafico della funzione loss viene mostrato in Figura 11

Quindi questa valutazione si riduce alla funzione di loss che utilizziamo. Ne abbiamo di diverso tipo

- 0-1 Loss: $L_{01}(z) = \begin{cases} 1, & \text{se } z \leq 0 \\ 0, & \text{se } z > 0 \end{cases}$
  Quando il margine è negativo a causa di errori di classificazione otteniamo una funzione di loss avente valore maggiore.

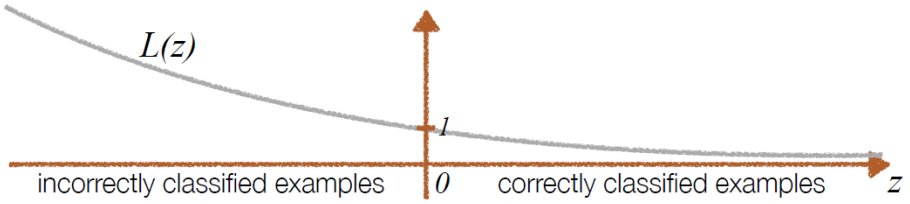

Figura 11: Loss function

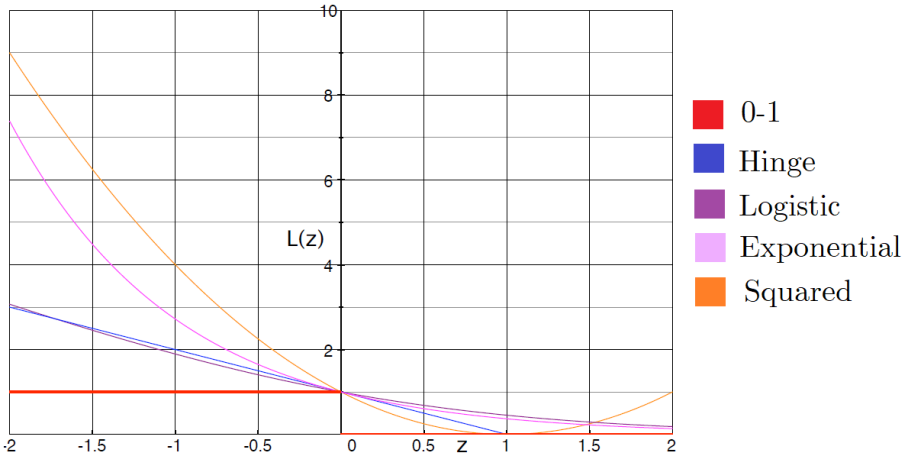

Figura 12: Comparazione fra le varie funzioni di loss.

- Hinge Loss: $L_h(z) = \begin{cases} (1-z), & \text{se } z \leq 1 \\ 0, & \text{se } z > 1 \end{cases}$
- Logistic Loss: $L_{log}(z) = log_2(1 + e^{-z})$
- Exponential Loss: $L_{exp}(z) = e^{-z}$
- Squared Loss: $L_{sq}(z) = (1-z)^2$

Riportiamo in Figura 12 il confronto fra le varie funzioni di loss

Adesso prendiamo invece in considerazione le tecniche di ranking. Cominciamo dicendo che vengono utilizzate delle tecniche di ranking quando non ci importa del punteggio effettivamente ottenuto dai dati ma dell'ordinamento dei dati in base al punteggio.
Notiamo anche che se abbiamo a disposizione una funzione di punteggio, abbiamo anche una funzione di ranking poiché ci è sufficiente ordinare i dati in base al punteggio ottenuto per ottenere un ranking.

Assumendo che punteggi alti esprimono una maggiore probabilità che un'istanza sia positiva, se vogliamo effettuare una classificazione in base al ranking spereremmo, dopo aver ordinato sul punteggio, di ottenere una serie di dati del tipo

$$+, +, \ldots, +, -, -, \ldots, -$$

dove il simbolo + indica un dato classificato positivamente mentre il simbolo − un dato classificato negativamente. Notiamo che prima abbiamo tutti i dati classificati positivamente, seguiti da tutti i dati classificati negativamente.
Per poter effettuare una valutazione della classificazione utilizziamo la seguente formula di *ranking error rate*

$$rank - err = \frac{\sum_x I(\hat{s}(x) < \hat{s}(x')) + \frac{1}{2} I(\hat{s}(x) = \hat{s}(x'))}{POS * NEG} \tag{4}$$

dove I è la funzione indicatore che restituisce 1 se il suo argomento è vero, restituisce 0 altrimenti. Intuitivamente, quello che stiamo dicendo è che se un punto che dovrebbe un punteggio inferiore ha invece un punteggio superiore a quello reale, allora otteniamo un punto di penalità dal membro $I(\hat{s}(x) < \hat{s}(x'))$. Se due punti classificati diversamente ottengono lo stesso punteggio, allora riceviamo mezzo punto di penalità dal membro $\frac{1}{2} I(\hat{s}(x) < \hat{s}(x'))$.
Cerchiamo di vedere come utilizzare la formula appena utilizzata su di un esempio

Esempio 2 : supponiamo che, dopo aver ordinato un insieme di dati, si ottenga il seguente ordinamento $x_1^+, x_2^+, x_3^-, x_4^+, x_5^+, x_6^-, x_7^-, x_8^-$. Il ranking error in questo caso è:

$$\text{rank-err}(x_1^+, x_2^+, x_3^-, x_4^+, x_5^+, x_6^-, x_7^-, x_8^-) =$$

$$= \frac{0+0+2+0+0+0+0+0}{4*4} = \frac{1}{8}$$

Notiamo che i due punti di penalità del numeratore derivano dall'errata posizione di x_3^- rispetto a x_4^+(1 punto di penalità) e a x_5^+(1 punto ulteriore di penalità). Questo poiché x_3^- dovrebbe essere collocato dopo x_4^+ e x_5^+.
Supponendo invece che l'ordinamento ottenuto sia:
$x_1^-, x_2^-, x_3^-, x_4^-, x_5^-, x_6^+, x_7^+, x_8^+$ abbiamo che il ranking error è pari a:

$$\text{rank-err}(x_1^-, x_2^-, x_3^-, x_4^-, x_5^-, x_6^+, x_7^+, x_8^+) =$$

$$= \frac{3+3+3+3+3+0+0+0}{3*5} = \frac{15}{15} = 1$$

Notiamo che la situazione nel secondo caso è evidentemente più disordinata visto che tutti i negativi precedono erroneamente tutti i positivi. Questo porta ad un rank-err maggiore infatti.

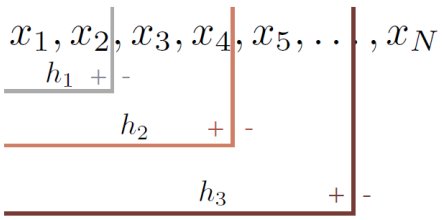

Figura 13: Tre differenti classificatori in base alla funzione di ranking

Esempio 3 : riprendiamo l'albero dei punteggi illustrato nella parte sinistra di Figura 9. In questo caso le 5 mail nella foglia destra vengono classificate più in alto delle 10 della foglia centrale(otteniamo 5*10 punti di penalità) e vengono anche classificate più in alto delle 20 contenute nella foglia sinistra(otteniamo dunque 5*20 punti di penalità). Inoltre le 5 mail della foglia centrale vengono classificate più in alto delle 20 della foglia sinistra(per questo otteniamo 5*20 punti di penalità.
Inoltre la foglia destra riceve 20*5 mezze penalità perché le 20 positive e le 5 negative ricevono uno stesso punteggio. Per questo stesso motivo la foglia centrale riceve 10*5 mezze penalità mentre la foglia di sinistra riceve 20*40 mezze penalità per lo stesso motivo. Nel complesso abbiamo:

$$\text{rank-err}(x) = \frac{50 + 100 + 100 + \frac{1}{2}(100) + \frac{1}{2}(50) + \frac{1}{2}(800)}{50 * 50} = \frac{725}{2500} = 29\%$$

Ovvero una ranking accuracy del 71%

Un aspetto interessante da notare è che data una funzione di ranking h, è possibile creare diversi classificatori binari in base al valore di h oltre il quale cambia la classificazione. Mostriamo tre differenti funzioni di classificazione in Figura 13.

04 / 10 /2017

2.3 Stima di probabilità

Uno stimatore probabilistico di classe è uno scoring classifier che per ogni classe fornisce un valore di probabilità associato ad essa. Ovvero:

$$\hat{p} : \mathfrak{X} \to [0,1]^k$$

Quindi $\hat{p}$ è un vettore a k componenti della forma $\hat{p}(x) = (\hat{p}_1(x), \ldots, \hat{p}_k(x))$. Chiaramente la somma delle varie probabilità deve essere pari a 1. Nel caso in cui vi siano solo 2 classi diremo che $\hat{p}(x)$ indica la probabilità per la classe positiva.
Anche in questo caso possiamo utilizzare un albero, come mostrato in Figura 14

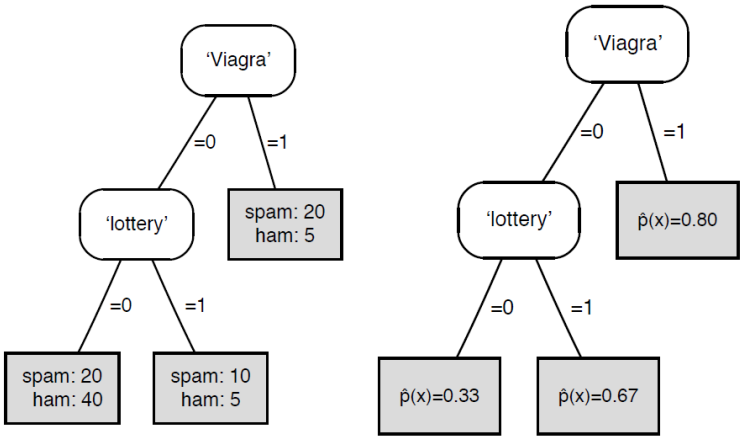

Figura 14: A sinistra l'albero delle feature, a destra l'albero di probabilità

Il contenuto di ogni foglia è facilmente intuibile: $\hat{p}(x) = \frac{\#spam}{\#mail}$.
Supponendo di avere a disposizione uno stimatore probabilistico, dobbiamo stabilire un criterio per valutarlo. Solitamente usiamo l'*errore quadratico* SE. Ovvero

$$SE(x) = \frac{1}{2} \parallel \hat{p}(x) - I_{c(x)} \parallel_2^2 = \frac{1}{2} \sum_{i=1}^{k} \hat{p}_i(x) - I[c(x) = C_i] \tag{5}$$

Dove $I_{c(x)}$ è un vettore che contiene valore 1 nella posizione relativa alla classe in cui sarebbe davvero classificato x. Tutte le altre componenti sono pari a 0. Cerchiamo di capirlo meglio con l'esempio che segue.

Esempio 1 : supponiamo di aver ottenuto il seguente stimatore probabilistico per un dataset avente tre classi possibili: $\hat{p}(x) = [0.2,\ 0.6,\ 0.2]$. Supponiamo che un certo esempio debba esser classificato nella seconda classe. Dunque il suo $I_{c(x)} = [0,\ 1,\ 0]$. In questo caso l'errore ottenuto è
$SE(x) = \frac{1}{2}((0.2-0)^2 + (0.6-1)^2 + (0.2-0)^2) = 0.24$

Esempio 2 : supponendo che $\hat{p}(x) = [0.7,\ 0.1,\ 0.2]$ e che $I_{c(x)} = (1,\ 0,\ 0)$ allora l'errore è:
$SE(x) = 0.07$

Esempio 3 : supponendo che $\hat{p}(x) = [0,\ 1,\ 0]$ e che $I_{c(x)} = (1,\ 0,\ 0)$ l'errore è:
$SE(x) = 1$
Notiamo che se lo stimatore effettua la giusta stima riceviamo errore 0, altrimenti errore 1.

Esempio 4 : supponendo di utilizzare i due stimatori che seguono:
$\hat{p}_1(x) = [0.7,\ 0.1,\ 0.2]$

$\hat{p}_2(x) = [0.99,\ 0,\ 0.01]$
Analizziamo l'errore ottenuto nel caso in cui $I_{c(x)} = (1,\ 0,\ 0)$ e $I_{c(x)} = (0,\ 0,\ 1)$. Sul primo caso otteniamo i due errori che seguono:

$SE_1(x) = 0.07$
$SE_2(x) = 0.0001$
Nel secondo caso invece:

$SE_1(x) = 0.57$
$SE_2(x) = 0.98$

Notiamo che, come nell'esempio precedente, avere uno stimatore con valori molto vicini ad 1 non è una cosa buona perché quando lo stimatore commette un errore il valore di SE diventa molto alto.

Un'altra valutazione dell'errore può essere fatta tramite l'*errore quadratico medio*:

$$MSE(Te) = \frac{1}{|Te|} \sum_{x \in Te} SE(x) \tag{6}$$

Vediamo su un esempio:

Esempio 5 : calcoliamo l'errore quadratico medio per l'albero mostrato in Figura 14. In questo caso dobbiamo considerare il contributo di ogni foglia. Ricordando che in totale abbiamo $|Te| = 100$, cominciamo da quella di destra:
poiché $\hat{p}(x) = [0.8,\ 0.2]$, dobbiamo considerare che abbiamo 20 classificazioni positive e 5 negative. Dunque l'errore per la prima foglia è:

$E_{rl} = (0.2)^2 * 20 + (0.8)^2 * 5 = 4$

Il contributo della foglia centrale è:

$E_{cl} = (0.33)^2 * 10 + (0.67)^2 * 5 \simeq 3.3335$

Similmente, l'errore della foglia sinistra è:

$E_{ll} = (0.67)^2 * 20 + (0.33)^2 * 40 \simeq 13.334$

Dunque l'errore totale è:

$$MSE(Te) = \frac{1}{|Te|}(E_{rl} + E_{cl} + E_{ll}) = \frac{20.6675}{100} \simeq 21\%$$

Visto che abbiamo valutato l'errore di un certo stimatore, una domanda che potremmo farci a questo punto è: ma come possiamo costruire uno stimatore? Solitamente utilizziamo un procedimento di *probabilità empiriche*. In questo caso, supponendo di avere un insieme di dati etichettati S, e dove il numero

di esempi all'interno di una classe C_i viene indicato con n_i, allora lo stimatore ottenuto è indicato con

$$\dot{p}(S) = (\frac{n_1}{|S|}, \ldots, \frac{n_k}{|S|}) \tag{7}$$

Esempio 6 : supponendo di avere a disposizione 10 dati suddivisi in 3 classi secondo la seguente tabella

classe	1	2	3
occorr.	3	2	5

Allora è facile capire che $\dot{p}(S) = (\frac{3}{10}, \frac{2}{10}, \frac{5}{10})$

Il problema di questo approccio empirico è che se una certa classe non è mai occorsa negli esempi, successivamente riterremo che sia impossibile che un elemento di quella classe occorra.
Per risolvere questo problema sfruttiamo la *correzione di Laplace* che assegna un elemento fittizio ad ogni classe affinché possiamo stimare nel seguito che gli elementi di tutte le classi possano occorrere. Dunque il valore di ogni componente di $\dot{p}$ passa da
$\dot{p}_i(S) = \frac{n_i}{|S|}$
a
$\dot{p}_i(S) = \frac{n_i + 1}{|S| + k}$

In realtà questo concetto può essere generalizzato andando ad aggiungere un numero a piacere di elementi fittizi. Le componenti di $\dot{p}$ diventano:
$\dot{p}_i(S) = \frac{n_i + m \cdot \pi_i}{|S| + m}$

dove π_i rappresenta la probabilità a priori. Notiamo che per ottenere la correzione di Laplace basta considerare $m = k$ e $\pi_i = \frac{1}{k}$

2.4 Oltre la classificazione binaria

Cosa succede quando invece abbiamo più di due classi? Fondamentalmente possiamo utilizzare un algoritmo dedicato(tratteremo questo argomento successivamente) oppure possiamo utilizzare un classificatore binario in modo intelligente per effettuare classificazioni su più classi. Questo secondo metodo che ci accingiamo a studiare può essere effettuato in due modi:

- One vs rest
- One vs one

Vediamo più nel dettaglio entrambi i metodi. Il primo consiste nell'andare a fingere che esistano solo due classi: una certa classe C_i e un'altra classe che è l'insieme di tutte le classi C_j con $i \neq j$. A questo punto, supponendo di avere k classi, possiamo realizzare k diversi classificatori binari uno per ogni classe.

Esempio 1 : supponiamo di avere a disposizione tre classi $C_1,\ C_2,\ C_3$. Supponiamo che il dataset iniziale sia quello mostrato nella parte sinistra della Tabella 8

x	y
	1
	3
...	2
	1
	3

x	y
	1
	-1
...	-1
	1
	-1

x	y
	-1
	-1
...	1
	-1
	-1

x	y
	-1
	1
...	-1
	-1
	1

Tabella 8: Rispettivamente: la tabella con i dati di input, a seguire le tre tabelle one vs rest: 1 vs {2, 3}, 2 vs {1, 3} e 3 vs {1, 2}

Subito dopo, sempre in Tabella 8 mostriamo le tre tabelle one vs rest. Notiamo che sono una per ogni classe.
Quando nel seguito riceviamo un esempio che appartiene (ad esempio) alla terza classe, scriveremo il vettore $[-1,\ -1,\ 1]$

Un modo più sintetico di rappresentare quanto visto nell'esempio precedente è tramite la *matrice one vs rest*

Esempio 2 : la matrice relativa all'esempio precedente potrebbe essere:

	1 vs {2, 3}	2 vs {1, 3}	3 vs {2, 3}
C_1	+1	-1	-1
C_2	-1	+1	-1
C_3	-1	-1	+1

Tabella 9: Matrice one vs rest

Possiamo ottenere una miglioria partendo dal caso precedente. Notiamo che nella matrice dell'esempio precedente non abbiamo alcun tipo di ordine. Possiamo sfruttare invece una qualche forma di ordinamento. L'idea è di utilizzare il primo classificatore. Se restituisce 1 ci fermiamo altrimenti procediamo con il secondo e così via.

Esempio 3 : la matrice ordinata costruita partendo dalla Tabella 9 sarebbe la Tabella 10

	1 vs {2, 3}	2 vs {1, 3}
C_1	+1	0
C_2	-1	+1
C_3	-1	-1

Tabella 10: Matrice one vs rest ordered

L'altro metodo che abbiamo per continuare ad utilizzare i classificatori binari è il one vs one. L'idea è di effettuare un confronto di ogni classe con ogni altra classe.

Esempio 4 : la matrice relativa al confronto one vs one, nel caso in cui il confronto sia simmetrico, ovvero:
(1 vs 2) $\equiv$ (2 vs 1)
è la seguente

	1 vs 2	1 vs 3	2 vs 3
C_1	+1	+1	0
C_2	-1	0	+1
C_3	0	-1	-1

Tabella 11: Matrice one vs one simmetrica

Se invece il confronto non è simmetrico abbiamo una matrice di dimensione raddoppiata:

	1 vs 2	2 vs 1	1 vs 3	3 vs 1	2 vs 3	3 vs 2
C_1	+1	-1	+1	-1	0	0
C_2	-1	+1	0	0	+1	-1
C_3	0	0	-1	+1	-1	+1

Tabella 12: Matrice one vs one asimmetrica

Per riuscire ad interpretare correttamente i dati possiamo utilizzare una formula di differenza tra la classe ottenuta e quelle contenute nella matrice. Per essere più precisi abbiamo che la *distanza* $d(w, c)$ dove w è il vettore costruito a partire dal nuovo input e c sono le varie classi, è:

$$d(w, c) = \sum_i \frac{(1 - c_i w_i)}{2} \tag{8}$$

Esempio 5 : supponendo che il vettore w sia $w = (+1, -1, -1)$, le varie distanze di w dalle classi della matrice mostrata in Tabella 9 sono:
$d(w, C_1) = 0$
$d(w, C_2) = 2$
$d(w, C_3) = 2$

Ad ogni modo, i due metodi appena mostrati soffrono di un grave difetto(one vs rest in particolare): tendono a far apparire un dataset bilanciato come se fosse estremamente sbilanciato. Questo può essere un problema per alcuni algoritmi di learning[2].

[2]Solitamente dataset sbilanciati generano maggiori problemi in fase di apprendimento

Esempio 6 : supponiamo di avere 10 classi distinte con 10 occorrenze l'una. Questo è un dataset perfettamente bilanciato, ovvero tutte le classi hanno stesso numero di occorrenze. Purtroppo one vs rest invece fa apparire il dataset come se fosse formato da coppie di classi dove una contiene 10 elementi mentre l'altra 90.

Questo problema risulta alleviato nel one vs one grazie al valore 0 all'interno della matrice per le classi che non occorrono nei confronti.

05 / 10 /2017

2.5 Regressione

L'obiettivo della regressione è quello di costruire uno stimatore di funzione della forma:

$$\hat{f} : \mathfrak{X} \to \mathbb{R}$$

Quindi partendo dai dati formati dalle coppie $(x_i, f(x_i))$, vogliamo cercare di costruire uno stimatore di f. Rispetto ai casi precedenti ottenere in output un valore in $\mathbb{R}$ ci permette di avere una precisione molto maggiore ma pone anche il rischio di overfitting. Il modello che andiamo a cercare dovrebbe essere il più semplice possibile che sia sufficientemente espressivo da cogliere la struttura matematica dei dati.
Un esempio in questo senso potrebbe essere il fitting[3] polinomiale. Vediamolo con l'esempio che segue.

Esempio 1 : supponiamo di avere a disposizione i dati mostrati in Tabella 13. Mostriamo in Figura 15 una serie di possibili modelli polinomiali per i dati

x	y
1.0	2.0
4.1	3.7
6.1	4.6
7.9	7.0

Tabella 13: Dati relativi all'esempio 1

illustrati nella tabella precedente

Ovviamente potremmo pensare di costruire un polinomio che passi per tutti i punti del nostro dataset. Questa è in generale una pessima scelta perché in seguito il nostro modello tenderà a stimare molto male nuovi dati in input, proprio a causa dell'overfitting. Un'alternativa possibile al modello polinomiale è quello delle funzioni a tratti, che mostriamo in Figura 16.

[3] Adattamento di una funzione ai dati

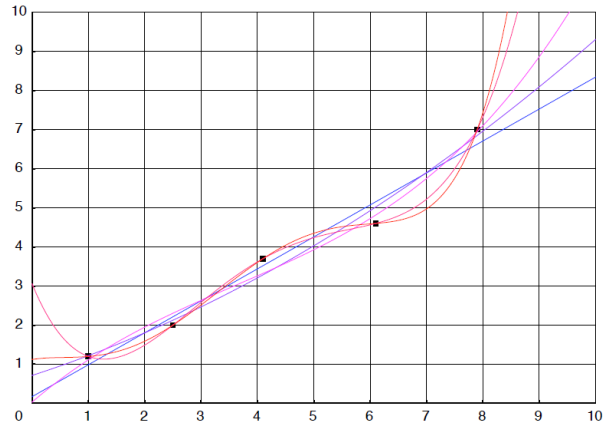

Figura 15: Possibili modelli polinomiali per i dati della Tabella 13

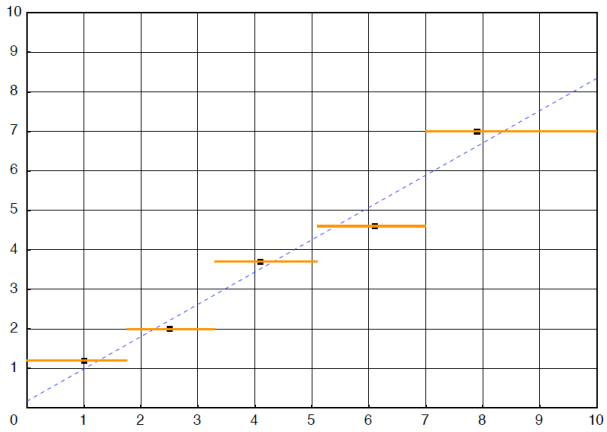

Figura 16: Modello di funzione a tratti per i dati della Tabella 13

Una buona regola generale per evitare l'overfitting è che il numero di parametri utilizzati dovrebbe essere nettamente minore del numero dei dati. Notiamo che usando troppi pochi parametri l'errore sarà alto perché non riusciremo a descrivere bene i dati. Usandone troppi incorriamo in overfitting. Questo viene detto *bias-variance dilemma*: modelli troppo semplici introducono un bias sistematico sui dati(dovuto alla semplicità del modello) che non è eliminabile neanche avendo molti dati a disposizione. D'altro canto un modello troppo complesso soffrirà di un errore dovuto alla varianza dei dati.
Per comprendere meglio questo concetto possiamo osservare la Figura 17.

Supponiamo di voler colpire il centro di un bersaglio. I risultati ottenuti nel primo caso presentano un basso bias(in questo esempio il bias è lo scostamento dal centro) e una bassa varianza. Dunque sono ottimi. Nel secondo caso

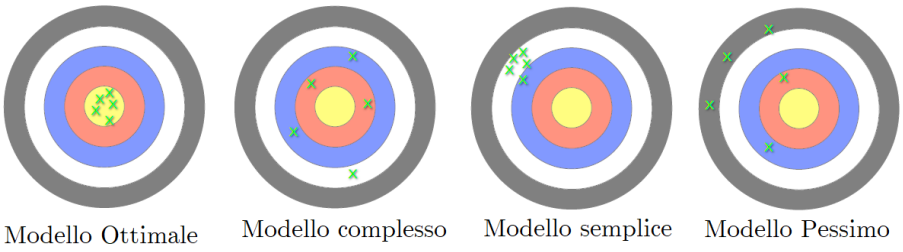

Figura 17: Rappresentazione della complessità dei modelli

abbiamo un basso bias ma un'elevata varianza. È il caso di un modello troppo complesso. Nel terzo caso abbiamo una bassa varianza ma un bias elevato, dunque il modello è troppo semplice e infatti tutti i punti subiscono un errore sistematico dovuto al bias che li porta nella stessa zona del bersaglio. L'ultimo caso è il peggiore perché presenta elevato bias ed elevata varianza.
Ad ogni modo è interessante l'idea che questo rapporto tra bias e varianza possa essere rappresentato matematicamente. Infatti supponendo di avere un esempio x noto e le due funzioni f[4] e la sua stima $\hat{f}$, possiamo scrivere che:
$E[(f(x) - \hat{f}(x))^2] =$
$= E[f(x)^2 - 2f(x)\hat{f}(x) + \hat{f}(x)^2] =$
$= E[f(x)^2] - 2f(x)E[\hat{f}(x)] + E[\hat{f}(x)^2] - E[\hat{f}(x)]^2 + E[\hat{f}(x)]^2 =$
$= E[\hat{f}(x)^2] - E[\hat{f}(x)]^2 + f(x)^2 - 2f(x)E[\hat{f}(x)] + E[\hat{f}(x)]^2 =$
$= E[(\hat{f}(x) - E[\hat{f}(x)])]^2 + (f(x) - E[\hat{f}(x)])^2 =$
$= Var(\hat{f}(x)) + Bias^2(\hat{f}(x))$
Dunque il valore atteso quando siamo a conoscenza di $f(x)$, può essere visto come la somma della varianza e del Bias al quadrato.
/ 11 /2017

3 Modelli lineari

Esistono diversi modelli lineari nell'ambito dell'apprendimento automatico.

3.1 Regressione

Supponiamo di avere a nostra disposizione un insieme di punti (x, y) di cui vogliamo determinare un modello.
Nel caso dei modelli lineari dobbiamo determinare il valore della coppia C, D di modo che il modello lineare $Cx + D = y$ si adatti al meglio ai nostri punti. Idealmente potremmo scrivere un sistema di equazioni formato da una equazione per ogni punto da modellare. Ad esempio, dato l'insieme di punti:

$(-1, -1.52)$

[4] Notare che $f(x)$ è dunque una costante!

$(-0.8, -1.21)$
$(-0.6, -0.67)$
$\vdots$

potremmo avere:

$$\begin{cases} -1C + D = -1.52 \\ -0.8C + D = -1.21 \\ \dots \end{cases}$$

Il problema di questo sistema è che, in generale non è risolvibile. Infatti se abbiamo più equazioni che incognite[5] il modello non ha una soluzione.
Questo fatto può essere mostrato considerando che il sistema che abbiamo illustrato poco fa può essere riscritto in forma matriciale come segue:

$$\underbrace{\begin{bmatrix} -1 & 1 \\ -0.8 & 1 \\ -0.6 & 1 \\ \dots & \dots \end{bmatrix}}_{X} \cdot \underbrace{\begin{bmatrix} C \\ D \end{bmatrix}}_{w} = \underbrace{\begin{bmatrix} -1.52 \\ -1.21 \\ -0.67 \end{bmatrix}}_{y}$$

Di fatto questo ci suggerisce che stiamo cercando la soluzione all'equazione

$$Xw = y \tag{9}$$

Dunque se volessimo trovare una soluzione w al sistema la otterremmo come segue: $w = X^{-1}y$
Ricordiamo però che X^{-1} è ottenibile solo se X è invertibile. E questo succede solo se X è quadrata, ovvero se abbiamo tante equazioni quante variabili.

Dunque abbiamo bisogno di un approccio più generale se vogliamo risolvere un sistema lineare. Per farlo cominciamo con il notare che variando il vettore dei pesi w otteniamo un y diverso, ovvero[6] stiamo coprendo lo spazio delle colonne di X e y non vi appartiene. Questo fatto è mostrato geometricamente in Figura 18

Notiamo che in tale figura viene definito l'errore come il vettore derivante dalla differenza di y e di p dove p è il risultato da noi ottenuto impostando i pesi.
Dunque $e = y - Xw$.
Per avere una valutazione numerica dell'errore commesso possiamo considerare la norma 2 del vettore e

$$\| e \|_2 = \sqrt{\sum_i (y_i - p_i)^2} \tag{10}$$

[5] Abbiamo n equazioni, una per ogni coppia (x, y) ma abbiamo solo due variabili C, D
[6] Ricordando che lo spazio generato dalle colonne di un matrice consiste nell'immagine della funzione che la matrice rappresenta

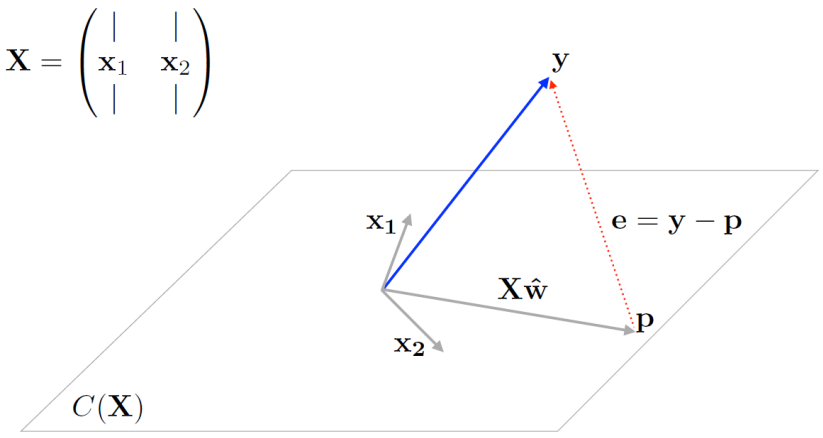

Figura 18: Interpretazione geometrica del prodotto $X\hat{w}$

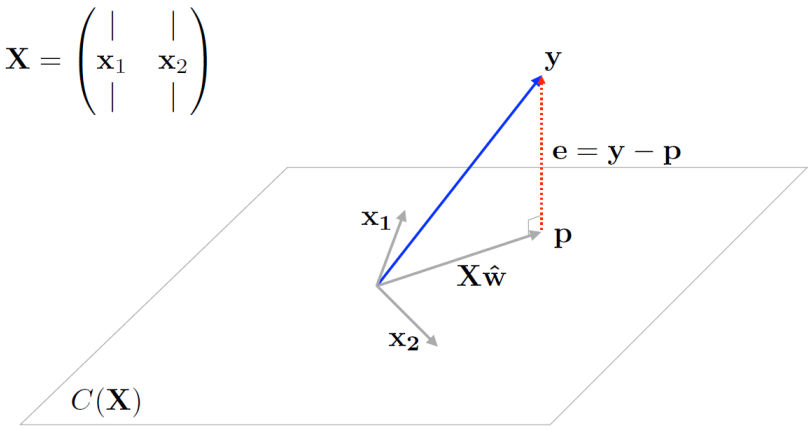

Figura 19: Il vettore X è perpendicolare al vettore di errore e

Dunque trovare un modello lineare per un insieme di punti si riduce a determinare un vettore $\hat{w}$ che minimizzi la seguente quantità

$$\min_{\hat{w}} \parallel X\hat{w} - y \parallel_2^2 \tag{11}$$

Da un punto di vista geometrico dobbiamo individuare il vettore p che si avvicini quanto più possibile al vettore y. Questo corrisponde a fare in modo che il generico vettore X sia perpendicolare al vettore di errore e. Questa situazione viene mostrata in Figura 19

Da un punto di vista algebrico la condizione di perpendicolarità tra X ed e è data dalla seguente equazione:

$X^T e = 0$

Da cui possiamo implicare che:

$X^T(y - X\hat{w}) = 0 \Leftrightarrow X^T y - X^T X\hat{w} = 0 \Leftrightarrow X^T X\hat{w} = X^T y \Leftrightarrow \hat{w} = (X^T X)^{-1} X^T y$
Dunque quello che dobbiamo determinare è:

$$\hat{w} = (X^T X)^{-1} X^T y \tag{12}$$

Vediamolo tramite l'esempio che segue:

Esempio 1 : supponiamo di avere a disposizione i tre seguenti punti:

$$(1,1) \qquad (2,2) \qquad (3,2)$$

Aggiungendo il vettore dei coefficienti di D(tutti pari a 1), otteniamo le seguenti matrici

$X = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{bmatrix}$ $y = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$ Per trovare la soluzione $\hat{w} = (X^T X)^{-1} X^T y$ procediamo come segue:

$$X^T X = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{bmatrix} = \begin{bmatrix} 14 & 6 \\ 6 & 3 \end{bmatrix}$$

Da cui dobbiamo ottenere l'inversa $(X^T X)^{-1}$ come segue:

$$\left[\begin{array}{cc|cc} 14 & 6 & 1 & 0 \\ 6 & 3 & 0 & 1 \end{array}\right] \Leftrightarrow \cdots \Leftrightarrow \left[\begin{array}{cc|cc} 1 & 0 & 1/2 & -1 \\ 0 & 1 & -1 & 7/3 \end{array}\right]$$

Complessivamente abbiamo che:

$$\hat{w} = (X^T X)^{-1} X^T y = \begin{bmatrix} 1/2 & -1 \\ -1 & 7/3 \end{bmatrix} \underbrace{\begin{bmatrix} 11 \\ 5 \end{bmatrix}}_{X^T y} = \begin{bmatrix} 1/2 \\ 2/3 \end{bmatrix}$$

Il modello lineare ottenuto con questo procedimento dunque è:

$$y = \frac{1}{2}x + \frac{2}{3}$$

Mostriamo il modello ottenuto in Figura 20

Quale è il problema del metodo visto finora? È fondamentalmente poco resistente agli *outlayer*, ovvero a quegli esempi "problematici" poiché misurati male o comunque impropri per diversi motivi.
Di fatto, in presenza di outlayer, il modello lineare tenderà a correggersi cercando di minimizzare l'errore tramite il metodo dei minimi quadrati. In particolare

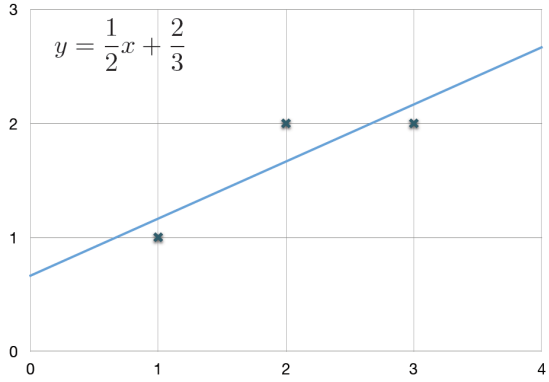

Figura 20: L'insieme dei punti e il modello lineare $y = \frac{1}{2}x + \frac{2}{3}$

notiamo che un punto molto distante dagli altri contribuirà significativamente all'accrescimento dell'errore visto che contribuisce -circa- proporzionalmente al suo quadrato.
Per evitare questo tipo di situazioni solitamente viene utilizzata una tecnica di regolarizzazione. La nuova formula da minimizzare diventa:

$$\hat{w} = \min_{\hat{w}} \underbrace{(y - Xw)^T(y - Xw)}_{=\|y-Xw\|_2^2} + \lambda \parallel w \parallel_2^2 \tag{13}$$

Di fatto abbiamo un termine correttivo $\lambda \parallel w \parallel_2^2$ che contribuisce proporzionalmente alla lunghezza del vettore $\parallel w \parallel_2^2$.
Perché minimizzare la lunghezza di w aiuta ad ottenere un modello meno sensibile alle fluttuazioni? Possiamo dare due motivi

1. Motivo matematico: supponendo di avere un vettore X affetto da un errore D allora abbiamo che

   $$(X + D)w = Xw + Dw$$

   Notiamo che minimizzare il modulo di w ci permette di ridurre il contributo di errore dovuto al termine Dw

2. Motivo filosofico: modelli con pesi ridotti solitamente descrivono modelli più semplici, e questo va incontro al principio del rasoio di Occam

Come in precedenza, esiste una soluzione che determina i valori di w. Pari a:

$$\hat{w} = (X^T X + \lambda I)^{-1} X^T y \tag{14}$$

dove I indica la matrice identità. Questo tipo di regressione, detta *regressione di Ridge* è più stabile delle precedenti. Una forma alternativa detta *regressione*

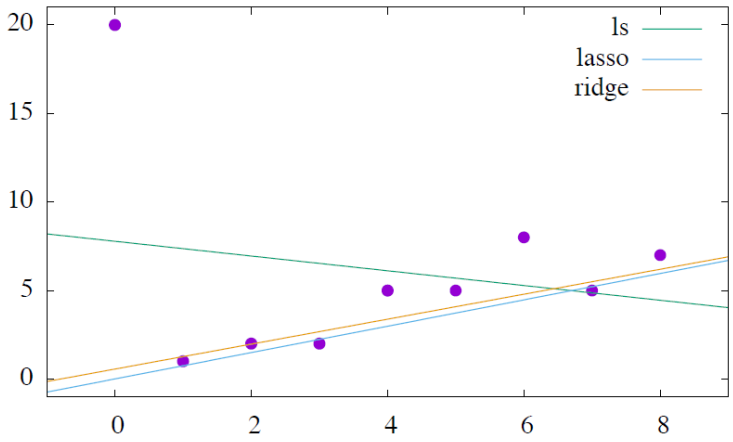

Figura 21: Confronto fra il metodo di regressione least square, ridge e lasso

lasso consiste nell'utilizzare la norma 1 come termine correttivo invece della norma 2. Ovvero

$$\hat{w} = \min_{\hat{w}} \underbrace{(y - Xw)^T(y - Xw)}_{=\|y-Xw\|_2^2} + \lambda \parallel w \parallel_1^2$$

I risultati ottenuti dalle varie regressioni sono mostrate in Figura 21

Notiamo la maggiore instabilità del metodo least square che tende a correggere a causa dell'outlayer.

08 / 11 /2017

In generale la tecnica lasso tende a favorire soluzioni più sparse[7]. Questo fatto può essere compreso osservando Figura 22

Supponiamo di avere un vettore dei coefficienti w. Supponiamo di considerare la distribuzione dei coefficienti, ovvero costruiamo un istogramma in cui l'altezza di ogni colonna è proporzionale al numero di coefficienti che assumono quel valore[8]. Notiamo che nella parte alta della Figura 22 abbiamo molte più occorrenze intorno al valore zero, dunque in base alla definizione la regressione lasso porta ad un vettore dei coefficienti più sparso. Questa situazione si verifica perché entrambi i metodi tenderebbero a portare il valore dei coefficienti vicino allo zero. Ma una volta che un coefficiente viene portato al di sotto del valore 1, il metodo lasso che fa uso della norma 1 presenta un costo lineare per quel coefficiente, mentre il costo per Ridge è quadratico. Dunque nell'intervallo $[0, 1]$ il costo secondo lasso è più alto(costo lineare) e dunque la tecnica tende a

[7] Ricordiamo che secondo la sua definizione, un vettore è tanto più *sparso* quante più componenti sono settate a zero

[8] O che cadono in quell'intervallo se consideriamo coefficienti a valori continui

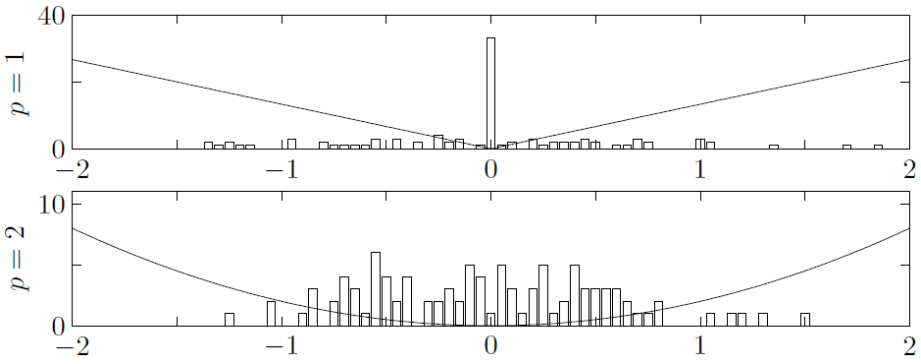

Figura 22: In alto la distribuzione dei coefficienti e il costo relativo secondo lasso. In basso la distribuzione dei coefficienti e il costo relativo secondo Ridge

schiacciare i valori verso lo zero per ridurre i costi. Invece la tecnica Ridge non se ne cura poi così tanto perché una volta che il valore dei pesi è inferiore a uno, influisce molto meno sui costi(a causa del costo quadratico).

A questo punto possiamo pensare di utilizzare il metodo least square per effettuare classificazione. Possiamo infatti rappresentare gli esempi della classe positiva con il valore 1 e quelli della classe negativa con il valore -1. In questo senso la classificazione diventa:

$$\hat{c}(x) = \begin{cases} 1, & \text{se } X^T\hat{w} - t > 0 \\ 0, & \text{se } X^T\hat{w} - t = 0 \\ -1, & \text{se } X^T\hat{w} - t < 0 \end{cases} \tag{15}$$

Dove t rappresenta il valore dell'intercetta.

Notiamo che i punti mostrati in Figura 23 sono della forma (x_1, x_2), dunque la loro classificazione (pari a 1 per quelli rossi e -1 per quelli verdi) non è rappresentabile su un piano, ma su uno spazio tridimensionale sui cui assi abbiamo x_1, x_2, y. Dunque una rappresentazione più veritiera di un classificatore dovrebbe essere qualcosa di simile a quanto mostrato in Figura 24

Complessivamente un piano del genere viene descritto da tre parametri, ovvero da un'equazione della forma $w_1x_1 + w_2x_2 + t > 0$

3.2 Support Vector Machine

Fino ad ora abbiamo visto che esistono tutta una serie di rette, piani e iperpiani che ci permettono di separare insiemi di dati. Ma visto che ne esistono molti, quale tra questi potrebbe essere selezionato come migliore? La risposta è che in uno spazio di dimensione $\mathbb{R}^n$ l'iperpiano migliore è quello che massimizza il

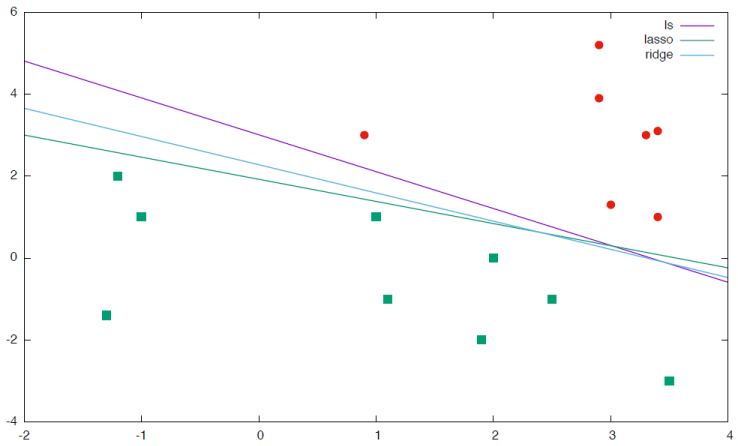

Figura 23: Un insieme di punti (x_1, x_2) e il piano che li separa

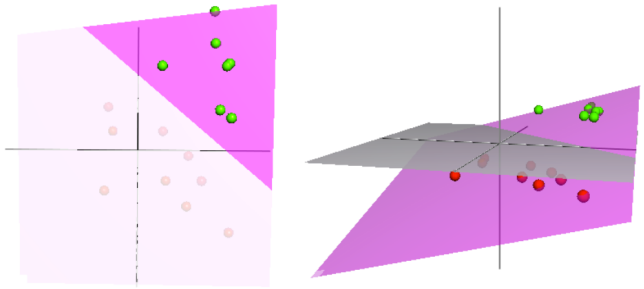

Figura 24: Rappresentazione nello spazio tridimensionale del classificatore. I punti in verde sono quelli classificati positivamente, quelli in rosso sono classificati negativamente

margine tra gli insiemi di punti. Mostriamo intuitivamente questa condizione in Figura 25

Quello che cercheremo di fare nel seguito consiste nell'individuare il margine minimo e cercare di massimizzarlo. Per procedere in questa direzione partiamo dal fatto che la nostra classificazione $(Wx_i - t)$ e l'effettiva classificazione y_i del punto devono essere concordi. Questo può essere facilmente espresso tramite:

$$y_i(Wx_i - t) > 0 \tag{16}$$

Di fatto preferiremmo che valesse $y_i(Wx_i - t) > \epsilon$, ovvero che tra il nostro piano e il punto la cui classificazione è y_i vi fosse una certa distanza. Possiamo moltiplicare sia a destra che a sinistra della disequazione per un valore positivo. Solitamente moltiplichiamo per $\frac{1}{\epsilon}$ in tal modo abbiamo che:

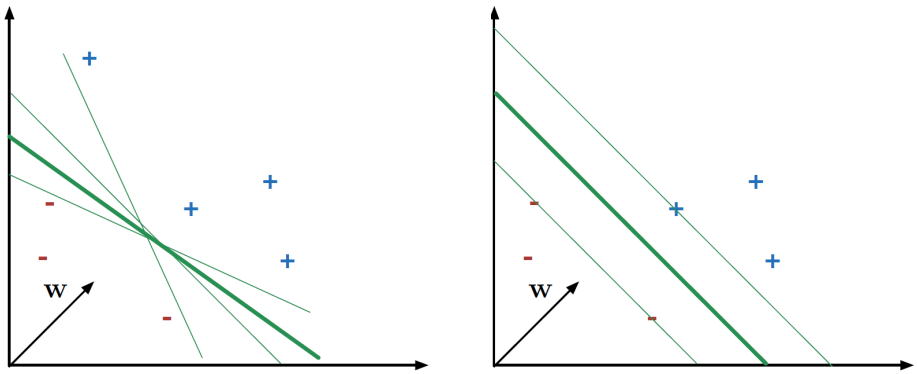

Figura 25: A sinistra alcuni possibili piani, a destra il miglior piano possibile, ovvero quello che massimizza il margine tra i punti classificati positivamente e quelli classificati negativamente

$$\frac{1}{\epsilon} y_i(Wx_i - t) > 1 \Leftrightarrow y_i(\frac{1}{\epsilon} Wx_i - \frac{1}{\epsilon} t) > 1 \Leftrightarrow y_i(W'x_i - t') > 1$$

Dunque, visto che W' e t' differiscono da W e da t a meno di una costante moltiplicativa, vale che:

$$y_i(Wx_i - t) > 1 \tag{17}$$

Notiamo che sui bordi del margine(su cui si trova almeno un punto) varrà che $y_i(Wx_i - t) = 1$. I punti(almeno un positivo e un negativo) che si collocano sui rispettivi bordi del margine sono detti *support vector*.
Ottenute queste premesse come possiamo ottenere la migliore soluzione possibile? Detti x_+ e x_- quei punti che determinano la posizione del bordo del margine(ovvero i due support vector), matematicamente dobbiamo calcolare il margine μ come segue

$$\mu = (x_+ - x_-) \cdot \frac{w}{\| w \|} \tag{18}$$

Il significato geometrico di questa formula viene mostrato in Figura 26

L'idea è la seguente: se il vettore $(x_+ - x_-)$ è quello rappresentato in rosso, a noi interessa la sua proiezione lungo l'asse $\frac{w}{\| w \|}$, dunque effettuiamo un prodotto scalare tra $(x_+ - x_-)$ e $\frac{w}{\| w \|}$
Riconsideriamo la definizione di margine μ: visto che x_+ e x_- sappiamo che sono support vector, per loro vale che $y_i(Wx_i - t) = 1$. Ma dunque se

$$x_+ \cdot w - t = 1 \Leftrightarrow x_+ \cdot w = 1 + t$$
$$-(x_- \cdot w - t) = 1 \Leftrightarrow x_- \cdot w = t - 1$$

allora:

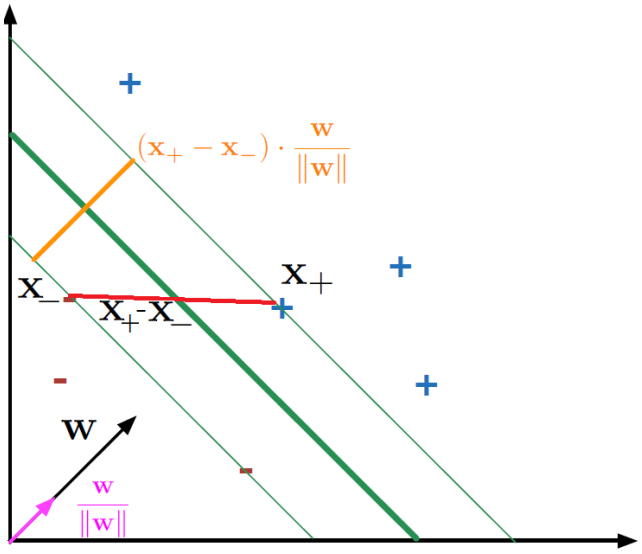

Figura 26: Rappresentazione geometrica di $(x_+ - x_-) \cdot \frac{w}{\| w \|}$

$$\mu = (x_+ - x_-) \cdot \frac{w}{\| w \|} = \frac{x_+ \cdot w}{\| w \|} - \frac{x_- \cdot w}{\| w \|}$$

E dunque

$$\mu = \frac{1+t}{\| w \|} - \frac{t-1}{\| w \|} = \frac{2}{\| w \|} \tag{19}$$

Notiamo che massimizzare $\frac{2}{\| w \|}$ di fatto vuol dire minimizzare $\| w \|$, o anche minimizzare $\| w \|^2$. Dunque il problema di determinazione del margine può anche essere riscritto come segue

$$\min_{w,t} \frac{1}{2} \| w \|^2 \tag{20}$$

dove per ogni punto i-esimo deve valere che $y_i(Wx_i - t) > 1$.
Notiamo che abbiamo elevato la norma al quadrato e l'abbiamo moltiplicata per $\frac{1}{2}$ solo per facilitarci i conti.

Andiamo adesso a considerare il problema duale. Il problema duale ci permette di evidenziare dei vincoli non triviali su ogni problema di ottimizzazione. Intuitivamente ci fornisce un punto di vista diverso sul problema e ci permette di considerarlo da un'altra prospettiva. Supponendo di avere un problema di ottimizzazione come quello del margine formulato come segue:

$\min_x f_0(x)$

con $f_i(x) \leq 0$ dove $i \in [1, m]$ e $g_j(x) = 0$ dove $j \in [1, p]$ allora la forma duale lagrangiana è:

$$g(\alpha, \eta) = \inf_x \Gamma(x, \alpha, \eta) = \inf_x (f_0(x) + \sum_{i=1}^{m} \alpha_i f_i(x) + \sum_{i=1}^{p} \eta_i g_i(x)) \qquad (21)$$

Notiamo che il nostro scopo è massimizzare $g(\alpha, \eta)$. Notiamo inoltre che

$$\sum_{i=1}^{m} \alpha_i f_i(x) < 0$$
$$\sum_{i=1}^{p} \eta_i g_i(x) = 0$$

Nel caso della support vector machine dobbiamo moltiplicare ogni vincolo di disuguaglianza per α_i di modo da ottenere la funzione lagrangiana. Dunque abbiamo che:

$$\Gamma(w, t, \alpha_1, \ldots, \alpha_n) = \frac{1}{2} \parallel w \parallel^2 - \sum_{i=1}^{n} \alpha_i (y_i (w \cdot x_i - t) - 1)$$

Da cui, spezzando la sommatoria e ricordando che $\parallel w \parallel^2 = w \cdot w$ otteniamo che:

$$\Gamma(w, t, \alpha_1, \ldots, \alpha_n) = \frac{1}{2} w \cdot w - w(\sum_{i=1}^{n} \alpha_i y_i x_i) + t(\sum_{i=1}^{n} \alpha_i y_i) + (\sum_{i=1}^{n} \alpha_i)$$

Adesso per trovare il massimo dobbiamo calcolare la derivata parziale di Γ rispetto a t e rispetto a w. Otteniamo che:

$$\frac{\partial \Gamma}{\partial t} = \textstyle\sum_i \alpha_i y_i$$
$$\frac{\partial \Gamma}{\partial w} = w - \textstyle\sum_i \alpha_i y_i x_i$$

Da cui troviamo che la derivata si annulla in:

$$\sum_i \alpha_i y_i = 0 \text{ per } t$$
$$w = \sum_i \alpha_i y_i x_i \text{ per } w$$

Dunque andando a sostituire all'interno della forma lagrangiana troviamo il minimo in

$$g(\alpha) = -\frac{1}{2} \textstyle\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i x_j + \sum_{i=1}^{n} \alpha_i$$

Dunque il problema duale diventa

$$\max_{\alpha} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i x_j + \sum_{i=1}^{n}\alpha_i \tag{22}$$

dove $\alpha_i \geq 0$ per ogni $i \in [1, n]$ e che valga $\sum_i y_i\alpha_i = 0$

Una domanda che potremmo porci a questo punto è: visto che sia il problema primale che quello duale trovano soluzione in un algoritmo numerico, come mai scomodare il problema duale e non farci bastare il primale?
Perché in tal modo abbiamo appreso alcune cose importanti:

- La soluzione del vettore w dei coefficienti è una combinazione lineare dei support vector
- i moltiplicatori di lagrange positivi sono associati ai support vector
- sia il learning che la classificazione possono essere svolti in termini di prodotti tra support vector
- nella formulazione duale della support vector machine possiamo usare il kernel trick per risolvere problemi non lineari

L'unico punto che non abbiamo propriamente dimostrato è il secondo. Questo deriva da una delle condizioni a contorno del dominio duale. Ovvero:

$\alpha_i(y_i(w \cdot x_i - t) - 1) = 0$

E visto che $(y_i(w \cdot x_i - t) - 1) \neq 0$ ne segue che $\alpha_i = 0$

09 / 11 /2017

3.3 Errore di margine

Fino ad ora tutto funziona se l'insieme di punti è linearmente separabile. Quello che dovremmo fare per ottenere una SVM più flessibile è di permettere che venga commesso un certo errore ξ sul nostro margine. Matematicamente viene espresso dalla disuguaglianza

$$y_i(wx_i - t) \geq 1 - \xi_i \tag{23}$$

L'ovvia conseguenza è che potremmo ottenere delle classificazioni sbagliate. Dunque cercheremo comunque di mantenere ξ abbastanza piccolo. La nuova funzione da minimizzare è:

$$\min_{w,t,\xi} \frac{1}{2} \parallel w \parallel^2 + C\sum_{i=1}^{n}\xi_i \tag{24}$$

dove $y_i(wx_i - t) \geq 1 - \xi_i$ con $1 \geq i \geq n$. E dove $\xi_i \geq 0$ sempre con $1 \geq i \geq n$. Notiamo che imponendo un valore di $C = 0$ andiamo a trascurare

completamente l'errore che commettiamo e otteniamo un vettore $w = 0$. Se invece C è elevato teniamo in grande considerazione l'errore. In tal caso avremo un margine ampio e pochi support vector. Per questo motivo C viene detto *parametro di complessità*.
Come già visto per il problema senza errori, possiamo costruire il problema duale per ricavare un'altra soluzione. Un po' sorprendentemente otteniamo lo stesso risultato del problema senza errore di margine con l'aggiunta di una sola condizione sul parametro α_i. Ovvero:

$$\max_{\alpha} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i x_j + \sum_{i=1}^{n}\alpha_i \tag{25}$$

dove $0 \leq \alpha_i \leq C$ per ogni $i \in [1, n]$ e che valga $\sum_i y_i\alpha_i = 0$
Notiamo che la condizione $\alpha_i \leq C$ proviene dal fatto che imponiamo la derivata del problema duale in ξ_i uguale a 0. Ovvero:

$$C - \alpha_i - \beta_i = 0 \Leftrightarrow C - \beta_i \geq \alpha_i \Leftrightarrow C \geq \alpha_i$$

Similmente a quanto visto in precedenza, anche in questo caso una condizione a contorno ci dice che $\beta_i\xi_i = 0$ e dunque possiamo fare un discorso analogo a quello mostrato in precedenza. In particolare notiamo che se un certo esempio ha un margine di errore pari a $\xi > 0$ allora ne segue che $\beta_i = 0$ e dunque abbiamo che(sempre dalla derivata del duale in $\partial\xi$) che $\alpha_i = C$. Questo ci fornisce l'idea di uno schema generale:

- $\alpha_i = 0$: per quei dati che sono classificati correttamente dalla linea del margine
- $0 \leq \alpha_i \leq C$: per quei dati che si trovano sulla linea del margine
- $\alpha_i = C$: per quei punti che si trovano sul vincolo $y_i(wx_i - t) = 1 - \xi_i$

Tutti e tre i casi vengono mostrati in Figura 27

3.4 Kernel

Adesso cerchiamo di andare a trattare problemi non linearmente separabili. Lo scopo delle prossime pagine è quello di introdurre il kernel trick, un algoritmo di costo lineare per trattare problemi non linearmente separabili.
L'idea intuitiva è quella di spostare i dati in uno spazio di dimensione più ampia, di modo siano linearmente separabili in essa.
Una *funzione kernel* è una funzione $K : \mathbb{V} \times \mathbb{V} \to \mathbb{R}$ per la quale esiste una funzione di mapping $\phi : \mathbb{V} \to \mathbb{F}$ dove $\mathbb{F}$ è uno spazio di Hilbert su cui daremo maggiori dettagli in seguito. Intanto vediamo questo esempio per comprendere meglio la definizione appena fornita

$$K(x, y) = < \phi(x), \phi(y) > \tag{26}$$

Vediamo un esempio per cominciare ad intuire la questione

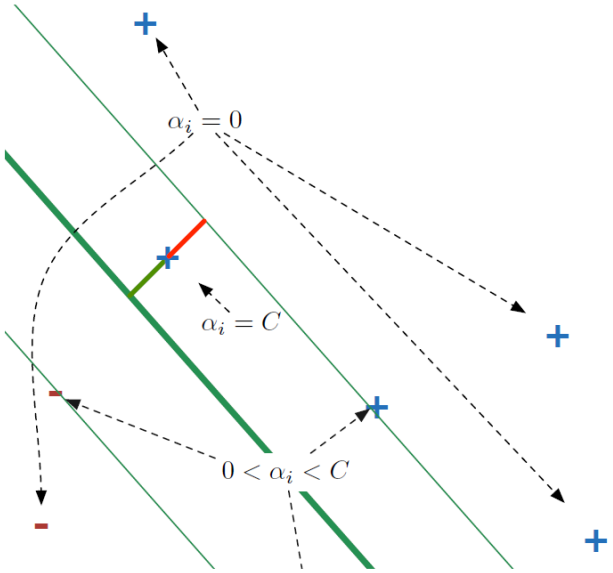

Figura 27: Posizione di un generico punto in funzione dei valori α_i, C

Esempio 1 : supponiamo di considerare la seguente funzione:

$\phi(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2, c)$

consideriamo anche la forma della funzione di classificazione usata nella svm

$\hat{c}(x) = segno(w \cdot x - t)$.

Andiamo a sostituire $w \cdot x$ con $K : \phi(w)\phi(x)$ nella funzione di classificazione. Quello che otteniamo è:

$\hat{c}(x) = segno(K(w, x) - t) = segno(\phi(w)\phi(x) - t) = segno(\phi(w)\phi(x) - t) =$

$= segno((w_1^2, \sqrt{2}e_1w_2, w_2^2, c) \cdot (x_1^2, \sqrt{2}x_1x_2, x_2^2, c) - t) =$
$= segno(w_1^2x_1^2 + 2w_1w_2x_1x_2 + w_2^2x_2^2 + c^2 - t)$

Notiamo che adesso il classificatore è quadratico e non più lineare.

Forniamo qualche dettaglio in più rispetto agli spazi di Hilbert. A noi è sufficiente sapere che in uno spazio di Hilbert è definita l'operazione di *prodotto interno*, ovvero una generalizzazione del prodotto scalare ad uno spazio vettoriale arbitrario. È definito assiomaticamente come la funzione che associa ad ogni coppia di vettori uno scalare, solitamente un numero reale. Per essere definito

prodotto interno la funzione utilizzata deve godere delle tre seguenti proprietà:

- linearità : $< a\mathbf{x}_1 + b\mathbf{x}_2, y >= a < \mathbf{x}_1, y > +b < \mathbf{x}_2, \mathbf{y} >$
- simmetria: $< \mathbf{x}, \mathbf{y} >=< \mathbf{y}, \mathbf{x} >$
- definita positiva: $\forall \mathbf{x}\ t.c\ < \mathbf{x}, \mathbf{x} > \ \geq 0$

Ricordiamo ancora una volta perché utilizzare i kernel. Abbiamo diversi motivi:

- Di rappresentazione: estende i classificatori lineari per trattare problemi non linearmente separabili
- Computazionali: calcola un criterio di similarità tra vettori in modo molto semplice
- Teorici: esistono delle condizioni precise per stabilire se una certa funzione sia un kernel o meno. Inoltre esistono delle proprietà di composizione che ci permettono di creare molto facilmente nuovi kernel

Quali sono i tipi di kernel più noti?

- Polinomiali di grado d: della forma $K(x,y) = (x \cdot y)^d$ oppure $K(x,y) = (x \cdot y + 1)^d$
  Notiamo che per $d = 1$ abbiamo il kernel identità che non effettua alcuna operazione. Per $d = n$ abbiamo un kernel che porta i dati in uno spazio di dimensione esponenziale in d
- Gaussiani: della forma

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

Ad ogni modo esistono molti altri kernel. Uno molto usato in biologia consiste nel mappare una sequenza di caratteri nel numero di volte che ogni sotto sequenza possibile(di dimensione fissata) compare in quella sequenza.
Un aspetto importante da ricordare è che i kernel possono essere pensati come funzioni di similarità legate al coseno. Infatti quando due vettori sono "simili" puntano (circa) in una stessa direzione e dunque il loro prodotto scalare è maggiore[9].
In generale, creata una funzione di similarità associata ad un kernel, dovremmo cercare un criterio per stabilire se il kernel sia valido. Ovvero dovremmo cercare un metodo per essere sicuri che K valuti il prodotto interno nello spazio di Hilbert corrispondente H. Ci viene in aiuto la *condizione di Mercer*

Teorema 3.1 (Condizione di Mercer)**.** *Una funzione K è un kernel valido se e solo se per un numero finito di punti $\{x_1, \ldots, x_m\}$, la matrice K, in cui ogni elemento è definito come $K_{ij} = K(x_i, x_j)$, è simmetrica e semi definita positivamente*[10].

[9] perché l'angolo compreso fra essi è ridotto, questo fa crescere il prodotto scalare

[10] Ricordiamo che una matrice è semi definita positivamente se vale che $\forall z, z^T kZ \geq 0$

Dimostrazione. ($\Rightarrow$) la simmetricità della matrice si ottiene facilmente dalla definizione di prodotto interno. Quello che interessa è invece mostrare che la matrice è semi definita positivamente. Dunque andiamo a calcolare

$$\mathbf{z}^T\mathbf{K}\mathbf{z} = \sum_i \sum_j z_i z_j \mathbf{K}_{i,j} = \sum_i \sum_j z_i z_j K(\mathbf{x}_i, \mathbf{x}_j) = \sum_i \sum_j z_i z_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$$

Ma per linearità dell'inner product:

$$\sum_i \sum_j z_i z_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = \sum_i \sum_j \langle z_i \phi(\mathbf{x}_i), z_j \phi(\mathbf{x}_j) \rangle$$

E sempre per la linearità dell'inner product abbiamo che:

$$\sum_i \sum_j \langle z_i \phi(\mathbf{x}_i), z_j \phi(\mathbf{x}_j) \rangle = \sum_i \langle z_i \phi(\mathbf{x}_i), \sum_j z_j \phi(\mathbf{x}_j) \rangle =$$
$$= \langle \sum_i z_i \phi(\mathbf{x}_i), \sum_j z_j \phi(\mathbf{x}_j) \rangle = \langle \sum_i z_i \phi(\mathbf{x}_i), \sum_i z_i \phi(\mathbf{x}_i) \rangle \geq 0$$

Dimostrazione. ($\Leftarrow$): non dimostrabile in questa sede poiché troppo complessa □

Per concludere riportiamo alcune delle proprietà di composizione dei kernel di cui avevamo parlato in precedenza, riportando alcuni esempi a riguardo. Le composizioni dei kernel che stiamo per elencare preservano la condizione di Mercer. Ovvero se i kernel originali rispettavano la condizione di Mercer, anche i kernel che seguono la rispettano:

- $K(\mathbf{x}, \mathbf{y}) = K_1(\mathbf{x}, \mathbf{y}) * K_2(\mathbf{x}, \mathbf{y})$
- $K(\mathbf{x}, \mathbf{y}) = K_1(\mathbf{x}, \mathbf{y}) + K_2(\mathbf{x}, \mathbf{y})$
- $K(\mathbf{x}, \mathbf{y}) = \alpha K_1(\mathbf{x}, \mathbf{y})$ per $\alpha > 0$

18 / 12 /2017

4 Ensemble Learning

L'ensemble learning raccoglie molte tecniche che cercano di risolvere un problema di apprendimento combinando diversi modelli con la speranza di ottenere un risultato migliore di quello ottenuto dai singoli algoritmi.
Una domanda teorica a cui daremo una risposta in seguito è: *l'apprendimento debole è equivalente all'apprendimento forte?*
Dove l'apprendimento *debole* si verifica quando, per un problema di apprendimento riusciamo ad ottenere un errore di generalizzazione minore rispetto a tirare casualmente un dado. Basta ottenere risultati con accuratezza leggermente superiore al 50% in sostanza. Se invece l'accuratezza che otteniamo si

avvicina a piacimento al 100%, allora l'apprendimento viene detto *forte*. La domanda che abbiamo posto prima è importante perché se la risposta è affermativa, possiamo andare ad ottenere sempre una soluzione forte partendo da una debole.
Nell'ensemble learning prendiamo un insieme di T dataset $D_1, \ldots, D_T$ da cui il nostro algoritmo ottiene T modelli $M_1, \ldots, M_T$ che vengono combinati per ottenere un modello unico M.
Come possiamo ottenere M? Solitamente, supponendo di avere un caso binario, consideriamo il risultato ottenuto da ogni modello e lo pesiamo come segue:

$$M(x) = f(\sum_{i=1}^{T} w_i M_i(x)) \tag{27}$$

dove w_i è il peso associato e viene utilizzato per tenere in maggiore considerazione i modelli che riteniamo più affidabili. Invece f è solitamente la funzione *segno*.

4.1 Begging

Il primo algoritmo di ensemble learning è *bagging* che sta per *Bootstrap aggregation*. L'idea del bagging sta nel come vengono costruiti i dataset. L'algoritmo è quello che segue

Algorithm 1 Algoritmo di bagging per l'addrestamento di T modelli tramite re-sampling del dataset D originario

```
function BAGGING(D,T,𝒜)
    M ← ∅
    for i = 0 to T do
        D_i ← RE-SAMPLING(D)
        m_i ← EXEC(D_i, 𝒜)
        M ← M ∪ m_i
    end for
    return M
end function
```

Dove D è un dataset, T è il numero di classificatori che vogliamo combinare, mentre A è l'algoritmo di apprendimento. Costruisce una replica di bootstrap del dataset, andando a costruire un nuovo dataset partendo dall'originale in cui abbiamo tolto alcuni elementi che vengono sostituiti con alcune repliche dei dati restanti. Alla fine viene restituito l'insieme dei modelli. Queste repliche possono essere fatte perché non alterano la distribuzione dei dati.

Un aspetto importante per tutti gli algoritmi di ensemble è che vorremmo che i vari modelli restituiti da A siano differenti fra loro, ovvero vogliamo andare a promuovere la diversità. Questo viene fatto andando a costruire i dataset

di bootstrap che essendo un po' diversi fra loro[11] permettono la costruzione di modelli diversi.
I risultati ottenuti tramite un algoritmo così semplice sono comunque degni di nota e portano ad un miglioramento delle prestazioni rispetto ad altri modelli.
Spesso tramite begging andiamo ad ottenere dei modelli ad albero e al giorno d'oggi un particolare modello legato agli alberi è quello a *random forest* in cui una serie di alberi non potati vengono utilizzati come learner deboli. L'unica differenza con il modello che abbiamo mostrato poco fa è che vengono prese un sottoinsieme casuale delle feature di cardinalità $\sqrt{F}$. Questa scelta va incontro al concetto di diversità di cui abbiamo parlato prima.

4.2 AdaBoost

La seconda tecnica di ensemble learning è quella di *adaptive boosting*. Cominciamo col dare una definizione più formale di concetti *strongly learnable* e *weak learnable*.
Diremo che una classe di concetti è apprendibile in modo forte se esiste un algoritmo polinomiale in tempo che ottiene un basso errore con alta confidenza per tutti i concetti della classe. Notiamo che l'alta confidenza indica che otteniamo un certo errore piccolo a piacere con elevata probabilità.
Diremo invece che una classe di concetti è apprendibile in modo debole semplicemente rilasciando la condizione sull'accuratezza alta a piacere.
Se apprendimento forte e debole fossero equivalenti è una questione rimasta aperta per un certo periodo di tempo. Nel 1988 Kearns fornisce delle argomentazioni che fanno pensare che apprendimento forte e debole siano distinti. Kearns mostra che sotto certe condizioni l'apprendimento debole differisce da quello forte.
Nel 1990 invece Schapire dimostra che una classe di concetti C è apprendibile in modo debole se e solo se è apprendibile in modo forte.
Schapire propone anche un algoritmo costruttivo che apprende in modo forte partendo dall'apprendimento debole. Tale algoritmo ha nome *adaBoost*.
Dunque l'error rate ottenuto utilizzando adaBoost tende a decrescere in modo esponenziale all'aumentare delle iterazioni.
L'intuizione fondamentale è che, nel momento in cui viene usato un algoritmo di base per apprendere, ci saranno degli esempi che sono "facili" mentre altri sono più "difficili". Gli esempi su cui l'algoritmo ottiene performance peggiori vengono pesati maggiormente, di modo che l'algoritmo per raggiungere la soglia di accuratezza superiore al 50% debba apprendere in particolare almeno qualcuno di quegli esempi.
All'inizio tutti gli esempi hanno lo stesso peso. In seguito gli esempi valutati bene vengono pesati meno, quelli valutati male vengono pesati di più.
Un aspetto veramente interessante di adaBoost è che dopo aver raggiunto una prestazione perfetta sul training set, l'errore sul test set continua a calare poiché

[11] Solitamente un dataset di bootstrap contiene il 63% del dataset originale

viene ottimizzato il margine.
Andiamo a vedere nel dettaglio l'algoritmo adaBoost

Algorithm 2 Algoritmo di adaboost che itera T volte

function ADABOOST(D,A,T)
  $\mathbf{w}^1 = [1/|D|]^n$
  for $t = 0$ **to** T **do**
    $m_t =$EXEC$(A, \mathbf{w}^t, D)$
    $\epsilon_t = \sum_{i=1}^{n} w_i^t I[y_i \neq m_t(\mathbf{x}_i)]$
    $\alpha_t = ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
    for $i = 1$ **to** n **do**
      $\mathbf{w}_i^{t+1} = \mathbf{w}_i^t \mathsf{exp}\left(-\alpha_t y_i m_t(\mathbf{x}_i)\right)$
    end for
    $\mathbf{w}^{t+1} \leftarrow \mathbf{w}^{t+1} / \mid\mid \mathbf{w}^{t+1} \mid\mid$
  end for
  return $M(x) = \sum_{t=1}^{T} \alpha_t m_t(x_i)$
end function

L'algoritmo prende in input un insieme di esempi e un algoritmo. Itera T volte e inizia andando a generare il modello m_t. In seguito calcola l'errore ϵ_t basato sugli errori pesati. In seguito viene calcolato il valore α_t che determina quanto peserà il modello m_t. Poiché per ipotesi $\epsilon_t < 0.5$, allora vale sempre che $\alpha_t > 1$. In seguito viene aggiornato il vettore dei pesi w_i^{t+1} andando a ridurre i pesi degli esempi classificati correttamente, ovvero quelli per cui $m_t(x_i) = 1$. Va invece ad aumentare il valore dei pesi degli esempi classificati erroneamente. Infine i pesi vengono normalizzati.

18 / 12 /2017

L'idea importante è che AdaBoost cerca di minimizzare in modo greedy la loss esponenziale pesata. In particolare vengono selezionati in modo greedy le ipotesi deboli e i pesi α_t.
Anzitutto notiamo che

$$M_T(x) = M_{T-1}(x) + \alpha_T m_T(x)$$

Se andiamo a calcolare la loss esponenziale, per definizione sommiamo iterando su tutti gli esempi il termine $e^{-y_i M_T(x_i)}$ dove ricordiamo che come altre volte in precedenza l'esponente è positivo se stiamo classificando bene, negativo altrimenti. Dunque:

$$E = \sum_i e^{-y_i M_T(x_i)} = \sum_i e^{-y_i(M_{T-1}(x_i)+\alpha_T m_T(x_i))} = \sum_i e^{-y_i M_{T-1}(x_i)} e^{-y_i \alpha_T m_T(x_i)}$$

Ma ponendo $w_i^T = \begin{cases} e^{-y_i M_{T-1}(x_i)}, & \text{se } T > 1 \\ 1, & \text{se } T = 1 \end{cases}$

Notiamo che questa formulazione di w_i^T non è diversa da quella dell'algoritmo. Infatti vale che:

$w_i^T = e^{-y_i M_{T-1}(x_i)} = e^{-y_i \sum_{t=1}^{T-1} \alpha_t m_t(x_i)} = e^{-y_i M_{T-2}(x_i)} e^{-y_i \alpha_{T-1} m_{T-1}(x_i)} =$
$= w_i^{T-1} e^{-y_i \alpha_{T-1} m_{T-1}(x_i)}$

Dunque andiamo ad ottenere

$$E = \sum_i w_i^T e^{-y_i \alpha_T m_T(x_i)}$$

A questo punto andiamo a separare i punti classificati correttamente da quelli classificati in modo errato:

$$E = \sum_{\{i|y_i = m_t(x_i)\}} w_i^T e^{-\alpha_T} + \sum_{\{i|y_i \neq m_t(x_i)\}} w_i^T e^{\alpha_T} =$$
$$= \sum_{\{i|y_i = m_t(x_i)\}} w_i^T e^{-\alpha_T} + \sum_{\{i|y_i \neq m_t(x_i)\}} w_i^T e^{\alpha_T} + \sum_{\{i|y_i \neq m_t(x_i)\}} w_i^T e^{-\alpha_T} - \sum_{\{i|y_i \neq m_t(x_i)\}} w_i^T e^{-\alpha_T} =$$
$$= \sum_{\{i|y_i = m_t(x_i)\}} w_i^T e^{-\alpha_T} + \sum_{\{i|y_i \neq m_t(x_i)\}} w_i^T e^{-\alpha_T} + \sum_{\{i|y_i \neq m_t(x_i)\}} w_i^T e^{\alpha_T} - \sum_{\{i|y_i \neq m_t(x_i)\}} w_i^T e^{-\alpha_T} =$$
$$= \sum_i w_i^T e^{-\alpha_T} + \sum_{\{i|y_i \neq m_t(x_i)\}} w_i^T (e^{\alpha_T} - e^{-\alpha_T})$$

Dunque l'errore che vogliamo minimizzare deriva dalla somma di due termini, una sommatoria su tutti gli esempi(indipendentemente da come gli ho classificati) e da una sommatoria relativa ai termini che ho classificato male. Il primo termine è una costante, quindi cercheremo di minimizzare il secondo termine relativo agli elementi classificati male.
Dobbiamo però mostrare che la quantità

$$\sum_{\{i|y_i \neq m_t(x_i)\}} w_i^T (e^{\alpha_T} - e^{-\alpha_T})$$

è sempre positiva. Ma questo è vero poiché ogni elemento di w_i è compreso tra 0 e 1. Invece, ricordando che $\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$ dove $\epsilon_t < 0.5$. Ma dunque l'argomento del logaritmo è maggiore di 1 e allora il logaritmo è maggiore di 0. Dunque anche la quantità $(e^{\alpha_T} - e^{-\alpha_T})$ è maggiore di zero.

Per minimizzare il secondo termine ne calcoliamo la derivata rispetto agli α_t

$$\frac{\partial E}{\partial \alpha_t} = \sum_{\{i|y_i \neq m_t(x_i)\}} w_i^T e^{\alpha_T} - \sum_{\{i|y_i = m_t(x_i)\}} w_i^T e^{-\alpha_T}$$

da cui, ponendo a zero la derivata

$$\alpha_T = \frac{1}{2} ln(\frac{1-\epsilon_T}{\epsilon_T})$$

che è proprio come vengono calcolati gli α_T nell'algoritmo.

Un problema che potremmo porci è: adaBoost vuole in input un dataset pesato. Ma nessuno dei nostri algoritmi fornisce in input un dataset pesato. Esistono due soluzioni a questo

1. Usare una tecnica di *reweighting* in cui andiamo a modificare gli algoritmi di apprendimento di modo che tengano conto dei pesi sui dati
2. Usare una tecnica di *dataset resampling* con cui creiamo un nuovo dataset duplicando gli esempi di modo che la proporzione di esempi duplicati sia corrispondente al peso che ci interessa associare ad un certo esempio. Oppure andando a selezionare gli esempi su cui fare apprendimento utilizzando una funzione di distribuzione desiderata. Ovvero dividiamo l'intervallo $[0, 1]$ in intervalli proporzionali ai vari pesi. Dopo generiamo un numero casuale tra in $[0, 1]$ e andiamo a selezionare l'esempio relativo al segmento in cui ricade il numero casuale. Quante volte andiamo ad estrarre valori? Solitamente tante volte quanto è la dimensione del dataset.
   Non sempre però usiamo resampling. Infatti costa di più e in certi casi è meno preciso perché alcuni esempi non vengono neanche presi in considerazione.

4.3 Perché l'ensemble funziona

Abbiamo diversi modi empirici per vedere che l'ensemble funziona. Noi ne vedremo due: uno basato sul discorso bias/variance e uno basato su altre tecniche empiriche.

Sappiamo che l'errore è formato da tre parti: bias, varianza, rumore. È stato mostrato in modo empirico che tutti i metodi di ensemble riducono sia il bias che la varianza. Alcuni metodi agiscono di più su una componente o su un'altra. Begging ad esempio riduce per lo più la varianza. Dunque tenderemo ad usare begging per quei modelli ad alta varianza come gli alberi di decisione. Per comprenderlo ci focalizziamo su un singolo esempio e verifichiamo se le singole componenti dell'ensemble commettono un errore su quegli elementi. Chiamiamo con X il numero di errori su un certo numero T di dati. Supponendo che il nostro classificatore binario ci fornisca una soluzione corretta con probabilità maggiore di 0.5, sappiamo che X errori su T tentativi si distribuiscono secondo una binomiale. Dunque $X \sim Bin(p, T)$.
L'intero ensemble commette un errore quando più della metà dei suoi componenti commettono un errore. Ovvero:

$P(X > \lfloor T/2 \rfloor) = \sum_{x=\lfloor T/2 \rfloor+1}^{T} P(X = x)$

Questa probabilità tende a zero all'aumentare del numero di modelli che uso.

AdaBoost invece riduce sia il bias che la varianza. In particolare AdaBoost viene usato anche per classificatori di cui vogliamo correggere il bias elevato.

Quali sono le motivazioni che spingono all'utilizzo dell'ensemble learning? Ne abbiamo tre

- Statistica: non potendo calcolare il classificatore ottimo di Bayes(che è ideale), vado a calcolare diversi classificatori e ne considero la media. Se l'algoritmo è scritto decentemente ogni classificatore avrà una probabilità abbastanza alta di classificare bene. Dunque andiamo a mediare una serie di buoni risultati, che ci avvicina al classificatore ottimo di Bayes
- Rappresentazionale: mediando i vari classificatori possiamo "uscire" dallo spazio delle ipotesi. Talvolta il vero classificatore potrebbe essere davvero esterno al nostro spazio delle ipotesi
- Computazionale: supponiamo di voler scegliere tra tutti i polinomi di un certo grado il migliore per i nostri dati. Solitamente ci muoviamo nello spazio delle ipotesi guidati dalla funzione di loss. Ma poiché lo spazio delle ipotesi è popolato da molti minimi locali, solitamente si preferisce eseguire più volte uno stesso algoritmo più volte come succede nelle tecniche di ensemble