**Introduction**

Sommario

[Introduzione 2](#_Toc61166506)

[Cos’è l’apprendimento automatico 2](#_Toc61166507)

[No free lunch Theorem 2](#_Toc61166508)

[Risoluzione di un task 3](#_Toc61166509)

[Task 4](#_Toc61166510)

[Task predittivi e descrittivi 4](#_Toc61166511)

[Overfitting e underfitting 4](#_Toc61166512)

[Models 4](#_Toc61166513)

[Classificatori geometrici 5](#_Toc61166514)

[Classificatore probabilistico 5](#_Toc61166515)

[Classificatore logico 5](#_Toc61166516)

[Features 5](#_Toc61166517)

[Tasks predittivi 6](#_Toc61166518)

[Classificatori 6](#_Toc61166519)

[Classificazione binaria 6](#_Toc61166520)

[Roc Plots 9](#_Toc61166521)

[Oltre alla classificazione binaria 17](#_Toc61166522)

[Regressione 20](#_Toc61166523)

[Task predittivi vs task descrittivi 23](#_Toc61166524)

[Clustering 23](#_Toc61166525)

[Task descrittivi 24](#_Toc61166526)

[Supervised subgroup-discovery 24](#_Toc61166527)

[Association rules 24](#_Toc61166528)

Lezione 1

# Introduzione

## Cos’è l’apprendimento automatico

Capiamo cosa sia l’apprendimento automatico e quali siano le componenti principali che lo costituiscono. Vediamo un sistema di apprendimento automatico come la combinazione di tre grandi concetti:

1. **Task**, cioè il compito che vuole affrontare il sistema di apprendimento automatico.  
   Ci sono diversi task risolti dal sistema di apprendimento, quelli più noti sono:
   * Classificazione
   * Regressione
   * Stima di probabilità
   * Clustering
   * …
2. **Models**, ovvero il mondo in cui si modella il problema:
   * Modelli geometrici
   * Modelli logici
   * Alberi di decisione
   * Naïve bayes (modello probabilistico)
   * …
3. **Features**, ovvero come viene descritto il problema e ciò viene fatto in base al model e al task. Ci sono diversi tipi di features:
   * Numerica
   * Categoriche
   * costruite
   * trovate
   * …

Definiti Task, Features e modelli possiamo dare una definizione di **apprendimento automatico** (o machine learning):

*L’apprendimento automatico è lo studio sistematico degli algoritmi che migliorano la propria conoscenza o performance con l’esperienza.*

## No free lunch Theorem

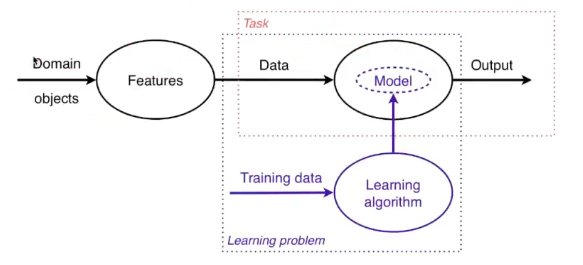
Il **No Free Lunch Theorem** dice che:

*“tutti gli algoritmi di apprendimento sono equivalenti in media”.*

Nonostante ciò, a seconda del problema esiste un algoritmo con **bias** **induttivo** migliore rispetto ad un altro (per questo vedremo diversi algoritmi), dove per bias induttivo si intende un pregiudizio, cioè il modo in cui un algoritmo fa una previsione del futuro.

Le soluzioni degli algoritmi non sono tutte identiche, possono dare soluzioni molto differenti. Ciò che dice il teorema è che senza alcuna ipotesi sul mondo vero, si assume che tutti gli algoritmi siano equivalenti per quanto riguarda le previsioni future.

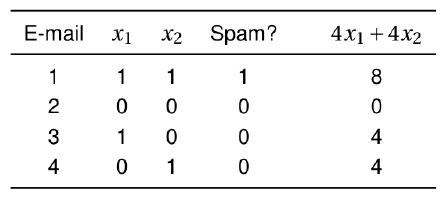
## Risoluzione di un task



Possiamo immaginarci uno strumento di apprendimento automatico come segue:

* Abbiamo un dominio che ci fornisce oggetti da analizzare descritti tramite features.
* Le features vengono date in pasto ad un algoritmo di apprendimento che si occupa di un certo task.
* L’algoritmo crea un modello e sulla base di questo modello quando arrivano nuovi dati restituisce un certo risultato.

I **Task** sono risolti dai modelli. Il problema di apprendimento (ovvero qual è l’algoritmo migliore per risolvere questo task) viene risolto dall’algoritmo di apprendimento, il quale produce un modello che risolve il task.



L’apprendimento automatico si occupa di usare le **features** (x1 e x2) giuste per costruire **modelli** (4x1 + 4x2) per risolvere un particolare **task** (dividi le mail spam da quelle ham).

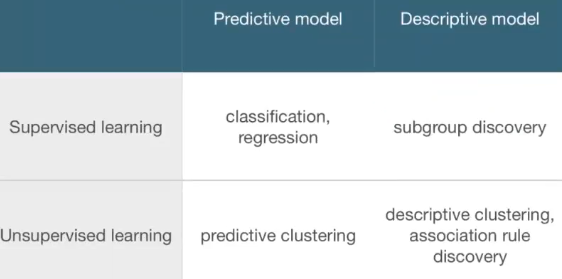
## Task

### Task predittivi e descrittivi

La prima grossa categorizzazione dei task che possiamo fare è

* **Task predittivi**, ovvero quelli in cui si vuole predire qualcosa.   
  Inizialmente vengono fornite un certo numero di istanze all’inizio da cui l’algoritmo di apprendimento impara qualcosa e infine cercare di predire qualcosa sulle istanze successive. I task predittivi più comuni sono:
  + **Classificazione**, cerca di predire un valore non numerico discreto e definito;
  + **Regressione**, simile a quello di classificazione ma la variabile che cerca di predire è numerica;
  + **Clustering**, si cerca di trovare dei raggruppamenti spontanei di oggetti nel mondo, e può essere usato sia in maniera predittiva che descrittiva;
* **Task descrittivi**, sono quelli che vogliono descrivere i dati in qualche modo. Hanno l’obiettivo di imparare qualcosa dai dati forniti che ci permetta di dire qualcosa su qualche nuova istanza che viene fornita

Di seguito viene mostrato un riassunto dei setting di apprendimento



Per **supervisionato** si intende che è presente dell’informazione in più fornita insieme ai dati (ad esempio una immagine con descrizione).

### Overfitting e underfitting

Una questione che bisogna conoscere bene è che i modelli dedotti possono soffrire di un problema chiamato **overfitting**, ovvero quando si ha maggiore accuratezza sui modelli simili a quelli del training set rispetto ad altri che si discostano un po’ (**underfitting** è il contrario). Vedremo le tecniche per bilanciare questo problema.

## Models

Cerchiamo di capire quali sono i modelli che spesso vengono utilizzati:

* **Geometrici**, si vuole modellare il problema utilizzando qualche metodo geometrico;
* **Probabilistici**, si vuole trovare una formalizzazione del tipo “qual è la probabilità che x accada?”;
* **Logici**: si modella il problema utilizzando delle proposizioni logiche e valutandole

### Classificatori geometrici

Costruiscono una funzione con n variabili a cui viene associato un peso

### Classificatore **probabilistico**

È un classificatore che utilizza la teoria del calcolo delle probabilità per effettuare le previsioni (min circa 15-20)

### Classificatore logico

È uno che utilizza come modello per descrivere i dati e il risultato il formalismo logico.

## Features

Sono gli elementi di base per descrivere gli oggetti con cui fare apprendimento. Le features consentono due utilizzi

* Per fare lo split delle decisioni
* Per calcolare la decisione stessa

# Tasks predittivi

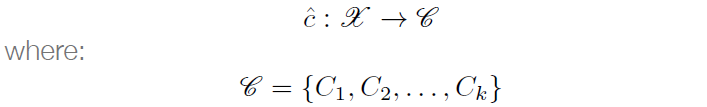
Lezione 2

Nei task predittivi abbiamo un data set etichettato (etichette date da un essere umano). Un esempio può essere l’etichettatura delle immagini (tigre, gatto, cavallo, …). L’obiettivo è usare un algoritmo di apprendimento che dato esempi ed etichetta riesca a indurre una regola generale per classificare **nuovi** **esempi** usando l’etichetta opportuna.

A seconda di come è descritta l’etichetta data abbiamo diversi problemi. Se vogliamo distinguere un numero finiti di etichette parliamo di **classificazione**, se invece vogliamo predire un numero reale si parla di **regressione**, se vogliamo predire la probabilità di appartenere ad una certa classe si parlerà di **stima di probabilità**. Nello **scoring** associamo un numero alla predizione che rappresenta la confidenza della predizione dell’esempio nella classe positiva. Nel **ranking** invece di classificare gli esempi cerchiamo di ordinarli in un ordine particolare.

## Classificazione

Un classificatore è un **mapping** **ĉ** che va dallo spazio degli esempi all’insieme finito e discreto che contiene le etichette (proprietà) da poter associare a ciascun esempio



Dove

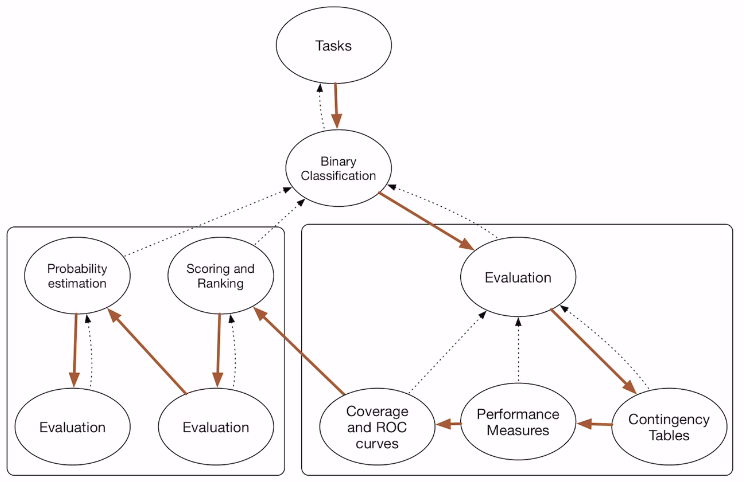
* ***X*** è lo spazio degli esempi, l’insieme di tutti i possibili esempi (potrebbe essere uno spazio vettoriale e non, perché non per forza la descrizione dell’oggetto contiene solo numeri, magari sono immagini)
* ***C*** è l’insieme delle etichette da associare agli esempi quando viene fatta classificazione
* **ĉ** è un’approssimazione di qualche concetto vero che esiste in natura

Un **esempio** è una coppia (x, c(x)) formata da istanza ed etichetta corretta (idealmente è la classe corretta, ma potrebbe non esserlo sempre).

Apprendere un classificatore vuol dire costruire una funzione ĉ tale che approssimi il più possibile il concetto reale c, approssimandolo anche su esempi non ancora visti.

### Classificazione binaria

Andremo a parlare delle seguenti cose:



Tutti gli altri casi di classificazione possono essere ricondotti alla classificazione binaria (non sempre utile perché se abbiamo una tecnica ad hoc per il problema allora useremo quella).

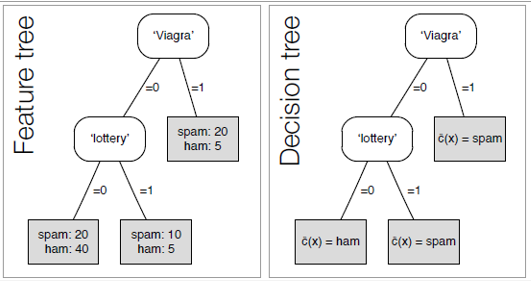
Il caso più semplice di classificazione è la **classificazione binaria** in cui solitamente C = {0,1}. Parleremo di come valutarlo, di tavole di contingenza, introdurremo i plot, passeremo a parlare di task collegati alla classificazione binaria e parleremo di scoring e ranking per poi passare alla stima di probabilità, che non è più classificazione binaria.

Introduciamo

* **Features tree**, è un albero in cui nei nodi ci sono i valori dei singoli attributi. Nelle foglie non ci sono delle decisioni ma ci sono esempi del data set
* **Decision tree**, si costruisce a partire da un features tree, andando a mettere nelle foglie le predizioni

Nel decision tree nelle foglie abbiamo un’etichettatura, nel feature tree abbiamo invece degli esempi.

In un feature tree introdurremo un intero data set, se, ad esempio, una mail contiene la parola **viagra**, allora inserisco le mail che hanno quella parola andando ad indicare quante sono spam e quante non lo sono. Uguale verrà fatto con **lottery**.



In base alle etichette finali scelte nel decision tree, come diciamo quanto bene fa il nostro modello? Per valutare la bontà dell’albero basta vedere quanti errori faccio, andando a vedere **l’accuratezza**

*Accuratezza = somma delle previsioni corrette / previsioni totali*

**

L’accuratezza sul training set è fuorviante, perchè potrei fare sempre il 100%, però vogliamo rendere il sistema generale per istanze che non conosciamo. Le vere misure di performance che ci interessano sono su mail nuove mai viste, li di solito le performance decrescono (dipende da quanto è grande il data set, ci sono tecniche che ci dicono come fare nel caso di pochi dati). Quindi per valutare veramente le performance dovremo valutare il classificatore sul test set.

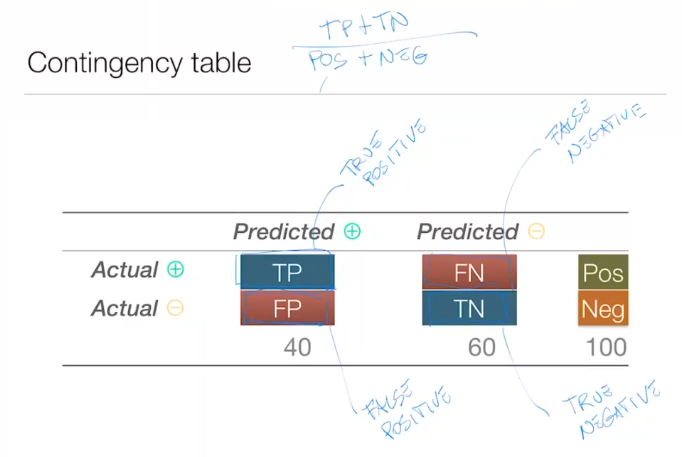
**PS. test set = data set - training set.**

Ci sono **altre** **misure** **di** **bontà**, per introdurle è utile introdurre il concetto di **tavole di contingenza**. È una tabella 2x2, in cui nelle righe uno riporta le etichette attuali presenti nel data set e nelle colonne riporto le etichette predette nel classificatore

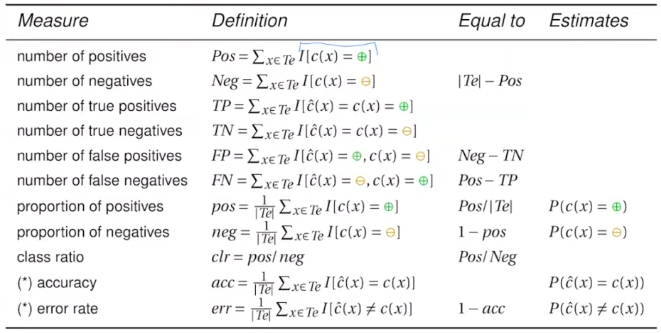
Immagine che contiene tavolo

Descrizione generata automaticamente

* Ultima colonna prima riga ci da il totale degli esempi positivi del data set (pos)
* Ultima colonna seconda riga ci da il totale degli esempi negativi del data set (neg)
* Ultima riga, prima e seconda colonna, ci da il numero di predizioni positive e negative (ci interessano poco)



A partire da questi dati (TP, FN, Pos, …) si possono calcolare delle misure di performance:



L’**accuratezza** conta il numero di classificazioni corrette

PS. ***Te*** che sta per il test set e I è la funzione indicativa (tra le quadre c’è la condizione di verità)

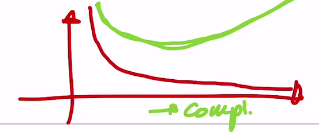


Altre misure interessanti sono il

* **true positive rate** (detta anche **rate**, **sensitivity** e **recall**), è la divisione del numero dei true positivi con il numero degli esempi positivi (TP/Pos)
* **true negative rate** (TN/Neg)
* false positive rate
* false negative rate
* **precisione** (chiamata anche **confidence**), numero dei true positivi / (true positive + false positive) (TP/(TP+FP)) 🡪 compete con la recall, un modo facile per avere una recall perfetta e dare tutto positivo, però in tal caso la precisione sarà molto bassa

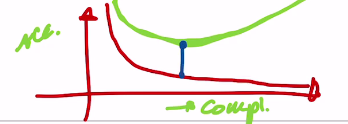
Di solito useremo **accuratezza** e **average recall** (la media tra il True positive rate e il true negative rate).

Fare overfitting significa prendere la curva che mi dice come sto procedendo sul training set (può arrivare a zero), prendere la curva che mi dice come sto precedendo sul test set (può andare verso l’alto)



* asse x: complessità del modello
* asse y: accuratezza

overfitting è la differenza in qualunque punto del grafico tra funzione test set e training set



Nel punto in blu ci va ancora bene (perché è un overfitting accettabile), ma guardando più a destra vediamo che è patologico, quindi non va più bene. Come risolverlo? Bisogna complicarlo (nel caso di underfitting) o rilassarlo (nel caso di overfitting). Quindi **overfitting** c’è quando il modello è troppo specifico per il riconoscimento degli esempi conosciuti a discapito di quelli non conosciuti, **underfitting** e quando non riconosce bene neanche gli esempi conosciuti.

Come capire se il problema di discordanza di accuratezza tra training set e data set è causata dall’overfitting o dai pochi dati?

* Dipende da come sono fatti i dati. Se il dataset è molto rumoroso, è facile che il modello overfitti verso il dataset.

Quindi se il dataset è pulito allora è possibile risolvere l’overfitting, se è rumoroso è difficile da risolvere.

Osservare le due curve aiuta molto (se non arrivano mai molto vicine e poi si allontanano vuol dire che non si può fare meglio con quel modello)

Non si hanno delle regole precise per individuare l’overfitting, bisogna studiarlo ogni volta e analizzare le curve.

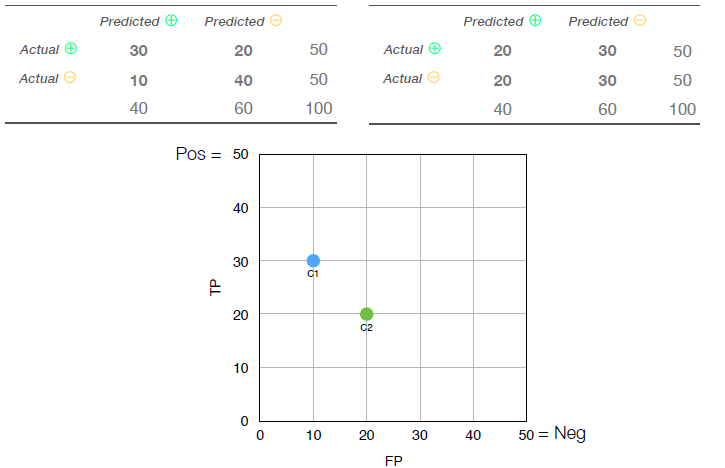
Che legame c’è tra numero di esempi e overfitting?

* Avendo pochi esempi è più facile fare overfitting, a parità di complessità del modello; con tantissimi esempi invece si rischia di fare underfitting

Due possibilità di data set non ben formato

* Mal formato in partenza
* Non più al passo con i tempi (le etichette nel corso del tempo sono cambiate e quindi il dataset non è più valido)

È possibile rappresentare la **tavola di contingenza** per poter cogliere a colpo d’occhio le performance del classificatore? Si, attraverso il **coverage plot**



Sull’asse x ci sono i falsi positivi, sull’y ci sono i veri positivi.

Le 2 tavole di contingenza sopra il coverage plot rappresentano le performance di due classificatori differenti. Nel coverage plot rappresentiamo solo FP e TP (trascurando FN e TN), sto perdendo informazione? No, perché si possono ricavare

* FN = Pos – TP
* TN = Neg - FP

Il classificatore migliore è c1 perché è quello che fa meno FP (errori di etichettatura) e fa più TP (etichette corrette).

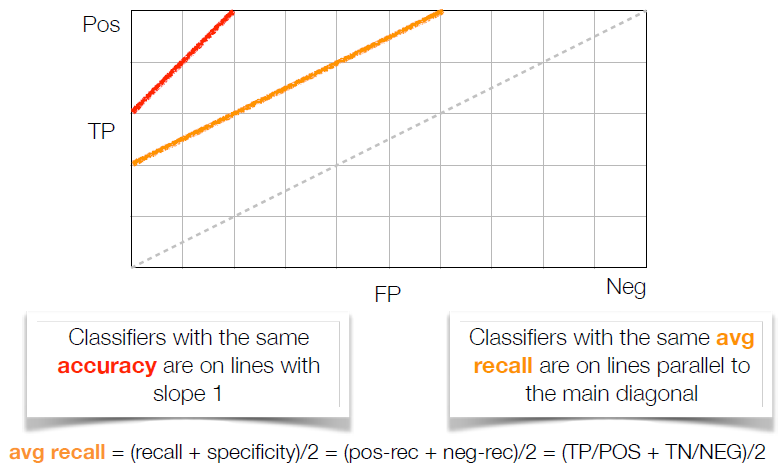
Un classificatore posizionato in basso a sinistra sarebbe un buon classificatore perché basterà etichettare il contrario di quello che dice. La diagonale che parte da zero e va verso l’alto è il posto peggiore in cui si può posizionare il classificatore

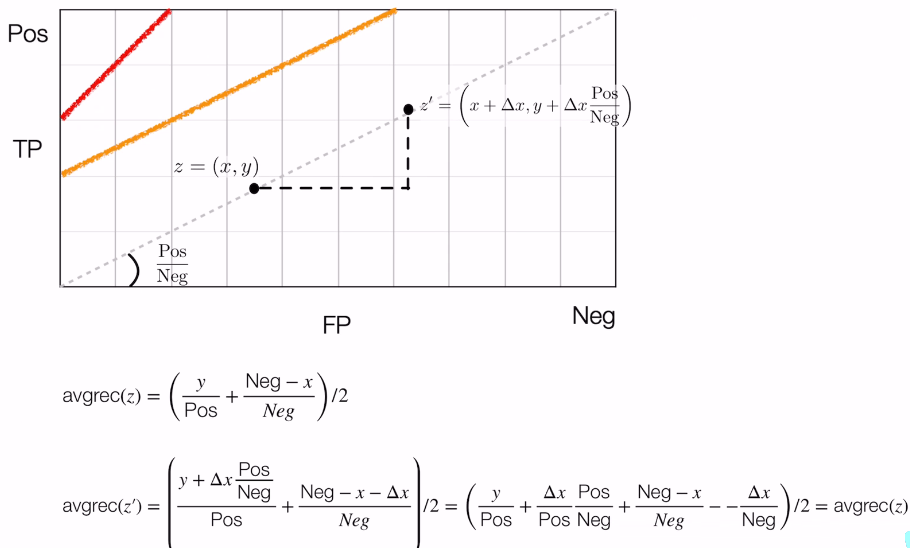
La zona in alto a sinistra viene chiamato **Roc heaven**, quella in basso a destra **Roc hell**

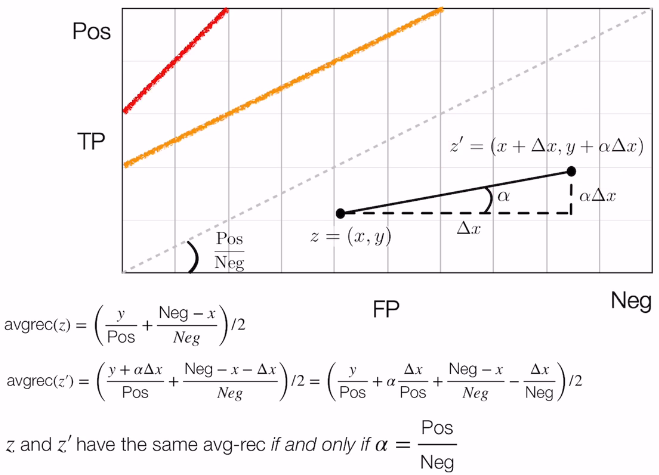
C1 e C2 sono le due tabelle. Stiamo mettendo i valori, presi dalla tabella di contingenza, all’interno di un grafico, il **coverage plot**, basta mettere TP e FP tralasciando TN e FN perché sapendo ad esempio che il totale actual + è 50 e avendo TP = 30 allora possiamo risalire a FN che sarà 20 in questo caso. Il coverage plot serve per capire quale tabella è meglio di un’altra

Il caso ideale sarà avere TP = 50 e FP = 0, la tabella che si colloca in alto a sinistra e la tabella migliore. Il punto in basso a destra invece non è la cosa peggiore che può succedere perché sappiamo che se sbaglia sempre basterà fare il contrario. La cosa peggiore che può succedere è trovarsi nella retta che divide il grafico in due (50% può avere ragione 50% può avere torto).

* Se 2 classificatori stanno in una linea con pendenza 1 allora hanno la stessa **accuratezza**, questo perché ad esempio
  + Prendiamo **x** in posizione (2,2) e **y** in posizione (3,3)
  + Sappiamo l’accuratezza di x si calcola con la formula Accuratezza(x) = (TP + TN) /TOT = c
  + Sappiamo che TP(y) = TP(x) + 1 e quindi che TN(y) = TN(x) – 1
  + Abbiamo che Accuratezza(y) = (TP(x) + 1 + TN(x) – 1) / TOT = c
  + E da ciò abbiamo che **Accuratezza(x) = Accuratezza(y) = c**
* Se i classificatori stanno su una linea parallela alla diagonale principale allora hanno la stessa **average recall**.







Muovendomi quindi sulla diagonale principale non cambia l’average recall.

**PS. se Pos e Neg coincidono avg recall = accuracy**

### Roc Plots

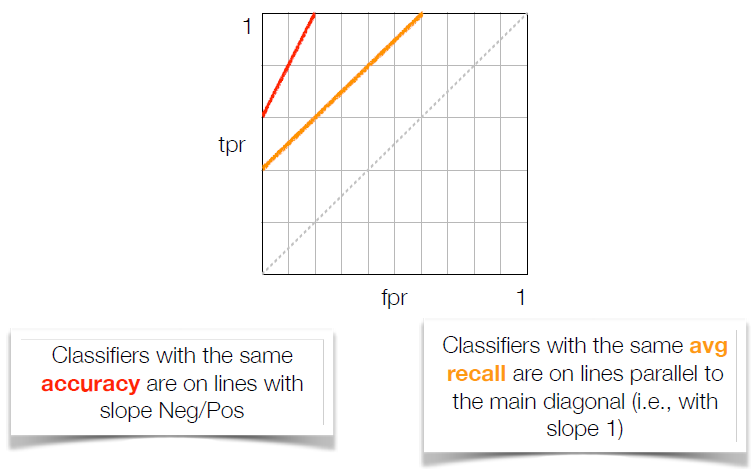
Lezione 3

Si tratta di una variante dei coverage plot (facile passare da un tipo di plot all’altro). Nei coverage posso inserire classificatori testati solo sullo stesso dataset.

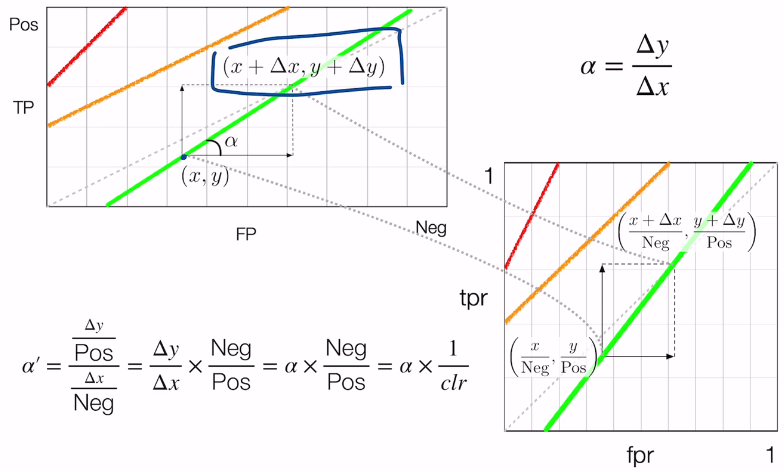
L’idea del **Roc Plot** è che il numero di esempi viene stato normalizzato a 1 (asse x e y terminano a 1). Rappresenta la percentuale di falsi e veri positivi ottenute dal nostro classificatore. Servono quando si vuole mettere a confronto classificatori con data set diversi.

A volte ci sono degli algoritmi di apprendimento che variano secondo qualche parametro, ad esempio il parametro di soglia. A volte serve scegliere il parametro migliore. Per farlo si inseriscono i diversi casi nel roc plot e si valuta qual è il migliore.

Cosa significa per le nostre proprietà? Avremo proprietà analoghe, ma le rette potrebbero avere pendenze diverse (perché il roc plot è un quadrato, il coverage non per forza, dipende da Neg e Pos)



La dimostrazione è molto simile a quella del coverage plot



* (x, y) nel roc plot diventa (x/neg, y/pos) e così via.
* Alfa’ è l’angolo che ha la retta verde.

**Cosa ce ne facciamo delle curve Roc e Coverage?**

Potrebbe non esserci grande beneficio, perché non sempre si usano tanti classificatori. Ma è utile quando sto usando uno **scoring classifier**. In questo caso avremo un valore di confidenza di quanto è buona la classificazione. Questa confidenza viene usata per ordinare gli esempi (da dx a sx). più si è a destra più l’esempio ha classe positiva. Avendo n esempi posso costruire n-1, n+1, infiniti classificatori, oppure dipende da come sono suddivise le etichette degli esempi

Immagine che contiene testo

Descrizione generata automaticamente

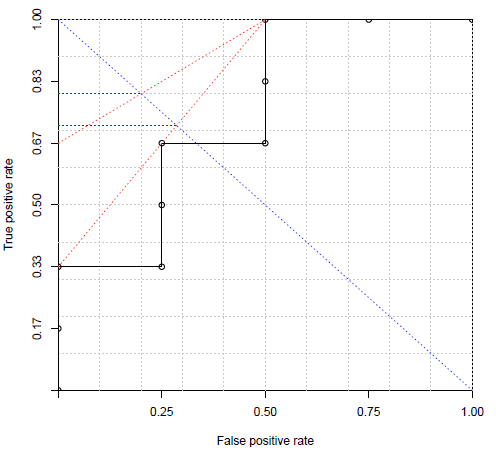
Il punto è dove mettere il threshold. Indipendentemente da dove mettiamo il threshold è dove lo mettiamo e poi ci chiediamo quale di questi classificatori è il migliore, poi andremo a creare una curva in un roc plot, più la curva forma un angolo di 90 gradi in alto a destra meglio è. La curva mi dice chi è il miglior classificatore e quanto è buono il modello che sto guardando.

Dobbiamo chiederci qual è il punto migliore nella curva.

Qua sotto si hanno scalini perché è stato costruito su un albero di decisione di pochi nodi e con esempi con stesso score (è quello che abbiamo visto su lottery e viagra).

Quale di questi punti è il migliore? Ovviamente uno dei tre vicini al roc heaven, ma quale dei tre? In rosso abbiamo 2 possibili rette che rappresentano l’accuratezza dei classificatori. Sono due perché essendo un grafico roc, l’isometria dell’accuratezza dipende dagli esempi in input (dal rapporto tra positivi e negativi).

La migliore in realtà non la sappiamo perché non abbiamo abbastanza informazione, quello che dobbiamo sapere è il rapporto tra positivi e negativi. Dobbiamo quindi sapere quale delle due isometrie è quella valida, se quella in alto allora sarebbe il classificatore in alto, se no invece è l’altra allora sono tutti e 3 i migliori.



Ci sono dei casi in cui il numero di errori che commetto sulle classi negative ha una diversa importanza del numero di errori che commetto sulle classi positivi (importanza che do ai falsi positivi e minore di quella che do ai falsi negativi, pensa all’esempio medico). Quindi a volte è utile dare diverso peso agli errori. Come fare? Posso ad esempio raddoppiare il numero di negativi e lasciare invariato quello di positivi, in quel caso daremo peso 2 ai falsi negativi, quindi il costo sarà c = cost(FN)/cost(FP). Ciò creerà diverse isometrie, che ci permette di ottenere il classificatore ottimale al costo che vogliamo imporre.

**ANNO 20-21**



Il Roc Plot ci serve quando vogliamo confrontare classificatori che hanno dataset diversi.

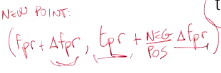
Come cambiano **accuratezza** e **average recall**?

* L’average recall, visto che l’asse x e l’asse y hanno lunghezza 1 allora la diagonale principale avrà pendenza 1
* L’accuratezza assume una pendenza **neg/pos**

L’accuratezza si può scrivere come (TP + TN)/TOT, ma visto che in questo caso, nel plot, non abbiamo TP e TN ma **tpr** e **fpr** allora la formula cambia e diventa



Prendendo un nuovo punto nel grafico abbiamo

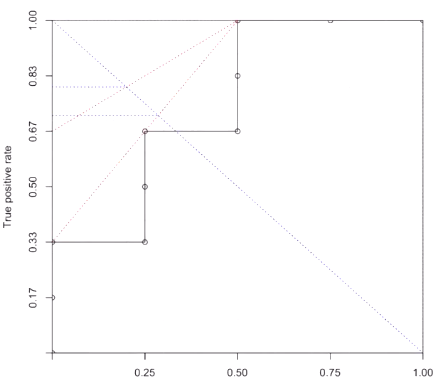


(Delta fpr perché non ha senso spostarsi di uno se gli assi hanno lunghezza 1)

E sappiamo che **tpr = 1 – fpr**

Il Roc plot server per mettere a confronto dei classificatori che sono stati appresi su data set con numero di esempi diversi.

A volte ci sono algoritmi di apprendimento che variano secondo qualche parametro, che potrebbe essere ad esempio un parametro di soglia. A volte serve scegliere il parametro migliore, per farlo si fa variare il parametro, si inseriscono nel roc plot i punti e si cerca di prendere una decisione



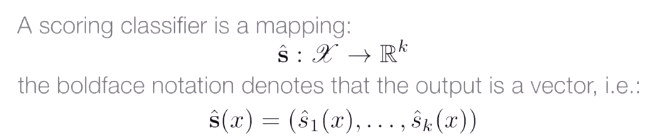
In questo caso qual è il migliore? Dipende dal contesto in cui si valutano i classificatori, perché a volte ci si può trovare in una situazione in cui il costo dovuto ad un errore positivo è maggiore del costo dovuto ad un errore negativo o viceversa. In un caso del genere, in cui entra in gioco il fattore costo, come retta che andrò a guardare non sarà più quella di pendenza 1, ma di pendenza **1/c**, dove c = cost(FN)/cost(FP).

Minuti 30-40

**FINE ANNO 20-21**

#### Scoring classifier

Al posto di dare in output una classe danno in output uno score

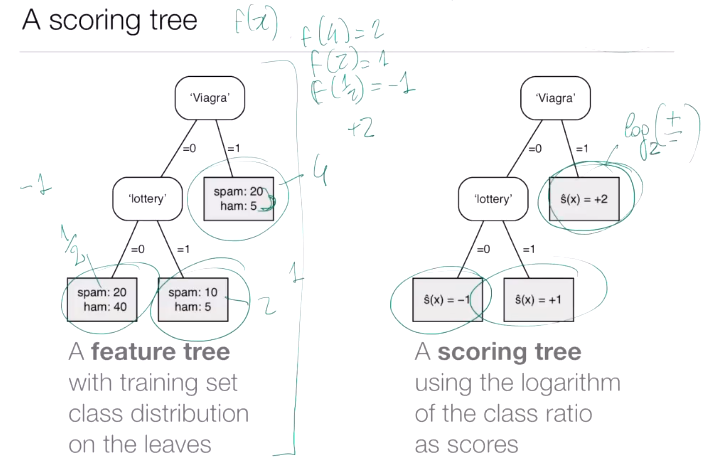


Si va da un set di esempi a un vettore di k elementi con valori reali. Perché è utile usare questi modelli? **Mi dice con che percentuale viene classificato l’oggetto con una classe C**.

Di solito si ha uno score per ogni classe.

Il problema di apprendimento non diventa un problema di regressione, gli esempi non sono etichettati con gli score, sono etichettati con le classi. Gli score sono una informazione che il classificatore mi da in più per darmi una misura della confidenza sulla previsione di una certa classe.

Vediamo l’esempio visto in precedenza facendolo diventare uno scoring classifier.

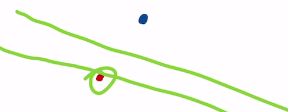


Come misura è stato usato il logaritmo f(x) = log(x).

Quando abbiamo lo score è importante introdurre due concetti

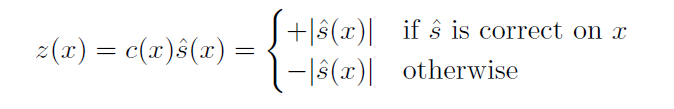
* **Margine**
* **Funzione di loss**

Il **margine** è una misura di quanto è confidente il classificatore nelle predizioni che fa, misura importante per migliorare le prestazioni del classificatore, l’obiettivo è massimizzare il margine (misura migliore dell’accuratezza, perché l’accuratezza non tiene conto dell’errore dei dati, un buon margine tiene conto anche di quello, perché andrebbe a mettere la funzione in punto migliore).



Nel caso binario c’è un altro concetto collegato con lo score, il **margine**. È un numero che mi dice quanto è buona la predizione. È un valore numerico che mi dà una quantificazione di quanto buona è la mia predizione. Combina lo score con la predizione corretta

* Margine positivo se la predizione è corretta
* Margine negativo se la predizione è sbagliata



C(x) è la classe associata all’esempio mentre s(x) è lo score della predizione. Assumiamo che c(x) appartiene a {-1, 1}

***z*** indica il margine, ed è calcolata moltiplicando la classe associata all’esempio e lo score dell’esempio. Viene usato come misura di bontà è molti algoritmi cercano di aumentare il margine.

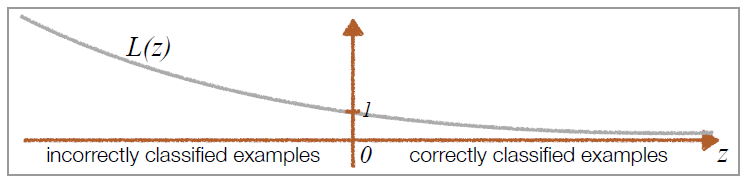
La **loss** in generale è una misura che ci dice quanto sta facendo male il classificatore, noi vorremo minimizzare la loss.

La loss function prende in input il margine del classificatore e restituisce un valore. Cresce al decrescere del margine e viceversa.

Sono importanti le loss function perché negli algoritmi sono il mezzo usato per guidare l’apprendimento. Si vuole minimizzare il valore della loss sul training set.

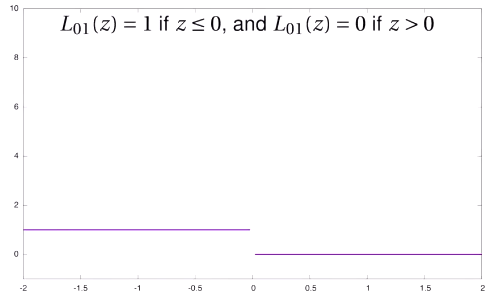
Il margine viene utilizzato spesso negli algoritmi di apprendimento per ottimizzare i modelli che stiamo valutando. Quello che viene ottimizzato è il concetto di **loss function**, ovvero una funzione che misura quanto sta perdendo il modello rispetto alle performance ideali. Gli algoritmi di apprendimento minimizzano una qualche misura di loss. Si vuole tendere a loss = 0. Assumiamo che quando il margine 0 c’è loss = 1. Se il margine è positivo il loss tende a 0, viceversa tende all’infinito.

Un modo per caratterizzare le loss function più usate è ricostruire la loss function come la funzione del margine, quindi L(z(x)). Parliamo sempre di classificazione binaria.

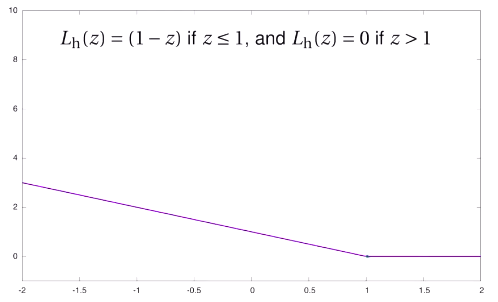


Le loss più comuni sono

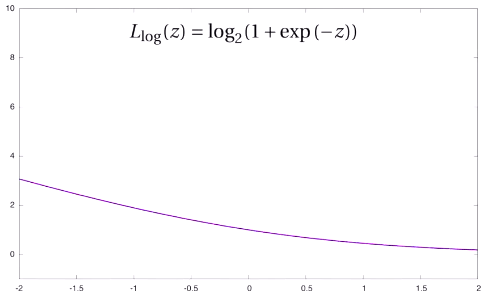
* **0-1 Loss**, corrisponde all’accuratezza, da un errore pari a uno da –infinito a 0 compreso, e da errore pari a 0 da n>0. Corrisponde all’accuratezza perché se io calcolo la loss su qualsiasi esempio la loss mi dice se l’ho sbagliato o meno, se poi faccio la media otterrò l’errore che è uguale a 1-accuratezza. Se voglio minimizzare questa loss e ho classificato completamente tutto il data set, io da lì non ho più modo di fare minimizzazioni.



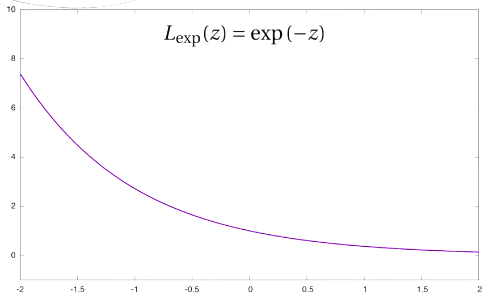
* **Hinge loss**, decresce fino al valore 1 e quando arrivo a uno la loss diventa 0. Penalizza i margini positivi, ma anche quelli piccoli negativi (molto usata nelle reti neurali e anche in altri modelli). Interessante perché fa usare una tecnica basata sulla discesa del gradiente.



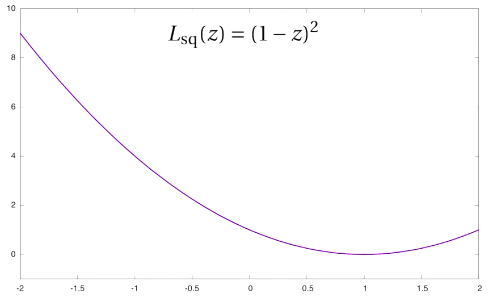
* **Logistic loss**, è un’approssimazione continua della hinge loss, questo perché la hinge loss contiene un punto di discontinuità sul punto 1.  
  PS. Esponenziale di meno z è la speculare dell’esp e aggiungendo il logaritmo prende la forma che c’è sotto



* **Exponential loss**, ha la caratteristica che cresce esponenzialmente per valori negativi del margine.



* **Loss quadratica**, ha il difetto che oltre l’uno ritorna a crescere, e quindi penalizza gli esempi con margine alto (permette di semplificare i calcoli)



I passaggi che ci sono dalla classificazione alla loss function sono

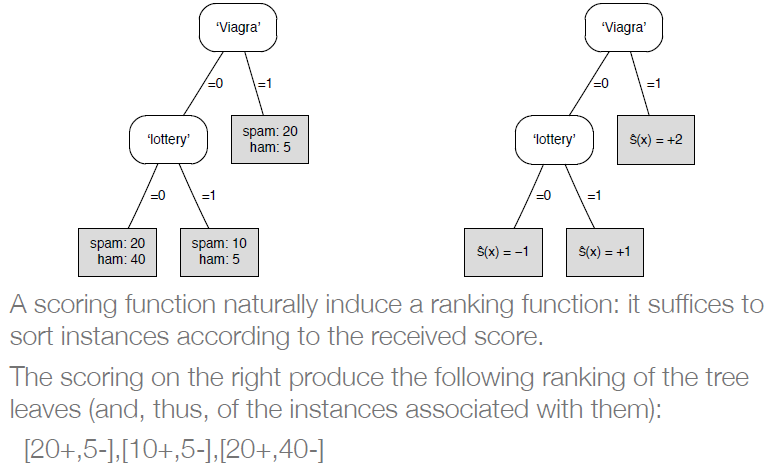
* Si ha l’esempio x
* X viene classificato dallo scoring classifier e si ottiene s(x)
* Lo score che si ottiene viene messo dentro il margine z(x)
* Z(x) viene valutato dalla funzione di loss, ad esempio L01(z(x)) che sarà in questo caso 1 o 0, e lo stesso fale per le altre funzioni di loss.

Questo viene calcolato su un singolo esempio, se poi voglio calcolare una misura dell’errore su tutto il training set in genere farò una media come nel seguente esempio



#### Ranking

Permettono di mettere in ordine le istanze del data set e poi decidere le classi



Qua sopra in basso vediamo il ranking fatto.

A volte è utile perché il task vuole proprio quello, altre volte viene utilizzato come strumento per confrontare più classificatori.

Vedi spiegazione rec.

Come si misura l’errore (misura che ci dice quanto è buono il ranking)? Utilizziamo il **ranking error rate**, la somma su tutti gli esempi positivi e negativi, se l’esempio positivo è stato messo nel ranking dopo un esempio negativo allora lo conto come errore, se il positivo è a pari merito di un positivo allora lo conto mezzo errore. Divido tutto per num Positivi\*num Negativi

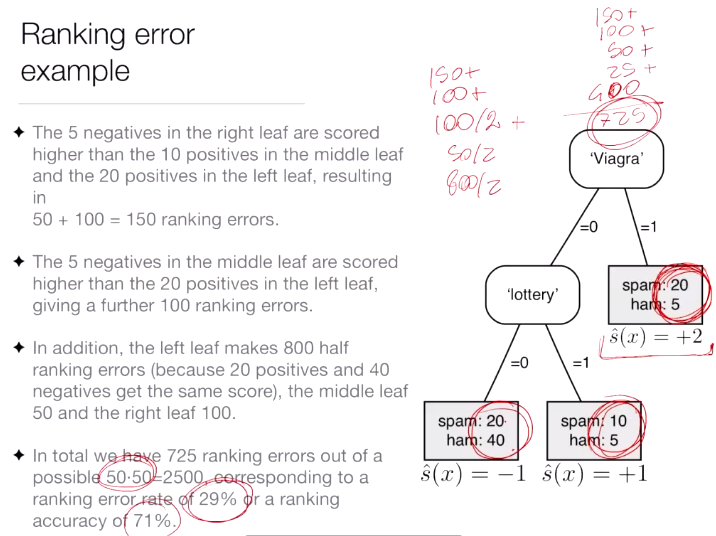


Questo oggetto sta calcolando l’area sotto la curva roc, calcolandola otteniamo lo stesso errore.

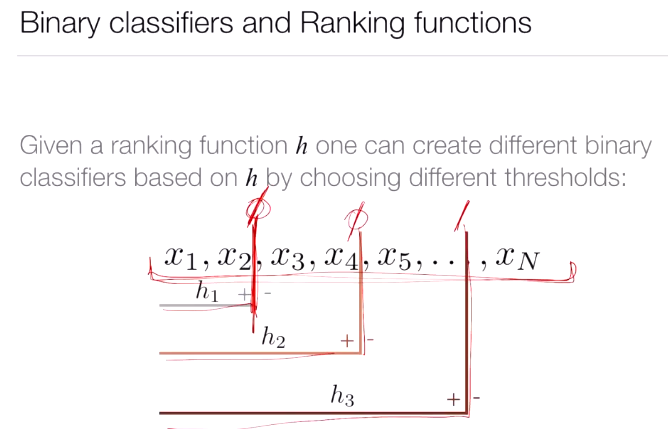
Immagine che contiene testo

Descrizione generata automaticamente

Farlo su un albero di decisione è un po’ più complicato.



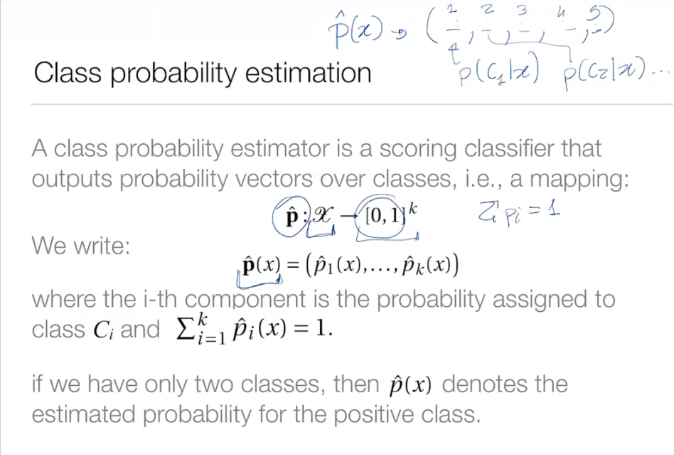
Il ranking a volte ci permette di confrontare classificatori.



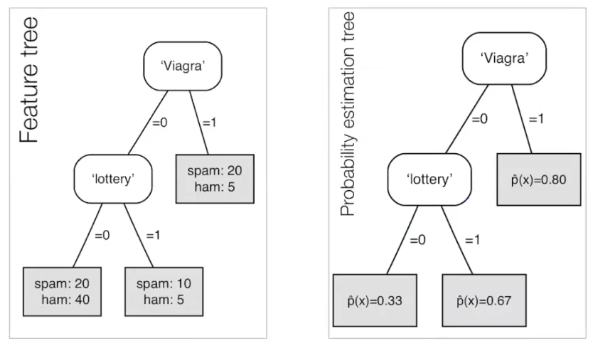
#### Probability estimation

Situazione molto vicina allo scoring, con la differenza che i valori di score che tiro fuori devono essere sotto forma di probabilità (devono essere tutte <=1 e la loro somma deve essere 1).

Il classificatore P prende in input una istanza e restituisce come output un vettore in cui ogni elemento del vettore è la probabilità che all’istanza venga assegnata una determinata classe.



Guardiamo la frequenza delle classi nei nodi foglia



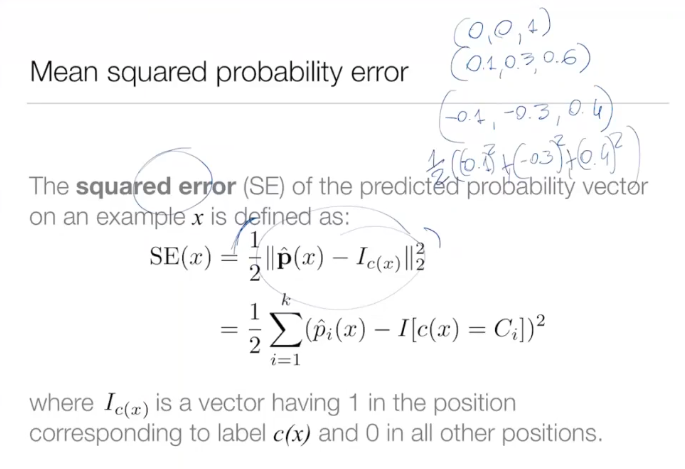
I valori delle foglie si calcolano facendo spam/spam + ham.

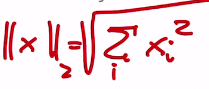
Es. 20/(20+5) = 0.8

0.8 sarà etichettato come +1, 0.33 come -1

Avendo il vettore di probabilità, come calcolo l’errore totale? Come misuriamo la bontà?

Iniziamo introducendo **squared error** (errore quadratico). Avendo tre classi e la classe corretta è 3 il vettore ottenuto corretto sarebbe (0, 0, 1). Ma se avessimo ad esempio (0.1, 0.3, 0.6) dobbiamo calcolare l’errore





Che è la distanza euclidea (cioè la lunghezza del vettore)

Esempio

Immagine che contiene tavolo

Descrizione generata automaticamente

Vedi rec es.3. A volte è preferibile che la probabilità per una determinata classe non sia troppo tendente a 1, perché in caso di errore ci penalizza molto (ci vuole il giusto bilanciamento tra l’essere confidente e lo spalmare le probabilità quando si è più incerti).

Ciò visto è l’errore su singolo esempio. Il **Mean squared error** è la media dello squared error su tutti gli elementi del data set.

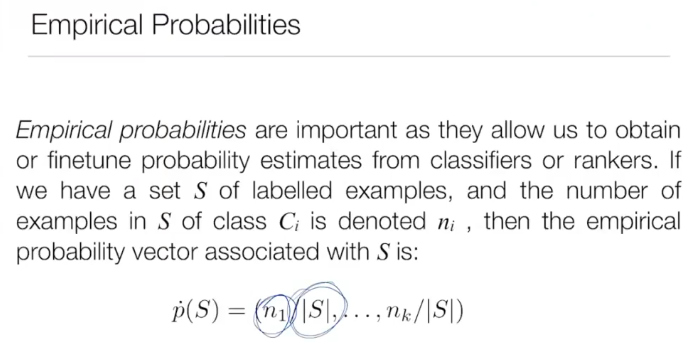
Immagine che contiene testo

Descrizione generata automaticamente

Fare es.

Noi finora abbiamo stimato la probabilità sul numero di positivi fratto il numero totale di oggetti, ciò si chiama **calcolo delle probabilità empiriche**. Se una delle probabilità diventa zero, tutto diventa zero (al posto di mettere un valore molto basso). Idea è fare uno smoothing dei valori apportando la **correzione di laplace**.

Ogni volta che c’è da stimare una probabilità dei dati c’è sempre il problema di mettere insieme il fatto che noi vorremo avere una probabilità il più vicina possibile alla distribuzione di probabilità che ha generato i dati e dall’altra parte abbiamo pochi dati per stimare queste probabilità. Il fatto di avere pochi dati per stimare è molto probabile che la probabilità sia stimata 0.



Siccome si potrebbe arrivare a moltiplicare tra loro le probabilità delle varie classi, avendo magari la classe n5 con 0 esempi, si avrebbe il prodotto totale zero. Per evitare ciò si tende a fare la Laplace correction, in modo da aggiungere un esempio per ogni classe (con k il numero delle classi) ed evitare l’azzeramento del prodotto.

È quasi sempre una buona idea fare un ammorbidimento di queste probabilità calcolate, e il modo si chiama **Laplace correction**



Esiste una tecnica più generale, chiamata **m-estimate**, la cui idea è la stessa ma, mentre prima assumevamo di aggiungere una distribuzione uniforme, ora assumiamo di prendere m nuovi esempi e che la probabilità di estrarre la classe pi sia data da pii (usiamo la stima delle probabilità a priori della classe). PIi è la probabilità della classe i e non deve essere uguale a 0.

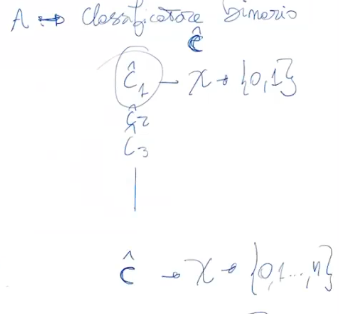


La Laplace correction è un caso speciale della m-estimate in cui m = k e pii = 1/k.

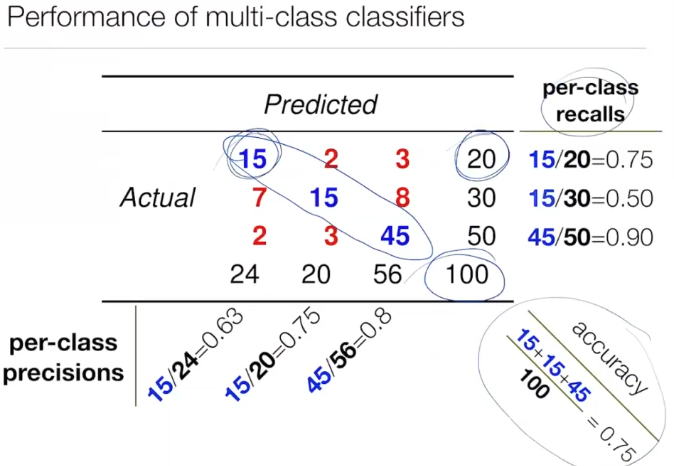
Va usato quando si hanno pochi dati, perché maggiori sono i dati meno la correzione è influente. Ciò si dice fare lo **smoothing** della distribuzione.

### Oltre alla classificazione binaria

Si tratta di una over view di quali sono le cose che possiamo fare nel caso non volessimo fare classificazione non binaria, ma partendo da algoritmi che ci forniscono solo un classificatore binario.



Nelle tavole di contingenza con n classi non cambia praticamente nulla, il numero di righe colonne cresce al crescere delle classi da 2x2 diventa nxn



Le misure calcolate su righe e colonne, sono accuratezza (sommando i valori sulle diagonali, che sono gli esempi predetti correttamente dividendo per il totale). Oltre a ciò vengono calcolate la recall per classe (per le righe), mentre la precisione per classe (per le colonne)

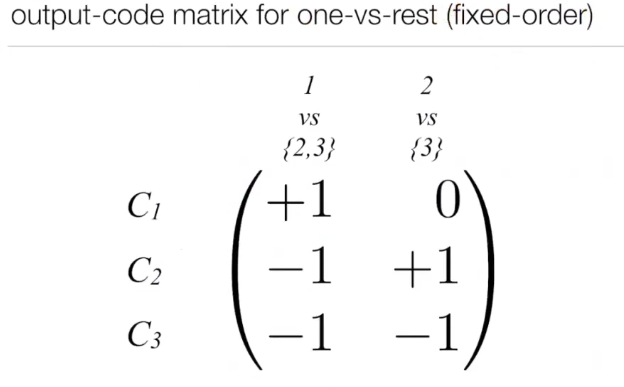
**Assumendo di avere classificatori binari, come possiamo utilizzarli per gestire problemi multi-classe?**

Ci sono sostanzialmente 4 schemi, per tutti questi schemi l’idea e di prendere il data set dato con etichette che vanno da 1 a n, noi costruiamo tanti data set binari. Verranno create le matrici per fare ciò, quello che cambia tra gli schemi e come verranno costruite

Lezione 4

Non vedremo un algoritmo che supporti nativamente la classificazione multiclasse, ma vedremo, dato un algoritmo che supporta la classificazione binaria, come effettuare la classificazione multiclasse. Il classificatore multiclasse nascerà dal composizione dei classificatori binari.

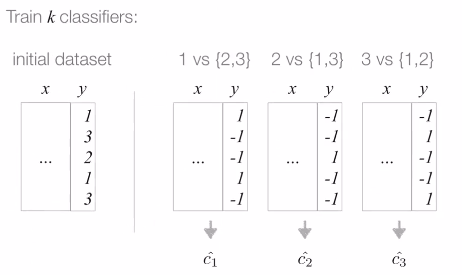
* Schema uno contro tutti
  + **Versione ordinata**

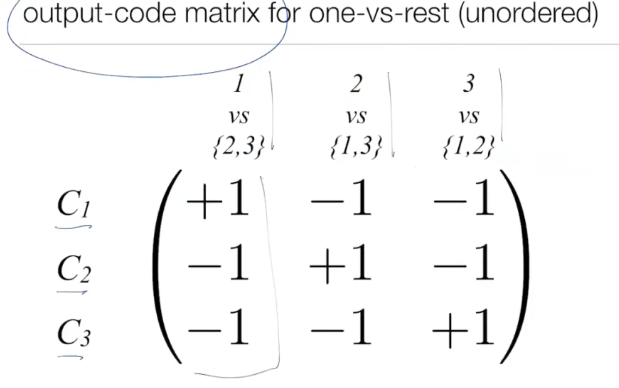


* + **Versione non ordinata**

Abbiamo un oggetto L che prende in input un dataset D e ci restituisce in output un classificatore C. il classificatore però è binario (quindi restituisce 0 e 1). Però noi abbiamo 3 etichette (1, 2, 3).

Costruiamo tanti problemi di classificazione binaria.

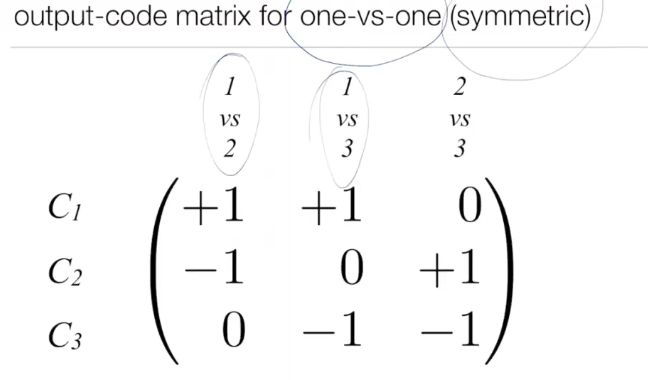




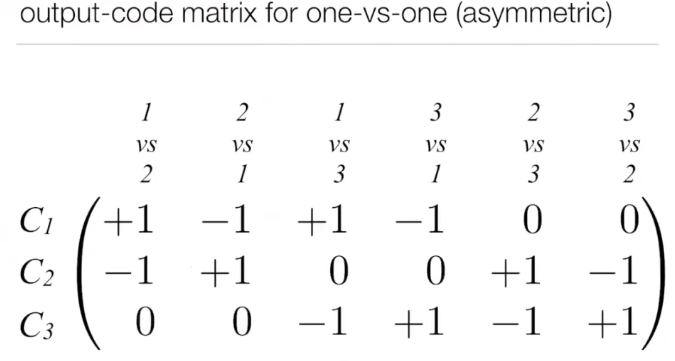
Per predire data una matrice ad esempio (-1 1 -1) vado a guardare le righe, in questo caso verrà restituito C2. In caso di (1 -1 1) si ha un **caso di ambiguità**, vedremo come risolverlo.

Quindi:

* + - Nella parte di apprendimento vado a costruire n classificatori binari
    - Nella parti di classificazione i dati vengono passati ai classificatori che mi restituiscono i risultati
* **Schema uno contro uno**
  + **Versione simmetrica**



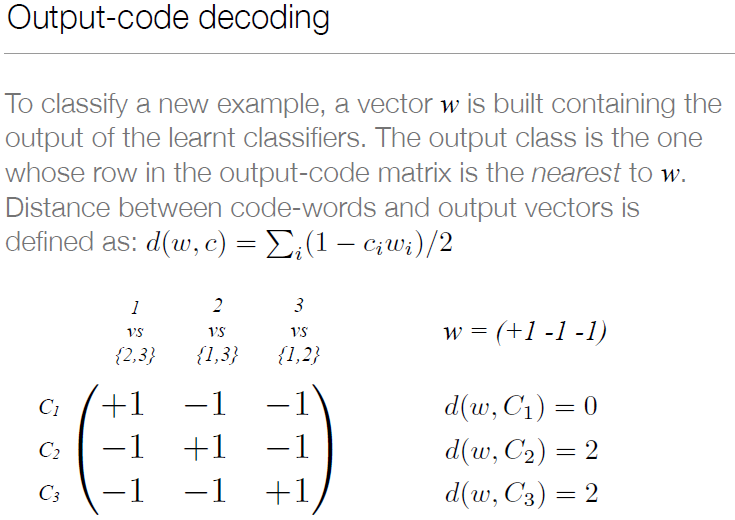
* + **Versione asimmetrica**



Viene usata quando ho un oggetto molto sensibile alle variazioni sul dataset e quindi mi ad esempio facendo 1 vs 2 e 2 vs 1 mi restituisce classificazioni simili ma non identiche, quindi entrambi sicuramente faranno errori sui casi limite (quindi quelli vicini alla funzione di separazione). Avere entrambi i classificatori può essere d’aiuto in caso di ambiguità, ad esempio

PS. Nei casi in cui ci sono gli zeri è perché è possibile astenersi.

Vediamo come trovare l’etichetta, quindi come fare il decoding del vettore di output che mi è stato restituito dal mio modello. Come viene trovata la riga migliore?



* + d(w, c) mi da la distanza tra il vettore di output e una delle righe della matrice. Andremo a prendere quello con distanza minore. w è l’output del classificatore appreso. Con d andiamo a decodificare il risultato. In situazione di conflitto (ad esempio, d(w, C1) = d(w, C2) = 1) non sappiamo scegliere, si può tirare a caso, fare uno score. Le tecniche migliori comunque sono
    - aggiungere nuovi classificatori (come viene fatto nel caso asimmetrico)
    - se il classificatore restituisce uno scoring classifier, prendiamo la grandezza maggiore (quella con maggiore confidenza)

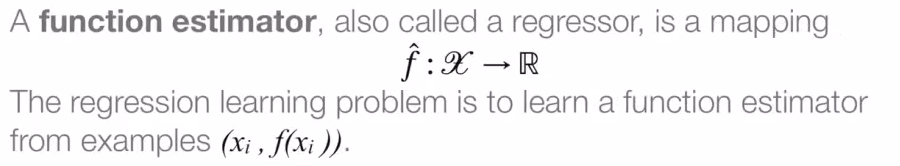
Quali sono gli svantaggi e i vantaggi di applicare questi schemi?

Un problema degli schemi uno vs tutti, è che nel costruire il dataset, passo da un dataset bilanciato (assumiamo) (ad esempio, che ha 100 etichette 1, 100 etichette 2, 100 etichette 3) ad uno sbilanciato (100 etichette 1, 200 etichette {2, 3}). **Più un dataset è sbilanciato più è difficile restituire un buon modello**.

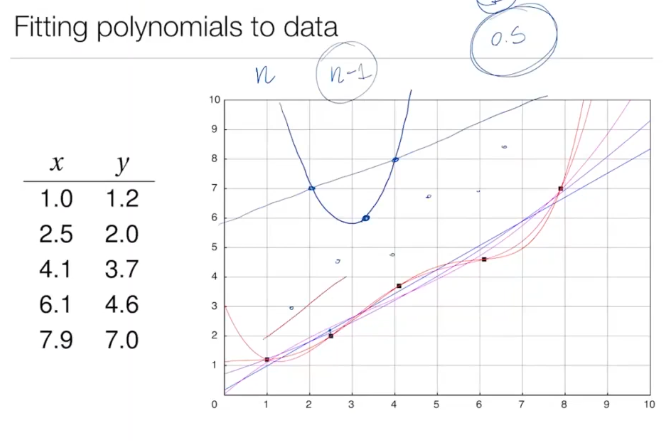
Nel caso 1 vs 1, non c’è quel problema, ma il dataset sarà più piccolo, perché non andiamo a considerare le altri classi (a parte le 2 prese in considerazioni), quindi se i dati sono pochi è difficile restituire un buon modello.

## Regressione

Si parla di regressione quando l’output della funzione acquisita dall’algoritmo di apprendimento non è più un insieme discreto e finito ma è l’insieme dei numeri reali. Prendo in input un valore x e restituisco un valore che va da –infinito a +infinito.



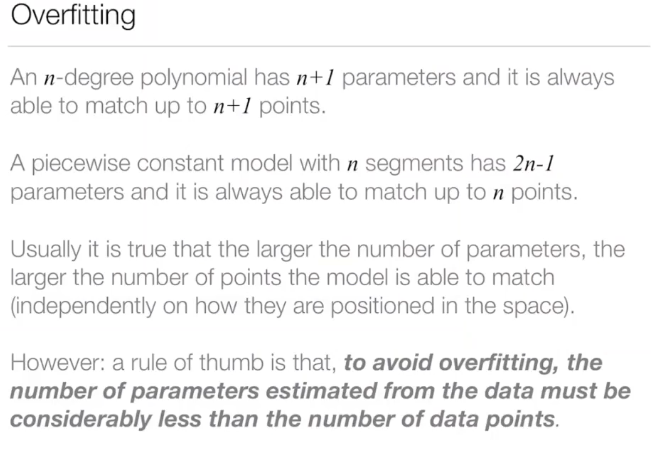
Non si potrà tenere traccia delle singole classi. È sempre vero che la previsione si distacchi dalla soluzione reale. È futile cercare di predire con il massimo della precisione possibile, ad esempio prevedrò 0.5, ma il risultato sarà un suo intorno (altrimenti l’oggetto farebbe troppo overfitting).



La retta non passa esattamente sopra i punti, se volessi farlo dovrei costruire una funzione di grado almeno 4 (che però farà eccessivamente overfitting).

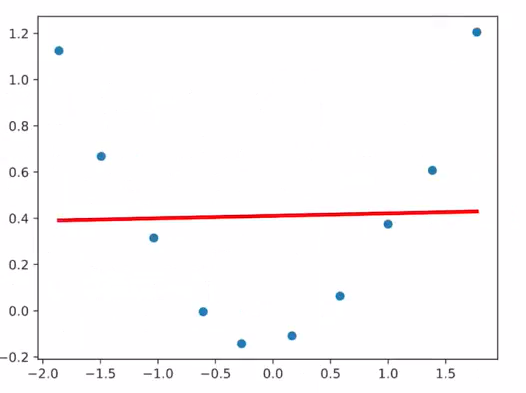


Anche con una funzione costante a tratti si può commettere overfitting

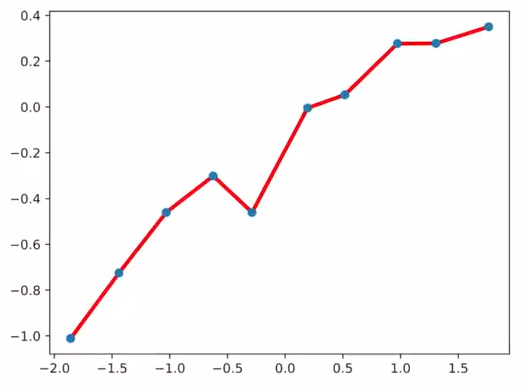


Piecewise costant (funzione costante a tratti)

Se il numero di parametri cresce è un problema, perché al crescere del numero di parametri cresce la complessità del modello e cresce la propensione del modello di fare overfitting. La regola da usare quando è che il numero di parametri necessari per rappresentare il modello dovrebbe essere minore in modo significativo del numero di punti che voglio stimare (ad es. se uso 5 punti per rappresentarne 100 allora va bene). Però ciò non basta, perché se sottostimo il numero di parametri necessari si crea l’underfitting, come di seguito



Con un numero di parametri alto invece si ottiene il seguente risultato (overfintting)



Si va in contro quindi al **dilemma di** **bias e variance**.

I due errori che si trovano sempre sono

* **errore di polarizzazione**
* **errore di varianza**

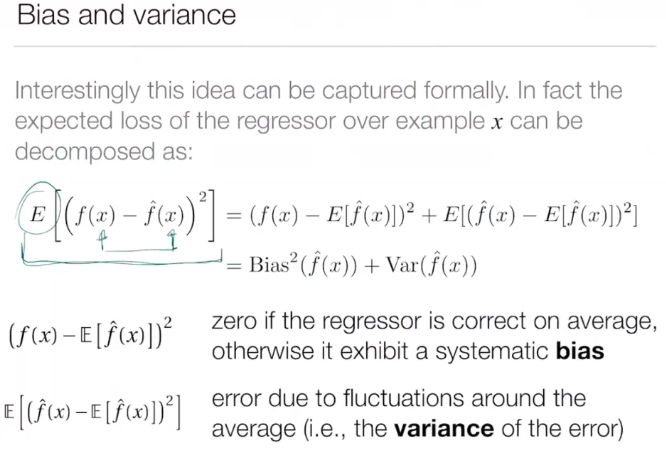
gli errori sono in conflitto (diminuendo uno aumento l’altro). Esiste un punto di minimo in cui entrambi gli errori sono bilanciati per essere i minori possibili.

Nel caso di underfitting sto commettendo un errore di polarizzazione (errore di bias), cioè l’errore è dovuto alla forma del classificatore che sto apprendendo che non riesce ad adattarsi bene ai dataset.

Nel caso di overfitting sto commettendo errore di varianza.

Se sottostimo la complessità del modello (ovvero il numero dei parametri che devo stimare) eviterò l’overfitting, ma non sarò in grado di adattarmi a sufficienza al modello (underfitting). Dall’altra parte se il numero di parametri cresce troppo avrò il problema opposto. Ho il problema di trovare il numero di parametri adatto e questo problema si chiama **dilemma bias and variance**. Si chiama così perché un modello con bassa complessità la funzione varierà facilmente senza cambiare troppo, però non passerò sui dati e quindi ci sarà un problema di bias. Con complessità alta invece elimino il problema del bias perché andrò a passare su tutti i dati con la funzione, ma quando aggiungo un nuovo dato cambierò in modo sostantivo la funzione

Vediamo come risolvere il problema



Qua sopra abbiamo l’errore quadratico medio ottenuto da f cappello.

E[…] equivale a valore medio

Immagine che contiene testo

Descrizione generata automaticamente

E[f(x)2] = f(x)2 perché la media di una costante è sempre uguale alla costante.

F(x) quindi è un numero noto

PS. La varianza può essere scritta in diversi modi (in base a come giochiamo con la formula)

# Task predittivi vs task descrittivi

Nei task predittivi data una nuova istanza di un problema voglio dire qualcosa anche se non l’ho mai vista. Nei descrittivi invece voglio trovare un modello per un dato data set in modo da imparare qualcosa da esso.

Gli **unsupervised learning** sono task di apprendimento automatico in cui non c’è una supervisione da parte dell’utente delle etichette associate ai dati sono associati con i task descrittivi. Spesso l’obiettivo è trovare regolarità nei dati usando solo i dati.

## Clustering

L’idea è investigare se nei dati ci sono dei gruppi che sorgono naturalmente.

Il clustering può essere usato in modo

* **Predittivo**, sto cercando di apprendere una funzione che va dall’interno spazio degli esempi ad un certo numero di classi che sono i cluster che sto cercando e posso dare un’etichetta di cluster anche a istanze che non fanno parte del mio data set
* **Descrittivo**, il dominio del modello che sto cercando di creare è solo un data set

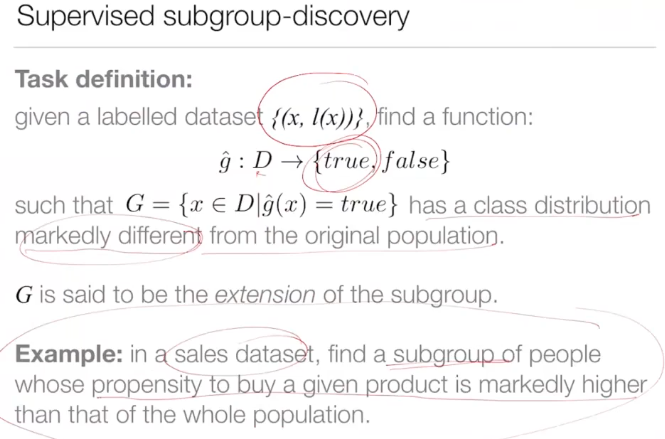
## Subgroup discovery supervisionato

L’idea è, dato un input un dataset etichettato, vogliamo ottenere una funzione che dato un dataset mi restituisca true o false, ovvero, vogliamo che ci dica che l’elemento del dataset fa parte di un sottogruppo o meno. Si vuole trovare una funzione g cappello tale per cui la distribuzione la classe degli esempi del gruppo che ho individuato è marcatamente differente da quella dell’intera popolazione.

Ad esempio, dato un dataset che contiene tutte le vendite di un determinato prodotto, voglio trovare il sottoinsieme delle persone che sono più propense ad acquistare uno specifico prodotto.

La maggior parte delle metriche per produrre questi sottogruppi preferiscono sottogruppi grandi e simmetrici.

Ci viene dato un data set e dobbiamo trovare una funzione che etichetti gli elementi del data set in true e false, in cui distinguiamo due gruppi molto diversi

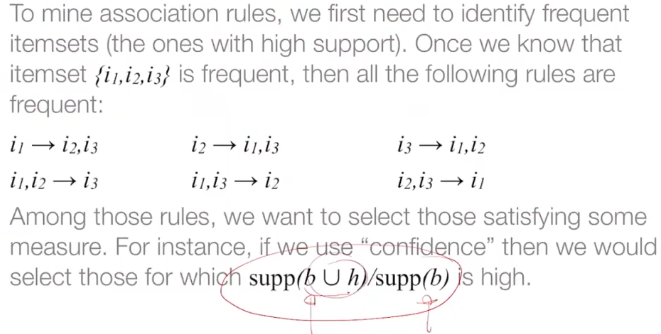


Preferiscono sottogruppi grandi e spesso circa la metà del data set

## Association rules

Creo i sottoinsiemi frequenti e poi si fanno delle associazioni





Si usano per esempio per fare market analysis.

Supponiamo che il nostro dataset è formato da scontrini. Ogni scontrino ha un elenco di prodotti acquistati. Il task è trovare delle regole (**di associazione**) che mi dicano che quando uno compra il pane e il latte allora probabilmente è interessato anche ai pannolini (ad esempio)

Sono regole del tipo { b -> h }

Se è vero che si acquistano i prodotti nel body b allora si acquistano anche i prodotti nella head h.

Per costruire ciò devo assicurarmi che queste regole siano generate da situazioni che accadono spesso (**itemset b U h**)

I possibili sottoinsiemi sono tutte le possibili combinazioni tra i prodotti (2n). la cosa difficile è calcolare la frequenza di 2^n elementi, difficile da fare. Le association rules ci forniscono metodi che semplificano ciò.

Vengono presi gli elementi più frequenti e poi vengono guardate tutte le casistiche di acquisto.