

APPUNTI

1 Intro (esposito)

Cos'è il machine learning

È lo studio sistematico di algoritmi e sistemi che migliorano la loro conoscenza e performance grazie all'esperienza. L'algoritmo di apprendimento crea un modello, il modello a sua volta viene usato per risolvere il task.

Il machine learning si occupa di utilizzare le giuste features per creare il giusto modello per risolvere il giusto task.

Componenti del machine learning

1. **Tasks:** ovvero l'obiettivo → classificazione, regressione, stima delle probabilità, clustering...
2. **Models:** come modellare il problema
 - geometrico(lineare, cubico)
 - logico(alberi di decisione)
 - probabilistico(naive bayes)
3. **Features:** come descrivo il problema
 - numerico(affianco al modello lineare)
 - categorico(un insieme di valori discreto e finito, da affiancare ad un modello logico)
 - feature construction(posso costruire features da quelle che già posseggo)
 - feature selection(posso selezionare un sottoinsieme da un insieme di features)

Bias

Ragionamento dell'algoritmo

No Free Lunch Theorem

The “No Free Lunch” theorem states that there is no one model that works best for every problem. The assumptions of a great model for one problem may not hold for another problem, so it is common in machine learning to try multiple models and find one that works best for a particular problem.

Tasks

I task vengono risolti dal modello. Quale modello scegliere lo fa l'algoritmo di apprendimento. I tasks sono predittivi o descrittivi.

Predittivi

Voglio predire un certo valore. Imparo prima da delle istanze, poi quando me ne viene data una nuova cerco di predire delle informazioni su quella. I task predittivi più comuni sono quelli di:

1. **classificazione**: dove cerco di predire un valore da un insieme discreto e finito.
2. **regressione**: la variabile che cerco di predire è numerica
3. **clustering**: cerco di trovare dei raggruppamenti e quando vado a predire nuove istanze vado ad associare la variabile in questione ad uno di questi raggruppamenti

Overfitting

È uno dei problemi dei task predittivi. Quando creo un modello troppo preciso per i dati di learning, ma poi con i dati di test non riesco a dedurre correttamente l'informazione.

Descrittivi

Voglio descrivere un certo dato. Quindi posso fare clustering e descrivere tutti i gruppi (senza fare predizioni). Voglio imparare qualcosa sui dati che ho in questo momento, non i dati futuri come nel caso dei task predittivi.

Esempio: predire preferenze sui film

Models

I modelli del machine learning si distinguono per il loro approccio.

1. **Geometrici**: modelli in cui cerco di modellare il problema utilizzando della geometria (ad esempio un'equazione di primo grado che mi genera una retta sul piano). In questi modelli ho dei pesi, weight, che cambia per raffinare il modello.
2. **Probabilistici**: cerco di trovare una forma di probabilità sulla base dei dati che ho. (fanno uso di distribuzioni probabilistiche)
 - a. **Naive Bias**: Assunzione di indipendenza. Considero le features come indipendenti l'una dalle altre. Perché se tengo conto delle dipendenze, computazionalmente diventa oneroso fare i calcoli sulle distribuzioni di probabilità
3. **Logici**: cerco di modellare il problema utilizzando proposizioni logiche (ad esempio una serie di if in cascata, ovvero mutualmente esclusive → alberi di decisione)

Features

Gli elementi per descrivere un problema. Spesso è utile manipolare le features per una semplificazione. Le features sono le variabili e i loro valori. Le features hanno due usi, sia come splitting feature e sia come regression (oppure decision ecc) variabile

2 Tasks (esposito)

Classificazione

Un classificatore è un mapping che data un'istanza di un esempio X lo associa alla classe di appartenenza C . Imparare un classificatore vuol dire costruire un mapping il più possibile vicino alla realtà.

Classificazione binaria

Un tipo di classificatore che si occupa di distinguere le istanze tra sole due classi

Accuracy: totale predetti correttamente / totali

Error rate: $1 - \text{accuracy}$

Tabella di contingenza

		Predicted \oplus	Predicted \ominus	
		TP	FN	Pos
Actual \oplus	FP	TN	Neg	
	40	60		100

Misure di performance

Performance measures

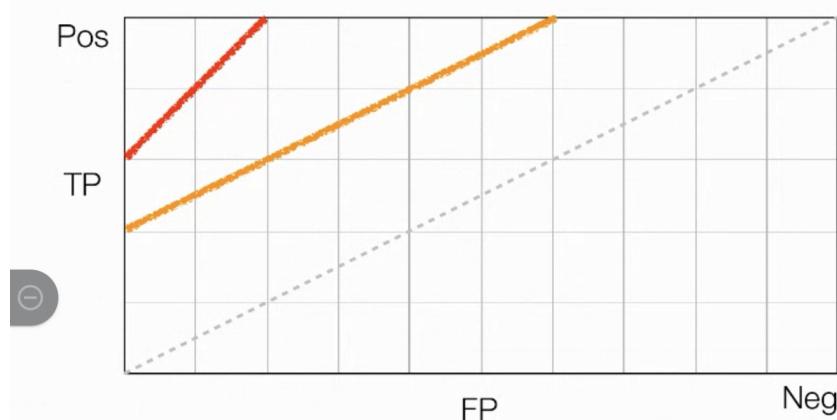
Measure	Definition	Equal to	Estimates
number of positives	$Pos = \sum_{x \in Te} I[c(x) = \oplus]$		
number of negatives	$Neg = \sum_{x \in Te} I[c(x) = \ominus]$	$ Te - Pos$	
number of true positives	$TP = \sum_{x \in Te} I[\hat{c}(x) = c(x) = \oplus]$		
number of true negatives	$TN = \sum_{x \in Te} I[\hat{c}(x) = c(x) = \ominus]$		
number of false positives	$FP = \sum_{x \in Te} I[\hat{c}(x) = \oplus, c(x) = \ominus]$	$Neg - TN$	
number of false negatives	$FN = \sum_{x \in Te} I[\hat{c}(x) = \ominus, c(x) = \oplus]$	$Pos - TP$	
proportion of positives	$pos = \frac{1}{ Te } \sum_{x \in Te} I[c(x) = \oplus]$	$Pos/ Te $	$P(c(x) = \oplus)$
proportion of negatives	$neg = \frac{1}{ Te } \sum_{x \in Te} I[c(x) = \ominus]$	$1 - pos$	$P(c(x) = \ominus)$
class ratio	$clr = pos/neg$	Pos/Neg	
(*) accuracy	$acc = \frac{1}{ Te } \sum_{x \in Te} I[\hat{c}(x) = c(x)]$		$P(\hat{c}(x) = c(x))$
(*) error rate	$err = \frac{1}{ Te } \sum_{x \in Te} I[\hat{c}(x) \neq c(x)]$	$1 - acc$	$P(\hat{c}(x) \neq c(x))$

true positive rate, $tpr = \frac{\sum_{x \in Te} I[\hat{c}(x) = c(x) = \oplus]}{\sum_{x \in Te} I[c(x) = \oplus]}$	TP/Pos	$P(\hat{c}(x) = \oplus c(x) = \oplus)$
sensitivity, recall		

true negative rate, specificity	$tpr = \frac{\sum_{x \in T_e} I[\hat{c}(x) = \ominus, c(x) = \ominus]}{\sum_{x \in T_e} I[c(x) = \ominus]}$	TN/Neg	$P(\hat{c}(x) = \ominus c(x) = \ominus)$
false positive rate, false alarm rate	$fpr = \frac{\sum_{x \in T_e} I[\hat{c}(x) = \oplus, c(x) = \ominus]}{\sum_{x \in T_e} I[c(x) = \ominus]}$	$FP/Neg = 1 - tpr$	$P(\hat{c}(x) = \oplus c(x) = \ominus)$
false negative rate	$fnr = \frac{\sum_{x \in T_e} I[\hat{c}(x) = \ominus, c(x) = \oplus]}{\sum_{x \in T_e} I[c(x) = \oplus]}$	$FN/Pos = 1 - tpr$	$P(\hat{c}(x) = \ominus c(x) = \oplus)$
precision, confidence	$prec = \frac{\sum_{x \in T_e} I[\hat{c}(x) = \oplus, c(x) = \oplus]}{\sum_{x \in T_e} I[\hat{c}(x) = \oplus]}$	$TP/(TP + FP)$	$P(c(x) = \oplus \hat{c}(x) = \oplus)$

Coverage Plot

Serve per rappresentare l'accuratezza di più classificatori come punti sul piano. Sulla Y i TP, sulla X i FP



Classifiers with the same **accuracy** are on lines with slope 1

Classifiers with the same **recall** are on lines parallel the main diagonal

Classificatori con la stessa accuracy si trovano nella retta con pendenza a 1

Classificatori con la stessa avg recall sono paralleli alla diagonale principale.

Se il rapporto fra negativi e positivi e' diverso da 1, la diagonale principale non avrà pendenza a 1.

Roc Plot

E' un coverage plot ma con X e Y normalizzati ad 1 e si usa la recall sulle Y e 1-recall sulle X (di solito, poi uno sulla fine può mettere il cazzo che gli pare). La diagonale principale ha sempre pendenza a 1. Un classificatore non e' più un punto nel piano ma una serie di segmenti. Questi segmenti sono stati creati dall'unione di punti dove ogni punto rappresenta lo stesso classificatore ma con threshold differenti per classificare. Per comparare più classificatori ci saranno quindi più "curve" nel plot e quella con AUC (area under the curve) maggiore indica il classificatore maggiore.

Classificatori con la stessa avg recall si trovano nella retta con pendenza a 1

Classificatori con la stessa accuracy sono paralleli alla diagonale principale.

Classificazione a score

Un classificatore a score è un mapping che data un'istanza di un esempio X genera uno score per ogni possibile classe di appartenenza C .

Margin

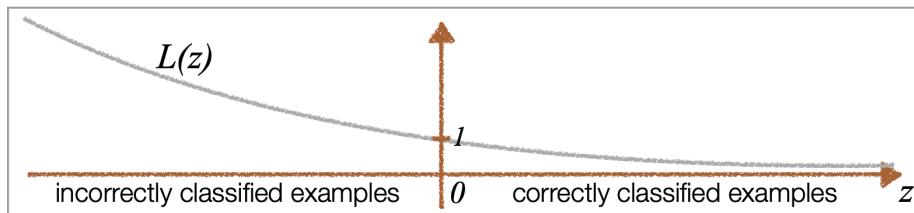
E chiamato margine $z(x)$ il risultato della classificazione a score. Un margine positivo indica una predizione avvenuta con confidenza, un margine negativo indica un errore di classificazione. Più è alto il margine meglio è, più è basso il margine peggio è.

$$z(x) = c(x)\hat{s}(x) = \begin{cases} +|\hat{s}(x)| & \text{if } \hat{s} \text{ is correct on } x \\ -|\hat{s}(x)| & \text{otherwise} \end{cases}$$

Loss Function

Una funzione che fa il mapping tra il margine ed un numero reale. Più è alto il margine allora più è basso l'errore. La funzione di loss è importante perché ci permette di calcolare l'errore di classificazione. Funzioni di loss diverse ritornano errori diversi ed è quindi importante scegliere la loss corretta.

- 0-1 Loss
- Logistic Loss
- Hinge Loss
- Exponential Loss
- Squared Loss



Ranking

Un classificatore a score induce implicitamente una funzione di ranking per effettuare valutazioni sullo score tree. E sufficiente ordinare le istanze sulla base del ranking.

[20+,5-],[10+,5-],[20+,40-]

Con questo ranking è possibile calcolare il ranking error rate. Più è alto l'error rate, più fa schifo il tuo modello

The **ranking error rate** is defined as:

$$\text{rank-err} = \frac{\sum_{x \in T^{+}, x' \in T^{-}} I[\hat{s}(x) < \hat{s}(x')] + \frac{1}{2} I[\hat{s}(x) = \hat{s}(x')]}{\text{Pos} \cdot \text{Neg}}$$

1 point of penalty due to a ranking error: a positive example is ranked below a negative example

1/2 point of penalty for tying examples having different classes

Classificatore probabilistico

E' un tipo di classificatore a score che al posto di mappare ad un'istanza uno score per ogni classe, genera un vettore di probabilità di appartenenza per ogni classe dove la somma delle probabilità fa 1.

Errore quadratico e errore quadratico medio

The **squared error** (SE) of the predicted probability vector on an example x is defined as:

$$\begin{aligned} \text{SE}(x) &= \frac{1}{2} \|\hat{\mathbf{p}}(x) - I_{c(x)}\|_2^2 \\ &= \frac{1}{2} \sum_{i=1}^k (\hat{p}_i(x) - I[c(x) = C_i])^2 \end{aligned}$$

where $I_{c(x)}$ is a vector having 1 in the position corresponding to label $c(x)$ and 0 in all other positions.

Mentre l'errore quadratico medio calcola la media tra tutte le istanze del dataset

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

Probabilità empirica

E' un vettore che conta la frequenza di appartenenza delle classi C in un insieme di istanze. Se in una foglia ho 5 classi + e 3 classi -, saranno [5 / 8, 3 / 8]

$$\dot{p}(S) = (n_1/|S|, \dots, n_k/|S|)$$

Laplace correction

E' una buona idea fare un po di smoothing. Lo smoothing serve per risolvere il problema della 0-probability. Questo serve perché un evento che ha probabilità 0 di accadere non ci da alcuna informazione, e' inutile. Invece facendo smoothing otteniamo valori > 0 che sono di conseguenza utili.

La laplace correction e' la modalità più usata per fare smoothing.

$$\dot{p}_i(S) = \frac{n_i + 1}{|S| + k}$$

k=numero di classi

M-Estimate (laplace generalizzata). pigreco i sarebbe la probability della classe i. La laplace e' un caso particolare dove m=k e pigreco i = 1/k

$$\dot{p}_i(S) = \frac{n_i + m \cdot \pi_i}{|S| + m}$$

Classificazione multiclasse

Come estendere classificatori binari a classificatori multiclasse

1. one vs rest
2. one vs one

1. one vs rest (unordered)

Vengono allenati k classificatori. In ogni classificatore, la k-esima classe va contro le restanti (1 vs 2-3; 2 vs 1-3; 3 vs 1-2)

Il downside è che se i classificatori sono discordi succede un bordello. Perché runno tutti i classificatori insieme senza ordinamento. Se il primo mi ritorna +1 ed il secondo o il terzo anche +1 non so quale sia corretto.

1a. one vs rest (fixed order)

Faccio 1 vs 2-3 e 2 vs 3. La differenza con one vs rest (unordered) è che mi fermo al primo classificatore se mi dice +1. Se mi ritorna -1 devo continuare.

2. one vs one

classificatori messi contro uno alla volta

Vengono prima creati vettori (composti da 1, -1 e 0) di risultati dai vari classificatori messi contro uno ad uno e poi viene creata una matrice di 1 e -1. Quando poi si va a classificare un nuovo esempio, questo ritorna un vettore di 1 e -1, si va poi a cercare la riga della matrice che sottratta al vettore precedente ritorna come risultato quello più vicino allo 0.

$$\begin{array}{ccc} 1 & 2 & 3 \\ \text{vs} & \text{vs} & \text{vs} \\ \{2,3\} & \{1,3\} & \{1,2\} \end{array} \quad w = (+1 \ -1 \ -1)$$
$$C_1 \begin{pmatrix} +1 & -1 & -1 \\ -1 & +1 & -1 \\ -1 & -1 & +1 \end{pmatrix} \quad d(w, C_1) = 0$$
$$C_2 \begin{pmatrix} +1 & -1 & -1 \\ -1 & +1 & -1 \\ -1 & -1 & +1 \end{pmatrix} \quad d(w, C_2) = 2$$
$$C_3 \begin{pmatrix} +1 & -1 & -1 \\ -1 & +1 & -1 \\ -1 & -1 & +1 \end{pmatrix} \quad d(w, C_3) = 2$$

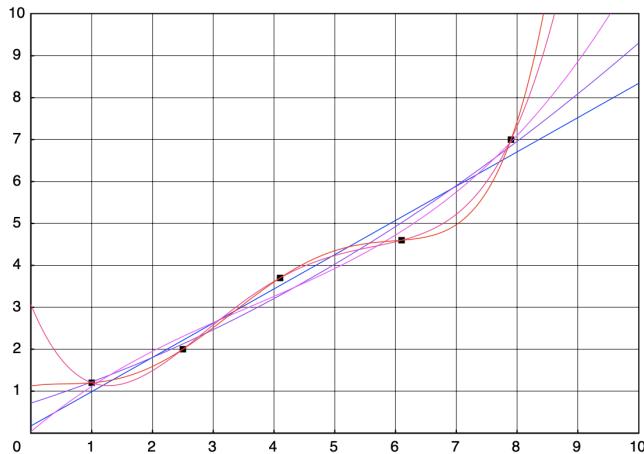
Il problema di questi approcci è che il classificatore vede il dataset molto sbilanciato anche se in verità non lo è. Questo perché lui vede solo valori +1 e -1 ed il valore -1 include più classi.

Regressione

E' una funzione di estimazione che mappa un'istanza con un valore Reale. A differenza di un classificatore qui abbiamo infiniti target.

Quindi c'è il rischio di fare overfitting se si cerca di massimizzare la precisione durante il training. Altrimenti se il modello è troppo semplicistico è possibile infatti che i dati siano rumorosi e l'estimatore è solo in grado di capire il trend generale dei dati.

Overfitting



per evitare overfitting

Il grado del polinomio deve essere minore del numero di esempi del training set

Bias–Variance dilemma

Un modello poco complesso soffre meno della variabilità dei dati di training (low variance), ma introduce il problema del bias (high bias), ovvero e' costretto a generalizzare con qualche errore.

Un modello altamente complesso elimina questo bias (low bias), ovvero riesce a essere preciso su tutti i dati visti, ma introduce l'errore di varianza (high variance), ovvero su dati nuovi che sono diversi da quelli di train non riesce a coprirli con precisione.

Nell'immagine sopra, la linea rossa e' un modello complesso, mentre il blu un modello non complesso.

Task Descrittivi

Nei task descrittivi non abbiamo dati di training. Lo scopo dell'algoritmo e' quello di descrivere i dati del dataset. I dati non sono etichettati.

Clustering

Ha lo scopo di trovare dei gruppi omogenei all'interno del dataset

- **predictive** opera anche su esempi fuori dal dataset
- **descriptive** opera solo su dataset (crea un mapping)

Subgroup-Discovery

Dato un dataset bisogna allenare una funzione g che riesca a trovare un sottogruppo con una distribuzione molto differente rispetto al dataset originale.

3 Concept Learning

L'obiettivo dell'apprendimento e quello di trovare una funzione che generalizzi partendo da esempi specifici. Delle ipotesi che approssimano correttamente la funzione target su esempi di training, approssimano anche su esempi non ancora visti.

L'obiettivo e' trovare un ipotesi che sia consistente con tutti i dati di training. Le ipotesi sono congiunzioni di termini, dove questi termini possono essere valori di features o simboli dont-care/no value allowed

Si parte da un'ipotesi generale. Per ogni esempio visto si vanno a svolgere delle azioni:

- se l'esempio è positivo e l'ipotesi è soddisfatta, allora e' ok, si passa al prossimo
- se l'esempio è positivo e l'ipotesi non e' soddisfatta, allora si droppa una feature facendo quindi generalizzazione dell'ipotesi
- se l'esempio è negativo e l'ipotesi è soddisfatta, allora bisogna rendere più specifica l'ipotesi su una feature in modo che quell esempio non venga più soddisfatto dall'ipotesi
- se l'esempio è negativo e l'ipotesi non e' soddisfatta, allora e' ok, si passa al prossimo

Find-S Algorithm

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
 - For each attribute constraint a_i in h
 - If the constraint a_i in h is satisfied by x
 - then do nothing
 - else replace a_i in h by the next more general constraint that is satisfied by x
3. Output hypothesis h

Vantaggi

1. Trova l'ipotesi più specifica

Svantaggi

1. Ignora gli esempi negativi, quindi non verifica la consistenza
2. Non controlla se ci sono ipotesi massimali multiple

Version Space

E' l'insieme delle ipotesi valide, ovvero quel sottoinsieme di ipotesi H che e' consistente con tutti i dati del dataset. Dato che possono comunque essere tante queste ipotesi, bisogna trovare un algoritmo che eviti l'enumerazione di ognuna. Il version space e' parzialmente ordinato.

List-Then Eliminate

1. $\text{VersionSpace} \leftarrow$ a list containing every hypothesis in H
2. For each training example $\langle x, c(x) \rangle$:
remove from VersionSpace any hypothesis that is inconsistent with the training example $h(x) \neq c(x)$
3. Output the list of hypotheses in VersionSpace

Candidate Elimination

$G \leftarrow$ maximally general hypotheses in H

$S \leftarrow$ maximally specific hypotheses in H

For each training example $d = \langle x, c(x) \rangle$

If d is a positive example:

Remove from G any hypothesis that is inconsistent with d

For each hypothesis s in S that is not consistent with d

- remove s from S .
It will make S a summary of the positive examples
- Add to S all minimal generalizations h of s such that
 - h consistent with d
 - Some member of G is more general than h
- Remove from S any hypothesis that is more general than another hypothesis in S .
It will keep the set S minimal, with the most specific members

If d is a negative example:

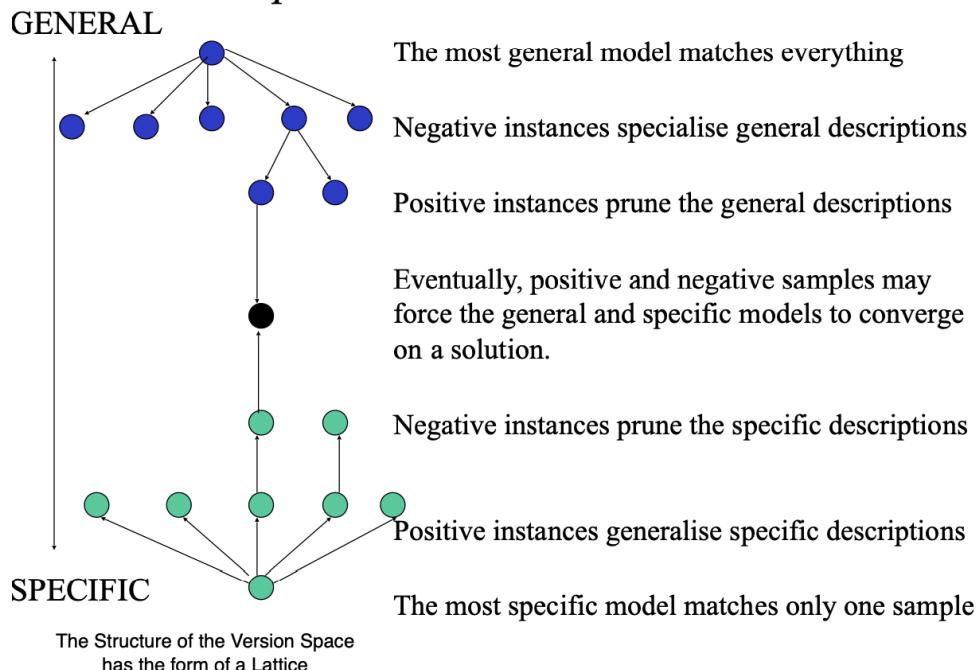
It will make G a summary of the members consistent with the negative examples

Remove from S any hypothesis that is inconsistent with d

For each hypothesis g in G that is not consistent with d

- remove g from G .
- Add to G all minimal specialisations h of g such that
 - h consistent with d
 - Some member of S is more specific than h
- Remove from G any hypothesis that is less general than another hypothesis in G .
It will keep the set G minimal, with the most general members

Version space



NB: Se d e' un esempio negativo, si svolgono le stesse cose ma invertendo S con G e viceversa.

Dopo aver eseguito questo algoritmo, gli esempi positivi e negativi forzano le ipotesi specifiche e generali a convergere in una soluzione.

Se non si hanno esempi a sufficienza può capitare che la soluzione di candidate elimination non sia univoca, ovvero rimangano multiple ipotesi nel Version Space. Per risolvere il problema possiamo:

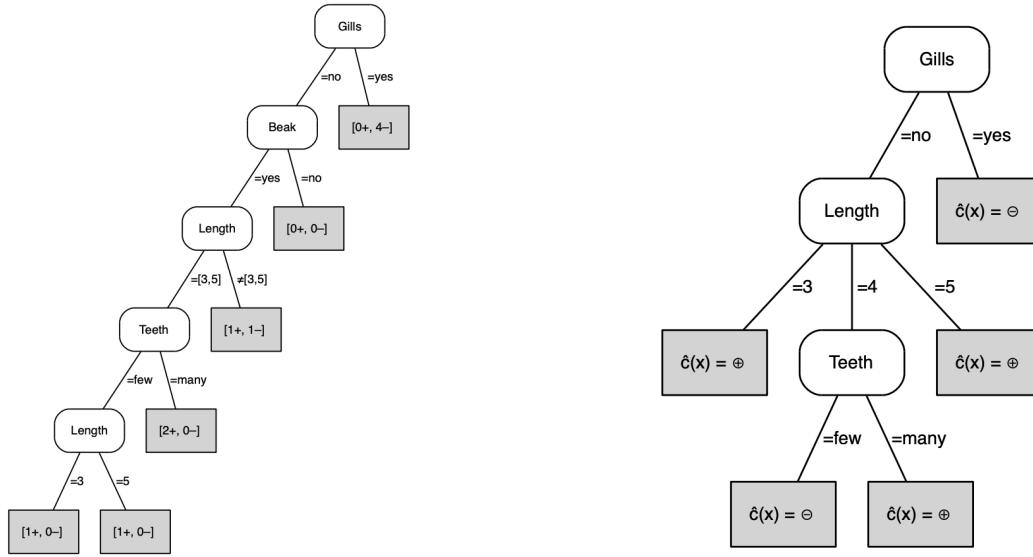
1. Chiedere per altri esempi labellati: viene chiesto un esempio che soddisfi metà delle ipotesi così il version space si dimezza
2. Usare le ipotesi del version space per classificare esempi futuri

Algoritmi per la creazione di un version space possono essere con o senza bias. Per bias si intende il “ragionamento” che l’algoritmo esegue sotto. Ad es. gli algoritmi che abbiamo fatto fino al terzo anno di uni sono senza bias perché “non ragionavano”. Find-s e Candidate Elimination invece hanno un bias e diventano capaci a generalizzare.

Gli algoritmi senza bias ragionano infatti per induzione mentre con bias ragionano per deduzione.

4 Learning Trees

Decision tree



Feature Tree

Decision Tree

Un feature tree e' un albero che ai nodi ha le feature. Da ogni nodo si dirama uno split dove ogni ramo che si va a formare prende il nome di letterale. Le foglie dell'albero rappresentano espressioni logiche: la congiunzione di espressioni dal nodo radice alla foglia

Da un feature tree si può passare ad un decision tree. Da un decision tree si possono ricavare espressioni logiche equivalenti:

$Gills = \text{no} \wedge Length = 3 \vee Gills = \text{no} \wedge Length = 5 \vee Gills = \text{no} \wedge Length = 4 \wedge Teeth = \text{many}$

Se si crea un feature tree dove ogni possibile percorso copre un solo esempio, lavora benissimo sul training set ma male nel test set → **overfitting**.

Imparare un feature tree vuol dire trovare un modo più compatto per rappresentare espressioni logiche.

Grow Tree

Algorithm GrowTree(D, F) – grow a feature tree from training data.

Input : data D ; set of features F .

Output : feature tree T with labelled leaves.

if Homogeneous(D) **then return** Label(D);

$S \leftarrow \text{BestSplit}(D, F)$; // e.g., BestSplit-Class (Algorithm 5.2)

split D into subsets D_i according to the literals in S ;

for each i **do**

if $D_i \neq \emptyset$ **then** $T_i \leftarrow \text{GrowTree}(D_i, F)$;

else T_i is a leaf labelled with Label(D);

end

return a tree whose root is labelled with S and whose children are T_i

Homogeneous ritorna true se la foglia ha elementi di una sola classe

Label ritorna la classe di maggioranza

BestSplit fa lo split utilizzano l'impurity

Per calcolare la bontà' di uno split basta calcolare la purezza dei sottoinsiemi dopo lo split:

$$\hat{p} = \frac{n^+}{n^+ + n^-} \quad \text{Empirical probability}$$

Per calcolare l'errore (**Impurity**) invece

Minority class: $\min\{\hat{p}, 1 - \hat{p}\}$

Gini index: $2\hat{p}(1 - \hat{p})$

Entropy: $-\hat{p} \cdot \log_2(\hat{p}) - (1 - \hat{p}) \cdot \log_2(1 - \hat{p})$

(probabilità di trovare un falso positivo più un falso negativo)

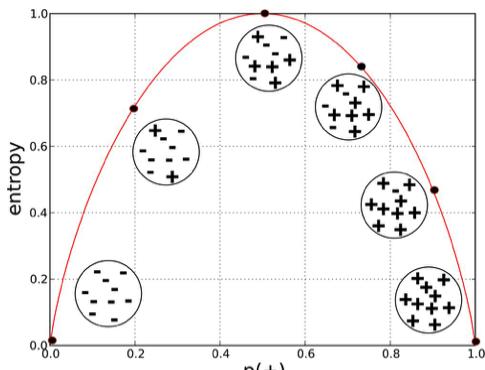
Entropy and Gini index are sensitive to fluctuations in the class distribution, $\sqrt{\text{Gini}}$ isn't (praticamente è come normalizzare l'impurità visto che la radice riduce di tanto un valore alto tipo 1000 e poco uno piccolo come 10)

Impurity di uno split

$$\text{Imp}(\{D_1, \dots, D_l\}) = \sum_{j=1}^l \frac{|D_j|}{|D|} \text{Imp}(D_j)$$

Entropy

E' una misura della quantità di confusione (può anche essere vista come misura della impurità)



nell'immagine si può vedere come cambia l'entropia in base a quanti esempi positivi abbiamo.

Per chi volesse approfondire check [this](#) out (*keanu reves intensifies*)

Best Split

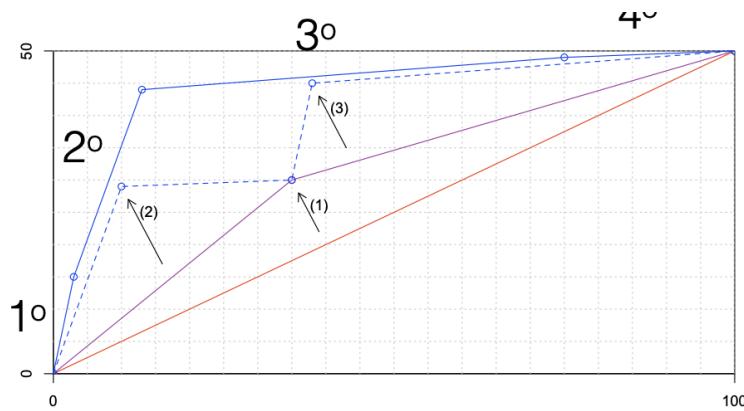
Algorithm BestSplit-Class(D, F) – find the best split for a decision tree.

Input : data D ; set of features F .
Output : feature f to split on.
 $I_{\min} \leftarrow 1;$
for each $f \in F$ **do**
 split D into subsets D_1, \dots, D_l according to the values v_j of f ;
 if $\text{Imp}(\{D_1, \dots, D_l\}) < I_{\min}$ **then**
 $I_{\min} \leftarrow \text{Imp}(\{D_1, \dots, D_l\});$
 $f_{\text{best}} \leftarrow f;$
 end
end
return f_{best}

Con questo algoritmo viene calcolata l'impurità per ogni features su cui fare uno split tramite entropia o gini index. La features con valore di impurità minore viene presa per lo split poiché è quella con meno confusione

Ranking Tree

Usato per valutazioni tra alberi di decisione. Decision tree possono essere trasformati in ranking tree se si trova un modo per ordinare le foglie in base all'empirical probability. Dopo aver ottenuto l'ordinamento e' possibile sfruttare questa informazione per usarla nei roc plot. Se non usassimo nessun ordinamento e printassimo sul roc plot le informazioni così come sono, da sinistra verso destra, otterremo segmenti sparsi lungo il piano. Invece se ordiniamo le foglie sulla base della loro purezza, otteniamo sul roc plot una curva di copertura sempre convessa.



(in blu quello ordinato. in blu tratteggiato quello non ordinato, segue quindi il semplice sinistra-destra)

clr: rapporto tra classi positive / negative

c: rapporto fra Costo fn / Costo fp

Classificazione delle foglie

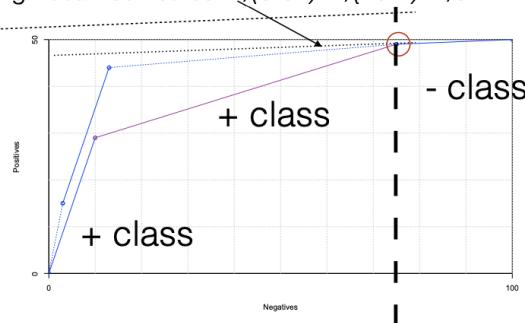
se $c > \#neg/\#pos \rightarrow$ la foglia avrà la positive class

se $c < \#neg/\#pos \rightarrow$ la foglia avrà negative class

se ho un $c=1000$ (sono più costosi i falsi negativi), mi servono 1000 negativi per battere un positivo (quindi sono più proni a dire ‘positivo’)

Un altro modo per decidere se assegnare la classe pos o neg, faccio quello spiegato sotto:
 Nei ranking tree si sceglie la label di un nodo foglia tenendo in considerazione anche i due elementi sopra. Viene infatti calcolata la pendenza di una retta con valore: $1/(c*clr)$. A questo punto bisogna andare a fare la proiezione del ranking sul roc plot. Tutti gli elementi a sinistra dell’intersezione tra la retta con slope $1/(c*clr)$ e la curva di segmenti convessa saranno positivi, negativi i restanti.

$$\text{slope avg. recall isometrics} = 1/(c*clr) = 1/(26*2) = 1/52$$



Prune

Pruno un sottoalbero se sostituendo con una foglia la majority class mi aumenta l’accuratezza rispetto a lasciare il sottoalbero.

Algorithm PruneTree(T, D) – reduced-error pruning of a decision tree.

Input : decision tree T ; labelled data D .

Output : pruned tree T' .

for every internal node N of T , starting from the bottom **do**

$T_N \leftarrow$ subtree of T rooted at N ;

$D_N \leftarrow \{x \in D | x \text{ is covered by } N\}$;

if accuracy of T_N over D_N is worse than majority class in D_N **then**

| replace T_N in T by a leaf labelled with the majority class in D_N ;

end

end

return pruned version of T

- Estimation generalization error: An alternative to the technique of reduced error pruning on the pruning set is the estimation directly on the training set of the generalization error (error on the test set) during the construction of the tree.
 Aggiunge una penalità ad ogni foglia, così da evitare la creazione della foglia se lo split non mi riduce l’errore di un valore maggiore della penalità aggiunta

- Artificial inflation: tecnica che consiste nell'aumentare le istanze di una classe dentro una foglia per coprire la differenza di costi (tipo moltiplicare per 10 il numero di classi positive)

Regression Tree

Nei regression tree la situazione è molto simile (rispetto ai feature tree). Avendo però come task l'obiettivo di trovare un numero reale e non la classe, non e' possibile utilizzare il calcolo dell'impurezza. Come valore al nodo foglia si può fare la media. Si utilizza quindi la varianza per calcolare la bontà di uno split. Alta varianza=poco coeso. Poca varianza=alto coeso. Problema di overfitting, ovvero creare una foglia per ogni esempio risultando in varianza a 0.

$$\text{Var}(Y) = \frac{1}{|Y|} \sum_{y \in Y} (y - \bar{y})^2$$

$$\text{Var}(\{Y_1, \dots, Y_l\}) = \sum_{j=1}^l \frac{|Y_j|}{|Y|} \text{Var}(Y_j) = \dots = \boxed{\frac{1}{|Y|} \sum_{y \in Y} y^2 - \boxed{\sum_{j=1}^l \frac{|Y_j|}{|Y|} \bar{y}_j^2}}$$

Quindi quando vado a fare uno split considero la varianza generata dai dataset di uno split per ogni possibile features. La varianza minore vince.

Homogeneous e' una funzione che ritorna true se la varianza si trova sotto una certa soglia t.

Label invece ritorna solitamente la media dei valori nella foglia.

Clustering Tree

Fa lo split sulle feature categoriche e anche numeriche. Associa i cluster alle istanze. Le foglie rappresentano i cluster. Anche qui non posso usare impurity. Introduco il concetto di dissimilarity: da due istanze x_1 e x_2 si cerca di capire quanto siano simili o diverse. In un sottoinsieme D calcolo la dissimilarità tra ogni coppia di istanze. La funzione di split cerca di minimizzare la Dis.

$$Dis(D) = \frac{1}{|D|^2} \sum_{x_1 \in D} \sum_{x_2 \in D} Dis(x_1, x_2)$$

La funzione di best-split cerca di minimizzare il Dis su tutti i dataset generati dallo split.

$$Dis(\{D_1, \dots, D_l\}) = \sum_{j=1}^l \frac{|D_j|}{|D|} \cdot Dis(D_j)$$

La formula di Dis su un dataset può essere riassunta in:

$$Dis(D) = 2 \cdot Var(D) = 2 \sum_{j=1}^d Var_j(D) = \frac{2}{N} \sum_{i=1}^N [x_i - \bar{x}]^2$$

Homogeneous ritorna true se la Dis è sotto una certa soglia.

Label ritorna un vettore che rappresenta la media dei valori nella foglia (sono punti del piano, quindi tipo un centroide)

A differenza del regression tree non lo uso per predire un nuovo numero, ma lo uso per trovare un punto che mi rappresenta il cluster della foglia.

Cluster più piccoli tendono ad avere dissimilarity bassa ma più overfitting. Consigliato fare pruning oppure rimuovere gli esempi “anomali”.

Cluster fatti da piccoli esempi hanno dis a 0, e queste dis dominano il calcolo delle dis
E’ raccomandato tenere un pruning set per rimuovere degli split nei nodi di basso livello se non portano benefici.

5 Rule Models

Ordered Rules List

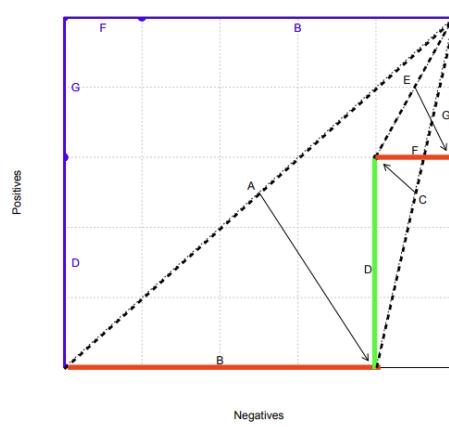
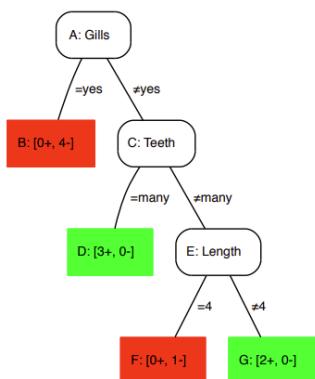
Procedura:

1. si fa crescere la regola antecedente (rule body) aggiungendo letterali in forma congiuntiva; (si cerca di ottenere un segmento grande con un alta purezza).
2. Dopo si seleziona un label (una classe target) come conseguenza del body del punto 1
3. Si rimuove le istanze coperte da questa regola e si riparte dal punto 1

Questo procedimento genera una rule list

Le rule list sono simili ai decision tree, ovvero imponiamo una regola di split. Nei rule model questi split sono nella forma IF-THEN. La key difference tra rule model e decision tree sta nel fatto che mentre nei decision tree calcoliamo l’impurezza di tutti i dataset splittati da quello originario, nei rules model invece ci interessa solo la purezza del dataset conseguente al IF (TRUE). Non ci importa del THEN in altre parole.

The feature tree



Algorithm LearnRuleList(D) – learn an

Input : labelled training data D .
Output : rule list R .

```
 $R \leftarrow \emptyset;$ 
while  $D \neq \emptyset$  do // c
     $r \leftarrow \text{LearnRule}(D)$  ;
    append  $r$  to the end of  $R$ ;
     $D \leftarrow D \setminus \{x \in D \mid x \text{ is covered by } r\}$ ;
end
return  $R$ 
```

Algorithm LearnRule(D) – learn a single rule.

Input : labelled training data D .
Output : rule r .

```
 $b \leftarrow \text{true};$ 
 $L \leftarrow \text{set of available literals};$ 
while not Homogeneous( $D$ ) do // k
     $l \leftarrow \text{BestLiteral}(D, L)$  ;
     $b \leftarrow b \wedge l$ ;
     $D \leftarrow \{x \in D \mid x \text{ is covered by } b\}$ ;
     $L \leftarrow L \setminus \{l' \in L \mid l' \text{ uses same feature as } l\}$ ;
end
 $C \leftarrow \text{Label}(D)$  ;
 $r \leftarrow \cdot \text{if } b \text{ then Class} = C \cdot$ ;
return  $r$ 
```

Best Literal: literale con purezza più alta

Rule lists for ranking

Non sempre la lista generata delle regole genera un insieme ordinato per ranking di foglie dell'albero. Calcolo quindi l'empirical probability per ogni foglia dell'albero generato dal rule model. Ordino le foglie sulla base dell'empirical probability e calcolo il ranking error e poi posso disegnare il roc plot. In questo modo posso capire quale rule model è migliore se vogliono confrontare 2 o più.

Un-ordered Rules Set

Procedimento:

1. Prima si seleziona una classe target(e mi concentro su quella fino alla fine)
2. Poi si selezionato le regole (body) che coprono al meglio le istanze appartenenti alla classe del punto 1(in genere si cercano regole che generino insiemi di istanze puri: si usa la precision (TP/TP+FP o il contrario se siamo nella classe neg). Queste istanze poi vengono rimosse. Si ripete il punto 2 finché tutte le istanze sono coperte)
3. Questo produce un insieme di regole. Non c'è motivo di ordinarle dato che tutte le regole coprono la stessa classe (sarebbe if(){+} if(){+} if(){+}, quindi una vale l'altra come ordine)

Algorithm LearnRuleSet(D) – learn an unorder

Input : labelled training data D .
Output : rule set R .

```
R ←  $\emptyset$ ;
for every class  $C_i$  do                                // we iterate over classes
     $D_i \leftarrow D$ ;                                // we take the examples of class  $C_i$ 
    while  $D_i$  contains examples of class  $C_i$  do
         $r \leftarrow \text{LearnRuleForClass}(D_i, C_i)$ ; // LearnRuleForClass( $D_i, C_i$ )
        R ← R ∪ {r};
         $D_i \leftarrow D_i \setminus \{x \in C_i | x \text{ is covered by } r\}$ ;
    end
end
return R
```

Algorithm LearnRuleForClass(D, C_i) – learn a

Input : labelled training data D ; class C_i .
Output : rule r .

```
b ← true;
L ← set of available literals;                      // car
while not Homogeneous(D) do
    l ← BestLiteral(D, L, C_i);           // e.g. m
    b ← b ∧ l;
    D ← {x ∈ D | x is covered by b};
    L ← L \ {l' ∈ L | l' uses same feature as l};
end
r ← ·if b then Class =  $C_i$ ·;
return r
```

Uno dei problemi è che cercando sempre di creare insiemi puri non si riconoscono altre strade che possono essere altrettanto o persino più buone. Un modo per superare il problema è usare la laplace correction. Perché la purezza di una foglia [2,0] sara' sempre più alta di un [999998, 1], anche se la seconda foglia e' senza ombra di dubbio la più cazzuta. Quindi per fotttere la prima foglia aggiungo un 1 ad entrambe le parti → [3,1] vs [999999, 2]

Ranking

Come nelle rule list si può fare ranking ed ordinare le foglie del decision tree. In questo modo posso confrontare più modelli. Però dato che le regole non sono (sempre) in mutua esclusione non posso fare il ranking classico senza fare prima operazioni. Quindi se io ho 3 regole A, B, C, devo calcolare questi sottoinsiemi (NB:nell'esempio sotto non considero AC perché sono due regole già in mutua esclusione)

Da questo

- (A) ·if Length = 4 then Class = ⊖
- (B) ·if Beak = yes then Class = ⊕
- (C) ·if Length = 5 then Class = ⊖

A questo. Ora posso fare il roc plot

$$\begin{aligned}
 & (\mathbf{B} \setminus \{\mathbf{A} \cup \mathbf{C}\}) [\{p_1, p_3\}, \{\}] \quad p_B = 2/2 = 1 \\
 & (\mathbf{A} \setminus \{\mathbf{B} \cup \mathbf{C}\}) [\{\}, \{n_4\}] \quad p_A = 0/1 = 0 \\
 & (\mathbf{C} \setminus \{\mathbf{A} \cup \mathbf{B}\}) [\{\}, \{n_3\}] \quad p_C = 0/1 = 0 \\
 & (\mathbf{AB}) [\{p_2\}, \{n_2, n_5\}] \quad p_{AB} = 1/3 = 0.33 \\
 & (\mathbf{BC}) [\{p_4, p_5\}, \{n_1\}] \quad p_{BC} = 2/3 = 0.66
 \end{aligned}$$

Pero calcolare questi sottoinsiemi e' troppo oneroso. Quindi posso approssimare questi sottoinsiemi (AB, BC, AC) stimando l'empirical probability. Non avrò la stessa precisione ma almeno non divento pazzo a enumerare tutti i sottoinsiemi.

Summary

Rule Set vs Rule List

$$\begin{aligned}
 & (\mathbf{A}) \cdot \text{if Length} = 4 \text{ then Class} = \ominus \quad [1+, 3-] \quad [\{p_2\}, \{n_2, n_4, n_5\}] \\
 & (\mathbf{B}) \cdot \text{if Beak} = \text{yes} \text{ then Class} = \oplus \quad [5+, 3-] \quad [\{p_1-p_5\}, \{n_1, n_2, n_5\}] \\
 & (\mathbf{C}) \cdot \text{if Length} = 5 \text{ then Class} = \ominus \quad [2+, 2-] \quad [\{p_4, p_5\}, \{n_1, n_3\}]
 \end{aligned}$$

$$\begin{aligned}
 & [\{p_2\}, \{n_2, n_4, n_5\}] \quad \cdot \text{if Length} = 4 \text{ then Class} = \ominus \quad [1+, 3-] \\
 & [\{p_1, p_3-p_5\}, \{n_1\}] \quad \cdot \text{else if Beak} = \text{yes} \text{ then Class} = \oplus \quad [4+, 1-] \\
 & [\{\}, \{n_3\}] \quad \cdot \text{else if Length} = 5 \text{ then Class} = \ominus \quad [0+, 1-]
 \end{aligned}$$

I rule set generalmente si comportano meglio because (shoc) a rule list composed by n rules will identify n instance segments, while a rule set can potentially give access also to the rule overlaps that are at most 2^n .

In un rule set, dato che non sono ordinate le regole, possiamo avere che un esempio può essere coperto da due regole contemporaneamente. Fin qui tutto bene, ma se queste due regole predicono classe opposte a chi do ascolto? In generale è meglio considerare la regola con copertura più ampia per un discorso associato alla maximum likelihood.

Descriptive rule learning

I modelli descrittivi non si occupano di predire una classe, vengono usati per trovare una descrizione comune a quelle istanze che sappiamo che fanno parte di una porzione omogenea del dataset (in termini di classe).

Si usa average recall perché la formula di quella misura di precisione non si focalizza unicamente sulla classe di positivi. Stesso discorso nel descriptive rule learning, a noi non interessa focalizzarci su una classe specifica, vogliamo solo descrivere le istanze

Subgroup discovery

Trovare la regola (anche composta da più condizioni in AND) per avere il subset il più puro possibile.

La bontà dei subgroup si può misurare con **avg recall** o **precision(avg recall)** è più precisa poiché considera sia la classe negativa che quella positiva).

A differenza della classificazione, non togliamo gli esempi dopo aver fatto la regola.

Weighted covering

Una differenza della classification rule learning rispetto a subgroup discovery è che in subgroup si cerca di trovare più regole che coprono lo stesso esempio.

Un modo per fare ciò è di non rimuovere gli esempi già coperti ma di ridurre i pesi (inizializzati a 1 e dimezzati ogni volta).

Le euristiche di ricerca sono poi valutate in termini di somma dei pesi degli esempi coperti (non più in base solo al numero)

Algorithm WeightedCovering(D) – learn overlapping rules by weighting examples.

Input : labelled training data D with instance weights initialised to 1.

Output : rule list R .

```
1  $R \leftarrow \emptyset$ ;
2 while some examples in  $D$  have weight 1 do
3    $r \leftarrow \text{LearnRule}(D)$ ;                                // LearnRule: see Algorithm 6.2
4   append  $r$  to the end of  $R$ ;
5   decrease the weights of examples covered by  $r$ ;
6 end
7 return  $R$ 
```

LearnRule è lo stesso algoritmo trattato per i rule set (BestLiteral potrebbe usare avg_recall)

Association rule mining

Serve per trovare set di items che stanno bene insieme. Ovvero se io compro pane, magari voglio comprare anche il prosciutto e la birra ma non mi interessano le mele. Lo scopo di questo algoritmo è quindi trovare queste associazioni tra items (probabilmente è quello che si usa nelle ads per esempi pratici)

Definitions for classes of item sets

- An **item set** I is a set of items $\{i_1, i_2, \dots, i_n\}$ such that for any i_x belonging to I they all belong to the same transaction t
- A **frequent item set** I is an item set $\{i_1, i_2, \dots, i_n\}$ such that for any item i_x belonging to I it belongs to the same set of transactions $T = \{t_1, \dots, t_m\}$ with $|T| \geq f_o$ (minimum support)
- A **closed item set** I is an item set such that another item set I_2 , such that $I_2 \supseteq I$ with the same support ($f(I_2) = f(I)$) does not exist
- A **maximal item set** I is a frequent closed item set such that another item set I_2 , $I_2 \supsetneq I$ with sufficient support does not exist.
In other terms, the maximal item sets are the vertices at the lower border in the lattice separating frequent item sets from unfrequent item sets (above the border we find only vertices having a sufficient frequency and under the border we find only vertices with non sufficient frequency).

Item sets: albero con tutti i nodi che rappresentano gli itemset delle transazioni

closed sets: albero con tutti i nodi massimali per una certa(certe) transazioni. va bene usarlo perché non perde informazione, infatti da un closed sets posso derivare un qualsiasi item set che non e' coperto esplicitamente dall'albero.

frequent item sets: albero composto da item set frequenti almeno T

maximal item sets: intersezione fra closed e frequent item sets

Association rule

Dal frequent item sets possiamo scrivere association rules. L'obiettivo e' creare delle regole per cui gli item sets siano frequenti e la confidenza della regola sia sopra una certa soglia.

Sono if del tipo if *nappies* then *beer* $\rightarrow 2/4$. dove 2 sono il numeri di transazioni (confidenze) $\{\text{nappies}, \text{beer}\}$ e 4 le transazioni (support) $\{\text{nappies}\}$.

Algorithm AssociationRules(D, f_0, c_0) – find all association rules exceeding given support and confidence thresholds.

Input : data $D \subseteq \mathcal{X}$; support threshold f_0 ; confidence threshold c_0 .

Output : set of association rules R .

```

1  $R \leftarrow \emptyset;$ 
2  $M \leftarrow \text{FrequentItems}(D, f_0);$  // FrequentItems: see Algorithm 6.6
3 for each  $m \in M$  do
4   for each  $H \subseteq m$  and  $B \subseteq m$  such that  $H \cap B = \emptyset$  do
5     if  $\text{Supp}(B \cup H)/\text{Supp}(B) \geq c_0$  then  $R \leftarrow R \cup \{\cdot \text{if } B \text{ then } H\};$ 
6   end
7 end
8 return  $R$ 
```

Algorithm FrequentItems(D, f_0) – find all maximal item sets exceeding a given support threshold.

Input : data $D \subseteq \mathcal{X}$; support threshold f_0 .

Output : set of maximal frequent item sets M .

```

1  $M \leftarrow \emptyset;$ 
2 initialise priority queue  $Q$  to contain the empty item set; 
3 while  $Q$  is not empty do
4    $I \leftarrow$  next item set deleted from front of  $Q$ ;
5    $max \leftarrow \text{true};$  // flag to indicate whether  $I$  is maximal
6   for each possible superset  $I'$  of  $I$  do
7     if  $\text{Supp}(I') \geq f_0$  then
8        $max \leftarrow \text{false};$  // frequent superset found, so  $I$  is not maximal
9       add  $I'$  to back of  $Q$ ;
10      end
11    end
12    if  $max = \text{true}$  then  $M \leftarrow M \cup \{I\};$ 
13  end
14 return  $M$ 
```

Supp= In quante transazione compare un itemset rapportato al totale delle transazioni, se tipo {beer, crisp} compare in 2 transizioni e il totale delle transazioni è 5 il risultato sarà 2/5
 Confidence= $\text{Supp}(b \cup h)/\text{Supp}(b)$ conta il rapporto di quante volte se si è comprato b come conseguenza ad aver comprato h

6 Linear Models (esposito)

Nei modelli lineari cerchiamo di trovare un iperpiano che fitti nel miglior modo possibile i dati. Vogliamo quindi trovare dei pesi per cui $Cx + D = y$. Dato un dataset x ed i target y possiamo scrivere un sistema di equazioni $x_i C + D = y_i$ per ogni i .

Questo sistema di equazioni può essere visto come un'equazione matriciale $Xw = y$ con $w = [C, D]$ vettore dei pesi, y vettore target e X matrice del mio dataset.

Possiamo risolvere ponendo $w = X^{-1} * y$, ma questo comporta avere una matrice X quadrata con numeri esempi = numero features → caso triviale.

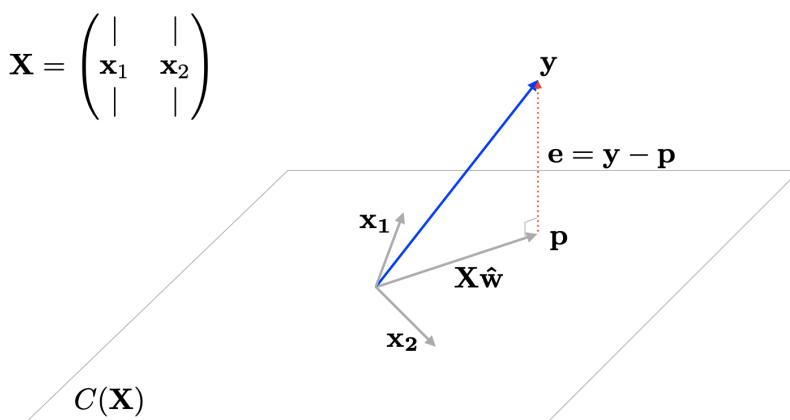
Dato che non possiamo risolvere perfettamente questa equazione, dobbiamo trovare una soluzione che minimizzi l'errore, con l'errore definito come $e = ||Xw - y||$. Abbiamo quindi che dobbiamo minimizzare i minimi quadrati.

Least squares

E' possibile arrivare a una soluzione approssimata minimizzando lo scarto quadratico medio

$$\text{minimize}_{\hat{w}} \|\mathbf{X}\hat{w} - \mathbf{y}\|_2^2$$

Da un punto di vista geometrico, siamo in un piano $C(X)$ dove il nostro vettore Xw spazia su questo piano. La soluzione y invece si trova in uno spazio differente. Dobbiamo comunque minimizzare la distanza fra il vettore Xw e y , ovvero il vettore di errore e . Questo vettore si trova al suo minimo quando è perpendicolare al piano $C(X)$. Ponendo quindi l'ortogonalità possiamo quindi trovare una soluzione ottima.



Least Squares Ortogonale

Si impone dunque $\mathbf{X}^T \mathbf{e} = 0$ poiché si pone il vettore e perpendicolare al piano che è composto da \mathbf{x} trasposto. Facendo tutti i passaggi:

$$\mathbf{X}^T(\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}) = 0$$

$$\mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\hat{\mathbf{w}} = 0$$

$$\mathbf{X}^T\mathbf{X}\hat{\mathbf{w}} = \mathbf{X}^T\mathbf{y}$$

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

ovvero matrice pseudo inversa

Regularised regression

Least Squared può essere usato per la regressione. Però, dato che ci possono essere outliers che attirano verso di sé la retta, aumentando di fatto l'errore di generalizzazione, bisogna introdurre un termine regolarizzazione. Uno di questi è la regularised regression.

Metodo 1: ridge (norma 2)

Aggiungendo un termine(uno scalare, iper parametro) mi permette di tenere basso il valore dei pesi \mathbf{W} , evitando overfitting. Pesi bassi rendono più basso l'errore, nel senso che soluzioni più semplici permettono maggiore generalizzazione (rasoio di occam). Quindi l'obiettivo del least square e':

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{arg\,min}} \ (\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \|\mathbf{w}\|^2$$

Ovvero la stessa formula del non regularized ma con l'aggiunta di lambda*norma2 dei pesi \mathbf{w} . lambda indica la scala di regolarizzazione, e la soluzione al problema è:

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

Metodo 2: lasso (norma 1)

$$\|\mathbf{w}\|_1 = \sum_i |w_i|$$

sostituisce $\|\mathbf{w}\|^2$ nella formula di ridge con $\|\mathbf{w}\|_1$
 con questa regolarizzazione alcuni pesi vengono rimpiccioltiti
 mentre la maggior parte diventano 0, quindi favorisce soluzioni sparse.

Least Squared For Classification

Least squared può anche essere usato per task di classificazione (1 per la classe positiva e -1 per la classe negativa)

$$\hat{c}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x}^T \hat{\mathbf{w}} - t > 0 \\ 0 & \text{if } \mathbf{x}^T \hat{\mathbf{w}} - t = 0 \\ -1 & \text{if } \mathbf{x}^T \hat{\mathbf{w}} - t < 0 \end{cases}$$

t e' tipo il bias delle reti neurali

Support Vector Machine

Il primo obiettivo delle support vector machine e' quello di trovare un iperpiano che separi correttamente le istanze → questo ci porta ad aver trovato un margine funzionale. Ovvero che ci dice semplicemente se le istanze sono classificate correttamente o meno.

Una volta trovato il margine funzionale, dobbiamo cercare di massimizzare la distanza fra i punti più vicini dei negativi e positivi → questo ci porta alla definizione di margine geometrico. Ovvero la distanza fra questi due punti che deve essere massimizzata.

Il margine funzionale e' scritto come:

$$\mu(\mathbf{x}_i) = y_i(\mathbf{w} \cdot \mathbf{x}_i - t)$$

dove $y_i(x_i) > 0$ se l'istanza e' classificata correttamente. Minore di 0 se ho errori di classificazione. Quindi vogliamo imporre che questo margine sia sempre maggiore di 0 per ogni esempio: $y_i(w \cdot x_i - t) > 0$.

Vogliamo pero' imporre che i nostri vettori di supporto corrispondano a $y_i(w \cdot x_i - t) = 1$, mentre tutti gli altri esempi posti a $y_i(w \cdot x_i - t) \geq 1$.

Questo posso farlo per qualsiasi costante c positiva, spiegazione matematica:

Se ho che $y_i(w \cdot x_i - t) > 0$ allora esiste un valore epsilon e per cui $y_i(w \cdot x_i - t) \geq e$.

Poi moltiplico da entrambi i lati per una costante positiva c , quindi:

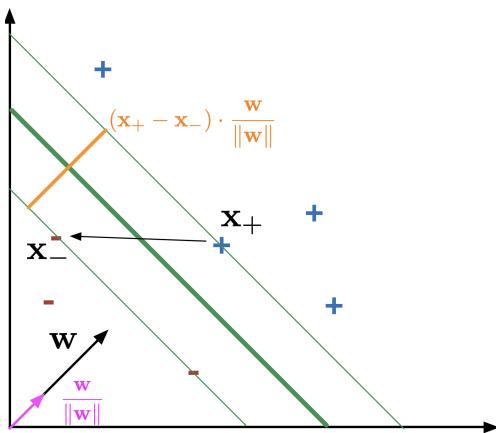
$c * y_i(w \cdot x_i - t) \geq e * c$ e poi divido da entrambe le parti per e , quindi:

$c/e * y_i(w \cdot x_i - t) \geq e/e * c$. Semplificando mi rimane quindi

$c/e * y_i(w \cdot x_i - t) \geq c$. Ora posso inserire c/e dentro la funzione:

$y_i(c/e * w \cdot x_i - t * c/e) \geq c$. Dato che sia w che t sono valori che vengono trovati dal mio algoritmo, posso inglobare c/e dentro questi valori. risultando quindi in $y_i(w \cdot x_i - t) \geq c$

La parte rosa in figura è il versore del vettore dei pesi ed è sempre perpendicolare alla retta principale: indica così il verso del margine. A livello algebrico quello detto sopra si traduce dal fatto che abbiamo posto l'originalità di w rispetto alla retta principale assumendo che la retta principale sia scritta come: $wx - t = 0$.



la distanza del margine geometrico si calcola:

$$\begin{aligned}\mu &= (\mathbf{x}_+ - \mathbf{x}_-) \cdot \frac{\mathbf{w}}{\|\mathbf{w}\|} \\ &= \frac{\mathbf{x}_+ \cdot \mathbf{w}}{\|\mathbf{w}\|} - \frac{\mathbf{x}_- \cdot \mathbf{w}}{\|\mathbf{w}\|}\end{aligned}$$

Ovvero vettore differenza moltiplicato per il versore dei pesi.

Dato che \mathbf{x}_+ e \mathbf{x}_- sono esempi sul bordo, abbiamo che:

- $\mathbf{w} \cdot \mathbf{x}_+ - t = 1 \rightarrow \mathbf{w} \cdot \mathbf{x}_+ = 1 + t$
- $-(\mathbf{w} \cdot \mathbf{x}_- - t) = 1 \rightarrow \mathbf{w} \cdot \mathbf{x}_- = t - 1$

Quindi sostituendo nella formula sopra (quella del margine) i numeratori con le formule ricavate sotto (dell'elenco puntato), abbiamo che:

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

Per ottimizzare il margine si può usare la formula appena ricavata, cambiando così in automatico pendenza e bias alla retta. Essendo che si massimizza $2/\|\mathbf{w}\|^2$ è la stessa cosa scrivere di minimizzare direttamente $\|\mathbf{w}\|^2$. Poi si minimizza $\|\mathbf{w}\|^2$ per qualche motivo che solo i matematici sanno. (per una questione di monotonia crescente)

$$\begin{array}{ll}\text{minimize}_{\mathbf{w}, t} & \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{subject to} & y_i(\mathbf{w} \cdot \mathbf{x}_i - t) \geq 1; \quad 0 \leq i \leq n\end{array}$$

Dual Problem & KKT

Un problema di minimo si può riscrivere in forma duale (e deve sottostare alle KKT conditions).

Alla fine uscirà che si deve massimizzare una funzione.

Vantaggi:

- la soluzione del vettore w è la combinazione dei vettori di supporto grazie all'ultima regola delle KKT $a_i f_i(x) = 0 \rightarrow a_i(y_i(w^*x_i - t) - 1) = 0 \quad a_i > 0$ allora $y_i(w^*x_i - t) - 1 = 0$ allora x_i è un vettore di supporto
- learning e classificazione si possono fare usando il prodotto scalare di vettori di supporto

Soft margins

I margins visti in precedenza (hard) non operano bene se sono presenti esempi anomali (outliers), tendono a fare overfitting, per ovviare a questo problema si introducono delle variabili (slack variables, che mi rilassano i vincoli) per permettere la misclassificazione e rendere il modello più generale.

$$\begin{aligned} \underset{\mathbf{w}, t, \xi}{\text{minimize}} \quad & \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i \\ \text{subject to} \quad & y_i(\mathbf{w} \cdot \mathbf{x}_i - t) \geq 1 - \xi_i; \quad 1 \leq i \leq n \\ & \xi_i \geq 0; \quad 1 \leq i \leq n \end{aligned}$$

Se $C = 0$ gli errori e rimangono ma $w = 0$? Valori alti di C invece permettono di penalizzare maggiormente gli errori di misclassificazione e evitando che il margine aumenti di troppo. C è direttamente proporzionale al peso dell'errore.

C è un parametro definito dall'utente per controllare gli errori sul margine

Versione duale

E' uguale a prima. si aggiunge questo vincolo $0 \leq a_i \leq C$

si deduce che

- $a_i = 0$ esempi sul margine o fuori
- $0 < a_i < C$ esempi sono vettori di supporto
- $a_i = C$ errori di margine

Kernels

Kernel Function

Calcola il prodotto interno tra due vettori \mathbf{x} e \mathbf{y} , ciò vuol dire che la funzione è simmetrica, lineare nel primo argomento, semipositiva definita. Quindi prende in input due vettori e ritorna un numero reale. Per funzionare deve esistere uno spazio (spazio di Hilbert) in cui mappare questi 2 vettori e dove si può svolgere un prodotto interno (sarebbe una generalizzazione del prodotto scalare).

Vantaggi:

- mi permette di estendere classificatori lineari a problemi non lineari
- calcola una funzione di similarità in uno spazio ad alta dimensionalità senza andare a calcolare direttamente il nuovo vettore in questo spazio
- esistono teoremi che mi verificano la validità di un kernel function o che mi permettono di creare funzioni kernel a partire da kernel esistenti

Kernel Trick

Un modo per usare computazionalmente il kernel. Nasce dall'idea che un problema non linearmente separabile, può diventare linearmente separabile se trasformato in un'altra dimensione.

Ci sono diversi tipi di kernel, il più comune è il polynomial kernel. Il polynomial kernel trasforma il dataset aumentando la dimensionalità di esso per un fattore d . Ovvero se ho i dati su una linea (1-d), quando ho $d=2$ i dati vengono trasformati in uno spazio 2-d, quindi bidimensionale. Questa trasformazione permette di rendere un problema linearmente separabile partendo da dati che non lo sono.

Per farlo il kernel eleva per un fattore d il prodotto interno fra due vettori senza però andare a mappare realmente i dati dalla dimensione originale alla nuova dimensione d .

Infine i kernel possono anche essere composti.

Kernel importanti:

Polynomial Kernel of degree d :

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d \quad \text{or} \quad K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y} + 1)^d$$

Gaussian Kernel:

$$K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2}\right)$$

Mercer's conditions

Ci aiutano a capire che una funzione siano funzioni kernel valide senza verificare che esista lo spazio di Hilbert.

A function K is a valid kernel if and only if for any finite set of points $\{x_1, \dots, x_m\}$, the matrix \mathbf{K} (defined as $\mathbf{K}_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$) is symmetric and positive semi-definite.

7 Distance Based Model

Si misura la similarità tra due esempi calcolando la distanza tra le loro feature.

Intro

Minkowski distance

Norma p

$$\text{Dis}_p(\mathbf{x}, \mathbf{y}) = \left(\sum_{j=1}^d |x_j - y_j|^p \right)^{1/p} = \|\mathbf{x} - \mathbf{y}\|_p$$

Distanza euclidea ($p=2$)

$$\text{Dis}_2(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{j=1}^d (x_j - y_j)^2} = \sqrt{(\mathbf{x} - \mathbf{y})^T (\mathbf{x} - \mathbf{y})}$$

Distanza di manhattan ($p=1$)

$$\text{Dis}_1(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^d |x_j - y_j|$$

Più p è grande, più il calcolo sarà orientato verso le feature che hanno valore più alto.

La distanza di Manhattan funziona generalmente solo se i punti sono disposti sotto forma di una griglia e il problema su cui stiamo lavorando dà più priorità alla distanza tra i punti solo insieme alle griglie, ma non alla distanza geometrica.

Edit Distance ($p=0$)

Conta quante feature sono diverse tra x e y . Conta il numero di elementi diversi da 0 nel vettore.

Dis

La funzione di distanza deve rispettare certe condizioni

1. distanza tra lo stesso punto e' 0
2. tutte le altre distanze > 0
3. distanze sono simmetriche
4. detours non restringono la distanza

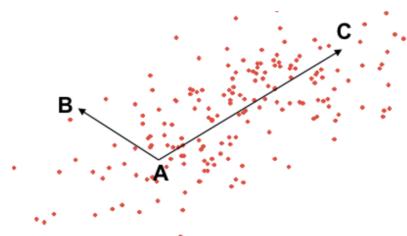
Se la seconda condizione non viene rispettata, cioè ho 0 tra due punti diversi, ho una pseudo-dis.

Distanza di Mahalanobis

Tiene in considerazione la distribuzione dei dati. Viene calcolata la distribuzione ellittica dei dati tramite l'inversa della matrice di covarianza. Questa è una matrice che rappresenta la variazione di ogni variabile rispetto alle altre (inclusa se stessa). La distanza euclidea e' un esempio della distanza di mahalanobis dove l'inversa della matrice di covarianza e' uguale a 1 in quanto i dati sono equamente distribuiti. La matrice di covarianza descrive la varianza dei dati per ogni feature.

$$\text{Dis}_M(\mathbf{x}, \mathbf{y} | \Sigma) = \sqrt{(\mathbf{x} - \mathbf{y})^T \Sigma^{-1} (\mathbf{x} - \mathbf{y})}$$

La $\text{dist}(A,C)$ e' < di $\text{dist}(A,B)$ a causa della matrice di covarianza



Centroidi e Medoidi

Un centroide e' un punto che si trova esattamente nella media aritmetica di tutti i punti di un dataset. Questo punto minimizza la sommatoria delle distanze euclidee rispetto a tutti i punti del dataset.

Un medoide è invece un punto del dataset che si comporta come un centroide, ovvero minimizza la sommatoria delle distanze euclidee. La differenza rispetto al centroide e' che in questo caso il punto di medoide e' di fatto un punto del dataset

Summary

Nei modelli basati sulla distanza dobbiamo quindi prendere in considerazione

- metrica di distanza(euclidea, manhattan, mahalanobis ecc)
- exemplars: i centroidi o medoidi per ogni classe/cluster/dataset
- decision rules: regole che prendono una decisione sulla base dei k exemplars più vicini

Classificatore k-nearest neighbour (knn)

In breve controlla i k -punti più vicini (presi dai punti del dataset), rispetto al punto in input, e di quali classe appartengono. La classe di maggioranza vince.

- k-NN con $k=1$, ha low-bias ma varianza alta (la classificazione cambia sempre in base agli input)
- k-NN con $k=n$, ha alto bias ma low varianza (perché classifico sempre la stessa classe indipendentemente dall'input)

k è un iperparametro, devo stare attento alla scelta. Per risolvere il problema introduco i pesi in base alla distanza.

Nel distance weighting si tiene conto di quanto distano i k -punti. Più un punto è lontano, minore il suo contributo alla scelta. Di conseguenza la scelta di k è meno influente rispetto a prima (anche con $k=n$ ho un certo livello di varianza)

K-nearest neighbour density

Approccio k-nearest usato non per classificare ma per description purposes, un esempio è dbSCAN:

DBSCAN

Density=variabile numerica usata per suddividere i punti. Eps=raggio. Ci sono tre tipi di punti

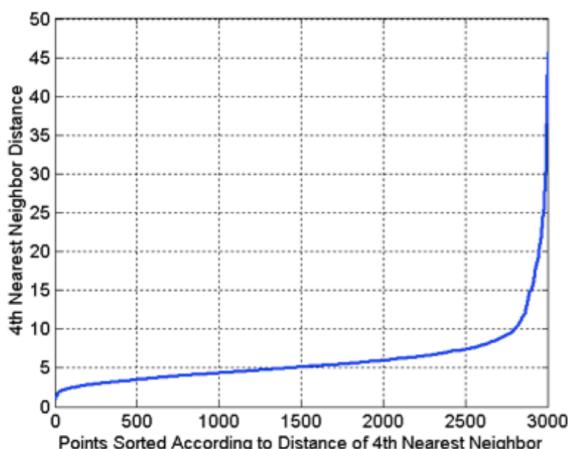
1. core point: punto che ha nel suo eps almeno n -density point
2. border point: non ha n -density vicini ma sta nel raggio di un core point
3. noise point: nessuno dei primi due (ovvero è un punto distante dagli altri)

DbSCAN riesce a distinguere cluster di diversa forma e grandezza.

Non funziona bene quando gli esempi sono in high-dimensionality (come knn) e quando le densità sono variabili.

Inoltre piccoli cambiamenti agli iperparametri portano a risultati anche molto differenti. Il raggio può essere trovato in maniera empirica. (vedi grafico)

Per trovare un valore ottimo per il raggio si può lanciare un algoritmo di k-nearest, e prendere la distanza del k-nearest per ogni esempio. Si plotta poi su un grafico e si prende come valore il punto dove sta la curva a gomito. Questo perché i noise point avranno questa distanza più alta rispetto ai core e border point. Quindi è un buon valore per poter distinguere bene i due tipi di punti.



Algoritmo

1. etichetto i punti come core, border o noise
2. elimino i noise points
3. collego i core point all'interno del raggio Eps
4. i punti core connessi identificano un cluster
5. assegno ai punti di bordo il cluster più vicino

Distance Based Clustering

WSS (Cluster cohesion): misura quanto sono coesi i punti interni a un cluster

BSS (Cluster separation): misura quanto sono distinti i vari cluster gli uni dagli altri

Un modo per fare questi calcoli e' usare la sum squared error (SSE).

SSE = WSS + BSS → costante. Questo perché se alzo la WSS si abbassa di conseguenza la BSS e viceversa. Si può anche vedere come la varianza di un dataset, che non cambia a seguito di clustering, i dati sono quelli e non cambiano.

La cosa migliore e' minimizzare WSS.

K-Means

Trovare una partizione del dataset composta da k subset dove WSS e' minimizzata (o BSS massimizzata)

Funzionamento:

1. Inizializza k centroidi a random
2. Associa per ogni punto del dataset il centroide più vicino creando quindi k cluster
3. Viene ricalcolato il centroide per ogni cluster
4. Vengono modificati i cluster di appartenenza ai punti del dataset: per ogni punto viene ri-associato il cluster rispettivo rispetto al centroide più vicino
5. Ritorna al punto 3 per x volte o finché non si trova convergenza.

Il problema di questo algoritmo e' che inizializza i centroidi a random, questo può portare a convergenza sub-ottimali. Inoltre k-means e' orientato verso i cluster di forma sferica (distanza euclidea?), provocando grossi errori nel caso di cluster con forma diversa.

Un altro problema è che bisogna fissare il parametro K.

E' possibile anche implementare una versione dell'algoritmo che utilizza i medoidi al posto dei centroidi. Può essere utile usare i medoidi perché i centroidi sono molto influenzati dagli outliers, rendendo di fatto la soluzione non ottimale.

Silhouette

Metrica di valutazione della coesione dei cluster.

$$s(\mathbf{x}_i) = \frac{b(\mathbf{x}_i) - a(\mathbf{x}_i)}{\max(a(\mathbf{x}_i), b(\mathbf{x}_i))}$$

a(\mathbf{x}_i) = media tra la distanza dell'esempio \mathbf{x}_i con gli esempi del suo cluster

b(\mathbf{x}_i) = media tra la distanza dell'esempio \mathbf{x}_i con gli esempi del cluster adiacente a lui

$s(x_i)$ = valore di silhouette che indica quanto è piazzato bene quell'esempio dentro il cluster.
la silhouette indica se un esempio è stato mal clusterizzato [-1, +1]

Il calcolo viene effettuato per ogni cluster e per ogni esempio di ogni cluster. I risultati possono essere plottati su un piano e ordinati. Se escono fuori blocchi più simili a rettangoli vuol dire che abbiamo dei cluster fatti bene, altrimenti se i blocchi sono più appuntiti e' peggio.

Hierarchical Clustering

Un tipo di clustering che ha il vantaggio di non specificare in precedenza il numero di K. Da questo tipo di clustering deriva un dendrogramma (albero "rovesciato") e da questo dendrogramma è possibile suddividere in un qualsivoglia k-cluster e ottenere i cluster migliori

Un dendrogramma ha come foglie gli elementi del dataset. I nodi interni invece rappresentano sottoinsiemi che contengono 1 o più foglie. L'altezza di un nodo rappresenta la distanza, ovvero quanto sono differenti, dei cluster che suddivide. La funzione di linkage calcola la suddetta distanza.

- single linkage: la distanza minore tra tutte le coppie di distanze tra due cluster
- complete linkage: la distanza maggiore tra tutte le coppie di distanze tra due cluster
- average linkage: la distanza media dei punti di due cluster
- centroid linkage: la distanza tra i centroidi dei due cluster

Algorithm HAC(D, L) – Hierarchical agglomerative clustering.

Input : data $D \subseteq \mathcal{X}$; linkage function $L : 2^{\mathcal{X}} \times 2^{\mathcal{X}} \rightarrow \mathbb{R}$ defined in terms of distance metric.

Output : a dendrogram representing a descriptive clustering of D .

- 1 initialise clusters to singleton data points;
 - 2 create a leaf at level 0 for every singleton cluster;
 - 3 **repeat**
 - 4 | find the pair of clusters X, Y with lowest linkage l , and merge;
 - 5 | create a parent of X, Y at level l ;
 - 6 **until** all data points are in one cluster;
 - 7 **return** the constructed binary tree with linkage levels;
-

From kernels to distances

I kernel possono essere usati nel calcolo di similarità tra due esempi, con il vantaggio che non devono calcolare esplicitamente il vettore nel nuovo spazio.

Possono essere usati in due modi:

Euclidean Distance

La distanza euclidea può essere vista come un dot product, e dato che i kernel sono dot product, si possono usare funzioni di kernel (può essere considerato un kernel trick)
Quindi la distanza euclidea diventa un dot product

$$\text{Dis}_2(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2 = \sqrt{(\mathbf{x} - \mathbf{y}) \cdot (\mathbf{x} - \mathbf{y})} = \sqrt{\mathbf{x} \cdot \mathbf{x} - 2\mathbf{x} \cdot \mathbf{y} + \mathbf{y} \cdot \mathbf{y}}$$

by the distributive property

e poi sostituiamo il dot product con una funzione di kernel

$$\text{Dis}_K(\mathbf{x}, \mathbf{y}) = \sqrt{K(\mathbf{x}, \mathbf{x}) - 2K(\mathbf{x}, \mathbf{y}) + K(\mathbf{y}, \mathbf{y})}$$

Cosine Similarity

Da un punto di vista geometrico, il prodotto scalare calcola implicitamente la similarità tra due vettori. L'angolo tra i due vettori sta infatti ad indicare la similarità fra questi

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos(\theta)$$

con sigma che e' l'angolo tra a e b . Possiamo trovare questo angolo così:

$$\cos \theta = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|} = \frac{\mathbf{x} \cdot \mathbf{y}}{\sqrt{(\mathbf{x} \cdot \mathbf{x})(\mathbf{y} \cdot \mathbf{y})}}$$

infine sostituiamo il dot product con una funzione kernel

$$\cos(\theta) = \frac{K(x, y)}{\sqrt{K(x, x) * K(y, y)}}$$

Summary:

Property	Euclidean Distance $\ \mathbf{a}-\mathbf{b}\ $	Cosine Similarity $\cos(\theta)$
Translation Independence	yes	no
Independence on Vectors Length	yes	yes

Infine possiamo usare la Disk dentro l'algoritmo di kmeans.

Algorithm Kernel-KMeans(D, K) – K -means clustering using kernelised distance Dis_κ .

Input : data $D \subseteq \mathcal{X}$; number of clusters $K \in \mathbb{N}$.
Output : K -fold partition $D_1 \uplus \dots \uplus D_K = D$.

1 randomly initialise K clusters D_1, \dots, D_K ; this is an originality w.r.t.
the other similar algorithms
2 **repeat**
3 assign each $\mathbf{x} \in D$ to $\arg \min_j \frac{1}{|D_j|} \sum_{\mathbf{y} \in D_j} \text{Dis}_\kappa(\mathbf{x}, \mathbf{y})$;
4 **for** $j = 1$ to K **do**
5 $D_j \leftarrow \{\mathbf{x} \in D \mid \mathbf{x} \text{ assigned to cluster } j\}$;
6 **end**
7 **until** no change in D_1, \dots, D_K ;
8 **return** D_1, \dots, D_K ;

8 Probabilistic Models

Intro

Teorema di Bayes

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

The diagram illustrates the components of the posterior probability formula:

- posterior** (probability of the class, given the observed data)
- likelihood** (probability of the observed features, in examples of a given class)
- prior** (a priori probability of the class) without knowledge of the features
- a priori probability of the observed features**

Dotted arrows point from the text labels to their corresponding terms in the formula. The term $P(\mathbf{X})$ in the denominator is crossed out with a large black X.

For a certain observed instance, this factor is constant, and it does not influence the choice of the class Y.

PROBABILITÀ A PRIORI: $p(y)$

PROBABILITÀ A POSTERIORI: si scrive $p(y|x)$ la probabilità di y dato x

LIKELIHOOD: $p(x|y)$ cioè la distribuzione di x dato y

Modelli Discriminativi

Modella la probabilità a posteriori

tengono conto della distribuzione dei dati e apprendono la classe corrispondente tramite una decision boundary (come ad esempio SVM). Usati per classificazione.

Modelli Generativi:

Modella la probabilità congiunta di X e Y $\rightarrow P(X,Y)$

In alternativa modellano la likelihood, dato che $P(X,Y) = P(X|Y)P(Y)$ con $P(Y)$ facilmente ricavabile. Si chiamano generativi perché generalmente vengono usati per generare dati.

Per classificare i punti si basano sulla probabilità e la likelihood. Se ad esempio si hanno due classi C1 e C2 dato x per classificare il modello calcola la probabilità($x|C1$) e ($x|C2$) e quella maggiore vince.

Bayes Ottimo aka Distribuzione perfetta

Un classificatore è detto bayas ottimo se e' sempre capace di assegnare il valore della classe da predire che renderebbe massima la probabilità a posteriori $P(Y|X)$ una volta osservate le feature dell'esempio in questione. $\underset{y}{\operatorname{argmax}} P^*(Y = y|X = x)$

Distribuzione normale

Un tipo di distribuzione dove esattamente il 68% (+ scarto) degli esempi sta compreso tra mu-sigma e mu+sigma.

mu = centroide

sigma = varianza

Mixture Models

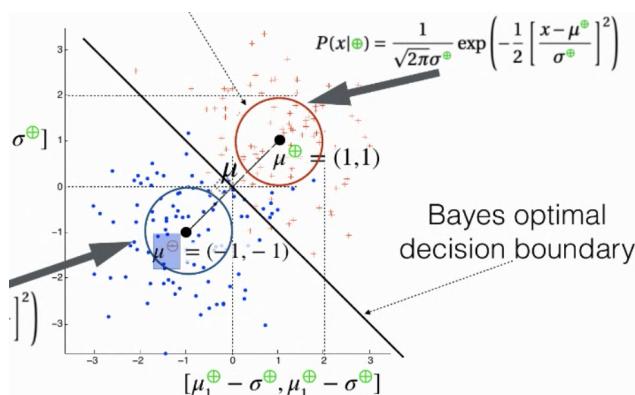
Ogni classe ha la sua distribuzione.

Quando ho un modello per ogni classe (ogni modello ha i suoi mu e sigma). Dato un esempio calcolo la likelihood per ogni classe, la probabilità maggiore per una certa classe indica maggiore confidenza. In altre parole prendo in considerazione la classe per la quale un esempio ha più possibilità di stare in quella distribuzione. Se ho due classi posso usare la formula del likelihood ratio.

LIKELIHOOD RATIO: (>1: classe positiva, <1 classe negativa, 1 incertezza totale)

$$LR(x) = \frac{P(x|\oplus)}{P(x|\ominus)}$$

Se supponiamo che i sigma delle classi siano uguali, e' possibile semplificare il calcolo del LR. Quindi quando i dati sono distribuiti normale e le varianze sono uguali, considerare il likelihood ratio = 1, equivale esattamente a costruire un decision boundary (punto di maggiore incertezza) da un classificatore di base lineare → Bayes Ottimo



Distribuzione Bivariata

Quando gli esempi hanno solo 2 feature (prima consideravo solo una feature, caso univariato). Quindi ora ho due centroidi e due sigma, uno per feature. Poi devo considerare anche la classi + e -, quindi avrò 4 centroidi e 4 sigma. In questo caso il bayes ottimi si ha quando tutte le coppie di sigma di una feature sono uguali per entrambi le classi.

Distribuzione Normale Multivariata

Quando lavoro su d features. Nel caso in cui le matrici di covarianza (la matrice di covarianza contiene informazioni sulla distribuzione) per le due classi siano uguali, siamo nel caso del byas ottimo, dove con $LR=1$ ho un iperpiano equidistante dai due centroidi delle due classi. Questo si può vedere come un classificatore lineare in quanto l'iperpiano è una retta.

NB: Non sempre siamo nel caso di bias ottimo se le matrici di covarianza sono uguali. Un'altra condizione è che le features non devono essere correlate(forse intende indipendenti?) tra di loro.

La multivariata essenzialmente traduce le distanze in probabilità usando la distanza di mahalanobis.

Maximum Likelihood Estimation

Suppose we want to estimate the mean μ of a multivariate Gaussian distribution with given covariance matrix from a set of data points X . The principle of maximum-likelihood estimation states that we should find the value of μ that maximises the joint likelihood of X . Assuming that the elements of X were independently sampled, the joint likelihood decomposes into a product over the individual data points in X . We thus find that the maximum-likelihood estimate of the mean of a multivariate distribution is the point that minimises the total squared Mahalanobis distance to all points in X .

Regressione

Derivare rispetto ai parametri della retta di regressione il MSE equivale a derivare sugli stessi parametri il log della likelihood

Si può dimostrare che la scarto degli errori sarà sempre uguale na^2 perché nella distribuzione iniziale abbiamo assunto un rumore di varianza pari a a^2 , con n come numero di esempi.

Probabilità Con Dati Categorici

Variabile Random

Può essere visto come un vettore dove in ogni posizione c'è presente la probabilità che un certo evento possa accadere. La somma delle probabilità fa 1.

Bernoulli trial

probabilità di un evento di accadere, si usa una serie di bernoulli trials per lo stesso evento in più situazioni. Ad es testa o croce nel lancio della moneta.

Variabile di Bernoulli

distribuzione probabilistica che modella due possibili risultati:

$$\begin{cases} \theta = P(X = 1) \\ 1 - \theta = P(X = 0) \end{cases} \quad \begin{cases} \mathbf{E}(X) = \theta \ (*) \\ \mathbf{E}[(X - \mathbf{E}(X))^2] = \theta(1 - \theta) \end{cases}$$

P(X = i) = $\theta^i(1 - \theta)^{1-i}$

with $i = 0, 1$

Binomiale

distribuzione probabilistica che modella una serie lunga n di variabili di bernoulli. È la somma delle distribuzioni delle variabili di bernoulli considerate in maniera indipendente. Tipo quante volte esce testa lanciando la moneta n volte.

$$P(S = k) = \binom{n}{k} \theta^k (1 - \theta)^{(n-k)}$$

$$\begin{cases} \mathbf{E}(S) = n\theta \\ \mathbf{E}[(S - \mathbf{E}(S))^2] = n\theta(1 - \theta) \end{cases}$$

Bernoulli multivariate (Modello 1)

Una bernoulli multivariata e' un modello che stima la presenza di feature in un documento, dove la probabilità di presenza di una parola in un documento e' rappresentata da dal modello multivariato come un vettore dove in ogni posizione rappresenta la parola i-esima.

Rappresentiamo un documento con un bit vector $X = \{X_1, \dots, X_d\}$ dove $X_i = \{0, 1\}$ e rappresenta la presenza della feature categorica i nel documento. Questi vettori vengono usati per stimare la probabilità delle features in un documento.

Bernoulli multinomiale (Modello 2)

E' un modello che stima la frequenze di feature in un documento. Anche qui il modello e' rappresentato come un vettore dove in ogni posizione si rappresenta la parola i-esima.

Rappresentiamo un documento con un count vector $X = \{X_1, \dots, X_d\}$ dove $X_i = \{x_1, \dots, x_d\}$ rappresenta la frequenza della feature categorica i nel documento. Quindi al posto di verificare semplicemente se c'e o no una certa feature, conto le occorrenze.

$$P(X = (x_1, \dots, x_d)) = \frac{n!}{x_1! \dots x_d!} \theta_1^{x_1} \dots \theta_d^{x_d}$$

with $\sum_{i=1}^{i=d} x_i = n$

with $k=x_1$ and $d=2$ it is $\binom{n}{k}$

Nella multinomiale calcoliamo la probabilità che si verifichi questo vettore di conteggi x_1, \dots, x_d . Cioè calcoliamo la probabilità che escano queste d feature categoriche in tutte le possibili posizioni di un documento composto da n parole.

Categorical variables or features (also called discrete or nominal) are ubiquitous in machine learning.

- ☞ Perhaps the most common form of the Bernoulli distribution models whether or not a word occurs in a document. That is, for the i -th word in our vocabulary we have a random variable X_i governed by a Bernoulli distribution. The joint distribution over the *bit vector* $X = (X_1, \dots, X_k)$ is called a *multivariate Bernoulli distribution*.
- ☞ Variables with more than two outcomes are also common: for example, every word position in an e-mail corresponds to a categorical variable with k outcomes, where k is the size of the vocabulary. The *multinomial distribution* manifests itself as a *count vector*: a histogram of the number of occurrences of all vocabulary words in a document. This establishes an alternative way of modelling text documents that allows the number of occurrences of a word to influence the classification of a document.

Multivariato vs Multinomiale

Vocabulary words:
 $d_i: <1, 0, 0, 1, 0>$

multivariato

Vocabulary words:
 $d_i: <3, 0, 5, 2, 0>$

multinomiale

Questi due modelli di bernoulli assumo indipendenza delle variabili → naive baies.

Naive bias

Assumi che le features siano indipendenti per semplificare il modello altrimenti c'è da spararsi → bam

Multivariato

Si creano due modelli, uno per le classi negative e uno per le classi positive. Ovvero le probabilità che una parola sia presente o meno in un documento:

The models for the classes are vectors of parameters θ_i :

$$\theta^+ = (\theta_a^+, \theta_b^+, \theta_c^+) = (0.5, 0.67, 0.33)$$

$$\theta^- = (\theta_a^-, \theta_b^-, \theta_c^-) = (0.67, 0.33, 0.33)$$

se vogliamo predire la classe di un documento che contiene le prime due parole ma non la terza, facciamo:

$$\operatorname{argmax}_y P(\mathbf{x} = (1,1,0) | Y = y)$$

$$P(\mathbf{x} = (1,1,0) | Y = \oplus) = \theta_a^+ \cdot \theta_b^+ \cdot (1 - \theta_c^+) = 0.5 * 0.67 * 0.66 = 0.22$$

$$P(\mathbf{x} = (1,1,0) | Y = \ominus) = \theta_a^- \cdot \theta_b^- \cdot (1 - \theta_c^-) = 0.67 * 0.33 * 0.66 = 0.14$$

In questo esempio vince la classe +. Questo funziona se le classi + e - sono equiprobabili. Se non lo sono, aggiungo alla formula di prima una moltiplicazione per LR.

Multinomiale

Il discorso e' simile. n è il numero di features

$$\operatorname{argmax}_y P(\mathbf{x} = (3,1,0) | Y = y)$$

$$P(\mathbf{x} = (3,1,0) | Y = \oplus) = n! \cdot \frac{\theta_{a+}^3}{3!} \cdot \frac{\theta_{b+}^1}{1!} \cdot \frac{\theta_{c+}^0}{0!} \quad P(\mathbf{x}|\oplus) = 4! \frac{0.3^3}{3!} \frac{0.5^1}{1!} \frac{0.2^0}{0!} = 0.054$$

$$P(\mathbf{x} = (3,1,0) | Y = \ominus) = n! \cdot \frac{\theta_{a-}^3}{3!} \cdot \frac{\theta_{b-}^1}{1!} \cdot \frac{\theta_{c-}^0}{0!} \quad P(\mathbf{x}|\ominus) = 4! \frac{0.6^3}{3!} \frac{0.2^1}{1!} \frac{0.2^0}{0!} = 0.1728$$

Recalibrated likelihood

Visto che assumere che le variabili siano indipendenti fa schifo a livello della precisione, cerchiamo di fare un po' meno schifo con questa roba.

Assegna un peso, in base alle classi, alla probabilità per ricalibrare il risultato. Questo peso può essere imparato tramite training.

recalibrated likelihood – predict $\operatorname{argmax}_y w_y P(X = x | Y = y)$

Discriminative Learning

Logistic Regression

Si chiama logistic regression ma in verità fa classificazione creando un boundary (come un modello lineare). E' molto simile alla linear regression, ovvero cerco una retta che mi fitti nel migliore dei modi i dati, però mentre di solito la linear regression si usa per regressione, la logistic regression si usa per classificazione. E come dice il nome, la logistic regression introduce una funzione logistica (come la sigmoide)

Il problema viene modellato come la probabilità a posteriori, ovvero $P(Y_i | X_i)$. Dato che cerchiamo una retta che fitti i dati, abbiamo la stessa equazione come un modello lineare:

$$P(y_i | \mathbf{x}_i) \stackrel{?}{=} \beta_0 + \beta_1 x_{i1} + \dots + \beta_d x_{id}$$

Dobbiamo capire come associare questa equazione alla soluzione del nostro problema, ovvero $P(y|x)$.

Per prima cosa il risultato di quella equazione spazia tra $+\infty$ e $-\infty$ e noi vogliamo stare nel range $[0,1]$, perché d'altronde stiamo parlando di modelli probabilistici e vogliamo una probabilità. Quindi applico una funzione sigmoide che fa al caso nostro:

$$P(y_i | \mathbf{x}_i) = \frac{\exp^{\beta_0 + \beta_1 x_{i1} + \dots + \beta_d x_{id}}}{1 + \exp^{\beta_0 + \beta_1 x_{i1} + \dots + \beta_d x_{id}}}$$

Ora il risultato sta tra 0 e 1.

Il classificatore ora per predire se classe positiva o negativa farà:

$$Odds(\mathbf{x}_i) = \frac{P(y_i | \mathbf{x}_i)}{1 - P(y_i | \mathbf{x}_i)}$$

Ovvero la parte sopra come definita prima, e la parte sotto che calcola la probabilità della classe opposta (siamo sempre e solo in un caso binario ovviamente). Quindi se > 0 avrò la classe positiva, negativa il contrario.

Dato che sia numeratore che denominatore sono due funzioni sigmoide con quindi diversi esponenti in gioco, può essere utile semplificare la formula. Abbiamo infatti che:

$$Odds(\mathbf{x}_i) = \frac{P(y_i | \mathbf{x}_i)}{1 - P(y_i | \mathbf{x}_i)} = \exp^{\beta_0 + \beta_1 x_{i1} + \dots + \beta_d x_{id}}$$

Riduciamo quindi il nostro problema ad un esponenziale. Inoltre la formula può essere vista anche come un logaritmo:

$$\ln(Odds(\mathbf{x}_i)) = \beta_0 + \beta_1 x_{i1} + \dots + \beta_d x_{id}$$

Quindi rimane che se il risultato finale è $> 0 \rightarrow$ classe positiva. Negativa altrimenti.

Nel caso le variabili siano categoriche e non numeriche, e quindi non posso fare una semplice equazione lineare, risolvo il problema calcolando la probabilità che una variabile assuma un certo valore categorico e poi calcolo normale.

Logistic regression è un caso particolare dei generalized linear models

Funzione Obiettivo

Ora bisogna definire il nostro problema, ovvero cosa dobbiamo massimizzare o minimizzare. Introduciamo la Conditional Log Likelihood (LCL):

$$-\text{LCL}(\mathbf{w}, t) = \sum_i \sum_{\mathbf{x}_i \in Tr} (y_i - \hat{p}(\mathbf{x}_i)) \cdot \text{bec}$$

Abbiamo quindi che il nostro problema è definito come

$$\underset{\mathbf{w}, t}{\operatorname{argmax}} \mathbf{LCL}(\mathbf{w}, t)$$

Con w e t che sono i nostri bellissimi Beta sopra.

The Minimum Description Length Principle

$$\text{Cost(Model, Data)} = \text{Cost(Model)} + \text{Cost(Data|Model)}$$

The best model would be that model whose information amount is minimum (minimizes Cost(Model)) which is in accordance with Occam's razor principle about the preference towards simple models.

In addition, the best model should also minimise the error of the model on the data which is represented by: Cost(Data|Model). It is the amount of information about the data that we need to represent explicitly, separately by the model.

Expectation Maximization

Clustering probabilistico. Basato su mixtures di modelli, dove ogni cluster ha la sua distribuzione. Se sapessimo a priori le associazioni cluster-esempi o le distribuzioni dei cluster, sarebbe molto facile calcolare l'altra cosa. Dato che non abbiamo né uno né l'altro, l'algoritmo di EM cerca di risolvere al problema. Calcola le associazioni degli esempi ai cluster e le distribuzioni di questi. L'algoritmo si basa su due cicli:

1. Fase expectation: fissa i parametri di distribuzione dei cluster e calcola la probabilità che gli esempi cadano in ciascun cluster.
2. Fase maximization: fissa la popolazione dei cluster e calcola la nuova distribuzione dei cluster dati gli aggiornamenti dello step 1
3. Si cerca di massimizzare la likelihood dei training data dati i cluster. I cicli 1 e 2 si alternano finché la likelihood non diventa satura, ovvero non cambia più molto ad ogni iterazione (converge).

Se non hai capito come funziona, pensa a k-means, è letteralmente uguale il ragionamento logico.

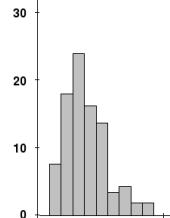
9 Features

Le features variano in base alla loro valorizzazione, abbiamo 3 tipi principalmente:

- categorical: categorie senza ordinamento (es tipi di cibo: frutta, verdura, carne)
 - boolean: un sottoinsieme del categorial dove ho solo true e false
- ordinal: categorie ordinate (es le taglie: s, m, l, xl)
- quantitative/continue: mapping su numeri reali

I tipi di computazione possibile sulle features possono essere raggruppate in 3 categorie

1. statistics of central tendency
 - a. media: valore medio
 - b. moda: valore più frequente

- c. mediana: valore a metà'
2. statistics of dispersion
 - a. variance: varianza
 - b. standard deviation: indica la dispersione in distribuzione
 - c. range: min - max
 - d. percentili: the p-th percentile is the value such that p% of the instances fall below it.
 3. shape statistics
 - a. skewness: A positive value of skewness means that the distribution is right-skewed, which means that the right tail is longer than the left tail (come nella foto sotto). Negative skewness indicates the opposite, left-skewed case
- A histogram illustrating a right-skewed distribution. The x-axis represents categories or bins, and the y-axis represents frequency, ranging from 0 to 30. The distribution is skewed to the right, with the highest frequency in the first bin (approximately 25) and a long tail extending towards higher values.
- b. kurtosis: Positive excess kurtosis means that the distribution is more sharply peaked than the normal distribution

Summary:

Kind	Order	Scale	Tendency	Dispersion	Shape
Categorical	x	x	mode	n/a	n/a
Ordinal	✓	x	median	quantiles	n/a
Quantitative	✓	✓	mean	range, interquartile range, variance, standard deviation	skewness, kurtosis

Feature Transformation

↓ to, from →	Quantitative	Ordinal	Categorical	Boolean
Quantitative	normalisation	calibration	calibration	calibration
Ordinal	discretisation	ordering	ordering	ordering
Categorical	discretisation	unordering	grouping	
Boolean	thresholding	thresholding	binarisation	

Discretizzazione e Thresholding

Il thresholding (quantitativo, ordinale -> boolean) esegue una discretizzazione del dominio dividendolo in intervalli

Unsupervised Discretization

(quantitativo -> ordinale, categorico)

- Equal-Width: divide in intervalli di larghezza uguale
- Equal-Depth: divide gli intervalli in ugual frequenza
- K-means: dividendo i gruppi (simili)

Supervised - Discretisation by Recursive partitioning

Scelgo la feature su cui voglio fare discretizzazione. Ordine gli esempi in base al valore di quella feature. Inizialmente considero tutti gli esempi in un unico set. Inizio ricorsivamente a partizionare il set utilizzando come metrica l'entropia finché non entra in gioco lo stopping criterion.

Algorithm $\text{RecPart}(S, f, Q)$ – supervised discretisation by means of recursive partitioning.

Input : set of labelled instances S ranked on feature values $f(x)$; scoring function Q .
Output : sequence of thresholds t_1, \dots, t_{k-1} .

1 **if** stopping criterion applies **then return** \emptyset ;
2 ;
3 Split S into S_l and S_r using threshold t that optimises Q ;
4 $T_l = \text{RecPart}(S_l, f, Q)$;
5 $T_r = \text{RecPart}(S_r, f, Q)$;
6 **return** $T_l \cup \{t\} \cup T_r$;

Supervised - Discretisation by Agglomerative merging

Come prima, ordino le istanze in base al valore della feature in questione. Vengono partizionati gli esempi raggruppando esempi vicini della stessa classe. Inizio poi a mergere set vicini (chiamati bins) sulla base di una metrica Q . Mergo finché non incontro lo stopping criterion.

Si toglie la suddivisione tra due insiemi vicini (uno di fianco all'altro) se la frequenza calcolata nella tabella di contingenza creata con i valori di x^2 è molto vicina a quella aspettata.

Algorithm AggloMerge(S, f, Q) – supervised discretisation by means of agglomerative merging.

Input : set of labelled instances S ranked on feature values $f(x)$; scoring function Q .

Output : sequence of thresholds.

- 1 initialise bins to data points with the same scores;
- 2 merge consecutive pure bins ; // optional optimisation
- 3 **repeat**
- 4 evaluate Q on consecutive bin pairs;
- 5 merge the pairs with best Q (unless they invoke the stopping criterion);
- 6 **until** no further merges are possible;
- 7 **return** thresholds between bins;

Normalizzazione

- **normalizzazione statistica:** centrare nella media e dividendo per la deviazione standard -> $\text{new_value} = (x - \mu) / \sigma$
- **normalizzazione min-max:** ridimensionare i valori in un range. I numeri inferiori di min sono posti a min ed i numeri maggiori di max sono posti a max

Calibrazione

ordinali, categoriche o booleane -> quantitative.

In pratica calcola sta roba: la probabilità a posteriori dato il valore della feature.

$$F^c(x) = P(\oplus|v)$$

dove v e' il valore della feature categorica. Questo funziona se le distribuzioni delle classi e' uniforme. Se non lo e', devo introdurre la probabilità a priori ($P(Y)$) e moltiplicarla da qualche parte nella formula di prima(non so bene dove)

Feature Selection

Con questo metodo si selezionano solamente le features che ci servono davvero.

Vantaggi:

- evito overfitting
- learning più veloce
- ridurre le dimensioni del problema

Ci sono due approcci alle features selection: filter e wrapper

1. filter selection: si selezionano le feature sulla base di uno score
 - a. relief: calcola la similarità dei valori di una feature sulle istanze. Lo score di una feature sarà più alto se valori di questa feature rispetto ad altri esempi della stessa classe differiscono meno rispetto ad esempi della classe opposta. L'algoritmo prende a caso un esempio x , calcola *hit* e *miss* (esempi più vicini della stessa e opposta classe), poi per ogni feature calcola:

$$\text{score} = \text{score} - |x - \text{hit}| + |x - \text{miss}|$$
. Quindi se x -miss è più alto rispetto a x -hit, lo score sale.
 - i. un problema è che non tiene conto della dipendenza fra feature diverse, cioè presuppone che siano sempre indipendenti
2. wrapper selection: calcola l'utilità delle feature, se non sono utili le toglie.
 - a. forward: parte con una singola feature e ne aggiunge pian piano finché le performance del modello smettono di migliorare
 - b. backward: parte con tutte le feature e le toglie finché le performance del modello smettono di migliorare

PCA

Meccanismo per fare sia feature construction e feature selection. Riduce la dimensionalità dei dati.

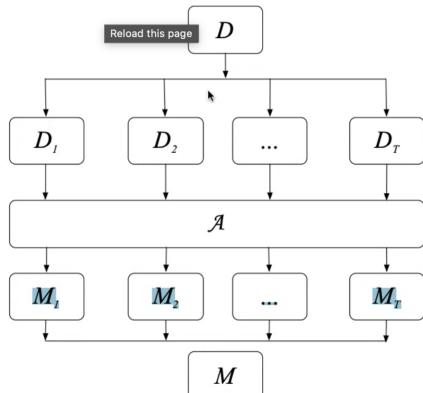
10 Ensemble learning (esposto)

Migliorare le prestazioni di classificazione combinando modelli differenti basati su subset del dataset differenti.

A → Algoritmo di apprendimento

D → Dataset, Di è un subset

M → Modello, Mi è un modello per il Di
subset



Per combinare i risultati di classificazione di ogni singolo modello, si calcola una funzione sulla sommatoria pesata di tutti i risultati. In questo modo, modelli più confidenti avranno più voce in capitolo nella decisione finale(w_i più grande). Di solito i pesi sono maggiori di 0, ma possono anche essere sotto.

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

Bagging

Algoritmo per fare ensemble.

Si basa sul Bootstrap Replicate: genera T dataset lunghi $|D|$ partendo dal dataset D . Per ogni D_t dataset inserisce a caso gli esempi provenienti da D (duplicati allowed). Poi per ogni D_t viene creato un modello M_t usando l'algoritmo A .

Il bagging funziona bene perché nei dataset D_t viene mantenuta la distribuzione originale dei dati. Infatti se nel dataset D originale ho esempi duplicati, questi avranno più possibilità di essere pescati a random per essere inseriti nei dataset D_t . Questo consente anche di mantenere i dataset D_t tutti diversi tra loro. In media, un dataset D_t contiene al suo interno il 63% di esempi unici, non replicati.

Algorithm Bagging(D, T, \mathcal{A}) – train an ensemble of models from bootstrap samples.

Input : data set D ; ensemble size T ; learning algorithm \mathcal{A} .

Output : ensemble of models whose predictions are to be combined by voting or averaging.

```

for  $t = 1$  to  $T$  do
    build a bootstrap sample  $D_t$  from  $D$  by sampling  $|D|$  data points with replacement;
    run  $\mathcal{A}$  on  $D_t$  to produce a model  $M_t$ ;
end
return  $\{M_t | 1 \leq t \leq T\}$ 

```

Random Forest

Una tipologia di bagging che usa alberi non potati come weak learner. Gli alberi saranno quindi molto lunghi e improntati a fare overfitting.

Per la creazione degli split dell'albero non si considera più l'intero set F di features e poi prendo la feature migliore per lo split, come in bagging, ma estraggo prima un piccolo sottoinsieme di feature a caso, e poi cerco la migliore. Questo permette di introdurre una componente randomica che rende molto differenti i modelli.

AdaBoost - [LINK](#)

Apprendimento forte: quando dei concetti possono essere appresi con confidenza e accuratezza in un tempo finito.

Apprendimento debole: come il forte ma cade il concetto di concetto appreso in maniera accurata. In altre parole faccio poco meglio di una stima random.

Boosting Ipotesi: se si può raggiungere un apprendimento forte dopo aver sostenuto un apprendimento debole su tutti i concetti. → E' stato dimostrato che dati distribuiti in maniera uniforme non sempre possono essere allenati in maniera forte partendo da un apprendimento debole. → Ma e' stato dimostrato che se i dati non sono distribuiti in maniera uniforme, allora e' possibile apprendere in maniera forte partendo da un apprendimento debole.

Boosting problem: quello di trasformazione weak learner in strong learner

AdaBoost quindi cerca di risolvere questo problema.

L'idea del boosting e' quella di associare dei pesi a tutti gli esempi e cambiarli dinamicamente durante l'esecuzione dell'algoritmo. Inizialmente i pesi sono tutti uguali (e sommano a 1), successivamente se un modello apprende in maniera errata un esempio, il peso dell'esempio viene aumentato. Abbassato invece se la predizione e' corretta. Questo costringe i modelli successivi a focalizzarsi sugli esempi classificati erroneamente.

Gli alberi di adaboost sono formati da un solo split.

Le due tecniche più importanti per tenere in considerazione i pesi per lo split sono:

- weighted gini index: e' come il gini index classico, solo che durante la conta si tiene in considerazione il peso. Esempi con peso più alto avranno maggiore contributo nella scelta dello split.
- dataset resampling: In pratica viene generato un nuovo dataset prendendo a caso gli esempi dal dataset originale. Però gli esempi con pesi più alti hanno più probabilità di uscire, risultando quindi con maggiore probabilità nell'avere duplicati nel nuovo dataset. Una volta creato il nuovo dataset i pesi vengono resettati e portati in maniera uniforme. In questo modo ho un dataset nuovo pesato sulla base dei pesi precedenti.

```

def adaboost ( $D = (x_1, y_1), \dots, (x_n, y_n)$ ,  $\mathcal{A}$ )
     $\mathbf{w}^1 = \left[ \frac{1}{|D|}, \dots, \frac{1}{|D|} \right]$ 

    for  $t \in \{1 \dots T\}$  do
         $m_t = \mathcal{A}(D, \mathbf{w}^t)$ 
         $\epsilon_t = \sum_{i=1}^n w_i^t I[y_i \neq m_t(x_i)]$ 
         $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$ 

        for  $i \in \{1 \dots n\}$  do
             $w_i^{t+1} = w_i^t \exp(-\alpha_t y_i m_t(x_i))$ 
        end
         $\mathbf{w}^{t+1} = \text{normalize}(\mathbf{w}^{t+1})$  WV
    end

    return  $M(x) = \text{sign}\left(\sum_{t=1}^T \alpha_t m_t(x)\right)$ 
end

```

Riga per riga:

- si inizializzano i pesi per ogni esempio in modo che siano tutti uguali e che sommino a 1
- viene generato l'albero(che di norma è un albero "semplice") con solo uno split su una features e due foglie (una per classe + e l'altra per -). L'albero è un decision tree e viene quindi generato come abbiamo già visto con la meo
- per calcolare l'errore totale si sommano i pesi di tutti gli esempi del dataset misclassificati (la variabile ϵ_t). Errore tot è zero se quell'albero è perfetto, 1 se fa schifo.
- viene calcolato l'amount of say di quell'albero, cioè quanto varrà la sua classificazione quando verrà fatta rispetto a tutti gli altri alberi generati. Questo valore non ha range. Più l'albero è preciso, più alto e' l'amount of say. Il contrario se l'albero fa tanti errori.
- i pesi vengono poi aggiornati tutti in base all'amount of say calcolato prima. Gli esempi misclassificati vedranno alzarsi il loro peso, mentre quelli classificati correttamente avranno il peso abbassato.
- vengono normalizzati i pesi per farli sommare a 1 (divido il peso per il numero di pesi $w/|w|$) Per tenere in considerazione lo split da usare per generare il prossimo albero di decisione si usa weighted gini index o dataset resampling prima descritti

Perché funziona

Ensemble learning funziona perché riducono bias e varianza. Ha principalmente 3 benefici

1. Statistical: Approssima il bias ottimo. Questo perché nel bias ottimo abbiamo una votazione pesata delle ipotesi, dove il peso è più alto se se l'ipotesi è più consistente. Di conseguenza in ensemble, combinando più classificatori ci avviciniamo a questo bias ottimo.
2. Computational: Permette di evitare minimi locali. Se un albero finisce in un minimo locale non e' un problema perche tanto io ne avro' altri che possono rimediare.
3. Representational: Se abbiamo che un albero di decisione da solo non potrà mai raggiungere l'ottimo, combinando piu classificatori riusciamo ad avvicinarci a questo ottimo.
- Bagging: riduce principalmente la varianza: perche faccio la media di tanti classificatori. Nel bagging l'errore diminuisce con l'aumentare delle iterazioni perche

- AdaBoost: riduce principalmente il bias, ma si comporta bene anche nel ridurre varianza

Decomposizione Bias Variance

Possiamo decomporre l'errore di un modello in due componenti: $E = \text{bias} + \text{variance}$

- bias: errore nel non riuscire a predire perfettamente → ovvero la differenza tra quello che predicono ed il ground truth
- variance: errore dato dal predire tutte le fluttuazioni dei miei dati, anche il rumore → il modello cerca di imparare queste fluttuazioni che non sono rilevanti e quindi si riducono le performance di generalizzazione

11 Machine Learning Experiments

Machine learning experiments indicano degli esperimenti atti a provare alcune assunzioni sui nostri algoritmi di apprendimento. Queste assunzioni possono riguardare ad esempio la convergenza del nostro modello.

Cosa misurare

- Accuracy: per valutare che la distribuzione del test set sia rappresentativa per il nostro contesto.

$$\circ \quad accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- Average Recall: per valutare se le distribuzioni delle classi sia uniforme.

$$\circ \quad average\ recall = \frac{tpr}{2} + \frac{tnr}{2}$$

- Precision and Recall: sposta il focus dall'accuratezza del modello alle performance ignorando la classe negativa.

$$\circ \quad precision = \frac{TP}{TP + FP}$$

$$\circ \quad recall = \frac{TP}{TP + FN}$$

- Predictive Positive Rate (PPR) and Area Under the Curve (AUC): misure utilizzate nel ranking

$$\circ \quad predicted\ positive\ rate = \frac{TP + FP}{TP + TN + FP + FN}$$

Come misurare

Fare k volte le misure in maniera indipendente e fare la media per ridurre al più possibile l'errore di misurazione. Nel caso di un classificatore, si possono creare k test set e fare le misurazioni su ogni test set. Se ho pochi dati posso usare la cross validation. Alla fine delle k misurazioni posso calcolare l'average e la standard deviation.

Come interpretare

Per interpretare i dati raccolti da una cross validation o da qualsiasi altra misura che involve k iterazioni, è utile calcolare la p-value.

P-Value

Quando si effettua un test d'ipotesi si fissa un'ipotesi nulla e un valore soglia α (per convenzione di solito 0.05, ovvero 5%) che indica il livello di confidenza del test. Calcolato il p-value relativo ai dati osservati, ovvero la probabilità di ottenere quel dato, è possibile comportarsi come segue:

- $p > \alpha$: la null ipotesi NON è rifiutata
- $p \leq \alpha$: la nulla ipotesi è rifiutata

In pratica e' un ipotesi in cui dico che non c'e' differenza tra due popolazioni. Se la rifiuto vuol dire che C'E' differenza. Se NON rifiuto l'ipotesi vuol dire che NON C'È' differenza. Il p-value invece mi ritorna la probabilità di ottenere una certa misura data la null ipotesi.