
Appunti di Apprendimento Automatico

A.A. 2021-2022

Matteo Brunello

Indice

1	Che cos'è il Machine Learning	6
1.1	Esempio SpamAssassin	7
1.2	Modelli	8
2	Machine Learning Tasks	8
2.1	Classificazione	9
2.2	Scoring e Ranking	10
2.2.1	Ranking: Verificare e visualizzare le performance	11
2.3	Stima delle probabilità delle classi	12
2.4	Probabilità Empiriche	13
3	Oltre la classificazione binaria	13
3.1	Estendere la classificazione binaria a multiclasse	13
3.2	Regressione	14
3.3	Predictive vs Descriptive learning	15
3.3.1	Clustering	15
4	Concept Learning	16
4.1	Ripasso Logica	16
4.2	Lo spazio delle ipotesi	16
4.3	Ordinamento delle ipotesi	17
4.4	Trovare massima ipotesi più specifica: L'algoritmo Find-S	17
4.5	L'algoritmo Candidate-Elimination e il Version Space	19
4.6	Inductive Bias	21
5	Modelli ad Albero	23
5.1	Feature tree	24
5.2	L'algoritmo GrowTree	24
5.3	Decision Trees	25
5.3.1	Entropia	26
5.4	Alberi di decisione e coverage plot	28
5.5	Ranking Trees	28
5.6	Labelling di un feature tree	29
5.7	Trasformazione da feature tree a modello	30
5.8	Semplificazione di alberi	30
5.8.1	Pruning	30
5.8.2	Stima degli errori di generalizzazione	31

5.9	Sensitivita' rispetto alla distribuzione delle classi	31
5.10	Apprendimento di alberi come riduzione della varianza	32
5.10.1	Regression trees	32
5.10.2	Clustering trees	32
5.10.3	Coesione e Separazione di Clusters	33
6	Modelli a Regole	35
6.1	Apprendimento di liste di regole	35
6.2	Apprendere insiemi non ordinati di regole	37
6.2.1	Miopia	38
6.3	Insiemi di regole come rankers	39
6.4	Subgroup Discovery	39
6.5	Mining di regole di associazione	40
6.5.1	Itemsets e proprieta'	41
6.5.2	Regole di associazione	42
7	Modelli Lineari	44
7.1	Least Squares (Metodo dei minimi quadrati)	44
7.1.1	Regolarizzazione	46
7.1.2	Classificazione con LSE	47
7.2	Support Vector Machines	47
7.2.1	Margine Funzionale	48
7.2.2	Margine Geometrico	49
7.2.3	Formulazione Primale	50
7.2.4	Formulazione Duale	50
7.2.5	Ammettere errori nel margine	52
7.3	Kernels	54
8	Modelli basati sulla distanza	57
8.1	Distanze di Minkowski	57
8.2	Distanze di tipo ellittico	58
8.3	Centroidi e Medoidi	59
8.4	Distance Based Classification	60
8.4.1	Nearest Neighbour Classifier	61
8.5	Distance Based Clustering	62
8.5.1	DBSCAN	62
8.5.2	K-Means	64
8.5.3	Silhouettes	65
8.6	Clustering Gerarchico	67

8.7	Dai kernels alle distanze	68
9	Modelli Probabilistici	70
9.1	Apprendimento come riduzione dell'incertezza	71
9.2	Modelli Probabilistici e Significato Geometrico	73
9.3	Connessione 1	73
9.3.1	Connessione 2	75
9.3.2	Connessione 3	75
9.4	Modelli Probabilistici per Dati Categorici	76
9.4.1	Modello Bernoulliano Multivariato	76
9.4.2	Modello Bernoulliano Multinomiale	77
9.5	Naive Bayes	77
9.5.1	Regole di Decisione Probabilistiche	78
9.5.2	Apprendimento di un modello Naive Bayes	79
9.6	Regressione Logistica	80
9.7	Modelli basati sulla Compressione	83
9.8	Expectation Maximisation	84
10	Features	87
10.1	Tipologie di Features	87
10.2	Trasformazioni delle Features	88
10.2.1	Operazioni di discretizzazione	89
10.2.2	Operazioni di Normalizzazione	92
10.2.3	Operazioni di Calibrazione	93
10.3	Discretizzazione per la riduzione del rumore	95
10.3.1	Imputazione	95
10.3.2	Costruzione di features	95
10.4	Selezione di features	96
10.4.1	Approccio Filtro	96
10.4.2	Approccio Wrapper	97
10.5	Principal Component Analysis	98
10.6	Singular Value Decomposition (SVD)	98
11	Ensemble Learning	99
11.1	Bagging	99
11.2	Random Forest	100
11.3	AdaBoost	100
11.3.1	Algoritmo e Spiegazione	102
11.3.2	Apprendimento da dataset pesati	104

11.4	Perche' ensemble learning funziona	105
11.5	Benefici dell'Ensemble Learning	107
12	Esperimenti nel Machine Learning	108
12.1	Cross-Validation	109
12.2	Intervalli di confidenza	110
12.3	Paired <i>t</i> -test	111
12.4	Wilcoxon Test	111
12.5	Friedman Test	112
12.5.1	Post-Hoc Test	112

1 Che cos'è il Machine Learning

- Come prima approssimazione possiamo dare la seguente definizione di machine learning

*Il machine Learning è lo studio sistematico di **algoritmi** e sistemi che **aumentano la loro conoscenza o performance con l'esperienza***

- L'esperienza da cui impara un algoritmo o sistema di machine learning sono i dati etichettati, e le performance si riferisce all'abilità di classificare correttamente un esempio (nel caso dello spam filter)
- I concetti fondamentali che ci servono per capire cos'è il machine learning sono i seguenti:
 - **Tasks:** i problemi che siamo in grado di affrontare con il ML (classificazione, regressione, stima della probabilità, clustering...)
 - **Modelli:** oggetti che vengono costituiti dall'algoritmo di apprendimento che risolvono effettivamente il problema (lineari, alberi di decisione, Naive Bayes, Knn...)
 - **Features:** cioè che ci permette di descrivere il problema. Il problema è descritto mediante un insieme di features

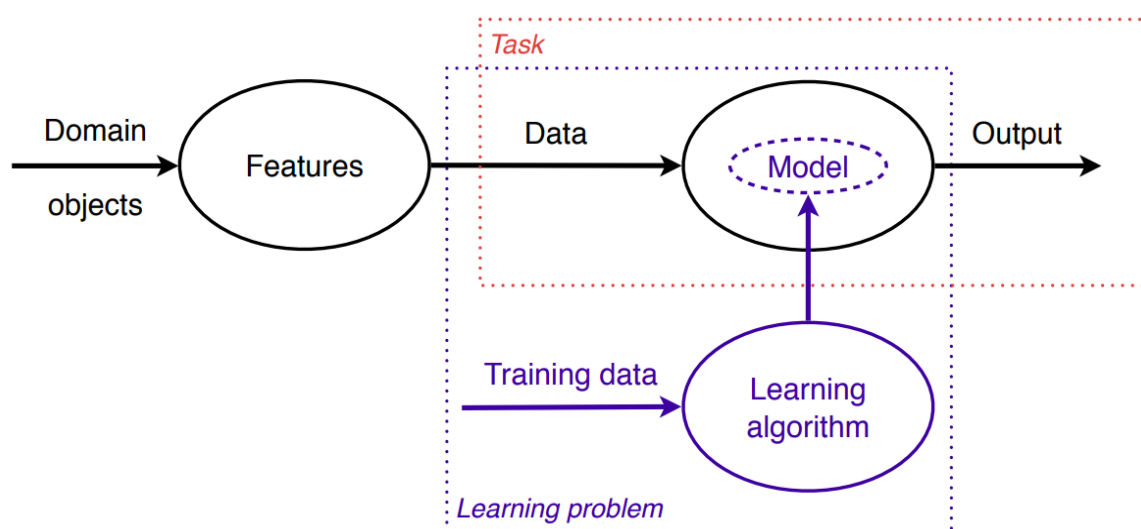


Figura 1: Overview di come viene impiegato il machine learning per la soluzione di un dato task. Per essere risolto, un task (rettangolo rosso) richiede un mapping tra i dati (descritti mediante features) e l'output. Tale mapping è dato dal modello. Ottenere il modello dai dati di train è ciò che costituisce il problema di apprendimento (rettangolo blu)

*Il Machine Learning e' un campo che si occupa di usare le giuste **features**, per costruire il giusto **modello** per svolgere il giusto **task**.*

- I task si suddividono in due categorie principali:
 1. **Task Predittivi**: che concernono la predizione del valore di una variabile target (es. classificazione binaria/multiclasse, regressione, clustering)
 2. **Task Descrittivi**: il cui obiettivo e quello di estrapolare la struttura descrittiva dei dati in modo da generalizzare (es. GAN)
- Per ogni tipologia di task avremmo anche due differenti settings per i modelli corrispondenti. Inoltre, i modelli variano in base alla tipologia di apprendimento che puo' essere impiegata che puo' essere **supervisionata** (cioe' che impiega un dataset di train etichettato) oppure **non supervisionata** (che non necessita di training set)
- La seguente tabella riassume i vari task predittivi/descrittivi in base al setting di apprendimento utilizzato

Apprendimento/Modello	Predittivo	Non Predittivo
Supervisionato	classificazione, regressione	subgroup discovery
Non supervisionato	clustering predittivo	clustering descrittivo, association rule discovery

- Un problema dei modelli predittivi e' l'**overfitting**. Esso accade quando l'algoritmo tenta di adattarsi con troppa aggressivita' al dataset nella fase di learning.
- Per aiutarci a capire meglio, immaginiamo per esempio di preparare un esame solamente usando esami passati. Al primo esame diverso dai facsimile su cui ci si e' preparati non si saprebbe risolvere. Questa e' una forma di overfitting.
- Dal punto di vista formale, l'overfitting avviene quando le performance sul training set sono piu' alte delle performance sul test set. Se ad esempio ho un'accuratezza di un modello del 90% sul training set e un'accuratezza del 60% sul test set, si dice che ho un overfitting del 30%.

1.1 Esempio SpamAssassin

- Si vuole costruire un **classificatore** di email che classifichi le emails tra spam e no. Un problema di classificazione consiste nel dover apporre un'etichetta agli oggetti, in questo caso *spam* o *ham* alle emails.
- SpamAssassin affronta questo problema utilizzando un insieme di regole, in cui ogni regola i ha un determinato peso associato w_i

- Un esempio di regola potrebbe essere quello ad esempio della regola `HTML_IMAGE_RATIO_02`, che si attiva quando il rateo tra testo e immagine e' maggiore del 20%
- Quello che succede e' che per una determinata email si attivano diverse regole dell'insieme
- La somma di tutti i pesi ($S = \sum_i w_i$) delle regole attive determina se la mail e' spam o ham. Nel caso di SpamAssassin, se $S > 5$, allora e' molto probabile che sia spam

1.2 Modelli

- I modelli di apprendimento automatico possono essere classificati secondo il loro approccio (in altri termini, nel modo in cui sono *descritti*):
 - **Geometrici**: usano intuizioni dalla geometria (ad esempio separando iperpiani, trasformazioni lineari ecc..).
 - **Probabilistici**: in cui il processo di apprendimento e' equivalente a ridurre l'incertezza, ed e' modellato attraverso distribuzioni di probabilita'
 - **Logici**: definiti in termini di espressioni logiche

2 Machine Learning Tasks

- Gli oggetti sono chiamate *istanze* nel machine learning
- L'insieme di tutte le possibili istanze e' chiamato *instance space*, denotato con \mathcal{X} . Negli esempi precedenti, \mathcal{X} corrisponde a tutte le possibili emails che possono essere scritte con l'alfabeto latino. L'input space e' descritto in termini di **features**, anche chiamati attributi. Piu' in generale, denotando \mathcal{F}_i l'insieme di valori di una singola feature, abbiamo che $\mathcal{X} = \{\mathcal{F}_1 \times \mathcal{F}_2 \times \dots \times \mathcal{F}_d\}$, per cui ogni istanza e' un vettore *d-dimensionale* contenente i valori delle features.
- Il *label space* e' l'insieme delle etichette assegnabili, denotato con \mathcal{L}
- Un *modello* e' una mappa $\hat{m} : \mathcal{X} \rightarrow \mathcal{Y}$, dove \mathcal{Y} e' l'*output space*.
- Il *training set* **Tr** e' definito come l'insieme di istanze chiamti **esempi** etichettate $(x, l(x))$, dove $l : \mathcal{X} \rightarrow \mathcal{L}$ e' una funzione etichettatrice
- Nella maggior parte dei modelli, $\mathcal{Y} = \mathcal{L}$, per cui quello che si vuole ottenere e' un modello $\hat{l} : \mathcal{X} \rightarrow \mathcal{L}$ che sia una buona approssimazione della funzione di labelling reale l , conosciuta solo tramite le etichette che sono state assegnate ai dati di training.
- In casi in cui $\mathcal{Y} \neq \mathcal{L}$, ad esempio se si volesse un modello che dia in output uno score di likelihood per ogni label, allora $\mathcal{Y} = \mathbb{R}^k$, dove $k = |\mathcal{L}|$
- Per testare il modello si utilizza il cosiddetto *test set* **Te**. Spesso il test set viene denotato con un apice per indicare le istanze che appartengono ad una determinata classe. Ad esempio

$Te^{\oplus} = \{(x, l(x)) \mid x \in Te \wedge l(x) = \oplus\}$ e' l'insieme dei campioni positivi, mentre Te^{\ominus} dei negativi

2.1 Classificazione

- Un task di classificazione e' il task piu' comune nel machine learning. Un classificatore e' una mappa $\hat{c} : \mathcal{X} \rightarrow \mathcal{C}$, dove $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ e' un insieme (solitamente piccolo) di *labels di classe*.
- Far imparare un classificatore significa costruire la funzione \hat{c} che sia un'approssimazione piu' precisa possibile di c , non solo sul training set ma idealmente sull'intero spazio \mathcal{X}
- La classificazione binaria (o *concept learning*) e' quando l'insieme \mathcal{C} e' composto da due classi: *positiva* (denotata con \oplus) e *negativa* (denotata con \ominus)
- Un *feature tree* e' un albero che ha come nodi le features, mentre come foglie il numero complessivo di istanze appartenenti alle classi
- Un feature tree puo' essere convertito in un *decision tree* semplicemente assegnando la classe maggioritaria in ogni foglia.
- La performance dei classificatori puo' essere riassunta nella **tabella di contingenza** o **matrice di confusione**. In questa tabella, ogni riga si riferisce alle classi attuali come registrate nel test set, mentre le colonne si riferiscono alle classi che ha predetto il classificatore.
- Dalla tabella di contingenza e' possibile calcolare diversi indicatori di performance. Uno di questi e' la *precisione (accuracy)*, calcolabile nel modo seguente:

$$acc = \frac{1}{|Te|} \sum_{x \in Te} I[\hat{c}(x) = c(x)]$$

Dove la funzione $I[\cdot]$ e' l'*indicator function* che vale 1 se l'argomento e' *true*, 0 altrimenti

- Possiamo anche definire un'altro indicatore chiamato *true positive rate* nel modo seguente:

$$tpr = \frac{\sum_{x \in Te} I[\hat{c}(x) = c(x) = \oplus]}{\sum_{x \in Te} I[c(x) = \oplus]}$$

A parole: E' il rapporto tra tutti i *true positives* (cioe' i positivi che vengono identificati correttamente) e tutti i *positivi*.

- *tpr* da una stima della probabilita' che un positivo arbitrario sia classificato correttamente dal classificatore, formalmente $P_{\mathcal{X}}(\hat{c}(x) = \oplus | c(x) = \oplus)$
- Analogamente e' definibile il *true negative rate*
- Queste quantita' sono dette anche *sensitivity* e *specificity*, e possono essere viste come stime di accuratezza **per classe**.

- *Coverage plot*: grafico per valutare le performance dei classificatori. Si costruisce mediante 4 ingredienti principali:
 - Numero di positivi totali
 - Numero di negativi totali
 - Numero di veri positivi
 - Numero di falsi positivi
- Gli altri dati possono essere presi facendo il complementare rispetto al totale.
- In un *coverage plot*, i classificatori con la stessa accuratezza stanno sulla stessa retta di coefficiente angolare pari a 1.
- *Normalized plot* (ROC plot) e' un coverage plot ma con gli assi normalizzati (sono divisi per il numero totale, in questo caso *Pos* e *Neg*), quindi tutti i punti sono divisi per la scala (pos e neg)
- In un plot normalizzato una retta con equazione $tpr = fpr + y_0$ non ha lo stesso significato di una stessa linea in un coverage plot. In generale in un plot normalizzato, su una linea con coefficiente angolare pari a 1 (e quindi parallela alla diagonale dal momento che si tratta di una griglia con dimensioni 1x1) stanno tutti i classificatori con lo stesso **average recall**. Analogamente, tutte le rette con coefficiente angolare pari a neg/pos (cioe' pari a $1/cfr$) hanno la stessa accuratezza.
- $avg-rec = (tpr + tnr)/2$
- *Coverage plot*: Utile quando si vuole tenere esplicitamente conto della distribuzione di classi, per esempio quando si sta lavorando con un singolo dataset
- *ROC plot*: Utile quando si vogliono combinare risultati provenienti di diversi datasets, con distribuzioni di classi differenti
- In generale, i classificatori meno efficienti sono quelli che stanno sulla diagonale principale, dal momento che fanno un random guess 50/50. Anche se quelli che stanno sotto hanno piu' negativi che positivi, basterebbe negare il risultato del classificatore per ottenerne uno comunque preciso

2.2 Scoring e Ranking

- Il task di scoring e' simile al task di classificazione. L'idea e' quella di assegnare degli scores ad ogni classe per una determinata istanza.
- Uno *scoring classifier* e' definito formalmente come una mappa $\hat{s} : \mathcal{X} \rightarrow \mathbb{R}^k$. Dove il generico $\hat{s}(x) = (\hat{s}_1(x), \dots, \hat{s}_k(x))$, in cui $\hat{s}_i(x)$ indica il punteggio assegnato alla classe C_i per l'istanza x
- Il task di scoring puo' essere visto anche come un task di classificazione binaria quando il numero di classi e' 2
- Per trasformare un *feature tree* in uno *scoring tree*, si calcola il class ratio delle foglie e se ne considera il *logaritmo*. Il risultato sara' lo score che assegnera' \hat{s} .

- Se consideriamo $c(x)$ come la *true class function*, che ritorna +1 per gli esempi positivi, -1 altrimenti, allora si può definire la funzione **margin**, definita come

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

- Idealmente, si vogliono penalizzare grossi valori negativi di $z(x)$, mentre si vogliono premiare grossi valori positivi.
- **Loss function** ($L : \mathbb{R} \rightarrow [0, \infty)$): Mappa ogni valore di $z(x)$ alla sua **perdita** corrispondente. Grossi valori quando il modello non ha buone performance, valori piccoli quando il modello ha buone performance.
- Le loss functions sono particolarmente utili in contesti in cui algoritmi di apprendimento che sfruttano la minimizzazione della stessa
- Perdita media sul test set T_e : $\frac{1}{|T_e|} \sum_{x \in T_e} L(z(x))$
- La **loss function** più semplice è la **0-1 loss** definita come

$$L_{01}(z) = \begin{cases} 1 & z \leq 0 \\ 0 & z > 0 \end{cases}$$

Il problema della 0-1 loss è che ignora la magnitudine dei margini, tenendo conto solo del loro segno. Inoltre, è difficile utilizzarla come funzione obiettivo in un problema di ottimizzazione (very very bad function)

- Un'altra funzione di perdita più interessante è la **hinge loss**, definita nel modo seguente:

$$L_h(z) = \begin{cases} (1 - z) & z \leq 1 \\ 0 & z > 1 \end{cases}$$

- Di seguito altre loss functions viste:
 - Logistic Loss: $L_{log}(z) = \log_2(1 + \exp(-z))$
 - Exponential Loss: $L_{exp}(z) = \exp(-z)$, meno tollerante su esempi che hanno una loss function molto grande
 - Squared Loss: $L_{sq}(z) = (1 - z)^2$

2.2.1 Ranking: Verificare e visualizzare le performance

- Supponiamo che x e x' siano due istanze tali che x riceva uno score più alto di x' , per cui $\hat{s}(x) < \hat{s}(x')$. In sostanza, il classificatore crede più fortemente che la classe di appartenenza di x' sia positiva rispetto a x

- Questa uguaglianza andrebbe bene solo quando effettivamente x e' un negativo e x' e' un positivo. Quando questo non succede si dice che il classificatore commette un **ranking error**
- Il **ranking error rate** e' definito come

$$rank - err = \frac{\sum_{x \in Te^{\oplus}, x' \in Te^{\ominus}} I[\hat{s}(x) < \hat{s}(x')] + \frac{1}{2} I[\hat{s}(x) = \hat{s}(x')]}{Pos \cdot Neg}$$

Dalla formula risultano evidenti alcuni punti:

- Gli errori in cui lo stesso score viene assegnato alle due classi vengono contati per meta' (fattore 1/2)
- Il numero massimo di ranking errors e' pari a $|Te^{\oplus} \times Te^{\ominus}| = |Te^{\oplus}| \cdot |Te^{\ominus}| = Pos \cdot Neg$
- Analogamente, si puo' definire la **ranking accuracy** come $1 - rank-err$. Puo' essere vista come la stima della probabilita' che una coppia arbitraria positiva-negativa sia classificata correttamente (*ranked correctly*), tenendo conto solo del loro segno. Inoltre, e' difficile utilizzarla come funzione obie
- Uno scoring classifier induce implicitamente anche un ranking classifier, semplicemente andando a ordinare le istanze in base allo score assegnato
- Data una funzione di ranking h , uno potrebbe creare diversi classificatori in base ad h scegliendo thresholds differenti

2.3 Stima delle probabilita' delle classi

- Un **class probability estimator** e' uno *scoring classifier* che ha come output un vettore di probabilita' delle classi. Formalmente, $\hat{p} : \mathcal{X} \rightarrow [0, 1]^k$ e' un cpe che associa ad ogni istanza x un vettore $(\hat{p}_1(x), \dots, \hat{p}_k(x))$ dove $\hat{p}_i(x)$ e' la probabilita' assegnata alla classe C_i (e naturalmente k e' il numero di classi)
- Essendo che parliamo di probabilita', per ogni vettore in output devono valere gli assiomi della probabilita', tra cui $\sum_{i=1}^k \hat{p}_i(x) = 1$
- Per trasformare un feature tree in un probability estimation tree si calcola il ratio tra $TP/TP+FP$
- Lo **squared error** (SE) del vettore in output di un cpe e' definita come

$$SE(x) = \frac{1}{2} \|\hat{p}(x) - I_{c(x)}\|_2^2$$

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

- Il **mean squared error** e' semplicemente l'SE medio su tutte le istanze $x \in T_e$. E' definito come:

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

2.4 Probabilita' Empiriche

- Le probabilita' empiriche sono importanti dal momento che possono essere utili per ottenere o migliorare le stime di probabilita' di classificatori o rankers.
- Dato un set di istanze etichettate S , e il numero n_i di istanze della classe C_i , allora il vettore di probabilita' empirica associato a S e'

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

- Dal momento che potremmo lavorare con alcuni dataset non troppo grossi, e' possibile che alcune classi ricevano una probabilita' molto vicina allo 0. Avere probabilita' vicine allo 0 e' problematico dal momento che molte volte si moltiplicano tra loro, per cui e' necessario applicare delle correzioni
- La ***correzione di Laplace** e' una correzione molto comune in questi contesti:

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

Questa formula assume che le distribuzioni di probabilita' siano uniformi (le classi siano ugualmente probabili tra loro)

- E' possibile anche applicare una correzione non uniforme:

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

3 Oltre la classificazione binaria

- Come gia' accennato, la classificazione multi classe e' un problema di classificazione che considera piu' di due classi
- Così come la classificazione, anche gli altri task di scoring, ranking e probability estimation possono essere estesi a questa generalizzazione del problema

3.1 Estendere la classificazione binaria a multiclasse

- Esistono diversi metodi per combinare diversi classificatori binari in un singolo **k**-class classifier:

- *One-versus-rest (unordered)*: viene fatto il training su k classificatori binari in cui il primo separa la prima classe C_1 da C_2, \dots, C_k , il secondo separa la seconda classe da C_1, C_3, \dots, C_k e così' via fino alla classe C_k . Nella fase di training dell' i -esimo classificatore, tutte le istanze della classe C_i vengono trattate come istanze positive, mentre tutte le altre come negative.
- *One-versus-rest (fixed order)*: viene fatto il training su $k - 1$ classificatori binari in base all'ordine di tests stabilito dove il classificatore $\hat{c}_i(x)$ separa C_i da C_{i+1}, \dots, C_n
- *One-versus-one (Simmetrico)*: il training viene fatto su $k(k - 1)/2$ classificatori binari, uno per ogni **coppia di classi** possibile.
- *One-versus-one (Asimmetrico)*: il training viene fatto su $k(k - 1)$ classi, uno per ogni **coppia di classi** in cui conta l'ordine. Non molto utilizzata, utilizza piu' classificatori senza un miglioramento significativo nelle performances rispetto alla controparte simmetrica
- **Output code matrix**: E' una matrice in cui la colonna i indica il classificatore binario i -esimo, mentre le righe indicano le classi C_i . E' uguale a +1 quando le istanze della classe C_i sono considerate positive dal classificatore \hat{c}_i , -1 negative, 0 non sono considerate.
- Per classificare gli esempi, si costruisce un vettore $w(x) = (\hat{c}_1(x), \dots, \hat{c}_n(x))$, contenente l'output degli n classificatori binari sull'istanza x
- Nei casi in cui $d_i = d_j$ per un qualsiasi i, j , si puo' utilizzare uno *scoring classifier* al posto di un classificatore, prendendo come buona la scelta con score massimo. In questo caso la possibilita' che si verifichi la situazione descritta sarebbe drasticamente ridotta.
- Per decodificare il vettore w in una classe, il classificatore calcola la distanza tra w e c_j (dove c_j e' la j -esima riga dell'output code matrix), calcolando per ogni riga $d(w, c) = \sum_i (1 - w_i c_i)/2$. Infine, ritorna la classe corrispondente a quella con la distanza minima (in altri termini, ritorna $\arg\min_j d(w, c_j)$). Il processo di utilizzare la matrice come lookup table e' detto *decoding process*.
- Nello schema *one-vs-rest* il singolo classificatore vede dei datasets molto sbilanciati, per cui avendo pochi esempi positivi il classificatore avra' piu' difficolta' a trovare la correlazione tra i dati nella fase di apprendimento

3.2 Regressione

- Un **function estimator** anche chiamato **regressore** e' una mappa $\hat{f}(x) : \mathcal{X} \rightarrow \mathbb{R}$. (Si noti che a differenza di cio' che e' stato visto fin'ora, il numero delle classi non e' piu' finito)
- Il task della regressione e' quello di imparare da una serie di esempi (*punti*) del tipo $(x_i, f(x_i))$ il valore dell'approssimazione reale di f
- Per ottenere un regressore che approssimi meglio l'andamento dei dati, si possono utilizzare:
 - **Polinomi interpolanti di grado n** : hanno $n + 1$ parametri (il valore di n incognite + il valore di una costante) e sono sempre in grado di interpolare $n + 1$ punti

- **Funzioni a tratti**: un modello costituito da n funzioni ha tratti complessivamente $2n - 1$ parametri ed è sempre in grado di interpolare n punti. Per evitare l'overfitting, il numero di parametri stimati dai dati deve essere considerevolmente più piccolo del numero di datapoints.
- Diversamente dai problemi di classificazione, la **loss function** non è applicata al **margin** ma ai **residui** $f(x) - \hat{f}(x)$. Tipicamente, una loss function è simmetrica rispetto all'asse x (dal momento che vorremmo minimizzare il discostamento in termini di valore assoluto).
- Tipicamente la scelta di una loss function ricade sullo scarto quadratico dal momento che non si vuole che gli scarti negativi e positivi molto simili si cancellino tra di loro. Il problema è che bisogna fare attenzione nei casi in cui ci siano outliers molto grandi dal momento che la funzione è molto sensibile in questi casi.
- **Bias variance dilemma**: Un modello a bassa complessità soffre meno delle variazioni random nei dati, ma potrebbe introdurre un bias sistematico che non può essere risolto nemmeno con grandi quantità di dati. Questo perché una precisione migliore può essere raggiunta solamente mediante l'aggiunta di più parametri. D'altra parte, un modello ad alta complessità, eliminerebbe tale bias a costo di introdurre più errori non sistematici dovuti all'elevata varianza introdotta. Questo può essere visto formalmente dal momento che vale la relazione

$$\mathbb{E}[(f(x) - \hat{f}(x))^2] = \underbrace{(f(x) - \mathbb{E}[\hat{f}(x)])^2}_{Bias^2} + \underbrace{\mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2]}_{Varianza}$$

- **Varianza**: Errore dovuto al fatto che il modello si è conformato troppo ai dati del test set, modellando anche le conseguenti fluttuazioni random dei suoi dati (*overfitting*)
- **Bias**: Errore dovuto al fatto che il modello, avendo pochi parametri, non riesce ad esprimere bene la correlazione tra i dati (*underfitting*)

3.3 Predictive vs Descriptive learning

- Descriptive learning è il task che concerne l'apprendimento di un modello che *descriva* i dati.
- In questo senso, l'output del processo di apprendimento e del problema di apprendimento sono lo stesso: il modello stesso. (ad esempio, prima l'output del problema di apprendimento era una label, score, probabilità ecc..)

3.3.1 Clustering

- **Predictive clustering**: Apprendere una funzione di labelling l da dati **non etichettati**. Così come un classificatore, anche un cluster è una mappa $\hat{q}(x) : \mathcal{X} \rightarrow \mathcal{L}$

4 Concept Learning

- Alla base dell'apprendimento induttivo si fa un'assunzione importante: se il modello riesce ad approssimare la funzione target molto bene dagli esempi, allora approssimerà la funzione target anche su esempi non osservati (futuri)
- Concept learning concerne l'apprendimento di espressioni logiche (anche chiamate *concetti*) dalle istanze di esempio $x \in Te$. Più nello specifico, si vuole apprendere una descrizione della classe dei positivi espressa in termini di espressione logica

4.1 Ripasso Logica

- Se l'espressione A è vera per l'istanza x si dice che A **copre** x
- L'insieme delle espressioni che sono coperte da A è chiamato **estensione** di A denotato come $\mathcal{X}_A = \{x \in \mathcal{X} | A \text{ covers } x\}$
- Se $\mathcal{X}_A \supseteq \mathcal{X}_{A'}$ si dice che A è *almeno generale quanto* A'
- Ogni espressione logica può essere riscritta come una congiunzione/disgiunzione di clausole (*Conjunctive/Disjunctive Normal Form*)
- **Regola**: Clausola $A \rightarrow B$ dove B è un singolo letterale. Viene anche chiamata *clausola di Horn*.

4.2 Lo spazio delle ipotesi

- Un'ipotesi h è una *congiunzione* di vincoli sugli attributi, in cui ogni vincolo può essere:
 - Posto su un valore specifico $Water = Warm$
 - Qualsiasi valore (ignorato) $Water = ?$
 - Nessun valore (*null hypothesis*) $Water = \emptyset$
- Lo spazio di tutte le ipotesi possibili è chiamato **hypothesis space**, denotato con la lettera H
- Il task del concept learning è uguale a quello della classificazione. Dati:
 - Delle istanze \mathcal{X} descritte in termini di i attributi a_i
 - Una funzione target (o *target concept*) $c : \mathcal{X} \rightarrow \{0, 1\}$
 - Un set di ipotesi H in cui le singole ipotesi sono descritte in termini di congiunzione di vincoli sugli attributi a_i
 - Un set di istanze etichettate $(x, c(x)) \in D$ Determinare un'ipotesi **target** $h \in H$ tale che $\forall x \in D, h(x) = c(x)$ cioè che h *copra* ogni x nel training set
- Il numero delle ipotesi è esponenzialmente grande rispetto al numero di attributi e valori che possono assumere. Consideriamo l'esempio in cui le istanze siano rappresentate da un attri-

buto che puo' assumere 3 valori e altri 5 che possono assumere solo 2 valori. Allora, possiamo calcolare:

- nr. istanze distinte: $3 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 96$
 - nr. ipotesi sintatticamente valide: $5 \cdot 4 \cdot 4 \cdot 4 \cdot 4 \cdot 4 = 5120$
 - nr. ipotesi semanticamente valide: $1 + (4 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3) = 973$
- Il task dell'apprendimento equivale a ricercare all'interno dello spazio delle ipotesi le ipotesi che siano consistenti con il target concept c

4.3 Ordinamento delle ipotesi

- Molti algoritmi di apprendimento migliorano la ricerca all'interno dello spazio delle ipotesi sfruttando una proprieta' inerente ad ogni problema di concept learning: l'ordinamento specifico-generale delle ipotesi
- Consideriamo due ipotesi:
 - $h_1 = \langle \text{Sunny}, ?, ?, \text{Strong}, ?, ? \rangle$
 - $h_2 = \langle \text{Sunny}, ?, ?, ?, ?, ? \rangle$
- Dal momento che h_2 impone meno vincoli di h_1 , coprirà un insieme di istanze più grande di quello coperto da h_1
- **Definizione:** (\geq_g) Siano h_j e h_k due ipotesi definite sull'insieme delle istanze \mathcal{X} . h_j è detta **più o ugualmente generale di** h_k (scritto come $h_j \geq_g h_k$) se e solo se $\forall x \in \mathcal{X} : [(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$
- Formalmente, la relazione \geq_g definisce un ordine parziale su H
- Lo spazio delle ipotesi insieme alla relazione di ordinamento appena descritta costituiscono un *reticolo*, in cui le ipotesi più generali sono nella parte inferiore. Se ci si sposta verso la parte superiore diventano più specifiche. Per cui sono ordinate dal basso verso l'alto mediante la relazione descritta.
- Nel vertice superiore del reticolo c'è l'ipotesi nulla (quella più specifica di tutte, composta da soli 0), mentre al vertice inferiore c'è l'ipotesi più generale (composta da tutti punti interrogativi)

4.4 Trovare massima ipotesi più specifica: L'algoritmo Find-S

- L'algoritmo Find-S si basa su due specifiche ipotesi per poter funzionare:
 1. Lo spazio delle ipotesi H contiene un'ipotesi che descrive il target concept c
 2. Il dataset di training D non contiene errori ed è consistente (non ha esempi contenenti "rumore", oppure non ci sono due istanze uguali ma con labels discordi)

- Se prendiamo per vere queste due ipotesi, si può definire un algoritmo per cercare l'ipotesi che meglio approssima c , chiamato **Find-S**:

Algorithm 1: Find-S

Input : D

Output: The maximal specific hypothesis h that covers every example in D

```

1 Initialize  $h$  with the most specific hypothesis in  $H$  ( $\emptyset, \dots, \emptyset$ )
2 for each positive training instance  $x_i$  do
3   if the constraint  $a_i$  in  $h$  is not satisfied by  $x_i$  then
4     replace  $a_i$  in  $h$  by the next more general constraint satisfied by  $x_i$ 
5   end
6 end

```

- L'idea dietro all'algoritmo è quella di partire inizialmente dall'ipotesi più specifica possibile in H (cioè quella composta da soli \emptyset), e generalizzarla ogni qual volta che si trovi un esempio che non viene coperto da tale ipotesi. In questo modo si trova l'ipotesi meno generale possibile ma che copre tutti gli esempi
- Gli esempi considerati per generalizzare h , però, sono solo gli esempi positivi per cui *vengono ignorati gli esempi negativi*. Questo perché il target concept non copre nessun esempio negativo, di conseguenza non è necessaria nessuna revisione dell'ipotesi h
- La proprietà chiave dell'algoritmo find-S è che tutte le ipotesi all'interno di H siano rappresentate come insieme di *congiunzioni* di vincoli
- h è l'ipotesi più specifica in H in grado di coprire tutti gli esempi. h è inoltre consistente anche con gli esempi negativi, posto che:
 - I dati siano corretti
 - Il target concept c sia presente in H
- L'algoritmo find-S, però, presenta alcune limitazioni evidenziate dai punti seguenti:
 - Non si ha modo di capire, né di avere una misura su quanto il learner abbia effettivamente confluito sul target concept
 - In caso ci siano più ipotesi consistenti con gli esempi di train, Find-S sceglierà sempre la più specifica, scartando le altre che potrebbero essere l'effettivo target concept
 - Non sempre i dati di train sono consistenti e l'algoritmo non ha modo di riconoscere quando questo accade
 - Nel caso in cui ci siano multiple ipotesi più generali in grado di coprire tutti gli esempi, l'algoritmo non fa nessuna scelta informata tra le scelte

4.5 L'algoritmo Candidate-Elimination e il Version Space

- **Definizione (Version Space):** Un concetto si dice *completo* se copre tutti gli esempi positivi. Un concetto e' *consistente* se non copre nessun esempio negativo. Il **version space** e' l'insieme di tutti i concetti *completi* e *consistenti*. Formalmente si dice che l'ipotesi h e' consistente con i dati di train D se:

$$Consistent(h, D) = (\forall (x, c(x)) \in D) h(x) = c(x)$$

dove c e' il *target concept*. Analogamente, il version space e' definito formalmente come

$$VS_{H,D} = \{h \in H | Consistent(h, D)\}$$

- Un modo ovvio per rappresentare il version space e' semplicemente quello di enumerare ogni elemento in una lista. L'algoritmo chiamato *List-Then-Eliminate* segue questo principio:

Algorithm 2: List then Eliminate

Input : D

Output: The Version Space VS

- 1 $VS \leftarrow$ list containing every hypothesis in H
 - 2 **for** each training example $(x, c(x))$ **do**
 - 3 | Remove from VS every hypothesis h such that $h(x) \neq c(x)$
 - 4 **end**
 - 5 Return VS
-

Sfortunatamente, l'algoritmo richiede di enumerare ogni possibile elemento all'interno del version space, che e' computazionalmente insostenibile

- Il *Version space* puo' essere anche definito in termini dei suoi limiti:
 - Un **general boundary** chiamato G contenente i membri generalmente massimi, consistenti con D
 - Uno **specific boundary** chiamato S contenente i membri specificamente massimi, consistenti con D
- G ed S definiscono l'intero version space
- **Piu' esempi muovono S verso il basso** (piu' generali), mentre **meno esempi muovono G verso l'alto** (piu' specifici)
- L'algoritmo Candidate-Elimination funziona con lo stesso principio dell'algoritmo *List-Then-Eliminate* ma utilizza una rappresentazione molto piu' compatta del version space. Piu' nello specifico, il version space e' rappresentato da G ed S

- Nel primo passo considera l'intero version space, impostando G ed S a contenere l'ipotesi piu' generale e quella piu' specifica rispettivamente
- Una volta che l'algoritmo termina, l'intero version space specificato dai due limiti G ed S contiene solo ed esclusivamente ipotesi consistenti con gli esempi

Algorithm 3: Candidate Elimination

Input : D

Output: The Version Space defined in terms of G and S

```

1 Initialize  $G$  to the set of maximally general hypothesis in  $H$  ( $?, \dots, ?$ )
2 Initialize  $S$  to the set of maximally specific hypothesis in  $H$  ( $\emptyset, \dots, \emptyset$ )
3 for each training example  $d$  do
4   if  $d$  is a positive example then
5     Remove from  $G$  any hypothesis inconsistent with  $d$ 
6     for each hypothesis  $s$  in  $S$  that is not consistent with  $d$  do
7       Remove  $s$  from  $S$ 
8       Add to  $S$  all minimal generalizations  $h$  of  $s$  such that  $h$  is consistent with  $d$  and some
          member of  $G$  is more general than  $h$ 
9       Remove from  $S$  any hypothesis that is more general than another hypothesis in  $S$ 
10    end
11  else
12    Remove from  $S$  any hypothesis inconsistent with  $d$ 
13    for each hypothesis  $g$  in  $G$  that is not consistent with  $d$  do
14      Remove  $g$  from  $G$ 
15      Add to  $G$  all minimal specializations  $h$  of  $g$  such that  $h$  is consistent with  $d$  and some
          member of  $S$  is more specific than  $h$ 
16      Remove from  $G$  any hypothesis that is less general than another hypothesis in  $G$ 
17    end
18  end
19 end

```

- Nell'algoritmo, le righe 8 e 15 servono a rendere S/G un riassunto di tutte le ipotesi consistenti con gli esempi positivi/negativi
- Le righe 9 e 16 servono a mantenere tali insiemi minimi, mantenendo solo le ipotesi piu'/meno generali
- Mano a mano che gli esempi vengono considerati dall'algoritmo, i limiti G ed S si avvicinano sempre di piu' monotonamente, delimitando il version space di ipotesi candidate sempre piu' piccolo.

- Se in H e' contenuta l'ipotesi univoca che rappresenta il target *target concept* e il numero di esempi e' abbastanza grande, allora l'algoritmo Candidate Elimination convergera' all'ipotesi target univoca.
- Nel caso invece in cui ci fossero esempi inconsistenti, G ed S eventualmente tendono a diventare vuoti con l'aumento del numero di esempi considerati. Tale situazione si verifica anche quando il target concept non puo' essere rappresentato con il linguaggio attuale (coniunzioni di vincoli)
- Nel caso in cui non si avessero abbastanza esempi per cui il VersionSpace contenga ancora molte ipotesi si potrebbero:
 - Richiedere altri esempi etichettati ad un oracolo
 - Usare le ipotesi del version space per classificare esempi futuri
- Nel primo caso, consideriamo lo scenario in cui il *learner* abbia accesso in qualche modo ad un *oracolo* esterno. Il learner puo' utilizzare delle *query* (istanze) che vengono classificate correttamente dall'oracolo per aumentare la velocita' di convergenza.
- Una strategia generale per la generazione di query e' scegliere quelle che vengono classificate correttamente da meta' delle ipotesi nel version space e negativamente dalla restante meta'. In questo modo il version space si dimezza ad ogni esempio, per cui il target concept puo' essere trovato in $\log_2(|VS|)$ esperimenti
- Unbiased Learner: modello che utilizza il linguaggio dell'intera logica (disgiunzioni, congiunzioni, negazioni). Il numero di concetti esprimibili e' enorme, ed e' pari al power set $P(X)$. **Sicuramente** H conterra' il target concept
- Un unbiased learner e' impossibile da avere. Il bias e' un concetto fondamentale che sta alla base dell'apprendimento.

4.6 Inductive Bias

- Serve per fare l'inductive leap
- I learners possono essere classificati anche in base al loro bias induttivo:
 - **Rote learner**: Salva esempi, classifica x se e solo se matcha un esempio precedente (nessun bias induttivo = nessuna abilita' di generalizzare)
 - **Version Space candidate elimination**: Classifica gli esempi se tutti i membri del version space concordano. (bias induttivo = lo spazio delle ipotesi H contiene il concetto target c)
 - **Find-S**: TODO

TODO: Inserire esercizi

5 Modelli ad Albero

- I modelli ad albero in generale nel libro vengono descritti come *feature trees*. Questi alberi generali possono poi essere specializzati in diversi tipi di modello ad albero in base al task da risolvere quali:
 - Decision trees
 - Ranking trees
 - Probability estimation trees
 - Regression trees
 - Clustering trees
- Un modello ad albero e' anche un modo per rappresentare diversi percorsi all'interno dello spazio delle ipotesi, un percorso tra ipotesi in cui i passi sono di *generalizzazione* o *specializzazione*. La specializzazione si ottiene scendendo nell'albero verso le foglie (aggiungendo vincoli), mentre la generalizzazione si ottiene risalendo l'albero verso la radice (rimuovendo vincoli)
- I feature trees raggruppano lo spazio degli esempi, cioe' lo segmentano in funzione ai concetti che rappresentano le foglie.
- Per trasformare un feature tree in un decision tree, si associa una label di classe nella foglia che rappresenta il concetto. In sostanza si associa una stima di classe da a tutti gli esempi coperti da quel concetto.

Feature tree  Decision tree
Labelling

- Alberi diversi possono rappresentare equivalenti concetti ma con forma differente. Perché?
 - Espressioni logiche sintatticamente differenti possono essere semanticamente uguali
 - Siccome un albero rappresenta diversi concetti (che sono di fatto espressioni logiche), allora concludiamo il ragionamento.

Gli alberi corrispondono ad espressioni logiche in forma normale disgiuntiva (DNF)

- Problema della troppa espressività: e' possibile costruire un decision tree che classifichi un qualunque training set con 0 errori. (Si fa costruendo un feature tree in cui le foglie coprono uno e un solo esempio del training set, questo ovviamente causa overfitting)
- Per evitare l'overfitting si possono introdurre due *bias induttivi*:
 - Diminuire l'espressività del linguaggio in fase di apprendimento
 - Utilizzare un modo differente per cercare lo spazio delle ipotesi, scoraggiando l'algoritmo ad apprendere ipotesi (alberi) troppo complesse

5.1 Feature tree

- Un feature tree e' un albero in cui ogni **nodo interno** e' etichettato con una **feature**. Ogni **diramazione** e' etichettata con un **letterale** (valore possibile della feature)
- L'insieme dei letterali di un dato nodo interno e' chiamato **split**. Ogni nodo interno di un feature tree costituisce uno split, cioe' una segmentazione dello spazio degli esempi coperto dall'ipotesi
- Per costruire un *feature tree* a partire da un dataset:
 - Si costruisce un **nodo** per ogni feature possibile
 - Per ogni nodo si dipartono tanti rami quanti sono i valori possibili della feature corrispondente a quel nodo (es. variabile binaria = 2 rami)
 - Nelle foglie si annotano quante sono le istanze etichettate come *positive* e quante *negative* che hanno *valore = ramo* in riferimento alla feature del nodo padre
- Come detto in precedenza, sono un modo compatto per rappresentare diverse ipotesi all'interno dello spazio delle ipotesi, in cui ognuna ipotesi e' rappresentata implicitamente da una foglia. L'ipotesi risultante e' la congiunzione dei vincoli sui letterali incontrati nel percorso dalla *foglia* alla *radice*

5.2 L'algoritmo GrowTree

Algorithm 4: GrowTree algorithm

Input : Data D , the set of features F

Output: Feature tree T with labelled leaves

```

1 if  $Homogeneous(D)$  then
2   | return  $Label(D)$ 
3 end
4  $S \leftarrow BestSplit(D, F)$ 
5 split  $D$  into subsets  $D_i$  according to feature values in  $S$ 
6 for each  $i$  do
7   | if  $D_i \neq \emptyset$  then
8     |    $T_i \leftarrow GrowTree(D_i, F)$ 
9   | else
10  |    $T_i$  is a leaf labelled with  $Label(D)$ 
11  | end
12 end
13 return a tree whose root is labelled with  $S$  and whose children are  $T_i$ 

```

- L'algoritmo **GrowTree** e' la procedura di apprendimento piu' utilizzata dalla maggior parte dei tree learners. Segue un approccio *divide and conquer* (segmentazione progressiva in modo ricorsivo) e *greedy* (sceglie sempre l'alternativa migliore alla riga 4, senza reconsiderarla mai).
- L'approccio greedy potrebbe portare ad una soluzione *sub-ottima*. Per risolvere il problema si potrebbero utilizzare algoritmi con forme di backtracking al costo di un maggior tempo di esecuzione e memoria
- **GrowTree** e' l'astrazione massima possibile, poiche' assume che alcune funzioni specifiche per la soluzione del task siano implementate. In questo modo e' possibile costruire modelli ad albero per diversi task, ma utilizzando lo stesso algoritmo per la fase di learning. Come si vede dall'algoritmo, le procedure sono:
 - **Label**(D): ritorna l'etichetta piu' appropriata per etichettare tutte le istanze del dataset D . Nel caso della classificazione potrebbe essere ad esempio la classe maggioritaria.
 - **Homogeneous**(D): ritorna **true** se il dataset D e' sufficientemente omogeneo. Anche qui, ogni task specifico ha il suo concetto di omogeneita'. Nella classificazione, potrebbe essere vero solo quando l'intero dataset ha istanze appartenenti ad una sola classe
 - **BestSplit**(D , F): ritorna la *feature* che genera lo split migliore

5.3 Decision Trees

- Introduciamo il criterio di *purezza*, con cui valuteremo gli split e le etichette da assegnare alle foglie per trasformare un feature tree in un decision tree.
- Supponiamo di avere a che fare con features booleane, per cui dato un dataset D , lo split risultante dividera' D in D_1 e D_2
- Supponiamo inoltre che le classi disponibili per la classificazione siano 2 (positiva e negativa)
- La **purezza** di una partizione D , costituita da n^+ esempi positivi e n^- esempi negativi e' data dalla *probabilita' empirica*

$$\dot{p} = \frac{n^+}{n^+ + n^-}$$

- Purezza massima: 0 oppure 1, che indica la presenza di soli positivi o soli negativi
- Avendo ora una nozione di purezza, possiamo introdurre delle misure per calcolare l'**impurezza** di una partizione:
 - **Minority class**: ($\min\{\dot{p}, 1 - \dot{p}\}$) rappresenta la probabilita' di fare un errore in caso sia stata scelta la classe maggioritaria come label
 - **Gini Index**: ($2\dot{p}(1 - \dot{p})$) rappresenta la probabilita' di fare un errore in caso siano state assegnate randomicamente le labels alle istanze, ma campionandole dalla distribuzione delle classi originaria

- **Entropy:** $(-\hat{p} \cdot \log_2(\hat{p}) - (1 - \hat{p}) \cdot \log_2(1 - \hat{p}))$ e' l'informazione attesa. Più e' pura la partizione, minore sarà l'informazione attesa.
- Si noti che l'impurezza' deve essere uguale nel caso si scambiassero le classi (positivi con negativi e viceversa). Inoltre deve essere pari a 0 quando $\hat{p} = 0, 1$ e raggiungere il massimo in $\frac{1}{2}$.
- L'impurezza' **attesa** di uno split $D \rightarrow \{D_1, \dots, D_l\}$ e' data dalla media delle impurezza' di ogni singolo sottoinsieme D_j :

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

dove il rapporto $\frac{|D_j|}{|D|}$ rappresenta la probabilita' che un esempio preso dalla partizione D "ricada" nella partizione D_j .

- Purity Gain: Immaginiamo che il dataset D venga splittato in $\{D_1, \dots, D_l\}$ dalla feature f . Con purity gain intendiamo la quantita'

$$Imp(D) - Imp(\{D_1, \dots, D_l\})$$

che concettualmente indica il *gain in purezza* che si avrebbe se si includesse f come feature di split.

- Questo concetto di impurezza, viene utilizzato quindi dall'algoritmo **BestSplit** per calcolare lo split che minimizza l'incertezza

Algorithm 5: BestSplit algorithm

Input : Data D , the set of features F

Output: Feature f to split on

```

1  $I_{min} \leftarrow 1$ 
2 for each  $f \in F$  do
3   if  $Imp(\{D_1, \dots, D_l\}) < I_{min}$  then
4      $I_{min} \leftarrow Imp(\{D_1, \dots, D_l\})$ 
5      $f_{best} \leftarrow f$ 
6   end
7 end
8 return  $f_{best}$ 

```

5.3.1 Entropia

- L'entropia e' una misura che quantifica la quantita' di confusione di un certo evento, ed e' misurata in **bits**

- Se un esperimento ha n possibili outcome (equiprobabili), e' possibile rappresentare ogni outcome possibile mediante $b = \lceil \log_2 n \rceil$ bits di informazione
- Per fare un'esempio, supponiamo di lanciare un dado a 6 facce:
 - Per rappresentare l'outcome si utilizzano $b = \lceil \log_2 6 \rceil = \lceil 2.58 \rceil = 3$ bits di informazione
 - Supponiamo ora di trovare un "suboutcome", cioe' una quantita' di informazione minore ma dipendente comunque dall'outcome finale, ad esempio, la **parita'** del risultato del lancio
 - Per rappresentare la parita', sarebbero necessari $b_1 = \lceil \log_2 2 \rceil = 1$ bits.
 - Dopo il lancio, se abbiamo l'informazione sulla parita', si riduce la *quantita' di confusione* rispetto all'evento, per cui avremmo gia' b_1 bits di informazione su b . La differenza $b - b_1$ quantifica quanta informazione rimane da comunicare per sapere l'intero outcome
- Questo e' cio' che capita anche quando vogliamo utilizzare l'entropia per scegliere gli split migliori:
 - Il dataset iniziale ha una certa quantita' di confusione, che dipende da come e' distribuita l'etichetta nel training set
 - Dopo che viene fatto lo split, il dataset e' diviso in diverse partizioni, ciascuna con una certa confusione associata (sempre legata a come l'etichetta di classe e' distribuita all'interno della partizione)
 - Un buono split ci permette di *guadagnare* parte dell'informazione, riducendo l'incertezza iniziale (quella del nodo padre)
- Se un evento E ha probabilita' p di accadere, allora l'informazione ottenuta se osserviamo l'evento e':

$$I(E) = \log_2\left(\frac{1}{p}\right) = -\log_2 p$$

- Se un evento e' altamente probabile $p \approx 1$, allora $I(E) \rightarrow 0$
- Se un evento e' improbabile $p \approx 0$, allora $I(E) \rightarrow 1$
- L'idea e': immagina di lanciare ripetutamente una moneta truccata con $P(T) = 0.9$. Quanto saresti ancora sorpreso se dopo 10 lanci in cui e' uscito solo testa, uscisse ancora T ?
- Se un esperimento ha n possibili outcomes E_1, \dots, E_n , ognuno con probabilita' p_1, \dots, p_n , allora la quantita' media di informazione ottenuta quando osserviamo un generico outcome e' l'*informazione attesa* o *entropia*

$$H(E) = \sum_{i=1}^n p_i \cdot \log_2\left(\frac{1}{p_i}\right) = - \sum_{i=1}^n p_i \cdot \log_2(p_i)$$

- E' la somma delle probabilita' di accadere degli eventi i esimi per la loro rispettiva quantita' di informazione attesa

5.4 Alberi di decisione e coverage plot

- Ciascun nodo (interno ed esterno) in un albero di decisione corrisponde ad un'ipotesi fatta nello spazio degli esempi. Analogamente, corrisponde anche ad un segmento nello spazio del coverage plot, in cui:
 - I segmenti verticali corrispondono ai nodi **puri**, con $\dot{p} = 1$
 - I segmenti orizzontali corrispondono ai nodi con $\dot{p} = 0$ (composti da soli esempi della classe negativa)
 - I segmenti in diagonale corrispondono ai nodi **massimamente impuri** (coprono la stessa quantità di esempi positivi e negativi)

L'ordine dei segmenti all'interno del coverage plot non dipende dalla struttura dell'albero ma dalla probabilità empirica dei nodi corrispondenti

- Nella fase di apprendimento, mano a mano che si aggiungono split, i segmenti si spostano sempre più in alto verso il ROC heaven (cioè il punto con AUC massimo), fino ad arrivare a coincidere con esso. Ciò significa che tutti gli esempi positivi (del training set) sono coperti dall'albero.

5.5 Ranking Trees

- Come già detto, i modelli ad albero dividono lo spazio degli esempi in segmenti (o partizioni)
- Se ordiniamo le foglie di un feature tree, è possibile ottenere un ranking
- Dal momento che si conosce la distribuzione delle classi all'interno del singolo segmento, si possono ordinare secondo l'*empirical probability* della foglia
- In caso di foglie con pari merito, si dà la precedenza a quelle che coprono un numero maggiore di esempi
- L'ordinamento predilige le foglie più adatte a coprire più esempi positivi andando via via verso quelle più adatte a coprire esempi negativi

Il ranking trees ottenuti mediante l'ordinamento delle foglie del feature tree secondo empirical probability, produce sempre una curva convessa nel ROC plot sui dati di train

- La citazione è dimostrabile nel modo seguente:
 - La pendenza di un segmento è data dal rapporto $\frac{\dot{p}}{(1-\dot{p})}$.
 - Al crescere di \dot{p} si ottiene una trasformazione monotona, poiché se scelgo $\dot{p}' > \dot{p}$, allora anche la pendenza $\frac{\dot{p}'}{(1-\dot{p}')} > \frac{\dot{p}}{(1-\dot{p})}$
 - Quindi, un ordinamento decrescente delle probabilità empiriche causa un ordinamento decrescente delle pendenze, di fatto generando una curva convessa

- La crescita di un albero man mano che l'apprendimento sul dataset progredisce puo' essere visto in termini di coverage plot come l'iterazione dei due passaggi seguenti: (o alternativamente, l'esecuzione di una sequenza del primo passaggio per un'esecuzione finale del secondo)
 1. Aggiungi uno split nel coverage plot. Questo dividera' il/i segmento/i piu' grandi (padri)
 2. Riodina i segmenti secondo probabilita' empirica associata crescente

5.6 Labelling di un feature tree

- Etichettare un albero consiste nell'apportare delle etichette di classe nei nodi foglia
- Se noi abbiamo c classi e l foglie in un albero, ci sono c^l modi possibili di etichettare le foglie
- In un albero con l foglie, ci sono $l!$ possibili *ordinamenti* dei segmenti corrispondenti alle foglie nel coverage space
- Tipicamente per etichettare una foglia generica si utilizza il criterio della classe di maggioranza. Questo criterio semplicistico nasconde pero' un problema: non tiene conto dei differenti costi a fronte di un errata missclassificazione.
- Spesso si vuole ottenere un labelling basato su dei costi di missclassificazione specifici al dominio di riferimento. Ad esempio, supponiamo di voler rilevare la presenza di una data malattia potenzialmente grave. Chiaramente, il costo di missclassificazione come falso negativo e' piu' grande di un costo di missclassificazione come falso positivo poiche' il costo di una vita umana e' piu' grande del costo economico di ulteriori test.
- $c = \frac{c_{FN}}{c_{FP}}$ denota il *cost ratio*, cioe' il rapporto tra costo di *missclassificazione dei positivi* e il costo di *missclassificazione dei negativi*
 - Quando $c > 1$, significa che costano piu' gli errori sui positivi che gli errori sui negativi. Il modello in questo modo, cerchera' di commettere meno errori sui positivi rispetto ai negativi.
 - Viceversa, quando $c < 1$ costano di piu' gli errori sui negativi rispetto ai positivi
 - Quando invece $c = 1$, i costi sono uguali.

Quando $c > 1$, il modello andra' ad assegnare la classe positiva ad un maggior numero di foglie rispetto all'assegnamento della classe maggioritaria, anche nelle foglie con un maggior numero di negativi. L'inverso succede invece quando $c < 1$, cioe' la classe negativa viene assegnata ad un maggior numero di foglie rispetto all'assegnamento con il criterio della classe maggioritaria

- L'ordine parte da tutte classi negative, e man mano che il *cost-ratio* cresce le etichette passano da negative a positive, arrivando infine alla stessa classificazione che si avrebbe se fossero state assegnate mediante *empirical probability*

5.7 Trasformazione da feature tree a modello

- Come detto in precedenza, un feature tree puo' essere trasformato in un qualsiasi modello ad albero per la soluzione di uno specifico task.
- E' possibile sfruttare la distribuzione delle classi nelle foglie per ottenere:
 1. Un **ranker**: ordinando in modo crescente i segmenti foglia in base alla loro empirical probability, calcolate sul training set
 2. Un **probability estimator**: applicando una stima di probabilita' ad ogni foglia calcolata mediante *Laplace* oppure *m-smoothing*
 3. Un **classificatore**: scegliendo il *punto di operazione ottimale* (decision threshold) come l'intersezione della curva del classificatore nel *ROC plot* e la curva con coefficiente angolare pari a $\frac{1}{c \cdot clr}$. Come risultato, le foglie associate ai segmenti dopo il punto di operazione prediranno positivi, mentre le rimanenti negativi.
- Ovviamente esiste un altro metodo piu' semplice per scegliere la classe senza dover effettuare una costruzione geometrica:
 1. Nella foglia *i-esima* calcoliamo il prodotto $\frac{n_i^-}{n_i^+} \cdot \frac{c_{FP}}{c_{FN}}$. In questo modo si tiene conto degli errori di missclassificazione come se ogni esempio positivo e ogni esempio negativo venisse missclassificato.
 2. Col risultato del prodotto calcolato al passo 1, utilizziamo il criterio della classe di maggioranza. Se > 1 allora assegnamo la label della classe a numeratore, altrimenti quella a denominatore.

5.8 Semplificazione di alberi

- Un aspetto importante dei modelli predittivi (e quindi anche dei modelli ad albero), e' che e' dimostrato che l'aumentare della complessita' del modello (numero di nodi), l'errore del modello sul *test set* e sul *training set* tende a diminuire fino ad una certa soglia.
- Da questa soglia in poi, l'aumentare del numero di nodi fa diminuire l'errore solamente sul *training set*, mentre l'errore sul *test set* continua a crescere. Questa situazione e' gia' stata vista e rappresenta l'*overfitting* del modello.
- Ci sono diversi modi per diminuire la complessita' dei modelli e evitare cosi' il conseguente overfitting.

5.8.1 Pruning

- E' possibile applicare una fase di postprocessing di potatura chiamata pruning

- Può essere effettuato dopo il labelling, in caso i figli dello stesso padre predicano la stessa classe, si considererebbe solo il nodo padre. In questo caso, è particolarmente utile in casi in cui si voglia trasmettere il modello (siccome pesa meno) o se lo si voglia leggere. Il problema è che peggiora la precisione del modello, al momento che l'AUC risultante è minore.
- Alternativamente può essere impiegato un passaggio dell'algoritmo di *post-processing* [PruneTree](#).
- L'algoritmo utilizza un dataset separato (*pruning set*) per testare se la precisione del modello semplificato (output dell'algoritmo) è maggiore di quello precedente

Algorithm 6: PruneTree(T, D) algorithm - reduce-error pruning of a decision tree

Input : Labelled data D , decision tree T

Output: Pruned tree T'

```

1 for every internal node  $N$  of  $T$ , starting from the bottom do
2    $T_N \leftarrow$  subtree of  $T$  rooted at  $N$ 
3    $D_N \leftarrow \{x \in D \mid x \text{ is covered by } N\}$ 
4   if accuracy of  $T_N$  over  $D_N$  is worse than majority class in  $D_N$  then
5     replace  $T_N$  in  $T$  by a leaf labelled with the majority class in  $D_N$ 
6   end
7 end
8 return pruned version of  $T$ 

```

5.8.2 Stima degli errori di generalizzazione

- Alternativa alla fase di post processing discussa in precedenza ([PruneTree](#)) che diversamente è impiegata durante la fase di apprendimento
- Si associa un valore k ad ogni foglia. Se la foglia non diminuisce l'errore del padre di almeno $k + g$, allora non viene creata a priori. In altri termini significa che non viene creata la foglia se la classificazione rispetto al nodo padre non migliora di almeno g istanze.
- $E_{tot} = \sum_{i=1}^N e_i + N \cdot g$ (errore di generalizzazione)

5.9 Sensitivita' rispetto alla distribuzione delle classi

I criteri di impurezza del *Gini index* e quello dell'*entropia attesa*, sono sensibili ai cambiamenti nella distribuzione delle classi, mentre \sqrt{Gini} non lo è.

- Se si volessero inoltre includere i costi nel criterio di splitting, si potrebbe decidere di introdurre delle *copie* di esempi positivi/negativi all'interno del train set.

- Supponiamo ad esempio che i costi sui positivi siano 10 volte piu' costosi della controparte negativa
- E' possibile *inflazionare* il dataset di train inserendo per ogni esempio positivo 10 copie dello stesso
- Il metodo dell'inflazione rappresenta un'*alternativa* per includere il *cost-ratio* rispetto al metodo discusso in precedenza (per determinare le condizioni operative del modello)
- L'inflazione del dataset aumenta i tempi di training

5.10 Apprendimento di alberi come riduzione della varianza

- Quando si parla di predizione di una classe, lo scopo e' quello di predire una variabile Booleana (*Bernoulli*) incerta che ha una certa probabilita' di accadere p
- L'apprendimento di un albero consiste nella minimizzazione di una funzione obiettivo, rappresentata dalla varianza di tale variabile pari a $(1 - p)$
- Possiamo definire negli stessi termini un problema di **regressione**, in cui la varianza e' calcolata come

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

5.10.1 Regression trees

- Negli alberi di regressione le variabili target son ocontinue e non discrete come visto fino ad ora
- Per apprendere un feature tree che verra' utilizzato per il task di regressione, si devono apporre delle modifiche alle procedure gia' viste per l'algoritmo [GrowTree](#):
 - Si utilizza al posto della misura di impurita' $Imp(Y)$ la varianza $Var(Y)$ nell'algoritmo [BestSplit](#)
 - La funzione [Label\(Y\)](#) ritorna il valor medio di tutti i valori nella foglia Y
 - La funzione [Homogeneous\(Y\)](#) ritorna **true** se la *varianza* della foglia Y e' al di sotto di un certo limite (*threshold*)
- Gli alberi di regressione sono piu' suscettibili all'overfitting dal momento che le foglie conterranno molti meno esempi

5.10.2 Clustering trees

- Introduciamo una funzione $dis : X \times X \rightarrow \mathbb{R}$ che indica, data una *coppia* di istanze nello spazio degli esempi la loro *dissimilarita'*

- La nozione di dissimilarita' puo' essere estesa ad un intero set D :

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

- $Dis(D)$ puo' essere quindi utilizzata come indice di impurezza nell'algoritmo **BestSplit**. Piu' bassa e' la dissimilarita' di un dataset D_j migliore sara' il cluster delle istanze D .
- La dissimilarita' di uno split e' data dalla media pesata di ogni split j -esimo

$$Dis(\{D_1, \dots, D_l\}) = \frac{|D_j|}{|D|} Dis(D_j)$$

(probabilita' di passare dal nodo padre al nodo j -esimo moltiplicata per la dissimilarita' dello split j -esimo)

- La dissimilarita' $dis(x_1, x_2)$ puo' essere calcolata mediante distanza euclidea: se l'istanza e' espressa solo in termini di features numeriche, si puo' definire come

$$dis(\vec{x}_1, \vec{x}_2) = (\vec{x}_1 - \vec{x}_2)^2 = \sum_{j=1}^d [x_{1j} - x_{2j}]^2$$

- E' dimostrabile sostituendo la formula di $dis(x_1, x_2)$ all'interno di $Dis(D)$ che $Dis(D) = 2 \cdot Var(D)$
- Gli split quindi diminuiscono la varianza rispetto alle features di valori numerici. Ogni split avra' quindi sempre varianza minore del padre.
- I clusters molto piccoli avranno una dissimilarita' molto piccola, per cui sarebbe necessario un *pruning* di tali nodi per evitare un overfitting
- I clusters composti invece da singole istanze, dovrebbero essere ignorati quando viene calcolata la dissimilarita' di uno split, poiche' ritornano la dissimilarita' piu' ottimistica possibile (0)

5.10.3 Coesione e Separazione di Clusters

- La varianza di una dataset D (e conseguentemente la sua dissimilarita') son costanti
- Consideriamo k cluster (D_1, \dots, D_k) di D
- Il centroide complessivo del dataset e' definito come

$$\vec{c} = \frac{1}{|D|} \sum_{\vec{x} \in D} \vec{x}$$

- Possiamo definire diverse quantita':

- **TSSE** = (*Total Sum of Squared Errors*)

$$TSSE = \sum_{\vec{x} \in D} (\vec{x} - \vec{c})^2 = |D| \cdot Var(D) = \frac{|D|}{2} Dis(D)$$

- **WSSE** = (*Within Sum of Square Errors*) - Indica quanto i clusters sono coesi, ottenuta calcolando quanto le istanze si discostano dal **centroide interno**, cioè l'istanza che più si avvicina alla media. (il centroide *i-esimo* è il centroide calcolato rispetto al cluster *i-esimo*).

$$WSSE = \sum_{i=1}^k \sum_{\vec{x} \in D_i} (\vec{x} - \vec{c}_i)^2$$

- **BSSE** = (*Between Sum of Square Errors*) - Indica quanto i clusters sono separati tra di loro, ottenuta calcolando quanto i centroidi interni dei singoli clusters si discostano dal centroide complessivo (quello dell'intero dataset *D*)

$$BSSE = \sum_{i=1}^k |D_i| (\vec{c}_i - \vec{c})^2$$

- È possibile derivare che $TSSE = WSSE + BSSE$, per cui quello che si vuole ottenere è:
 - Una minimizzazione del termine $WSSE$, siccome si vuole che le istanze di un cluster siano il *più coese possibile* tra loro
 - Una massimizzazione del termine $BSSE$, siccome si vuole che le istanze di un cluster siano il *più separate possibile* dalle istanze degli altri clusters

6 Modelli a Regole

6.1 Apprendimento di liste di regole

- Una regola si manifesta come un'implicazione logica. Più di preciso è formata da un *antecedente* e da una *conseguenza*. La conseguenza è l'assegnamento della classe, mentre l'antecedente è una congiunzione logica di vincoli sugli attributi.
- Una serie di regole può essere vista concettualmente anche nel modo seguente

```
1 if Literal1 then class=X
2 else if Literal2 then class=Y
3 else ...
4 ...
5 else class=Default
```

“Se un determinato letterale (Literal1) è soddisfatto da una determinata istanza, allora predico la classe X”

- La procedura per imparare un insieme di regole è riassumibile essenzialmente in 3 passi fondamentali:
 1. Si costruisce il *corpo* (antecedente) della regola, andando ad aggiungere letterali in congiunzione tra loro. Proprio come succedeva per gli alberi, il *corpo* della regola andrà ad identificare un *segmento* nello spazio degli esempi. Per scegliere tra le alternative possibili, si utilizza la *copertura*, cioè la dimensione del segmento delle istanze identificato dal corpo, che deve essere la più grande possibile e con *purezza* massima.
 2. Si seleziona poi una label che sarà inserita come *testa* (conseguenza) della regola (`Class = label`)
 3. Il segmento delle istanze coperto dalla *testa* della regola viene eliminato dal dataset di train, ripartendo infine dal punto 1.
- Questa procedura genera una lista (o sequenza) di regole

La motivazione del passo 3 è perché questo tipo di algoritmi è di tipo greedy, cioè viene scelta sempre la scelta ottimale. Se non si eliminassero le istanze coperte dalla regola appena appresa, l'algoritmo sceglierebbe di nuovo la stessa (siccome è sempre quella migliore)

- Le regole apprese mediante il metodo descritto devono essere interpretate nello stesso ordine in cui sono state apprese
- Un'altra differenza tra l'apprendimento di modelli a regole e modelli ad albero è che nei modelli ad albero si usa una misura di purezza che considera *tutti i valori possibili di una feature* utilizzando una media pesata, mentre nei modelli a regole si utilizza una misura di purezza sui singoli vincoli dei letterali

- Come già visto anche, i modelli a regole rispetto i modelli ad albero, utilizzano anche la *copertura* del segmento come euristica di scelta delle regole

Algorithm 7: LearnRuleList(D) - learn an ordered list of rules

Input : Labelled training data D

Output: Rule list R

```

1  $R \leftarrow \emptyset$ 
2 while  $D \neq \emptyset$  do
3    $r \leftarrow \text{LearnRule}(D)$ 
4   append  $r$  to the end of  $R$ 
5    $D \leftarrow D \setminus \{x \in D \mid x \text{ is covered by } r\}$ 
6 end
7 return  $R$ 

```

Algorithm 8: LearnRule(D) - learn a single rule

Input : Labelled training data D

Output: rule r

```

1  $b \leftarrow \text{true}$ 
2  $L \leftarrow$  set of available literals
3 while not Homogeneous( $D$ ) do
4    $l \leftarrow \text{BestLiteral}(D, L)$ 
5    $b \leftarrow b \wedge l$ 
6    $D \leftarrow \{x \in D \mid x \text{ is covered by } b\}$   $L \leftarrow L \setminus \{l' \in L \mid l' \text{ uses same feature as } l\}$ 
7 end
8  $C \leftarrow \text{Label}(D)$ 
9  $R \leftarrow$  if  $b$  then  $\text{Class} = C$ 
10 return  $r$ 

```

- Un *feature tree* con branching a destra (se falso, considera la prossima foglia) corrisponde ad una lista di regole a letterali singoli

Liste di regole ereditano la proprietà degli alberi di decisione di avere una coverage curve **convessa** nel training set

- Anche in questo caso è possibile ordinare le regole secondo probabilità empirica, ottenendo un ranker
- L'ordine delle probabilità empiriche associate ad ogni regola (l'ordine di ranking) è diverso dall'ordine appreso dall'algoritmo di inferenza

- Per la dimostrazione vedere esempio sulle slides che dimostra che nel ROC plot la curva con AUC massima e' quella ordinata secondo le prob. emp.

6.2 Apprendere insiemi non ordinati di regole

- In linea di massima, gli algoritmi per l'apprendimento di insiemi di regole funzionano in questo modo:
 1. Si seleziona la classe che si vuole apprendere
 2. Si selezionano i *corpi* delle regole che coprono la piu' grande porzione, cioe' quelli che comprendono piu' esempi della classe scelta. In altri termini, si sceglie la regola con la precisione piu' alta rispetto alla classe scelta
- Questi passaggi producono un insieme di regole. Non c'e' nessun motivo di ordinarle dal momento che coprono gli esempi della stessa classe.
- La differenza degli *insiemi* rispetto alle *liste* di regole, e' che nel primo caso le regole sono *applicate contemporaneamente* e non in sequenza come nel secondo. Per questa ragione hanno una capacita' espressiva maggiore con la conseguenza di riuscire ad esprimere segmenti piu' fini, composti da istanze raggiunte da molteplici regole.

Algorithm 9: LearnRuleSet(D) - learn an unordered set of rules

Input : Labelled training data D

Output: Rule set R

```
1  $R \leftarrow \emptyset$ 
2 for every class  $C_i$  do
3    $D_i \leftarrow D$ 
4   while  $D_i$  contains examples of class  $C_i$  do
5      $r \leftarrow \text{LEarnRuleForClass}(D_i, C_i)$ 
6      $R \leftarrow R \cup \{r\}$ 
7      $D_i \leftarrow D_i \setminus \{x \in C_i \mid x \text{ is covered by } r\}$ 
8   end
9 end
10 return  $R$ 
```

Algorithm 10: LearnRuleForClass(D_i, C_i) - learn a single rule for the given class

Input : Labelled training data D_i ; class C_i

Output: rule r

```

1  $b \leftarrow true$ 
2  $L \leftarrow$  set of available literals
3 while not Homogeneous( $D$ ) do
4    $l \leftarrow BestLiteral(D, L, C_i)$ 
5    $b \leftarrow b \wedge l$ 
6    $D \leftarrow \{x \in D \mid x \text{ is covered by } b\}$ 
7    $L \leftarrow L \setminus \{l' \in L \mid l' \text{ uses same feature as } l\}$ 
8 end
9  $r \leftarrow$  if  $b$  then  $Class = C_i$ 
10 return  $r$ 

```

6.2.1 Miopia

- Il problema nell'usare la *precisione* come euristica di ricerca e' che tende a concentrarsi troppo a cercare le regole con precisione piu' alta, senza considerare altre regole, potrebbero avere una precisione complessiva piu' grande mediante opportune estensioni (cioe' con l'aggiunta di una disgiunzione nel passo successivo)
- Succede perche' l'algoritmo non e' in grado di "vedere" e quindi considerare anche il livello successivo del prossimo passo
- Questo problema e' detto *miopia* della precisione, ed e' risolubile applicando una correzione di Laplace
- Applicando la correzione di Laplace si simulano delle imprecisioni nella copertura delle regole fatte di un solo antecedente, in modo da rendere valide anche le regole che non sarebbero considerate diversamente dall'algoritmo
- Un'altro metodo consisterebbe nell'impiego di algoritmi con approccio beam search, che tengono in memoria k candidati migliori al posto di uno, in modo da tenere aperto un numero maggiore di alternative

La correzione di Laplace aggiunge un esempio positivo ed un esempio negativo nel conteggio della probabilita' empirica

$$\dot{p}_i = \frac{n_i^+ 1}{n_i^+ + 1 + n_i^- + 1} = \frac{n_i^+ 1}{n_i + 2}$$

6.3 Insiemi di regole come rankers

- Anche in questo caso, e' possibile ottenere un ranker tramite l'ordinamento delle empirical probabilities
- Una particolare attenzione deve essere pero' posta alle regole che si attivano insieme per una determinata istanza
- Per poter calcolare l'empirical probability di questi casi particolari si fa una *micro-average* delle singole empirical probabilities.
 - Ad esempio, supponiamo che ci siano i segmenti A, B, C , ognuno identificato da una regola
 - Alcune regole potrebbero attivarsi insieme, generando i segmenti AB, BC
 - Per calcolare la probabilita empirica stimata di AB si fa $\frac{n_A^+ + n_B^+}{n_A + n_B}$ (lo stesso vale per BC)
- Se si utilizza il modello a stima dell'EP per gli insiemi di istanze coperte da piu' regole, la concavita' della curva nel ROC plot non sempre e' garantita

In termini di performance i modelli a insiemi di regole sono migliori dei modelli a liste di regole, poiche' sono in grado di distinguere gli esempi coperti da una sola regola e gli esempi coperti da piu' regole. Piu' in generale, un modello a liste di n regole e' in grado di identificare n segmenti nello spazio delle istanze, mentre un modello a insieme di n regole e' in grado di identificare 2^n segmenti.

ATTENZIONE: Anche se i modelli a insiemi di regole sono piu' espressivi, non sempre possono risultare in una curva di coverage convessa. Questo perche' la stima dei segmenti sovrapposti potrebbe non essere precisa abbastanza.

- Cosa succederebbe se in un modello a insieme di regole due regole che predicono due classi differenti si attivassero insieme per la stessa istanza?
 - Si sceglie la regola mediante il criterio della massima verosimiglianza: si sceglie quella che copre piu' istanze, quindi quella con la probabilita' piu' alta
 - In caso le alternative fossero sovrapposte allora si utilizza la stima vista in precedenza (si fa la micro average)

6.4 Subgroup Discovery

- I modelli a regole possono essere anche utilizzati per descrivere **sottogruppi**
- Un sottogruppo e' un sottoinsieme nello spazio delle istanze (alternativamente sono mappe $\hat{g} : \mathcal{X} \rightarrow \{true, false\}$)

- Un sottogruppo puo' essere visto come un sottoinsieme di istanze che descrivono qualcosa di differente dal task di origine

Un buon sottogruppo ha una distribuzione di classi molto differente rispetto alla popolazione di origine

- Il criterio per valutare la bonta' di un sottogruppo e' la **deviazione della distribuzione delle classi rispetto alla popolazione d'origine**
- Per valutare i sottogruppi non si usa la *precisione* ma si usa l'*average recall* per evitare qualsiasi tipo di bias verso la classe positiva
- Nello spazio degli esempi i sottogruppi si identificano mediante dei segmenti che partono dall'origine. Piu' questi si discostano dalla bisettrice verso l'alto, piu' questo indica che il sottogruppo identificato avra' un tasso di positivi piu' alto, mentre al contrario se e' discostato verso il basso avra' un tasso di positivi molto basso rispetto ai negativi
- Una differenza sostanziale rispetto al task di classificazione e' che nel subgroup discovery ci possono essere piu' regole che coprono gli stessi esempi
- Per permettere questo comportamento, non si eliminano piu' gli esempi che sono coperti da una regola che e' stata inferita (assegnando un peso = 0) ma si *dimezzano* in peso
- L'algoritmo `WeightedCovering(D)` permette di imparare delle regole che si sovrappongono tenendo a mente questo principio

Algorithm 11: `WeightedCovering(D)` - learn overlapping rules by weighting examples

Input : Labelled training data D , with instance weights initialized to 1

Output: Rule list R

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

- L'euristica che e' guidata dal peso e' "nascosta" all'interno dell'algoritmo `BestLiteral(D, L)`

6.5 Mining di regole di associazione

- Prima di parlare di mining di regole di associazione dobbiamo parlare prima di itemsets

6.5.1 Itemsets e proprieta'

- Consideriamo il caso in cui abbiamo un certo numero di transazioni, ognuna caratterizzata da un insieme di oggetti (che sono appunto l'oggetto della transazione)
- Possiamo rappresentare l'insieme di transazioni come una matrice binaria dove l'entrata (i, j) indica la presenza dell'item j nella transazione i
- Un **ItemSet** e' un insieme di oggetti che compaiono tutti **insieme** come oggetto di una o piu' transazioni. L'insieme di transazioni che hanno questi oggetti come oggetto della transazione e' detto copertura.
- Il task del mining delle regole di associazione consiste essenzialmente nel trovare gli **ItemSet** che sono comprati frequentemente insieme
- L'insieme di tutti i possibili item sets etichettati con la loro copertura, formano un **reticolo** (nel vertice c'e' l'itemset vuoto, al basso c'e' l'itemset composto da tutti gli items - quello massimale). Cosi' come nel concept learning, possiamo utilizzare questa struttura per andare a fare il mining delle regole.
- **Supporto**: numero di transazioni da cui e' coperto l'ItemSet
- In base al supporto possiamo distinguere tre tipi di itemsets:
 - **Item set frequente**: Itemset $I = \{i_1, i_2, \dots, i_n\}$ per cui ogni item i_x appartiene ad un insieme di transazioni con cardinalita' $\geq f_0$, cioe' con supporto minimo.
 - **Item set chiuso**: Item set per cui non esiste un itemset con lo stesso supporto.
 - **Item set massimale**: E' un item set frequente e chiuso tale per non esiste un'altro itemset che sia piu' generale (superset) di esso e che abbia un supporto sufficiente.
- Si dice che il supporto ha la proprieta' di essere monotono: se ci si sposta verso il basso nel reticolo il supporto puo' solo diminuire o rimanere invariato
- Possiamo andare a trovare tutti gli itemsets frequenti andando proprio a sfruttare questa proprieta', mediante una ricerca per:
 - **Ampiezza**: Essenzialmente si verifica che il supporto sia sufficiente livello per livello. In caso sia insufficiente, si puo' evitare di andare a considerare l'intero ramo sottostante (potatura)
 - **Profondita'**: Si considerano gli itemsets andando in profondita' a vedere se il supporto e' sufficiente o meno. Ci si ferma quando questa condizione non viene soddisfatta

Algorithm 12: FrequentItems(D, f_0) - get the set of frequent itemsets

Input : Dataset D , minimum support f_0 **Output:** Set of frequent itemsets M

```
1  $M \leftarrow \emptyset$ 
2 Initialize priority queue  $Q$  in order to contain an empty ItemSet
3 while  $Q$  is not empty do
4    $I \leftarrow$  next itemset deleted from  $Head(Q)$ 
5    $max \leftarrow \mathbf{true}$ 
6   for each superset  $I'$  of  $I$  do
7     if  $Supp(I') \geq f_0$  then
8        $max \leftarrow \mathbf{false}$ 
9       add  $I'$  to the back of  $Q$ 
10    end
11  end
12  if  $max = \mathbf{true}$ 
13     $M \leftarrow M \cup \{I\}$ 
14 end
15 return  $M$ 
```

- In sostanza l'algoritmo utilizza una coda di priorit  per visitare il reticolo in profondit . Per ogni elemento si considerano tutti i sottoinsiemi e ci si assicura siano massimali dal momento che se il supporto di un itemset   troppo grande, allora viene reinserito nella coda.
- Quando l'itemset   massimale, tutti i suoi superset non hanno di conseguenza un supporto sufficiente, per cui viene inserito il nodo nell'insieme
- Quando si imposta un valore di f_0 , si crea un limite nel reticolo che in cui gli itemsets al bordo di esso sono massimali

6.5.2 Regole di associazione

- Itemsets frequenti possono essere utilizzati per creare le ***regole di associazione**
- Una regola di associazione   una regola del tipo **if B then H** dove sia **B** che **H** sono gli itemsets che compaiono frequentemente nelle transazioni insieme.
- La **confidenza** misura la forza di associazione tra **B** e **H**. In altri termini misura la probabilit  che **H** sia presente in un insieme di transazioni nel caso **B** sia presente.

$$Confidence(\text{if } B \text{ then } H) = P(H \mid B) = \frac{P(B, H)}{P(B)}$$

- L'algoritmo per minare le regole di associazione si basa su questo concetto di confidenza per andare ad ottenere le singole regole. In generale, l'idea e' quella di scegliere dall'insieme degli itemsets frequenti (creato utilizzando l'algoritmo precedente) i corpi e le teste delle regole, andando a rimuovere le regole che sono sotto un certo threshold di confidenza

Algorithm 13: FrequentItems(D, f_0) - get the set of frequent itemsets

Input : Dataset D , support threshold f_0 , confidence threshold c_0

Output: set of association rules R

```

1  $R \leftarrow \emptyset$ 
2  $M \leftarrow \text{FrequentItems}(D, f_0)$ 
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
6        $R \leftarrow R \cup \{\text{if } B \text{ then } H\}$ 
7     end
8   end
9 end
10 return  $R$ 

```

- Molte volte il mining di regole di associazione richiede una fase di post processing in cui le regole superflue vengono filtrate (casi speciali in cui non si hanno delle confidenze piu' alte del caso generale)
- Una quantita' che e' spesso usata nel post processing e' il *lift*

$$\text{Lift}(\text{if } \emptyset \text{ then } H) = \frac{n \cdot \text{Supp}(\emptyset \cup H)}{\text{Supp}(\emptyset) \cdot \text{Supp}(H)}$$

Essenzialmente un valore di lift piu' grande di 1 ci indica che $\text{Supp}(B \cup H)$ e' interamente determinato dalle frequenze marginali di $\text{Supp}(B)$ e $\text{Supp}(H)$ e non e' il risultato di un'interazione significativa tra B e H . Solo regole di associazione con un lift piu' grande di 1 sono interessanti, per cui il resto viene filtrato.

7 Modelli Lineari

7.1 Least Squares (Metodo dei minimi quadrati)

- Caso: Problema di regressione in cui si vuole ottenere un'approssimazione lineare di n punti
- Si vuole trovare l'approssimazione che minimizzi l'errore
- Formalizzazione del problema con esempio in 2 dimensioni:
 - Dati n punti (x, y) rappresentati in formato matriciale
 - Vogliamo trovare i parametri (C, D) del modello lineare $Cx + D = y$ che approssimi meglio i dati
 - Il modello e' semplicemente una combinazione lineare dei coefficienti $\sum_{i=0}^n x_i \cdot w_i = y_i$ dove w e' il vettore delle variabili.
- Idealmente quello che si vuole ottenere e' che il modello non abbia errori sui dati di train. Formalmente parlando, si vuole per ogni punto dato (x_i, y_i) valga la relazione $Cx_i + D = y_i$ (ne risulta un sistema a n righe)
- (*Recall da metodi numerici*) Ogni sistema puo' essere rappresentato in forma matriciale $Ax = b$, in questo caso si ottiene $Xw = y$, dove X e' la matrice composta da colonne degli x compresa di una colonna addizionale composta da soli 1, w il vettore colonna composto dalle variabili (in questo caso C e D) e y il vettore colonna dei valori y .
- Il **caso ideale** per la soluzione di questa equazione lineare, e' quello in cui la matrice X ha rango pieno (tutte le colonne sono linearmente indipendenti tra loro), per cui esiste l'inversa X^{-1} . La soluzione e' ottenibile moltiplicando ambo i membri per l'inversa ottenendo la soluzione $w = X^{-1} \cdot y$
- Ovviamente questo caso e' possibile solo quando il numero di osservazioni (n) coincide con il numero di features. Quasi sempre questo non succede, rendendo quindi la matrice *non invertibile*.
- Possiamo pero' riformulare il problema:
 - Ricordati che lo spazio generato da una matrice (*span*) $C(X)$ e' generato dalla combinazione lineare dei vettori colonna x_i appartenenti alla matrice
 - Variando i valori di \hat{w} (vettore colonna con tanti componenti quante features), si puo' ottenere un qualsiasi vettore all'interno di $C(X)$. Questo perche' ogni vettore nello spazio e' esprimibile come combinazione lineare delle colonne di X nel modo seguente

$$\vec{p} = \sum_{j=0}^m \hat{w}_j \vec{x}_j = X \cdot \hat{w}$$

- $\text{Dim}[C(X)] = n$ cioe' il numero di esempi

- Dal punto di vista geometrico, y non giace nello spazio $C(X)$. Dato che se no X sarebbe invertibile. Un altro modo in cui possiamo vederlo e' che lo spazio generabile da ogni combinazione delle colonne ha dimensioni pari a m , mentre il vettore y giace in uno spazio a n dimensioni. Quasi sempre $n > m$, per cui vogliamo trovare un'approssimazione nello spazio delle colonne che si avvicini il piu' possibile a y .
- y pero' e' sempre un vettore di n componenti, per cui possiamo calcolarne la distanza normata tra un qualsiasi vettore $p \in C(X)$.

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

- Dato che la norma e' una quantita' sempre positiva per definizione, possiamo anche considerarne i quadrati ($\|e\|_2^2$), che corrisponde a minimizzare la quantita' $\sum_i (y_i - p_i)^2$
- Siccome per definizione $p = X\hat{w}$, dove \hat{w} e', possiamo dare una formalizzazione finale del problema:

Il metodo dei minimi quadrati consiste nel trovare il vettore \hat{w} che minimizzi la norma al quadrato della distanza tra p e y

$$\hat{w} = \arg \min_w \|Xw - y\|_2^2$$

- (Stiamo parlando in termini dell'esempio sulle slides ovviamente) Intuitivamente parlando si vuole trovare un vettore \hat{w} tale che generi un vettore $e = y - p$ che sia perpendicolare al piano $C(X)$
- La condizione di perpendicolarita' dal punto di vista algebrico e' semplice: Siano a e b due vettori. Essi sono considerati perpendicolari quando $a \cdot b = 0$. Possiamo utilizzare questa definizione per trovare la soluzione nel nostro caso.
- Dal momento che il piano $C(X)$ e' generato dalle colonne di X , quello che vogliamo e' che il vettore e sia perpendicolare ad ogni vettore colonna x_i della matrice X . Per far cio' quindi basta imporre $X^T e = \vec{0}$.
- L'ultimo passaggio e' esplicitare e e p nella relazione precedente, ottenendo:

$$X^T e = 0$$

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

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

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

- Risolvere il problema dei minimi quadrati consiste quindi essenzialmente nel risolvere l'ultima relazione

7.1.1 Regolarizzazione

- Uno dei problemi dei minimi quadrati, e' che e' molto sensibile agli *outliers* causando overfitting. Siccome la misura dell'errore su un singolo esempio e' elevata al quadrato, gli *outliers* pesano tantissimo sulla quantita' complessiva di errore. Per compensare questa situazione, la curva di classificazione cerchera' di avvicinarsi di piu' a questo punto piu' distante, sacrificando della precisione negli altri punti.

Outliers: Punti che sono stati campionati da una distribuzione diversa da quella originale. Corrispondono a degli errori nella misurazione.

- Attenuare questo problema e' possibile mediante una **regolarizzazione** della soluzione. L'idea e' quella di aggiungere dei vincoli sulla soluzione cercata. (nel nostro caso, sulla "forma" dei pesi $w_i \in \hat{w}$).
- Una versione normalizzata del problema dei minimi quadrati e' detta **ridge regression** e consiste semplicemente nell'aggiungere un termine $\lambda ||w||^2$ per far si che il vettore non cresca troppo in dimensioni e che rimanga in norma "abbastanza piccolo":

$$\begin{aligned}\hat{w} &= \arg \min_w ||y - Xw||_2^2 + \lambda ||w||_2^2 \\ &= \arg \min_w (y - Xw)^T (y - Xw) + \lambda ||w||_2^2\end{aligned}$$

(la seconda formulazione e' solo un modo diverso per scrivere $||y - Xw||_2^2$)

- La stessa relazione e' riscrivibile in forma chiusa

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

(Il fatto di aggiungere la matrice diagonale λI a $X X^T$ e' un trucchetto molto noto che serve ad aumentare la stabilita' numerica per l'inversione della matrice)

- Un'altra forma di regolarizzazione e' data dal **lasso** (*Least Absolute Shrinkage and Selection Operator*). La differenza e' semplicemente quella che al posto di considerare $\lambda ||w||_2^2$, si usa $\lambda ||w||_1$.
- Perche' minimizzare la norma di w porta a risultati migliori?
 - Supponiamo che la matrice delle osservazioni X sia affetta da degli errori δ , cioe' $(X + \delta)$.
 - Quando si moltiplica per w , otteniamo $(X + \delta)w = Xw + \delta w$
 - In questo senso, minimizzare w minimizza anche l'errore su X
 - Un'altra ragione e' che dal momento che i w devono essere piccoli, corrisponde a scegliere dei pesi piu' semplici. (*rasoio di Occam*)
 - Un'altro modo per vedere questo e' pensare alla regolarizzazione come un bias induttivo che viene inserito per ridurre l'errore di varianza del least square

7.1.2 Classificazione con LSE

- Fin'ora abbiamo visto il metodo dei minimi quadrati per minimizzare l'errore e quindi per fare essenzialmente *regressione*, ma e' possibile adattare il modello per fare *classificazione*
- Ad esempio nel caso binario, se rappresentiamo le classi positive e negative come 1 e -1:

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

dove t rappresenta l'*intercetta*, cioe' il termine in piu' che era incluso implicitamente in precedenza in \hat{w} che veniva catturato dalla colonna composta da soli 1 di X .

- L'idea e' pensare ad un classificatore come un iperpiano divisore dei punti. Se tali punti sono sopra all'intersezione con il piano allora saranno classificati positivi, negativi altrimenti (=0 invece astengo al voto). Parlando in caso di 2D che e' piu' semplice: si vuole trovare una retta per cui tutti i punti che sono a destra della sua intersezione con l'asse x vengono classificati positivi, negativi altrimenti.
- Posso quindi utilizzare le tecniche utilizzate fin'ora per la regressione, ma utilizzando una y con valori possibili -1 o 1.

7.2 Support Vector Machines

- Idea: Invece di lasciare al caso la scelta di quale sia la linea migliore che separi i punti, do un'indicazione precisa, indicando esplicitamente che si vuole la linea di separazione che passa piu' in centro possibile tra gli esempi positivi e negativi.
- Piu' formalmente, l'obiettivo delle SVM e' quello di trovare l'iperpiano che massimizza il *minimo margine*, cioe' la distanza tra i punti e l'iperpiano che separa le classi.
- Massimizzare il *minimo margine* significa che si vogliono massimizzare i punti piu' vicini all'iperpiano
- Inoltre, si vogliono considerare tra tutte le scelte dei modelli, solo quelli che separano correttamente tutti gli esempi, per questo e' necessario imporre un'ipotesi di fondo

L'ipotesi di fondo delle SVM e' che i dati siano linearmente separabili. Cioe' che esiste una retta che separa **perfettamente** positivi e negativi senza errori.

- Possiamo quindi distinguere tra due tipi di margine:
 - Funzionale: da un'indicazione sulla correttezza della classificazione del modello
 - Geometrico: indica la distanza dei punti piu' vicini all'iperpiano che separa gli esempi

7.2.1 Margine Funzionale

- Vogliamo trovare una funzione lineare che separi gli esempi

$$f(x_i) = w \cdot x_i - t$$

in cui x_i e' un esempio, w e' il vettore dei pesi che si vuole trovare, e $x_i - t$ e' una coordinata non omogenea.

- Utilizziamo $+1$ e -1 per classificare un esempio come positivo e negativo
- Come detto in precedenza, per discriminare se un esempio e' positivo o negativo si puo' usare la regola $f(x_i) \geq 0$. Che corrisponde a trovare il punto di operativita' del modello.
- Il *marginale funzionale* dell'esempio x_i rispetto all'iperpiano determinato da w e t e' definito come

$$\mu(x_i) = y_i(w \cdot x_i - t) = y_i f(x_i)$$

in cui y_i e' il valore target (che puo' essere $+1$ o -1) moltiplicato per la funzione che separa gli esempi.

$\mu(x_i)$ e' positivo se e solo se l'esempio x_i e' correttamente classificato.

- Misura **quanto e' confidente** il modello rispetto alla classificazione di un esempio arbitrario. Siccome gli esempi in cui bisogna essere piu' cauti sono quelli vicino al punto di operativita' del modello ($f(x_i) = 0$)
- Come primo vincolo si vuole che il margine funzionale sia strettamente positivo per ogni esempio

$$y_i(w \cdot x_i - t) > 0$$

cioe' che il classificatore classifichi correttamente tutti gli esempi

- Notiamo che esiste un'infinita' di rette che soddisfano questa equazione, che sono determinate sul grado di liberta' dato da w e t . Abbiamo quindi bisogno di un vincolo piu' stringente.

Asserire che $y_i(w \cdot x_i - t) > 0$ e' equivalente ad asserire che $y_i(w \cdot x_i - t) \geq c$, dove c e' una costante arbitraria

- Secondo questa ipotesi e' possibile modificare il vincolo precedentemente imposto e richiedere che per ogni esempio x_i valga

$$y_i(w \cdot x_i - t) \geq 1$$

Inoltre imponiamo un ulteriore vincolo: sugli esempi di **frontiera** deve valere la relazione seguente

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

Gli esempi sulla frontiera sono chiamati **Support Vectors** (siccome gli esempi sono rappresentati come dei vettori), per cui la definizione precedente e' anche quella di vettore di supporto.

7.2.2 Margine Geometrico

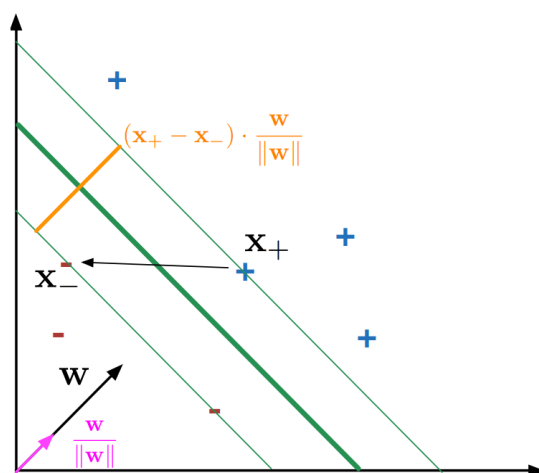


Figura 2: Margine Geometrico

- La figura precedente, mostra graficamente il margine geometrico. Esso e' rappresentato dalla distanza tra i due outliers x_+ , x_- (freccia nera).
 - Dal momento che vogliamo la loro distanza, vogliamo calcolare $x_+ - x_-$, e successivamente proiettarlo per renderlo ortogonale alla retta.
 - Sappiamo inoltre che w ha la proprieta' di essere un vettore ortogonale alla retta $y_i(w \cdot x_i - t)$ per definizione.
 - Sfruttando questa proprieta' possiamo definire il vettore proiezione come $(x_+ - x_-) \cdot \frac{w}{||w||}$, che coincide con la definizione di margine geometrico.
- Possiamo riscrivere la definizione nel modo seguente

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

Sappiamo inoltre che x_+ e x_- sono esempi nella frontiera, per cui soddisfano il vincolo prece-

dente

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

$$(-1)(x_- \cdot w - t) = 1 \rightarrow x_- \cdot w = t - 1$$

infine, sostituendo alla definizione ottenuta si ottiene

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

che e' la definizione finale di *margin geometrico*.

7.2.3 Formulazione Primale

- Avendo tutti gli ingredienti necessari, possiamo quindi definire il problema delle SVM nel modo seguente

$$\begin{aligned} & \underset{w,t}{\text{minimize}} && \frac{1}{2} \|w\|^2 \\ & \text{subject to} && y_i(w \cdot x_i - t) \geq 1; \quad 0 \leq i \leq n \end{aligned}$$

“Minimizza il *margin geometrico*, soggetto ai vincoli del *margin funzionale*”

- La forma di questo problema di ottimizzazione e' detta forma **primale**. Trovare una soluzione diretta a questo problema dal punto di vista matematico e' un problema difficile, e non esiste una soluzione in forma chiusa, per cui si utilizzano dei solver per ottenere soluzioni approssimate
- I solver per la soluzione del problema delle SVM pero' non utilizzano la forma del problema in forma primale, ma sfruttano una forma differente ma equivalente chiamata *duale*

7.2.4 Formulazione Duale

- Ogni problema di ottimizzazione, anche se concavo, puo' essere trasformato in un problema di ottimizzazione convesso equivalente.

Dato un problema di ottimizzazione in termini di minimizzazione

$$\begin{aligned} & \underset{w}{\text{minimize}} && f_0(x) \\ & \text{subject to} && f_i(x) \leq 0, \quad i = 0, \dots, m \\ & && g_i(x) = 0, \quad i = 0, \dots, p \end{aligned}$$

La corrispondente **funzione duale di Lagrange** e' per definizione

$$\begin{aligned} g(\alpha, \nu) &= \inf_x \Lambda(x, \alpha, \nu) \\ &= \inf_x \left(f_0(x) + \sum_{i=1}^m \alpha_i f_i(x) + \sum_{i=1}^p \nu_i g_i(x) \right) \end{aligned}$$

- La funzione non e' piu' in termini della variabile originale x , ma in termini di due variabili α, ν (variabili di Lagrange)
- \inf e' una generalizzazione del *minimo* per cui trova il valore piu' piccolo, anche se e' al di fuori dell'insieme originario
- Ci sono due tipi di dualita':
 - **Debole**: quando la soluzione del problema duale d^* e minore del problema primale p^* ($d^* \leq p^*$)
 - **Forte**: quando $d^* = p^*$
- Per avere soddisfatta la dualita' forte, devono essere soddisfatte le condizioni di **Karush-Kuhn-Tucker (KKT)**:
 - **Stazionarieta'**: il gradiente deve essere pari a 0 nel punto ottimale ($\nabla f_0(x^*), \nabla f_i(x^*), \nabla g_i(x^*) = 0$)
 - **Fattibilita' primale**: i vincoli del problema primale sono soddisfatti nel punto ottimale ($f_i(x^*) \leq 0, g_i(x^*) = 0$)
 - **Fattibilita' duale**: i vincoli del problema duale sono soddisfatti ($\alpha_i \geq 0, \beta_i \geq 0$)
 - **Complementary Slackness**: il prodotto tra la variabile di Lagrange e il vincolo primale deve essere sempre uguale a 0 ($\alpha_i f_i(x^*) = 0$ dove x^* e' il punto ottimale)
- Data la definizione di dualita' di Lagrange, possiamo quindi ottenere la forma duale corrispondente del problema delle Support Vector Machines

$$\begin{aligned} \Lambda(w, t, \alpha_1, \dots, \alpha_n) &= \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i (y_i (w \cdot x_i - t) - 1) \\ &= \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i y_i (w \cdot x_i) + \sum_{i=1}^n \alpha_i y_i t + \sum_{i=1}^n \alpha_i \\ &= \frac{1}{2} w \cdot w - w \cdot \left(\sum_{i=1}^n \alpha_i y_i x_i \right) + t \left(\sum_{i=1}^n \alpha_i y_i \right) + \sum_{i=1}^n \alpha_i \end{aligned}$$

- In sostanza si aggiunge una variabile di Lagrange (α_i) per ogni vincolo nel problema originale, per cui n (poiche' ci sono tanti vincoli quante istanze)

- Per ottenere il problema duale finale, vogliamo il minimo di Λ , per cui deriviamo l'intera quantità rispetto a t e w e le poniamo successivamente a 0 (vettore di 0)

$$\begin{aligned}\frac{\partial \Lambda}{\partial t} &= \sum_i \alpha_i y_i & \frac{\partial \Lambda}{\partial w} &= w - \sum_i \alpha_i y_i x_i \\ \sum_i \alpha_i y_i &= 0 & w &= \sum_i \alpha_i y_i x_i\end{aligned}$$

- Sostituendo le due uguaglianze trovate possiamo infine ottenere la formulazione del problema duale delle support vector machines:

$$\begin{aligned}\underset{\alpha}{\text{maximise}} \quad & -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j x_i x_j + \sum_{i=1}^n \alpha_i \\ \text{subject to} \quad & \alpha_i \geq 0, \quad i = 1, \dots, n \\ & \sum_i y_i \alpha_i = 0\end{aligned}$$

- Guardando il problema duale, abbiamo guadagnato diversi insights sul problema:
 1. I moltiplicatori di Lagrange strettamente positivi sono sempre associati a vettori di supporto, per cui i moltiplicatori per gli altri esempi sono uguali a 0
 2. w e' ottenuto come combinazione lineare dei vettori di supporto (per questa ragione possiamo utilizzare il kernel trick!)
 3. Sia l'apprendimento che la classificazione puo' essere fatta utilizzando soltanto prodotti scalari dei vettori di supporto
- La 1. segue dalle condizioni di KKT, per cui $\alpha_i(y_i(w \cdot x_i - t) - 1) = 0$ deve valere. Ma poiche' i vincoli impongono che $\alpha_i > 0$, per rendere vera la condizione deve valere $(w \cdot x_i - t) = 1$ che e' appunto la definizione di margine funzionale (per cui x_i e' un vettore di supporto)
- Perche' comunque utilizziamo una forma duale e non primale dal momento che entrambi sono problemi di ottimizzazione convessi e possono essere risolti con lo stesso algoritmo?

La forma duale del problema delle SVM puo' essere utilizzata (mediante kernel trick) per apprendere funzioni *non lineari* all'interno dello spazio degli esempi. Per cui puo' anche non valere l'ipotesi iniziale della separabilita' perfetta

7.2.5 Ammettere errori nel margine

7.2.5.1 Forma Primale

- Il problema formulato fin'ora non ammette gli errori nel margine funzionale. Cio' significa che se il vincolo $y_i(w \cdot x_i - t) \geq 1$ non viene soddisfatto la soluzione non viene considerata
- Quello che si vuole ottenere e' un errore nel margine, dipendente da ogni istanza x_i

$$y_i(w \cdot x_i) - t \geq 1 - \xi_i$$

L'idea e' quindi quella di introdurre una **variabile di slack** ξ_i per ogni esempio

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

- Tramite l'ultimo vincolo si ottiene un rilassamento del vincolo del margine funzionale ma il rilassamento e' penalizzato nella funzione obiettivo, inserendo la una costante moltiplicativa dell'errore complessivo C (che e' un parametro definito dall'utente)
- C : **costante di complessita'**, ci dice quanto vogliamo penalizzare l'errore totale sul margine. Controlla la complessita' del sistema.

7.2.5.2 Forma Duale

- Anche in questo caso possiamo ottenere una formulazione duale del problema. Per prima cosa calcoliamo Λ

$$\begin{aligned} \Lambda(w, t, \xi_i, \alpha_i, \beta_i) &= \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i (y_i(w \cdot x_i - t) - (1 - \xi_i)) - \sum_{i=1}^n \beta_i \xi_i \\ &= \frac{1}{2} \|w\|^2 + \sum_{i=1}^n C \xi_i - \sum_{i=1}^n \alpha_i (y_i(w \cdot x_i - t) - 1) - \sum_{i=1}^n \alpha_i \xi_i - \sum_{i=1}^n \beta_i \xi_i \\ &= \Lambda(w, t, \alpha_i) + \sum_{i=1}^n C \xi_i - \sum_{i=1}^n \alpha_i \xi_i - \sum_{i=1}^n \beta_i \xi_i \\ &= \Lambda(w, t, \alpha_i) + \sum_{i=1}^n \xi_i (C - \alpha_i - \beta_i) \end{aligned}$$

- Siccome le derivate rispetto a w e t di Λ si conoscono gia', rimane da derivare $E(\xi_i) = \sum_{i=1}^n \xi_i (C - \alpha_i - \beta_i)$ rispetto a ξ_i , per cui si ottiene

$$\frac{\partial E}{\partial \xi_i} = C - \alpha_i - \beta_i$$

settando la derivata calcolata a 0, si ottiene che

$$\Lambda(w, t, \xi_i, \alpha_i, \beta_i) = \Lambda(w, t, \alpha_i)$$

- Ottenendo infine tutti gli ingredienti necessari a scrivere la formulazione duale del problema delle SVM con i vincoli rilassati

$$\begin{aligned} \underset{\alpha}{\text{maximise}} \quad & -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j x_i x_j + \sum_{i=1}^n \alpha_i \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \dots, n \\ & \sum_i y_i \alpha_i = 0 \end{aligned}$$

- Una delle condizione KKT implica che $\beta_i \xi_i = 0$ per cui cio' ci dice che se $\xi_i > 0$ (ho un errore nel margine), allora segue che $\beta_i = 0$. Sostituendo le uguaglianze si ottiene $C - \alpha_i - \beta_i = 0 \rightarrow \alpha_i = C$. Seguendo la stessa linea di ragionamento possiamo concludere che:
 - $\alpha_i = C$ sono errori del margine
 - $\alpha_i = 0$ sono esempi al di fuori del margine
 - $0 < \alpha_i < C$ sono vettori di supporto

7.3 Kernels

- Il cosiddetto *kernel trick* e' un metodo per estendere i modelli lineari a problemi non lineari

Una *funzione kernel* e' una funzione $K : \mathbb{V} \times \mathbb{V} \rightarrow \mathbb{R}$ che prende in input una coppia di vettori e restituisce in output un valore reale e per cui vale la condizione che esiste una mappa $\phi(x) : \mathbb{V} \rightarrow \mathbb{F}$ per cui vale

$$K(x, y) = \langle \phi(x), \phi(y) \rangle$$

cioe' che corrisponde al prodotto scalare dei valori mappati dentro uno *spazio di Hilbert*. In altri termini, K prende una coppia di vettori, li mappa in uno spazio di Hilbert e ne calcola il prodotto scalare

- Una funzione $\langle \cdot, \cdot \rangle$ e' un **prodotto interno** se soddisfa le seguenti proprieta':
 1. $\langle x, y \rangle = \langle y, x \rangle$ (*Simmetria*)
 2. $\langle ax + by, z \rangle = a \langle x, z \rangle + b \langle y, z \rangle$ (*Linearita' sul primo argomento*)
 3. $\langle x, x \rangle \geq 0$; $\langle x, x \rangle = 0 \Leftrightarrow x = 0$ (*Definizione positiva*)
- Il prodotto interno e' una generalizzazione del prodotto scalare dal momento che soddisfa le proprieta' del prodotto interno

- Ci sono diverse ragioni per utilizzare i kernels:
 - Rappresentazionali: si possono estendere i classificatori a problemi non lineari
 - Computazionali: il calcolo del prodotto interno nel nuovo spazio viene spesso effettuata senza dover fare la proiezione in modo esplicito, per cui costa meno computazionalmente
 - Teorici: i kernel hanno un sacco di proprietà interessanti, ad esempio è possibile comporli tra di loro
- Alcuni Kernel importanti sono:
 - Kernel Polinomiale di grado d ($K(x, y) = (x \cdot y)^d$ oppure $(x \cdot y + 1)^2$): mappa i punti in uno spazio in cui le rette (classificatori) corrispondono a dei polinomi di grado d nello spazio di origine (come nell'esempio del video)
 - * Se $d = 1$ allora abbiamo un Kernel Lineare (identità)
 - * Se $d = 2$ allora abbiamo un Kernel quadratico
 - Kernel Gaussiano ($K(x, y) = \exp(-\frac{\|x-y\|^2}{2\sigma^2})$): impiega una funzione Gaussiana per determinare la similarità di esempi. Decresce esponenzialmente seguendo una curva gaussiana man mano che ci si allontana da un esempio
- Molti altri kernel esistono e possono essere creati in base al dominio del problema, poiché ci permettono di manipolare dati la cui forma non potrebbe essere utilizzata con le SVM.
- Intuitivamente un kernel può essere pensato come una *funzione di similarità* nello spazio delle features
 - Infatti, vettori “simili” tra di loro, puntano nella stessa direzione, per cui il loro prodotto scalare sarà molto grande (molto simili)
 - Contrariamente, se sono poco simili tra di loro, punteranno in direzioni sempre più opposte, rendendo il prodotto scalare piccolo
- Sapendo che un kernel è una funzione di dissimilarità, come fa uno a sapere se una data funzione di dissimilarità è un Kernel? Cioè, data una funzione $K(x, y)$, come possiamo verificare se essa rispetta le condizioni necessarie ad essere un kernel?
- La risposta sta nelle **condizioni di Mercer**:

Una funzione K è un kernel valido se e solo se per ogni insieme finito di punti $\{x_1, \dots, x_m\}$, la matrice K (definita come $J_{i,j} = K(x_i, x_j)$) è **simmetrica** e **positiva semi-definita**.

Una matrice $M^{n \times n}$ è detta semidefinita positiva se $\forall \vec{u} \in \mathbb{R}^n$ vale la seguente relazione

$$\vec{u}^T M \vec{u} \geq 0$$

Mentre una matrice $M^{n \times n}$ e' detta **simmetrica** se

$$M = M^T \rightarrow M^{-1}M^T = I$$

- Un proprieta' molto importante dei kernel e' che si possono combinare facilmente:
 - $K(x, y) = K_1(x, y)K_2(x, y)$ (Approssima l'operazione di **AND**)
 - $K(x, y) = K_1(x, y) + K_2(x, y)$ (Approssima l'operazione di **OR**)
 - $K(x, y) = \alpha K_1(x, y)$ per $\alpha > 0$
 - $K(x, y) = f(x)f(y)$, per ogni funzione f
- Questo tipo di manipolazioni sono interessanti perche' se si ha un dataset in cui gli esempi sono descritti con attributi categorici molto diversi tra loro, e' possibile raggruppare ogni categoria con un kernel e combinarli tra loro a seconda dell'intuizione di come vadano combinati

8 Modelli basati sulla distanza

- I modelli per la classificazione sfruttano le similarita' tra i dati di training per generalizzare a dati mai visti. Ad esempio, gli alberi di decisione segmentano lo spazio degli esempi in segmenti simili in qualche modo tra di loro.
- I modelli che vedremo utilizzano una forma piu' graduata di similarita'
- Notiamo che in uno spazio degli esempi, la scelta piu' banale come misura di dissimilarita' potrebbe essere la **distanza** tra due esempi
- In generale esistono pero' molte funzioni di distanza che si basano su altri principi rispetto a quelli di "senso comune"

8.1 Distanze di Minkowski

Se $\mathcal{X} = \mathbb{R}^d$ e' lo spazio degli esempi, la **distanza di Minkowski** di ordine $p > 0$ e' definita come

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

dove $x, y \in \mathcal{X}$. In altri termini, e' semplicemente la norma p -esima della differenza tra gli esempi.

- Dis_0 misura la distanza contando quanti elementi sono diversi da 0 all'interno di un vettore. Possiamo vederla come $Dis_0(x, y) = \sum_{j=1}^d I[x_j \neq y_j]$.
 - Se tutte le features di x e y sono binarie, allora si parla di **distanza di Hamming**
 - Se tutte le features di x e y non sono binarie e non hanno la stessa lunghezza, si parla invece di **distanza di Levenshtein**
- Dis_2 misura la **distanza euclidea**, che indica la distanza piu' corta tra due punti connessi da una linea retta (*a volo d'uccello*)
- Dis_1 e' detta **distanza di Manhattan** (*cityblock distance*), che determina la distanza tra due punti se fossero permessi solo movimenti lungo gli assi delle coordinate
- Mano a mano che si fa crescere p , la misura della distanza sara' sempre piu' dominata dalla componente piu' grande in valore assoluto. All'estremo troviamo $Dis_\infty = \max_j |x_j - y_j|$, chiamata anche **distanza di Chebyshev**.
- In alcuni casi in cui le features sono sparse (cioe' se alcune features sono assenti su alcune istanze) si utilizza di solito la **distanza di Jaccard**, che tiene conto solo delle istanze che sono entrambe presenti (entrambe 1)

Una **metrica di distanza** e' una funzione $Dis : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ tale che per ogni $x, y, z \in \mathcal{X}$ valgono le seguenti proprieta':

- La distanza tra un punto e se stesso e' 0: $Dis(x, x) = 0$
- Tutte le altre distanze sono maggiori di zero: se $x \neq y$ allora $Dis(x, y) > 0$
- La distanza e' simmetrica: $Dis(x, y) = Dis(y, x)$
- Percorsi alternativi non possono diminuire la distanza: $Dis(x, z) \leq Dis(x, y) + Dis(y, z)$

8.2 Distanze di tipo ellittico

- La distanza ellittica e' un modo differente di calcolare la distanza che tiene conto anche della direzione nello spazio in cui rappresentiamo le istanze
- Un tipo di distanza ellittica e' la distanza di Mahalanobis

- Sia $M = \Sigma^{-1}$ la *matrice della covarianza*, allora la distanza di Mahalanobis e' calcolabile come

$$Dis_M(x, y \mid \Sigma) = \sqrt{(x - y)^T \Sigma^{-1} (x - y)}$$

- Utilizzare la matrice della covarianza ha l'effetto di *eliminare la correlazione e normalizzare* le features
- Nella formula, Σ determina la forma dell'ellisse
- La distanza Euclidea e' un caso specifico della distanza di Mahalanobis, in particolare $Dis_2(x, y) = Dis_M(x, y, \mid I)$

- Spieghiamo ora come si calcola la matrice delle covarianze Σ

- Abbiamo che X e' una matrice $(n \times d)$ composta da n oggetti rappresentati mediante d features
- Allora la matrice delle covarianze e' una matrice $d \times d$, in cui la singola entrata σ_{lj} e' la *covarianza* tra le feature l e j su tutte le istanze nel dataset X

$$\sigma_{lj} = \frac{1}{n} \sum_{k=1}^n (x_{kl} - \bar{x}_{*l})(x_{kj} - \bar{x}_{*j})$$

- σ_{lj} e' una misura di quanto le features l e j variano **in relazione tra loro** (da qui il nome *co-varianza*). Ovviamente la diagonale della matrice di covarianza corrisponde alla varianza delle features stesse

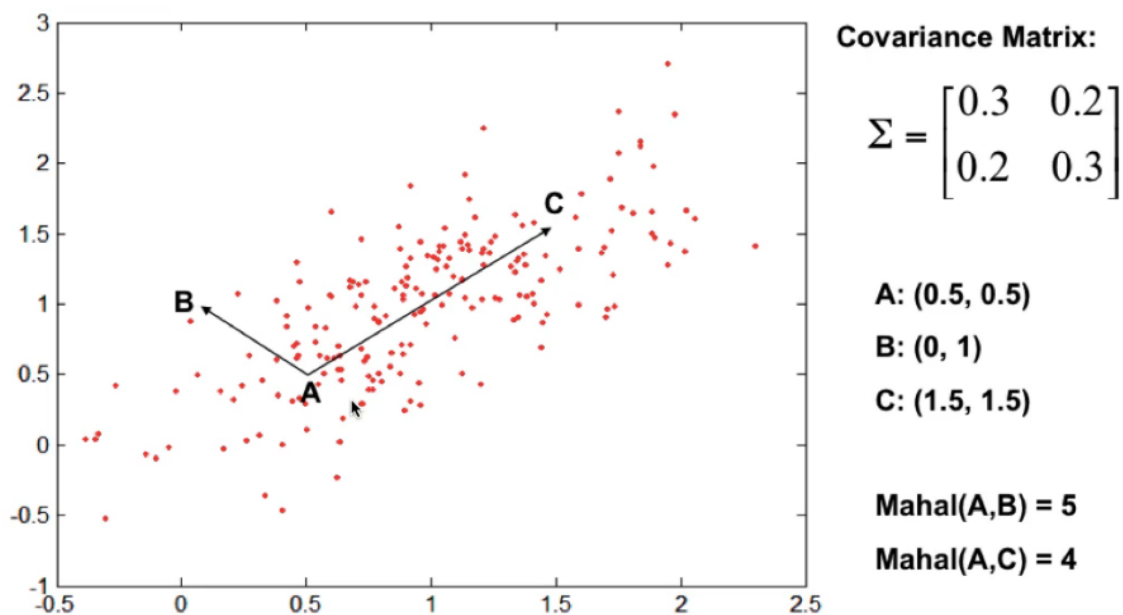


Figura 3: Effetto della distanza di Mahalanobis

- In questa immagine, si può notare come la distanza di Mahalanobis sia in disaccordo con la distanza Euclidea. Si può notare come sia più piccola tra A e C rispetto a A e B . Questo accade principalmente perché la direzione del vettore differenza $C - A$ segue la direzione dei dati.

8.3 Centroidi e Medoidi

- Gli esemplari sono delle istanze nello spazio degli esempi che rappresentano (sono rappresentative) delle classi

Teorema: La media aritmetica μ di un set di esemplari D è il punto univoco che *minimizza la somma totale* delle distanze Euclidee al quadrato tra tutte le istanze del set D e μ

Dimostrazione: Bisogna dimostrare che $\arg \min \sum \|x - y\|_2^2 = \mu$, imponendo che il *gradiente* della somma rispetto a y (∇_y) sia 0

$$\nabla_y \sum_{x \in D} \|x - y\|_2^2 = -2 \sum_{x \in D} (x - y) = -2 \sum_{x \in D} x + 2|D|y = y = \frac{1}{|D|} \sum_{x \in D} x = \mu$$

- In alcune situazioni non si vuole avere un *baricentro* ideale (in questo caso rappresentato da μ , che però non è descritto in termini di features) ma si vuole un'istanza che renda minime le distanze tra tutti gli altri. In questo caso parliamo di **medoide** se l'esempio è ristretto a far parte

esclusivamente del dataset, mentre **centroide** quando puo' anche non appartenere per forza al dataset.

- **Centroide**: Centro della massa ideale di una classe (puo' anche non appartenere al dataset)
- **Medoide**: Istanza localizzato piu' al centro della classe (piu' vicina al centroide)
- Rispetto al calcolo del centroide, il medoide richiede di calcolare le distanze tra tutte le coppie di punti, per cui ha complessita' $O(n^2)$

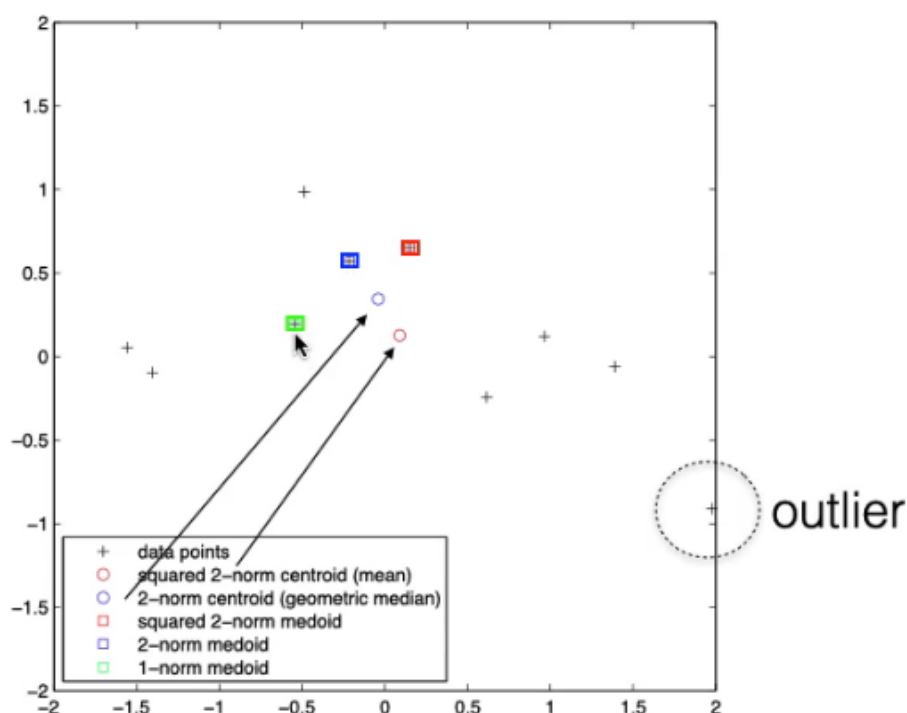


Figura 4: Esempio di dataset con 10 punti in cui i cerchi sono centroidi e i quadrati medoidi. Si noti come l'outlier ("spinga") i centroidi verso di se

8.4 Distance Based Classification

- Sappiamo che un classificatore lineare di base costruisce una linea di decisione dividendo i positivi e i negativi
- E' possibile ottenere la stessa cosa ma basandosi sul concetto di distanza:
 - Siano $\mu^{\ominus}, \mu^{\oplus}$ i centroidi delle rispettive classi negative e positive
 - Quando una nuova istanza x deve essere classificata, si controlla
 - * Se e' piu' vicina a μ^{\oplus} allora e' un esempio **positivo**

* Altrimenti e' un esempio **negativo**

- In altri termini, classifica un'istanza con la classe del piu' vicino *esemplare*
- In caso usassimo la distanza Euclidea, allora si otterrebbe lo stesso ed identico decision boundary ottenuto dal classificatore lineare

Creare un classificatore lineare puo' essere interpretato da un punto di vista di distanza come trovare gli *esemplari* che minimizzino la distanza euclidea al quadrato di ogni classe, per poi applicare una regola di decisione basata sull'esemplare piu' vicino.

- Questo cambio di prospettiva permette di estendere la classificazione a piu' di due classi molto facilmente.
- All'aumentare di esemplari (conseguentemente ad un aumento delle classi) alcune regioni dello spazio degli esempi diventano delle regioni convesse chiuse (delimitate dai decision boundaries), dando luogo a quella che si chiama **tassellazione di Voronoi**.

8.4.1 Nearest Neighbour Classifier

- Nella sua forma originale, il classificatore che si basa sui k -vicini piu' prossimi, prende un *voto di classe* per ognuno dei k esemplari piu' vicini e ne predice la classe maggioritaria

Scegliere un k dispari e' preferibile in modo da evitare *pareggi* nei voti

- **kNN** memorizza (infatti si chiamano anche modelli basati sulla memoria) tutti gli esempi del dataset e ad ogni esempio di test va a cercare le k istanze piu' vicine nel dataset memorizzato, per poi farne la votazione di classe.
- Per questa ragione, sia la *classificazione di una singola istanza* che il *training del modello* ha complessita' $O(|D|)$
- Il classificatore 1NN separa perfettamente i positivi e i negativi, per cui ha un **basso bias** e un **elevata varianza** (molto suscettibile a overfitting in casi in cui il dataset e' poco significativo o poco grande)
- D'altra parte, all'aumentare di k **diminuiamo la varianza** e **aumentiamo il bias** del modello.
- Relazione col *bias-variance dilemma*:
 - Con valori bassi di k , abbiamo una varianza alta e un basso bias
 - Con valori alti di k , abbiamo una bassa varianza e un alto bias
- **Pro**: Facilmente adattabile a valori di target reali per cui si puo' estendere facilmente a task di regressione, o alla stima di probabilita' quando $k > 1$
- **Contro**: E' affetto dalla maledizione della dimensionalita', cioe' che in spazi con dimensionalita' molto alta gli esempi sono molto distanti tra di loro per cui la distanza e' poco informativa (?)

Non esiste una regola precisa per trovare il valore di k ottimale per un dato dataset

- Una modifica che si può fare a kNN è quella di **pesare il voto** di un certo vicino per il reciproco della distanza tra il vicino e l'istanza di test. In questo modo, più è grande la distanza minore sarà il voto, il che coincide con l'intuizione dal momento che più è distante e' l'istanza, meno affidabile sarà la sua classe
- In generale, tale modifica si fa perché diminuisce l'effetto dell'aumento dell'errore dovuto all'aumento del bias quando il numero di k si fa salire
- Possiamo inoltre estendere kNN per andare a fare regressione semplicemente andando a predire il valore di target come la media dei valori dei k vicini pesata per l'inverso della distanza tra gli stessi e l'istanza. (proprio come nei weighted voting kNN)

8.5 Distance Based Clustering

8.5.1 DBSCAN

- DBSCAN è un algoritmo basato sulla *densità*, definita come il numero di punti all'interno di un raggio specificato ϵ
- Un punto è detto:
 - **Core point** se ha più di un certo numero **MinPts** all'interno del raggio specificato ϵ (punto interno al *cluster*)
 - **Border point** se ha meno di **MinPts** punti all'interno del raggio ϵ , ma è un *vicino* di un *core point*
 - **Noise point** se non è nessuno dei due
- Sfruttando queste definizioni si può definire lo pseudodice di DBSCAN:

Algorithm 14: DBSCAN

Input : Dataset D

Output: Labelled Dataset D

- 1 $\forall x \in D$ label x as a core, border or noise point
 - 2 Eliminate every noise point
 - 3 Put an edge between all core point that are within ϵ of each other
 - 4 Make each group of connected core points into a separate cluster
 - 5 Assign each border point to one of the clusters of its associated core points
-

- In generale DBSCAN riesce a rappresentare bene clusters di qualsiasi forma e dimensione, posto che le densità siano particolarmente omogenee e che formino zone contigue. Ad esempio, se si dovessero riconoscere le figure all'interno di una immagine in grayscale.

- DBSCAN si comporta particolarmente male in casi in cui ci sono datasets che presentano delle densita' molto differenti tra loro, per cui risulterebbe difficile far riconoscere i clusters giusti all'algoritmo.
- Inoltre anche DBSCAN soffre della maledizione dell'alta dimensionalita', per cui si comporta male anche in datasets caratterizzati da dati a dimensioni elevate. (questo perche' anch'esso si basa su un concetto di distanza)
- Una difficolta' di DBSCAN e' quella dell'inizializzazione dei parametri **MinPts** e **Eps** (ϵ).
 - Cio' che si puo' fare per risolvere questo problema e' analizzare il **Reachability Plot**, che si ottiene plottando la densita' di punti in base alla loro distanza dai k vicini fissati
 - Sull'asse delle y si mette la *distanza* che puo' variare
 - Sull'asse delle x si mette la *densita'* di punti
 - Un punto (x, y) rappresenta l' x numero di punti che hanno almeno k vicini con quella distanza y
 - Se si ordinano i punti per densita' e si plottano, si ottiene una curva che ha con valori crescenti di x un *gomito*.
 - I punti nella prossimita' del gomito indicano dei valori ottimali da impostare per la distanza ϵ

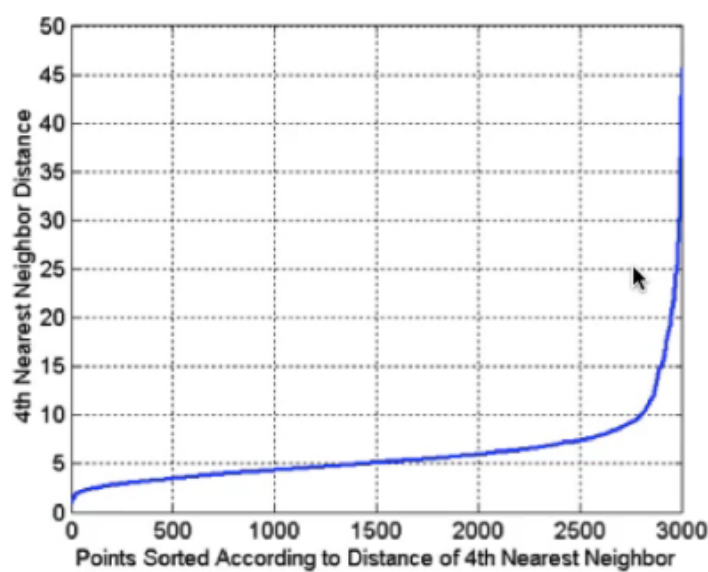


Figura 5: Plot per parametrizzare DBSCAN con $k = 4$. Sull'asse delle x c'e' il *numero* di punti, mentre sull'asse delle y la *distanza*. Un punto (x, y) rappresenta l' x numero di punti che hanno almeno 4 vicini all'interno di un cerchio di raggio y

8.5.2 K-Means

- **Nota:** Qui ho saltato una parte sulle misure dei clusters che è stata già spiegata nella parte dei clusters ad albero
- Possiamo notare che la SSE di un dataset D è costante e può essere scomposta in due sottoparti $SSE = WSS + BSS$
- Al variare del numero K di clusters (non utilizzeremo più k per indicare i vicini come fin'ora, ma per indicare il numero di clusters), le quantità WSS e BSS cambieranno ma la loro somma rimane sempre la stessa costante
- Il problema *k-means* consiste nel trovare K clusters in un dataset D tali che minimizzino la quantità WSS (o che massimizzino BSS), cioè che i clusters siano ben coesi
- In altri termini, si tratta di minimizzare la funzione obiettivo WSS
- Identificare K clusters si può anche vedere come trovare k centroidi (siccome ogni centroide individua un cluster), per cui da qui è preso il nome *K-means*
- Questo problema è *NP-Completo*, cioè non esiste una soluzione analitica chiusa, per cui si ricorre (come già fatto in precedenza) ad algoritmi che utilizzano euristiche per trovare un'approssimazione del minimo/massimo globale
- Un algoritmo molto noto che risolve il problema è l'**algoritmo di Lloyd**:
 - Essenzialmente itera tra due fasi:
 - * Partiziona i dati utilizzando la regola del centroide più vicino
 - * Ricalcola di conseguenza il centroide di ogni partizione
 - È un algoritmo molto efficiente che converge molto velocemente ad un punto stazionario, ma non si ha la garanzia che la soluzione sia effettivamente un minimo/massimo globale
 - Per questa ragione l'algoritmo viene eseguito più volte per poi scegliere la migliore tra tutte le soluzioni

Algorithm 15: KMeans(D, K) - K-means clustering using Euclidean distance Dis_2

Input : Dataset D , number of clusters K

Output: K cluster means $\mu_1, \dots, \mu_K \in \mathbb{R}^d$

```

1 randomly initialize  $K$  vectors  $\mu_1, \dots, \mu_K \in \mathbb{R}^d$ 
2 repeat
3   assign each  $x \in D$  to  $\arg \min_j Dis_2(x, \mu_j)$ 
4   for  $j = 1$  to  $K$  do
5      $D_j \leftarrow \{x \in D \mid x \text{ assigned to cluster } j\}$ 
6      $\mu_j = \frac{1}{|D_j|} \sum_{x \in D_j} x$ 
7   end
8 until no change in  $\mu_1, \dots, \mu_K$ 
9 return  $\mu_1, \dots, \mu_K$ 

```

- K-Means ha uno svantaggio particolare: e' sensibile ai clusters che hanno grandezza e forma non globulare per cui tende a dividerle in modo errato. Questo perche' il metodo ottimizza i centroidi del cluster, generando una tassellazione di Voronoi nello spazio delle istanze.
- Per attenuare il problema si suggerisce una normalizzazione delle features nello stesso range in modo che le features abbiano tutte lo stesso impatto sulla distanza
- Una variante dell'algoritmo *K-Means* e' *K-Medoids*, che al posto di cercare i **centroidi** cerca i **medoidi** all'interno del dataset per rappresentare i clusters.

Algorithm 16: KMedoids(D, K, Dis) - K-medoids clustering using arbitrary distance metric Dis

Input : Dataset D , number of clusters K

Output: K medoids $\mu_1, \dots, \mu_K \in D$

```

1 randomly pick  $K$  datapoints  $\mu_1, \dots, \mu_K \in D$ 
2 repeat
3   assign each  $x \in D$  to  $\arg \min_j Dis(x, \mu_j)$ 
4   for  $j = 1$  to  $K$  do
5      $D_j \leftarrow \{x \in D \mid x \text{ assigned to cluster } j\}$ 
6      $\mu_j = \arg \min_{x \in D_j} \sum_{x' \in D_j} Dis(x, x')$ 
7   end
8 until no change in  $\mu_1, \dots, \mu_K$ 
9 return  $\mu_1, \dots, \mu_K$ 

```

8.5.3 Silhouettes

- La **Silhouette** e' un valore di scoring per valutare la bonta' di una soluzione di clustering

- Denotiamo con $d(x_i, D_j)$ il valore medio delle distanze dell'istanza x_i con tutti i punti del cluster D_j .
- Denotiamo inoltre $j(i)$ l'index del cluster a cui appartiene x_i
- Possiamo allora definire:
 - $a(x_i) = d(x_i, D_{j(i)})$ che indica distanza media di x_i con tutti i *punti interni al proprio cluster*
 - $b(x_i) = \min_{k \neq j(i)} d(x_i, D_k)$ che indica la distanza media rispetto ai *punti del cluster piu' vicino*
- La differenza $b(x_i) - a(x_i)$ puo' essere presa come misura di quanto sia ben clusterizzato x_i
 - Se $a(x_i) > b(x_i)$ allora la differenza e' negativa, per cui descrive una situazione in cui x_i in media i membri del cluster piu' vicino sono piu' vicini rispetto ai membri del proprio cluster
 - Se $a(x_i) < b(x_i)$ allora la differenza e' positiva, per cui indica che x_i in media e' ben clusterizzato
- Possiamo infine definire un *clustering score* per la singola istanza x_i come

$$s(x_i) = \frac{b(x_i) - a(x_i)}{\max(a(x_i), b(x_i))}$$

- $s(x_i)$; positivo = buona clusterizzazione, negativo = cattiva clusterizzazione (in generale piu' grande e' meglio e')
- Un plot di una **Silhouette** consiste nel calcolare i valori $s(x)$ per ogni istanza, ordinarli e raggrupparli per cluster

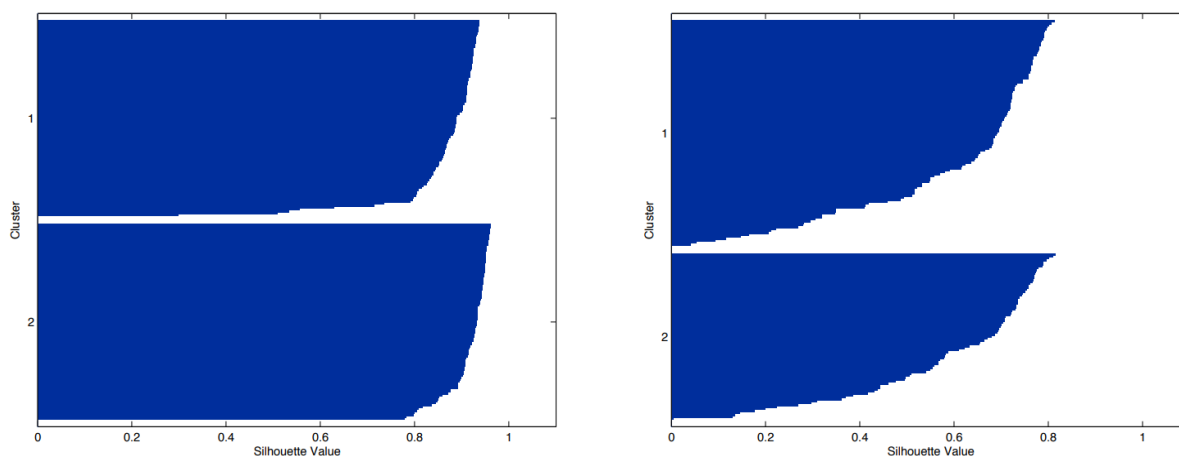


Figura 6: Silhouette per due clustering ottenuti. La prima (sinistra) e' chiaramente migliore della seconda (destra)

8.6 Clustering Gerarchico

- Il risultato di un clustering gerarchico e' un *dendrogramma*, cioè una struttura gerarchica ad albero
- Un vantaggio del dendrogramma e' che potrebbe rappresentare strutture che sono significative nel dominio di applicazione (esempio: mondo animale), per cui il clustering gerarchico e' molto popolare domini di applicazione biologici
- Dato un dataset D un **dendrogramma** e' un albero binario con gli elementi di D come sue foglie. I nodi interni dell'albero identificano un sottoinsieme di D composto dai nodi foglia che hanno come radice quel dato nodo interno
- **Funzione di Linkage**: E' una funzione $L : 2^{\mathcal{X}} \times 2^{\mathcal{X}} \rightarrow \mathbb{R}$ che calcola la distanza tra due sottoinsiemi arbitrari nello spazio degli esempi, data una metrica di distanza $Dis : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$. Funzioni di linkage molto comuni sono:

- **Single linkage**: definisce la distanza tra due clusters come la piu' piccola distanza tra gli elementi dei due clusters

$$L_{single}(A, B) = \min_{x \in A, y \in B} Dis(x, y)$$

- **Complete linkage**: definisce la distanza tra due clusters come la distanza piu' grande tra gli elementi dei due clusters

$$L_{complete}(A, B) = \max_{x \in A, y \in B} Dis(x, y)$$

- **Average linkage**: definisce la distanza tra due clusters come la distanza *media* punto-punto

$$L_{average}(A, B) = \frac{\sum_{x \in A, y \in B} Dis(x, y)}{|A| \cdot |B|}$$

- **Centroid linkage**: definisce la distanza tra due cluster come la distanza tra i due centroidi dei rispettivi clusters

$$L_{centroid}(A, B) = Dis\left(\frac{\sum_{x \in A} x}{|A|}, \frac{\sum_{y \in B} y}{|B|}\right)$$

- Tutte queste funzioni di linkage divergono per cluster molto grandi, mentre convergono tutte allo stesso valore per cluster singoli (composti da un solo elemento)
- Un algoritmo che performa il clustering agglomerativo ipotetico e' il seguente:

Algorithm 17: HAC(D,L) - Hierarchical agglomerative clustering

Input : Dataset D , linkage function L

Output: A dendrogram representing a descriptive clustering of D

- 1 Initialize cluster 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
-

- Utilizzando la funzione di linkage, e' evidente come il focus nel clustering gerarchico si sposti sul *BSS*

8.7 Dai kernels alle distanze

- Possiamo vedere un kernel come una funzione di similarita' tra due istanze
- Ricordiamo che un kernel e' una funzione $K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$ che calcola il prodotto scalare senza costruire esplicitamente il vettore $\phi(x)$
- Ogni metodo di learning che puo' essere espresso solamente in termini di prodotto scalare e' *kernelizzabile*
 - Possiamo fare lo stesso con molti modelli basati sulla distanza, siccome la distanza Euclidea puo' essere riscritta come

$$Dis_2(x, y) = \|x - y\|^2 = \sqrt{(x - y) \cdot (x - y)} = \sqrt{x \cdot x - 2x \cdot y + y \cdot y}$$

- Se sostituiamo il prodotto scalare con la funzione kernel, possiamo costruire la generica distanza kernelizzata

$$Dis_K(x, y) = \sqrt{K(x, x) - 2K(x, y) + K(y, y)}$$

- Possiamo trasformare il prodotto scalare in distanza anche in un altro modo:
 - Dal momento che il prodotto scalare $x \cdot y = \|x\| \cdot \|y\| \cos \theta$, possiamo derivare la *similarita' del coseno* come:

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

- Anche in questo caso possiamo kernelizzarla

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

9 Modelli Probabilistici

- Lo strumento principale per fare inferenza probabilistica e' il teorema di Bayes

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

Dove:

- $P(Y = y | X)$ e' detta **probabilita' a posteriori** (cioe' stimata *dopo* aver visto l'esempio X), e indica la probabilita' che la classe di una data istanza X sia y
 - $P(X | Y = y)$ e' detta **likelihood**, e indica la probabilita' di ottenere un'istanza X se ipotizziamo di campionarla dalla classe y
 - $P(Y = y)$ e' detta **probabilita' a priori**, e indica la probabilita' di una classe, senza conoscenze pregresse sulle features (quindi considera solamente la distribuzione delle classi nel dataset)
 - $P(X)$ e' anch'essa una **probabilita' a priori**, ma in questo caso sulle features osservate. E' la probabilita' di ottenere l'istanza X se presa un'istanza a caso all'interno del training set
- X e' un vettore **aleatorio** che rappresenta un singolo esempio in termini dei valori delle sue features
 - Anche Y e' una variabile aleatoria che indica invece il valore di target
 - Quando si fanno delle inferenze di tipo probabilistico sul valore di target, uno dei metodi che si utilizzano piu' spesso corrisponde alla **massimizzazione della probabilita' a posteriori** (MAP), cioe' si considera la probabilita' corrispondente al valore target y piu' probabile. Questo e' ragionevole dal momento che la probabilita' piu' alta minimizza l'errore.
 - Cio' corrisponde a scegliere il valore y che massimizzi la probabilita' sapendo che l'esempio sia X . Siccome X e' fisso, l'unica parte variabile sara' y , per cui possiamo ignorare nella legge di Bayes il termine $P(X)$ siccome non ne influenza il valore, e limitarci a massimizzare solo il numeratore
 - I modelli probabilistici si dividono in due tipologie principali:
 - **Modelli Discriminativi**: modellano la distribuzione di probabilita' a posteriori $P(Y | X)$. Si focalizzano sulla scelta della classe data l'istanza, scegliendo la classe piu' probabile per X . Non hanno l'obiettivo di descrivere in termini assoluti la distribuzione delle classi in modo che si possa utilizzare per determinare quale sia quella piu' probabile data una istanza da classificare
 - **Modelli Generativi**: modellano la distribuzione di probabilita' della classe in termini di features, in maniera congiunta ($P(X, Y)$). Si pone l'obiettivo di modellare *come si distri-*

buiscono le caratteristiche all'interno delle varie classi. Nonostante siano piu' espressivi, i modelli generativi non si usano sempre, perche' richiedono una quantita' molto grande di dati in modo che siano statisticamente significativi. Avere dei modelli generativi permette in oltre di **creare** esempi verosimili di una determinata classe, perche' vengono creati a partire dalle distribuzioni di probabilita' delle features

Nota: Esistono al giorno d'oggi dei modelli chiamati GAN (Generative Adversarial Networks) che modellano in maniera generativa le classi

- Caratteristiche dei modelli generativi:
 - Un modello generativo richiede di memorizzare la distribuzione di probabilita' congiunta, che aumenta esponenzialmente in dimensioni rispetto al numero delle features. Tuttavia, e' possibile semplificare lo spazio delle ipotesi introducendo l'indipendenza delle features (*naive Bayes*)
 - Ci sono casi in cui non riescono a modellare tanto bene come si distribuiscono le features tra le varie classi. Magari in una classe il modello e' molto preciso a descriverla in termini di combinazioni di valori di features mentre in altre no. In altri termini, e' possibile che l'accuratezza di $P(X)$ potrebbe essere non bilanciata rispetto all'accuratezza di $P(Y|X)$. Nonostante cio', se $P(X)$ e' piccola, sappiamo che sara' molto poco probabile che arrivino esempi di quel tipo, per cui il tasso di missclassificazione rimane molto basso.
 - L'apprendimento dai dati puo' essere visto come un processo progressivo di riduzione dell'incertezza (cioe' dell'entropia)

9.1 Apprendimento come riduzione dell'incertezza

- Supponiamo di voler stimare la probabilita' a priori θ ($P(Y)$) che una email sia spam
 - Una stima iniziale e' stimare $\hat{\theta} = d/n$ dove d e' il numero di istanze che sono spam
 - La stima data pero' e' una stima di massimizzazione della probabilita' a posteriori, per cui non significa che altri valori di θ siano completamente esclusi. Per poterli considerare, e' piu' appropriato modellare il valore θ come una distribuzione di probabilita' binomiale (β -distribution).
 - Ogni volta che si ispeziona una email si aggiorna la distribuzione, si riducendo l'incertezza riguardo la probabilita' di spam θ . In questo modo la distribuzione si "schiaccia" aumentando in altezza e diminuendo in larghezza verso uno specifico valore di likelihood di classe.
- In sostanza il metodo puo' essere riassunto nei seguenti passaggi:
 1. Fai una stima iniziale a priori $P(Y)$
 2. Aggiorna la stima iniziale guardandondo i dati, producendo una stima a posteriori $P(Y | X)$

3. Riutilizza come stima iniziale a priori la stima prodotta a posteriori $P(Y) = P(Y | X)$

The Beta distribution function has the form: $f(x; \alpha, \beta) = \text{constant} \cdot x^{\alpha-1}(1-x)^{\beta-1}$

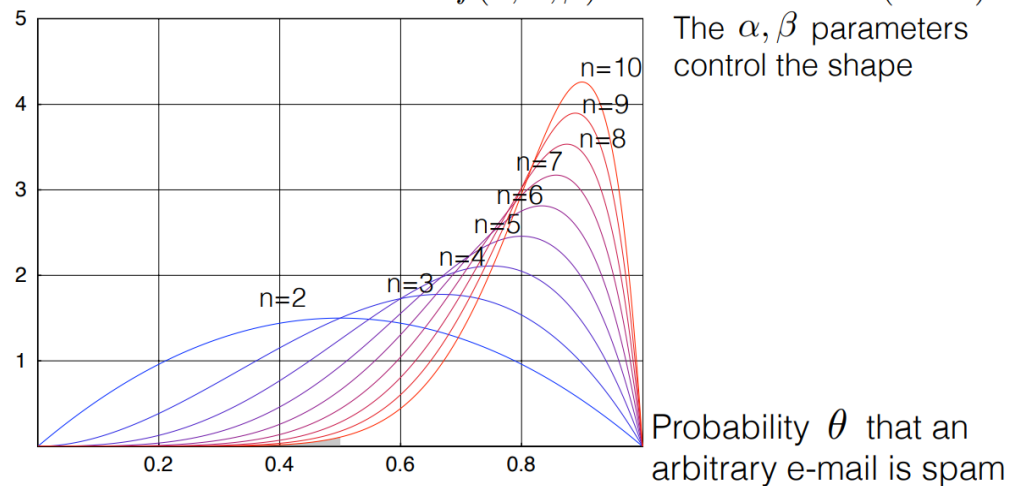


Figura 7: Distribuzione di probabilita' di θ man mano che consideriamo nuovi esempi di emails. In questo caso abbiamo 1 non spam e tutte il resto spams

- La riduzione di incertezza corrisponde ad una riduzione della varianza. Notiamo che nel picco (con $n = 10$) la varianza e' molto bassa significa che l'incertezza e' bassa di conseguenza
- Andando a modellare la stima attraverso una distribuzione di probabilita' otteniamo inoltre diversi vantaggi:
 - Possiamo generare un modello *generativo*, per cui campionare dalla distribuzione ci permette di creare nuove istanze verosimili (i metodi Monte Carlo fanno una cosa simile)
 - Come gia' detto e' possibile misurare l'incertezza misurando semplicemente la varianza rimanente
 - Possiamo quantificare la probabilita' di affermazioni sul parametro stesso

In breve, la prospettiva Bayesiana consiste nell'usare queste distribuzioni per codificare le nostre credenze. In questo modo si possono associare delle distribuzioni di probabilita' ad ogni cosa: non solo a features e a valori targets ma anche a parametri e modelli stessi.

- Un qualunque modello di predizione, se si comporta come si comporterebbe il teorema di Bayes (avendo opportunamente modellato le probabilita' necessarie), e quindi farebbe le stesse conclusioni che farebbe il teorema di Bayes, allora tale modello e' detto **Bayes ottimale**

Risultati teorici dicono che un classificatore probabilistico non puo' fare meglio del teorema di Bayes, cioe' il teorema di Bayes e' il meglio raggiungibile da un modello

9.2 Modelli Probabilistici e Significato Geometrico

- E' possibile vedere delle connessioni tra i modelli probabilistici e i modelli geometrici. Per far cio' introduciamo prima alcune definizioni

Distribuzione Normale Univariata: La distribuzione normale (o Gaussiana) ha la seguente pdf

$$P(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2} \left[\frac{x - \mu}{\sigma}\right]^2\right)$$

dove μ e' la media e σ la deviazione standard.

Distribuzione Normale Multivariata:

$$P(x|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2} \sqrt{|\Sigma|}} \exp\left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu)\right)$$

in cui $x = (x_1, \dots, x_d)^T \in \mathbb{R}^d$ e Σ e' la matrice della covarianza. $\mu = (\mu_1, \dots, \mu_d)$ e' il vettore che contiene le medie rispetto ai singoli valori di x

9.3 Connessione 1

- Sviluppiamo ora un possibile modello probabilistico e confrontiamolo con quello geometrico. Noteremo delle connessioni interessanti.
- Iniziamo considerando il caso univariato in due classi, per cui sappiamo che le istanze sono rappresentate da una sola feature $x \in \mathbb{R}$.
- Diciamo che il modello e' **misto** se ipotizziamo che le classi siano state "generate" da una legge di tipo probabilistico. Cio' significa che ogni classe ha la propria *distribuzione di probabilita'*, che puo' essere diversa dall'altra. Diciamo inoltre che entrambe sono delle distribuzioni *Gaussiane*, per cui parliamo di modello a **mistura di Gaussiane**.
- Rappresentiamo quindi la distribuzione delle classi secondo delle Gaussiane:

$$P(x|\oplus) = \frac{1}{\sqrt{2\pi}\sigma^\oplus} \exp\left(-\frac{1}{2} \left[\frac{x - \mu^\oplus}{\sigma^\oplus}\right]^2\right), \quad P(x|\ominus) = \frac{1}{\sqrt{2\pi}\sigma^\ominus} \exp\left(-\frac{1}{2} \left[\frac{x - \mu^\ominus}{\sigma^\ominus}\right]^2\right)$$

- Poiche' il valore della classe e' fissato, stiamo parlando di Likelihood. In altri termini le funzioni ci descrivono la **probabilita' di ottenere uno specifico esempio in una classe fissata**.

- Per poter definire il decision boundary del modello, sfruttiamo le proprietà degli esponenti e definiamo il **Likelihood Ratio** come:

$$LR(x) = \frac{P(x|\oplus)}{P(x|\ominus)} = \frac{\sigma^\oplus}{\sigma^\ominus} \exp \left(-\frac{1}{2} \left[\left(\frac{x - \mu^\oplus}{\sigma^\oplus} \right)^2 - \left(\frac{x - \mu^\ominus}{\sigma^\ominus} \right)^2 \right] \right)$$

Si noti che il decision boundary coinciderà con tutti quei punti in cui **LR = 1**, per cui quando le probabilità delle due classi sono uguali, e non si può prendere una decisione.

- Consideriamo ora un caso specifico e supponiamo che le due componenti abbiano la stessa deviazione standard $\sigma^\oplus = \sigma^\ominus = \sigma$. Ne deriva (calcoli omissi) che

$$LR(x) = \exp[\gamma(x - \mu)]$$

in cui

- $\gamma = (\mu^\oplus - \mu^\ominus)/\sigma^2$ e' la differenza tra le medie in proporzione alla varianza
- $\mu = (\mu^\oplus + \mu^\ominus)/2$ e' il **punto medio** tra le medie delle due classi
- Dalla relazione precedente, ne segue che il punto di decisione e' situato al valore di $x = \mu$ (cioe' il punto medio tra le due classi), poiche' rende $LR(x) = 1$.
- Possiamo quindi ottenere un classificatore lineare di base scegliendo come decision boundary la linea bisettrice perpendicolare al segmento che connette μ^\oplus e μ^\ominus .
- In generale, quando $\Sigma^\oplus = \Sigma^\ominus$ il decision boundary interseca il segmento $\mu^\oplus - \mu^\ominus$ nel mezzo, **con un angolo retto se le features non sono correlate**, mentre con **un angolo non retto se le features sono correlate***
- Alternativamente, quando $\Sigma^\oplus \neq \Sigma^\ominus$ **il decision boundary e' iperbolico** (cioe' coincide con un intervallo delimitato dai punti in cui le pdf si intersecano). In altri termini, non e' contiguo all'interno dello spazio degli esempi.

Piu' in generale, ne segue che $LR(x) = 1$ per tutte le istanze x che sono equidistanti dai centroidi rappresentati dai vettori μ^\oplus e μ^\ominus . Ma questo significa che il decision boundary sara' una linea dritta a distanza uguale dalle medie, per cui equivale ad un classificatore lineare

- Da questa ultima osservazione possiamo dire che nel caso in cui le features Gaussianne non siano correlate e abbiano varianza unitaria, il classificatore lineare e' **Bayes Ottimale** Piu' in generale quando le matrici di covarianza sono uguali (features non correlate), il decision boundary e' lineare ed e' perpendicolare al segmento $\mu^\oplus - \mu^\ominus$.

9.3.1 Connessione 2

- La distribuzione multivariata essenzialmente traduce distanze in probabilità. E questo lo si vede poiché compare la definizione della distanza di Mahalanobis. Conseguentemente, compare la definizione di distanza Euclidea in quella univariata.
- Se consideriamo il logaritmo negativo della likelihood Gaussiana, possiamo interpretarlo come la distanza al quadrato tra x e la media
- Un'altro esempio di connessione tra prospettive geometriche e probabilistiche lo si vede quando si vogliono stimare i parametri di una Gaussiana.

9.3.2 Connessione 3

- Ipotizziamo che si voglia stimare il parametro μ di una distribuzione con matrice di covarianza Σ da un set di istanze X . Per far ciò utilizzeremo il principio di **MLE**
- Facciamo anche l'ipotesi che le istanze del dataset siano state “generate” in maniera indipendente l'una dall'altra. Cioè se prendessimo un campione preso della popolazione, e scegliessimo a caso un'altro campione: la scelta di uno non determina il risultato di quello seguente
- Utilizzando questa ipotesi di indipendenza, possiamo formulare la likelihood di *tutto il dataset* come la produttoria della likelihood delle singole istanze

$$\prod_{X_i \in D} P(X_i | Y = y), \quad y \in \{\oplus, \ominus\}$$

- Per stimare μ data la matrice Σ , il principio di **maximum likelihood estimation** ci dice che μ deve coincidere con il valore che massimizza la likelihood di X (assumendo che le istanze che compongono X siano state campionate indipendentemente)

$$\begin{aligned} \hat{\mu} &= \arg \max_{\mu} P(x | \mu, \Sigma) \\ &\vdots \\ &= \arg \min_{\mu} \sum_{x \in X} (Dis_M(x, \mu | \Sigma))^2 \end{aligned}$$

Troviamo quindi che la stima di massimo likelihood corrisponde al punto che minimizza la somma delle distanze di Mahalanobis al quadrato di tutti i punti in X . μ è il vettore più vicino rispetto a tutti, cioè il *centroide*, ma questo corrisponde proprio al significato intuitivo e geometrico dell'apprendimento.

9.4 Modelli Probabilistici per Dati Categorici

- Sappiamo che una variabile aleatoria X e' una variabile che puo' assumere i valori $\{x_i\}, i = 1, \dots, d$ dove ogni valore x_i ha una certa probabilita' θ_i di valorizzare la variabile.
- Vogliamo classificare dei documenti come **spam** o **ham**, data l'occorrenza di diverse parole all'interno di un vocabolario prefissato di d parole.
- Rappresentiamo le occorrenze di ogni parola i in un documento con una *variabile aleatoria discreta* X_i , in quanto, preso un documento non sappiamo quante occorrenze delle parole contenga a priori.
- Quando abbiamo classi di documenti (ad esempio *spam* o *ham* nel caso dello spam filter) differenti, e' ragionevole pensare che diverse classi abbiano distribuzione diversa dell'occorrenza di parole, per cui ipotizzando di avere due classi \oplus, \ominus , avremo θ_i^{\oplus} e θ_i^{\ominus}
- Abbiamo principalmente due modi differenti per modellare i documenti dal punto di vista probabilistico

9.4.1 Modello Bernoulliano Multivariato

Un test bernoulliano e' una domanda "*binaria*" che puo' avere solo due outcomes (T/F)

- In questo tipo di approccio, la singola variabile X_i corrispondente alla *i-esima* parola rappresenta la presenza/assenza della stessa in un dato documento. Tale variabile ha una distribuzione di due valori: successo ($P(X = 1) = \theta$) o fallimento ($P(X = 0) = 1 - \theta$).
- Ogni documento e' quindi equivalente ad un vettore di variabili aleatorie Bernoulliane $X = (X_1, \dots, X_d)$.
- La distribuzione di probabilita' dell'intero vettore X e' chiamata **distribuzione di Bernoulli multivariata**.

La distribuzione di Bernoulli e' un caso speciale della distribuzione binomiale, in cui k (il numero di trials) e' 1

- Il modello in memoria si salva una matrice $M^{n \times d}$, dove n e' il numero di documenti e d sono le parole nel vocabolario. Tale matrice booleana, vale 1 in corrispondenza della parola j nel documento i , e indica la presenza di quella data parola in quel dato documento.
- Facendo cio' e' possibile quindi stimare le singole $P(X_j = 1) = \theta_j$:

$$\theta_j = \frac{\sum_{i=0}^n M(i, j)}{n}$$

cioe' il rapporto del numero di documenti che contengono la parola *j-esima* sul numero dei documenti totali.

9.4.2 Modello Bernoulliano Multinomiale

- Il modello Bernoulliano Multinomiale e' una generalizzazione di quello Multivariato, in cui non ci si limita a rappresentare se una parola e' presente o meno ma conta la **frequenza** delle occorrenze di una parola all'interno del documento
- Il modello usa una variabile aleatoria X_i che rappresenta il numero di occorrenze della parola i nel documento. La probabilita' θ_i indica la probabilita' di trovare quel numero di occorrenze della parola i nel testo.
- La distribuzione **binomiale** modella il numero di eventi di successo S (in questo caso il numero di occorrenze della parola) in n prove che sono indipendenti l'una dall'altra. Le prove consistono nel vedere se nella posizione di un documento composto da n parole c'e' quella parola. La singola prova corrisponde ad un Bernoulli trial

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

- Se prima nel modello multivariato un Bernoulli trial era il singolo *documento*, in questo caso il singolo trial e' una *singola posizione* di una parola nel documento.
- Il modello rappresenta il singolo documento con un vettore $X = (X_1, \dots, X_d)$, che rappresenta le *frequenze* (cioe' quante volte compaiono) delle parole all'interno del documento, in altri termini lo rappresenta con un istogramma
- Come gia' detto, il modello vede il documento come un processo di Bernoulli di n prove. La singola probabilita' della parola i di comparire in una qualsiasi prova e' $P(X_i = 0) = \theta_i$ e $P(X_i = 1) = 1 - \theta_i$.
- Utilizziamo una distribuzione **multinomiale** per modellare la stima della probabilita' tenendo conto delle d variabili in tutte le posizioni del documento composto da n parole:

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

In sostanza non e' altro che la **generalizzazione** della distribuzione binomiale vista poch'anzi. Ogni singola parola ha la sua probabilita' di comparsa θ_i , elevata al numero di volte che compare x_i . I fattoriali al denominatore servono a tener conto delle posizioni di comparsa delle parole nel testo.

9.5 Naive Bayes

- Entrambi i modelli visti in precedenza funzionano grazie all'ipotesi di **naive Bayes**: utilizzano il teorema di Bayes per stimare la probabilita' della classe, le osservazioni date dalle frequenze

per stimare la probabilita' di comparsa delle parole nei documenti di una certa classe, facendo l'assunzione che la comparsa delle parole sia indipendente dalla comparsa delle altre

- In questo modo possiamo evitare di rappresentare la likelihood condizionata alla classe ma in maniera *congiunta* rispetto a tutte le parole del vocabolario. (si pensi ad esempio che il vocabolario Inglese ha 8000 parole)
- In generale, l'ipotesi di indipendenza non rispecchia la realta'. Si pensi ad esempio di osservare la parola *Viagra* in un documento, per cui sara' molto piu' probabile che compaia *pills* come parola successiva. Nonostante questo, anche con l'ipotesi di indipendenza il modello da una stima particolarmente accurata delle distribuzioni
- Riassumiamo infine formalmente cio' che viene fatto con le ipotesi di Naive Bayes:
 1. Assumiamo che le istanze siano rappresentate su d attributi
 2. Applicando il teorema di Bayes, facciamo una stima di massimizzazione a posteriori. Cio' corrisponde a trovare il valore di Y tale che massimizzi

$$P(Y|X_1, \dots, X_d) = \frac{P(X_1, \dots, X_d|Y)P(Y)}{\underbrace{P(X_1, \dots, X_d)}}_1$$

3. Stimare $P(X_1, \dots, X_d|Y)$ e' difficile
4. Facendo l'ipotesi di indipendenza tra attributi X_i otteniamo

$$P(X_1, \dots, X_d|Y) = P(X_1|Y)P(X_2|Y) \dots P(X_d|Y)$$

che e' piu' semplice da stimare

9.5.1 Regole di Decisione Probabilistiche


- Con Naive Bayes calcoliamo la probabilita' di ottenere un esempio supponendo di essere in una data classe (*likelihood*). Piu' le likelihood sono differenti tra loro, piu' saranno utili le features di X nella classificazione.
- Dopo aver scelto una delle distribuzioni per modellare i nostri dati, per un singolo esempio x calcoliamo le likelihood della classe positiva $P(X|Y = \oplus)$ e negativa $P(X|Y = \ominus)$, per poi applicare una delle possibili regole di decisione:
 1. *maximum likelihood (ML)* - $\arg \max_y P(X = x|Y = y)$
 2. *maximum a posteriori (MAP)* - $\arg \max_y P(X = x|Y = y)P(Y = y)$
 3. *recalibrated likelihood* - $\arg \max_y w_y P(X = x|Y = y)$

- Le regole di decisione 1. e 2. sono equivalenti quando la distribuzione delle classi e' **uniforme**. La terza generalizza le prime due sostituendo la distribuzione delle classi ($P(Y = y)$) con dei pesi appresi dai dati tali che minimizzino la perdita.
- La regola del recalibrated likelihood si utilizza in caso le distribuzioni siano scalibrate.

9.5.2 Apprendimento di un modello Naive Bayes

- L'apprendimento dei modelli consiste anche nella stima dei parametri delle distribuzioni θ . Ad esempio, nel caso di un modello multivariato possiamo stimarlo come $\hat{\theta} = d/n$, cioe' contando il numero di documenti in cui esce la parola in questione (d) sul numero dei documenti totali (n)
- E' possibile pero' che alcune parole non compaiano mai all'interno del training set. Anche se poco probabili devono essere considerate comunque, per cui si applica uno smoothing alla stima $\hat{\theta}$, andando ad aggiungere degli **pseudo counts** nel training set. Essi sono essenzialmente 2 documenti: uno contenente tutte le parole e uno che non ne contiene nessuna. L'aggiunta di questi due documenti corrisponde ad effettuare una **Laplace Smoothing** rendendo la stima $\hat{\theta} = d + 1/n + 2$.
- Applicare questo smoothing permette di dare una chance anche quelle parole che sono presenti nel vocabolario ma che non appaiono neanche una volta nel dataset

E-mail	#a	#b	#c	Class
e_1	0	3	0	+
e_2	0	3	3	+
e_3	3	0	0	+
e_4	2	3	0	+
e_5	4	3	0	-
e_6	4	0	3	-
e_7	3	0	0	-
e_8	0	0	0	-



E-mail	#a	#b	#c	Class
e_1	0	3	0	+
e_2	0	3	3	+
e_3	3	0	0	+
e_4	2	3	0	+
	{ 1	1	1	{ +
	{ 0	0	0	{ +
	{ 1	1	1	{ -
	{ 0	0	0	{ -
e_5	4	3	0	-
e_6	4	0	3	-
e_7	3	0	0	-
e_8	0	0	0	-

Figura 8: Inserimento di outliers (*pseudocounts*) per applicare uno smoothing

- Guardare Esempio 9.5

Il main takeaway dell'esempio e' che cambia in generale il conteggio in base al tipo di distribuzione che si sceglie.

9.6 Regressione Logistica

- Fino ad ora abbiamo visto modelli generativi di Naive Bayes, vedremo ora un modello discriminativo: la **Regressione Logistica**
- Nella regressione lineare le variabili sono connesse tra loro tramite una relazione di tipo lineare, in cui il goal e' quello di predire l'output y_i , che e' un valore **reale**

$$y_i = \beta_0 + \beta_1 x_{i1} + \dots + \beta_d x_{id}$$

- Nella regressione logistica, invece, vogliamo predire il valore di una variabile binaria (ad esempio il valore di una classe *positiva/negativa*) In questo caso si vuole predire il valore di una variabile casuale Y_i che vale 1/0 in caso la classe sia positiva/negativa per l'esempio i
- Piu' precisamente, vogliamo stimare $P(Y_i|X_i)$. Sappiamo che Y_i e' una variabile con una distribuzione di Bernoulli.
- L'intuizione principale, e' quella che la regressione logistica utilizza la regressione lineare per stimare i parametri della distribuzione (cioe' la probabilita' di successo $\hat{\theta}$) che sono poi **logisticamente ricalibrati**
- Siccome i parametri della distribuzione sono delle stime di probabilita', bisogna rendere il valore di output della regressione lineare tra 0 e 1. Per far cio' si utilizza una funzione sigmoide (da qui il nome logistica), che trasforma il range del valore in output della regressione da $[-\infty; +\infty]$ a $[0, 1]$. Inserendo quindi della regressione lineare all'interno della sigmoide otteniamo:

$$P(Y_i|x_i) = \frac{\exp^{\beta_0 + \beta_1 x_{i1} + \dots + \beta_d x_{id}}}{1 + \exp^{\beta_0 + \beta_1 x_{i1} + \dots + \beta_d x_{id}}}$$

- Nella realta' per semplificare i calcoli si passa attraverso il logaritmo degli odds (probabilita' della classe positiva sulla probabilita' della classe negativa)

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

$$\ln(Odds(x_{i0})) = \beta_0 + \beta_1 x_{i1} + \dots + \beta_d x_{id}$$

L'interpretazione del logaritmo ci dice che se qualora sia < 0 , allora la probabilita' di predire una classe positiva e' piu' piccola di quella negativa. Viceversa quando e' > 0 . Cosi' come nella likelihood, il decision boundary e' al valore 0, per cui non si riesce a prendere una decisione tra le due classi.

La regressione logistica e' uno dei modelli piu' utilizzati dagli statistici

- La regressione logistica puo' anche funzionare quando le **variabili sono categoriche**. Per far cio', prendiamo per esempio il caso in cui la feature categorica X possa valere A oppure B , per

cui si può costruire la seguente tabella di contingenza:

X/Y	1	0	Tot
A	n_{A1}	n_{A0}	n_A
B	n_{B1}	n_{B0}	n_B
Tot	n_1	n_0	n

Figura 9: Tabella di contingenza in cui sono segnate le frequenze in cui compare il valore A e B negli esempi all'interno delle classi positive e negative

- L'Odds dei valori della feature sono facilmente calcolabili:

$$Odds(A) = \frac{P(Y = 1|X = A)}{P(Y = 0|X = A)} = \frac{n_{A1}}{n_{A0}}$$

$$Odds(B) = \frac{P(Y = 1|X = B)}{P(Y = 0|X = B)} = \frac{n_{B1}}{n_{B0}}$$

Per ottenere la regressione logistica, si sostituisce nella formula di regressione

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

il valore di $Odds(x_{ij})$ al posto del valore della feature categorica x_{ij}

- L'odds intuitivamente mi dice quanto è più possibile che un esempio osservato con quel valore della feature specifico sia assegnato alla classe positiva piuttosto che alla classe negativa
- Nel libro viene trattato in modo leggermente differente:
 - Supponiamo di avere un regressore $p(x) = w \cdot x - t$ (che corrisponde a $w_1, \dots, w_d = \beta_1, \dots, \beta_d$ e $-t = \beta_0$)
 - Appliciamo una normalizzazione logistica

$$\hat{p}(x) = \frac{\exp(w \cdot x - t)}{1 + \exp(w \cdot x - t)} = \frac{1}{1 + \exp(-(w \cdot x - t))}$$

- Se assumiamo che la distribuzione sia di Bernoulli e le classi siano $y = 0/1$ per negativa/positiva, per ottenere $P(y_i|x_i)$ basta sostituire il valore del regressore normalizzato $\hat{p}(x)$ al valore θ all'interno della distribuzione di Bernoulli:

$$P(y_i|x_i) = \hat{p}(x_i)^{y_i} (1 - \hat{p}(x_i))^{(1-y_i)}$$

- Il processo di conversione di un valore arbitrario a una probabilità è chiamato **calibrazione**

- La regressione logistica differisce dalla regressione lineare perché tende a modellare più accuratamente gli esempi più vicini al decision boundary (nei punti in cui le due classi si sovrappongono di più).

9.6.0.1 Apprendimento di regressori logistici

- L'apprendimento di un Logistic Regressor concerne il trovare i parametri del regressore lineare che massimizzino la likelihood (che corrisponde quindi ad un *Maximum Likelihood Estimation*).
- Possiamo ottenere la **conditional likelihood** nel modo seguente:

$$CL(w, t) = \prod_i P(y_i | x_i) = \prod_i \hat{p}(x_i)^{y_i} (1 - \hat{p}(x_i))^{(1-y_i)}$$

- Per massimizzare CL ne consideriamo il logaritmo LCL e se ne fa la derivata parziale rispetto a w e t , ponendole a 0.
- Omissi i calcoli, si ottiene che

$$\frac{\partial}{\partial t} LCL(w, t) = \sum_{x_i \in Tr} (\hat{p}(x_i) - y_i) = 0$$

otteniamo che per rendere il più possibile la quantità precedente a 0 è necessario che la probabilità predetta debba essere uguale alla proporzione dei positivi pos . In altri termini ci dice bisogna trovare una stima \hat{p} che minimizzi gli errori $(\hat{p}(x_i) - y_i)$

- Si noti che i ranking trees hanno questa proprietà dal momento che assegnano come probabilità predette le probabilità empiriche di un segmento
- L'apprendimento di un modello a regressione logistica può essere ridotto quindi al seguente problema di ottimizzazione:

$$w^*, t^* = \arg \max_{w, t} CL(w, t) = \arg \max_{w, t} LCL(w, t)$$

(LCL è il logaritmo della conditional likelihood CL)

- Tale problema di ottimizzazione è **convesso** (esiste una e una sola soluzione), per cui esistono diversi metodi combinatori per ottimizzarlo
- Uno degli approcci più semplici che vengono applicati è quello dato dalla seguente regola di update:

$$\mathbf{w}' = \mathbf{w} + \eta(y_i - \hat{p}_i)\mathbf{x}_i$$

in cui η è il *learning rate*. Tale metodo è utilizzato inoltre per l'apprendimento dei *percettroni* nelle Reti Neurali Profonde.

- In questa regola di update, notiamo come l'errore $(y_i - \hat{p}_i)$ indichi la **direzione** da prendere per il prossimo valore: se l'errore è positivo, il prossimo valore di \mathbf{w} sarà più grande del precedente,

se negativo sara' piu' piccolo.

9.7 Modelli basati sulla Compressione

- Uno dei problemi dell'apprendimento automatico e' l'overfitting: durante l'apprendimento il modello diventa troppo complesso, adattandosi molto bene ai dati ma perdendo la capacita' di generalizzare
- Possiamo quantificare un fenomeno del genere modellando i modelli tramite il principio della *quantita' di informazione*
- L'idea e' quella di rappresentare un modello come se fosse un messaggio:
 - Piu' il modello e' complesso, piu' e' lungo il messaggio
 - Viceversa, piu' facile e' il modello piu' corto sara' il messaggio
- Possiamo anche utilizzare i principi dell'information theory per fare classificazione. Questo perche' esiste una stretta relazione tra probabilita' e *information content* di un evento. Consideriamo, ad esempio, la stima di *a priori maximization*:

$$y_{MAP} = \arg \max_y P(X = x | Y = y)P(Y = y)$$

- Possiamo trasformarla in una stima di minimizzazione considerandone il logaritmo

$$y_{MAP} = \arg \min_y -\log P(X = x | Y = y) - \log P(Y = y)$$

Ma l'inverso del logaritmo della probabilita' e' proprio la quantita' di informazione. Per cui possiamo denotare con

$$IC(X | Y) = -\log_2 P(X | Y)$$

$$IC(Y) = -\log_2 P(Y)$$

e infine fare una stima y_{MAP} equivalente nel modo seguente

$$y_{MAP} = \arg \min_y IC(X = x | Y = y) + IC(Y = y)$$

- Come detto nell'introduzione, possiamo utilizzare questi concetti di information theory anche per ridurre l'overfitting del modello.
- Consideriamo un modello m e denotiamo con $L(m)$ la lunghezza in *bit* della **descrizione di tale modello**. m e' una variabile casuale che ha una certa probabilita' di avverarsi, cioe' i modelli hanno una certa probabilita' di avverare.

Ci sono vari modi per descrivere un modello. Ad esempio, in un albero di decisione potremmo dire che prima comunichiamo il livello di appartenenza di ogni nodo, poi comunichiamo l'attributo che viene utilizzato dentro il nodo interno per lo split node, infine l'etichetta assegnata alle foglie.

- Denotiamo infine con $L(D \mid m)$ la lunghezza in bit di un messaggio che descrive il dato D , dato il modello m . Intuitivamente, cio' che denota e' questo: m e' un modello che e' stato addestrato dai dati, ma nonostante cio' puo' effettuare degli errori. I dati su cui gli errori vengono fatti sono D . Possiamo dire che $L(D \mid m)$ quantifica l'errore del modello m
- Il principio di **Minimum Description Length** descrive la capacita' di un modello di essere compresso (compatto) e dall'altra essere accurato. Il modello che minimizza questa quantita' e':

$$m_{MDL} = \arg \min_{m \in M} L(m) + L(D \mid m)$$

quindi, vogliamo trovare il modello m che minimizza la complessita' del modello (data dalla lunghezza del messaggio per trasmetterlo $L(m)$), e la lunghezza dei messaggi che devo mandare a fronte di errori del modello $L(D \mid m)$

- Possiamo vedere il principio intuitivamente nel modo seguente. Supponiamo di voler trasmettere un dataset etichettato. L'idea e' quella di trasmettere prima un modello addestrato sui dati etichettati (m) in modo che il ricevente possa ricavarne le etichette. Le etichette pero' non saranno tutte corrette (poiche' il modello puo' fare degli errori di classificazione), per cui dovremmo anche trasmettere i dati in cui il modello fa errori ($D \mid m$) per poter ricostruire l'intero dataset etichettato originale.

9.8 Expectation Maximisation

- Vediamo ora come possiamo utilizzare la prospettiva probabilistica per fare del *clustering* di dati.
- L'idea e' quella di assegnare una probabilita' di appartenenza per ogni cluster. Cio' e' differente da quello che abbiamo fatto fin'ora poiche' diversi cluster possono essere verosimili dato un esempio.
- Gli algoritmi di clustering di questo tipo spesso modellano i dati utilizzando delle **mixture finite di distribuzioni di probabilita'**.
 - Cio' significa che facciamo un'assunzione di partenza che tutti i dati del dataset provengono da un numero **finito** di clusters.
 - Ogni cluster e' in grado di generare gli esempi che appartengono ad esso utilizzando una legge di tipo probabilistico (funzione di distribuzione di probabilita')
 - Tipicamente le funzioni di distribuzione sono di tipo Gaussiano, per cui per descrivere ciascun cluster e' necessario stabilire i parametri μ, σ di ogni Gaussiana

- Le distribuzioni differenti sono poi combinate tra loro utilizzando dei pesi che modellano l'impatto di un cluster piuttosto che un altro
- Secondo quanto detto, quindi, il generico cluster l -esimo è rappresentato da

$$f_l(x; \mu_l, \sigma_l) = \frac{1}{\sqrt{2\pi}\sigma_l} e^{-\frac{(x-\mu_l)^2}{2\sigma_l^2}}$$

- Sappiamo inoltre che data la **probabilità di appartenenza ad un generico cluster** $P(C_l)$ (si calcola semplicemente andando a vedere quanti sono gli esempi appartenenti a C_l) vale

$$1 = \sum_l^k P(C_l)$$

e quindi possiamo utilizzare $P(C_l)$ come peso per combinare assieme tutte le distribuzioni:

$$f(x) = \sum_l^k P(C_l) f_l(x; \mu_l, \sigma_l)$$

- Se noi sapessimo già a quali clusters appartengono tutti i dati, i parametri delle distribuzioni sarebbero facilmente stimabili, così come i pesi. La stessa cosa vale nel caso si conoscessero a priori i parametri delle distribuzioni. Ovviamente ciò non è possibile nel task di clustering.
- **Expectation Maximisation** risolve questo problema elegantemente, tramite l'iterazione di due passaggi:
 - Nel primo passo (*expectation step*), fissa i parametri di tutti i clusters ($\mu_c, \sigma_c, P(C_l)$) e determina per ogni istanza il cluster di appartenenza secondo i parametri assegnati
 - Nel secondo passo (*maximization step*), vengono ricalcolati tutti i parametri $\mu_l, \sigma_l, P(C_l)$ secondo l'assegnazione fatta nel primo step. Si chiama di maximization perché questo passo massimizza la *likelihood* dei parametri delle Gaussiane, cioè ci dà i parametri più verosimili una volta osservati gli esempi.
- **EM** si aspetta che si sappia a priori il numero di clusters k , così come in **k-Means**. In realtà **EM** è una generalizzazione di **k-Means** ma in una prospettiva probabilistica: entrambi i metodi consistono in una procedura iterativa di assegnamento e ricalcolo.

9.8.0.1 Expectation Step

- I parametri $\mu_l, \sigma_l, P(C_l)$ di ogni cluster l sono assegnati secondo una regola specifica, che può essere anche randomica

- Per ogni cluster l si calcolano poi dei pesi per ogni istanza i

$$\hat{P}(C_l | x_i) = \frac{f(x_i; \mu_l, \sigma_l)P(C_l)}{f(x_i)} = w_{li}$$

in cui $f(x_i)$ e' la somma di tutte le distribuzioni pesate per la probabilita' vista precedentemente.

In questa formula, $f(x_i; \mu_l, \sigma_l)$ e' anche denotato come $f(x_i, C_l)$

9.8.0.2 Maximization Step

- In questo step avviene la stima vera e propria dei parametri. Anziche' andare a fare la massimizzazione della likelihood complessiva, si vanno a stimare i parametri utilizzando i pesi calcolati nello step E:

$$\hat{P}(C_l) = \frac{1}{n} \sum_{i=1}^n \hat{P}(C_l | x_i) = \frac{1}{n} \sum_{i=1}^n w_{li}$$

$$\mu_l = \frac{\sum_{i=1}^n w_{li} x_i}{\sum_{i=1}^n w_{li}}$$

$$\sigma_l = \frac{\sum_{i=1}^n w_{li} (x_i - \mu_l)^2}{\sum_{i=1}^n w_{li}}$$

- La procedura termina quando il logaritmo della likelihood si satura

10 Features

10.1 Tipologie di Features

- Le features possono essere definite formalmente come delle mappe $f_i : \mathcal{X} \rightarrow \mathcal{F}_i$, che mappano ogni istanza in \mathcal{X} al dominio delle features \mathcal{F}_i .
- Il dominio delle features puo' essere reale (\mathbb{R}), intero (\mathbb{Z}) ma puo' anch essere un insieme discreto come i colori, i booleani e cosi' via
- Possiamo distinguere le features anche in base alle operazioni che possiamo fare sui valori. Si pensi ad esempio alla media, che puo' essere fatta solamente su alcune tipologie di valori come ad esempio l'altezza di una persona ma non il suo gruppo sanguigno.
- Il range delle operazioni possibili dipende anche dall'esistenza di una scala di misura. Quando abbiamo una scala di misura possiamo effettivamente fare delle operazioni algebriche sui valori (somma, prodotto ecc..)
- Distinguiamo quindi in base a questo criterio diversi tipi di features:
 - **Categoriche**: features i cui valori non possono essere ordinati (es. Booleane)
 - **Ordinali**: features i cui valori possono essere ordinati. Non necessariamente possono essere dei valori numerici (es. insiemi di caratteri o stringhe che esprimono un ranking come "primo, secondo, terzo" ecc..)
 - **Quantitative**: features in cui si possono fare operazioni algebriche (somma/prodotto ecc..)
- In base alla tipologia di feature possiamo calcolare diverse statistiche:
 - **Statistiche di tendenza centrale**: come la *media* (o valor medio), *mediana* (valore a meta' dell'ordinamento) o la *moda* (valore piu' frequente). La media possiamo calcolarla solo nelle features quantitative, la mediana per quelle ordinali/quantitative e la moda per tutte. La media inoltre viene calcolata in modi differenti in base alla scala della feature:
 - * *Lineare*: la media viene calcolata come media **aritmetica**
 - * *Reciproca* (ad esempio se ho delle velocita'): la media viene calcolata come media **armonica**
 - * *Frequenza* (in musica ad esempio): la media viene calcolata come media **geometrica**
 - **Statistiche di dispersione**: Ci dicono di quanti si disperdono i valori rispetto a dei valori centrali o altri valori. Possono essere:
 - * *Varianza* (scarto quadratico medio)
 - * *Deviazione Standard* (la radice quadrata della varianza)
 - * *Range* (differenza tra valor massimo e valor minimo) o il *midrange point* (media della somma tra minimo e massimo).
 - * *Percentili* (il p -esimo percentile e' il valore tale che il p -per cento degli esempi hanno il valore della feature piu' piccolo di tale valore)

- **Statistiche di forma:** indicano quanto una distribuzione e' concentrata intorno ad un *picco* (*peakedness*):
 - * *Skeweness*: definita come m_3/σ^3 , cioe' il rapporto tra il terzo momento centrale e deviazione standard al cubo. Un valore positivo indica che la coda a destra della distribuzione e' piu' lunga di quella a sinistra. Viceversa se negativo.
 - * *Kurtosis*: definita come m_4/σ^4 . Spesso si considera l'eccesso di kurtosis rispetto al kurtosis della Gaussiana (3). Piu' tale eccesso e' alto piu' la distribuzione ha un picco piu' evidente che nella distribuzione Gaussiana normale.

Momento Centrale: Il k -esimo momento centrale puo' essere visto come una generalizzazione dello scarto quadratico medio. In generale si calcola come

$$m_k = \frac{1}{n} \sum_{i=1}^n (x_i - \mu)^k$$

Notiamo che il primo momento centrale e' sempre 0 perche' gli scarti positivi e negativi si cancellano tra loro, mentre $k = 2$ corrisponde alla varianza e con $k = 3$ puo' essere positivo o negativo.

- Abbiamo detto che lo spazio delle istanze e' rappresentato come $\mathcal{X} = \mathcal{F}_1 \times \dots \times \mathcal{F}_d$. Ogni istanza e' rappresentata come un vettore (v_1, \dots, v_d) in cui il valore v_i rappresenta il valore della feature i -esima.
- Avere troppe features appesantisce troppo gli algoritmi di ricerca, per cui si cerca sempre di ridurle il piu' possibile per evitare la maledizione della dimensionalita'. In questo caso, quindi, stiamo dicendo che l'informazione necessaria e' espressa solo dalle d features, per cui tutte le altre vengono filtrate.
- Le features strutturate consistono nell'accorpamento di una o piu' features in una sola feature. In questo modo si riduce il numero di features complessivo, pure senza dover eliminare features che potrebbero essere significative.
- Il problema principale della creazione di features strutturate e' l'esplosione combinatoriale in termini dei valori delle features previste

10.2 Trasformazioni delle Features

- Ogni modello che abbiamo visto, tratta diverse tipologie di features in modi differenti:
 - I modelli probabilistici trattano ogni feature come se fosse *categorica*
 - I modelli ad albero ignorano la scala delle feature quantitative considerandone solo l'ordine (di fatto trattandole come features ordinali)

- I modelli geometrici (basati sulla distanza) sono in grado di trattare solo features quantitative
- La notizia positiva e' che possiamo passare da una tipologia di feature ad un'altra tramite delle trasformazioni. La tabella seguente riassume le varie trasformazioni necessarie a passare da una tipologia all'altra.

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

Figura 10: Overview delle possibili trasformazioni per passare a varie tipologie di trasformazioni. Sulle righe ci sono le tipologie di partenza, mentre sulle colonne ci sono le tipologie di arrivo

10.2.1 Operazioni di discretizzazione

- Concernono le operazioni per trasformare variabili di tipo quantitativo in variabili di tipo ordinale (ordinati ma finiti) e categorico. Le operazioni di Thresholding sono dei casi specifici di discretizzazione.
- Le operazioni di discretizzazione possono essere **supervisionate** o **non-supervisionate**

10.2.1.1 Discretizzazione non supervisionata

- Il Thresholding non supervisionato consiste nel dividere i valori in due intervalli chiamati *bins*. Il threshold puo' essere dato dalle statistiche di tendenza centrale.
- Altre operazioni comuni di discretizzazione non supervisionata sono invece:
 - **Equal-Width partitioning**: Consiste nel dividere il range in N intervalli di uguale lunghezza data da $MAX - min / N$. E' il piu' semplice da ottenere ma gli outliers dominano la scelta dei bin piu' estremi. Se la distribuzione e' sbilanciata non gestiscono bene i dati, per cui ci potrebbero essere bins anche vuoti.
 - **Equal-Depth partitioning**: consiste nel dividere il range in N intervalli, ognuno contenente approssimativamente lo stesso numero di istanze (cosi' come si fa nei percentili). Cosi' facendo ottengo dei bins che hanno quasi tutti lo stesso numero di istanze.
 - **k-Means**: possiamo utilizzarlo in casi di features univariate

10.2.1.2 Discretizzazione supervisionata

- L'operazione fondamentale nella discretizzazione supervisionata e' quella di trattare le feature in questione come un *ranker*: si ordinano i dati in base alla feature e successivamente si a a vedere nel segmento identificato dalla etichetta di classe, se in quel segmento l'etichetta di classe e' uniforme o variabile.
- I segmenti identificati dall'etichetta di classe sono chiamati **run(s)**

Discretizzazione per partizionamento ricorsivo:

- Similmente a quello che si fa nei feature trees, si utilizza il principio di *entropia*
- L'idea e' quella di prendere il dataset, calcolarne l'entropia e considerare tutti i possibili split, scegliendo quello che ottimizza l'*Information Gain*.
- Lo split generera' cosi' due intervalli, per cui il passo successivo e' quello di applicare ricorsivamente lo stesso principio per i due intervalli generati.
- L'algoritmo termina quando le probabilita' empiriche sono le stesse in tutto il ranking oppure i gli intervalli (bins) sono *puri* o hanno lo stesso valore di feature.

Algorithm 18: Recursive Partitioning algorithm

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
2   | return  $\emptyset$ 
3 end
4 split  $S$  into  $S_l$  and  $S_r$  using threshold  $t$  that optimizes  $Q$ 
5  $T_l = \text{RecPart}(S_l, f, Q)$ 
6  $T_r = \text{RecPart}(S_r, f, Q)$ 
7 return  $T_l \cup \{t\} \cup T_r$ 

```

- La scoring function Q potrebbe non essere per forza l'Information Gain, ma e' valido anche ad esempio il criterio del Minimum Description Length
- Un alternativa dell'algoritmo ricorsivo che e' di tipo *divisivo* e' data dall'algoritmo [AggloMerge](#), che invece e' di tipo *agglomerativo*. L'idea e' quella di partire dai bin piu' fini di tutti (singoli esempi) andandoli a "fondere" in modo da ottenere sottoinsiemi piu' ampi, ripetendo fino alla stopping condition.

Algorithm 19: Recursive Partitioning algorithm

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 initialize bins to data points with the same scores
 - 2 merge consecutive pure bins
 - 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
-

- In questo caso, la scoring function e la condizione di stop puo' essere la funzione [Chi-square](#).

10.2.1.3 Chi-Square

- Consideriamo un esempio. Suponiamo che l'algoritmo [AggloMerge](#) abbia inizializzato i bins nel modo seguente:

$\ominus | \oplus | \ominus \ominus \ominus | \oplus \oplus | [\ominus \oplus] | \ominus \ominus \ominus$

consideriamo solo l'ultimo split ($[\ominus \oplus] | \ominus \ominus \ominus$) e costruiamone la tabella di contingenza

	Left Bin	Right Bin	
\oplus	1	0	1
\ominus	1	3	4
	2	3	5

- Il criterio χ^2 si basa sul confronto delle frequenze osservate (nella tabella di contingenza) rispetto alle frequenze che osserverei nelle due celle se le variabili fossero statisticamente indipendenti
- Dalla tabella possiamo stimare che $P(\text{LeftBin}) = 2/5, P(\text{RightBin}) = 3/5, P(\oplus) = 1/5, P(\ominus) = 4/5$. Se ipotizziamo che la scelta della classe sia indipendente dalla scelta del bin possiamo calcolare le probabilita' marginali, moltiplicando le probabilita' nel modo seguente:

$$P(\text{LeftBin}) \cdot P(\oplus) \cdot Tot = \frac{2}{5} \cdot \frac{1}{5} \cdot 5 = 0.4$$

$$P(\text{LeftBin}) \cdot P(\ominus) \cdot Tot = \frac{2}{5} \cdot \frac{4}{5} \cdot 5 = 1.6$$

Per cui possiamo riscrivere la tabella dei valori attesi come:

	Left Bin	Right Bin
\oplus	0.4	0.6
\ominus	1.6	2.4

- Che ci indica essenzialmente la probabilit  di ottenere positivi/negativi nel bin sinistro/destro in caso le classi e i bins fossero indipendenti.
- La statistica **Chi-Square**, fa il confronto tra i valori osservati (veri) nella tavola di contingenza con quelli attesi, normalizzando rispetto a quelli attesi. Piu' nello specifico, chi squared   uguale alla somma degli scarti al quadrato tra **valore reale** e **valore atteso** normalizzati rispetto ai valori attesi, per cui il risultato rispetto all'esempio   pari a:

$$\chi^2 = \frac{(1 - 0.4)^2}{0.4} + \frac{(1 - 0.6)^2}{0.6} + \frac{(3 - 2.4)^2}{2.4} + \frac{(1 - 1.6)^2}{1.6} = 1.88$$

- Piu'   alta la statistica, piu'   alto lo scarto totale, per cui significa che nel complessivo le frequenze osservate si discostano dalle frequenze attese, che sono quelle che mi aspetto di ottenere in caso di indipendenza statistica tra i due criteri (bins e classe)
- L'algoritmo quindi effettuer  i merge tra i bins che hanno il valore di **Chi-squared** piu' basso
- La stopping conditon   data dal valore critico della statistica dato dal *p-value*. Se tale valore supera quel threshold, l'algoritmo termina.

10.2.2 Operazioni di Normalizzazione

- La normalizzazione viene effettuata principalmente per neutralizzare l'impatto diverso che possono avere le features quantitative che utilizzano diverse scale, soprattutto in modelli che necessitano di calcolare la distanza.
- Essenzialmente ci sono due tipologie di normalizzazione:
- **Normalizzazione statistica** che consiste nella normalizzazione per mezzo dello **z-score**:

$$z = \frac{x - \mu}{\sigma}$$

per cui la feature in questione diventer  a media nulla ($\mu = 0$) e varianza unitaria ($\sigma = 1$).

- **Normalizzazione Min-Max**: che consiste nel ridurre i valori della features all'interno del range $[0, 1]$. L'idea   quella di applicare uno scaling lineare $f \rightarrow (f - l)/(h - l)$, in cui $[l, h]$ sono i valori estremi della feature. Tale scaling a volte richiede di fare delle ipotesi sul valore minimo e massimo (l, h) .

10.2.3 Operazioni di Calibrazione

- La calibrazione serve ad aggiungere informazioni qualitative in riferimento alla scala delle features che non ce l'hanno (ordinali o categoriche). Essa si colloca in un contesto di tipo supervisionato
- Calibrare le features permette a modelli come il classificatore lineare di trattare feature categoriche come features ordinali
- Per fare un esempio, prendiamo il caso di una classificazione binaria. Il problema della calibrazione di feature puo' essere quindi formalizzato nel seguente modo:

Calibrazione: Data una feature $F : \mathcal{X} \rightarrow \mathcal{F}$, trova la feature calibrata $F^C : \mathcal{X} \rightarrow [0, 1]$ tale per cui $F^C(x)$ e' dato dalla stima a posteriori $F^C(x) = P(\oplus | v)$ dove $v = F(x)$ e' il valore della feature originale per x .

- Siccome stiamo considerando la probabilita' a posteriori della classe positiva, per cui si dice che siamo *biased* verso la classe positiva

10.2.3.1 Features Categoriche

- La calibrazione di features categoriche consiste semplicemente nella collezione delle frequenze relative sul training set
- Prendiamo un esempio. Supponiamo di avere un dataset composto da una sola feature categorica che ci indica se il soggetto e' obeso o meno, etichettato con una classe positiva o negativa (che indica la presenza di diabete o meno). Collezioniamo delle statistiche e otteniamo che 1 persona su 18 obesa ha il diabete, mentre in quelle non obese sono 1 su 55. Da questa statistica possiamo stimare che

$$P(class = diabetes | obese = True) = \frac{1}{18} P(class = diabetes | obese = False) = \frac{1}{55}$$

- Questi valori possono essere poi utilizzati come valori calibrati della feature
- Il problema e' che in questo caso (come anche in altri) la probabilita' a priori (cioe' la distribuzione delle classi) non e' uniforme. Se si rimembra la formula di Bayes, la probabilita' a posteriori e' il prodotto della likelihood e della probabilita' a priori (distribuzione delle classi). Se le distribuzioni non sono uniformi, la probabilita' a priori (che viene moltiplicata per la likelihood) pesa notevolmente sul risultato finale.
- Per prendere in considerazione le distribuzioni differenti, consideriamo l'*odds* a priori, che e' il

rapporto tra le probabilita' della classe positiva su quella negativa $P(\oplus)/P(\ominus)$.

$$\underbrace{\frac{P(\oplus | v)}{P(\ominus | v)}}_{o=p/(1-p)} = \underbrace{\frac{P(v | \oplus)}{P(v | \ominus)}}_l \cdot \underbrace{\frac{P(\oplus)}{P(\ominus)}}_c$$

- Esplicitando il likelihood ratio ($l = o/c$) e' possibile neutralizzare l'effetto della distribuzione delle classi data dall'odds della distribuzione a priori
- Se poniamo che $p = m/n$ e quindi $o = \frac{m/n}{1-m/n}$ otteniamo

$$calibrated_feature(x) = \frac{m}{m + c(n - m)}$$

Nel caso volessimo ottenere una Laplace Correction basta aggiungere un conteggio ad m e due a n :

$$calibrated_feature(x) = \frac{m + 1}{m + 1 + c(n - m + 1)}$$

10.2.3.2 Features ordinali e quantitative

- Nel caso delle features ordinali e quantitative, e' possibile discretizzarle per poi calibrarle come se fossero delle feature categoriche
- Un'altra tipologia di calibrazione consiste nell'impiego della funzione sigmoide, riassunta dai seguenti passaggi:

1. Stima le medie delle classe μ^+, μ^- e la deviazione standard σ
2. Trasforma i valori della feature $F(x)$ negli z-score $z = \frac{x - \mu}{\sigma}$
3. Ri-scala lo z-score a $F^d(x) = d' \cdot z(x)$ con $d' = \frac{\mu^+ - \mu^-}{\sigma}$
4. Applica la trasformazione sigmoidale in modo da ottenere le probabilita' calibrate:

$$F^C(x) = \frac{\exp(F^d(x))}{1 + \exp(F^d(x))}$$

- La seconda tecnica di calibrazione per questa tipologia di features, e' chiamata *calibrazione isotonica*:

1. Ordina le istanze di train rispetto ai valori originali della feature e creane la ROC curve (sposta a destra di uno se classe e' negativa, sposta verso l'alto se la classe e' positiva)
2. Per ogni segmento della curva, conta il numero di positivi (m_i) e il numero totale delle istanze (n_i)

3. Discretizza la feature secondo i segmenti della curva. I singoli valori della feature calibrata saranno calcolati in ogni segmento i per mezzo della seguente formula:

$$v_i^C = \frac{m_i + 1}{m_i + 1 + c(n_i - m_i + 1)}$$

In caso sia richiesta una scala additiva, si utilizza la seguente formula

$$v_i^d = \ln \left(\frac{v_i^C}{1 - v_i^C} \right) = \ln(v_i^C) - \ln(1 - v_i^C)$$

10.3 Discretizzazione per la riduzione del rumore

- Come già sappiamo, in alcuni dataset alcune misurazioni di features continue potrebbero presentare forti rumori. Siccome la discretizzazione consiste nel trasformare un range continuo di valori in diversi sottointervalli (*bins*), potrebbe risultare utile per appiattire la quantità di errori. Questo perché i valori discreti sono considerati più stabili rispetto ai valori di tipo continuo, dal momento che sia i valori che gli errori vengono “aggregati” all’interno dei bins

10.3.1 Imputazione

- Un altro problema che si può presentare è che in alcuni dataset i valori delle feature potrebbero mancare per una determinata istanza. Tale fenomeno è detto **imputazione** e tipicamente per mitigarlo si possono usare due strategie:
 1. Nei problemi di classificazione si può stabilire la media, la moda e la mediana di ogni classe e usare uno di questi valori al posto del valore mancante (ovviamente rispetto alla classe a cui appartiene l’istanza)
 2. Un’altra possibilità sarebbe quella di apprendere un modello predittivo per l’attributo che presenta valori mancanti e usare tale modello per predire tali valori

10.3.2 Costruzione di features

- Per costruire nuove features da diverse features di partenza, si può partire costruendo il prodotto cartesiano dei domini delle features. Ad esempio, questa tecnica può essere utilizzata per migliorare Naive Bayes, poiché supera l’assunzione che le features siano indipendenti, riducendo il bias del modello verso le features indipendenti.
- Un’altra tecnica può essere quella di prendere le combinazioni aritmetiche o polinomiali delle features quantitative (metodi Kernel), in cui i valori di tale feature sono i valori del polinomio per quell’istanza

- Oppure si potrebbe imparare un concetto con il metodo di subgroup discovery e rappresentare tale gruppo con una feature booleana

10.4 Selezione di features

- Una volta costruite nuove features, e' spesso buona pratica selezionarle per diverse ragioni quali:
 - Velocizzare l'apprendimento poiche' riduce lo spazio di ricerca
 - Ridurre l'overfitting (in quanto anche la quantita' di features puo' favorire l'overfitting)
 - Ridurre il problema della maledizione dell'altra dimensionalita' (gli algoritmi si perdono nello spazio di ricerca)
- Ci sono due approcci principali nelle selezioni di features: l'approccio di tipo **filtro** e l'approccio di tipo **wrapper**

10.4.1 Approccio Filtro

- Consiste nel selezionare le features in base ad uno *score*
- Tale score viene calcolato applicando delle metriche (spesso di tipo supervisionato) quali l'information Gain, il Chi-Square, il coefficiente di correlazione ecc.. che vengono calcolate utilizzando tutte le istanze nel training set
- Vengono poi selezionate le features con lo score migliore
- Un algoritmo che si basa sull'approccio filtro e' l'algoritmo **Relief**.
 - L'idea principale e' quella di campionare ripetutamente istanze dal training set
 - Considerando l'istanza campionata x , ne cerca l'istanza piu' vicina h chiamata *nearest hit* e quella piu' vicina della classe opposta m chiamata *nearest miss*.
 - Sulla base dei valori di x, h, m si calcola uno *score* su ogni feature f_i . L'idea e' quella che lo score aumenti se la differenza del valore della feature f_i tra x e h e' piu' piccolo della differenza della feature f_i tra x e m . (In sostanza, ci si aspetta che i valori siano in concordanza con la distanza tra le istanze)
 - Dopo m iterazioni, l'algoritmo divide lo *score* per m . Tale risultato e' chiamato *relevance score* della feature.
 - Infine, le features vengono selezionate se lo score di rilevanza supera un certo threshold τ

Algorithm 20: Relief Algorithm - compute *feature relevance score*

Input : dataset D of instances from instance space \mathcal{X}

Output: vector f in instance space \mathcal{X} of feature relevance score

```

1 initialize vector  $f$  to  $(0, \dots, 0)$ 
2 for  $m$  times do
3   | extract random sample instance  $x_i$  from  $D$ 
4   | find in  $D$  the nearest instance  $h_i$  of the ***same*** class of  $x_i$ 
5   | find in  $D$  the nearest instance  $m_i$  of the ***opposite*** class of  $x_i$ 
6   | for all components  $f_j$  of  $f$  do
7   |   |  $f_j \leftarrow f_j - (x_{ij} - h_{ij})^2 + (x_{ij} - m_{ij})^2$ 
8   | end
9 end
10  $f \leftarrow f/m$ 
11 return  $f$ 

```

- Un lato negativo dell'approccio a filtro e' che non tiene conto ne della *ridondanza* tra features ne della loro possibile *dipendenza*. Si immagini ad esempio il caso in cui due features prese separatamente abbiano una misura di valutazione scadente, mentre se considerate insieme potrebbero essere un'ottimo predittore

10.4.2 Approccio Wrapper

- L'approccio wrapper mira a risolvere i problemi descritti in precedenza. L'idea e' quella di determinare insiemi di features in cui la feature viene valutata solo se e' utile ai fini dell'apprendimento del modello nel contesto di altre features
- Il problema di questo tipo di approcci e' che il numero degli insiemi di features aumenta esponenzialmente (*PowerSet*) con il numero totale di features possibili
- Ci sono diversi metodi per cercare nello spazio di combinazioni di features, che solitamente impiegano metodi *greedy*:
 - *Forward selection*: parti da una singola feature e continua ad aggiungere altre feature finquando la misura di valutazione migliora
 - *Backward elimination*: parte dal set di tutte le feature ed elimina le features finquando la misura di valutazione smette di migliorare

10.5 Principal Component Analysis

- E' un metodo sia per **costruire** che per **selezionare** features. Lo scopo e' quello di favorire le features per il quale c'e' una maggiore varianza nei dati. Questo perche' si ipotizza che una minore varianza nei dati rifletta una minore quantita' di informazione utile.
- Consiste in una trasformazione che riduce la dimensionalita' del dataset al costo di introdurre una perdita di informazione controllata.
- Le nuove features calcolate sono chiamate *componenti principali*
- **PCA** e' riassumibile nei seguenti passi:
 1. Prima si *traslano* i dati in modo che la media di ogni dimensione venga annullata. Cioe' la media sia centrata sull'origine degli assi.
 2. Trova gli autovettori della matrice di scatter S
 3. Gli autovettori definiscono un nuovo spazio di rappresentazione delle istanze
- Matrice di scatter $S(d \times d)$: matrice in cui il σ_{ij} elemento e' n volte la covarianza tra la feature i e j su tutti gli oggetti.

$$\sigma_{ij} = \sum_{k=1}^n (x_{ki} - x_{*i})(x_{kj} - x_{*j})$$

σ_{ij} e' una misura di quanto fortemente le features i e j variano insieme.

- I valori λ_i (con $i = 0, \dots, d$) degli autovalori associati agli autovettori calcolati, rappresentano la varianza della nuova feature i
- Guardare su slides PCA. (C'e' veramente troppa roba da scrivere lol)

10.6 Singular Value Decomposition (SVD)

- Una tecnica simile alla PCA e' la SVD. La differenza e' che SVD al posto di calcolare le autocopie per la matrice di scatter, li calcola direttamente sulla matrice dei dati X
- Siccome in generale non e' una matrice quadrata, viene decomposta in diverse matrici

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

- U e' una matrice $n \times n$, matrice degli autovettori
- Σ e' una matrice $n \times d$, e' la matrice degli autovalori
- V e' una matrice $d \times d$, che contiene i patterns tra le features

DA RIVEDERE LEZIONE

11 Ensemble Learning

- L'ensemble learning e' una collezione di metodi atti ad apprendere una funzione obiettivo andando a far *apprendere diversi modelli e combinarli assieme*
- L'idea principale e' quella di partire da un dataset D e ottenere T diversi modelli derivando dataset diversi D_1, \dots, D_T . Una volta ottenuti vengono appresi con l'algoritmo di apprendimento A ottenendo M_1, \dots, M_T modelli, che vengono combinati infine nello stesso modello M
- Un modo molto comune per combinare i modelli ottenuti e' per mezzo del voto maggioritario:

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

dove:

- w_i e' il peso associato al modello M_i . Tipicamente e' compreso tra 0 e 1 e la loro somma totale e' 1.
- $f(x)$ e' una funzione di threshold
- Nella formula, $w_i > w_j$ significa " M_i e' piu' affidabile di M_j ". Tipicamente $w_i > 0$, ma potrebbe anche essere < 0 per indicare che il modello i e' sbaglia piu' di quanto fa giusto. In generale e' quindi una misura che indica quanto *buono* e' il modello (in relazione a tutti gli altri)
- A partire da questo schema generale di combinazione possiamo poi utilizzare diversi algoritmi per determinare come vengono generati i dataset D_T e come vengono calcolati i pesi w_i e la f

11.1 Bagging

- Il primo di questi algoritmi e' **Bagging** (*Bootstrap Aggregation*) [Breiman et.al., 1996]

Algorithm 21: Bagging - Train an ensemble of models from bootstrap samples

Input : dataset D ; ensemble size T ; learning algorithm \mathcal{A}

Output: ensemble of models whose proportions are combined by voting or averaging

```

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

```

- Bootstrap: insieme costruito campionando n elementi con rimpiazzamento da D
- $M(x)$ si ottiene attraverso una votazione semplice (maggioritaria). Per cui basta imporre che $w_i = \frac{1}{T}$ e $f(x) = x > 0.5$.

- Nel problema dell'ensemble learning, idealmente si vuole che tutti i T modelli che sono stati appresi siano diversi tra loro. Bagging promuove questa proprietà attraverso il bootstrapping.
- Grazie al sampling, Bagging ha inoltre la proprietà di costruire insiemi di esempi che seguono la distribuzione originale del dataset originale D
- La tecnica di Bagging funziona molto bene quando i classificatori hanno un'alta varianza perché se avessero una bassa varianza i vari classificatori differirebbero poco tra di loro e non si avrebbe un'effettivo miglioramento con il bagging. D'altra parte, classificatori con alta varianza differiscono di molto tra di loro, rendendo più efficace il bagging.

Mediamente, una replica di bootstrap conterrà circa il 63% di esempi unici del dataset originale. Questo perché la probabilità di **NON** estrarre un esempio è $(1 - \frac{1}{n})^n$ (l'esponente deriva dal fatto che i lanci sono indipendenti). Tale numero è un limite notevole che tende a $1/e = 0.3678\dots$, per cui ne segue che $1 - 0.3678\%$ degli esempi di D saranno inclusi nella replica di bootstrap.

11.2 Random Forest

- Una random forest è un'istanziatura del **Bagging** che utilizza alberi di decisione non potati (per migliorarne la diversità tra gli altri modelli) come modelli.
- L'idea è quella di considerare solo un sottoinsieme (scelto casualmente) delle features su cui fare split quando si costruisce il singolo albero.
- Più precisamente, si sceglie un sottoinsieme di cardinalità $|F'| = \sqrt{|F|}$. In questo modo si ottengono diversi modelli che sono meno precisi rispetto ad un albero se l'avessimo appreso "normalmente", ma che performano meglio quando viene effettuato il bagging.

11.3 AdaBoost

- AdaBoost (*Adaptive Boosting*) [Freund and Schapire et.al., 1997]

Una classe di concetti (dataset) è learnable (o **strongly learnable**) se esiste un algoritmo a tempo polinomiale per ottenere un modello con errore relativamente basso con un'alta confidenza per tutti i concetti della classe. Contrariamente, un modello più debole di apprendibilità (**weak learnability**) ha un rilassamento su questi vincoli, facendo cadere i requisiti che il learner debba essere in grado di raggiungere una precisione arbitrariamente elevata: un algoritmo di apprendimento debole può dare in output ipotesi che performino *leggermente* meglio di un random guess

- Ipotesi del problema del boosting: "è vero che i problemi di apprendimento fortemente apprendibili sono equivalenti ai problemi di apprendimento deboli?"

- Nel '90 e' stato visto da Kearns et.al. che non erano equivalenti per certe restrizioni imposte sulla distribuzione dello spazio delle istanze. Per cui era palusibile che i due problemi non fossero equivalenti anche in tutti gli altri casi
- Sempre nel '90, pero', Schapire dimostro' che i due problemi fossero effettivamente equivalenti.

Teorema: Una classe di concetti C e' apprendibile in modo debole se e solo se e' apprendibile in modo forte

- In altri termini, se abbiamo un algoritmo di apprendimento \mathcal{A} che sia in grado di generare una soluzione leggermente migliore di un random guess, allora esiste una procedura di boosting in grado di dare una soluzione **accurata a piacere**, ma che usa \mathcal{A} come algoritmo di apprendimento di base
- Il miglioramento aumenta inoltre esponenzialmente con il numero di volte con cui si utilizza \mathcal{A}

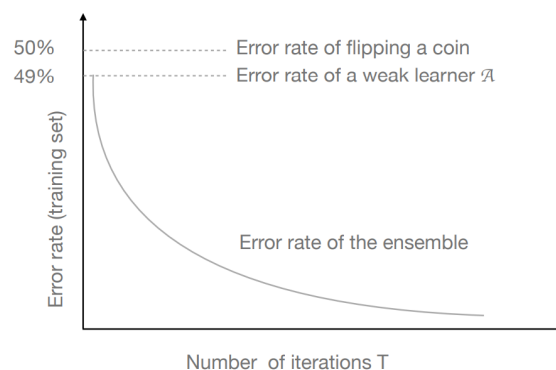


Figura 11: Grafico dell'error rate sul numero di iterazione T di AdaBoost. Si noti come decresca esponenzialmente se l'algoritmo scelto genera weak learners

- Il teorema di Schapire non invalida pero' il teorema di Kearns poiche' suppone che la distribuzione non sia uniforme, che si suppone venga modificata in base alle esigenze da una fase di preprocessing iniziale
- L'idea generale del Boosting e' la seguente:
 - Nel training set T assegna ad ogni esempio un peso
 - Man mano che il boosting va avanti, i pesi vengono *aggiustati* in base ai risultati delle predizioni:
 - * Il peso degli esempi che sono correttamente predetti vengono abbassati
 - * Il peso degli esempi che non sono correttamente predetti vengono alzati
- In questo modo man mano che va avanti la procedura di apprendimento, l'algoritmo di apprendimento viene forzato sempre di piu' a "concentrarsi" sulle porzioni dello spazio degli esempi che non sono correttamente classificate

- Possiamo notare inoltre che AdaBoost man mano che va avanti “*complica*” (impostando i pesi) sempre di più il problema di apprendimento, rendendo sempre meno valida l’ipotesi di ottenere un weak learner in output dall’algoritmo di apprendimento, per cui ha delle limitazioni sul numero di iterazioni

11.3.1 Algoritmo e Spiegazione

Algorithm 22: AdaBoost Algorithm - learn an ensemble model

Input : dataset D of labelled instances; weak learning algorithm \mathcal{A} ; no. iterations T

Output: ensemble model $M(x)$ using T models learned over D with \mathcal{A}

```

1  $w^1 = [\frac{1}{|D|}, \dots, \frac{1}{|D|}]$ 
2 for  $t \in \{1, \dots, T\}$  do
3    $m_t = \mathcal{A}(D, w^t)$ 
4    $\epsilon_t = \sum_{i=1}^n w_i^t I[y_i \neq m_t(x_i)]$ 
5    $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$ 
6   for  $i \in \{1, \dots, n\}$  do
7      $w_i^{t+1} = w_i^t \exp(-\alpha_t y_i m_t(x_i))$ 
8   end
9    $w^{t+1} = \text{normalize}(w^{t+1})$ 
10 end
11 return  $M(x) = \text{sign}(\sum_{t=1}^T \alpha_t m_t(x))$ 

```

- Nel precedente algoritmo:
 - \mathcal{A} e' un algoritmo che deve dare in output modelli con accuratezza *leggermente* maggiore del 50% (puo' anche essere ad esempio 50.1%)
 - Linea 1: I pesi vengono inizializzati allo stesso peso iniziale
 - Linea 3: Viene appreso un classificatore m_t sui dati coi pesi correnti w_t
 - Linea 4: Calcola l'errore pesato di tutti gli esempi che non sono stati classificati correttamente (la funzione indicatrice vale 1 se l'argomento e' True, 0 altrimenti)
 - Linea 5: Calcola il peso associato al voto del classificatore corrente ($\alpha_t \in [0, +\infty]$)
 - Linea 6-9: Aggiorna i pesi nel dataset considerando anche quelli precedenti
- Il perche' viene calcolato in quel modo α ha delle ragioni teoriche precise. Quella principale e' che l'algoritmo visto puo' essere derivato come un *algoritmo di minimizzazione greedy della perdita esponenziale pesata dell'ensemble*
- Ad ogni step AdaBoost cerca in modo greedy il peso α_t che minimizza M_T (dove M_T e' l'ensemble del passo attuale)

- Assumiamo ad esempio che abbiamo già costruito parte dell'ensemble

$$M_{T-1}(x) = \sum_{t=1}^{T-1} \alpha_t m_t(x)$$

e concentriamoci su come scegliere m_T e α_T sotto le condizioni che la perdita sia esponenziale

- Per definizione, sappiamo innanzitutto che $M_T(x) = M_{T-1}(x) + \alpha_T m_T(x)$, per cui basta applicare la definizione di errore esponenziale e effettuare delle manipolazioni algebriche

$$\begin{aligned} E &= \sum_i e^{-y_i M_T(x_i)} \\ &= \sum_i e^{-y_i (M_{T-1}(x_i) + \alpha_T m_T(x_i))} \\ &= \sum_i e^{-y_i M_{T-1}(x_i)} \cdot e^{-y_i \alpha_T m_T(x_i)} \\ &= \sum_i w_i^T e^{-y_i \alpha_T m_T(x_i)} \\ &= \sum_{\{i|y_i=m_T(x_i)\}} w_i^T e^{-\alpha_T} + \sum_{\{i|y_i \neq m_T(x_i)\}} w_i^T e^{\alpha_T} \end{aligned}$$

e otteniamo che E è la somma degli errori di cui la classificazione è corretta e quelli in cui la classificazione non è corretta.

- Nel passo 4 si impone che:

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

- Ora w_i^T è effettivamente la formulazione ottenuta nell'algoritmo (linea 6-9) poiché

$$\begin{aligned} w_i^T &= e^{-y_i M_{T-1}(x_i)} \\ &= e^{-y_i (\alpha_{T-1} m_{T-1}(x_i) + M_{T-2}(x_i))} \\ &= e^{-y_i m_{T-2}(x_i)} \cdot e^{-y_i \alpha_{T-1} m_{T-1}(x_i)} \\ &= w_i^{T-1} \cdot e^{-y_i \alpha_{T-1} m_{T-1}(x_i)} \end{aligned}$$

che è appunto la formulazione che si trova nell'algoritmo.

- Con la formula trovata dell'errore, possiamo anche rispondere ad un'altra domanda. Il fatto è che si vorrebbe trovare un modo per dire all'algoritmo quale tra gli m_T scegliere. Il problema è complicato però dal fatto che in entrambe nelle somme

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

compare in entrambe m^T . L'idea e' quella di riuscire a "rimuovere" da una delle due somme m^T in modo da ritrovarci a discutere solo una delle due somme, per cui sommiamo e togliamo la stessa quantita' $\sum_{\{i|y_i \neq m_T(x_i)\}} w_i^T e^{\alpha_T}$

$$\begin{aligned} E &= \sum_{\{i|y_i=m_T(x_i)\}} w_i^T e^{-\alpha_T} + \sum_{\{i|y_i \neq m_T(x_i)\}} w_i^T e^{\alpha_T} \\ &= \sum_{\{i|y_i=m_T(x_i)\}} w_i^T e^{-\alpha_T} + \sum_{\{i|y_i \neq m_T(x_i)\}} w_i^T e^{\alpha_T} + \sum_{\{i|y_i \neq m_T(x_i)\}} w_i^T e^{\alpha_T} - \sum_{\{i|y_i \neq m_T(x_i)\}} w_i^T e^{\alpha_T} \\ &= \sum_i w_i^T e^{-\alpha_T} + \sum_{\{i|y_i \neq m_T(x_i)\}} w_i^T (e^{\alpha_T} - e^{-\alpha_T}) \end{aligned}$$

abbiamo trovato quindi che per minimizzare l'errore rispetto a m_T e' sufficiente minimizzare

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

- Facciamo un passo indietro ora e vogliamo trovare il peso ottimale α_T che minimizzi il numero di missclassificazioni. Per scegliere il peso ottimale, cerchiamo un α_T che annulla la derivata di E .

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

Troviamo quindi che

$$\begin{aligned} e^{\alpha_T} \sum_{\{i|y_i \neq m_t(x_i)\}} w_i^T - e^{-\alpha_T} \sum_{\{i|y_i = m_t(x_i)\}} w_i^T &= 0 \\ e^{\alpha_T} \epsilon_T - e^{-\alpha_T} (1 - \epsilon_T) &= 0 \\ \alpha_T &= \frac{1}{2} \ln \left(\frac{1 - \epsilon_T}{\epsilon_T} \right) \end{aligned}$$

dove nel secondo passaggio imponiamo che

$$\epsilon_T = \sum_{\{i|y_i \neq m_t(x_i)\}} w_i^T$$

11.3.2 Apprendimento da dataset pesati

- La richiesta che l'algoritmo di apprendimento \mathcal{A} sia in grado di apprendere tenendo conto dei pesi puo' essere soddisfatta in due modi principali:
 - Modificando le misure per l'apprendimento in modo che tengano conto dei pesi associati agli esempi (es. information gain negli alberi di decisione, pesare i voti per kNN, ecc..)

- Facendo un’inflazione artificiale del dataset in accordo ai pesi (chiamata dataset resampling)
- Nell’articolo queste due metodologie sono denominate come **boosting by weighting** e **boosting by resampling**.
- Vediamo ora un algoritmo di resampling

Algorithm 23: Resample(D, w, n) - Resample the dataset according to the distribution induced by the weights

Input : dataset D ; weights w ; number of instances of the new dataset n

Output: weighted dataset D'

```

1  $D' = \emptyset$ 
2 for  $n$  times do
3    $v = \text{random}(0, 1)$ 
4    $D' = D' \cup \{x_k\}$  with  $k$  s.t.  $\sum_{i=1}^{k-1} w_i < v \leq \sum_{i=1}^k w_i$ 
5 end
6 return  $D'$ 
```

- L’idea e’ essenzialmente che la probabilita’ di includere x_k sia proporzionale al peso di x_k . In sostanza la somma iniziale somma tutti i pesi degli x precedenti e l’altra somma quelli successivi. In quel modo permette di far stare k “dentro” l’area denominata dal peso di x_k
- In linea di massima e’ meglio campionare dataset che abbiano la stessa cardinalita’ del dataset originale $|D'| = |D|$
- Ovviamente, utilizzare il resampling puo’ appesantire le risorse in fase di apprendimento, per cui sarebbe meglio utilizzare quando possibile il metodo di weighting.

11.4 Perché ensemble learning funziona

- Ci sono diverse motivazioni del perché l’ensemble learning funziona. In primo luogo e’ per una proprietà della decomposizione del Bias/Variance.
- Sappiamo che per il teorema del bias/varianza tradeoff possiamo scomporre l’errore in

$$\mathbb{E}(\epsilon(x)) = \text{Bias}(x) + \text{Var}(x) + \text{Noise}(x)$$

dove $\epsilon(x)$ e’ l’errore medio di un algoritmo su un esempio x .

- La componente di Bias e’ legata al fatto che il tipo di ipotesi che sto considerando (del modello) non si adatta abbastanza bene all’ipotesi reale

- La componente di Varianza invece una quantita' che ci dice quanta parte dell'errore, se noi ripetiamo l'esperimento piu' e piu' volte non apprendiamo lo stesso concetto, ma ne apprendiamo delle variazioni che sono indotte da dei cambiamenti del dataset. Piu' queste variazioni sono grandi piu' l'algoritmo non e' stabile perche' a fronte di piccole variazioni nel dataset cambia anche l'ipotesi in modo significativo.
- E' stato dimostrato che tutti i metodi di ensemble riducono sia la componente di bias che la componente di varianza dell'errore. E' stato dimostrato empiricamente pero' che il *Bagging* agisce principalmente come macchina di riduzione della varianza, per cui e' piu' efficace con classificatori quali alberi di decisione che hanno basso bias e alta varianza.
- Questa proprieta' e' spiegabile in termini probabilistici. L'idea e' quella di considerare la variabile casuale che conta il numero di errori di missclassificazione fatti dall'ensemble.
- Sia quindi $X \text{ Bin}(p, T)$ la variabile binomiale che conta il numero di errori su un numero T di test, con probabilita' p di fallimento per il singolo test.
- Assumiamo che in un caso di classificazione binaria abbiamo un errore se *piu' della meta' dei membri dell'ensemble* sbagliano.

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

- Se assumiamo che la probabilita' del singolo weak learner e' di fare un errore sia leggermente piu' piccola di 0.5, allora vediamo che nella distribuzione binomiale, al crescere dei weak learners (e quindi del parametro T), la distribuzione si schiaccia sempre di piu' verso il valor medio, fino a diventare una funzione Delta di Dirac.
- Di conseguenza, la probabilita' totale di fare errore descritta in precedenza tende a 0 (siccome e' un'area)

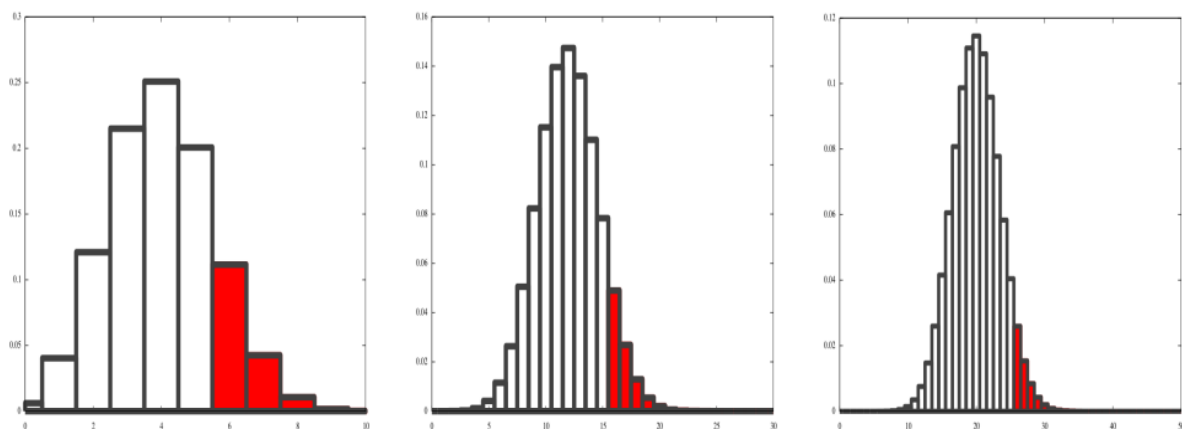


Figura 12: Effetto del Bagging sulla distribuzione binomiale della variabile X che descrive la probabilità l'errore dell'ensemble descritta in precedenza. Si noti come all'aumentare degli ensambles la curva si schiacci sempre di piu' sul valor medio. In rosso e' denotata l'area di $P(X > \lfloor T/2 \rfloor)$

- Evidenze empiriche ci dicono invece che *AdaBoost* e' molto efficiente a ridurre la componente di bias dell'errore, riducendo anche la componente di varianza
- In generale, *AdaBoost* e' piu' efficace su classificatori ad alto bias (eg. modelli lineari)

11.5 Benefici dell'Ensemble Learning

- Essenzialmente i benefici dell'ensemble learning sono 3:
 - **Statistici:** Possiamo vedere un ensemble come un'approssimazione di un classificatore di Bayes Ottimale. Questo perche' un classificatore di questo tipo prende un voto di maggioranza di tutte le ipotesi pesate per la loro probabilita' a posteriori. Ma l'ensemble fa una cosa simile pesando le ipotesi (modelli) appresi in base a dei pesi che sono piu' consistenti coi dati
 - **Rappresentazionali:** La funzione da apprendere potrebbe non essere mediante dei classificatori individuali, ma potrebbe essere approssimata molto bene da un ensemble averaging
 - **Computazionali:** Tutti gli algoritmi di apprendimento fanno una sorta di ricerca all'interno dello spazio delle ipotesi che puo' contenere molti minimi locali. L'ensemble inserisce una stocasticita' facendo ripartire gli algoritmi da punti diversi nello spazio delle ipotesi in modo da permettere di raggiungere potenzialmente il minimo globale.

12 Esperimenti nel Machine Learning

- Nei capitoli precedenti abbiamo visto che esistono diverse condizioni o distribuzioni di dati in cui alcuni modelli sono ottimali (per esempio per Naive Bayes o per il classificatore lineare di base)
- In applicazioni reali, però, vorremmo sapere se la convergenza dell'algoritmo di apprendimento al modello ottimale possa avvenire in un tempo praticabile, per cui abbiamo bisogno di dati provenienti da tali domini applicativi per poter **testare le performance degli algoritmi di apprendimento e dei modelli appresi**.
- L'idea è quella di valutare diversi modelli anche su datasets differenti, ottenendo diverse misure che serviranno a rispondere a determinate domande sui modelli utilizzati.
- La scelta delle metriche di performance da utilizzare, è guidata dalle **assunzioni** che vengono fatte sulle condizioni operative del modello:

- **Accuratezza**: buona quando la distribuzione delle classi nel test set è *rappresentativa delle condizioni operative del modello*

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

- **Average Recall**: buona quando la distribuzione delle classi nel test set è *ugualmente bilanciata* (50

$$avg - recall = \frac{tpr}{2} + \frac{tnr}{2}$$

- **Precision and recall**: media tra *precision* (abilità del modello di classificare la classe positiva correttamente) e *recall*. Il problema è che questa stima ignora i *true negatives* dal momento che considera solo la classe positiva.

$$precision = \frac{TP}{TP + FP} \quad recall = \frac{TP}{TP + FN}$$

- **AUC (Area Under The Curve)**: buona per i ranker classifiers
- All'interno delle misurazioni, è quasi impossibile non trovare dell'errore. Pensiamo ad esempio di misurare più volte l'altezza di una persona. Sicuramente tra una misura e l'altra ci aspettiamo delle variazioni.
- Questa variazione può essere modellata trattando le misure come una variabile casuale. Siccome la media e la varianza di tale variabile non sono note, si possono stimare attraverso un trucco molto noto.
- Si considera la media di k misurazioni, in modo da diminuire la varianza totale della stima fino a $\frac{\sigma^2}{k}$, ma funziona solo se le misurazioni sono *indipendenti* l'una dall'altra. Dim.

$$Var(\bar{X}_k) = Var\left(\frac{1}{k}(X_1 + \dots + X_k)\right) = \frac{1}{k}Var(X_1 + \dots + X_k) = \frac{k\sigma^2}{k^2} = \frac{\sigma^2}{k}$$

Questo segue dalla proprietà della varianza. supponendo l'indipendenza degli eventi ($Var(\alpha \cdot X) = \alpha^2 \cdot Var(X)$, $Var(\sum_i X_i) = \sum_i Var(X_i)$).

- Supponiamo ora di non voler stimare l'altezza ma di voler stimare l'accuratezza di un classificatore. Un modo naturale per modellare la situazione è trattando ogni istanza del test set passata al classificatore come un test di Bernoulli (classificazione corretta/non corretta).
- Poniamo con a (aka θ) la probabilità di una classificazione corretta di un singolo test
- Noi vogliamo stimare \hat{a} , che effettivamente corrisponde all'accuratezza complessiva del modello (cioè la capacità, dato un esempio qualsiasi, di classificarlo correttamente)
- Possiamo stimare l'accuratezza utilizzando il valor medio $\hat{a} = A/n$ in cui A è il numero di esempi correttamente classificati nel *test set* dal modello
- Dal momento che ci sono più istanze nel test set, per modellare diversi lanci si utilizza una *variabile Binomiale*. La funzione di distribuzione di probabilità della variabile è una funzione con due parametri:
 - a = probabilità di successo di un singolo esperimento
 - n = numero di esperimenti eseguiti
 - $f(a, n) = a(1 - a)^n$
- La varianza di un *singolo* esperimento Bernoulliano è pari a $a(1 - a)$. Possiamo quindi usare il trucco precedente e dividere la quantità per il numero di lanci in caso gli esperimenti siano indipendenti, per cui $a(1 - a)/n$ è la stima della varianza di A .
- Ovviamente utilizziamo la **stima** di a , per cui parliamo di **stima della varianza** $\hat{a}(1 - \hat{a})/n$.

12.1 Cross-Validation

- Se volessimo migliorare ulteriormente la stima, possiamo dividere il dataset in porzioni indipendenti e fare la stima dell'accuratezza in ciascuna di queste porzioni indipendenti.
- L'idea principale è quella di fare la media di k stime indipendenti \hat{a}_i e calcolarne la loro varianza campionaria $\frac{1}{k-1} \sum_{i=1}^k (\hat{a}_i - \bar{a})^2$ dove $\bar{a} = \frac{1}{k} \sum_{i=1}^k \hat{a}_i$.
- Le k stime di a possiamo ottenerle campionando *indipendentemente* dal dataset k sottoinsiemi di cardinalità n e stimare a in ognuno di loro. (Viene fatto quando abbiamo molti dati, es *Big Data*)
- Quando non abbiamo molti dati a disposizione, però, non possiamo fare la procedura seguente e utilizziamo la cosiddetta *cross-validation*
- L'idea della cross validation è quella di partizionare il dataset in k partizioni chiamate *fold*. Un *fold* viene messo da parte e viene considerato come test set, mentre viene fatto l'apprendimento sui rimanenti $k - 1$ folds. Il procedimento viene reiterato k volte fin quando tutti i folds sono stati considerati come test set.

- Per ottenere l'accuratezza basta fare la media delle accuratèzze di tutti i modelli sui test set. Quando siamo soddisfatti dei risultati possiamo infine fare il training sull'intero dataset

12.2 Intervalli di confidenza

- Supponiamo che la stima dell'accuratezza \hat{a} segua una distribuzione normale intorno alla *media reale* a con deviazione standard σ . Assumendo per il momento che si sappiano a priori questi parametri, possiamo calcolare per ogni intervallo la *likelihood* che la stima *cada in un intervallo* calcolando l'area sotto la curva della distribuzione normale in quell'intervallo
- Per calcolare l'intervallo di confidenza sono necessarie due cose:
 - Conoscere la distribuzione della stima
 - Conoscere i parametri della distribuzione
- Nel caso dell'accuratezza che abbiamo visto fin'ora, sappiamo che la distribuzione è binomiale, con varianza $\sigma^2 = [\hat{a}(1 - \hat{a})]/n$
- Calcolare l'intervallo di confidenza nel caso della binomiale non è semplice poiché non è simmetrica. Possiamo allora approssimare la distribuzione binomiale ad una normale se la condizione seguente è soddisfatta

$$na(1 - a) \geq 5$$

per cui possiamo utilizzare una Gaussiana assumendo che $\sigma = \sqrt{\hat{A}(1 - \hat{a})/n}$

- Sappiamo che l'intervallo di confidenza per la normale sarà $[a - 2\sigma, a + 2\sigma]$
- Gli intervalli di confidenza sono delle affermazioni su quanto è probabile che le stime su un parametro sconosciuto ricada all'interno di un certo intervallo, assumendo che questo parametro abbiamo una distribuzione in accordo all'**ipotesi nulla**.
- **Esempio:**
 - Supponiamo di fare l'ipotesi nulla che l'accuratezza sia distribuita intorno ad una media di 0.5, per cui si ottiene che $\sigma = 0.05$ (stimandola con la formula precedente) in un test set di 100 istanze
 - Avendo ottenuto dagli esperimenti una stima dell'accuratezza di 0.8, calcoliamo la probabilità che una stima sia ≥ 0.8 data l'ipotesi nulla. Questa probabilità è chiamata **p-value**.
 - Se il *p-value* è basso, l'ipotesi nulla è inverosimile siccome il valore si discosta dall'ipotesi nulla.
 - Si utilizzano dei thresholds predefiniti per accettare/rifiutare le ipotesi nulle, chiamati *livelli di significatività* α

- Ipotizziamo di utilizzare $\alpha = 5$, per cui abbiamo che $p = 1.9732 \cdot 10^{-9}$, il che e' molto piu' piccolo di $\alpha = 0.05$, dovendo quindi abbandonare l'ipotesi nulla a favore di una piu' verosimile.

12.3 Paired t-test

- Utilizziamo gli intervalli di confidenza negli esperimenti effettuati con la cross validation. In questi esperimenti utilizzeremo le osservazioni accoppiate negli stessi fold.
- Per una coppia di algoritmi calcoliamo la differenza di accuratezza in ogni fold
- Utilizziamo l'ipotesi nulla che la media della distribuzione delle differenze sia 0. Calcoliamo poi il p -value usando la distribuzione di t -student e rifiutiamo l'ipotesi nulla se il p -value sta sotto il nostro livello di significativita' α . Questo corrisponde a rifiutare l'ipotesi che non ci siano delle differenze tra i diversi algoritmi, dovendo riformulare un'ipotesi che ci dice che invece gli algoritmi hanno delle differenze.
- La distribuzione t -student si utilizza perche' pur assumendo che le differenze delle accuratezze siano distribuite in maniera normale non si ha comunque accesso alla deviazione standard reale delle differenze.
- t -student e' semplicemente una normale ma ha le code piu' "spesse". Il *grado di liberta'* indica quanto sono spesse le code. Siccome siamo in un'esperimento di cross-validation, i DOF saranno pari a *numero di folds*-1

12.4 Wilcoxon Test

- Il t -test non e' appropriato per comparare algoritmi su diversi datasets perche' le misure di performance non possono essere comparate tra loro siccome sarebbero sicuramente in disaccordo.
- Il *signed-rank test di Wilcoxon* e' un test che ci permette di comparare le performance di diversi algoritmi su diversi datasets.
 - L'idea e' quella di fare un ranking dei valori assoluti delle differenze delle misure
 - Calcoliamo poi la somma dei ranghi assegnati per differenze positive e negative prese separatamente e consideriamo la piu' piccola di queste somme come il test statistico.
 - La piu' piccola somma dei ranghi viene confrontata con il valore critico (corrispondente nella tabella): se minore o uguale allora l'ipotesi nulla (nessuna differenza in performance) viene rifiutata
- Per un numero grande di datasets (piu' di 25) questa statistica potrebbe essere approssimata alla distribuzione normale e si potrebbero utilizzare quindi gli intervalli di confidenza visti prima. Nel caso contrario si utilizza il Wilcoxon test

- Siccome questo test non fa nessuna assunzione sulla distribuzione delle differenze e' anche meno sensibile agli outliers
- Questa tipologia di test e' detta *non parametrica* nella terminologia statistica, cioe' significa che non assumono nessuna distribuzione. Sono l'opposto dei test parametrici come il *t-test* che assumono invece una distribuzione particolare.

In generale i test parametrici sono piu' potenti quando la distribuzione che si assume e' appropriata, ma potrebbero essere ingannevoli nel caso non lo sia

12.5 Friedman Test

- Se volessimo comparare k algoritmi su n datasets differenti, utilizzare il Wilcoxon's signed rank test porta a far crollare il livello di significativita' del test, per cui e' opportuno utilizzare il test di Friedman
- Come nel test di Wilcoxon, anche il test di Friedman e' un test non parametrico che utilizza il ranking delle performance
 - Preliminarmente si ordinano le performance dei k algoritmi dalla migliore (rank 1) alla peggiore (rank k)
 - Successivamente si costruisce una tabella in cui le singole entrate R_{ij} indicano il rango dell'algoritmo j sull' i -esimo dataset
 - Denotiamo poi con $R_j = (\sum_i R_{ij})/n$ il rango medio dell'algoritmo j . Sotto l'ipotesi nulla che tutti gli algoritmi sono uguali questa media dovrebbe essere uguale per tutti i j
 - Infine, per effettuare il test calcoliamo il rango medio effettivo $\bar{R} = \frac{k+1}{2}$ e lo confrontiamo con due quantita':
 - * Varianza rispetto ai modelli: $n \sum_j (R_j - \bar{R})^2$
 - * Varianza rispetto a tutte le osservazioni: $\frac{1}{n(k-1)} \sum_{ij} (R_{ij} - \bar{R})^2$ e confrontiamo il rapporto delle due varianze. Se questo rapporto e' piccolo, vuol dire che i ranghi assegnati ai vari metodi non sono sufficientemente diversi rispetto ai ranghi assegnati in tabella, in quanto la varianza complessiva della tabella intera e' piu' o meno equiparabile a quando vengono raggruppati per modelli e confrontati rispetto al rango medio
 - Tale rapporto e' detto *statistica di Friedman*. Per fare il test la confrontiamo con il valore critico. Se minore o guale, allora **NON RIFIUTIAMO** l'ipotesi nulla, se no la rifiutiamo.

12.5.1 Post-Hoc Test

- Siccome il test di Friedman ci da delle informazioni sul rango medio su tutti i modelli, un secondo test e' necessario per ottenere un'analisi che consideri le coppie di algoritmi. Tale analisi viene

effettuata con il **Post-Hoc test**, dopo aver effettuato il test di Friedman.

- L'idea e' quella di calcolare la *differenza critica (CD)* e di compararla rispetto alla differenza dei ranghi medi di due algoritmi
- Il **test di Nemenyi** calcola la differenza critica nel modo seguente:

$$CD = q_{\alpha} \sqrt{\frac{k(k+1)}{6n}}$$

dove q_{α} dipende dal livello di significativita' α cosi' come k .

- Se la differenza del rango medio di due algoritmi e' piu' grande di CD , allora rifiutiamo l'ipotesi nulla.
- Al posto di considerare tutte le coppie di modelli come nel test di Nemenyi, possiamo utilizzare una sua variante che confronta tutti i modelli con un solo modello di riferimento. Tale variante prende il nome di **test di Bonferroni-Dunn**.