

**APPUNTI SPERABILMENTE COMPLETI E
AGGIORNATI DEL CORSO DI
APPRENDIMENTO AUTOMATICO**

Basati sulle lezioni svolte in aula

FABIO LOCHE - RICCARDO POLLO



Corso di Apprendimento Automatico
Rosa Meo - Roberto Esposito
Anno Accademico 2017/2018

Corso di Laurea Magistrale in Informatica
Scuola di Scienze della Natura
Università degli Studi di Torino

Fabio Loche, Riccardo Pollo: *Appunti sperabilmente completi e aggiornati del corso di Apprendimento Automatico*, basati sulle lezioni svolte in aula nell'anno accademico 2017/2018, © marzo 2018

Quest'opera è distribuita con Licenza Creative Commons Attribuzione - Non commerciale - Condividi allo stesso modo 4.0 Internazionale.



*"Si dice che un programma apprende dall'esperienza E
con riferimento a alcune classi di compiti T
e con misurazione della performance P,
se le sue performance nel compito T,
come misurato da P,
migliorano con l'esperienza E."*

— "Machine Learning", Thomas M. Mitchell

SOMMARIO

Gli autori di questi appunti sono Fabio Loche e Riccardo Pollo, studenti presso il Dipartimento di Informatica di Torino, Corso di Laurea Magistrale in Informatica, curricula "Sistemi per il Trattamento dell'Informazione" (attuale "Intelligenza Artificiale e Sistemi Informatici Pietro Torasso").

Questi appunti sono basati sulle lezioni svoltesi in aula del corso di Apprendimento Automatico tenute dai professori Roberto Esposito e Rosa Meo nell'anno accademico 2017/2018.

Ci teniamo a ricordare che si tratta di appunti, pertanto vanno considerati tali con tutti i limiti dovuti al tempo a disposizione per scriverli e alla nostra preparazione. Raccomandiamo pertanto di leggerli con occhio critico e non dare per scontata la correttezza di definizioni e formule, potrebbero esserci errori e ce ne scusiamo.

Consigliamo caldamente di utilizzare come riferimenti i libri di testo che trattano adeguatamente gli argomenti affrontati in questi appunti. In particolare il libro di testo consigliato per tale corso è *Machine Learning*, Peter Flach, Cambridge University Press. Da questo libro abbiamo ripreso la divisione in capitoli degli argomenti.

RINGRAZIAMENTI

Questi appunti sono il frutto di un grande sforzo, vista anche la mole finale del lavoro, il quale è stato possibile anche grazie a tutti quei compagni di corso e di studio con i quali c'è stato un proficuo confronto utile a dirimere dubbi e chiarire i concetti.

In particolare ci teniamo a ringraziare la nostra principale compagnia di studio, Katherine M. May, la quale ci ha considerevolmente aiutato durante la stesura e la rilettura di questi appunti, fornendo un contributo prezioso ed indispensabile.

Ai ringraziamenti si aggiungono anche Stefano Tedeschi, Marco Corona e Paolo Alfano, i quali sono stati estremamente disponibili nel condividere i loro appunti, che si sono rivelati un utile supporto e punto di riferimento durante la stesura dei nostri appunti.

Infine ringraziamo anche i professori Roberto Esposito e Rosa Meo per la loro disponibilità e pazienza a fronte delle numerose richieste di ricevimento e chiarimenti.

Fabio e Riccardo

INDICE

1	INTRODUZIONE	1
1.1	Tasks	1
1.1.1	Task predittivi	2
1.1.2	Il problema dell'overfitting	3
1.1.3	Task descrittivi	3
1.2	Models	4
1.2.1	Modelli geometrici	4
1.2.2	Modelli probabilistici	7
1.2.3	Modelli logici	12
1.3	Feature	13
1.3.1	Scelta delle feature	14
1.3.2	Modifica delle features	16
2	CLASSIFICAZIONE BINARIA E TASK CORRELATI	19
2.1	Classificazione	19
2.1.1	La classificazione binaria	20
2.1.2	Misure della performance	21
2.1.3	I <i>coverage plot</i>	26
2.1.4	I <i>ROC plot</i>	29
2.2	Classificazione con scoring	32
2.2.1	Margini di errore e funzioni di penalizzazione	33
2.3	Ranking	35
2.3.1	Performance di un ranking classifier	37
2.3.2	Esempi di calcolo del <i>ranking error rate</i>	38
2.4	Classificatori basati sulle probabilità	40
2.4.1	<i>Probability estimation tree</i>	41
2.4.2	Errore quadratico	41
2.4.3	Errore quadratico medio	44
2.4.4	La probabilità empirica	46
3	OLTRE LA CLASSIFICAZIONE BINARIA	49
3.1	Classificazione multoclasse	49
3.1.1	One-vs-rest - Non ordinato	50
3.1.2	One-vs-rest - Ordinato	51
3.1.3	One-vs-one - Simmetrico	51
3.1.4	One-vs-one - Asimmetrico	52
3.1.5	Classificazione con le matrici output-code	52
3.1.6	Difficoltà applicative dei diversi approcci	53
3.2	Regressione	53
3.2.1	Il problema dell'overfitting II - Il ritorno	54
3.2.2	Il problema <i>bias-variance</i>	55
3.3	Apprendimento non supervisionato e modelli descrittivi	57
3.3.1	Clustering predittivo vs descrittivo	58
3.3.2	Subgroup discovery	59

3.3.3	Association rule discovery	59
4	APPRENDIMENTO DI CONCETTI	61
4.1	Un primo esempio	61
4.2	Spazio delle ipotesi	62
4.2.1	Le relazioni d'ordine parziale	64
4.2.2	Proprietà dei concetti	65
4.3	Algoritmo LGG	65
4.4	Introduzione di esempi negativi	66
4.5	Paths through the hypothesis space	67
4.6	Dati non congiuntamente separabili	70
4.6.1	Esempio di applicazione dell'algoritmo MG Consistent	70
4.7	Clausole di Horn	72
4.8	ILP - Inductive Logic Programming	75
4.9	Ridurre l'insieme di esempi	75
5	TREE MODELS	77
5.1	Introduzione ai Feature Tree	77
5.2	Decision Tree	82
5.2.1	La purezza di uno split	82
5.2.2	L'entropia: l'ammontare di confusione di una classe	83
5.2.3	L'applicazione degli indici di purezza	86
5.2.4	La procedura di split	87
5.2.5	Rappresentazione del partizionamento dello spazio delle istanze	87
5.2.6	Imporre un ordinamento sulle foglie	89
5.3	Ranking trees	90
5.3.1	Applicazione dell'algoritmo di costruzione di un albero	91
5.4	L'attribuzione della classe	92
5.5	Utilizzi di un feature tree	93
5.6	Potare un albero	93
5.6.1	Algoritmo di potatura di un feature tree	94
5.6.2	Stima delle generalizzazioni degli errori	95
5.7	Impatto della distribuzione delle classi	96
5.7.1	Un esempio sconvolgente	96
5.7.2	L'impurità relativa	97
5.7.3	La ricetta di Peter Flach per il learning di un decision tree	98
5.8	Apprendimento di alberi basato sulla riduzione della varianza	98
5.8.1	Introduzione al problema della regressione	98
5.8.2	Alberi di regressione	99
5.8.3	Introduzione al problema del clustering	100
5.8.4	Alberi di Clustering	101
5.8.5	Osservazioni finali sugli alberi di clustering	104

6 MODELLI A REGOLE	107
6.1 Introduzione	107
6.2 Apprendere liste di regole ordinate	107
6.2.1 L'idea del modello a regole	107
6.2.2 Un esempio introduttivo	108
6.2.3 Algoritmo di generazione delle regole ordinate	111
6.2.4 L'analogia con i feature tree	111
6.2.5 Ranking con liste di regole	111
6.3 Modelli a regole non ordinate	114
6.3.1 Apprendere insiemi di regole	114
6.3.2 Un esempio	114
6.3.3 L'algoritmo di apprendimento di un insieme di regole	115
6.3.4 L'introduzione di una valutazione probabilistica	116
6.3.5 Il ranking tramite un insieme non ordinato di regole	116
6.3.6 La predizione	118
6.4 Apprendimento di regole descrittive	118
6.4.1 Subgroup discovery	118
6.4.2 Un esempio di valutazione dei sottogruppi . . .	120
6.4.3 Algoritmo per apprendere più regole	121
6.4.4 Items set	121
6.4.5 Association rules	126
7 MODELLI LINEARI	129
7.1 Regressione	129
7.1.1 Metodo dei minimi quadrati	132
7.1.2 Overfitting e regressione regolarizzata	133
7.1.3 Classificazione con il metodo dei minimi quadrati	138
7.2 Support Vector Machine	139
7.2.1 I Support Vector	141
7.2.2 Il problema di ottimizzazione	142
7.2.3 Formulazione del problema duale	143
7.2.4 Margini di errore	148
7.3 Kernel	152
7.3.1 La funzione Kernel	153
7.3.2 Kernel importanti	155
7.3.3 Condizioni per le funzioni Kernel	157
8 MODELLI BASATI SULLA DISTANZA	159
8.1 Similarità e distanza	159
8.1.1 Distanza di Minkowski	159
8.1.2 La distanza di Mahalanobis	162
8.2 Vicinanza e esemplari	164
8.2.1 Media e distanza	164
8.2.2 Il classificatore lineare di base	165
8.2.3 Il punto della situazione	166
8.3 Classificazione (e clustering) Nearest-neighbour . . .	167

8.3.1	Classificatore Nearest-neighbour	167
8.3.2	Il dilemma bias-variacie	169
8.3.3	La densità dei vicini: dbscan	169
8.4	Clustering basato su distanza	172
8.4.1	Matrice di scatter	173
8.4.2	Algoritmo K-means	173
8.4.3	Clustering attorno ai mediodi	177
8.4.4	Silhouettes	179
8.5	Clustering gerarchico	180
8.5.1	Le funzioni di collegamento	181
8.5.2	Algoritmo per il cluster gerarchico agglomerativo	183
8.5.3	Il dibattito sul linkaggio	183
8.6	Dai Kernel alle distanze	184
8.6.1	La kernelizzazione	185
8.6.2	Algoritmo K-Means Kernelizzato	185
8.6.3	Cosine Similarity	185
9	MODELLI PROBABILISTICI	187
9.1	Modelli descrittivi	187
9.2	I modelli generativi	188
9.2.1	Vantaggi e svantaggi di un modello generativo	189
9.2.2	Riduzione dell'incertezza nella stima: distribuzione β	190
9.3	La prospettiva Bayesiana	191
9.4	L'ottimalità di Bayes	192
9.5	Model selection	192
9.6	La distribuzione normale	193
9.6.1	Gli obiettivi	193
9.6.2	Una distribuzione normale multivariata	194
9.7	Likelihood ratio	196
9.8	Conclusioni	199
9.9	Dalla distanza alla probabilità	200
9.10	Stima della maximum-likelihood	200
9.11	Minimizzare lo scarto quadratico per fare regressione lineare	201
9.11.1	Un esempio pratico	202
9.12	Due criteri coincidenti	202
9.13	Modelli probabilistici per dati non numeriche	203
9.13.1	Modello basato su distribuzione di Bernoulli . .	203
9.13.2	Modello basato su distribuzione Multinomiale .	204
9.14	Naive Bayes	205
9.14.1	Il modello Naive Bayes formalmente	205
9.14.2	Un esempio di applicazione	206
9.14.3	Un esempio multinomiale	207
9.14.4	Un terzo esempio con probabilità ricalibrate .	208
9.14.5	La stima della densità	209
9.15	Logistic Regression	211

9.15.1 I modelli lineari generalizzati	213
9.15.2 Il modello di regressione logistica applicata a feature categoriche	213
9.15.3 Un piccolo cambio di notazione	214
9.15.4 Ritorno alla likelihood	215
9.15.5 La regressione logistica a confronto	216
9.16 Algoritmo EM	217
9.16.1 Obiettivi	217
9.16.2 Introduzione all'algoritmo	219
9.16.3 L'algoritmo	220
9.17 Modelli basati su compressione	222
9.17.1 Esempio di applicazione del principio	222
9.17.2 Il principio della "minimum descriptpion lenght"	223
10 FEATURES	225
10.1 Diversi tipi di feature	225
10.2 Trasformazione delle feature	225
11 APPRENDIMENTO ENSEMBLE	227
11.1 Funzionamento dell'apprendimento ensemble	227
11.1.1 Come combinare diversi modelli	228
11.1.2 Come generare diversi modelli di base	229
11.2 Bootstrap Aggregation	229
11.2.1 Esempio applicativo del Bagging	230
11.2.2 Random forest	231
11.3 Adaptive Boosting	233
11.3.1 L'intuizione di AdaBoost	233
11.3.2 L'algoritmo AdaBoost	235
11.3.3 Correttezza di AdaBoost	236
11.3.4 Come fa a funzionare 'sta roba	242
12 LA MISURAZIONE DEGLI ESPERIMENTI	245
12.1 Accuratezza attesa	245
12.2 Come effettuare le misurazioni	248
12.3 Esempi di cross validation	250
12.4 L'interpretazione delle misurazioni	250
12.5 Esempio di valutazione dell'intervallo di confidenza .	252
12.6 Conclusioni	252
12.7 Le ipotesi nulle e il p-value	253
12.8 Paired t-test	253
12.8.1 Esempio di applicazione del Paired t-test	254
12.9 Wilcoxon's signed-rank test	254
12.9.1 Esempio di applicazione del Wilcoxon's signed- rank test	255
12.9.2 Confrontare diversi algoritmi su più dataset .	255
12.9.3 Esempio di applicazione del Friedman's test .	256
12.10 Post-hoc tests	257
12.10.1 Bonferroni-Dunn test	257
12.10.2 Rappresentazione grafica del post-hoc test .	257

ELENCO DELLE FIGURE

Figura 1.1	Un esempio grafico di classificatore lineare.	6
Figura 1.2	Schema riassuntivo del funzionamento del machine learning per task di tipo predittivo.	6
Figura 1.3	Schema del funzionamento del machine learning nello specifico esempio del filtro anti-spam SpamAssassin per ottenere un modello geometrico.	7
Figura 1.4	Schema del funzionamento del machine learning nello specifico esempio del filtro anti-spam SpamAssassin per ottenere un modello probabilistico.	12
Figura 1.5	Schema del funzionamento del machine learning nello specifico esempio del filtro anti-spam SpamAssassin per ottenere un modello logico.	13
Figura 1.6	Esempio di approssimazione della funzione $y = \cos \pi x$ nell'intervallo $-1 \leq x \leq 1$ disegnata in blu.	14
Figura 1.7	Esempio di approssimazione della funzione $y = \cos \pi x$ nell'intervallo $-1 \leq x \leq 1$ con un albero di regressione che usa la variabile x per lo split e dei classificatori lineari all'interno delle foglie.	14
Figura 1.8	Grafico sull'incidenza del diabete nelle persone in base al loro peso riportato sull'asse delle ascisse con campionamento ogni 10 chilogrammi.	15
Figura 1.9	Grafico sull'incidenza del diabete nelle persone in base al loro peso riportato sull'asse delle ascisse con campionamento ogni 20 chilogrammi.	16
Figura 1.10	Esempio di mapping dei dati, trasformazione delle feature. Nel grafico a sinistra i punti corrispondono agli esempi originali con coordinate (x, y) ; nel grafico a destra sono riportati gli stessi esempi, ma con coordinate modificate (x^2, y^2)	17
Figura 2.1	Schema riassuntivo degli argomenti collegati alla classificazione binaria e di come sono correlati fra loro.	20
Figura 2.2	Esempio di albero di decisione (a destra) che svolge la classificazione binaria di e-mail per distinguere quelle contenenti spam e del corrispettivo albero delle feature (a sinistra).	21

Figura 2.3	Da sinistra a destra trasposizione grafica dei concetti di training set, diviso in esempi positivi <i>Pos</i> ed esempi negativi <i>Neg</i> ; evidenziazione della porzione di esempi classificati correttamente come positivi (<i>TP</i>); evidenziazione della porzione di esempi classificati correttamente come negativi (<i>TN</i>).	23
Figura 2.4	Trasposizione grafica dei concetti di falsi positivi (<i>FP</i>) e falsi negativi (<i>FN</i>), messi in evidenza rispettivamente nel disegno a sinistra e in quello a destra.	23
Figura 2.5	Trasposizione grafica del concetto di accuratezza, identificata dalla regione ombreggiata, come indicato dalla legenda.	25
Figura 2.6	Trasposizione grafica dei concetti di tasso di <i>TP</i> , a sinistra, e tasso di <i>TN</i> , a destra.	25
Figura 2.7	Esempio di coverage plot con due diversi classificatori, <i>c1</i> e <i>c2</i> , messi a confronto.	27
Figura 2.8	Due coverage plot nei quali sono stati evidenziati nel primo in verde il <i>ROC heaven</i> e nel secondo in rosso il <i>ROC hell</i>	27
Figura 2.9	Esempio di coverage plot con tre diversi classificatori, <i>c1</i> , <i>c2</i> e <i>c3</i> , messi a confronto.	28
Figura 2.10	Esempio di un generico coverage plot dove sono state messe in evidenza due rette. Tutti i classificatori che giacciono su una retta parallela a quella rossa hanno la stessa accuratezza; tutti i classificatori che giacciono su una retta parallela a quella verde hanno la stessa sensibilità (<i>recall</i>) media.	29
Figura 2.11	Esempio di <i>ROC plot</i> ottenuto dalla normalizzazione del coverage plot in Figura 2.10.	30
Figura 2.12	<i>ROC plot</i> di esempio con diversi classificatori identificati dai cerchietti (quelli sui bordi sinistro e superiore sono tagliati a metà).	31
Figura 2.13	Esempio di <i>scoring tree</i> ottenuto dall'albero delle feature della Figura 2.2.	33
Figura 2.14	Esempio di funzione loss per regolare la penalizzazione, sull'asse delle ordinate, correlata al margine di errore di un classificatore, sull'asse delle ascisse.	34
Figura 2.15	Riassunto delle principali funzioni loss per la penalizzazione.	36
Figura 2.16	Esempio di ordinamento con tre classificatori binari <i>h1</i> , <i>h2</i> e <i>h3</i> che lo dividono in diversi punti.	37

Figura 2.17	Albero delle feature per l'esempio della classificazione binaria delle e-mail contenenti spam, con annessi punteggi assegnati dallo scoring tree costruito in precedenza.	39
Figura 2.18	Albero delle feature per l'esempio della classificazione binaria delle e-mail contenenti spam (a sinistra), con corrispondente albero delle stime della probabilità (a destra) nelle cui foglie è riportato il valore compreso fra 0 e 1 che esprime la probabilità che l'esempio appartenga alla classe positiva, in questo caso alla classe spam.	41
Figura 3.1	Esempio di utilizzo di tre classificatori binari per risolvere un problema di classificazione con 3 classi. A sinistra è riportato il dataset iniziale, a destra i tre classificatori utilizzati per la classificazione con schema one-vs-rest non ordinata.	50
Figura 3.2	Diverse funzioni che cercano di approssimare l'andamento dei cinque punti riportati nel grafico. Modelli più complessi sono più precisi, con il rischio dell'overfitting, modelli più semplici sono meno precisi, ma possono generalizzare meglio.	54
Figura 3.3	Schema riassuntivo del funzionamento del machine learning per task di tipo descrittivo, che si differenzia dallo schema già visto in Figura 1.2 per i task di tipo predittivo.	58
Figura 4.1	Grafo raffigurante lo spazio delle ipotesi trattato nella sezione 4.1	62
Figura 4.2	Grafo raffigurante lo spazio delle ipotesi ristretto agli esempi della classe positiva	63
Figura 4.3	Rappresentazione di <code>LGG</code> , <code>lub</code> e <code>glb</code> in un reticolo formato dagli esempi.	64
Figura 4.4	Relazione d'ordine parziale raffigurante l'inclusione di insiemi facenti parti dell'insieme delle parti di $X = \{a, b, c, d\}$	64
Figura 4.5	Introduzione di un esempio della classe negativa	66
Figura 4.6	Version space dell'esempio considerato.	67
Figura 4.7	Esempio di generalizzazione basato sul version set	69
Figura 4.8	Esempio di applicazione dell'Algoritmo 4 . . .	71
Figura 5.1	Rappresentazioni ad albero dell'esempio introdotto nella sezione 4.1	77
Figura 5.2	Passaggio di pruning di un feature tree e relativo decision tree	79

Figura 5.3	Information Plot $0 \leq x \leq 1$	85
Figura 5.4	Entropia $H(X)$ di un generico output binario $X = 1/X = 0$	85
Figura 5.5	Ipotetico decision tree ottenuto dall'applicazione dell'Algoritmo 6	88
Figura 5.6	Esempio di coverage plot per il decision tree riportato in Figura 5.5	89
Figura 5.7	Model tree dell'esempio visto in precedenza . .	90
Figura 5.8	Esempio di rappresentazione della curva di copertura a partire da un albero decisionale	91
Figura 5.9	L'individuazione delle classi può essere particolarmente complessa	92
Figura 5.10	Effetti della potatura di un albero	94
Figura 5.11	Manifestazione grafica di overfitting	94
Figura 5.12	Tabella che riporta un certo numero di rilevazioni su aste di organi musicali	100
Figura 5.13	Esempio di regression tree per l'esempio presentato alla sottosottosezione	101
Figura 5.14	Matrice del dataset con nuove feature	102
Figura 5.15	Matrice di dissimilarità con dati conosciuti a priori per l'esempio	102
Figura 5.16	Albero di clustering risultante basato sulla distanza euclidea delle feature numeriche	103
Figura 6.1	Gli esempi in rosso sono puri per la classe negativa, mentre quello in verde lo è per la classe positiva. Le altre linee sono di colore tendente al verde se $p > 1/2$ e tendenti al rosso se $p < 1/2$ (incontrandosi sull'arancione dove $p = 1/2$)	109
Figura 6.2	Misure di impurità applicate all'esempio presentato	109
Figura 6.3	Selezione della seconda regola per l'esempio. Come si nota, i 4 esempi negativi individuati dalla prima regola Gills = yes non vengono più considerati.	110
Figura 6.4	Terzo step	110
Figura 6.5	Rappresentazione tramite feature tree e rispettivo coverage plot del modello a regole	112
Figura 6.6	Ranking curve prodotto dall'applicazione dei due modelli a regole AB e BA	113
Figura 6.7	Le due curve di copertura a confronto: diversi pesi per gli errori sulle due classi porteranno a scelte diverse	114

Figura 6.8	In figura 6.8a si mostra il risultato dell'applicazione della Laplace correction all'esempio. Il grafico risultante (figura 6.8b) mostra i segmenti a partire dalla nuova origine (-1,-1))	117
Figura 6.9	Confronto tra un ranker prodotto da un modello a regole ordinate (in blu) ed un modello a regole non ordinate (in arancione)	118
Figura 6.10	Confronto tra le valutazioni basate su average recall e precision	120
Figura 6.11	Esempio di ranking prodotti su sottogruppi seguendo sia la misurazione della precision (con Laplace correction) che con average recall	121
Figura 6.12	Esempio di transazioni (identificate dal numero nella prima colonna), ciascuno con un insieme di beni acquistati (seconda colonna)	122
Figura 6.13	Esempio di grafo indicante il supporto per ogni insieme di oggetti	123
Figura 6.14	Rappresentazione con diagramma di Venn delle varie sottocategorie di item set	124
Figura 6.15	Confronto tra grafi ottenuti con e senza tener conto dei soli item set chiusi	128
Figura 7.1	Un esempio di modello lineare, ovvero la retta che meglio approssima tutti i punti sul piano.	130
Figura 7.2	Interpretazione geometrica del prodotto $\hat{X}\hat{w}$ e dell'errore e commesso rispetto all'output atteso y	131
Figura 7.3	Interpretazione geometrica della minimizzazione dell'errore e , che deve quindi essere ortogonale al piano $C(X)$, commesso rispetto all'output atteso y	132
Figura 7.4	Un esempio di modello lineare ottenuto con il metodo dei minimi quadrati. Si nota come il punto di coordinate (0,20), che è evidentemente discostato da tutti gli altri, influenza negativamente la regressione la quale finisce con il generare un modello distorto rispetto alla distribuzione della quasi totalità dei punti.	133
Figura 7.5	Rappresentazione dei tre modelli lineari ottenuti con diverse funzioni di regolarizzazione a confronto sullo stesso dataset di punti.	136
Figura 7.6	Confronto tra le funzioni di regressione regolarizzata lasso (in alto) e ridge (in basso) in termini di minimizzazione della norma e dispersione dei vettori dei pesi ottenuti.	137

Figura 7.7	Ricerca di un classificatore lineare fra i diversi (e infiniti) possibili classificatori lineari per gli esempi positivi e negativi proposti.	139
Figura 7.8	Il classificatore lineare ideale, equidistante dagli esempi più vicini di ambedue le classi. . . .	140
Figura 7.9	Interpretazione matematica del calcolo del margine fra i support vector, gli esempi più vicini alla soluzione.	141
Figura 7.10	Esempio di SVM con un margine di tolleranza per gli errori con un esempio che giace fra la soluzione ed un support vector.	148
Figura 7.11	Interpretazione grafica del ruolo che giocano le slack variable nella classificazione di esempi in base alla loro posizione rispetto al decision boundary.	152
Figura 7.12	Esempio di come si applica l'idea del metodo Kernel per rimappare esempi di uno spazio bidimensionale (a sinistra) in uno spazio tridimensionale (a destra) che permetta di trovare una soluzione lineare con una SVM.	153
Figura 8.1	La distanza di Mahalanobis prende in considerazione diverse correlazioni tra le features. . .	163
Figura 8.2	Rappresentazione dei centroidi e dei medoidi	165
Figura 8.3	Regione di decisione definita da una regola di decisione basata sulla vicinanza dall'esemplare per tre esemplari	166
Figura 8.4	L'ombreggiatura rappresenta il distribuzione di probabilità prevista su cinque classi.	168
Figura 8.5	Esempio di individuazione dei punti di centro, di bordo e di rumore per DBSCAN	170
Figura 8.6	Rappresentazione grafica del punto 3 dell'algoritmo DBScan	170
Figura 8.7	Eps=10, MinPts = 4	171
Figura 8.8	In questo caso varia la desità tra i cluster e il numero di dati è particolarmente elevato	171
Figura 8.9	Andamento della distanza	171
Figura 8.10	Esempio di dataset per la valutazione di SSE al variare di K = numero di cluster	172
Figura 8.11	Dendrogramma raffigurante l'albero gerarchico dei possibili clustering dei modelli. I punti neri e quelli rossi rappresentano rispettivamente l'appartenza a due diverse classi.	181
Figura 8.12	Rappresentazione grafica delle principali linkage function	182

Figura 8.13	Valutazione di una distribuzione generata random a partire da campionamento uniforme di 20 elementi	184
Figura 8.14	Rappresentazione geometrica del prodotto scalare	185
Figura 9.1	La distribuzione β ha la forma $f()$, dove $\beta - 1$ è il numero di fallimenti e $\alpha - 1$ è il numero di successi negli esempi osservati.	190
Figura 9.2	Rappresentazione del Mixture Model nel caso univariato	195
Figura 9.3	Rappresentazione del Mixture Model nel caso bivariato	195
Figura 9.4	Varie possibilità di discretizzazione di legge a feature continue (tratteggiata)	210
Figura 9.5	Probabilità della classe modellata in modo dicotomico	211
Figura 9.6	Applicazione della funzione sigmoide al risultato della regressione lineare.	212
Figura 9.7	213
Figura 9.8	Due esempi di dataset in cui si pone un confronto tra la regressione logistica (in blu), il classificatore lineare di base (in rosso) e il classificatore basato sulla riduzione degli scarti quadratici (in giallo).	217
Figura 9.9	Confronto della regressione lineare con esempi univariati.	218
Figura 9.10	Rappresentazione del susseguirsi delle fasi di Expectation e Maximisation nell'algoritmo EM	220
Figura 9.11	Probabilità dell'evento di comparsa della parola Viagra in una classe Y che può essere spam o ham.	222
Figura 11.1	Schema del funzionamento dell'ensemble learning. In alto si ha il dataset di partenza D che viene successivamente utilizzato per ottenere T (parametro a scelta dell'utente) nuovi dataset (la ripartizione delle istanze può avvenire in modi diversi). Tali nuovi dataset sono l'input dell'algoritmo di apprendimento prescelto \mathcal{A} , dal quale si ottengono lo stesso numero T di modelli che vengono combinati per ottenere un unico modello finale M	228
Figura 11.2	Dataset di esempio per l'algoritmo di Bagging con due classi identificate dai colori blu e rosso.	231

Figura 11.3	In rosso è evidenziata la superficie corrispondente al risultato ottenuto dall'applicazione di un albero di decisione Classification And Regression Tree (CART) per la risoluzione del problema di classificazione introdotto con il dataset in Figura 11.2. In nero l'ellisse corrispondente alla classificazione corretta.	232
Figura 11.4	Approssimazione grafica della media di 100 alberi di decisione CART utilizzati per classificare gli esempi rossi e blu divisi dall'ellisse nera. Il colore bianco evidenzia le zone in cui il modello è più incerto nella classificazione.	232
Figura 11.5	Grafico che mostra la diminuzione esponenziale dell'errore commesso in relazione al numero di volte T che si utilizza l'algoritmo di apprendimento debole A.	234
Figura 11.6	Tre grafici rappresentanti la distribuzione binomiale dei modelli rispetto all'errore di classificazione che commettono. Sull'asse delle ordinate vi è la probabilità di commettere un errore, sull'asse delle ascisse il numero dei classificatori (nei tre grafici, da sinistra a destra, rispettivamente 10, 20 e 30). All'aumentare del numero T di modelli utilizzati, diminuisce la probabilità che l'insieme di tutti questi modelli commetta un errore (area evidenziata in rosso).	243
Figura 11.7	A sinistra si possono distinguere tre diversi modelli che cercano di classificare le istanze divise nelle due classi, mentre a destra c'è il modello finale ottenuto mediando i risultati dei tre diversi modelli.	244
Figura 12.1	Esempio di Tabella di Contigenza	245
Figura 12.2	Example of ranking	247
Figura 12.3	Esempi di dati estratti dall'applicazione di tre algoritmi di apprendimento	250
Figura 12.4	Esempio di applicazione del t-paired test . . .	254
Figura 12.5	Ranking ottenuto dall'applicazione di un certo algoritmo su 10 dataset	255
Figura 12.6	Esempio di applicazione del Friedman test . .	256
Figura 12.7	Rappresentazione grafica della CD utilizzata graficamente.	258

ELENCO DELLE TABELLE

Tabella 1.1	Classificazione dei task basata sugli obiettivi del modello e sulle modalità di apprendimento.	2
Tabella 1.2	Tabella con valori numerici di esempio per diverse combinazioni di risultati dei test di verifica sulle e-mail.	5
Tabella 1.3	Tabella riassuntiva della distribuzione congiunta per l'esempio del filtro SpamAssassin. Si noti che la somma delle probabilità corrispondenti alla classe spam sia proprio 0.3.	9
Tabella 1.4	Tabella riassuntiva delle probabilità stimate dall'osservazione di tutti i messaggi e-mail ricevuti in precedenza associate alla presenza o meno delle parole riferite dalle due variabili aleatorie nel vettore in input $X = \{X_1 = \text{Viagra}, X_2 = \text{Lottery}\}$	9
Tabella 1.5	Calcolo di Y_{MAP} e classificazione con la formula $P(\text{spam} X_1X_2) \geq 0.5$	10
Tabella 1.6	Calcolo di Y_{MAP} e classificazione con la formula $\arg \max_{Y \in \{\text{spam}, \neg \text{spam}\}} P(X_1X_2 Y)P(Y)$.	10
Tabella 1.7	Calcolo di Y_{ML} e classificazione con la formula $\arg \max_{Y \in \{\text{spam}, \neg \text{spam}\}} P(X_1X_2 Y)$	10
Tabella 1.8	Probabilità che una e-mail contenga o meno spam considerando che la variabile aleatoria X_1 sia indipendente da X_2	11
Tabella 1.9	Probabilità che una e-mail contenga o meno spam considerando che la variabile aleatoria X_2 sia indipendente da X_1	11
Tabella 1.10	Calcolo di Y_{ML} e classificazione con la formula $\arg \max_{Y \in \{\text{spam}, \neg \text{spam}\}} P(X_1X_2 Y)$	12
Tabella 2.1	Tabella di contingenza per l'esempio di classificazione binaria con albero di decisione costruito per e-mail contenenti spam (esempi positivi). In evidenza i numeri di predizioni corrette.	22
Tabella 2.2	Tabella di contingenza con riportati i nomi di alcuni valori particolarmente rilevanti.	22
Tabella 2.3	Elenco di misure delle performance ottenibili partendo da una tabella di contingenza, con rispettive formule per calcolarle.	24
Tabella 2.4	Tabella di contingenza per un secondo classificatore c2.	26

Tabella 3.1	Tabella di contingenza costruita per un esempio con 3 classi con alcune misure delle performance (recall e precisione) per ciascuna classe e la accuratezza finale.	49
Tabella 7.1	Coordinate (x, y) dei punti riportati nell'esempio in Figura 7.1.	130

ACRONYMS

CART Classification And Regression Tree

LGG Least General Generalization

lub Least Upper Bound

glb Greatest Lower Bound

PAC Probably Approximately Correct

RBF Radial Basis Function

ROC Receiver Operating Characteristic

SOM Self-Organizing Map

SVM Support Vector Machine

INTRODUZIONE

L'apprendimento automatico, da qui in poi *machine learning* perché in inglese fa figo, è una disciplina sotto il cappello dell'intelligenza artificiale che studia tecniche e approcci differenti per insegnare ad una macchina l'abilità di apprendere in modo autonomo, migliorare le proprie performance o la propria conoscenza tramite l'esperienza. Tutto ciò senza essere esplicitamente programmata e senza una diretta supervisione dell'uomo.

Il machine learning è la combinazione dei seguenti elementi:

TASK La descrizione degli obiettivi, ovvero quali problemi si intende risolvere (classificazione, regressione, clustering, etc...);

MODEL La rappresentazione matematica delle soluzioni ai problemi sopra citati (classificatori lineari, alberi decisionali, etc...);

FEATURE La rappresentazione degli oggetti coinvolti nei modelli (numeri, categorie, costruzione/selezione di feature, etc...).

In seguito si affronterà nel dettaglio ognuno di questi tre aspetti.

1.1 TASKS

I task possono essere classificati in base agli obiettivi del modello preso in considerazione, in questo modo possono essere divisi in:

TASK PREDITTIVI Il loro obiettivo è quello di prevedere una certa variabile (ad esempio una categoria di appartenenza, una valore numerico, un cluster, etc...);

TASK DESCRITTIVI Il loro obiettivo è quello di individuare una struttura sottostante ai dati.

I task possono anche essere suddivisi in base alla tipologia di apprendimento adottata, nello specifico:

APPRENDIMENTO SUPERVISIONATO Altrimenti noto come *supervised learning*, l'apprendimento viene guidato da esempi correlati dalla loro soluzione (ad esempio per i task di classificazione di immagini apprendono partendo da una serie di immagini alle quali sono già associate le etichette corrette);

APPRENDIMENTO NON SUPERVISIONATO Altrimenti noto come *unsupervised learning*, l'apprendimento viene guidato da esempi

privi di ulteriori informazioni (ad esempio nelle Self-Organizing Map ([SOM](#)) le immagini di esempio sono prive di informazioni sul loro contenuto).

Si possono riassumere nella Tabella 1.1 i diversi tipi di task mettendo insieme i due criteri appena definiti.

	Predictive model	Descriptive model
Supervised learning	Classification, regression	Subgroup discovery
Unsupervised learning	Predictive clustering	Descriptive clustering, association rule discovery

Tabella 1.1: Classificazione dei task basata sugli obiettivi del modello e sulle modalità di apprendimento.

1.1.1 Task predittivi

Come accennato in precedenza lo scopo dei task predittivi è quello di fare inferenza, ovviamente corretta!, su dati non ancora osservati grazie ad un modello predittivo. Tale modello viene ottenuto dalla fase di apprendimento su un insieme di esempi chiamato *training set* e deve essere in grado di generalizzare su dati sconosciuti, che non appartengono al suddetto insieme.

Alcuni dei principali esempi di questa categoria di task vengono riportati di seguito:

CLASSIFICAZIONE BINARIA E MULTICLASSE Lo scopo è classificare i dati in input in due (binary classification) o più (multiclass classification) categorie;

REGRESSIONE L'associazione non viene fatta con un valore appartenente ad un insieme finito, bensì con un valore reale risultante da una funzione (il modello) in grado di rappresentare i punti nello spazio;

CLUSTERING PREDITTIVO L'obiettivo è determinare il cluster di appartenenza di una determinata istanza basandosi su un criterio di vicinanza.

Gli esempi elencati sono sicuramente i più diffusi nell'ambito del machine learning, ma non gli unici. Si possono ancora menzionare i task di stima della probabilità, scoring e ranking, i quali verranno affrontati successivamente.

1.1.2 Il problema dell'overfitting

Il principale problema che affligge in particolare i task predittivi nel machine learning è l'*overfitting* (letteralmente sovradattamento, eccessivo adattamento). In termini pratici si traduce in un modello estremamente performante sul training set, il quale però si rivela inefficiente sul test set, o più in generale su qualsiasi istanza al di fuori di quelle usate in fase di addestramento.

Questo si rivela particolarmente problematico: un modello che soffre di questo problema si rivela inutile perché è inutilizzabile per fare inferenza corretta su dati che non appartengono al training set, cosa per il quale invece sarebbe destinato.

Proprio per valutare le capacità del modello di generalizzare correttamente si usano due insiemi diversi di istanze, sono già stati menzionati:

TRAINING SET L'insieme delle istanze utilizzate per la fase di addestramento;

TEST SET Un insieme di istanze diverse da quelle contenute nel training set utilizzate per la verifica delle prestazioni e la capacità del modello di generalizzare adeguatamente.

Nella valutazione delle performance quindi non si deve puntare solamente a massimizzare quelle sul training set, si rischierebbe di cadere nella trappola dell'overfitting, bensì si deve cercare di minimizzare la distanza fra le performance del training set e le performance del test set. Questo vale a dire che il modello è stato in grado di generalizzare molto bene, riuscendo a classificare efficacemente anche tutte le istanze nuove, mai viste durante l'addestramento. Al contrario quando le performance sul test set si discostano maggiormente si ha un modello eccessivamente adattato al training set.

1.1.3 Task descrittivi

L'obiettivo di questo tipo di task, è esplicitare una struttura sottostante ai dati, ad esempio legami funzionali fra gli attributi delle istanze, distribuzione delle istanze in alcune classi, notevoli generalizzazioni e astrazioni del campione di dati. Verranno trattati in modo esaustivo nella sottosezione [6.4.1](#).

Fra gli esempi principali di questi task ci sono quelli già citati nella Tabella [1.1](#):

SUBGROUP DISCOVERY L'obiettivo è la ricerca di un sottoinsieme di istanze nel dataset con caratteristiche peculiari, come ad esempio una particolare distribuzione delle classi rispetto all'insieme di partenza;

DESCRIPTIVE CLUSTERING L'obiettivo è la ricerca dei possibili cluster con i quali si possono classificare tutte le istanze nel dataset;

ASSOCIATION RULE DISCOVERY L'obiettivo è la ricerca di regole ricorrenti in diversi attributi delle istanze nel dataset (si veda la sottosezione 6.4.5).

Si noti che il clustering può quindi essere declinato in due versioni differenti: nei task predittivi conosciamo i cluster e si devono classificare al loro interno nuove istanze, nei task descrittivi dalle istanze del training set si devono scoprire i cluster in cui dividerle.

1.2 MODELS

Dopo aver discusso approfonditamente i diversi tipi di task che il machine learning può affrontare, si deve dare spazio ad un aspetto ugualmente importante: i modelli, ovvero *come* nella fase di apprendimento vengono mappati i dati in input descritti sotto forma di feature agli output opportuni. I modelli si possono classificare essenzialmente in tre categorie, sicuramente rappresentative ma non esaustive:

MODELLO GEOMETRICI I dati vengono mappati in uno spazio geometrico e la soluzione è descrivibile anch'essa geometricamente (ad esempio una retta che divide un piano);

MODELLO PROBABILISTICO Il dominio viene descritto attraverso elementi probabilistici, associando ai dati una probabilità, la quale permette di fare inferenza probabilistica o calcolare distribuzioni probabilistiche per ridurre l'incertezza¹;

MODELLO LOGICO Sono costruiti con formule logiche grazie alle quali è possibile fare inferenza.

Si prenda ora in considerazione uno dei più famosi esempi nell'ambito del machine learning, il filtro anti-spam SpamAssassin. Con questo esempio verranno presentati i diversi tipi di modelli che si possono realizzare per questo task.

1.2.1 Modelli geometrici

Lo scopo del filtro per le e-mail è identificare nella posta in arrivo quali messaggi sono spam (*junk mail*). SpamAssassin è un filtro anti-spam molto diffuso che, alla ricezione di una e-mail, analizza testo, immagini, mail server, collegamenti, etc... con una serie di test booleani assegnando loro dei pesi che possono essere positivi, negativi o

¹ Ci scusiamo per l'abuso del termine *probabilità* e dei suoi derivati, probabilmente abbiamo esagerato.

E-mail	x_1	x_2	Spam?	$4x_1 + 4x_2$
1	1	1	1	8
2	0	0	0	0
3	1	0	0	4
4	0	1	0	4

Tabella 1.2: Tabella con valori numerici di esempio per diverse combinazioni di risultati dei test di verifica sulle e-mail.

nulli. Se la somma finale di tali pesi supera una determinata soglia (in questo specifico filtro corrisponde a 5) allora la e-mail viene marcata come spam.

Di seguito viene riportata una tabella con degli esempi numerici. Nella prima colonna sono elencate quattro e-mail; x_1 e x_2 sono due test condotti, se il risultato del test è positivo i peso assegnato è pari a 1, 0 altrimenti; la colonna Spam? contiene un'etichetta fornita dall'utente (1 spam, 0 altrimenti) che classifica le mail per addestrare il sistema; l'ultima colonna contiene il possibile modello proposto, $4x_1 + 4x_2$, con il quale si potranno valutare le nuove e-mail in arrivo. L'introduzione di una soglia permette infine di poter classificare le e-mail: se il modello fornisce un valore in output strettamente maggiore di 5 allora si tratterà di spam, come per la prima delle quattro proposte nell'esempio. Si noti come il modello classifichi correttamente le e-mail rispetto alle etichette fornite dall'utente. È altresì chiaro che il modello proposto in questo esempio non è l'unico possibile, ve ne possono infatti essere diversi, potenzialmente infiniti, in grado di fornire un output che ci permetta di svolgere lo stesso task di classificazione in maniera ugualmente corretta.

Riassumendo si possono identificare i tre elementi introdotti in precedenza:

TASK La classificazione delle e-mail in entrata;

MODEL La funzione definita per calcolare il punteggio di ogni e-mail da valutare, un modello lineare;

FEATURES I risultati dei due test, x_1 e x_2 .

1.2.1.1 Classificatori lineari

Come già accennato l'esempio appena affrontato si risolve con un classificatore lineare. Per determinare tale modello si devono trovare dei pesi tali per cui si ottiene una soglia che divide esattamente (o come meglio è possibile) gli esempi positivi da quelli negativi.

Tale concetto è espresso graficamente e molto intuitivamente dalla Figura 1.1, nella quale si può identificare il vettore dei pesi w evi-

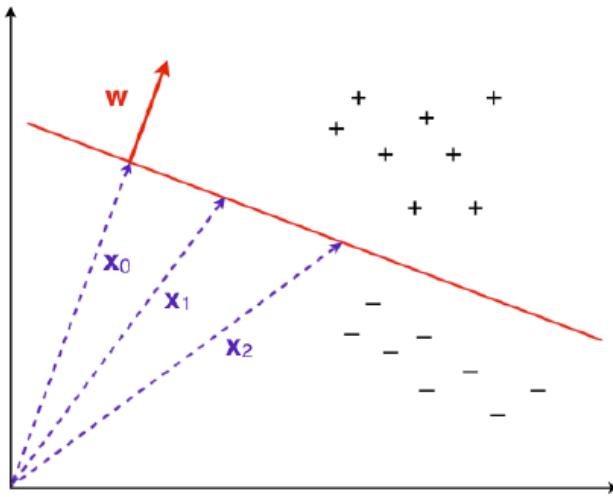


Figura 1.1: Un esempio grafico di classificatore lineare.

denziato in rosso, perpendicolare alla soglia che divide gli esempi altrimenti nota come *decision boundary* o *linear boundary*.

Quanto espresso a parole e graficamente può essere formalizzato come segue:

- $x_i = \{0, 1\}$, risultato binario dell'i-esimo test;
- $w = [w_1, w_2, \dots, w_n]$, il vettore dei pesi;
- $\sum_{i=1}^n w_i \cdot x_i > t$, modello lineare tramite il quale si classifica un input con una soglia di decisione t .

Nella Figura 1.2 si possono identificare le diverse parti che compongono il processo di apprendimento automatico e come sono fra loro relazionate per i task di tipo predittivo.

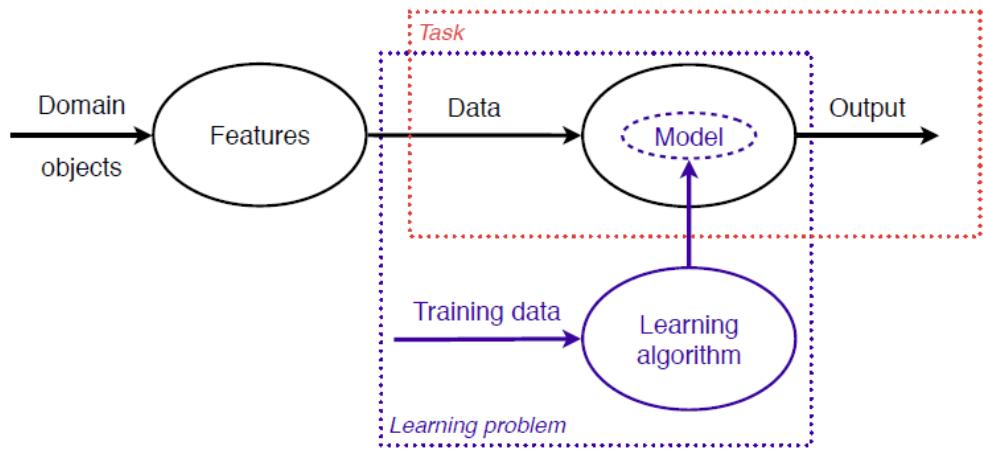


Figura 1.2: Schema riassuntivo del funzionamento del machine learning per task di tipo predittivo.

Il task si definisce come i dati che possono essere osservati e modellati al fine di ottenere l'output desiderato.

Il modello per raggiungere il suddetto task è solitamente parametrizzato (ad esempio con dei pesi, come nell'esempio di SpamAssassin). Tramite i parametri si può migliorare e raffinare il modello affinché sia più preciso e acquisisca maggiore esperienza. Il learning problem è realizzare un algoritmo di apprendimento, il learning algorithm, capace di trovare i giusti parametri per il modello.

I dati coinvolti nel processo di addestramento possono essere grezzi e inadeguati, pertanto è opportuno definire adeguatamente delle feature che diano un input ad hoc per la realizzazione del modello.

Applicando quanto schematizzato in Figura 1.2 all'esempio del filtro anti-spam si ottiene lo schema in Figura 1.3 dove si possono facilmente identificare tutte le parti viste in precedenza, in particolare il modello geometrico, ovvero il classificatore lineare, che si vuole ottenere.

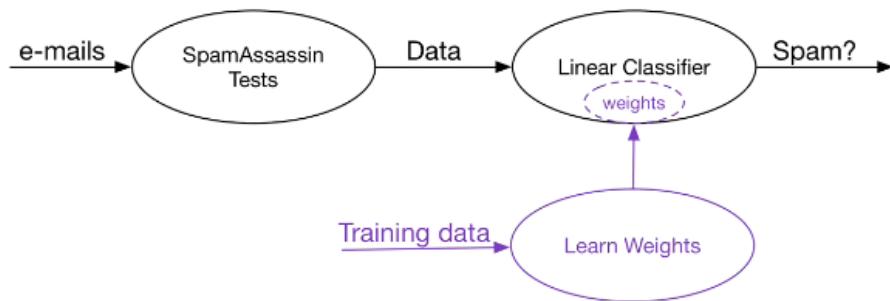


Figura 1.3: Schema del funzionamento del machine learning nello specifico esempio del filtro anti-spam SpamAssassin per ottenere un modello geometrico.

Il learning problem è apprendere un modello per classificare correttamente le e-mail in input. A questo fine si adotta un classificatore lineare, ovvero un modello geometrico, i cui pesi vengono definiti dall'algoritmo di apprendimento grazie al training dataset, ovvero le e-mail inizialmente etichettate dall'utente. I pesi ottenuti dalla fase di addestramento sono un sottoinsieme delle possibili combinazioni accettabili per questo task. Le features sono i test condotti dal filtro anti-spam che forniscono un input adeguato al classificatore.

1.2.2 Modelli probabilistici

Come anticipato in precedenza questa tipologia di modelli si basa sulla teoria della probabilità, attraverso la quale vengono descritti i dati, le variabili e i modelli.

Date due variabili aleatorie:

- X, i dati in input, possono essere un unico valore o un vettore di valori;
- Y, i dati in output.

Ciò che si desidera scoprire è la *maximum a posteriori hypothesis*, che successivamente viene abbreviata con Y_{MAP} , ovvero il valore di Y che massimizza la probabilità a posteriori $P(Y|X)$.

$$\begin{aligned} Y_{MAP} &= \arg \max_Y P(Y|X) \\ &= \arg \max_Y \frac{P(X|Y) \cdot P(Y)}{P(X)} \\ &= \arg \max_Y P(X|Y) \cdot P(Y) \end{aligned} \quad (1.1)$$

Nel primo passaggio viene applicata la regola di Bayes per il calcolo delle probabilità, per cui si sa valere la seguente uguaglianza:

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

Nel secondo passaggio invece si elimina il denominatore. Poiché si intende trovare il massimo valore per la variabile aleatoria Y e il fattore $P(X)$ è indipendente rispetto ad essa, allora lo si può ignorare e considerare solo il numeratore.

Oltre a Y_{MAP} è di interesse il calcolo di Y_{ML} , ovvero il valore di *maximum likelihood*, la funzione di verosimiglianza. A volte infatti si può ignorare la distribuzione di $P(Y)$ (o assumere che sia uniforme). Per questo Y_{MAP} si può ridurre a:

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

Nonostante Y_{ML} sia una approssimazione, all'atto pratico e sotto le dovute condizioni spesso risulta funzionare bene.

1.2.2.1 Modello probabilistico per SpamAssassin

Riprendendo l'esempio del filtro per le e-mail contenenti spam si ha:

- $\mathbf{X} = \{X_1 = \text{Viagra}, X_2 = \text{Lottery}\}$, l'input è un vettore di valori binari che indicano se determinate parole sono presenti o meno nel messaggio, ad esempio $\mathbf{X} = \{X_1 = 0, X_2 = 1\}$ indica che non è presente la parola Viagra associata alla variabile aleatoria X_1 , e che invece è presente la parola Lottery associata ad X_2 ;
- $\mathbf{Y} = \{\text{Ham}, \text{Spam}\}$, l'output è una classificazione delle e-mail fra i due possibili valori Ham o Spam;
- Si assume che un messaggio e-mail sia spam nel 30% dei casi, quindi si ha $P(\text{spam}) = 0.3$;
- Nella Tabella 1.3 è riassunta la distribuzione congiunta per tutti i calcoli probabilistici che seguiranno;
- Nella Tabella 1.4 sono riportate le stime delle probabilità che le e-mail siano spam riferite all'osservazione dei termini Viagra e Lottery nei messaggi già ricevuti.

Viagra	Lottery	Spam	Probabilità
0	0	0	0.6
0	0	1	0.01
0	1	0	0.06
0	1	1	0.02
1	0	0	0.03
1	0	1	0.07
1	1	0	0.01
1	1	1	0.2

Tabella 1.3: Tabella riassuntiva della distribuzione congiunta per l'esempio del filtro SpamAssassin. Si noti che la somma delle probabilità corrispondenti alla classe spam sia proprio 0.3.

X ₁	X ₂	P(X ₁ X ₂ spam)	P(X ₁ X ₂ ¬spam)	P(X ₁ ,X ₂)
0	0	0.033	0.857	0.61
0	1	0.067	0.086	0.08
1	0	0.233	0.043	0.1
1	1	0.667	0.014	0.21

Tabella 1.4: Tabella riassuntiva delle probabilità stimate dall'osservazione di tutti i messaggi e-mail ricevuti in precedenza associate alla presenza o meno delle parole riferite dalle due variabili aleatorie nel vettore in input X = {X₁ = Viagra, X₂ = Lottery}.

1.2.2.2 Calcolo della maximum a posteriori hypothesis

Il modello probabilistico definito per l'esempio del filtro SpamAssassin può essere utilizzato in diversi modi per classificare le e-mail in arrivo.

Nel primo esempio le e-mail vengono classificate stimando la probabilità Y_{MAP} che siano spam attraverso la formula $P(\text{spam}|X_1X_2)$ già introdotta con la precedente Formula 1.1 nella sua versione originale. Se il risultato è maggiore o uguale alla soglia fissata a 0.5 allora il messaggio viene etichettato come spam. Nella Tabella 1.5 vengono riportati i valori calcolati per la *maximum a posteriori hypothesis* e la conseguente classificazione.

Un secondo modo per ottenere lo stesso risultato in termini di classificazione dei messaggi in arrivo è utilizzare la Formula 1.1 per la stima della probabilità. I calcoli relativi a questo approccio sono riportati nella Tabella 1.6, in evidenza sono stati posti i valori massimi selezionati dalla funzione $\arg \max_{Y \in \{\text{spam}, \neg \text{spam}\}}$ che hanno determinato la classificazione.

Nonostante i valori numericamente diversi i due approcci hanno prodotto risultati in termini di classificazione ugualmente corretti,

X_1	X_2	$P(\text{spam} X_1 X_2)$	Y_{MAP}
0	0	0.164	not spam
0	1	0.25	not spam
1	0	0.7	spam
1	1	0.95	spam

Tabella 1.5: Calcolo di Y_{MAP} e classificazione con la formula $P(\text{spam}|X_1 X_2) \geq 0.5$.

X_1	X_2	$P(X_1 X_2 \text{spam})P(\text{spam})$	$P(X_1 X_2 \neg\text{spam})P(\neg\text{spam})$	Y_{MAP}
0	0	$0.033 \cdot 0.3 = 0.0099$	$0.857 \cdot 0.7 = \mathbf{0.5999}$	not spam
0	1	$0.067 \cdot 0.3 = 0.0201$	$0.086 \cdot 0.7 = \mathbf{0.0602}$	not spam
1	0	$0.233 \cdot 0.3 = \mathbf{0.0699}$	$0.043 \cdot 0.7 = 0.0301$	spam
1	1	$0.667 \cdot 0.3 = \mathbf{0.2001}$	$0.014 \cdot 0.7 = 0.0098$	spam

Tabella 1.6: Calcolo di Y_{MAP} e classificazione con la formula $\arg \max_{Y \in \{\text{spam}, \neg\text{spam}\}} P(X_1 X_2|Y)P(Y)$.

d'altra parte tale conclusione è quasi ovvia dato che i due approcci utilizzano la stessa formula, sebbene in due formulazioni diverse.

1.2.2.3 Calcolo della maximum likelihood hypothesis

Dopo aver verificato l'efficacia della *maximum a posteriori hypothesis* nelle sue due diverse formulazioni, è di interesse mettere alla prova anche la funzione di verosimiglianza per vedere come si comporta con l'esempio del filtro SpamAssassin.

Nella Tabella 1.7 sono riportati i valori di Y_{ML} per le diverse combinazioni di parole presenti nelle e-mail. Si nota subito la semplificazione rispetto ai calcoli che invece si erano resi necessari nella Tabella 1.6 dove si teneva conto di $P(Y)$.

X_1	X_2	$P(X_1 X_2 \text{spam})$	$P(X_1 X_2 \neg\text{spam})$	Y_{ML}
0	0	0.033	$\mathbf{0.857}$	not spam
0	1	0.067	$\mathbf{0.086}$	not spam
1	0	$\mathbf{0.233}$	0.043	spam
1	1	$\mathbf{0.667}$	0.014	spam

Tabella 1.7: Calcolo di Y_{ML} e classificazione con la formula $\arg \max_{Y \in \{\text{spam}, \neg\text{spam}\}} P(X_1 X_2|Y)P(Y)$.

È facile vedere come il risultato approssimato con Y_{ML} sia chiaramente corretto come nei primi due casi dove si calcolava la soluzione esatta con Y_{MAP} .

1.2.2.4 Assunzione *naive* di Bayes

Ci si può rendere conto come con un numero più alto di variabili aleatorie (di feature) in input le tabelle di cui sopra diventerebbero enormemente lunghe, infatti le loro righe crescono esponenzialmente rispetto al numero di variabili aleatorie in input. Inoltre per ogni possibile diversa combinazione delle feature bisognerebbe disporre di diversi esempi per poter avere delle accurate stime probabilistiche, un dataset di esempi enorme perché le suddette combinazioni sono anch'esse esponenziali.

Per ovviare a questo gravoso problema computazionale si può fare una assunzione *naive*, grezza, la più semplice che si possa fare. Si assume che le variabili X in input siano indipendenti rispetto ad Y . In questo modo si riduce il problema a tabelle che crescono linearmente invece che esponenzialmente. Molto meglio!

Questa assunzione in realtà viene violata molto spesso, le variabili aleatorie in input possono essere correlate fra loro e non sempre essere tutte indipendenti. Nonostante ciò il risultato dell'applicazione di questa semplificazione si rivela spesso molto efficace, anche se i risultati non sono perfetti.

La conseguenza di questa assunzione *naive* è che a questo punto basta valutare la probabilità correlate alle singole variabili aleatorie in input. Per l'esempio del filtro SpamAssassin si ottengono le Tabelle 1.8 e 1.9, decisamente più semplici rispetto alla Tabella 1.4.

X_1	$P(X_1 \text{spam})$	$P(X_1 \neg\text{spam})$
0	0.1	0.943
1	0.9	0.057

Tabella 1.8: Probabilità che una e-mail contenga o meno spam considerando che la variabile aleatoria X_1 sia indipendente da X_2 .

X_2	$P(X_2 \text{spam})$	$P(X_2 \neg\text{spam})$
0	0.27	0.9
1	0.73	0.1

Tabella 1.9: Probabilità che una e-mail contenga o meno spam considerando che la variabile aleatoria X_2 sia indipendente da X_1 .

Questa semplificazione ovviamente produce a cascata degli effetti sul passaggio successivo, il calcolo della probabilità a posteriori Y_{MAP} che viene rifatto e riportato nella Tabella 1.10.

La formula per il calcolo del valore di Y_{MAP} per questo esempio è:

$$\begin{aligned} Y_{MAP} &= \arg \max_{Y \in \{\text{spam}, \neg\text{spam}\}} P(X_1 X_2 | Y) P(Y) \\ &= \arg \max_{Y \in \{\text{spam}, \neg\text{spam}\}} P(X_1 | Y) P(X_2 | Y) P(Y) \end{aligned}$$

X_1	X_2	$P(X_1 \text{spam}) \cdot P(X_2 \text{spam}) \cdot P(\text{spam})$	$P(X_1 \neg\text{spam}) \cdot P(X_2 \neg\text{spam}) \cdot P(\neg\text{spam})$	Y_{MAP}
0	0	0.0081	0.594	not spam
0	1	0.0219	0.066	not spam
1	0	0.0729	0.036	spam
1	1	0.1971	0.004	spam

Tabella 1.10: Calcolo di Y_{ML} e classificazione con la formula $\arg \max_{Y \in \{\text{spam}, \neg\text{spam}\}} P(X_1 X_2 | Y)$.

Applicando quanto schematizzato in Figura 1.2 all'esempio del filtro anti-spam si ottiene lo schema in Figura 1.4 dove si possono facilmente identificare tutte le parti viste in precedenza, in particolare il modello probabilistico che si vuole ottenere con le relative probabilità stimate.

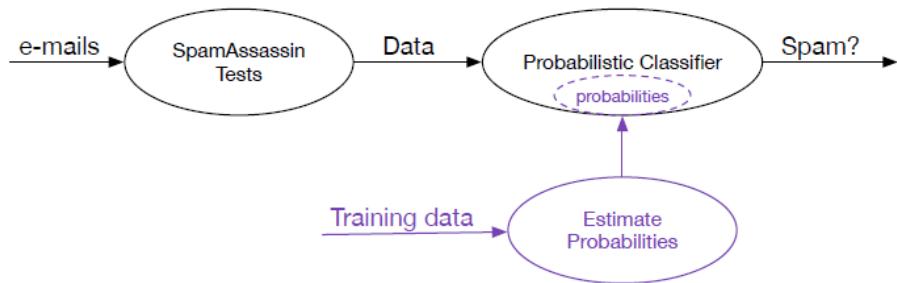


Figura 1.4: Schema del funzionamento del machine learning nello specifico esempio del filtro anti-spam SpamAssassin per ottenere un modello probabilistico.

1.2.3 Modelli logici

Per il task di esempio proposto, il filtro anti-spam SpamAssassin, sono stati proposti un modello geometrico, ovvero un classificatore lineare, ed un modello probabilistico. Di seguito si proporrà un ultimo modello, questa volta di tipo logico, al fine di dimostrare come sovente sia possibile approcciarsi in molteplici modi alla risoluzione di uno stesso problema.

Un classificatore logico cerca di costruire delle regole con le quali poter svolgere inferenza. Nell'esempio di SpamAssassin alcune regole potrebbero essere espresse informalmente come segue:

- "Se la e-mail contiene la parola *Viagra* allora la stima della probabilità che si tratti di spam è 4:1²;

² 4:1 indica che è 4 volte più probabile che la e-mail sia spam piuttosto che non lo sia.

- "Se la e-mail contiene la frase *blue pill* allora la stima della probabilità che si tratti di spam è 3:1";
- "In tutti gli altri casi la stima della probabilità che si tratti di spam è 1:6³".

La prima regola copre tutte le e-mail che contengono la parola "Viagra", indipendentemente dal fatto che contengano anche la frase "blue pill"; la seconda regola riguarda tutte le e-mail contenenti esclusivamente la frase "blue pill", così da non sovrapporsi con la precedente; la terza regola copre tutte le e-mail rimanenti. Come queste regole sono state costruite viene affrontato nei capitoli successivi, ad esempio con gli alberi di decisione.

Applicando quanto schematizzato in Figura 1.2 all'esempio del filtro anti-spam si ottiene lo schema in Figura 1.5 dove si possono facilmente identificare tutte le parti viste in precedenza, in particolare il modello logico che si vuole ottenere con le relative regole decisionali che permettono di fare inferenza.

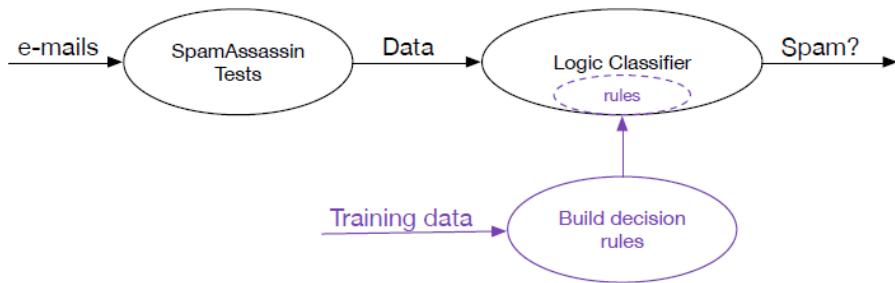


Figura 1.5: Schema del funzionamento del machine learning nello specifico esempio del filtro anti-spam SpamAssassin per ottenere un modello logico.

1.3 FEATURE

La scelta e la manipolazione delle feature in relazione al task si rivela un passaggio molto importante nelle tecniche di machine learning, è un fattore decisivo che permette al modello di essere efficace nell'apprendere e successivamente essere performante. In particolare bisogna capire quali feature vanno tenute in considerazione in base al problema che si vuole risolvere, soprattutto perché non sempre tutti i dati di cui si dispone sono utili per ogni task, anche solo una parte può essere sufficiente, una parte che non necessariamente deve comprendere i dati che apparentemente possono essere considerati importanti e che in realtà sono irrilevanti per alcuni problemi.

Con l'ausilio di tre diversi esempi si approfondirà come la scelta delle feature e la loro manipolazione si rivela assai rilevante nei processi di apprendimento automatico.

³ 1:6 indica che è 6 volte più probabile che la e-mail non sia spam piuttosto che lo sia.

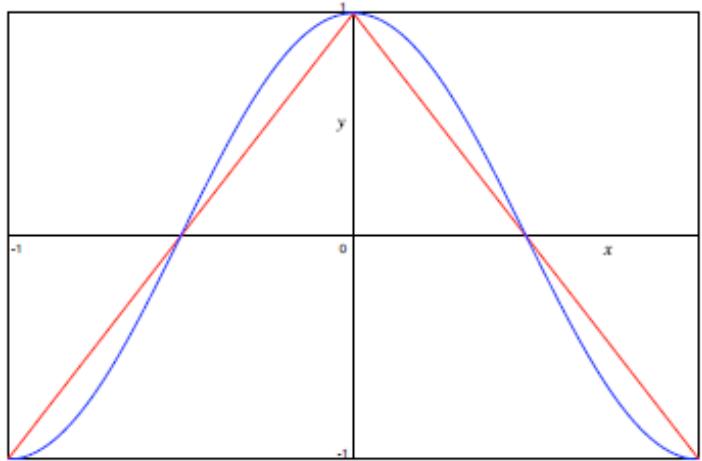


Figura 1.6: Esempio di approssimazione della funzione $y = \cos \pi x$ nell'intervallo $-1 \leq x \leq 1$ disegnata in blu.

1.3.1 Scelta delle feature

Il primo esempio che viene introdotto per spiegare come la scelta delle feature si riveli fondamentale per la risoluzione di problemi di apprendimento automatico è l'approssimazione di una funzione in un determinato intervallo.

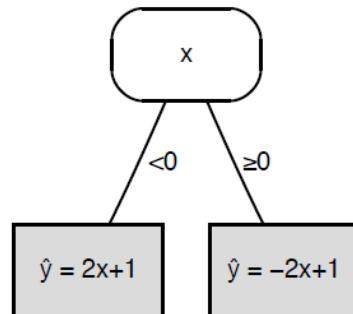


Figura 1.7: Esempio di approssimazione della funzione $y = \cos \pi x$ nell'intervallo $-1 \leq x \leq 1$ con un albero di regressione che usa la variabile x per lo split e dei classificatori lineari all'interno delle foglie.

La funzione scelta è $y = \cos \pi x$ nell'intervallo $-1 \leq x \leq 1$, vedasi Figura 1.6. L'approssimazione più grezza è banalmente la retta di equazione $y = 0$, la quale ovviamente non approssima per niente la curva nell'intervallo desiderato. Tuttavia se si sceglie di spezzare l'intervallo in due intervalli uguali $-1 \leq x < 0$ e $0 \leq x \leq 1$ si possono trovare facilmente due funzioni lineari che approssimano molto meglio la curva. Per fare ciò quindi la variabile x diventa la scelta migliore sia come feature di split, sia come feature di regressione, nella soluzione che unisce due modelli riportata in Figura 1.7.

Un secondo esempio che mette in risalto quanto sia importante una scelta adeguata delle feature che si prendono in considerazione

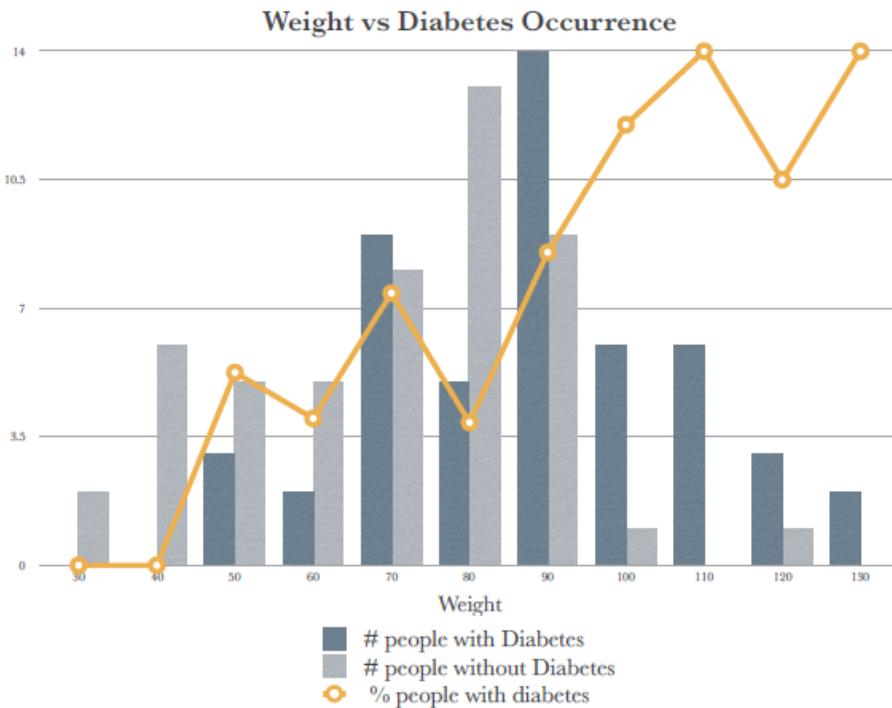


Figura 1.8: Grafico sull'incidenza del diabete nelle persone in base al loro peso riportato sull'asse delle ascisse con campionamento ogni 10 chilogrammi.

nel machine learning è basato su dei dati legati alla malattia del diabete. In particolare nel grafico riportato in Figura 1.8 si combina un istogramma le cui barre verticali rappresentano il numero di persone con o senza il diabete, come specificato nella legenda, mentre in verde viene evidenziata la percentuale di persone con il diabete; sull'asse delle ascisse viene specificato il peso delle persone in chilogrammi.

Il task in questo esempio è predire il numero di persone con o senza diabete in base ai dati a disposizione. Un possibile modo per svolgere questa predizione è valutare la percentuale come nel grafico in Figura 1.8 ed utilizzare la curva, o meglio la linea verde che si ottiene. Purtroppo in questo caso detta linea non è per niente regolare, ha alti e bassi, sale e scende, è rumorosa, e di conseguenza non affidabile per fare delle predizioni, al punto che si potrebbe anche mettere in dubbio la correlazione fra peso e incidenza del diabete nelle persone. Tuttavia questa correlazione c'è, ed è possibile farla risaltare dal grafico con una attenta scelta su una feature. In particolare se la linea ottenuta è troppo irregolare, proprio come in questo caso, il motivo può essere dato da un eccessivo campionamento, un livello di dettaglio troppo alto per le necessità del problema da risolvere.

Nella Figura 1.8 il campionamento dei dati sul diabete veniva fatto dividendo le persone in fasce di 10 chilogrammi, ma se si modifica leggermente questa feature raggruppando le persone in fasce di 20 chilogrammi allora si ottiene il risultato in Figura 1.9, ottenendo

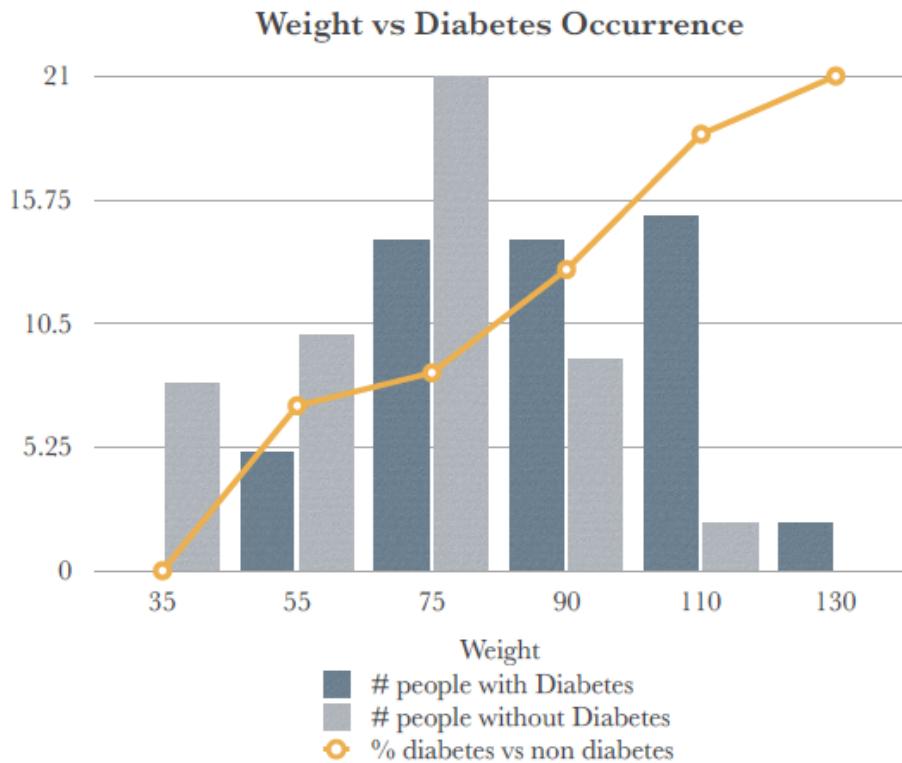


Figura 1.9: Grafico sull'incidenza del diabete nelle persone in base al loro peso riportato sull'asse delle ascisse con campionamento ogni 20 chilogrammi.

quindi una linea verde molto più regolare ed adatta allo scopo di fare predizioni precise ed accurate sull'incidenza della malattia.

Grazie a questo secondo esempio quindi è chiaro che la scelta delle feature è essenziale per permettere agli algoritmi di apprendimento automatico di risolvere in modo intelligente e performante i task. Se per il diabete avessimo scelto il numero di dita di una persona al posto del suo peso per stimare con quale probabilità avrebbe potuto essere affetta da diabete, chiaramente sarebbe stato impossibile venire a capo del problema con una stima sensata.

In particolare nella scelta delle feature si deve tenere ulteriormente conto della loro rappresentazione, argomento che verrà ampiamente discusso nel Capitolo 10, che nell'esempio del diabete corrispondeva alla granularità scelta nella suddivisione dei dati.

1.3.2 Modifica delle features

Con un ultimo esempio introduciamo la modifica delle feature, ovvero come è possibile trasformarle per adattarle meglio agli scopi dei task di apprendimento automatico. Anche questa parte viene trattata più in dettaglio nel Capitolo 10. L'esempio che viene preso in considerazione è una introduzione all'argomento trattato nella Sezione

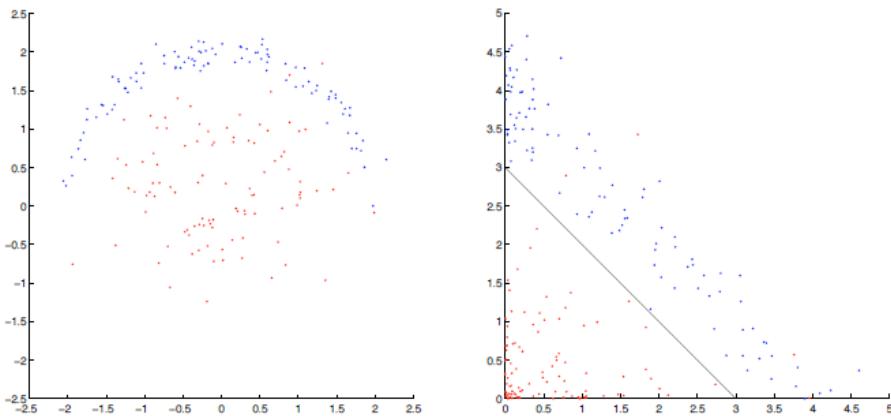


Figura 1.10: Esempio di mapping dei dati, trasformazione delle feature. Nel grafico a sinistra i punti corrispondono agli esempi originali con coordinate (x, y) ; nel grafico a destra sono riportati gli stessi esempi, ma con coordinate modificate (x^2, y^2) .

7.3.

Il concetto di modifica delle feature è molto ampio, può essere declinato in molteplici modi. In questo esempio si vedrà come un mapping delle feature in un nuovo spazio, quindi sostanzialmente una loro modifica sistematica, permetta di ricondurre il problema ad uno di più facile risoluzione.

Si prenda in considerazione il primo dei due grafici riportati in Figura 1.10. Come è facile notare i due insieme di punti, rossi e blu, non sono linearmente separabili, per poter svolgere la classificazione bisognerebbe trovare l'equazione di una funzione di secondo grado, almeno. Un problema non banale, tuttavia facilmente risolvibile con una semplice operazione, una modifica delle feature che consiste nell'elevamento al quadrato di tutti i valori delle coordinate. Il risultato di questa operazione, riportato nel secondo grafico della medesima Figura di cui sopra, mostra come questa trasformazione renda le due classi linearmente separabili. A questo punto è molto più facile trovare un classificatore lineare che divida gli esempi in maniera efficace. Si tenga inoltre presente che questa modifica delle feature non altera il problema, che rimane esattamente lo stesso.

2

CLASSIFICAZIONE BINARIA E TASK CORRELATI

Nel capitolo precedente sono stati introdotti i diversi tipi di task del machine learning, in particolare sono stati divisi in base al tipo di compito da svolgere, descrittivi o predittivi, e al tipo di apprendimento utilizzato, supervisionato o meno.

Nei task di tipo predittivo, dato un training set con esempi etichettati (apprendimento supervisionato), il compito dell'algoritmo di apprendimento è costruire un modello capace di fare previsioni corrette circa alcune proprietà di nuovi esempi mai sottoposti prima all'algoritmo.

In base al tipo di output del modello ottenuto dall'algoritmo di predizione si possono distinguere i seguenti task:

- classificazione;
- scoring e ranking;
- stima della probabilità;
- regressione.

Ovviamente questo breve elenco non è esaustivo, sono state riportate solo le principali categorie di task. In questo capitolo verranno affrontati più nel dettaglio i task appena elencati.

2.1 CLASSIFICAZIONE

Si definisce un classificatore \hat{c}^1 come un mapping da uno spazio di input \mathcal{X} , il quale definisce come i dati sono descritti, e un insieme finito di etichette $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$.

$$\hat{c} : \mathcal{X} \rightarrow \mathcal{C}$$

Poiché si parla di apprendimento supervisionato, si parla anche degli esempi. Questi si definiscono formalmente come una coppia:

$$(x, c(x)) \in \mathcal{X} \times \mathcal{C}$$

Con $x \in \mathcal{X}$ e $c(x)^2$ la corretta classe di appartenenza di x .

Il task dell'algoritmo di apprendimento in questo caso è costruire la funzione \hat{c} che approssimi al meglio possibile la vera funzione c , non

¹ Il cappelletto sulla lettera è una notazione che indica che il classificatore è solo una approssimazione della vera (e sconosciuta) funzione c .

² Si noti che l'assenza del cappelletto indica che si tratta della vera funzione di classificazione, qualsiasi essa sia, che fornisce le etichette corrette per tutti gli esempi.

solo nella classificazione degli esempi appartenenti al training set, ma dell'intero spazio delle istante \mathcal{X} . Questo ultimo aspetto non è un dettaglio, è anzi l'aspetto principale: lo scopo dell'apprendimento è proprio quello di trovare un modello in grado di fare previsioni su dati che non gli sono mai stati sottoposti prima. Deve generalizzare.

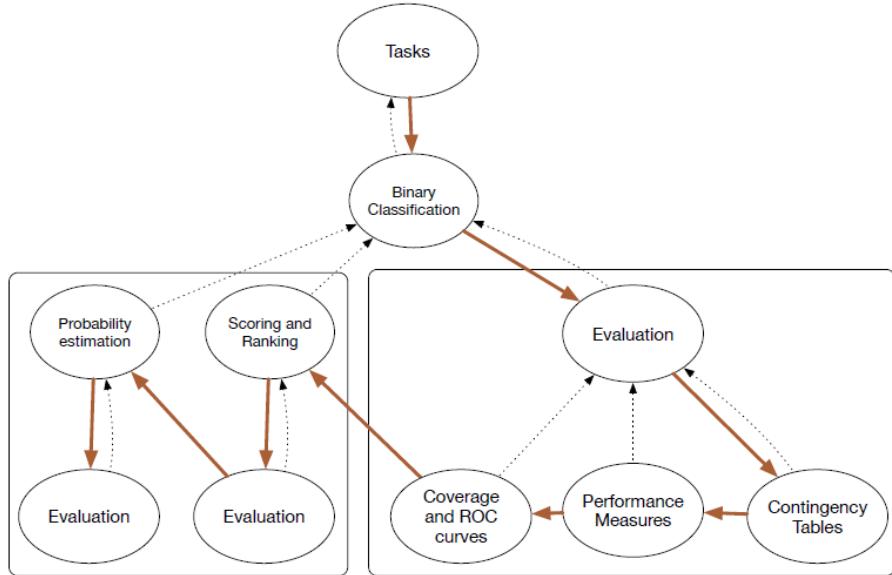


Figura 2.1: Schema riassuntivo degli argomenti collegati alla classificazione binaria e di come sono correlati fra loro.

Si faccia riferimento alla Figura 2.1 per avere una più chiara idea degli argomenti che si vanno a trattare e di come sono fra di loro correlati. In particolare si sta parlando di task, nello specifico si sta per introdurre la classificazione binaria. Questa comprenderà come si valutano le sue prestazioni, ovvero le tabelle di contingenza, le possibili misure delle performance basate su tali tabelle, la copertura (a.k.a. *coverage*) e le curve Receiver Operating Characteristic (*ROC*) (a.k.a. *ROC curves*). Dopo si tratteranno alcuni diversi usi che si possono fare dei classificatori binari, scoring e ranking e stima delle probabilità, e dei rispettivi metodi di valutazione delle performance.

2.1.1 La classificazione binaria

L'esempio più semplice che si può fare è la classificazione binaria, ovvero con due sole classi: $\mathcal{C} = \{C_1, C_2\}$. Assumendo di avere già a disposizione un classificatore binario, l'aspetto di rilievo è: come si valutano le sue prestazioni.

Si riprenda l'esempio delle e-mail di spam utilizzato nel Capitolo 1. Il modello che si utilizza per categorizzare le e-mail in arrivo è un albero di decisione come quello riportato a destra nella Figura 2.2. In tale albero i nodi sono i punti di decisione poiché contengono le feature sulle quali vengono effettuati i test binari, ovvero parole che si

verifica essere presenti o meno nel testo della e-mail, in ordine partendo dal nodo radice. Nelle foglie invece sono riportati i numeri degli esempi che soddisfano i test, divisi in base al loro contenuto (spam oppure ham). Se si prende ad esempio in considerazione la foglia in basso a sinistra, si sa che di tutte le e-mail che non contengono nessuna delle due parole 20 sono spam e 40 non lo sono.

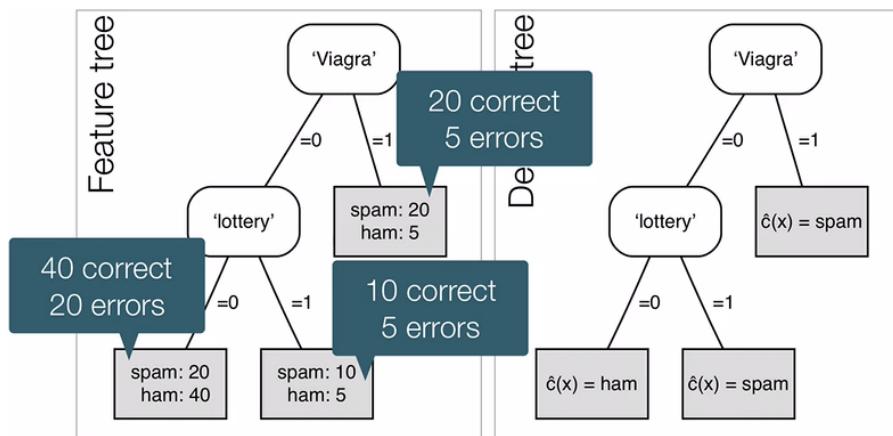


Figura 2.2: Esempio di albero di decisione (a destra) che svolge la classificazione binaria di e-mail per distinguere quelle contenenti spam e del corrispettivo albero delle feature (a sinistra).

Costruito quindi un albero delle feature bisogna decidere quale sia il migliore modo di etichettare le foglie con le due classi a disposizione. In questo modo si ottiene l'albero di decisione. La scelta delle etichette da assegnare alle foglie è guidata dalla classe maggiormente rappresentata nella foglia stessa.

2.1.2 Misure della performance

Il modo più naturale di misurare l'accuratezza di questo modello è rapportare il numero di esempi classificati correttamente con il numero totale degli esempi del training set. In base ai valori riportati in Figura 2.2 si può quindi calcolare la seguente accuratezza:

$$\text{acc} = \frac{40 + 10 + 20}{100} = 70\%$$

A volte, tuttavia, si può desiderare una misurazione delle performance migliore, più precisa ed accurata, l'accuratezza appena stimata quindi risulta essere troppo grossolana. Per ovviare a ciò si introducono le tabelle di contingenza, note agli angosassoni come *contingency tables*, nelle quali le righe riportano i valori attuali ed effettivi degli esempi, mentre le colonne le etichette predette. Per l'esempio che si sta portando avanti è stata costruita la Tabella 2.1.

La lettura di questa tabella è molto semplice. Si prenda la prima colonna: questa contiene il numero di esempi che sono stati predetti positivi dal modello che svolge la classificazione, divisi in base a

	Predetti +	Predetti -	
Attuali +	30	20	50
Attuali -	10	40	50
	40	60	100

Tabella 2.1: Tabella di contingenza per l'esempio di classificazione binaria con albero di decisione costruito per e-mail contenenti spam (esempi positivi). In evidenza i numeri di predizioni corrette.

quanti di questi sono effettivamente tali (prima riga, attuali positivi) e quanti invece sono in realtà esempi negativi (seconda riga, attuali negativi); nell'ultima riga è riportata la loro somma. Un discorso analogo si può fare se si vuole leggere la tabella per righe invertendo semplicemente i concetti, sull'ultima colonna la somma dei valori sulla riga fornisce il totale degli esempi che sono effettivamente positivi (o negativi, nella seconda riga). La somma dei valori nell'ultima riga deve corrispondere alla somma dei valori nell'ultima colonna ed è riportata in basso a destra. In questo esempio il numero totale di esempi utilizzati è 100. Se si esclude questo numero, la somma dei valori evidenziati nella diagonale della tabella di contingenza corrisponde alla misura dell'accuratezza utilizzata poc'anzi.

	Predetti +	Predetti -	
Attuali +	TP	FN	Pos
Attuali -	FP	TN	Neg

Tabella 2.2: Tabella di contingenza con riportati i nomi di alcuni valori particolarmente rilevanti.

La Tabella 2.2 è la stessa tabella di contingenza appena vista, ma vi sono riportati i nomi di alcuni valori in essa contenuti, in particolare:

TP True Positives. Sono gli esempi attualmente positivi e predetti come positivi, quindi correttamente.

FP False Positives. Sono gli esempi attualmente negativi ma predetti erroneamente come positivi.

TN True Negatives. Sono gli esempi attualmente negativi e predetti come negativi, quindi correttamente.

FN False Negatives. Sono gli esempi attualmente positivi ma predetti erroneamente come negativi.

POS Positives. Il totale degli esempi attualmente positivi.

NEG Negatives. Il totale degli esempi attualmente negativi.

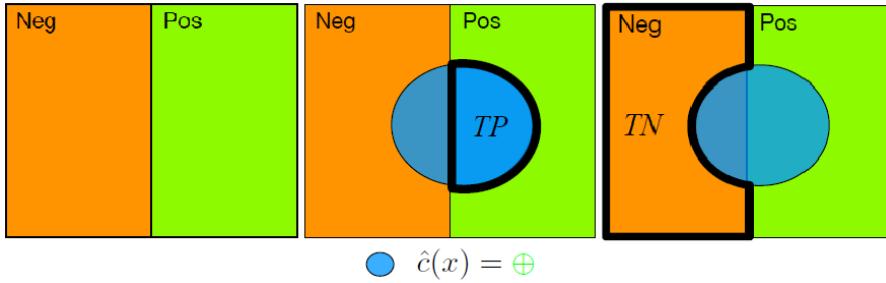


Figura 2.3: Da sinistra a destra trasposizione grafica dei concetti di training set, diviso in esempi positivi *Pos* ed esempi negativi *Neg*; evidenziazione della porzione di esempi classificati correttamente come positivi (*TP*); evidenziazione della porzione di esempi classificati correttamente come negativi (*TN*).

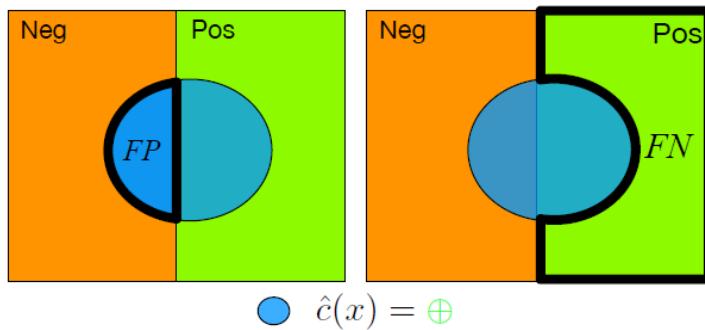


Figura 2.4: Trasposizione grafica dei concetti di falsi positivi (*FP*) e falsi negativi (*FN*), messi in evidenza rispettivamente nel disegno a sinistra e in quello a destra.

A questo punto si dispone di tutte le informazioni necessarie per poter presentare un riassunto di una serie di misure delle performance basate sulle tabelle di contingenza. Si veda la Tabella 2.3, le cui prime sei righe riportano esattamente le stesse misurazioni della tabella di contingenza in modo formale. I valori *pos* e *neg* sono il risultato della normalizzazione di *Pos* e *Neg*.

Nelle Figure 2.3, 2.4, 2.5 e 2.6 si tenga presente che il rettangolo più grande rappresenta l'intero spazio degli esempi, diviso a metà fra positivi e negativi. Il cerchio azzurro, come da legenda, indica gli esempi classificati come positivi.

La Figura 2.3 è una trasposizione grafica delle prime quattro righe della Tabella 2.3: esempi positivi, esempi negativi, esempi predetti correttamente come positivi ed esempi predetti correttamente come negativi. Il cerchio azzurro, come da legenda, identifica gli esempi classificati come positivi dal modello. La Figura 2.4 invece mette in evidenza le regioni indicanti gli esempi classificati erroneamente: a sinistra i falsi positivi, a destra i falsi negativi. La Figura 2.5 invece identifica con una ombreggiatura la regione corrispondente all'accuratezza calcolata in precedenza.

Misura	Formula	Uguale a	Stima
Num. di positivi	$Pos = \sum_{x \in Te} I[c(x) = +]$		
Num. di negativi	$Neg = \sum_{x \in Te} I[c(x) = -]$	$ Te - Pos$	
Num. di TP	$TP = \sum_{x \in Te} I[\hat{c}(x) = c(x) = +]$		
Num. di TN	$TN = \sum_{x \in Te} I[\hat{c}(x) = c(x) = -]$		
Num. di FP	$FP = \sum_{x \in Te} I[\hat{c}(x) = +, c(x) = -]$	$Neg - TN$	
Num. di FN	$FN = \sum_{x \in Te} I[\hat{c}(x) = -, c(x) = +]$	$Pos - TP$	
Positivi in %	$pos = \frac{1}{ Te } \sum_{x \in Te} I[c(x) = +]$	$\frac{Pos}{ Te }$	$P(c(x) = +)$
Negativi in %	$neg = \frac{1}{ Te } \sum_{x \in Te} I[c(x) = -]$	$1 - pos$	$P(c(x) = -)$
Rapporto classi	$clr = \frac{pos}{neg}$	$\frac{Pos}{Neg}$	
Accuratezza	$acc = \frac{1}{ Te } \sum_{x \in Te} I[\hat{c}(x) = c(x)]$		$P(\hat{c}(x) = c(x))$
Tasso di errore	$err = \frac{1}{ Te } \sum_{x \in Te} I[\hat{c}(x) \neq c(x)]$	$1 - acc$	$P(\hat{c}(x) \neq c(x))$
Tasso di TP	$tpr = \frac{\sum_{x \in Te} I[\hat{c}(x) = c(x) = +]}{\sum_{x \in Te} I[c(x) = +]}$	$\frac{TP}{Pos}$	$P(\hat{c}(x) = + c(x) = +)$
Tasso di TN	$tnr = \frac{\sum_{x \in Te} I[\hat{c}(x) = c(x) = -]}{\sum_{x \in Te} I[c(x) = -]}$	$\frac{TN}{Neg}$	$P(\hat{c}(x) = - c(x) = -)$
Tasso di FP	$fpr = \frac{\sum_{x \in Te} I[\hat{c}(x) = +, c(x) = -]}{\sum_{x \in Te} I[c(x) = -]}$	$\frac{FP}{Neg} = 1 - tnr$	$P(\hat{c}(x) = + c(x) = -)$
Tasso di FN	$fnr = \frac{\sum_{x \in Te} I[\hat{c}(x) = -, c(x) = +]}{\sum_{x \in Te} I[c(x) = +]}$	$\frac{FN}{Pos} = 1 - tpr$	$P(\hat{c}(x) = - c(x) = +)$
Precisione	$prec = \frac{\sum_{x \in Te} I[\hat{c}(x) = c(x) = +]}{\sum_{x \in Te} I[c(x) = +]}$	$\frac{TP}{TP + FP}$	$P(c(x) = + \hat{c}(x) = +)$

Tabella 2.3: Elenco di misure delle performance ottenibili partendo da una tabella di contingenza, con rispettive formule per calcolarle.

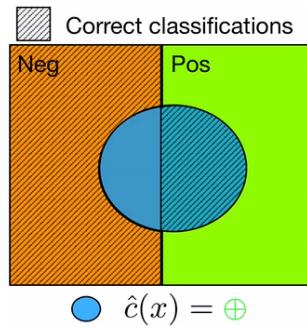


Figura 2.5: Trasposizione grafica del concetto di accuratezza, identificata dalla regione ombreggiata, come indicato dalla legenda.

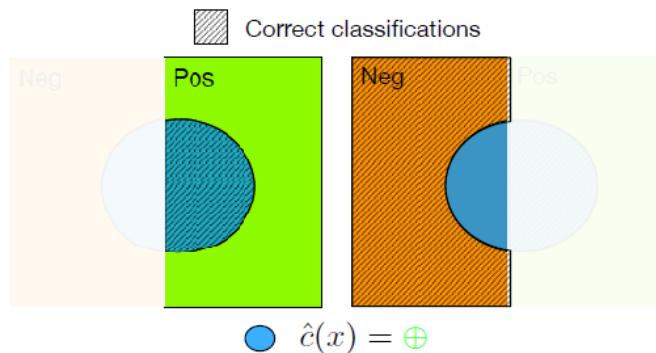


Figura 2.6: Trasposizione grafica dei concetti di tasso di TP , a sinistra, e tasso di TN , a destra.

A questo punto è possibile presentare le misure delle performance alternative alla accuratezza. In particolare si tratta di:

TASSO DI TP Nelle formule indicato come tpr (*True Positives Rate*³), è il rapporto fra il numero di esempi classificati correttamente come positivi (TP) e il numero totale effettivo degli esempi positivi (Pos), vedasi Figura 2.6 a sinistra dove si ricorda che TP corrisponde all'area azzurra ombreggiata e Pos all'intero rettangolo verde;

TASSO DI TN Nelle formule indicato come tnr (*True Negatives Rate*⁴), è il rapporto fra il numero di esempi classificati correttamente come negativi (TN) e il numero totale effettivo degli esempi negativi (Neg), vedasi Figura 2.6 a destra dove si ricorda che TN corrisponde all'area verde ombreggiata e Neg all'intero rettangolo;

PRECISIONE Nelle formule indicato come $prec$ (*Precision*), è il rapporto fra il numero di esempi predetti come positivi correttamente e il numero totale di esempi predetti come positivi, corretti e non.

³ Altrimenti noto come sensibilità o *recall*.

⁴ Altrimenti noto come specificità.

In virtù di quanto detto fino ad ora, se si punta ad avere un tasso di *TP* molto alto si avrà come diretta conseguenza un abbassamento della precisione, questo perché per massimizzare il *tpr* si avrà anche un gran numero di falsi positivi che influiscono negativamente sulla precisione.

2.1.3 *I coverage plot*

Dopo aver visto tutte le diverse misure delle performance che è possibile ottenere dalle informazioni condensate nelle tabelle di contingenza, sarebbe interessante poter vedere graficamente, a colpo d'occhio, come performano diversi classificatori. Per fare questo si introducono i *coverage plot*, i quali possono essere ottenuti sempre dalle tabelle di contingenza.

Assumendo che dato un dataset il numero di esempi positivi e negativi è fissato a priori, quindi non può variare, allora dei quattro principali valori riportati nella 2.2 basta conoscerne due, ad esempio *TP* e *FP*, perché da questi è facilmente possibile ottenere gli altri due.

In particolare si può costruire un grafico come in Figura 2.7, dove sull'asse delle ascisse si hanno valori di *FP* e sull'asse delle ordinate i valori di *TP*. A questo punto si possono indicare con dei punti su tale grafico diversi classificatori, utilizzando come coordinate appunto i suddetti due valori; in particolare c1, la cui tabella di contingenza è già stata riportata in precedenza (Tabella 2.1), e c2 la cui tabella di contingenza viene riportata di seguito (Tabella 2.4).

	Predetti +	Predetti -	
Attuali +	20	30	50
Attuali -	20	30	50
	40	60	100

Tabella 2.4: Tabella di contingenza per un secondo classificatore c2.

Ora si dispone di uno strumento grafico che ci permette di visualizzare in uno spazio bidimensionale come si posizionano diversi classificatori in base al loro numero di *TP* e *FP*. Bisogna però definire quali sono dei buoni classificatori e quali invece no, in quale zona del grafico si può parlare di buone performance, e dove invece no.

Tenendo presente il sistema di coordinate che è stato scelto, si può concludere logicamente che desiderando il più alto numero di esempi⁵ classificati correttamente e il minor numero classificati in modo sbagliato si vorrebbe che il classificatore si trovi il più possibile vicino

⁵ Viene usato il termine esempi in modo generale nonostante siano stati scelti per i coverage plot gli esempi classificati come positivi (corretti e non) perché il discorso è speculare nel caso in cui si dovesse decidere di riprodurre lo stesso grafico con quelli classificati come negativi.

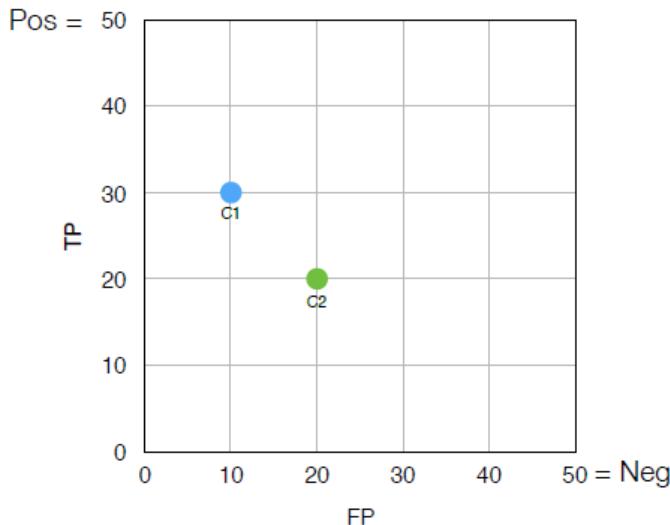


Figura 2.7: Esempio di coverage plot con due diversi classificatori, c1 e c2, messi a confronto.

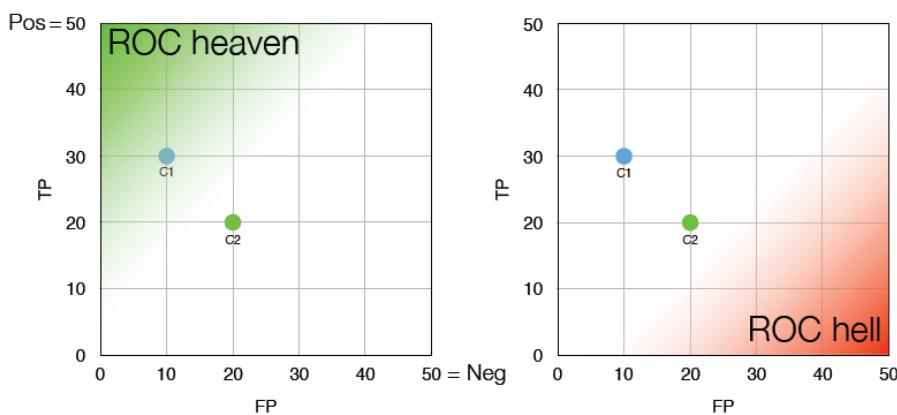


Figura 2.8: Due coverage plot nei quali sono stati evidenziati nel primo in verde il *ROC heaven* e nel secondo in rosso il *ROC hell*.

no all'angolo in alto a sinistra. Se si applicasse un analogo ragionamento per capire dove si trovano i classificatori peggiori si potrebbe concludere che sia l'angolo opposto. Tuttavia è sbagliato, è fuorviante perché un classificatore che si trovi in quella zona classificherebbe tutti gli esempi positivi come negativi e viceversa, pertanto basterebbe semplicemente invertire le risposte per ottenere un modello che predica perfettamente tutte le risposte.

A questo punto si può concludere che la zona peggiore dove possa trovarsi un classificatore su un coverage plot è la zona centrale, intorno alla diagonale che va dall'angolo in basso a sinistra, l'origine, fino all'angolo in alto a destra. In questi casi il numero di predizioni giuste e corrette è simile, pertanto il modello è fra i peggiori, tanto che le sue risposte possono essere considerate casuali come il lancio di una moneta.

Nella Figura 2.8 sono stati messi in evidenza le due zone che sono

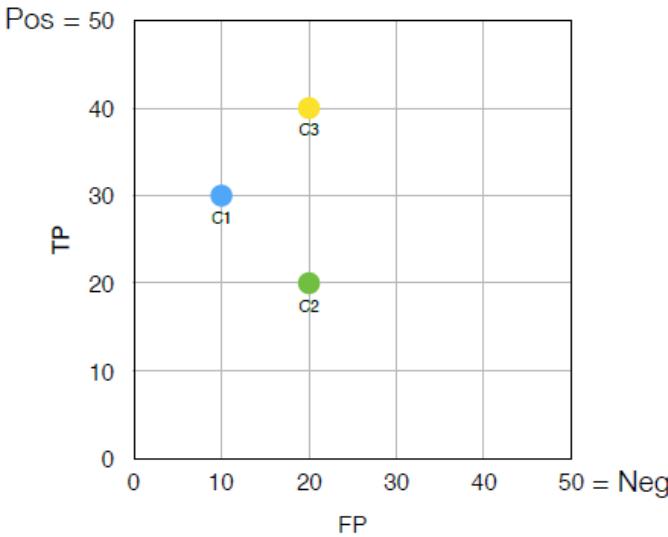


Figura 2.9: Esempio di coverage plot con tre diversi classificatori, c_1 , c_2 e c_3 , messi a confronto.

state identificate prima, alle quali vengono anche assegnati dei nomi. In particolare l’angolo in alto a sinistra e la zona limitrofa si definisce come *ROC heaven* e viene evidenziata con una sfumatura di verde, al contrario l’angolo opposto e la relativa zona limitrofa è il *ROC hell* al quale è stato associato il colore rosso.

Si aggiunga ora al coverage plot utilizzato fino a questo momento come esempio un terzo classificatore c_3 , per il quale non verrà riportata la tabella di contingenza, i cui valori di TP e FP sono rispettivamente 40 e 20, pertanto si ottiene il nuovo coverage plot riportato in Figura 2.9.

Sulla base dei concetti di *ROC heaven* e *ROC hell* introdotti poc’anzi si evince chiaramente come fra c_1 , c_2 e c_3 il peggiore sia il secondo, c_2 . Fra i classificatori c_1 e c_3 però risulta difficile a occhio capire quale sia il migliore, anche perché sono equidistanti dal *ROC heaven*. I classificatori equidistanti dal *ROC heaven* giacciono sulla cosiddetta superficie ottima di Pareto.

Se si prende un generico coverage plot come quello in Figura 2.10 (i coverage plot non devono necessariamente essere quadrati come quelli utilizzati negli esempi precedenti) si possono affermare le due seguenti proprietà:

- tutti i classificatori che giacciono su una stessa retta parallela a quella rossa, la quale ha una pendenza pari a 1, ovvero è perpendicolare alla bisettrice dell’angolo in alto a sinistra, hanno identica accuratezza;
- tutti i classificatori che giacciono su una stessa retta parallela a quella verde, che a sua volta è parallela alla diagonale tratteggiata, hanno identica sensibilità media (*average recall*).

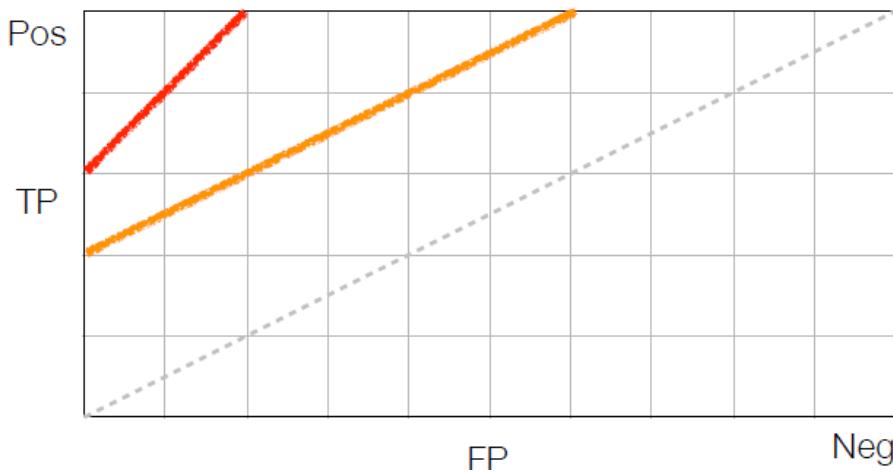


Figura 2.10: Esempio di un generico coverage plot dove sono state messe in evidenza due rette. Tutti i classificatori che giacciono su una retta parallela a quella rossa hanno la stessa accuratezza; tutti i classificatori che giacciono su una retta parallela a quella verde hanno la stessa sensibilità (*recall*) media.

Il valore della sensibilità media (*average recall*) non è presente nella Tabella 2.3, tuttavia dai valori riportati nella stessa è possibile ricavarla con la seguente formula:

$$\begin{aligned} \text{avg recall} &= \frac{tpr + tnr}{2} \\ &= \frac{TP}{Pos} + \frac{TN}{Neg} \\ &= \frac{2}{2} \end{aligned}$$

2.1.4 I *ROC* plot

A questo punto si introduce un nuovo tipo di grafico, legato ai precedenti coverage plot, che si chiama *ROC plot*; l'unica differenza è nella normalizzazione ad 1 dei valori di *TP* e *FP* utilizzati, pertanto ora gli assi delle ascisse e delle ordinate usano come scala rispettivamente i valori *fpr* e *tpr* già introdotti nella Tabella 2.3. Un effetto di questa normalizzazione applicata al coverage plot visto in Figura 2.10 lo si può vedere con il *ROC plot* in Figura 2.11.

Le due proprietà enunciate per il generico coverage plot in Figura 2.10 si adattano ovviamente al *ROC plot* ottenuto dalla normalizzazione ad 1:

- tutti i classificatori che giacciono su una stessa retta parallela a quella rossa, la quale ha una pendenza pari a $clr = \frac{Neg}{Pos}$ e non più 1 come conseguenza della normalizzazione, hanno identica accuratezza;

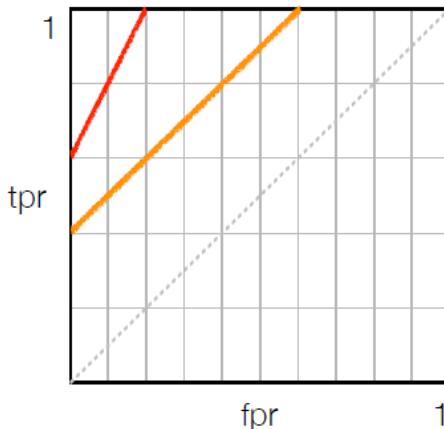


Figura 2.11: Esempio di *ROC plot* ottenuto dalla normalizzazione del coverage plot in Figura 2.10.

- tutti i classificatori che giacciono su una stessa retta parallela a quella verde, la quale ha pendenza pari a 1, hanno identica sensibilità media (*average recall*).

Si prenda ora in considerazione il seguente esempio basato sul *ROC plot* della Figura 2.12. In questo grafico sono presenti diversi classificatori, identificati da dei cerchietti (quelli sui bordi sinistro e superiore sono tagliati a metà), fra questi si desidera trovare il migliore.

Per poter trovare il migliore classificatore fra quelli proposti nell'esempio però si necessita di una informazione fondamentale: il rapporto fra il numero di esempi positivi e il numero di esempi negativi, riportato nella Tabella 2.3 come clr . Nel *ROC plot* di esempio viene indicata con una retta punitinata etichettata appunto come clr , la quale scorrendo lungo la bisettrice blu dall'angolo in alto a sinistra a quello in basso a destra incontra per primo il classificatore con la migliore accuratezza, messo in evidenza con una stellina rossa.

La scelta del rapporto fra il numero di esempi negativi e il numero di esempi positivi ha lo scopo di dare la stessa importanza ad entrambe le classi, indipendentemente dal numero di errori che si compiono per una o per l'altra. Tuttavia questa scelta si adatta bene al dominio dell'esempio preso in considerazione; si tenga presente che il criterio con cui si definisce la performance di un classificatore, e di conseguenza quale sia il modello migliore rispetto ad altri, dipende fortemente dal dominio in cui si opera.

In un dominio ove lo scopo del classificatore è predire la presenza o meno di un tumore si evince come falsi positivi e falsi negativi abbiano una rilevanza, un costo, molto differente. Un falso positivo non crea troppi problemi, perché se il modello indica un'incidenza tumorale che successive analisi possono approfondire e negare con maggiore precisione allora non si hanno conseguenze gravi. Al contrario con un falso positivo si corre il serio rischio di considerare sana

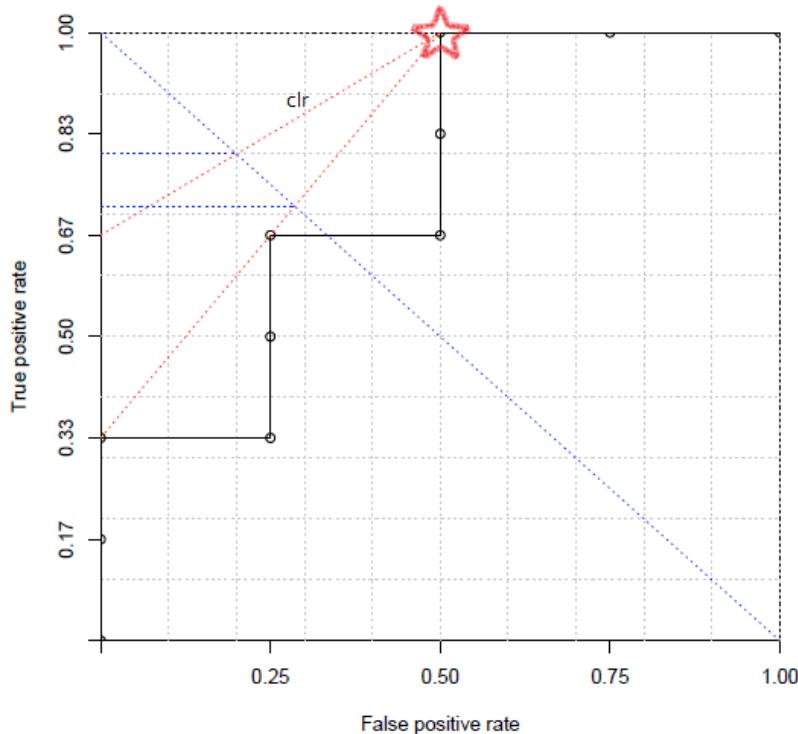


Figura 2.12: *ROC plot* di esempio con diversi classificatori identificati dai cerchietti (quelli sui bordi sinistro e superiore sono tagliati a metà).

una persona potenzialmente malata di tumore, con un forte rischio di mettere in pericolo la sua vita.

Questo esempio mette in luce come falsi positivi e falsi negativi possono non essere ugualmente importanti, pertanto si potrebbe desiderare associare agli esempi dei pesi in modo tale da dare maggiore importanza ai falsi negativi piuttosto che ai falsi positivi, o viceversa.

Si introduce a questo punto una funzione per il calcolo dei costi, `cost()`, che utilizziamo per calcolare il costo per gli esempi *FN* e *FP*. In questo modo possiamo definire un parametro *c* come:

$$c = \frac{\text{cost}(FN)}{\text{cost}(FP)} \quad (2.1)$$

Da questo valore si può ottenere l'inclinazione della retta con cui valutare quale sia il miglior classificatore:

$$\frac{1}{c}$$

Tutti i classificatori che giacciono su una stessa retta di detta pendenza commetteranno lo stesso errore, calcolato appunto con la Formula 2.1.

2.2 CLASSIFICAZIONE CON SCORING

Uno *scoring classifier* è una funzione di classificazione che assegna un punteggio alle predizioni:

$$\hat{s} : \mathcal{X} \rightarrow \mathbb{R}^k$$

L'input è lo spazio degli esempi \mathcal{X} ; l'output di questa funzione è un vettore composto da k numeri reali:

$$\hat{\mathbf{s}}(x) = (\hat{s}_1(x), \hat{s}_2(x), \dots, \hat{s}_k(x))$$

In tale vettore la i -esima componente è il punteggio assegnato alla classe C_i per l'istanza x . In termini pratici per ogni istanza x si ha un vettore $\hat{\mathbf{s}}(x)$ contenente il suo punteggio $\hat{s}_i(x)$ per ciascuna delle k classi C . Se si vuole fare un piccolo esempio si prenda il vettore dei punteggi per una istanza x e $k = 4$ possibili classi, allora si ha:

$$\hat{\mathbf{s}}(x) = (-17, 94, 8, 15)$$

I punteggi indicano che il classificatore è molto sicuro che x non appartenga alla prima classe, infatti il primo punteggio è negativo, mentre al contrario con un punteggio molto alto per la seconda classe il modello è molto sicuro che x vi appartenga. Il terzo e il quarto punteggio sono bassi, indicando che il classificatore è poco sicuro con la scelta delle rispettive classi.

Nel caso più semplice di un classificatore binario basta un solo punteggio per determinare l'appartenenza ad una o all'altra classe. Se infatti il punteggio è positivo allora l'esempio apparterrà alla classe di riferimento, se invece è negativo allora apparterrà all'altra.

Si tenga inoltre presente che i punteggi sono da considerarsi solo nel contesto del classificatore: essi infatti non vengono in alcun modo appresi da esempi reali, bensì sono un'espressione del livello di confidenza del modello rispetto alle sue predizioni, costituiscono quindi un'informazione accessoria che arricchisce la classificazione svolta dal modello.

Tali punteggi possono essere considerati anche nell'ottica geometrica di un classificatore lineare, a tale fine si riprenda in considerazione quello riportato in Figura 1.1. Il punteggio assegnato dalla funzione di scoring può essere interpretato in questo contesto come la distanza fra l'esempio e il classificatore lineare. In particolare maggiore è la distanza (positiva o negativa), maggiore è la sicurezza che la classificazione sia corretta, perché vuol dire che si è lontani dal *decision boundary* vicino al quale si possono commettere più facilmente errori, proprio per questo punteggi bassi corrispondono ad una minore confidenza nel risultato.

Si riprenda in considerazione l'albero delle feauture già visto in Figura 2.2, dove la classe positiva corrispondeva a spam e quella negativa a ham, e nelle foglie il numero di esempi del training set che

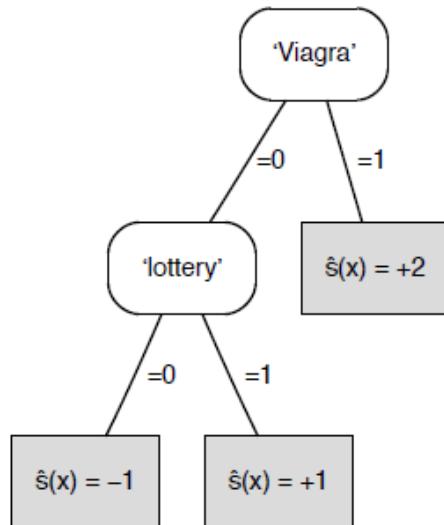


Figura 2.13: Esempio di *scoring tree* ottenuto dall’albero delle feature della Figura 2.2.

appartengono all’una e all’altra classe. A questo punto si vuole capire come definire la funzione che assegna i punteggi date le informazioni a disposizione, che per questo esempio corrisponde al logaritmo in base 2 del rapporto fra il numero di esempi positivi e negativi. Il risultato è lo *scoring tree* riportato in Figura 2.13.

2.2.1 Margini di errore e funzioni di penalizzazione

L’informazione aggiunta dal punteggio che viene assegnato dalla funzione di scoring assume ulteriore importanza nel momento in cui può essere utilizzata per migliorare il processo di apprendimento del modello. A questo fine se si considera la classe positiva $c(x)$ come $+1$ e quella negativa come -1 , si può introdurre il concetto di margine definito come segue:

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

Il margine calcolato dallo scoring classifier viene associato ad ogni esempio, è positivo quando la classificazione è corretta, negativo quando è sbagliata. In questo modo l’algoritmo di apprendimento può sfruttare il punteggio assegnato e fare tesoro di questa informazione aggiuntiva per migliorare la qualità della soluzione trovata, basandosi appunto sull’utilizzo di penalità per margini negativi e premi per quelli positivi.

Il concetto di margine può essere inoltre declinato per sfruttarlo al meglio ed apportare quindi un ulteriore vantaggio all’algoritmo di apprendimento. In particolare si vorrebbe dare una penalità maggiore

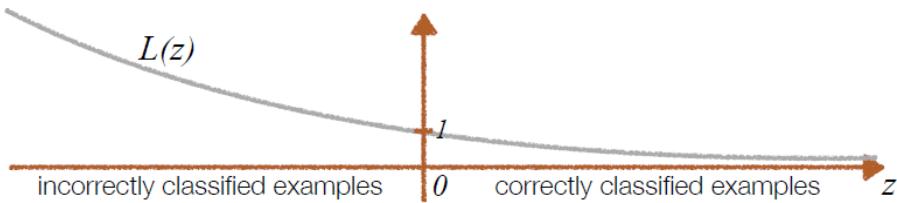


Figura 2.14: Esempio di funzione loss per regolare la penalizzazione, sull'asse delle ordinat e m correlate al margine di errore di un classificatore, sull'asse delle ascisse.

per esempi con un margine fortemente negativo, e darne una inferiore a fronte di un margine più piccolo.

Esistono ovviamente numerose funzioni con proprietà diverse che regolano le penalità assegnate in base al margine di errore del classificatore, si chiamano funzioni *loss* ed un esempio è riportato in Figura 2.14, con un grafico dove si ha il margine di errore sull'asse delle ascisse e la penalità corrispondente sull'asse delle ordinate. Una funzione loss in generale si definisce nel seguente modo:

$$L : \mathbb{R} \rightarrow [0, \infty)$$

In input si ha un valore reale, solitamente il margine stesso $z(x)$ per un esempio x , e in output viene associata una penalità $L(z(x))$ che va da 0 a ∞ e che vale sempre 1 per margine nullo.

$$\begin{cases} L(z) \geq 1 & \text{se } z < 0 \\ L(z) = 1 & \text{se } z = 0 \\ 0 \leq L(z) < 1 & \text{se } z > 0 \end{cases} \quad (2.2)$$

Facendo ora riferimento sia alla Figura 2.14 e alla Formula 2.2 è chiaro come per margini positivi la penalizzazione sia ovviamente minima, mentre per margini che crescono negativamente è associata una penalità anch'essa crescente.

Alla luce di quanto visto fino ad ora le funzioni loss assumono particolare importanza, sono proprio queste infatti che "guidano" gli algoritmi di apprendimento al miglioramento dei risultati verso soluzioni ottimali attraverso la minimizzazione della somma delle penalità associate al margine di ciascun esempio.

Come già accennato in precedenza esistono molti tipi di funzioni loss, di seguito ne vengono riassunte alcune fra le principali per completezza:

0-1 LOSS è la più banale, è un ritorno ad una situazione in cui non vengono considerati i punteggi e i margini: se l'esempio non è classificato correttamente riceverà una penalità pari a 1, 0 altrimenti:

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

Questo tipo di funzione loss non permette il raggiungimento dell'ottimalità di cui è stato fatto cenno poc'anzi.

HINGE LOSS è la generalizzazione più semplice delle funzioni loss e c'è una crescita lineare della penalizzazione per gli esempi con un margine di errore sempre maggiore:

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

LOGISTIC LOSS è un'ottima approssimazione della Hinge loss ed è continua, quindi anche differenziabile, diversamente della precedente. Questo è un vantaggio.

$$L_{\log}(z) = \log_2(1 + \exp(-z))$$

EXPONENTIAL LOSS con questa funzione loss si ha una forte penalizzazione per i margini di errore che cresce esponenzialmente, quindi gli algoritmi che si basano su questa saranno fortemente indotti a ridurli per minimizzare le penalità; la sua formulazione è:

$$L_{\exp}(z) = \exp(-z)$$

SQUARED LOSS è stata molto usata storicamente, questa penalizza i margini molto negativi in modo analogo alla Exponential loss, ma anche quelli molto positivi:

$$L_{sq}(z) = (1 - z)^2$$

2.3 RANKING

Ricordando che si parla di classificazione binaria, per la quale basta l'assegnamento di un solo punteggio ad ogni esempio (invece di un punteggio per ciascuna classe se sono più di due), come conseguenza dell'uso di punteggi si può introdurre il concetto di *ranking*, ovvero un ordinamento degli esempi basato proprio sui punteggi stessi che talvolta diventa più rilevante di questi ultimi.

Facendo nuovamente riferimento alla Figura 2.13 nella quale era riportato l'albero dei punteggi assegnati per l'esempio della classificazione delle e-mail, da questo è possibile ricavare il seguente ordinamento:

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

Questo ordinamento è stato ottenuto riportando in ordine decrescente le foglie dell'albero delle feature già visto in Figura 2.2, e di

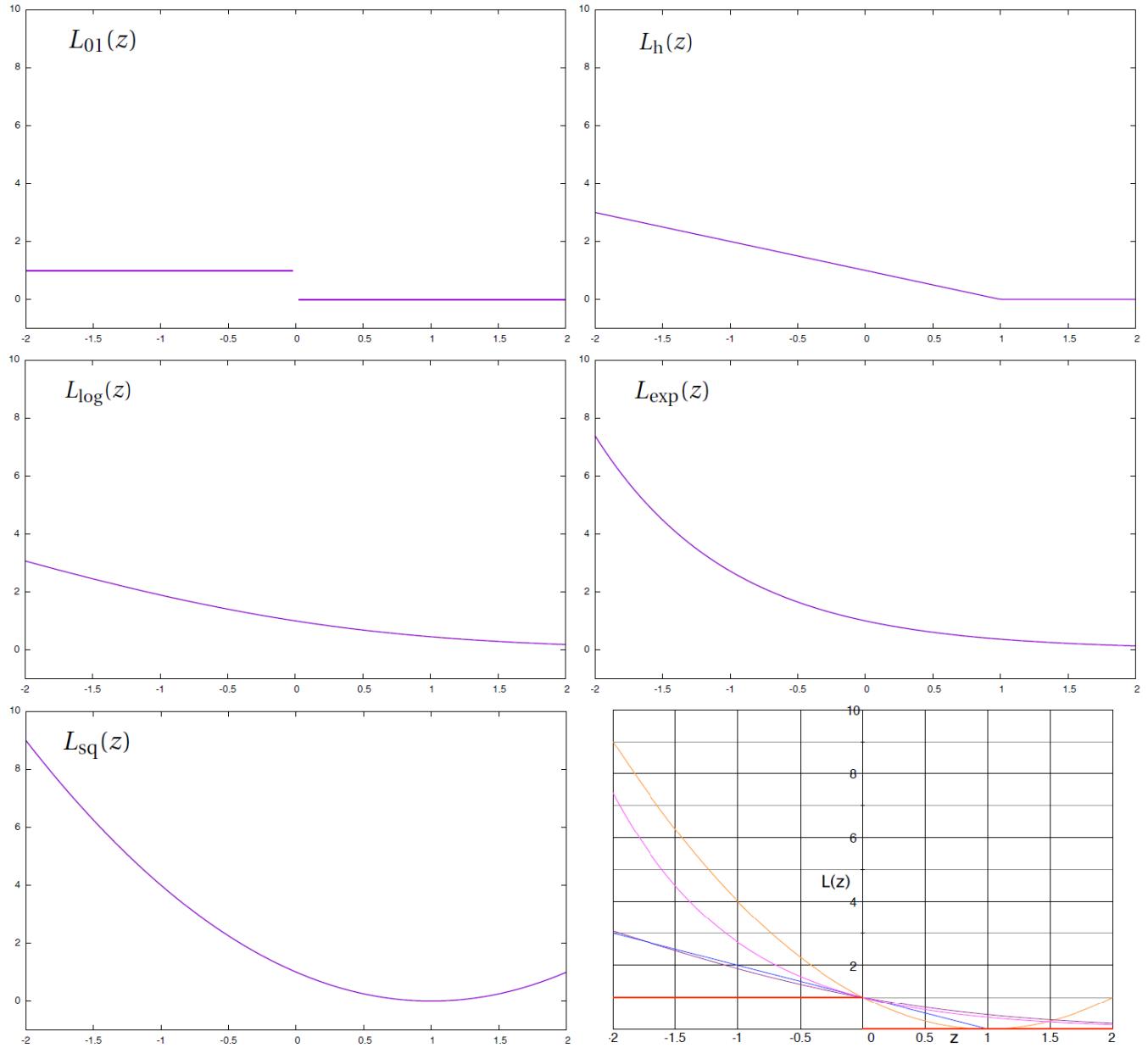


Figura 2.15: Riassunto delle principali funzioni loss per la penalizzazione.

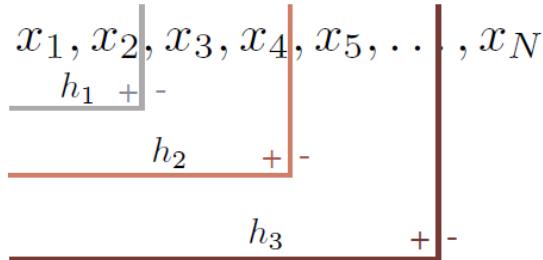


Figura 2.16: Esempio di ordinamento con tre classificatori binari h_1 , h_2 e h_3 che lo dividono in diversi punti.

conseguenza gli esempi a cui si faceva riferimento al loro interno. La prima foglia aveva un punteggio pari a +2, quindi nell'ordinamento sono presenti per primi gli esempi corrispondenti, ovvero 20 positivi e 5 negativi, e così via per tutte le altre foglie.

Una volta ottenuto un ordinamento per gli esempi si desidera tornare all'obiettivo primario, la classificazione binaria. Per fare ciò basta trovare un punto in cui dividere l'elenco di esempi ordinato, ogni punto corrisponde ad un diverso classificatore.

Si prenda ad esempio la Figura 2.16 dove è stato riportato un ranking con tre classificatori binari che dividono l'elenco ordinato in diversi punti. Ognuno dei tre classificatori divide gli esempi classificando quelli a sinistra come positivi e quelli a destra come negativi.

Si può quindi pensare ad un modo diverso per trovare il migliore classificatore. Si assegna un punteggio a ciascun esempio, si ordinano tutti gli esempi e poi si valutano tutti i possibili classificatori che dividono tale elenco ordinato di esempi alla ricerca del migliore.

2.3.1 Performance di un ranking classifier

Una volta ottenuto un ordinamento si gradirebbe valutarne la qualità, la bontà. A questo fine si introduce il *ranking error rate*, ovvero la seguente formulaccia:

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

In realtà la formula è molto più apprezzabile di quanto non sembri, basta infatti prendere in analisi tutte le sue parti. La sommatoria prende in considerazione tutte le possibili combinazioni di esempi ordinati presi a coppie, per ciascuna coppia può essere assegnata una delle due penalità:

- 1 punto se un esempio positivo x ha un punteggio associato $\hat{s}(x)$ minore del punteggio di un esempio negativo $\hat{s}(x')$, perché ciò vorrebbe dire che l'esempio positivo verrebbe ordinato dopo quello negativo, contrariamente a ciò che invece si desidera ottenere;

- $\frac{1}{2}$ punto se un esempio positivo x ha un punteggio associato $\hat{s}(x)$ uguale al punteggio di un esempio negativo $\hat{s}(x')$, perché ciò vorrebbe dire che l'esempio positivo verrebbe ordinato parimenti quello negativo, contrariamente a ciò che invece si desidera ottenere.

Poiché la sommatoria somma una penalità per ciascuna possibile coppia di esempi⁶, il risultato viene diviso per il numero di possibili coppie, ovvero il prodotto dei valori di Pos e Neg .

2.3.2 Esempi di calcolo del ranking error rate

Si prendano ora in considerazione i seguenti esempi di calcolo del *ranking error rate* per diversi ordinamenti.

2.3.2.1 Esempio 1

$$\text{rank-err}(x_1^+, x_2^+, x_3^+, x_4^-, x_5^-, x_6^+, x_7^-, x_8^-) = \frac{1+1}{4 \cdot 4} = \frac{1}{8}$$

Nel primo esempio si hanno solo 2 punti di penalità dovuti all'esempio x_6^+ che si trova dopo due esempi negativi (x_4^- e x_5^-), tutti gli altri sono nell'ordine giusto. A denominatore di ha il prodotto fra il numero di esempi positivi, 4, e il numero di esempi negativi, 4. Il risultato è $\frac{1}{8}$.

2.3.2.2 Esempio 2

$$\text{rank-err}(x_1^-, x_2^-, x_3^-, x_4^+, x_5^+, x_6^+, x_7^+, x_8^+) = \frac{5+5+5}{5 \cdot 3} = 1$$

Nel secondo esempio si ha il caso estremo in cui l'ordinamento è invertito, pertanto si ottiene il massimo delle penalità perché ciascuno dei 3 esempi negativi viene prima dei 5 esempi positivi, per un totale di 15 punti. A denominatore di ha il prodotto fra il numero di esempi positivi, 5, e il numero di esempi negativi, 3. Il risultato è 1.

2.3.2.3 Esempio 3

Si riprenda ora in considerazione l'esempio della classificazione di e-mail, in particolare con la Figura 2.17 dove viene riportato l'albero delle feature già visto in Figura 2.2 e i rispettivi punteggi assegnati con lo scoring tree in Figura 2.13. L'ordinamento che si ottiene è:

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

⁶ Ovviamente per coppie di esempi ordinate correttamente la penalità è nulla.

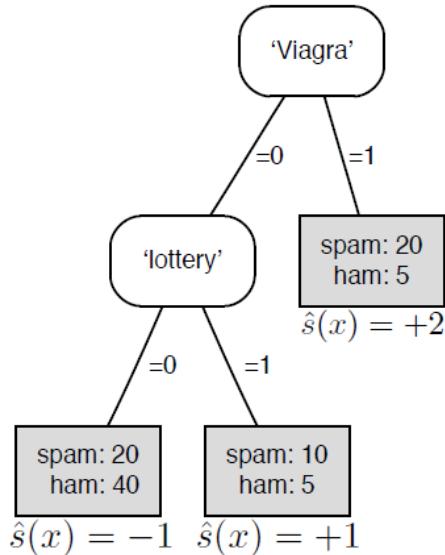


Figura 2.17: Albero delle feature per l'esempio della classificazione binaria delle e-mail contenenti spam, con annessi punteggi assegnati dallo scoring tree costruito in precedenza.

A questo punto il calcolo del ranking error rate risulta essere:

$$\begin{aligned}
 rank\text{-}err &= \frac{\frac{1}{2}(20 \cdot 5) + \frac{1}{2}(10 \cdot 5) + \frac{1}{2}(20 \cdot 40) + (5 \cdot (10 + 20)) + (5 \cdot 20)}{50 \cdot 50} \\
 &= \frac{725}{2500} \\
 &= 0.29
 \end{aligned}$$

Per completezza di seguito viene riportato in dettaglio come sono stati calcolati tutti i punti di penalità analizzando ogni addendo del numeratore:

- il primo fattore conta $\frac{1}{2}$ punto di penalità per gli esempi di classe diversa che hanno stesso punteggio, quindi per la prima foglia dell'ordinamento, quella con punteggio $+2$, si hanno 20 esempi positivi che hanno uguale punteggio a 5 esempi negativi, per cui si ha $\frac{1}{2}(20 \cdot 5) = 50$;
- in modo analogo a quanto fatto al punto precedente qua si hanno 10 esempi positivi che hanno lo stesso punteggio di 5 esempi negativi, pertanto si assegna a tutte queste coppie la penalità di $\frac{1}{2}$ punto ottenendo $\frac{1}{2}(10 \cdot 5) = 25$;
- per la terza foglia si calcola la stessa penalità per esempi di classi diverse con uguale punteggio, quindi si ha $\frac{1}{2}(20 \cdot 40) = 400$;
- il quarto elemento della somma a numeratore conta 1 punto di penalità per i 5 esempi negativi che hanno punteggio pari a

+2 e che di conseguenza vengono ordinati prima dei 10 esempi positivi che hanno punteggio +1 e dei 20 esempi positivi che hanno punteggio -1, quindi si ha $5 \cdot (10 + 20) = 150$;

- in modo analogo si assegna 1 punto di penalità per i 5 esempi negativi con punteggio +1 che nell'ordinamento precedono i 20 esempi positivi che invece hanno punteggio -1, pertanto si ha $5 \cdot 20 = 100$.

Il risultato quindi è un ranking error rate del 29%, o equivalentemente ad una accuratezza di ordinamento del 71%.

2.4 CLASSIFICATORI BASATI SULLE PROBABILITÀ

Sono maggiormente noti in inglese come *class probability estimators*, questa famiglia di classificatori determina la classe di appartenenza di un esempio con una funzione che assegna un punteggio, ovvero un vettore di probabilità relative alle diverse classi. Più formalmente si ha che la funzione di stima⁷ $\hat{\mathbf{p}}$ è un mapping dallo spazio degli esempi \mathcal{X} ad un vettore di k valori (ovvero il numero delle classi) compresi fra 0 e 1:

$$\hat{\mathbf{p}} : \mathcal{X} \rightarrow [0, 1]^k$$

Quindi la funzione $\hat{\mathbf{p}}(x)$ stima per l'esempio x la probabilità che questo appartenga a ciascuna delle k classi, ottenendo così in output un vettore di k elementi.

$$\hat{\mathbf{p}}(x) = (\hat{p}_1(x), \hat{p}_2(x), \dots, \hat{p}_k(x))$$

Questo è del tutto simile alle funzioni che assegnano un punteggio agli esempi per le diverse classi già viste in questo capitolo. La differenza principale però è che i punteggi non erano in alcun modo vincolati, mentre le probabilità di ogni singolo esempio devono sommarsi ad 1. Il vincolo quindi si scrive:

$$\sum_{i=1}^k \hat{p}_i(x) = 1$$

L'espressione dei punteggi sotto forma di valori probabilistici fornisce il vantaggio di poter leggere più facilmente i risultati ottenuti, questo costituisce un valore aggiunto perché si ha quindi a disposizione una potenza espressiva maggiore e la possibilità di svolgere confronti e, in una visione più ampia, utilizzare i risultati nell'ambito dello studio delle probabilità.

Nel caso basilare di un problema di classificazione binaria, quindi con 2 sole classi, è ancora più facile perché basta una sola delle due stime della probabilità per ottenere con una semplice sottrazione la seconda, giacché si è appena detto che devono sommare ad 1.

⁷ Si ribadisce che anche in questo caso si parla di stime, non di risultati esatti.

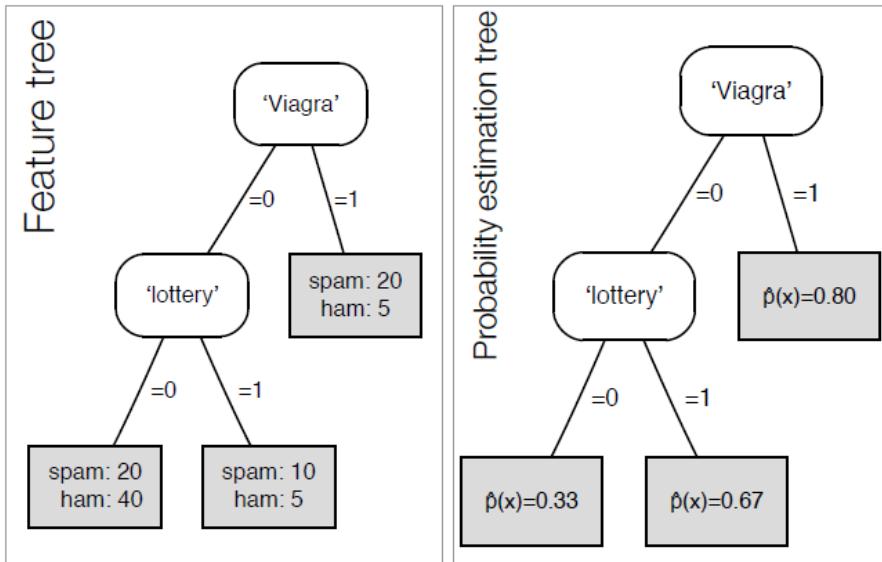


Figura 2.18: Albero delle feature per l'esempio della classificazione binaria delle e-mail contenenti spam (a sinistra), con corrispondente albero delle stime della probabilità (a destra) nelle cui foglie è riportato il valore compreso fra 0 e 1 che esprime la probabilità che l'esempio appartenga alla classe positiva, in questo caso alla classe spam.

2.4.1 Probability estimation tree

Analogamente a quanto già fatto in precedenza con gli scoring tree, anche in questo caso partendo da un albero delle feature è possibile ricavare un albero delle stime della probabilità, a.k.a. *probability estimation tree*, come quello riportato in Figura 2.18 che fa riferimento allo stesso esempio del quale fra un po' ci si stuferà anche.

All'interno delle foglie sono riportati i valori corrispondenti alla probabilità che l'esempio appartenga alla classe positiva, di conseguenza il valore complementare esprime la probabilità che questo appartenga alla classe negativa. Però ora si desidera approfondire come si ricavano questi valori probabilistici.

Nell'esempio che viene ripreso nella Figura 2.18 il valore viene ottenuto semplicemente come il rapporto fra il numero di esempi positivi e il numero totale di esempi all'interno della foglia (probabilità empirica). Salvo ulteriori assunzioni questo è il metodo migliore per calcolare le probabilità e ridurre al minimo l'errore che commette il classificatore.

2.4.2 Errore quadratrico

Come per ogni classificatore visto fino ad ora si desidera capire come calcolare l'errore commesso, assumendo che il classificatore ideale fornisce un vettore $I_{c(x)}$ che contiene per la classe corretta il valore 1

e 0 per tutte le altre.

Si può quindi calcolare per un esempio x l'errore al quadrato, o *squared error* che dir si voglia, $\text{SE}(x)$ come:

$$\begin{aligned}\text{SE}(x) &= \frac{1}{2} \|\hat{\mathbf{p}}(x) - I_{c(x)}\|_2^2 \\ &= \frac{1}{2} \sum_{i=1}^k (\hat{p}_i(x) - I[c(x) = C_i])^2\end{aligned}\quad (2.3)$$

La brutta formula appena presentata però può sembrare poco familiare, pertanto alcuni esempi della sua applicazione.

2.4.2.1 Esempio 1

Si assuma in questo primo esempio di avere una stima delle probabilità per l'esempio x e per 3 classi come segue:

$$\hat{\mathbf{p}}(x) = (0.7, 0.1, 0.2)$$

E si assuma inoltre che la classe corretta sia C_1 , quindi:

$$I_{c(x)} = (1, 0, 0)$$

L'errore commesso allora si può calcolare nel seguente modo:

$$\begin{aligned}\text{SE}(x) &= \frac{\|(0.7, 0.1, 0.2) - (1, 0, 0)\|_2^2}{2} \\ &= \frac{\|(-0.3, 0.1, 0.2)\|_2^2}{2} \\ &= \frac{0.09 + 0.01 + 0.04}{2} \\ &= \frac{0.14}{2} = 0.07\end{aligned}$$

2.4.2.2 Esempio 2

Si assuma in questo secondo esempio di avere una stima delle probabilità per l'esempio x e per 3 classi come segue:

$$\hat{\mathbf{p}}(x) = (0, 1, 0)$$

In questo caso la stima è certa che l'esempio appartenga alla classe C_2 . Si assuma a questo punto che la classe corretta sia C_1 , quindi:

$$I_{c(x)} = (1, 0, 0)$$

L'errore commesso allora si può calcolare nel seguente modo:

$$\begin{aligned}\text{SE}(x) &= \frac{\|(0, 1, 0) - (1, 0, 0)\|_2^2}{2} \\ &= \frac{\|(-1, 1, 0)\|_2^2}{2} \\ &= \frac{1 + 1 + 0}{2} \\ &= \frac{2}{2} = 1\end{aligned}$$

In questo caso si è voluto mettere in evidenza sia l'errore che viene commesso da un classificatore sicuro della classe sbagliata, sia l'importanza della normalizzazione con il fattore $\frac{1}{2}$. Ambedue questi aspetti portano l'errore al valore massimo pari ad 1.

2.4.2.3 *Esempio 3*

In questo terzo esempio verranno prese in considerazione diverse casistiche al fine di mostrare come diversi vettori che classificano in modo simile una stessa istanza possano commettere errori più o meno grandi.

Si assuma che la classe corretta sia $c(x) = C_1$, e che si abbiano due diversi vettori:

$$\begin{aligned}\hat{\mathbf{p}}_1(x) &= (0.7, 0.1, 0.2) \\ \hat{\mathbf{p}}_2(x) &= (0.99, 0, 0.01)\end{aligned}$$

È evidente come il secondo classificatore sia molto più sicuro della scelta di C_1 sulle altre classi rispetto al primo. Questa differenza ha una ricaduta sull'errore non indifferente (a questo punto tralasciamo tutti i conti poiché banali) dovuta al fatto che la differenza viene elevata al quadrato, quindi più la stima si discosta dalla realtà, maggiore sarà l'errore poiché questo cresce quadraticamente.

$$\begin{aligned}SE_1(x) &= 0.07 \\ SE_2(x) &= 0.0001\end{aligned}$$

In conclusione: nonostante tutti e due i classificatori siano corretti, il secondo commette un errore di ben due ordini di grandezza inferiore, una differenza non da poco che lo rende ovviamente maggiormente desiderabile.

2.4.2.4 *Esempio 4*

In questo quarto ed ultimo esempio verranno nuovamente prese in considerazione diverse casistiche al fine di mostrare come diversi vettori, ambedue che classificano in modo più o meno errato una stessa istanza, possano commettere errori più o meno grandi.

Si assuma che la classe corretta sia $c(x) = C_3$, e che si abbiano due diversi vettori:

$$\begin{aligned}\hat{\mathbf{p}}_1(x) &= (0.7, 0.1, 0.2) \\ \hat{\mathbf{p}}_2(x) &= (0.99, 0, 0.01)\end{aligned}$$

In questo caso i due modelli sono errati, ma il calcolo del loro errore risulta comunque interessante.

$$\begin{aligned}SE_1(x) &= 0.57 \\ SE_2(x) &= 0.98\end{aligned}$$

In questo esempio si ha l'effetto opposto a quello del precedente esempio: la certezza sulla prima classe si rivela fatale per il secondo modello, che commette un errore molto alto, rispetto al primo che invece a questo punto se la cava meglio. Ciò mette in evidenza come, nonostante fosse stato detto che il secondo modello fosse preferibile al primo, sia comunque necessario trovare una sorta di bilanciamento fra i due aspetti.

Ovviamente tutte le analisi svolte fino ad ora su questi esempi sono strettamente vincolate dal metodo di calcolo dell'errore adottato, che ovviamente può variare e portare a conclusioni diverse rispetto a queste, ma soprattutto può portare l'algoritmo di apprendimento a diversi risultati.

2.4.3 Errore quadratico medio

L'errore quadratico è stato utile fino a questo momento per calcolare quanto il modello si discostasse dalla corretta classificazione per un preciso esempio. Si vuole però svolgere una valutazione complessiva che tenga conto di tutti gli esempi di cui si dispone (T_e), pertanto si introduce l'errore quadratico medio, a.k.a. *mean squared error*. Come suggerisce il nome stesso, si tiene conto dell'errore commesso su tutti gli esempi facendo la media dei singoli errori quadratici.

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

Se si vuole applicare questa formula al solito esempio si faccia riferimento ai valori riportati nei due alberi in Figura 2.18. Il procedimento per il calcolo dell'errore quadratico medio si articola così:

- si considerino le classi nel seguente ordine $C = \{C_1 = \text{spam}, C_2 = \text{ham}\}$;
- si identifichino i vettori con le stime probabilistiche che costituiscono il modello, in questo caso sono tre e corrispondono alle foglie dell'albero della Figura 2.18 da sinistra a destra: $(0.33, 0.67)$, $(0.67, 0.33)$ e $(0.8, 0.2)$;
- per le e-mail contenenti spam il corrispondente vettore ideale è $I_{c(\text{spam})} = (1, 0)$, mentre per tutte le altre è $I_{c(\text{ham})} = (0, 1)$;
- a questo punto con la Formula 2.3 si può calcolare l'errore quadratico per i singoli esempi, operazione agevolata dal fatto che questi sono raccolti all'interno delle foglie alle quali poi sono associati i vettori con i valori delle probabilità con cui vengono predetti. Di seguito un il calcolo dell'errore quadratico per un

esempio predetto correttamente come ham nella prima foglia (sempre considerandole da sinistra):

$$\begin{aligned} \text{SE} &= \frac{\|(0.33, 0.67) - (0, 1)\|_2^2}{2} \\ &= \frac{\|(0.33, -0.33)\|_2^2}{2} \\ &= \frac{0.33^2 + (-0.33)^2}{2} = 0.1089 \end{aligned}$$

- l'errore quadratico appena ottenuto va moltiplicato per il numero di esempi correttamente classificati come ham nella prima foglia, quindi:

$$0.1089 \cdot 40 = 4.356$$

- questa operazione va svolta per gli esempi positivi (spam) e quelli negativi (ham) di ognuna delle tre foglie, i risultati sommati e normalizzati in base al numero totale di esempi.

A questo punto si hanno tutti gli strumenti per calcolare l'errore quadratico medio:

$$\begin{aligned} \text{MSE}(Te) &= \frac{1}{100} \cdot \left(\frac{\|(0.33, 0.67) - (0, 1)\|_2^2}{2} \cdot 40 + \frac{\|(0.33, 0.67) - (1, 0)\|_2^2}{2} \cdot 20 + \right. \\ &\quad + \frac{\|(0.67, 0.33) - (1, 0)\|_2^2}{2} \cdot 10 + \frac{\|(0.67, 0.33) - (0, 1)\|_2^2}{2} \cdot 5 + \\ &\quad \left. + \frac{\|(0.8, 0.2) - (1, 0)\|_2^2}{2} \cdot 20 + \frac{\|(0.8, 0.2) - (0, 1)\|_2^2}{2} \cdot 5 \right) \\ &= \frac{1}{100} \cdot \left(\frac{\|(0.33, -0.33)\|_2^2}{2} \cdot 40 + \frac{\|(-0.67, 0.67)\|_2^2}{2} \cdot 20 + \right. \\ &\quad + \frac{\|(-0.33, 0.33)\|_2^2}{2} \cdot 10 + \frac{\|(0.67, -0.67)\|_2^2}{2} \cdot 5 + \\ &\quad \left. + \frac{\|(-0.2, 0.2)\|_2^2}{2} \cdot 20 + \frac{\|(0.8, -0.8)\|_2^2}{2} \cdot 5 \right) \\ &= \frac{1}{100} \cdot \left(\frac{0.33^2 + (-0.33)^2}{2} \cdot 40 + \frac{-0.67^2 + 0.67^2}{2} \cdot 20 + \right. \\ &\quad + \frac{-0.33^2 + 0.33^2}{2} \cdot 10 + \frac{0.67^2 + (-0.67)^2}{2} \cdot 5 + \\ &\quad \left. + \frac{-0.2^2 + 0.2^2}{2} \cdot 20 + \frac{0.8^2 + (-0.8)^2}{2} \cdot 5 \right) \\ &= \frac{0.33^2 \cdot 40 + 0.67^2 \cdot 20 + 0.33^2 \cdot 10 + 0.67^2 \cdot 5 + 0.2^2 \cdot 20 + 0.8^2 \cdot 5}{100} \\ &= \frac{20.6675}{100} = 0.206675 \end{aligned}$$

2.4.4 La probabilità empirica

Nell'ambito dell'apprendimento automatico capita molto sovente di dover stimare delle probabilità partendo da un insieme di dati, pertanto è molto rilevante trovare il modo migliore per farlo in base al dominio e alle esigenze. In molti casi la probabilità empirica è una buona soluzione.

Se si ha un insieme S di istanze etichettate e si denota con n_i il numero di istanze che appartengono ad una delle k classi C_i , allora il vettore della probabilità empirica associato è:

$$\dot{p}(S) = \left(\frac{n_1}{|S|}, \dots, \frac{n_k}{|S|} \right)$$

Il problema con questa stima della probabilità è collegato ad insiemi di dati ridotti. In tali casi alcune classi contenenti poche istanze potrebbero subire una stima non sufficientemente precisa o corretta, o addirittura non essere per nulla presenti, il ché vorrebbe dire che nel vettore della probabilità empirica per tale classe sfortunata potrebbe corrispondere un valore nullo. Con un valore pari a 0 per una certa classe il modello riterrà quindi erroneamente che questa non si manifesti mai, invece che raramente come sarebbe corretto.

2.4.4.1 Correzione di Laplace

Poiché si è concluso che nel vettore della probabilità stimata si vogliono evitare vari nulli, pertanto si introduce tale correzione di Laplace. Questa sorta di limatura, raffinamento della stima, consiste nella seguente aggiunta:

$$\dot{p}_i(S) = \frac{n_i + 1}{|S| + k} \tag{2.4}$$

Una analisi più attenta di questa aggiunta rende possibile capire che in realtà si tratta di un esempio *fake* per ciascuna classe. In questo modo se n_i dovesse valere 0, si ha la certezza che il risultato non sarà nullo. E se si aggiunge 1 per ciascuna classe, allora a denominatore bisogna sommare tutti questi finiti esempi introdotti che saranno esattamente k . Tale correzione è una aggiunta distribuita uniformemente.

Tuttavia è possibile anche applicare una correzione non uniforme se si dispone di maggiori informazioni riguardanti le probabilità legate alle k classi:

$$\dot{p}_i(S) = \frac{n_i + m \cdot \pi_i}{|S| + m} \tag{2.5}$$

Questa tecnica più generale permette di scegliere i due parametri π_i e m . Il primo riflette una conoscenza a priori della probabilità associata ad ogni classe, il secondo il numero di istanze *fake* che si vogliono

aggiungere in totale al dataset e che verranno distribuite fra le classi secondo la probabilità π_i che si conosce a priori.

Si noti che la correzione di Laplace della Formula 2.4 è un caso particolare della versione generale della Formula 2.5 nella quale i parametri sono $m = k$ per aggiungere tanti esempi quanti sono le classi e $\pi_i = \frac{1}{k}$ per avere una distribuzione uniforme di detti esempi fra tutte le classi.

3

OLTRE LA CLASSIFICAZIONE BINARIA

Il passo successivo al problema della classificazione binaria affrontato nel precedente Capitolo 2 è il passaggio alla classificazione multiclass, ovvero con più di due classi.

3.1 CLASSIFICAZIONE MULTICLASSE

Vi sono principalmente due metodi per costruire modelli che risolvano task di questo tipo: nel primo caso si cerca di costruire un algoritmo di apprendimento in grado di classificare su più classi, nel secondo si combinano più classificatori binari (algoritmi di classificazione binari). Questo secondo approccio sarà quello affrontato nel presente Capitolo.

Anche le misure delle performance di un classificatore binario ampiamente presentate nella sottosezione 2.1.2 verranno utilizzate per valutare algoritmi che lavorano con più classi, si possono infatti costruire matrici di contingenza per più di due classi di cui un esempio è riportato nella seguente Tabella 3.1.

	Predetti			Recall per classe
	C ₁	C ₂	C ₃	
C ₁	15	2	3	20
Attuali C ₂	7	15	8	30
C ₃	2	3	45	50
	24	20	56	100
Precisione per classe	$\frac{15}{24} = 0.63$	$\frac{15}{20} = 0.75$	$\frac{45}{56} = 0.80$	$acc = \frac{15+15+45}{100} = 0.75$

Tabella 3.1: Tabella di contingenza costruita per un esempio con 3 classi con alcune misure delle performance (recall e precisione) per ciascuna classe e la accuratezza finale.

Sebbene questa tabella di contingenza sia più grande la sua lettura rimane molto facile: nelle colonne ci sono le previsioni per ogni classe fatte dal modello, sulle righe invece l'effettiva suddivisione degli esempi per classi, sulla diagonale sono evidenziati i valori corrispondenti al numero di istanze classificate correttamente per ogni classe, la quarta riga riporta le somme dei valori di ogni colonna e la quarta colonna le somme dei valori di ogni riga. La Tabella 3.1 è stata ulteriormente arricchita con alcune misure più importanti delle performance: l'accuratezza del modello, la precisione del modello per ogni classe e la recall del modello per ogni classe.

Ci sono numerosi modi per combinare più classificatori binari per ottenere un classificatore multiclasse, altrimenti noto come *k-class classifier*, che possono essere riassunti principalmente sotto due famiglie:

- schemi *one-vs-rest*:
 - non ordinati;
 - ordinati;
- schemi *one-vs-one*:
 - simmetrici;
 - asimmetrici.

Di seguito viene presentato un esempio di apprendimento con un dataset di istanze etichettate con 3 classi applicando e confrontando fra di loro questi quattro diversi approcci.

3.1.1 One-vs-rest - Non ordinato

Il primo approccio per la risoluzione del task di classificazione multiclasse prevede la combinazione di tre diversi classificatori binari come riportato in Figura 3.1. Questo si traduce in tre diversi problemi di apprendimento, uno per ciascun classificatore binario \hat{c}_1 , \hat{c}_2 e \hat{c}_3 isolando in ciascuno una delle 3 diverse classi.

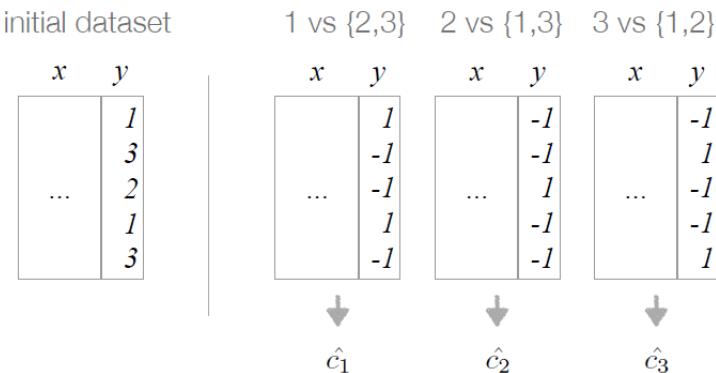


Figura 3.1: Esempio di utilizzo di tre classificatori binari per risolvere un problema di classificazione con 3 classi. A sinistra è riportato il dataset iniziale, a destra i tre classificatori utilizzati per la classificazione con schema one-vs-rest non ordinata.

In ciascuno dei tre problemi di apprendimento i dati iniziali rimangono invariati, fatta eccezione per le etichette che vengono aggiornate in base alla classe che viene isolata. A questo punto le etichette avranno un valore binario: +1 se l'istanza appartiene alla classe isolata, -1 se invece appartiene ad una delle altre due.

Tale schema può essere riassunto più formalmente con una *output-code matrix*, ogni riga corrisponde ad una classe e ogni colonna ad un classificatore binario. In altre parole attraverso tale matrice si specifica

$$\begin{array}{c}
 \begin{matrix} & \begin{matrix} 1 & 2 & 3 \end{matrix} \\ \begin{matrix} vs \\ \{2,3\} \end{matrix} & \begin{matrix} vs \\ \{1,3\} \end{matrix} & \begin{matrix} vs \\ \{1,2\} \end{matrix} \end{matrix} \\
 C_1 \quad \begin{pmatrix} +1 & -1 & -1 \\ -1 & +1 & -1 \\ -1 & -1 & +1 \end{pmatrix} \\
 C_2 \\
 C_3
 \end{array}$$

per ogni classificatore quale classe esso sarà in grado di distinguere, intuitivamente utilizzando l'etichetta $+1$, e quali invece no con -1 , implicando la sostituzione delle etichette originarie per il dataset di ogni classificatore.

Tale matrice è utile anche per analizzare i risultati del classificatore multiclassse. Quest'ultimo restituisce un vettore composto dalle etichette binarie, una per ciascun classificatore binario che lo componete, che identifica la classe predetta. Confrontando tale vettore con le righe della matrice si risale alla classe predetta.

3.1.2 One-vs-rest - Ordinato

Per ridurre la mole di lavoro si cerca di sfruttare un ordinamento: si svolge il primo confronto con il primo classificatore, se questo riconosce l'esempio come appartenente alla classe C_1 il problema è risolto, altrimenti si rimane in dubbio fra le altre due classi. In questo caso si può sfuggire un secondo classificatore che si concentra solo su queste due.

$$\begin{array}{c}
 \begin{matrix} & \begin{matrix} 1 & 2 \end{matrix} \\ \begin{matrix} vs \\ \{2,3\} \end{matrix} & \begin{matrix} vs \\ \{3\} \end{matrix} \end{matrix} \\
 C_1 \quad \begin{pmatrix} +1 & 0 \\ -1 & +1 \\ -1 & -1 \end{pmatrix} \\
 C_2 \\
 C_3
 \end{array}$$

3.1.3 One-vs-one - Simmetrico

Nei primi due approcci i modelli isolavano una classe contro tutte le altre, al contrario in questo e nel prossimo schema si confronteranno le classi a coppie, una contro una. Anche in questo caso la costruzione della matrice è molto semplice: ogni riga corrisponde ad una classe, ogni colonna ad un classificatore binario. I valori nulli corrispondono a classi non contemplate dal modello.

Questo approccio è simmetrico perché un modello che distingua fra la classe C_1 e la classe C_2 non considera l'ordine delle classi, il cui cambiamento richiederebbe solo l'inversione dei segni dei risultati.

$$\begin{array}{ccc}
 & \begin{matrix} 1 & 1 & 2 \\ vs & vs & vs \\ 2 & 3 & 3 \end{matrix} \\
 C_1 & \left(\begin{matrix} +1 & +1 & 0 \\ -1 & 0 & +1 \\ 0 & -1 & -1 \end{matrix} \right) \\
 C_2 & \\
 C_3 &
 \end{array}$$

3.1.4 One-vs-one - Asimmetrico

Tuttavia possono presentarsi dei casi in cui l'ordine con cui vengono considerate le classi è rilevante, pertanto con l'approccio asimmetrico si considerano tutti i classificatori e i rispettivi speculari per un risultato più robusto.

$$\begin{array}{ccccccc}
 & \begin{matrix} 1 & 2 & 1 & 3 & 2 & 3 \\ vs & vs & vs & vs & vs & vs \\ 2 & 1 & 3 & 1 & 3 & 2 \end{matrix} \\
 C_1 & \left(\begin{matrix} +1 & -1 & +1 & -1 & 0 & 0 \\ -1 & +1 & 0 & 0 & +1 & -1 \\ 0 & 0 & -1 & +1 & -1 & +1 \end{matrix} \right) \\
 C_2 & \\
 C_3 &
 \end{array}$$

3.1.5 Classificazione con le matrici output-code

Per poter svolgere la classificazione è quindi necessario disporre della matrice output-code. Come anticipato infatti il vettore risultante per un'istanza va confrontato con ogni riga della matrice. Vince la classe associata alla riga più vicina al vettore risultante. Il concetto di "vincita" può essere declinato in vari modi, di seguito viene riportato quello maggiormente in uso.

Data la classe corretta c_i e il vettore risultante dal classificatore w_i per una istanza, allora è possibile calcolare la distanza come:

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

Se ad esempio si considera per una certa istanza il vettore risultante $w = (+1, -1, -1)$, allora le distanze fra questo e le tre righe della matrice vista nella sottosezione 3.1.1 saranno:

$$\begin{aligned}
 d(w, C_1) &= \frac{(1 - (+1) \cdot (+1)) + (1 - (-1) \cdot (-1)) + (1 - (-1) \cdot (-1))}{2} = 0 \\
 d(w, C_2) &= \frac{(1 - (-1) \cdot (+1)) + (1 - (+1) \cdot (-1)) + (1 - (-1) \cdot (-1))}{2} = 2 \\
 d(w, C_3) &= \frac{(1 - (-1) \cdot (+1)) + (1 - (-1) \cdot (-1)) + (1 - (+1) \cdot (-1))}{2} = 2
 \end{aligned}$$

3.1.6 Difficoltà applicative dei diversi approcci

Tenendo presente che la scelta dell'approccio è sempre basata sul dominio, sui dati a disposizione e su eventuali test, nel guidare tale decisione si possono tenere conto delle difficoltà che hanno le due famiglie di approcci che sono state prese in considerazione.

Per gli approcci basati sullo schema *one-vs-rest* ci sono problemi di bilanciamento nei dataset utilizzati nei singoli classificatori binari, problemi che persistono anche se il dataset di partenza è ben bilanciato. È molto facile fare un esempio che renda palese lo sbilanciamento di cui si parla. Si prenda in considerazione un dataset con 10 classi e 10 istanze per ciascuna classe, per un totale di 100 istanze: è un dataset perfettamente bilanciato, l'ideale. Tuttavia applicando l'approccio *one-vs-rest* il dataset viene diviso in modo molto sbilanciato perché una classe isolata avrà solamente 10 istanze, contro le 90 delle altre 9 classi contro cui si pone. Tale problema non va sottovalutato perché crea difficoltà alla maggior parte degli algoritmi di apprendimento.

Il problema appena rilevato ovviamente non si pone nel caso di approcci basati sullo schema *one-vs-one* grazie all'introduzione dell'etichetta 0 per le istanze che non appartengono a nessuna delle due classi prese in considerazione dal classificatore. Tuttavia anche in questo caso si ha un problema da non sottovalutare: la riduzione del dataset. Considerando infatti le classi a coppie si riduce il numero di istanze da utilizzare per l'addestramento, ché in caso di dataset poco popolati in partenza crea gravi difficoltà agli algoritmi di apprendimento con conseguenti performance ridotte.

3.2 REGRESSIONE

I task di regressione possono essere visti in un'ottica di evoluzione della classificazione vista fino ad ora. In questo problema di apprendimento non si ha più un limitato numero di classi, l'output richiesto ora ha un dominio molto più ampio, quello dei valori continui: \mathbb{R} . Si ha quindi una funzione di stima, anche nota come regressore (in inglese *regressor*), definita come:

$$\hat{f}: \mathcal{X} \rightarrow \mathbb{R}$$

Il problema di apprendimento della regressione quindi è imparare una funzione di stima dagli esempi x_i ai quali sono associati valori continui dati dalla vera funzione f^1 : $(x_i, f(x_i))$.

Cambiando l'obiettivo della funzione da un numero relativamente ristretto di classi ad un infinito spazio delle soluzioni, l'algoritmo cercherà di stimare con la massima precisione i valori associati ad ogni

¹ Si ricorda che il cappelletto sulla funzione \hat{f} indica che si tratta di una approssimazione della vera funzione f .

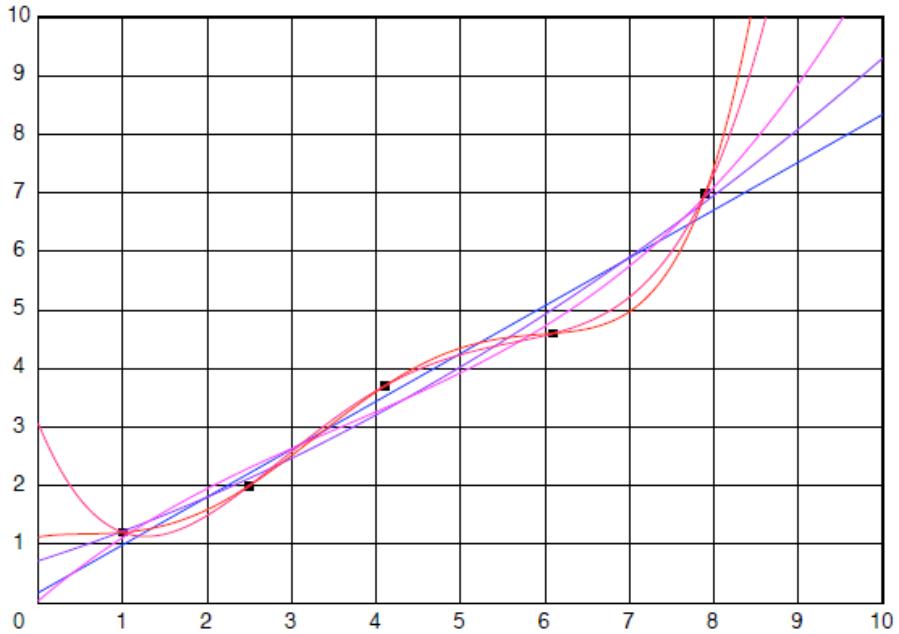


Figura 3.2: Diverse funzioni che cercano di approssimare l'andamento dei cinque punti riportati nel grafico. Modelli più complessi sono più precisi, con il rischio dell'overfitting, modelli più semplici sono meno precisi, ma possono generalizzare meglio.

esempio, portandosi sicuramente verso il problema dell'*overfitting*. Tenendo presente questo ci si deve preparare psicologicamente ad accettare un bilanciamento fra precisione ed approssimazione delle soluzioni, è inevitabile a fronte delle normali fluttuazioni dei dati nel dataset che il modello non può catturare con esattezza.

Il vero scopo del regressore, se non può essere troppo preciso senza peccare di overfitting, è riuscire ad approssimare nel migliore dei modi l'andamento della funzione f .

3.2.1 Il problema dell'overfitting II - Il ritorno

Era stato già introdotto e discusso nella sottosezione 1.1.2, ora il problema dell'overfitting fa di nuovo ritorno con l'argomento della regressione e delle funzioni reali.

Un esempio di quanto appena descritto si può vedere nella Figura 3.2 dove si ha un grafico con cinque punti e si vuole trovare la funzione che riesce a passare per tutti questi. Fra le cinque funzioni disegnate ve ne sono alcune molto precise, che toccano esattamente tutti i punti o vi passano molto vicino, ma peccano di overfitting e non generalizzano adeguatamente oltre ad essere anche complesse. Al contrario ci sono funzioni molto più semplici (la più semplice è una retta) che non soffrono di overfitting, generalizzano maggiormente, e anche se non toccano tutti i punti vi passano comunque abbastanza vicino da poter commettere un errore contenuto.

Se si considera un generico dataset con n istanze, una funzione polinomiale che passi per tutte queste dovrebbe essere di grado $n - 1$. Come ormai si sa bene, una soluzione del genere sarebbe perfetta per il training set, ma non generalizzerebbe adeguatamente per avere performance soddisfacenti anche sul test set. E questo è male. Senza contare che i dataset non contengono poche istanze, quindi stimare funzioni di grado molto alto può anche rivelarsi complesso e costoso.

Per evitare quindi il problema dell'overfitting e abbracciare la generalizzazione tanto agognata bisogna ridurre il numero dei parametri stimati, ovvero il grado della funzione di stima, per ottenere un modello più semplice secondo il principio del *Novacula Occami*².

3.2.2 Il problema bias-variance

Sulla base di quanto appena discusso circa l'overfitting e la semplificazione dei modelli, si potrebbe essere indotti a credere che questo principio debba guidare sempre la risoluzione dei problemi di apprendimento verso modelli tendenzialmente lineari, i più semplici che ci sono. Questa è una conclusione inesatta, ed è il nocciolo del *bias-variance dilemma*.

Non basta infatti puntare al modello più semplice, perché questo potrebbe rivelarsi insufficiente per minimizzare le penalità ed ottenerne un modello in grado di svolgere con accuratezza le predizioni.

Si può quindi concludere che modelli molto complessi soffrono della varianza (*variance*) dei dati nel dataset ma sono molto precisi, fino all'overfitting, e non hanno bisogno di alcun *bias*; al contrario modelli molto semplici non patiscono la varianza, ma al contrario devono introdurre un *bias* che non può essere nemmeno risolto da grandi quantità di dati in input.

Alla luce di quanto appena detto si può quindi analizzare l'errore che un modello commette e distinguerne due cause. I modelli troppo semplici sono maggiormente soggetti ad errori dovuti alla mancanza di complessità che impedisce loro di stimare con sufficiente precisione; i modelli troppo complessi che invece commettono errori perché si adattano troppo ai dati e quindi vengono penalizzati non appena le istanze si discostano anche poco.

Questi concetti di bias e varianza visti fino ad ora possono essere formalizzati. L'errore quadratico atteso dal regressore $\hat{f}(x)$ per una singola istanza x del dataset può essere decomposto con la seguente formula:

$$\begin{aligned} E \left[(f(x) - \hat{f}(x))^2 \right] &= \left(f(x) - E[\hat{f}(x)] \right)^2 + E \left[(\hat{f}(x) - E[\hat{f}(x)])^2 \right] \\ &= \text{Bias}^2(\hat{f}(x)) + \text{Var}(\hat{f}(x)) \end{aligned} \quad (3.1)$$

² Altrimenti noto come rasoio di Occam.

All'interno di questa formula è possibile identificare due principali elementi:

- $\left(f(x) - E[\hat{f}(x)] \right)^2$ questo primo termine è il bias, vale 0 se il regressore è mediamente corretto, ed è il quadrato della differenza (ovvero la distanza) fra la funzione f corretta e il valore medio del modello \hat{f} che si sta costruendo;
- $E \left[\left(\hat{f}(x) - E[\hat{f}(x)] \right)^2 \right]$ questo secondo termine è la varianza, ovvero il quadrato della differenza attesa tra la funzione stimata e la funzione stimata attesa, è l'errore causato dalle variazioni dei dati rispetto alla loro media.

La dimostrazione che è vera la Formula 3.1 a partire dal termine a sinistra dell'uguale viene di seguito riportata.

$$\begin{aligned}
 E \left[\left(f(x) - \hat{f}(x) \right)^2 \right] &= E \left[f(x)^2 - 2f(x)\hat{f}(x) + \hat{f}(x)^2 \right] \\
 &= E[f(x)^2] - 2f(x)E[\hat{f}(x)] + E[\hat{f}(x)^2] \\
 &= E[f(x)^2] - 2f(x)E[\hat{f}(x)] + E[\hat{f}(x)^2] - E[\hat{f}(x)]^2 + E[\hat{f}(x)]^2 \\
 &= E[\hat{f}(x)^2] - E[\hat{f}(x)]^2 + f(x)^2 - 2f(x)E[\hat{f}(x)] + E[\hat{f}(x)]^2 \\
 &= E \left[\left(\hat{f}(x) - E[\hat{f}(x)] \right)^2 \right] + \left(f(x) - E[\hat{f}(x)] \right)^2 \\
 &= \text{Var}(\hat{f}(x)) + \text{Bias}^2(\hat{f}(x))
 \end{aligned}$$

Le operazioni che vengono svolte sono molto semplici:

- nel primo passaggio viene svolto il quadrato del binomio;
- nel secondo passaggio l'operatore E viene distribuito sui singoli addendi ottenuti poc'anzi;
- nel terzo passaggio viene tirato fuori dal secondo addendo la componente costante che non varia nella stima, ovvero $2f(x)$, e poi viene semplicemente sommata e sottratta una stessa quantità che si rivelerà utile in seguito;
- nel quarto passaggio viene tirato fuori dall'operatore E anche $f(x)^2$ poiché anche in questo caso trattandosi della funzione vera è una quantità costante, vengono anche riordinati i diversi termini;

- nel quinto passaggio i primi due termini sono una forma equivalente della varianza³, gli altri tre invece vengono riscritti come quadrato di una differenza;
- nel sesto ed ultimo passaggio vengono risconosciute sia la varianza che il quadrato del bias. QED.

3.3 APPRENDIMENTO NON SUPERVISIONATO E MODELLI DESCRITTIVI

Fino a questo momento sono stati tenuti in considerazione quasi esclusivamente i task di tipo predittivo, ovvero quelli il cui scopo era costruire un modello in grado di svolgere previsioni corrette su determinate classi di istanze.

In questa sezione si vogliono affrontare quindi i task di tipo descrittivo, che nella sottosezione 1.1.3 erano stati brevemente introdotti, e in aggiunta a questi in generale i task correlati all'apprendimento non supervisionato. Questi due argomenti infatti sono parzialmente sovrapposti, in quanto come si poteva già vedere dalla Tabella 1.1 l'apprendimento non supervisionato può essere utilizzato proprio per ottenere modelli di tipo descrittivo.

Se nei task di tipo predittivo si avevano a disposizione dati etichettati, training set e test set, in quelli di tipo descrittivo la situazione è diversa, in particolare le differenze sono principalmente due:

- nell'apprendimento descrittivo non si dispone di dati etichettati, ma di dati "e basta", pertanto non si distingue più un training set;
- lo scopo dell'apprendimento descrittivo è apprendere un modello in grado di descrivere i dati e non predire la classe di dati mai visti.

Si può quindi affermare che per i problemi di apprendimento descrittivo il task corrisponde esattamente al learning problem, proprio come si può vedere nella Figura 3.3. In tale immagine si possono notare le differenze appena presentate anche tramite il confronto con la Figura 1.2 che invece era riferita ai task di tipo predittivo.

³ Dalla Formula della varianza $E[(x - E[x])^2]$ è possibile sviluppare il quadrato e ottenere $E[x^2 - 2xE[x] + [E(x)]^2]$ che, sfruttando la linearità del valore atteso e redistribuendolo, regalerà la formula $[E[x^2] - 2E[x]E[x] + [E(x)]^2]$ a cui possono essere applicati semplici passaggi algebrici per ottenere la formula equivalente della varianza:

$$\begin{aligned} [E[x^2] - 2E[x]E[x] + [E(x)]^2] &= [E[x^2] - 2(E[x])^2 + [E(x)]^2] \\ &= E[x^2] - (E[x])^2 \end{aligned}$$

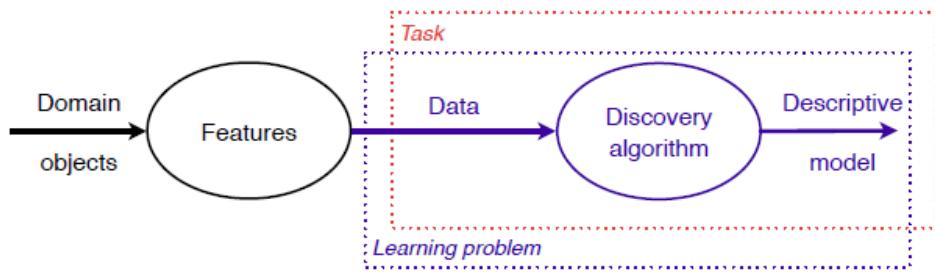


Figura 3.3: Schema riassuntivo del funzionamento del machine learning per task di tipo descrittivo, che si differenzia dallo schema già visto in Figura 1.2 per i task di tipo predittivo.

L'assenza di etichette associate ai dati cui si è fatto cenno poc'anzi è una costante nell'apprendimento non supervisionato.

3.3.1 Clustering predittivo vs descrittivo

Lo scopo del clustering è di trarre conoscenza dai dati scoprendo al loro interno gruppi omogenei nei quali vengono condivisi alcuni aspetti comuni.

Il clustering tuttavia può essere di due tipi principalmente, entrambi rientranti sotto il cappello dell'apprendimento non supervisionato:

CLUSTERING PREDITTIVO Processo di apprendimento di una funzione in grado di etichettare correttamente istanze dopo un addestramento su dati privi di etichette. Dato un insieme di nuove etichette $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ e lo spazio delle istanze \mathcal{X} , il cluster si può essere formalizzato come:

$$\hat{q} : \mathcal{X} \rightarrow \mathcal{C}$$

CLUSTERING DESCRITTIVO Processo di apprendimento di un insieme di etichette $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ a partire da un certo dataset D :

$$\hat{q} : D \rightarrow \mathcal{C}$$

È importante rimarcare come il clustering, il buon clustering, deve raggruppare i dati in gruppi coerenti, all'interno dei quali si trovino istanze più simili fra loro piuttosto che a quelle appartenenti a gruppi diversi. Proprio in quest'ottica il concetto chiave che permette di identificare cluster adeguati in base al dominio e alle esigenze è la *distanza*, in particolare *come* questa viene calcolata.

La distanza fra due istanze viene misurata sulle loro feature, e può essere una distanza euclidea, oppure una distanza di Mahalanobis che verrà affrontata in seguito. Tale misura non è limitata a feature numeriche, può essere infatti estesa ad altri tipi di dati.

Il clustering e in generale i modelli basati sulla distanza verranno affrontati in dettaglio nel Capitolo 8.

3.3.2 Subgroup discovery

Premesso che non è chiarissimo come sia finita all'interno di una sezione dedicata all'apprendimento non supervisionato, in questa sottosezione si introducono i task di *subgroup discovery* che rientrano nella famiglia degli algoritmi di apprendimento supervisionato.

Dato un dataset etichettato $\{(x, l(x))\}$, lo scopo di tali task è trovare una funzione che si definisce come:

$$\hat{g} : D \rightarrow \{\text{true}, \text{false}\}$$

tale che:

$$G = \{x \in D \mid \hat{g}(x) = \text{true}\}$$

ha una distribuzione nelle classi fortemente differente dalla popolazione del dataset originario D . L'insieme di istanze G è quindi un sottoinsieme di D e si chiama *estensione*.

Si osserva inoltre che avendo la funzione \hat{g} un codominio booleano è possibile ricondurre la ricerca di sottogruppi a problemi di classificazione binaria, permettendo quindi l'utilizzo dei classificatori binari di cui si è già ampiamente scritto.

Gli algoritmi di subgroup discovery vengono guidati da una misura di valutazione. Di queste ne esistono molte, ma la maggior parte condivide due caratteristiche:

- vengono preferiti grandi sottogruppi;
- sono solitamente simmetriche, quindi viene riportato lo stesso valore per un sottogruppo e per il suo complementare.

3.3.3 Association rule discovery

Siano b e h due insiemi di attributi o due valori. Dato un dataset D senza alcuna etichetta, lo scopo dei task di association rule discovery è trovare un insieme di *regole* della forma $\{b \rightarrow h\}$ tali che $b \cup h$ sia frequente e che h sia spesso presente dove compare b .

Per scoprire regole di associazione si rendere in primo luogo necessario identificare insiemi di istanze frequenti, a questo punto è possibile valutare tutte le regole frequenti per tali insiemi. Se ad esempio l'insieme di istanze $\{i_1, i_2, i_3\}$ è molto frequente, si possono identificare tutte le seguenti regole:

$$i_1 \rightarrow i_2, i_3$$

$$i_2 \rightarrow i_1, i_3$$

$$i_3 \rightarrow i_1, i_2$$

$$i_1, i_2 \rightarrow i_3$$

$$i_1, i_3 \rightarrow i_2$$

$$i_2, i_3 \rightarrow i_1$$

Tuttavia fra tante regole bisogna selezionare quelle migliori. Quali siano le migliori può essere deciso con una misura della confidenza che fa uso di una *support threshold*⁴:

$$\frac{\text{supp}(b \cup h)}{\text{supp}(b)}$$

Tale approccio basato sulla forza bruta porta rapidamente ad una quantità di regole enorme, un impiego di tempo esponenziale se si tiene conto che si deve calcolare il valore di *support* per ciascun elemento, identificare tutti gli insiemi di elementi frequenti e per ciascuno di questi tutte le regole possibili.

Tuttavia esistono algoritmi molto efficienti per i task di scoperta delle regole di associazione che si basano sul fatto che se un insieme di elementi non è frequente, allora non lo è nemmeno nessuno dei suoi soprainsiemi.

⁴ Tale *support threshold* può essere interpretata come una soglia sotto la quale non si considera frequente un elemento.

4

APPRENDIMENTO DI CONCETTI

In questo capitolo ci si focalizza sull'apprendimento di concetti, che sono ipotesi fatte a partire da esempi riguardanti una popolazione di elementi della realtà omogenei e descritti tramite attributi (anche chiamati *features*). L'attributo di classe è, tra questi, uno di particolare importanza ed è usato per assegnare una categoria (una classe) all'esempio.

Il nostro obiettivo è costruire una generalizzazione comune (meglio detta *concetto*) che possa descrivere tutti gli esempi di una certa categoria (classe positiva) ma che sia nel contempo abbastanza restrittiva da escludere gli elementi appartenenti alla classe negativa.

Le descrizioni degli esempi su cui si lavora sono rappresentate dalla congiunzione di predicati logici (anche chiamati *letterali*) della forma *attribute = value* o *attribute < value*, il cui significato riteniamo essere intuitivo (lo è vero?!).

L'approccio alla generalizzazione è di tipo *bottom-up*: partendo dalla descrizione di ogni esempio i si escludono mano a mano tutti i letterali in conflitto con un altro esempio j.

Si costruiscono così diverse generalizzazioni, a seconda dell'ordine in cui gli esempi vengono considerati e si terrà in considerazione solo un set di generalizzazioni, ovvero quello che rappresenta la "Least General Generalization".

Si sfrutta in genere un grafo per avere visivamente chiaro i passaggi.

4.1 UN PRIMO ESEMPIO

Il primo esempio che verrà proposto avrà come scopo la classificazione degli animali, ed in particolare il riconoscimento della classe dei Delfini.

In questo primo esempio utilizzeremo letterali formati solamente da uguaglianza. Supponiamo all'inizio di avere i seguenti attributi:

```
Length = "valore numerico" ∧ Branchie = "yes/no" ∧  
Beak = "yes/no" ∧ "Teeth = "few/many"
```

Non tutti gli esempi devono necessariamente istanziare tutti gli attributi ed è quindi possibile avere per alcuni attributi un valore indefinito.

Supponiamo di avere un set di esempi così composto:

1. Length = 3(A1) ∧ Branchie = no(B1) ∧ Beak = yes(C1) ∧ Teeth = many(D1)

2. $\text{Length} = 4 \wedge \text{Branchie} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{many}$

3. $\text{Length} = 3 \wedge \text{Branchie} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{few}$

Supponendo di sottoporre gli esempi nello stesso ordine in cui sono stati presentati, al primo passo avremo la prima generalizzazione sull'attributo "Length", che verrà rimosso, rimanendo con una descrizione di classe generica così formata:

$\text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{many}$

Il terzo esempio ci porta poi alla rimozione dell'attributo "Teeth" ed otterremo così la classe più generale composta da:

$\text{Branchie} = \text{no} \wedge \text{Beak} = \text{yes}$

Ogni esempio la cui descrizione soddisfa questa disgiunzione sarà classificato "Delfino" dal nostro sistema.

4.2 SPAZIO DELLE IPOTESI

Lo spazio delle ipotesi è rappresentato dall'insieme delle specifiche di ogni possibile esempio del set preso in considerazione.

Per quanto riguarda l'esempio visto nella sezione 4.1 possiamo ritrovare lo spazio delle ipotesi in Figura 4.1. Come si nota dal livello

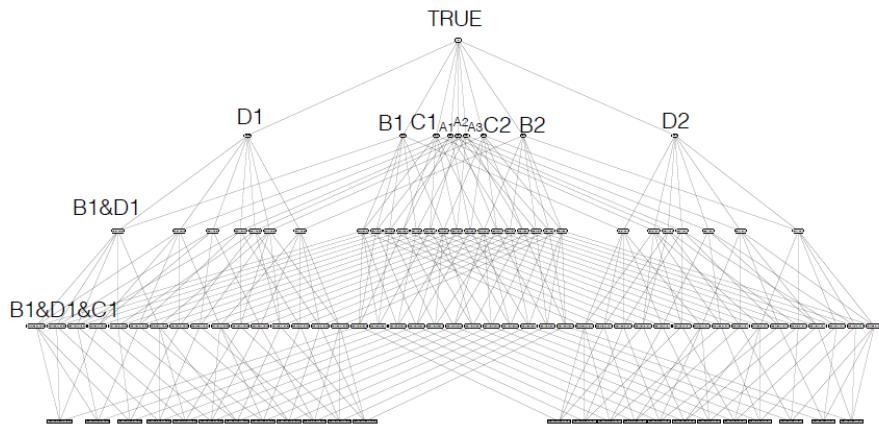


Figura 4.1: Grafo raffigurante lo spazio delle ipotesi trattato nella sezione 4.1

foglia, abbiamo in tutto 24 possibili istanze date dalla congiunzione dei 4 letterali: con 3 possibili istanze dell'attributo "Length", e 2 per ciascun altro attributo, $24 = 3 * 2 * 2 * 2$. Se consideriamo anche l'assenza di valore per gli attributi, data dalla eliminazione dello stesso a seguito di una generalizzazione, avremo $4 * 3 * 3 * 3 = 108$ possibili concetti (rappresentati dai nodi intermedi del grafo).

Quindi, lo spazio delle ipotesi è esponenziale nel numero di attributi, ma lo spazio delle *estensioni* possibili (numero di possibili insiemi di istanza che possiamo avere) è decisamente maggiore ($2^{24} >$

16 milioni). Quindi abbiamo deciso di utilizzare un linguaggio per descrivere gli esempi, con un certo numero di attributi, così da ottenere un numero finito di concetti da rappresentare, ma questo ci ha portati ad avere un numero di possibili insiemi di esempi (che è possibile vengano sottoposti al sistema per essere appresi) decisamente più grande: questo può far pensare quanto è difficile apprendere concetti a partire dai dati.

Tuttavia il fatto che l'algoritmo di apprendimento debba generalizzare può giocare a nostro favore, in quanto in questo modo potrebbe autonomamente includere nella sua generalizzazione concetti che coprono anche esempi mai visti.

Continuiamo ora l'esempio e, partendo dal grafo, cerchiamo di restringere lo spazio delle ipotesi considerando solo quei concetti che coprono gli esempi del dataset di tre esempi dato. Ci restano 32 concetti, di cui solo 3 ricoprono la classe positiva.

Ciò che viene suggerito dalla Figura 4.2 è esattamente quanto fatto nell'esempio in sezione 4.1, e poi proseguito fino all'ottenimento della generalizzazione TRUE (l'universo) che è si una generalizzazione, ma che intuitivamente include anche esempi della classe negativa.

Intuitivamente la LGG di due esempi è il concetto nello spazio delle

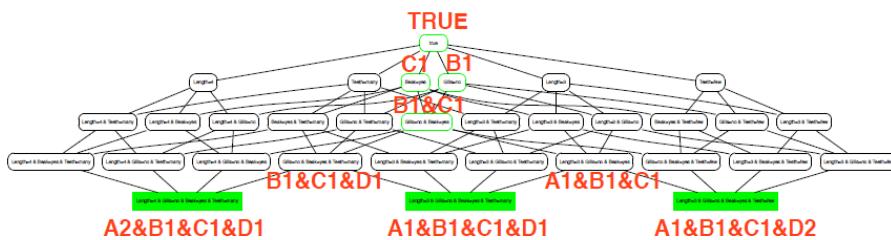


Figura 4.2: Grafo raffigurante lo spazio delle ipotesi ristretto agli esempi della classe positiva

ipotesi dove il cammino a partire da due o più esempi e verso la radice si interseca. Una proprietà utile assicura l'unicità di questo punto. Più precisamente, nella struttura formata dallo spazio delle ipotesi avremo che due elementi hanno un Least Upper Bound (**lub**) che è il minimo set di ipotesi in comune e un Greatest Lower Bound (**glb**) che è invece il massimo set di ipotesi in comune. Se consideriamo, anziché lo spazio delle ipotesi, solamente le istanze riconducibili all'interno di questi de confini, l'**LGG** è esattamente il **glb** della struttura reticolare da essi formato (4.3).

Possiamo inoltre dire che tutte le possibili generalizzazioni sono almeno generali quanto l'**LGG**, in tal senso è la generalizzazione più conservativa che il sistema può apprendere.

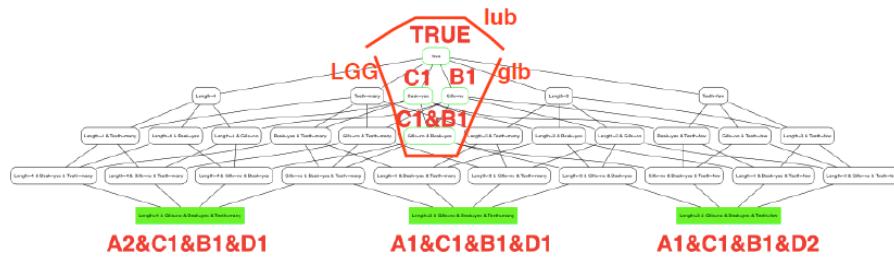


Figura 4.3: Rappresentazione di **LGG**, **lub** e **glb** in un reticolo formato dagli esempi.

4.2.1 Le relazioni d'ordine parziale

Prima di continuare con il discorso intrapreso ed arrivare a definire l'algoritmo di costruzione di una LGG daremo un paio di nozioni di background. Qualora vi sentiate particolarmente abili e non peccate di modestia avete il nostro permesso di saltare questa parte. La prima nozione riguarda una relazione d'ordine parziale. Una relazione binaria R definita sugli insiemi A e B è un sottoinsieme del prodotto cartesiano $A \times B$. Un ordine parziale R è una relazione binaria definita sugli insiemi A e B che rispecchia le seguenti proprietà:

RIFLESSIVITÀ ovvero $\forall x \in A, (x, x) \in R$;

ANTISIMMETRIA ovvero $\forall x, y \in A$, se $(x, y) \in R$ e $(y, x) \in R$ allora $x = y$;

TRANSITIVITÀ ovvero $\forall x, y, z \in A$, se $(x, y) \in R$ e $(y, z) \in R$ allora $(x, z) \in R$.

Per esempio, considerando $X = \{a, b, c, d\}$ e $A = \text{Insieme_delle_parti_di}_X = \{\{a\}, \{b\}, \{c\}, \{d\}, \{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{c, d\}, \{a, b, c\}, \{a, b, d\}, \{a, c, d\}, \{b, c, d\}, \{a, b, c, d\}\}$, si può dare una rappresentazione a grafo per A secondo la relazione di inclusione. Se ne da una rappresentazione grafica in Figura 4.4 in cui è inoltre possibile individuare **LGG**, **lub** e **glb**.

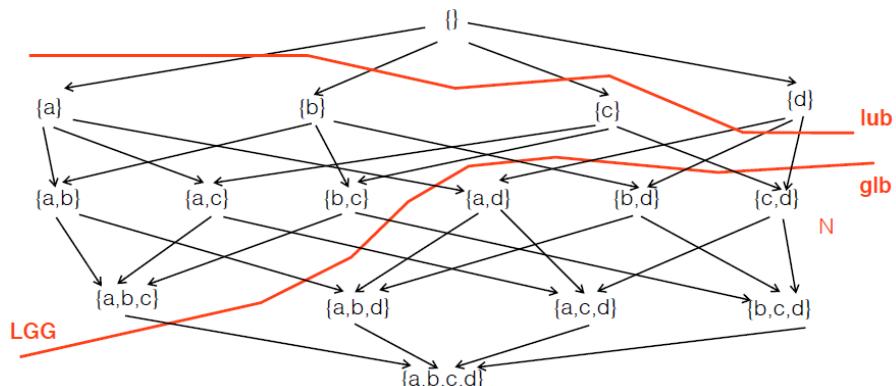


Figura 4.4: Relazione d'ordine parziale raffigurante l'inclusione di insiemi facenti parti dell'insieme delle parti di $X = \{a, b, c, d\}$.

4.2.2 Proprietà dei concetti

Se una espressione booleana A è vera per un esempio x (ovvero i letterali che descrivono x rispettano l'espressione) allora diremo che A copre x .

Un insieme di esempi coperti da A è anche chiamato "estensione di A " e si indica con χ_A dove χ indica lo spazio delle istanze considerato.

Si vanno ora ad elencare un insieme di proprietà utili:

- $\chi_{A \wedge B} = \chi_A \cap \chi_B$;
- $\chi_{A \vee B} = \chi_A \cup \chi_B$;
- $\chi_{\emptyset} = \chi$;
- se $\chi_A \supseteq \chi_B$ allora diremo che A è almeno generale quanto B ;
- l'ordine di generalità è un ordine parziale nei letterali.

4.3 ALGORITMO lgg!

Si può quindi definire l'algoritmo che, a partire da un dataset D , restituisce una espressione logica H che è esattamente l'[LGG](#).

Algorithm 1 Algoritmo per ricavare [LGG](#) da un insieme D di dati.

- 1: Input: data D .
 - 2: Output: logical expression H .
 - 3: $x :=$ first instance from D ;
 - 4: $H := x$;
 - 5: **while** instances left **do**
 - 6: $x :=$ next instance from D ;
 - 7: $H := \text{LGG}(H, x)$; // e.g., LGG-Conj or LGG-Conj-ID
 - 8: **end while**
 - 9: return H
-

Algorithm 2 Algoritmo che effettua la congiunzione tra letterali.

- Input : conjunctions x, y .
- 2: Output : conjunction z .
 - $z :=$ conjunction of all literals common to x and y ;
 - 4: return z
-

L'algoritmo comincia considerando il primo dato $x \in D$, ed essendo l'unico esempio finora considerato non potrà che asserire $H = x$. Dopo di che, per ogni iterazione, considera un nuovo dato e valuta la nuova LGG come visto in precedenza (rimuovendo letterali non coerenti se messi in congiunzione).

Terminata l'iterazione, restituisce H .

Plotkin, come tesi del suo PhD (1971) ha dimostrato che grazie alla

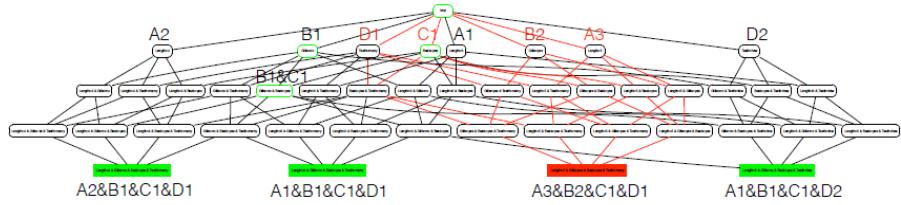


Figura 4.5: Introduzione di un esempio della classe negativa

struttura dello spazio delle ipotesi e all'unicità di [LGG](#) per una coppia di elementi dati, è garantito che la H restituita dall'algoritmo è unica e non dipende dall'ordine in cui i dati vengono considerati.

4.4 INTRODUZIONE DI ESEMPI NEGATIVI

Se i concetti accettati fino a un certo punto coprono anche qualche esempio negativo la generalizzazione (costruita come fin'ora visto) sarà inconsistente.

Dovremo intervenire in modo da rendere consistenti i concetti. Non occorre intervenire su [LGG](#), bensì (come vedremo) sul margine [lub](#). Riprendendo l'esempio introdotto nella sezione [4.1](#) in cui avevamo i seguenti esempi positivi

1. $\text{Length} = 3 \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{many} \wedge \text{D1}$
2. $\text{Length} = 4 \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{many}$
3. $\text{Length} = 3 \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{few}$

Che aveva portato ad avere la generalizzazione $\text{Gills} = \text{no} \wedge \text{Beak} = \text{yes}$.

Viene ora aggiunto il seguente esempio:

1. $\text{Length} = 5 \wedge \text{Gills} = \text{yes} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{many}$

Ci si accorge subito che il letterale $\text{Beak} = \text{yes}$ (da solo) non rende più accettabili alcune generalizzazioni prima possibili, e in particolare porta alla eliminazione della generalizzazione più alta TRUE.

Come accennato occorre aggiornare l'upper border "abbassandolo", ovvero specificando maggiormente lo spazio delle ipotesi fermo restando l'accettazione degli esempi che possono risultare ancora validi. Questo viene fatto attraverso la procedura di ricerca della *specializzazione minima di lub*, con cui si aggiungono delle clausole che permettono di non coprire gli esempi negativi. Nella Figura 4.5 si rappresenta in rosso l'esempio negativo, i percorsi in rosso rappresentano cammini che portano all'esempio negativo e che quindi non sono più corretti.

Quindi solamente tre congiunzioni coprono tutti gli esempi positivi e nessun esempio negativo: $\text{Gills} = \text{no} \wedge \text{Beak} = \text{yes}$ e $\text{Gills} = \text{no}$. È

ora utile introdurre nel linguaggio quella che Flach chiamò *disgiunzione interna*, che introduce la possibilità di attribuire una lista di valori ad una *feature*, chiedendo così che quest'ultima assuma un valore tra quelli possibili. Con riferimento al solito esempio, ed in particolare alle istanze 3 e 4, si potrà così ricavare la nuova clausola:

$$\text{Length} = [3, 4] \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes} \wedge \text{Teeth} = \text{many}$$

che, considerato che la feature Teeth potrà assumere qualsiasi valore del suo dominio (gli esempi positivi potranno avere sia Teeth = few e Teeth = many ed è quindi inutile usare la disgiunzione interna) si avrà:

$$\text{Length} = [3, 4] \wedge \text{Gills} = \text{no} \wedge \text{Beak} = \text{yes}$$

Si nota come l'esempio negativo non è coperto da tale congiunzione. Generalizzando potremmo mantenere $\text{Length} = [3, 4] \wedge \text{Gills} = \text{no}$, perché coprirebbe gli esempi positivi ma escluderebbe correttamente l'esempio negativo.

Guardando lo spazio delle ipotesi (Figura 4.5) ci si accorge che esistono ipotesi interne tra lub e glb, vedasi Figura 4.6, che sono comunque accettabili. Questo suggerisce che questa teoria è formata da diverse ipotesi, tutte valide e alternative fra di loro, contenute nello spazio delle ipotesi dei dati.

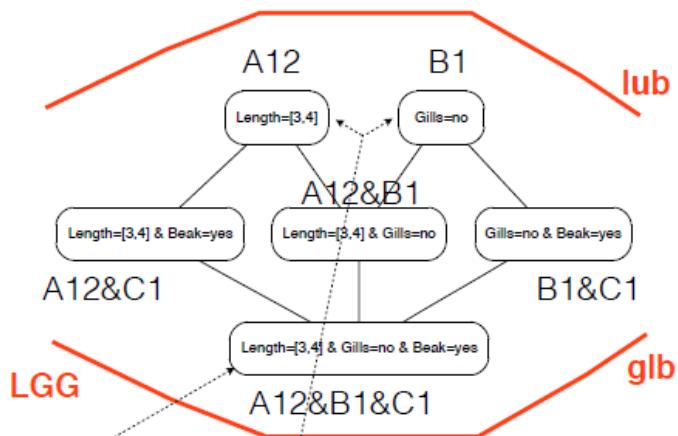


Figura 4.6: Version space dell'esempio considerato.

Tale insieme di ipotesi è detto *version space* ed è un insieme *convesso*: si può descrivere una qualsiasi ipotesi in essa considerando le due ipotesi agli estremi (la più specifica e la più generale). Vediamo ora la procedura che costruisce LGG quando si utilizza la disgiunzione interna.

4.5 PATHS THROUGH THE HYPOTHESIS SPACE

Si introdurrà ora un po' di notazione, facendo anche un ripasso di quanto visto in precedenza.

Algorithm 3 Find least general conjunctive generalisation of two conjunctions, employing internal disjunction.

```

1: Input: conjunctions x,y;
2: Output: conjunction z;
3: z:= true;
4: for each feature f do
5:   if f= $v_x$  is a conjunct in x and f =  $v_y$  is a conjunct in y then add
   f = Combine-ID( $v_x, v_y$ ) to z
6:   end if
7: end for
8: return z
9:           ▷ Combine-ID([set-values-x],[set-values-y])=[set-values-x
   UNION set-values-y]

```

Un concetto è *completo* se copre tutti gli esempi positivi.

Un concetto è *consistente* se non copre nessun esempio negativo.

Il *version space* è un insieme di tutti i concetti completi e consistenti.

Quest'ultimo insieme è convesso¹ e pienamente definito entro **lub** e **glb**.

Vediamo un esempio. A partire dal grafo, si sa essere possibile salire dal **glb** mantenendo la validità dei concetti e incrementando la generalità.

Si seguirà ora la rappresentazione della Figura 4.7 dandone una didascalia esaustiva.

All'inizio si pone A, che è la descrizione completa dell'esempio p₁ che quindi ricopre alla perfezione. Salendo di livello si trova l'esempio p₂ e il concetto B, ottenuto tramite l'applicazione di disgiunzione interna sulla feature Length e che quindi copre gli esempi p₁ e p₂. Arrivando al concetto C, si raggiunge il cosiddetto *Roc Heaven*, in quanto questo concetto riesce a coprire pienamente gli esempi positivi senza coprire alcun esempio negativo.

È possibile generalizzare ulteriormente escludendo (cioè considerando dato) l'attributo Gills, raggiungendo così un'altra ipotesi egualmente valida.

Se si generalizza ancora (lasciando solamente l'attributo Beak = yes) si otterrebbe però un concetto che copre anche l'esempio negativo precedentemente introdotto.

Tracciando un percorso attraverso lo spazio delle ipotesi si avrà una curva corrispondente alla curva di copertura: se un percorso (nel nostro esempio dato da A-B-C-D) include elementi del *version space* (completi e consistenti) allora corrisponde alla curva di copertura passante per il *Roc Heaven* e quindi è un percorso ottimale.

¹ Dati due membri (H_1, H_2) dell'intero insieme delle ipotesi (insieme convesso), ciascun membro H_x dell'insieme convesso che è meno generico di H_1 ($H_1 \text{ preccurlyeq } H_x$) e più generale i H_2 appartiene anch'esso all'insieme convesso.

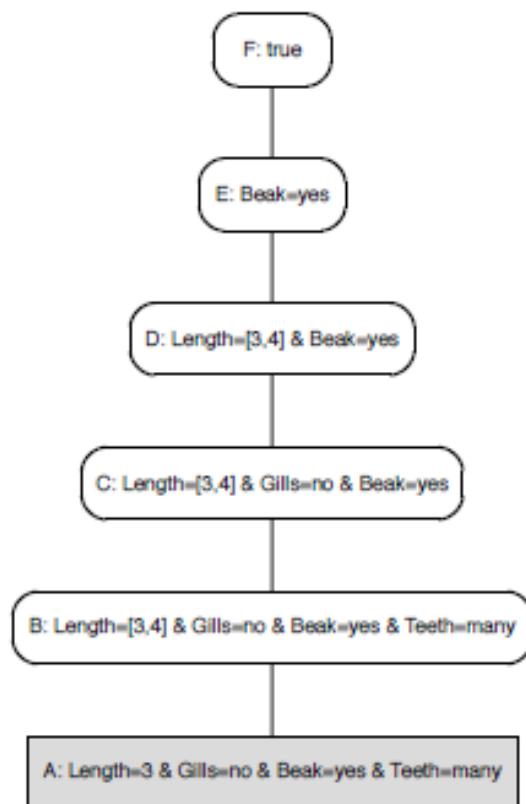


Figura 4.7: Esempio di generalizzazione basato sul version set

L'obiettivo del concept learning può essere definito come la ricerca dell'insieme dei concetti che permettono di tracciare un percorso passante per il *Roc Heaven*, percorso oltre il quale sconfiniamo dal version space.

4.6 DATI NON CONGIUNTAMENTE SEPARABILI

Questa procedura non sempre funziona: talvolta può capitare di avere dati non congiuntamente separabili. In questo caso non si sarebbe in grado (usando il linguaggio così come definito con la sola congiunzione logica e la disgiunzione interna) di definire ipotesi valide (complete e consistenti): il version space risulterebbe così vuoto. Un caso di questo tipo è proprio l'esempio sotto riportato, in cui si hanno 5 esempi positivi e altrettanti negativi:

P1 Length = 3 \wedge Gills = no \wedge Beak = yes \wedge Teeth = many

P2 Length = 4 \wedge Gills = no \wedge Beak = yes \wedge Teeth = many

P3 Length = 3 \wedge Gills = no \wedge Beak = yes \wedge Teeth = few

P4 Length = 5 \wedge Gills = no \wedge Beak = yes \wedge Teeth = many

P5 Length = 5 \wedge Gills = no \wedge Beak = yes \wedge Teeth = few

N1 Length = 5 \wedge Gills = yes \wedge Beak = yes \wedge Teeth = many

N2 Length = 4 \wedge Gills = yes \wedge Beak = yes \wedge Teeth = many

N3 Length = 5 \wedge Gills = yes \wedge Beak = no \wedge Teeth = many

N4 Length = 4 \wedge Gills = yes \wedge Beak = no \wedge Teeth = many

N5 Length = 4 \wedge Gills = no \wedge Beak = yes \wedge Teeth = few

Viene quindi introdotto l'Algoritmo 4 che ha lo scopo di ottenere una *Most General Consistent Specialisation*.

Come ricavare la specializzazione C' di C? Nell'algoritmo non è specificato ma una opzione iniziale potrebbe essere quella di cominciare da un singolo letterale di C (con il vincolo di appartenere alla classe positiva, altrimenti si rappresenterebbe la classe negativa).

Un modo per ottenere ciò è considerare l'[LGG](#), il "minimal border" che copre tutti gli esempi positivi e nessun esempio negativo. Ovviamente non si è certi che [LGG](#) sia consistente con gli esempi negativi: per questo, come si vede, si richiama ricorsivamente la procedura.

4.6.1 Esempio di applicazione dell'algoritmo MG Consistent

Se si considera ora l'esempio in figura 4.8 in cui si denota un percorso nello spazio delle ipotesi, si nota che il concetto A copre solamente

Algorithm 4 Find most general consistent specialisations of a concept.

Input: current concept C to update (upper border); negative examples N

Output: set of concepts S

if C doesn't cover any element from N **then**

return C

end if

$S \leftarrow \emptyset$

for each minimal specialisation C' of C **do**

$S \leftarrow S \cup \text{MGConsistent}(C', N)$

end for

return S \triangleright Add a conjunct or eliminate a value in a disjunction; in particular the conjunct could be taken from the clauses of the LGG (low border)

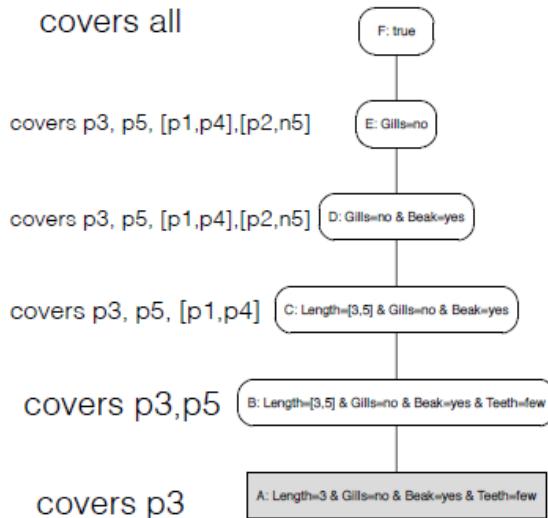


Figura 4.8: Esempio di applicazione dell’Algoritmo 4.

l’esempio p3, B copre l’esempio p5, C copre tutti gli esempi positivi tranne p2 e D ed E che sono LGG di tutti e 5 gli esempi positivi ma che coprono anche l’esempio negativo n5.

Soffrendosi però sui concetti D ed E, si vuole evidenziare come questi due concetti occupino lo stesso punto nella curva di copertura: generalizzando D in E (rimuovendo cioè Beak = yes) la curva non cambia e questo significa che nel contesto del concetto E la condizione Beak = yes è in realtà implicita (perché segue da Gills = no e quindi ogni volta che abbiamo quest’ultima avremo anche la prima).

Un concetto che include implicitamente delle condizioni è detto concetto chiuso. Tuttavia, tutto ciò non significa che D ed E sono logicamente equivalenti: esistono (probabilmente) istanze nel dominio che sono coperte da E ma non da D, ma queste istanze non sono

presenti nel nostro insieme di dati. Non potendo prendere in considerazioni queste istanze, D ed E sono per noi indistinguibili.

Oltre quanto detto, il libro e le slide si soffermano parecchio sulla differenza di questi due punti, ma noi sorvoleremo².

In generale nei casi come quello dell'esempio che stiamo portando avanti in cui non siamo in grado di raggiungere il Roc Heaven perché non esiste modo di trovare un concetto che sia contemporaneamente completo e consistente, ci si attiene a considerare il concetto più generale, rinunciando alla consistenza ma garantendo la completezza se si ritiene che questa assunzione porti nel peggiore dei casi a danni minore e viceversa (meglio un falso positivo che un true negative).

4.7 CLAUSOLE DI HORN

È stato finora utilizzato un certo tipo di linguaggio definito all'inizio del Capitolo. Si introduce ora una nuova componente: le *clausole di Horn*.

Una *clausola di Horn* è un'implicazione della forma $A \rightarrow B$ dove A è una congiunzione di letterali e B è un letterale (così come definito e utilizzato finora).

La clausola $A \rightarrow B$ equivale a $\neg A \vee B$. Da qui in poi si tratteranno espressioni logiche in *Forma Normale Congiuntiva* che altro non sono che rappresentate dalla congiunzione di letterali³.

Già, ma cosa cambia? Quanto è stato fatto finora è, a partire da un insieme di esempi, apprendere una teoria che permette di coprire tutti gli esempi positivi: se un esempio positivo non veniva coperto dalla teoria attuale occorreva generalizzarla per includerlo mentre, qualora la teoria includeva un esempio della classe negativa, occorreva escluderlo specializzando l'ipotesi.

Ora, con le clausole di Horn, si hanno più opzioni:

- nel caso in cui un esempio non sia coperto dalla teoria attuale (ora descritta tramite la congiunzione di letterali) possiamo:
 - applicare la generalizzazione (come appena ricordato);
 - eliminare le clausole che violano l'esempio (ad esempio una clausola del tipo $A \rightarrow B$ con A è vera e B è falsa);
- nel caso in cui un esempio negativo sia erroneamente incluso nella teoria possiamo:
 - effettuare una specializzazione tramite l'aggiunta di più clausole (come appena ricordato);
 - supponendo che l'esempio negativo sia coperto da un concetto del tipo $A \wedge \neg B$, si aggiunge all'insieme delle mo-

² Speriamo anche in sede di esame

³ Nel caso in esame si tratta di disgiunzioni ottenute a partire da implicazioni, ovvero le clausole di Horn.

st general conjunctive hypothesis la clausola opposta $\neg(A \wedge \neg B) = (\text{DeMorgan})(\neg A \vee B) = (\text{Tavolediverità})(A \leftarrow B)$. (l'algoritmo che vedremo a breve farà uso di questa osservazione);

- supponendo che l'esempio sia coperto dal concetto $A \wedge B$, si aggiunge all'insieme delle *most general conjunctive hypothesis* la clausola opposta $\neg(A \wedge B) = (\text{DeMorgan})(\neg A \vee \neg B) = (\text{Tavolediverità})(A \leftarrow \neg B)$.

Si introduce quindi il nuovo Algoritmo 5 che fa uso delle clausole di Horn per ottenere concetti che includono tutti gli esempi positivi e che escludono tutti gli esempi negativi.

Nella situazione in cui opera, questo algoritmo apprende una teoria h basata sugli esempi della classe positiva. Il training set contiene però esempi di entrambe le classi: accade spesso nel machine learning che si ricorra a conoscenza pregressa⁴, che assumiamo essere il nuovo background di verità. Si assume quindi di avere già una teoria valida per la classe positiva, e si indicherà con l'espressione booleana f . Inoltre, nell'algoritmo avremo si hanno due oracoli, ovvero una "presenza" in grado di rispondere a delle questioni in modo corretto e in un tempo finito:

- il primo è M_b , che risponde alla classificazione di un esempio sulla base della teoria h costruita.
- il secondo è Eq , che risponde "true" qualora la teoria h è logicamente equivalente all'espressione booleana f .

L'algoritmo utilizza una operazione di intersezione tra un esempio x e un altro esempio $s \in S$ (insieme degli esempi negativi). Questa operazione ritorna due insiemi:

- un insieme di verità composto da tutti i letterali che sono verificati sia per x che per s .
- un insieme di falsità composto da tutti i letterali che sono falsi per x o per s

La terminazione dell'algoritmo è garantita in un tempo polinomiale nel numero di features e nel numero di clausole.

Si nota inoltre che l'algoritmo è in accordo con quello che si definisce *PAC-learning model*, ovvero un modello appreso dagli esempi che sappiamo essere probabilmente corretto (con un certo livello di fiducia) data la distribuzione dei valori nel dominio.

⁴ Questo ramo del machine learning rientra nel campo dell'*Active Learning*, in cui si assume esistere una "entità esterna" che interagisce con l'algoritmo al fine di modificare il training set

Algorithm 5 Learn a conjunction of Horn clauses from membership and equivalence oracles.

```

1: Input: equivalence oracle Eq; membership oracle Mb.
2: Output: Horn theory h equivalent to target formula f.
3:  $h \rightarrow \text{true}$ ;            $\triangleright$  conjunction of Horn clauses, initially empty
4:  $S \rightarrow \emptyset$ ;           $\triangleright$  a list of negative examples, initially empty
5: while Eq(h) returns counter-example x do
6:   if x violates at least one clause of h then  $\triangleright$  x is a false negative
7:     generalize h by removing every clause that x violates
8:   else                                 $\triangleright$  x is a false positive
9:     find the first negative example  $s \in S$  such that  $z = s \cap$ 
     $x$  has fewer true literals than, and  $Mb(z)$  labels it as a negative;
10:    if such an example exists then replace s in S with z, else
      append x to the end of S;
11:     $h \rightarrow \text{true}$ 
12:    for all  $s \in S$  do                   $\triangleright$  rebuild h from S
13:       $p \rightarrow$  the conjunction of literals true in s;
14:       $Q \rightarrow$  the set of literals false in s;
15:      for all  $q \in Q$  do
16:         $h \rightarrow h \wedge (p \leftarrow q);$ 
17:      end for
18:    end for
19:  end if
20: end while
21: return h

```

4.8 ILP - INDUCTIVE LOGIC PROGRAMMING

Per anni (circa gli ultimi 20) si è considerato inefficiente (soprattutto per le grosse moli di dati) l'approccio logico alla programmazione, tanto che la professoressa che tiene questo corso era quasi decisa a non trattare più questa parte di programma.

Se non fosse che, per la fortuna della nostra conoscenza, partecipò ad una conferenza in cui un ricercatore italiano presso la California University in Database e Machine Learning he risponde al nome di Carlo Zaniolo, presentò la sua ricerca sulla sorprendente potenza di linguaggi come *Datalog* e *Prolog* nel gestire algoritmi di Machine Learning. Questo nuovo approccio risponde al nome di *Inductive Logic Programming*.

Mentre nel linguaggio visto prima si utilizzano solamente proposizioni booleane per esprimere i concetti, nel linguaggio logico si utilizzano predicati della logica del prim'ordine che permettono di esprimere letterali più complessi utilizzando termini composti anche da simboli funzionali e da costanti. I vantaggi introdotti saranno:

- poter utilizzare singoli termini per riferirsi agli oggetti;
- poter descrivere la struttura di un oggetto usando i predicati;
- poter introdurre variabili per riferirsi ad oggetti non specifici. Ad esempio, a partire da `BodyPart(x, PairOf(Gill))` si può ricavare quello che prima erano le operazioni di generalizzazione e specializzazione e che ora chiameremo:

`ANTI-UNIFICATION BodyPart(x, PairOf(y))`

`UNIFICATION BodyPart(Dolphin42, PairOf(Gill))`

Questo approccio permette quindi di utilizzare ad un livello più alto un linguaggio che fa uso della logica del prim'ordine (che è, come appena visto, più completo e complesso), e ad un livello più basso un linguaggio proposizionale semplice in grado di rispondere in tempi molto veloci alle query che gli vengono sottoposte.

4.9 RIDURRE L'INSIEME DI ESEMPI

Si vuole, in buona sostanza, avere un modo di escludere alcuni esempi fermo restando la correttezza di ciò che si intende apprendere e migliorando addirittura la conoscenza?

La risposta è ovviamente affermativa, e in parte è quello che già si fa nell'Algoritmo 5.

Supponendo di avere i seguenti esempi:

`M ManyTeeth ∧ ¬Gills ∧ ¬Short ∧ ¬Beak;`

`G ¬ManyTeeth ∧ Gills ∧ ¬Short ∧ ¬Beak;`

s \neg ManyTeeth \wedge \neg Gills \wedge Short \wedge \neg Beak;

b \neg ManyTeeth \wedge \neg Gills \wedge Short \wedge Beak.

è chiaro che avremo 16 possibili sottoinsiemi di {m, g, s, b} e ciascuno di essi può essere rappresentato da un concetto: per ogni esempio che vogliamo escludere, aggiungiamo al concetto (tramite congiunzione) la negazione.

5

TREE MODELS

In questo capitolo si analizzerà un metodo per rimodellare la conoscenza attraverso i modelli ad albero. In particolare si ripercorrerà quanto è stato già introdotto nel capitolo 1 in cui si affrontavano i problemi di classificazione, ranking, regressione e clustering tramite la costruzione di un albero (cosiddetti *Tree models*).

5.1 INTRODUZIONE AI FEATURE TREE

Il grande vantaggio di questa struttura è il loro essere nativamente auto-descrittivi e comunque facili da capire (ecco che allora si avrà il beneficio di poter predire un certo valore, con la spiegazione esplicita del perché si possa farlo).

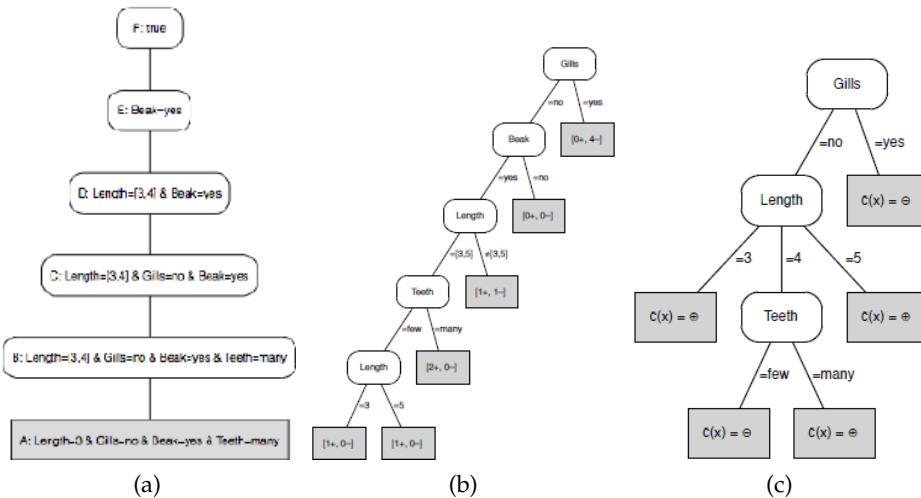


Figura 5.1: Rappresentazioni ad albero dell'esempio introdotto nella sezione 4.1

Focalizzandosi sul path in Figura 5.1a, che è quello già visto in precedenza, in cui nella foglia si trova l'esempio più specifico (che ricopre cioè la singola istanza) e alla radice si trova il concetto più generale: questo albero è equivalente a quelli delle Figure 5.1b e 5.1c.

Nelle foglie dell'albero centrale si trova una statistica dei valori che rispecchiano quanto stabilito nei nodi precedenti (gli antenati rappresentano dei "filtri" per questa classificazione). In questo tipo di albero la stessa feature può apparire più volte, in più nodi. Intuitivamente se abbiamo n feature si potrà rappresentare ciascun esempio in uno spazio n -dimensionale. Inoltre, queste n feature, segmentano lo spazio in

n partizioni e partizionando una di questa tramite un'altra feature si otterranno altre sotto-partizioni a loro volta filtrabili.

Si nota che il problema in esempio è binario ma l'albero è facilmente generalizzabile al caso con k classi. Si evidenzia inoltre il fatto che questi due alberi introdotti hanno esattamente le stesse capacità di classificazione (se siete come San Tommaso, vi basterà fare qualche prova), ma utilizzano metodologie differenti di descrizione delle partizioni del dataset.

Nell'albero a destra troviamo un vero *decision tree* appreso sugli stessi esempi degli altri alberi. Quest'ultimo separa gli esempi positivi e quelli negativi, generando diverse foglie a seconda delle etichette imposte.

Si supponga di avere i seguenti esempi:

- Esempi positivi (p):
 1. Length = 3 \wedge Gills = no \wedge Beak = yes \wedge Teeth = many
 2. Length = 4 \wedge Gills = no \wedge Beak = yes \wedge Teeth = many
 3. Length = 3 \wedge Gills = no \wedge Beak = yes \wedge Teeth = few
 4. Length = 5 \wedge Gills = no \wedge Beak = yes \wedge Teeth = many
 5. Length = 5 \wedge Gills = no \wedge Beak = yes \wedge Teeth = few
- Esempi negativi (n):
 1. Length = 5 \wedge Gills = yes \wedge Beak = yes \wedge Teeth = many
 2. Length = 4 \wedge Gills = yes \wedge Beak = yes \wedge Teeth = many
 3. Length = 5 \wedge Gills = yes \wedge Beak = no \wedge Teeth = many
 4. Length = 4 \wedge Gills = yes \wedge Beak = no \wedge Teeth = many
 5. Length = 4 \wedge Gills = no \wedge Beak = yes \wedge Teeth = few

Quello che si vuole ottenere è, a partire da un feature tree¹ che è il modello rappresentato nella seconda Figura, un decision tree (che è il modello rappresentato nella terza figura).

Per prima cosa, proprio nell'albero della Figura 5.1b si nota che nelle ultime tre foglie non esistono esempi della classe negativa e ciò ci permette di raggruppare le partizioni. Si stabilisce allora una procedura che permette di abbassare l'altezza del dato: se si vuole rappresentare l' i -esimo concetto si può tagliare le i foglie (da sinistra) e unirle in un unico nodo.

Si può vedere questo passaggio nelle Figure 5.2a e 5.2b: in questo caso l'abilità di classificazione dell'albero rimane invariata.

Un decision tree si ottiene a partire da un feature tree potato (Figura 5.2c), ad esempio per diminuire i tempi computazionali e/o per evitare il problema dell'overfitting, assegnando alle foglie delle etichette e a ciascun arco un valore. Le etichette assegnate inducono

¹ termine generale per indicare un qualsiasi albero di quelli presentati in precedenza

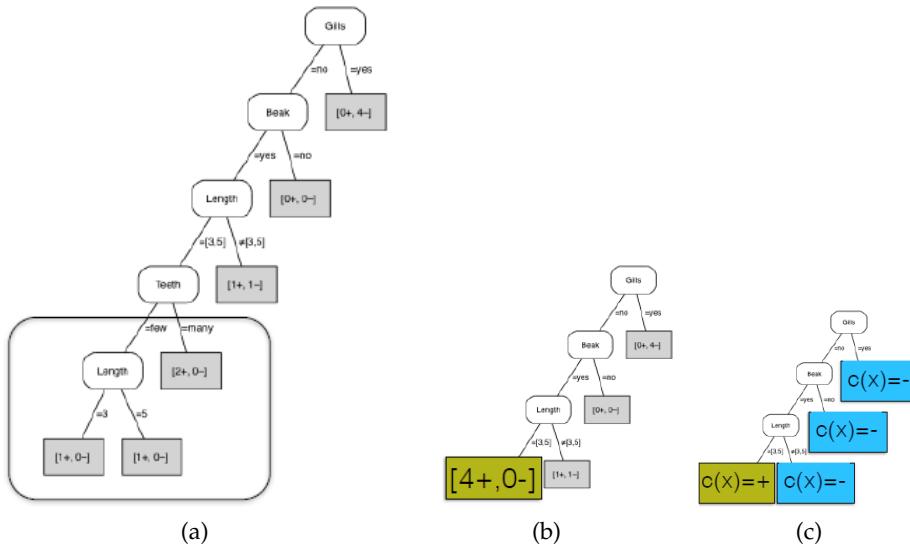


Figura 5.2: Passaggio di pruning di un feature tree e relativo decision tree

alla decisione: scegliendo la classificazione proposta dall’etichetta si commette il minor errore possibile.

A partire da un feature tree è possibile ottenere inoltre diverse espressioni logiche fra loro equivalenti, verificabili ad esempio applicando le leggi di De Morgan, e ottenibili percorrendo l’albero dalla radice alle foglie rappresentanti il concetto voluto. Dall’albero dell’esempio si potrebbe ottenere la seguente espressione:

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

E a partire da una rappresentazione de concetto negativo ($(\text{Gills} = \text{no} \wedge \text{Length} = 4 \wedge \text{Teeth} = \text{few}) \vee (\text{Gills} = \text{yes})$) è possibile negarlo per ottenere ulteriore informazione positiva sull’albero.

Come si può quindi confrontare l’abilità descrittiva di un concetto espresso in forma congiuntiva con un decision tree?

Si può dire che i decision tree sono più espressivi, sono quindi migliori nella predizione di esempi del dataset, ed è possibile convincersene pensando ad un caso estremo: in un dataset composto da n esempi, rappresentati ciascuno con una forma congiuntiva si avranno n differenti congiunzioni e ciascuna risulta essere rappresentabile con uno specifico percorso e l’albero risultante ricoprirebbe in modo perfetto il dataset, rischiando però overfitting e la sua capacità di generalizzazione non sarebbe così buona.

Infatti ogni decision tree corrisponde ad una espressione logica scritta sotto forma normale disgiuntiva (DNF), ed è così capace di ricoprire gli esempi che, se non limitati, porterebbero ad un l’albero che non generalizza adeguatamente.

Un modo per evitare questo problema è di introdurre un bias oppure, in qualche modo, limitare la possibilità di ricoprire in modo co-

sì esatto gli esempi, pur mantenendo una buona generalizzazione (si tratta di utilizzare modelli più semplici, meno complessi). Il principio che si introduce è basato sul fatto che è il linguaggio (di modellazione usato per rappresentare i concetti) a fare la differenza: se è troppo potente, ci porterà a descrivere troppi dettagli delle ipotesi.

Una minor complessità è definita con il termine *inductive bias* e gli algoritmi di apprendimento lo impongono attraverso due approcci:

- il modo in cui avviene la ricerca nello spazio delle ipotesi;
- introducendo una funzione obiettivo che lavora su una penalità correlata alla complessità di ogni ipotesi (in questo modo si preferisce un certo modello piuttosto che un altro).

Non abbiamo però finora dato una definizione di feature tree. Si fa ora ripercorrendo formalmente quanto appena visto: un feature tree è un albero in cui i nodi interni sono etichettati con feature e ogni arco è etichettato con un letterale. L'insieme dei letterali dell'arco di un nodo è detto *split* (perchè produce appunto una suddivisione nello spazio delle ipotesi tramite una comparazione). Ciascuna foglia dell'albero rappresenta una espressione logica, che è una congiunzione di letterali ordinati e associati al percorso dalla radice dell'albero alla foglia. L'estensione di questa congiunzione (l'insieme delle istanze coperte) è chiamato *segmento di spazio delle istanze* associato alla foglia. È proprio sulle foglie che si lavora per effettuare regression, classificazione, ranking, etc...

Quindi, riassumendo:

- un feature tree è un modo compatto per rappresentare un largo numero di concetti (in forma congiunta) nello spazio delle ipotesi (diversi "percorsi", a.k.a. espressioni logiche);
- l'apprendimento si basa sulla decisione di quali possibili concetti sono i migliori per risolvere un determinato task;
- si vedrà nel Capitolo 6 i modelli a regole, che condividono molto con l'apprendimento dei concetti e con l'apprendimento di alberi. In essi i modelli vengono descritti sempre attraverso espressioni logiche congiuntive ma gli algoritmi eseguono diversamente, apprendendo una regola diversa alla volta percorrendo diversi percorsi allo stesso tempo (mentre nell'algoritmo per il modelli ad albero si apprendono più percorsi alla volta, eseguendo una ricerca top-down con un approccio divide et impera, in cui si parte apprendendo la condizione della radice e poi si divide lo spazio delle ipotesi fino alle foglie).

Vediamo quindi lo pseudocodice di un algoritmo ricorsivo (vedasi Algoritmo 6) di apprendimento per gli alberi che può essere adattato in base a diverse specifiche situazioni (differenti task), definendo accuratamente le sotto-procedure.

Algorithm 6 Growing a feature tree

```

1: Input: dataset D; set of features F
2: Output: feature tree T with labelled leaves
3: if Homogeneous(D) then return Label(D)  $\triangleright$  l'assegnazione della
   label dipende dal task (classificazione, ranking ecc)
4: end if
5:  $S \leftarrow \text{BestSplit}(D, F)$ ;  $\triangleright$  e.g., BestSplit-Class 7
6: split D into subsets  $D_i$  according to the literals in S  $\triangleright$  Il numero
   di i dipende dal numero di diramazioni dell'albero
7: for each i do
8:   if  $D_i \neq \emptyset$  then
9:      $T_i \leftarrow (D_j, F)$ ;
10:    else  $T_i$  is a leaf labelled with Label(D)
11:   end if
12: end for
13: return a tree whose root is labelled with S and whose children
    are  $T_i$ 

```

Alla riga 3 si controlla che l'insieme dei dati D sia omogeneo: questa proprietà dipende ovviamente dal task e dal dominio e occorre specificarla disponendo di conoscenza intrinseca. Ad esempio, nel caso della classificazione un insieme di dati possono essere considerati omogenei se appartengono tutti alla stessa classe (potremmo inoltre implementare questo controllo in modo da considerarlo superato qualora la maggior parte dei dati siano omogenei, e scegliere con Label l'etichetta della classe predominante, eventualmente random se due o più si equivalgono), mentre in un problema di clustering lo saranno se sono tutti "vicini".

Se sono omogenei, allora possiamo tranquillamente etichettare i dati D (se ad esempio si tratta di classificazione si assegnerà ad ogni dato D l'etichetta corrispondente alla classe più probabile).

Quindi la procedura prosegue alla riga 5 solo se i dati non erano omogenei. In tal caso si cercherà la miglior suddivisione per il dataset in quel dato istante (prendendo in input i dati stessi e l'insieme di feature F). La suddivisione di D in più sottoinsieme deve essere in accordo con lo split S (si ritrova qui l'approccio divide et impera, in quanto si divide l'insieme dei dati in sottoinsiemi costruendo un sottoalbero per ogni sottoinsieme).

Poi, per ogni partizione, si verifica se questa è vuota: se non lo è, significa che è stato trovato un nuovo ramo dell'albero su cui possiamo ora chiamare ricorsivamente la procedura, altrimenti si giudica come foglia il nuovo elemento trovato e lo si etichetta con la procedura apposita.

Al termine di ogni sottoprocedura si avrà un albero con radice S (la "ragione" dello split) e con tutti gli elementi T_i al fondo.

Si nota che l'approccio che si intrapende con questo algoritmo è *greed*-

dy: ad ogni chiamata cerca la "miglior alternativa" e non la riconsidera mai nelle chiamate successive. Questo implica che l'albero trovato dall'algoritmo non sia ottimale: si potrebbe pensare a una sorta di ricerca con backtracking, sapendo però di andare ad intaccare i tempi computazionali e lo spazio di memoria.

Le righe 3 e 13 garantiscono la terminazione della ricorsione.

5.2 DECISION TREE

5.2.1 La purezza di uno split

Dopo aver visto la procedura per la costruzione di un feature tree introduciamo la notazione utile a trattare ciò che è più sensibile alla nostra materia: i decision tree.

Supponendo quindi di dover trattare features booleane che separano l'insieme delle istanze D in due sottoinsiemi, rispettivamente D_1 e D_2 , e per esse di avere inoltre esattamente due classi (classe positiva e classe negativa).

Diremo allora che la bontà della suddivisione di D è data dalla *purezza* di D_1 e D_2 : la purezza di una partizione di n esempi dipende solo dalla quantità dei valori della classe (in questo n^- e n^+) che vi sono contenute. Questa misura può quindi essere definita come:

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

Questo è il rapporto tra il numero di esempi positivi e il numero di esempi totale: altro non è che l'*empirical probability* introdotta nel capitolo 1, cioè la probabilità che induce alla scelta di una classe piuttosto che l'altra (più alta è la probabilità empirica, maggiore sarà la purezza dello split²).

Oltre alla probabilità empirica esistono altre misure per la (misura inversa alla) purezza:

CLASSE MINORITARIA generalizzata come $1 - \max(\dot{p})$ e specificata per i problemi binari (con solo due classi) con la formula $\min\{\dot{p}, 1 - \dot{p}\}$. Considerando il valor minimo tra la probabilità empirica delle due classi si considera il valore associato alla classe meno pura;

GINI INDEX ³ dato dalle formule $\sum_i \dot{p}_i(1 - \dot{p}_i)$ nel caso generico e $2\dot{p}(1 - \dot{p})$ nel caso binario. Si nota che nel caso di variabili Bernoulliane questo indice corrisponde alla varianza, ma nel caso della variabili booleana corrisponderà esattamente all'errore atteso se la predizione è random. Questo perchè se la predizione

² Ovviamente l'interpretazione della purezza non cambia considerando n^- al numeratore al posto di n^+ .

³ Il nome è dato in onore del matematico italiano Gini che propose questo indice.

sulla classe positiva è casuale, la probabilità che la predizione sia positiva sarà p (e $1 - p$ che sia negativa). Quindi la probabilità di un falso positivo sarà $p(1 - p)$, cioè dalla moltiplicazione delle probabilità dei due eventi indipendenti fra loro e quella di un falso negativo sarà per le stesse ragioni $(1 - p)p$. Sommando si ottiene esattamente la formula data;

ENTROPIA proposta da Shannon negli anni '40, si tratta di una misura che rappresenta la quantità aspettata di informazione contenuta in un messaggio trasmesso a seguito del verificarsi di un certo evento (siamo quindi nell'ambito di una comunicazione di messaggi in cui si comunica un determinato evento con una probabilità di successo p). La formula è data, nel caso binario, dalla formula $-p \cdot \log_2(p) - (1 - p) \cdot \log_2(1 - p)$ che generalizzando darà $-\sum_i p_i \cdot \log_2(p_i)$. Il caso di utilizzo più semplice è quello in cui un singolo bit viene trasmesso settato a 1 se l'evento è avvenuto e a 0 altrimenti, ma come Shannon ha chiarito è utile considerare le due possibilità equiparabili in quanto se si è (per lo più) certi dell'avvenimento dell'evento ci sarà un elevato spreco (è importante quindi considerare l'*outcome*).

Indicando con $\text{Imp}(D_j)$ l'impurità di una foglia D_j , l'impurità di un insieme di l foglie D_1, \dots, D_l mutualmente esclusive (quindi $D = D_1 \cup D_2 \cup \dots \cup D_l$) è definito dalla seguente media pesata:

$$\text{Imp}(D_1, \dots, D_l) = \sum_{j=1}^l \frac{|D_j|}{|D|} \text{Imp}(D_j) \quad (5.1)$$

5.2.2 L'entropia: l'ammontare di confusione di una classe

La definizione di entropia per come data da Shannon è quella di misura di informazione collegata ad un dato evento stocastico. L'unità di misura sono i *bit*, che rappresentano appunto l'*outcome* dell'evento. I bit possono essere usati sia per rappresentare l'informazione, sia per misurare la quantità di informazione del messaggio associato all'evento.

Se un esperimento ha n possibili outcome equiprobabili, il numero di bit necessari per rappresentare ciascuno di questi outcome sarà $b = \lceil \log_2 n \rceil$ (dove il ceiling serve solamente per avere una rappresentazione esatta in termini di memoria).

5.2.2.1 Un esempio di calcolo di entropia

Consideriamo un esempio per chiarire le idee. Supponiamo di lanciare un dado. Se vogliamo rappresentare e trasmettere una delle possibili sei situazioni necessitiamo di:

$$b = \lceil \log_2 6 \rceil = \lceil 2.58 \rceil = 3$$

Se vogliamo invece comunicare solamente la parità del valore ottenuto (che contraddistingue la situazione) necessiteremo invece solamente di $b_1 = \lceil \log_2 2 \rceil = 1$ bit: essendo le due situazioni del tutto equiprobabili non esiste un modo per propendere per una di esse, ed è quindi proprio questo il caso di un cui la quantità di informazione è massima.

5.2.2.2 Il nesso tra l'entropia, decision trees e la crittografia

Ovviamente, se, ripensando all'esempio presentato nella sottosezione [5.2.2.1](#), trasmettessimo prima il messaggio circa la parità, rimane una certa quantità di *confusion* minore per la comunicazione successiva rispetto alla situazione in cui non vi sia alcuna conoscenza pregressa.

Consideriamo ora un ulteriore caso per capire quanto appena sancito. Esistono 3 situazioni in cui il valore del lancio sia dispari e per rappresentarle occorrono $b_2 = \lceil \log_2 3 \rceil = \lceil 1.58 \rceil = 2$. Come notiamo, avendo a monte la conoscenza circa la parità, l'entropia si riduce esattamente della differenza tra la confusione iniziale e l'entropia data dalla prima comunicazione: $b_2 = b - b_1$. Generalizzando:

```
information_gained_by_first_message =
initial_amount_of_confusion - remaining_amount_of_confusion
```

Questo ragionamento è esattamente quello seguito da un decision tree quando deve effettuare la scelta circa il miglior split da considerare: all'inizio si considera un certo quantitativo di confusione dato dal problema, ma una volta applicata una suddivisione in sottoproblemi, la confusione legata ad ogni sottoproblema cambia esattamente seguendo questa legge ed è possibile comparare i possibili split comparando l'ipotetica confusione rimanente a seguito dell'applicazione di ogni split (l'obbiettivo è ovviamente massimizzare per ogni split l'*information gain*).

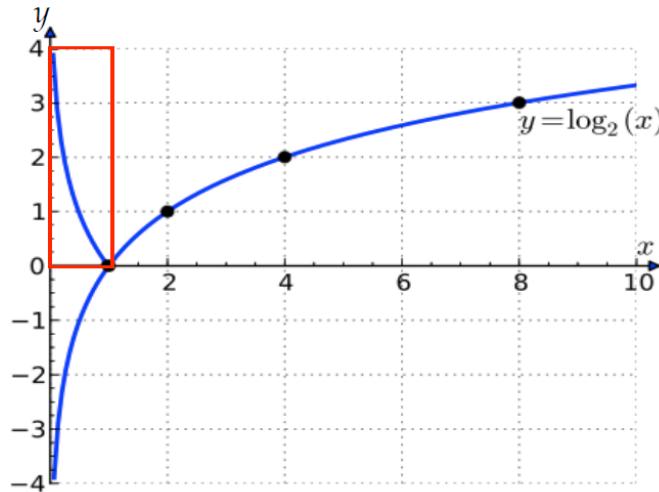
Consideriamo ora il caso in cui gli eventi possibili non sono fra loro equiprobabili e concentriamoci su un evento E che ha probabilità p, allora l'*information gain* sarà:

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

In particolare, se E è molto probabile ($p \approx 1$) $I(E)$ sarà basso, mentre se E è poco probabile ($p \approx 0$) $I(E)$ è molto alto. Si nota qui lo stretto collegamento tra l'*information theory* e la crittografia: l'entropia è strettamente connessa alla possibilità di rendere il messaggio il più corto possibile⁴.

Per concludere, se consideriamo una probabilità x (essendo probabilità avremo che $0 \leq x \leq 1$) possiamo vedere la curva del guadagno di informazione evidenziato in rosso nella Figura [5.3](#).

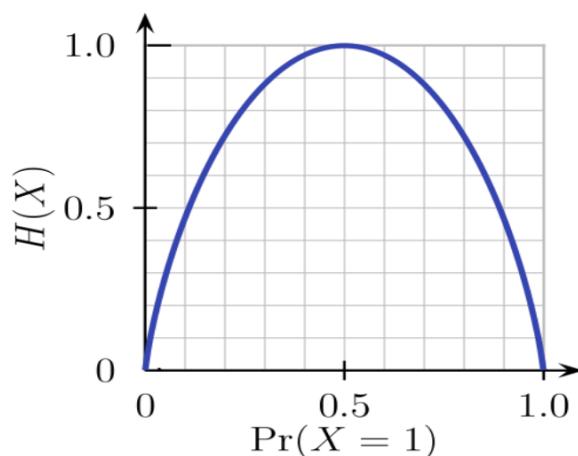
⁴ Motivo per cui gli studi di Shannon sono stati condotti proprio durante il secondo conflitto mondiale, in cui questi temi assumevano particolare importanza

Figura 5.3: Information Plot $0 \leq x \leq 1$

Se un esperimento ha n possibili casi (E_1, \dots, E_n), ciascuno dei quali ha probabilità p_1, \dots, p_n allora l'*average information gain* di una generica situazione i dell'esperimento sarà dato dalla seguente formula (entropia):

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

Questa formula ci permette quindi di ottenere la somma della quantità di informazione per una partizione. Vediamo in Figura 5.4 l'entropia riferita ad una variabile binaria: notiamo ancora che è massima in un caso preciso in cui l'incertezza è massima, mentre nel caso di certezza assoluta (positiva o negativa) sull'evento l'entropia è minima.

Figura 5.4: Entropia $H(X)$ di un generico output binario $X = 1/X = 0$

5.2.3 L'applicazione degli indici di purezza

Riprendiamo l'esempio introdotto nella sezione 5.1 e che qui riportiamo:

```

P1 Length = 3 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many
P2 Length = 4 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many
P3 Length = 3 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few
P4 Length = 5 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many
P5 Length = 5 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few

N1 Length = 5 ∧ Gills = yes ∧ Beak = yes ∧ Teeth = many
N2 Length = 4 ∧ Gills = yes ∧ Beak = yes ∧ Teeth = many
N3 Length = 5 ∧ Gills = yes ∧ Beak = no ∧ Teeth = many
N4 Length = 4 ∧ Gills = yes ∧ Beak = no ∧ Teeth = many
N5 Length = 4 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few

```

Si prova ora a individuare la miglior feature da inserire come radice nell'albero, in modo da ottenere il miglior split, tramite l'applicazione delle formule viste alla sottosezione 5.2.1, in particolare l'entropia e l'indice Gini.

Si ricorda anzitutto la distribuzione di appartenenza degli esempi positivi per ogni feature:

- Length = [3, 4, 5] → [2+, 0-] [1+, 3-] [2+, 2-]
- Gills = [yes, no] → [0+, 4-] [5+, 1-]
- Beak = [yes, no] → [5+, 3-] [0+, 2-]
- Teeth = [many, few] → [3+, 4-] [2+, 1-]

Si calcola quindi l'impurità in tutti i possibili casi del primo split. Nel caso della feature "Length" si hanno 3 partizioni: la prima è pura (troviamo infatti solo esempi positivi) e quindi ha entropia 0, il secondo ha invece entropia

$$-\left(\frac{1}{4}\right)\log_2\left(\frac{1}{4}\right)-\left(\frac{3}{4}\right)\log_2\left(\frac{3}{4}\right)=0.5+0.31=0.81$$

E infine l'entropia del terzo segmento è 1 (è il caso peggiore in cui la confusione è massima).

L'entropia totale, come visto dalla formula, sarà quindi $\frac{2}{10} \cdot 0 + \frac{4}{10} \cdot 0.81 + \frac{4}{10} \cdot 1 = 0.72$. Allo stesso modo, effettuiamo i calcoli anche per le altre features ottenendo:

$$\text{GILLS } \frac{4}{10} \cdot \frac{6}{10} \cdot \left(-\left(\frac{5}{6}\right) \log_2 \left(\frac{5}{6}\right) - \left(\frac{1}{6}\right) \log_2 \left(\frac{1}{6}\right) \right) = 0.39$$

$$\text{BEAK } \frac{8}{10} \cdot \left(-\left(\frac{5}{8}\right) \log_2 \left(\frac{5}{8}\right) - \left(\frac{3}{8}\right) \log_2 \left(\frac{3}{8}\right) \right) + \frac{2}{10} \cdot 0 = 0.76$$

$$\text{TEETH } \frac{7}{10} \cdot \left(-\frac{3}{7} \log_2 \left(\frac{3}{7}\right) - \frac{4}{7} \log_2 \left(\frac{4}{7}\right) \right) + \frac{3}{10} \cdot \left(-\frac{2}{3} \log_2 \left(\frac{2}{3}\right) - \frac{1}{3} \log_2 \left(\frac{1}{3}\right) \right) = 0.97$$

Ricordando che l'entropia è misura dell'impurità (della confusione) rimanente, e deve perciò essere minima, si nota subito che con la feature "Gills" si ottiene un ottimo split. A questo punto la curiosità è tanta ed il confronto è d'obbligo: si vede allora come si comporta il medesimo calcolo con il Gini index:

$$\text{LENGTH } \frac{2}{10} \cdot 2 \cdot \left(\frac{2}{2} \cdot \frac{0}{2}\right) + \frac{4}{10} \cdot 2 \cdot \left(\frac{1}{4} \cdot \frac{3}{4}\right) + \frac{4}{10} \cdot 2 \cdot \left(\frac{2}{4} \cdot \frac{2}{4}\right) = 0.35$$

$$\text{GILLS } \frac{4}{10} \cdot 0 + \frac{6}{10} \cdot 2 \cdot \left(\frac{5}{6} \cdot \frac{1}{6}\right) = 0.17$$

$$\text{BEAK } \frac{8}{10} \cdot 2 \cdot \left(\frac{5}{8} \cdot \frac{3}{8}\right) + \frac{2}{10} \cdot 0 = 0.38$$

$$\text{TEETH } \frac{7}{10} \cdot 2 \cdot \left(\frac{3}{7} \cdot \frac{4}{7}\right) + \frac{3}{10} \cdot 2 \cdot \left(\frac{2}{3} \cdot \frac{1}{3}\right) = 0.48$$

Come era sperabile la nostra scelta ricadrebbe sulla stessa feature anche se si utilizzasse quest'altra misura (anticipiamo che non sempre è così).

5.2.4 La procedura di split

Si può allora introdurre l'Algoritmo 7 generale per la scelta della feature migliore su cui effettuare lo split in un decision tree.

Si ricorda che questa procedura è parte dell'Algoritmo 6 e in quanto tale viene richiamata su ogni partizione creata ricorsivamente.

Algorithm 7 BestSplit-Class(D, F)

```

1: Input: data D; set of feature F
2: Output: feature f to split on
3:  $I_{min} \leftarrow 1$ ;
4: for each  $f \in F$  do
5:   split D into  $D_1, \dots, D_l$  according to the values  $v_j$  of f
6:   if  $Imp(D_1, \dots, D_l) < I_{min}$  then
7:      $I_{min} \leftarrow Imp(D_1, \dots, D_l)$ 
8:      $f_{best} \leftarrow f$ 
9:   end if
10: end for
11: return  $f_{best}$ 
```

5.2.5 Rappresentazione del partizionamento dello spazio delle istanze

Ma come rappresentare graficamente i diversi spazi delle istanze prodotti dagli split applicati seguendo all'albero delle decisioni? Si sup-

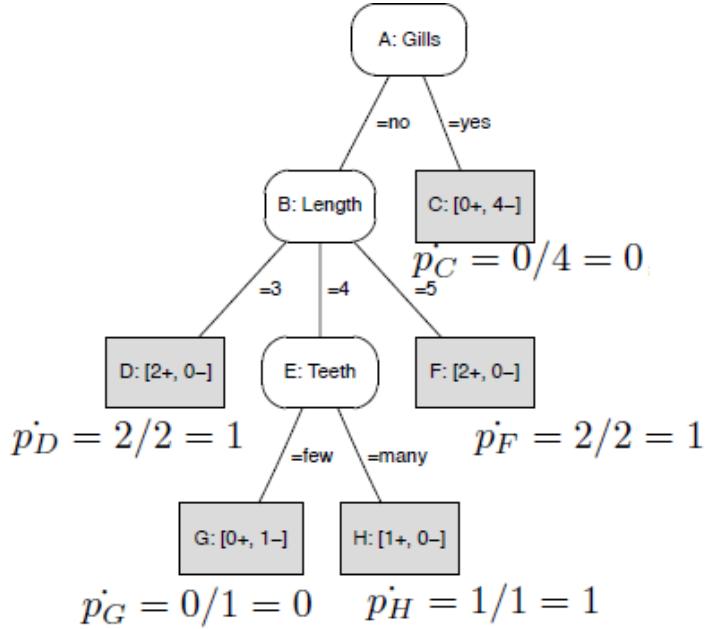


Figura 5.5: Ipotetico decision tree ottenuto dall'applicazione dell'Algoritmo 6.

ponga di aver applicato l'Algoritmo 6 ed aver ottenuto in output il decision tree in Figura 5.5.

Andremo quindi a calcolare la probabilità empirica della classe positiva su ogni singola foglia dell'albero (possiamo vederle inserite in Figura). Si nota che alla foglia C corrisponde una forte regolarità, in quanto 4 esempi negativi sui 5 totali risultano avere l'attributo Gills = yes: da questa osservazione possiamo aggiungere che questo albero è particolarmente in grado di generalizzare.

Si vedrà ora come costruire la *coverage curve* che troviamo in Figura 5.5 a partire dal decision tree. Le linee verticali e orizzontali rappresentano foglie pure, ovvero che ricoprono solamente esempi positivi o negativi (probabilità empirica pari a 1 oppure pari a 0). H ci permette di riconoscere come positivi tutti gli esempi positivi e nessun esempio negativo (come D e F), copre un numero di esempi ridotti (esattamente 1) e permette di coprire l'ultimo esempio positivo (p_1) raggiungendo il *Roc Heaven*: in questo punto si riesce a riconoscere come positivi tutti e soli gli esempi positivi, ed è quindi il punto perfetto per un classificatore. Se si volesse effettuare un ranking basandolo su questo albero si troverebbe al primo posto p_1 e p_3 , poi p_4 e p_5 coperti da F, e poi p_1 coperto da H. Se mettessimo in questo punto un *threshold* tutti i punti rimanenti ricadrebbero nella classe opposta (quella negativa).

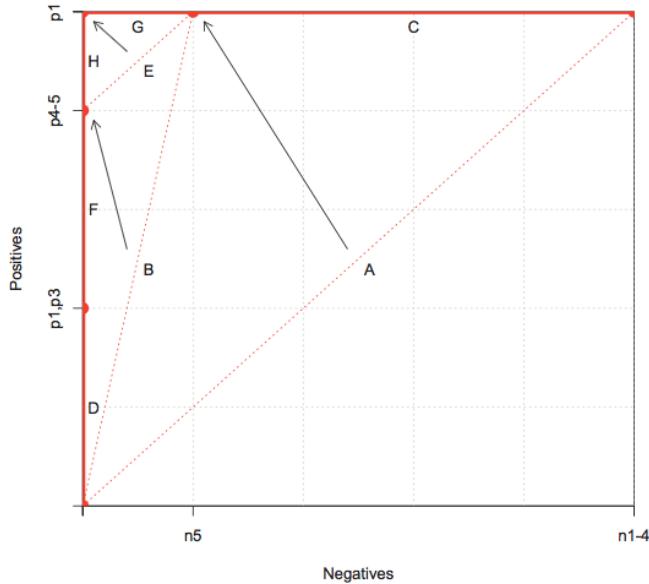


Figura 5.6: Esempio di coverage plot per il decision tree riportato in Figura 5.5.

5.2.6 Imporre un ordinamento sulle foglie

In generale è possibile in qualche modo imporre un ordinamento alle curve (che dividono gli spazi delle istanze) in modo da selezionare la migliore e conoscere la miglior condizione di split sul dataset. Nel nostro caso, A risulta essere la più impura, poiché non è ancora stato fatto alcuno split e quindi la confusione è massima, ma anche la curva E risulta essere parecchio impura poiché uno split su di essa porta ad avere esattamente un esempio positivo e uno negativo.

È ovvio che l'ordine delle curve non dipende dalla struttura dell'albero bensì dalla distribuzione delle classi nelle foglie: questo perchè la probabilità empirica del padre \hat{p} non vincola quella del figlio. Infatti:

$$\hat{p} = \frac{n^+}{n} = \frac{n_1}{n} \frac{n_1^+}{n_1} + \frac{n_2}{n} \frac{n_2^+}{n_2} = \frac{n_1}{n} \hat{p}_1 + \frac{n_2}{n} \hat{p}_2$$

con $n = n^+ + n^- = n_1 + n_2 = n_1^+ + n_1^- + n_2^+ + n_2^-$.

La probabilità empirica del padre è quindi una media pesata delle probabilità empiriche dei figli. Questo ci permette, inoltre, di concludere direttamente che l'ordinamento sulle foglie è dato dalle loro probabilità empiriche, e avremo quindi: $\hat{p}_1 \leq \hat{p} \leq \hat{p}_2$ xor $\hat{p}_2 \leq \hat{p} \leq \hat{p}_1$.

Una volta ottenuto l'ordinamento possiamo rappresentare lo split tracciando la linea che divide i due spazi delle istanze nel grafico.

5.3 RANKING TREES

Si vede ora come applicare i decision tree ad un particolare task: il ranking.

I modelli ad albero dividono lo spazio delle istanze in partizioni. Un tree model, che ordina le partizioni in base alle probabilità empiriche delle foglie, può essere visto come un *ranker*.

I modelli ad albero hanno accesso alla distribuzione locale (in un particolare sottospazio) delle classi nelle foglie (data dalla probabilità empirica) che può essere usata per costruire un ordinamento delle foglie che porta a conoscere così l'ordinamento ottimo per il training set.

Ad esempio nell'albero in Figura 5.7 l'ordinamento ottenuto ordinando le foglie in ordine di probabilità empirica non crescente è [D, F], H, G, C (la probabilità di [D, F] e di H è la stessa ma H copre un numero minore di esempi e quindi viene posto dopo).

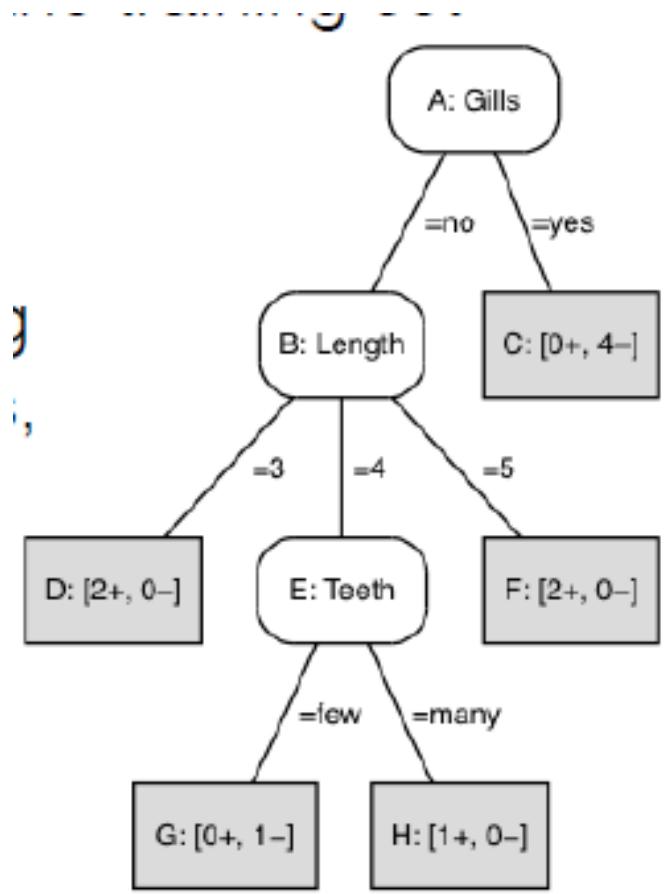


Figura 5.7: Model tree dell'esempio visto in precedenza.

Si ottiene così un ranking tramite la probabilità empirica sulle foglie di un albero decisionale, e questo ranking produce una curva ROC sui dati di addestramento.

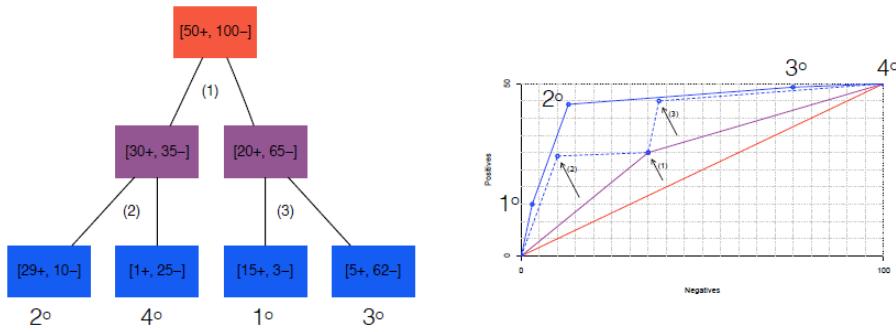


Figura 5.8: Esempio di rappresentazione della curva di copertura a partire da un albero decisionale.

La pendenza di questo segmento (con probabilità empirica p) sarà $\frac{p}{1-p}$: essendo una trasformazione monotona, l'ordinamento non crescente sulla probabilità empirica produce una curva formata da segmenti con pendenza crescente ($p' > p \Leftrightarrow \frac{p'}{1-p'} > \frac{p}{1-p}$), e quindi la curva di copertura così generata sarà convessa.

Siamo interessati al fatto che la curva sia convessa poiché vogliamo selezionare il punto migliore per il classificatore di un certo dataset con una certa distribuzione di classe.

5.3.1 Applicazione dell'algoritmo di costruzione di un albero

Come si sa, se si considera un albero delle feature, ciascun nodo raffigura nella sua etichetta il numero di esempi positivi e negativi che ricopre. Per esempio, nell'esempio in Figura 5.8 la radice ricopre tutti gli esempi considerati (ad esempio 50 positivi e 100 negativi). La sua etichetta sarà quindi $[50+, 100-]$ e il segmento dello spazio delle istanze è rappresentato dalla diagonale.

Al primo split dell'albero il nuovo ranking sarà dato da $[30+, 35-]$ e $[20+, 65-]$ che formano due segmenti. Gli split 2 e 3 formano a loro volta ulteriori segmenti a partire da quelli dei rispettivi padri. L'aggiunta di uno split all'albero aggiunge un segmento alla curva di copertura, ma dopo l'aggiunta di un nuovo segmento questi dovranno essere riordinati.

Il ranking prodotto dall'intero albero sarà $[15+, 3-][29+, 10-][5+, 62-][1+, 25-]$. Basterà quindi riordinare i segmenti ottenuti dalle foglie in base alla loro probabilità empirica per ottenere una curva di copertura convessa.

In questo esempio ci si accorge come la curva muta aggiungendo nuove foglie. A questo punto è doveroso ricordare che la stima della precisione del modello è data dall'area sotto la curva.

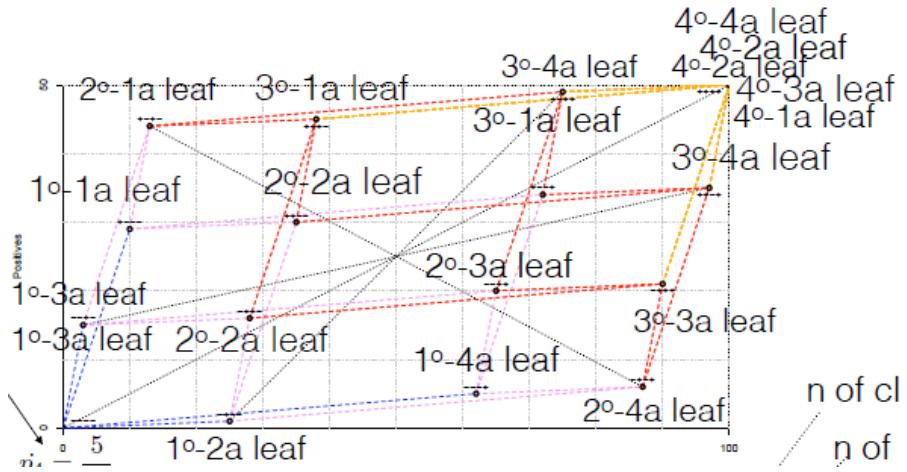


Figura 5.9: L'individuazione delle classi può essere particolarmente complessa.

5.4 L'ATTRIBUZIONE DELLA CLASSE

La Figura 5.9 dà un'idea di quanto può essere complicato individuare la giusta classe per ciascuna foglia senza un criterio: un problema con $n = 2$ classi e $m = 4$ foglie può produrre $n^m = 2^4 = 16$ outcome (e $m! = 4! = 24$ possibili percorsi) e si deve scegliere la soluzione migliore tra queste 16.

Questa soluzione migliore non può essere predeterminata, bensì dipende dal dataset su cui la scelta andrà ad operare. Si supponga, seguendo l'esempio presentato, che il rapporto `class_ratio` = 50/100 sia rappresentativo. Si supponga di poter attribuire una scelta tra 5 labelling, a seconda del *cost ratio* $c = \frac{c_{FN}}{c_{FP}}$ basato sulle classificazioni errate degli esempi positivi su quelle degli esempi negativi:

- + - +- sarà scelto se $c = 1$ o, più in generale, se $\frac{10}{29} < c < \frac{62}{5}$;
- + - ++ scelto se $\frac{62}{5} < c < \frac{25}{1}$;
- + + ++ scelto se $\frac{25}{1} < c$ (verrà predetta sempre la classe positiva se commettere un falso negativo è 25 volte più costoso rispetto al commettere un falso positivo, come può essere il caso della predizione sul cancro);
- - +- scelto se $\frac{3}{15} < c < \frac{10}{29}$;
- - -- scelto se $c < \frac{3}{15}$ (verrà predetta sempre la classe negativa se commettere un falso positivo è 5 volte più costoso rispetto a commettere un falso positivo).

Quindi il semplice class ratio sulle foglie non è sufficiente per decidere la classe, ma entra in gioco la distribuzione del dataset e il peso che gli errori hanno nella applicazione.

5.5 UTILIZZI DI UN FEATURE TREE

Raccogliendo quanto visto, sfruttano le distribuzioni delle classi nelle foglie, ottenute secondo i criteri introdotti nella sezione 5.4 si possono ottenere a partire da un feature tree:

RANKING TREE ordinando le foglie in ordine non crescente di probabilità empiriche (se ci pensiamo, un albero pensato per il ranking permette anche di fare i due task successivi, ed è in tal senso uno strumento più generale);

STIMATORE DI PROBABILITÀ prede la probabilità empirica di una foglia per rendere la stima più robusta anche per le foglie con pochi esempi (applicando la *Laplace correction* $\hat{p} = \frac{n^+ + 1}{n_{tot} + n_c}$ o la *correzione delle m-stime* $\hat{p} = \frac{n_1 + m\hat{\pi}_1}{n_{tot} + m}$);

CLASSIFICATORE in cui si sceglie la condizione operativa come conseguenza di una proporzione delle frequenze tra classi $clr = Pos/Neg$ e il cost ratio $c = c_{FN}/C_{FP}$ al fine di trovare il punto operativo ottimale (*decision threshold*) della ROC Curve, ovvero il punto di intersezione tra la curva isometrica (insieme di punti di eguale accuratezza) con la retta $\frac{1}{c \cdot clr}$: le foglie che cadono sul punto operativo prediranno la classe positiva, le altre la classe negativa.

5.6 POTARE UN ALBERO

Un punto interessante è capire se si vuole, in qualsiasi caso, raggiungere il ROC heaven: sarebbe ovviamente fantastico, ma spesso questo implica un albero troppo complesso, che come si è ormai compreso, implica overfitting.

Spesso si è allora portati ad introdurre una potatura dell'albero piuttosto che fare riferimento all'intero albero ottenuto. Potare un albero significa semplificarlo (utile soprattutto nella comunicazione dell'albero) riducendo il numero di foglie e rendendolo più efficiente in quanto dovrà tener conto di meno condizioni, come viene fatto in Figura 5.10a.

La potatura non ha effetto circa la scelta del punto operativo, ma incide ovviamente sulla prestazioni di un modello basato sull'albero in quanto diminuisce la precisione (possiamo vederlo in Figura 5.10b e questo in alcuni casi (ad esempio quelli in cui si vuole evitare il problema dell'overfitting) è un bene.

Il grafico in Figura 5.11 mostra un caso in cui ci si accorge della presenza di overfitting: dopo una determinata soglia di esempi, l'errore sul training set continua a diminuire abbastanza rapidamente, mentre sul test set aumenta sensibilmente.

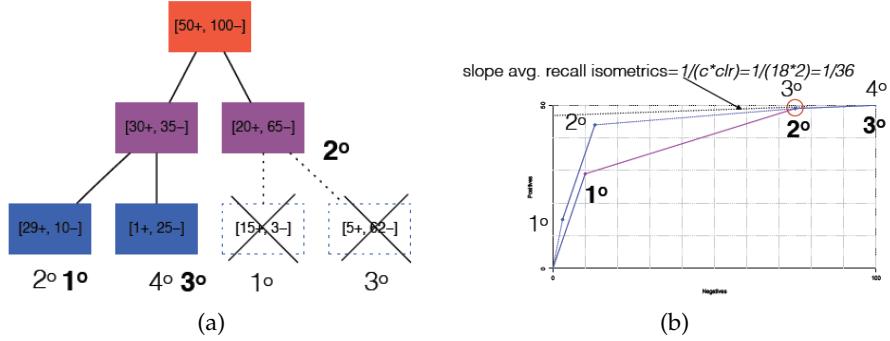


Figura 5.10: Effetti della potatura di un albero.

Nello stesso grafico si nota nella parte iniziale una fase di *underfitting*, in cui il numero ridotto di esempi non permette un buon operato.

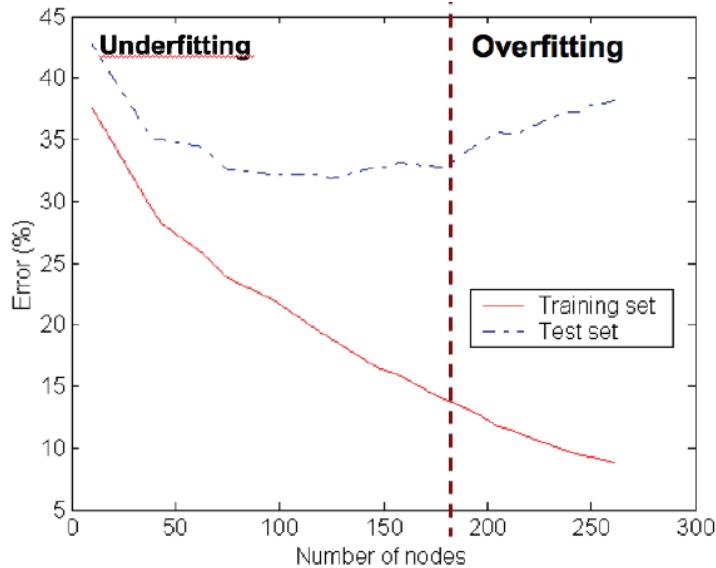


Figura 5.11: Manifestazione grafica di overfitting

Il punto minimo della curva di test rappresenta ideologicamente il punto ottimo in cui smettere di fare apprendimento sul modello, ma individuare questo momento durante il training non è cosa scontata: un modo per farlo è utilizzare un set dedicato, ed è quello che vedremo fare nell'Algoritmo 8.

5.6.1 Algoritmo di potatura di un feature tree

In questo approccio qui riportato prima deve necessariamente essere costruito l'albero delle feature e poi su questo viene (eventualmente) applicato un algoritmo di potatura, ma per completezza si cita l'esistenza di algoritmi per la costruzione degli alberi delle feature che hanno meccanismi di potatura intrinseci basata proprio sulla sti-

ma dell'errore commesso sul test set (ne parleremo nella sottosezione [5.6.2](#)).

Viene riportato quindi l'algoritmo per effettuare la potatura di un albero delle feature precedentemente creato.

Algorithm 8 PruneTree(T, D) - Reduced error pruning of a decision tree

```

1: Input: decision tree  $T$ ; labelled data  $D$ .
2: Output: pruned tree  $T'$ .
3: for every internal node  $N$  of  $T$ , starting from the bottom do
4:    $T_N \leftarrow$  subtree of  $T$  rooted at  $N$ ;
5:    $D_N \leftarrow x \in D \mid x$  is covered by  $N$ ;
6:   if accuracy of  $T_N$  over  $D_N$  is worse than majority class in  $D_N$ 
    then                                      $\triangleright$  In realtà, questo
      if considera le classi equalmente frequenti, altrimenti dovremmo
      considerare quanto esplicato nella sottosezione 5.4
7:     replace  $T_N$  in  $T$  by a leaf labelled with the majority class in
       $D_N$ ;
8:   end if
9: end for
10: return pruned version of  $T$ 
```

5.6.2 Stima delle generalizzazioni degli errori

Un metodo alternativo alla tecnica vista basata sulla riduzione dell'errore nel pruning si basa sulla stima diretta della generalizzazioni dell'errore sul test set durante la costruzione dell'albero, senza fare ulteriore lavoro in fase di costruzione che poi sarà disfatto.

Possiamo procedere in diversi modi:

1. Il primo metodo prevede l'introduzione di una penalità k per ogni foglia che non permette la creazione di una ulteriore foglia se (ad eventuale seguito di ciò) l'errore del nodo padre non diminuisce di almeno $k + 1$. Questo meccanismo rende molto più complesso l'algoritmo di generazione dell'albero.
2. Un altro meccanismo prevede l'utilizzo di una formula che stima direttamente l'errore sul test (a partire dal training set). Si assume in partenza che, qualsiasi sarà il test set, l'errore commesso su di esso sarà maggiore rispetto a quello commesso sul training set. Se si considera che
 - l'errore sul training set è dato ed è esattamente $\sum_{i=1}^N e_i$
 - l'errore sul test set (errori di generalizzazione) è esattamente dato da $\sum_{i=1}^N e'_i$

dove e_i e e'_i sono gli errori commessi nella foglia i -esima. Possiamo stabilire una approssimazione dell'errore commesso sul test set in questo modo:

$$e'(T) = e(T) + N * 0.5$$

dove N è il numero di foglie e 0.5 è una penalità introdotta per ciascuna foglia: la creazione di una foglia è consentita se migliora la classificazione del nodo padre di almeno un numero di istanze pari a $\lceil 0.5 \rceil = 1$. Se ad esempio avessimo un albero con 30 foglie che commette 10 errori su un totale di 1000 istanze del training set (tutto sommato, un albero su cui è stato fatto un buon apprendimento) concluderemmo:

$$\text{TRAINING ERROR} = (10/1000) = 1\%$$

$$\text{GENERALIZATION ERROR} = (10 + 30 * 0.5)/1000 = 2.5\%$$

Sulla base di queste formule si possono inoltre costruire criteri di arresto per gli algoritmi di apprendimento dei modelli ad albero (ad esempio, se una foglia ricopre un segmento dello spazio delle istanze molto ristretto, l'apprendimento può terminare).

5.7 IMPATTO DELLA DISTRIBUZIONE DELLE CLASSI

5.7.1 Un esempio sconvolgente

Si è visto in precedente che l'indice Gini e l'entropia (e le altre misure) funzionano spesso in modo simile, accennando al fatto che non sempre l'equivalenza vale ma non spingendoci oltre nella discussione. In realtà, proprio le misure profondamente diverse che spesso si ottengono dalla loro applicazione hanno portato la ricerca a proporre nuove misure più veritiera e omogenee.

Viene presentato un esempio a riguardo. Si supponga di avere 10 esempi positivi e altrettanti negativi e vogliamo scegliere tra i seguenti split:

- [8+, 2-][2+, 8-]
- [10+, 6-][0+, 4-]

Si può quindi calcolare l'entropia di entrambi gli split e concludere che il primo split è il migliore. Poi si potrebbe però voler applicare il Gini index, e ancora sarà il primo split ad avere la meglio.

Poi ci si ricorda però che un uccellino disse che la radice quadrata del Gini index era una misura di impurità più accurata, poiché non è sensibile ai cambiamenti di frequenza delle classi nel dataset, e si decide di applicare questa misura: questa simpatica se ne esce dicendo che il secondo split è migliore. Cosa si fa?

Se si conosce la distribuzione delle classi nel dataset è possibile ritoccare (gonfiare) manualmente i valori delle distribuzioni negli split per costringere anche l'entropia e il Gini index a prendere in considerazione questo parametro (al posto di perdere la testa dietro la matematica per applicare il cost ratio).

Si supponga di sapere che un errore sui positivi è dieci volte più costoso di un errore sui negativi ($c = 10$). Possiamo allora trasformare gli split con i valori

- [80+, 2-][20+, 8-]
- [100+, 6-][0+, 4-]

e ricalcolare entropia, Gini index e il suo quadrato e... magia! Tutti e tre i criteri dicono che il secondo split è il migliore.

Ovviamente gonfiare il training set in questo modo significa aumentare (e non di poco) i tempi di addestramento. Ciò che però si desidera evidenziare è che la radice del Gini Index è una misura stabile, che non varia all'aumentare degli esempi e quindi della distribuzione delle classi.

5.7.2 L'impurità relativa

Se il Gini index di un nodo padre è dato da $2\frac{n^+}{n}\frac{n^-}{n}$, il Gini index pesato di uno dei suoi figli sarà $\frac{n_1}{n}2\frac{n_1^+}{n_1}\frac{n_1^-}{n_1}$. L'impurità pesata del figlio in proporzione all'impurità del nodo padre è definita come *impurità relativa* e si ottiene dal rapporto tra l'impurità del nodo padre e quella del nodo figlio:

$$\frac{n_1^+ n_1^- / n_1}{n^+ n^- / n}$$

Possiamo ripetere gli stessi calcoli per la radice del Gini Index:

- impurità del nodo padre: $\sqrt{\frac{n^+}{n}\frac{n^-}{n}}$
- impurità pesata di un nodo figlio: $\frac{n_1}{n} \sqrt{\frac{n_1^+}{n_1}\frac{n_1^-}{n_1}}$
- impurità relativa: $\sqrt{\frac{n_1^+}{n^+}\frac{n_1^-}{n^-}}$

L'ultimo rapporto non cambia se moltiplichiamo tutti i valori che coinvolgono la classe positiva per un fattore c . Ciò significa che se si utilizza la radice del Gini index come misura per decidere lo split migliore no si favoriscono i nodi alla copertura di più esempi (in dataset in cui la distribuzione delle classi non è equivalente).

5.7.3 La ricetta di Peter Flach per il learning di un decision tree

Quanto visto nelle sottosezioni 5.7.1 e 5.7.2 rappresenta una serie di risultati degli studi di Peter Flach, che ha dettato una "ricetta" per l'apprendimento dei decision tree.

Si ripercorre quindi tale ricetta.

1. Prima di tutto, occorre concentrarsi sul generare un feature tree in grado di produrre un ranking sugli esempi (come accennato, applicabile anche per classificazione e stima probabilistica)
2. usare la radice del Gini Index come misura di impurità per i motivi ampiamente discussi
3. Se si vuole fare stima probabilistica, occorre applicare qualche forma di correzione (Laplace correction o m-estimate) alle probabilità empiriche per rendere le probabilità più robuste nel caso in cui gli esempi siano ridotti
4. Riconoscere il miglior punto in cui applicare il threshold o per cui assegnare le label.
5. Infine, applicare del pruning a tutti i rami con foglie che scelgono le stesse label (tenendo eventualmente conto del cost ratio).

5.8 APPRENDIMENTO DI ALBERI BASATO SULLA RIDUZIONE DELLA VARIANZA

5.8.1 Introduzione al problema della regressione

Applicheremo ora il modello ad albero al task del clustering: cambieremo la funzione obiettivo dell'algoritmo per produrre un albero differente, facendole valutare la varianza su un insieme di esempi appartenenti ad uno stesso spazio delle istanze, tentando di minimizzarla.

Cominciamo pensando ad una variabile Bernoulliana (booleana), che è una variabile associata ad una certa probabilità \hat{p} di successo: la sua varianza è $\hat{p}(1 - \hat{p})$ che è esattamente la metà del valore del Gini index.

In problemi di regressione la varianza su una variabile Y è definibile come

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

Se una condizione di split partiziona l'insieme target di Y in sottinsiemi indipendenti Y_1, \dots, Y_l , la varianza media pesata sarà data da

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

dove il primo termine è una costante dato il set Y e quindi il termine da massimizzare è solo il secondo.

5.8.2 Alberi di regressione

Nei problemi di regressione i valori su cui si opera (gli elementi dell'insieme Y) sono continui.

Quindi, occorre modificare l'algoritmo in cui si seleziona il miglior split (algoritmo 7) come segue:

- la misura Imp dovrà valutare la funzione $\text{Var}(Y)$ (anzi, come accennato prima, solo il secondo termine $-\sum_{j=1}^l \frac{|Y_j|}{|Y|} \bar{y}_j^2$, poiché il primo termine è solo una costante data dalla somma sui valori contenuti nel nodo padre) allo scopo di identificare le partizioni formate in modo tale da minimizzare la distanza quadratica media attorno al centroide \bar{y} (vogliamo partizioni "ben distinte").
- la funzione $\text{Label}(Y)$ ritorna il valor medio presente nel set Y identificato dalla foglia
- la funzione $\text{Homogeneous}(Y)$ ritorna vero se la varianza dei valori nell'insieme Y appartiene a un certo threshold.

L'albero che ne consegue, sarà un *albero di regressione*, ed è un albero che potrebbe soffrire di overfitting (come d'altronde anche i decision tree) per via della possibile presenza di foglie con pochi esempi.

5.8.2.1 Un esempio di creazione di un albero di regressione

Immaginiamo di essere dei collezionisti di organi⁵ e di monitorare una certa asta online dopo aver collezionato alcuni dati circa aste simili. Tali dati sono riportati in Tabella 5.12. Vogliamo costruire, a partire da questa tabella, l'albero di regressione che ci possa aiutare a determinare un prezzo ragionevole per il nostro possibile acquisto.

Vediamo tre feature, quindi, inizialmente, tre possibili split:

```
MODEL =[A100,B3,E112,M102,T202] → [1051,1770,1900][4513][77][870][99,270,625]
CONDITION =[excellent, good, fair] → [1770,4513][270,870,1051,1900][77,99,625]
LESLIE = [yes, no] → [625,870,1900][77,99,270,1051,1770,4513]
```

I valori medi sui tre split e i loro quadrati

$$\text{MODEL} = 1574, 4513, 77, 870 \text{ e } 331 \rightarrow \frac{1}{3}(1574^2) + \frac{1}{9}(4513^2) + \frac{1}{9}(77^2) + \frac{1}{9}(870^2) + \frac{1}{3}(331^2) = 3.21 \cdot 10^6$$

⁵ L'accezione è riferita allo strumento musicale, ci si discosta da atti legati ad eventuali commerci illegali di organi umani.

#	Model	Condition	Leslie	Price
1.	B3	excellent	no	4513
2.	T202	fair	yes	625
3.	A100	good	no	1051
4.	T202	good	no	270
5.	M102	good	yes	870
6.	A100	excellent	no	1770
7.	T202	fair	no	99
8.	A100	good	yes	1900
9.	E112	fair	no	77

Figura 5.12: Tabella che riporta un certo numero di rilevazioni su aste di organi musicali

CONDITION = 3142, 1023 e 267 $\rightarrow 2.68 \cdot 10^6$

LESLIE = 1132 e 1297 $\rightarrow 1.55 \cdot 10^6$

Sceglieremo quindi lo split sulla prima feature, Model, poiché vogliamo massimizzare il valor medio al quadrato riferito allo split, in modo da massimizzare la varianza (ricordiamo che nella formula della varianza questo termine viene sottratto).

Proseguendo nella costruzione, per la foglia rappresentante la partizione sul valore A100 possiamo applicare due possibili split:

CONDITION = [1770][1051, 1900]

LESLIE = [1900][1051, 1770]

Per il valore T202 proseguiamo in modo analogo:

CONDITION = []|[270][99, 625]

LESLIE = [625][99, 270]

Conviene in entrambi i casi (lasciamo i calcoli a voi) splittare sulla feature "Leslie".

A procedura ultimata si otterà l'albero in Figura 5.13. Possiamo immediatamente notare la presenza di overfitting, in quanto alcune foglie coprono istanze singole (o addirittura nessuna).

5.8.3 Introduzione al problema del clustering

Abbiamo già accennato in una timida parentesi che minimizzare la varianza equivale a rendere più solida la separazione tra le partizioni.

Introduciamo una funzione $Dis : X \times X \rightarrow \mathbf{R}$ che misura la *dissimilità* tra due istanze x_1 e $x_2 \in X$.

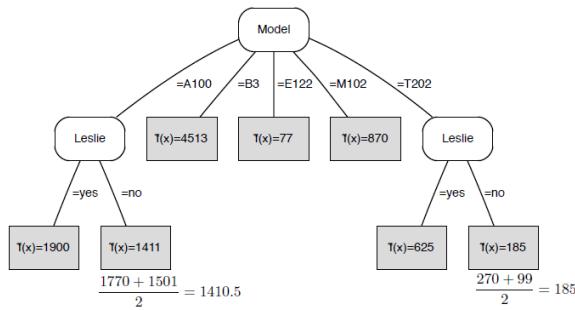


Figura 5.13: Esempio di regression tree per l'esempio presentato alla sotto-sottosezione

Se vogliamo applicare un albero allo scopo di creare gruppi di esempi (*cluster*) possiamo misurare la dissimilarità di un cluster formato da un set di istanze D come:

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

Più piccola sarà $\text{Dis}(D)$, migliore sarà l'omogeneità all'interno del cluster.

5.8.4 Alberi di Clustering

Per implementare l'algoritmo di costruzione di questi alberi occorre introdurre come funzione obiettivo la misura della dissimilarità. In particolare l'algoritmo che cerca il migliore split [7](#) sarà implementato sulla base di un valore medio dei $\text{Dis}(D_i)$ dove D_i è una delle partizioni ottenute dopo lo split di un nodo (padre):

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

L'algoritmo cercherà di minimizzare questa misura di dissimilarità.

5.8.4.1 Un esempio di creazione di un albero di clustering

Riprendendo in esame l'esempio della sotto-sottosezione [5.8.2.1](#) e supponendo di avere altre due features numeriche da aggiungere all'esempio (prezzo di riserva e numero di offerte), si ottiene la rappresentazione degli esempi del dataset in Figura [5.14](#). La relativa matrice di dissimilarità è quella in Figura [5.15](#). Le modalità di lettura di questa tabella mostrano ad esempio come la prima transazione è particolarmente differente dalle altre 8.

Valutando poi sulle stesse features (Model, Condition e Leslie) i possibili split saranno

`MODEL =[A100, B3, E112, M102, T202] → [3, 6,8][1][9][5][2, 4,7]`

	Model	Condition	Leslie	Price	Reserve	Bids
T1	B3	excellent	no	45	30	22
T2	T202	fair	yes	6	0	9
T3	A100	good	no	11	8	13
T4	T202	good	no	3	0	1
T5	M102	good	yes	9	5	2
T6	A100	excellent	no	18	15	15
T7	T202	fair	no	1	0	3
T8	A100	good	yes	19	19	1
T9	E112	fair	no	1	0	5

Figura 5.14: Matrice del dataset con nuove feature

	T1	T2	T3	T4	T5	T6	T7	T8	T9
T1	0	11	6	13	10	3	13	3	12
T2	11	0	1	1	1	3	0	4	0
T3	6	1	0	2	1	1	2	2	1
T4	13	1	2	0	0	4	0	4	0
T5	10	1	1	0	0	3	0	2	0
T6	3	3	1	4	3	0	4	1	3
T7	13	0	2	0	0	4	0	4	0
T8	3	4	2	4	2	1	4	0	4
T9	12	0	1	0	0	3	0	4	0

Figura 5.15: Matrice di dissimilarità con dati conosciuti a priori per l'esempio

CONDITION = [excellent,good, fair] → [1,6][3, 4, 5, 8][2, 7,9]

LESLIE = [yes,no] [2,5,8][1,3,4,6,7,9]

La dissimilarità nelle partizioni formate dallo split su Model sono:

Model = A100 →

$$\begin{aligned}
 & \frac{1}{|Y|^2} ((d(T_3, T_3) + d(T_3, T_6) + d(T_3, T_8) + d(T_6, T_3) + d(T_6, T_6) + \\
 & + d(T_6, T_8) + d(T_8, T_3) + d(T_8, T_6) + d(T_8, T_8)) = \\
 & = \frac{1}{3^2} (0 + 1 + 2 + 1 + 0 + 1 + 2 + 1 + 0) = \\
 & = 0.89
 \end{aligned} \tag{5.2}$$

Model = B3/E112/M102 → Avendo una sola istanza la dissimilarità è pari a zero (la dissimilarità dell'unico esempio con se stesso è ovviamente nulla)

Model = T202 →

$$1 \frac{1}{3^n} (0 + 1 + 0 + 1 + 0 + 0 + 0 + 0 + 0) = 0.22.$$

La dissimilarità media dei cluster è allora

$$\frac{3}{9} \cdot 0.89 + \frac{1}{9} \cdot 0 + \frac{1}{9} \cdot 0 + \frac{1}{9} \cdot 0 + \frac{3}{9} \cdot 0.22 = 0.37$$

Gli stessi calcoli condotti su ipotetici split su Condition e Leslie potrebbero a

CONDITION

$$\frac{2}{9} \cdot 1.5 + \frac{4}{9} \cdot 1.19 + \frac{3}{9} \cdot 0 = 0.86,$$

LESLIE

$$\frac{3}{9} \cdot 1.56 + \frac{6}{9} \cdot 3.56 = 2.89$$

Per i motivi sopra spiegati, sceglieremo di effettuare il primo split sulla feature Model ($0.37 < 0.86 < 2.89$).

Possiamo inoltre notare come la feature Leslie sia quasi del tutto scorrelata al suo interno (la dissimilarità risultante da una sua applicazione sul primo split è di 2.89 che è prossima alla dissimilarità iniziale di 2.94).

I valori medi sulle tre feature numeriche formano il vettore (12.6, 8.6 e 7.9) e le loro varianze sono date dal vettore (171.1, 101.8 e 48.8). La distanza quadratica media (Euclidea) dal centroide è misura della stessa dissimilarità ed è data dalla somma delle varianze, che è 321.7 (questo anticipa una formula che permette di ottenere la varianza complessiva a partire dalle singole varianze delle features).

Se si eseguono gli stessi calcoli sul cluster ottenuto dallo split Model = A100 i due vettori sono rispettivamente (16, 14, 9.7) e (12.7, 20.7, 38.2) e la distanza quadratica media dal centroide sarà 71.6. Per il cluster ottenuto dallo split Model = T202 otterremmo i vettori (3.3, 0, 4.3) e (4.2, 0, 11.6) una distanza quadratica media di 15.8.

Notiamo quindi che applicando questi split e ottenendo i cluster possiamo ridurre la distanza quadratica media, ovvero la dissimilarità iniziale. Si può allora ricostruire l'albero di clustering (Figura 5.16) con le foglie etichettate con il vettore delle medie dei valori delle feature numeriche.

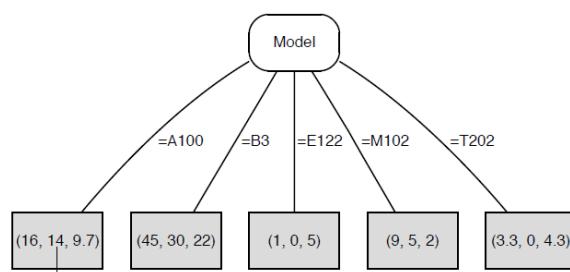


Figura 5.16: Albero di clustering risultante basato sulla distanza euclidea delle feature numeriche

5.8.5 Ossevazioni finali sugli alberi di clustering

5.8.5.1 Pruning e labelling di un cluster

È stato osservato che ottenere cluster di dimensioni ridotte permette di raggiungere bassa dissimilarità e queste situazioni si traducono inevitabilmente in overfitting.

Possiamo quindi applicare del pruning (deliberando un pruning set) sugli split ridotti che non portano migliorie in termini di diminuzione della dissimilarità. Oppure, si può rimuovere i cluster ridotti ad esempi singoli (i c.d. *outliers*, che sporcano la misura di dissimilarità) e ricalcolare la dissimilarità.

Ma come scegliere le label per i cluster? Si può procedere scegliendo l'esempio più rappresentativo, oppure si potrebbe applicare la scelta del *medoide*, ovvero scegliere l'esempio con la dissimilarità totale (rispetto a tutti gli altri esempi del cluster) minore.

5.8.5.2 Ancora sulla distanza Euclidea

È stato già visto già visto in azione lo stratagemma dell'utilizzo della distanza Euclidea come computazione della funzione di dissimilarità. Si scende ora un po' più nel dettaglio. Se un oggetto x_j è descritto da d feature numeriche nello spazio $X \subseteq \mathbb{R}^d$, allora può essere anche descritto da un vettore con d componenti $\vec{x}_j = \langle x_{j1}, x_{j2}, \dots, x_{jd} \rangle$ dove x_{ji} rappresenta il valore della feature i nell'oggetto j .

La dissimilarità $Dis(x_1, x_2)$ può essere calcolata tramite una funzione della distanza tra i due oggetti:

$$Dis(x_1, x_2) = (\vec{x}_1 - \vec{x}_2)^2 = \sum_{i=1}^d (x_{1i} - x_{2i})^2$$

È inoltre possibile riscrivere la formula della varianza della feature i in un set di N oggetti come segue:

$$\text{Var}_i(X) = \frac{1}{N} \sum_{j=1}^N [x_{ji} - \bar{x}_{\cdot i}]^2$$

dove $\bar{x}_{\cdot i}$ è sottointeso il centroide.

La somma delle varianze su tutte le features può essere interpretata come la distanza Euclidea quadratica media tra i vettori dello spazio d -dimensionale:

$$\sum_{i=1}^d \text{Var}_i(X) = \frac{1}{N} \sum_{i=1}^d \sum_{j=1}^N [x_{ji} - \bar{x}_{\cdot i}]^2 = \frac{1}{N} \sum_{j=1}^N (\vec{x}_j - \vec{x})^2$$

Inoltre, la varianza equivale alla metà della distanza quadratica media tra ciascuna coppia di oggetti, ovvero

$$Dis(X) = \frac{1}{N^2} \sum_{\vec{x}' \in X} \sum_{\vec{x}'' \in X} (\vec{x}' - \vec{x}'')^2 = 2 \cdot \text{Var}(X)$$

e la varianza può essere computata come somma delle varianze dei singoli oggetti:

$$\text{Var}(X) = \sum_{i=1}^d \text{Var}_i X$$

In conclusione per calcolare la dissimilarità di un dataset X , necessitiamo solamente del vettore delle medie delle istanze $\vec{\bar{x}}$ e computare le singole varianze $\text{Var}_i(X)$ (quest'ultime calcolabili in tempo lineare $\mathcal{O}(|D|)$)

5.8.5.3 Usare feature numeriche per fare split

Nell'esempio presentato nella sotto-sottosezione 5.8.4.1 abbiamo usato le feature categoriche per effettuare gli split e le feature numeriche per i calcoli sulla dissimilarità.

Nella pratica anche le feature numeriche possono essere sfruttate per lo splitting, ad esempio basandoci su una condizione di maggiorazione o minorazione della feature: $f \geq t$ oppure $f \leq t$.

Alla funzione BestSplit spetta l'arduo compito di selezionare il miglior valore di t .

Si nota la vicinanza di questa operazione con le operazioni di discretizzazioni.

6

MODELLO A REGOLE

6.1 INTRODUZIONE

Si vedranno in questo Capitolo diversi modelli, primo fra tutti i modelli a liste ordinate, in cui le regole apprese dal training set sono ordinate, per ogni esempio, secondo un particolare criterio in modo tale che la prima regola sarà quella da applicare e deciderà la classe a cui l'esempio appartiene.

Si descriveranno inoltre altri modelli in cui le regole sono ordinate in set, e queste saranno migliori poiché più regole saranno applicabili ad un singolo esempio. Inoltre, questi modelli, sono in grado di segmentare lo spazio delle istanze con molta precisione.

Si tratteranno ancora regole applicate a task diversi da quelli predittivi, ovvero quelli descrittivi (si vuole con questi descrivere una certa classe per poi effettuare delle analisi sui loro elementi descrittivi).

Infine, si introdurranno modelli in cui vi sono regole associative estratte dai letterali estratti dal dataset (qui si tratteranno solo per i task predittivi, ma esistono nella letteratura anche per scopi descrittivi).

6.2 APPRENDERE LISTE DI REGOLE ORDINATE

6.2.1 *L'idea del modello a regole*

Se si ricorda quello che avveniva nell'apprendimento basato sui modelli ad albero, si partiva dalla radice attribuendo ad essa una certa feature e poi si procedeva nella generazione dell'albero aggiungendo letterali.

Vogliamo ora fare qualcosa di simile, aggiungendo letterali alla regola nel rispetto della forma congiunta.

A differenza degli alberi di decisione, viene però appresa una regola alla volta (mentre nei decision tree uno split permetteva la crescita in ampiezza di più sottoalberi).

Ciascuna regola potrà identificare un segmento dello spazio delle istanze e si preferiranno regole in grado di identificare ampia parte del dataset e che quindi consentano una precisione alta nella predizione della classe positiva.

La conseguenza della regola dovrà essere la label della classe predetta dal modello.

Se si ripete il learning senza apportare modifiche al dataset si rischierebbe di riottenere in output la stessa regola, non raggiungendo mai un ipotetico criterio di arresto: per evitare ciò occorre eliminare gli

esempi dello spazio delle istanze individuate dalla regola generata dall’iterazione corrente per permettere alle regole successive di individuare spazi diversi. Fatto questo, possiamo ripartire alla ricerca di nuove regole.

Esisterà ovviamente un criterio di arresto che forzerà la procedura a terminare, anche al fine di evitare overfitting.

Questa procedura genererà una lista di regole, che dovranno necessariamente essere considerate in ordine e mai isolate.

6.2.2 *Un esempio introduttivo*

In questa sottosezione viene ripreso un esempio sperabilmente ormai a voi caro per comprendere il funzionamento del modello a regole. In questo esempio ricordiamo esistere 5 esempi positivi

P1 Length = 3 \wedge Gills = no \wedge Beak = yes \wedge Teeth = many

P2 Length = 4 \wedge Gills = no \wedge Beak = yes \wedge Teeth = many

P3 Length = 3 \wedge Gills = no \wedge Beak = yes \wedge Teeth = few

P4 Length = 5 \wedge Gills = no \wedge Beak = yes \wedge Teeth = many

P5 Length = 5 \wedge Gills = no \wedge Beak = yes \wedge Teeth = few

e altrettanti negativi:

N1 Length = 5 \wedge Gills = yes \wedge Beak = yes \wedge Teeth = many

N2 Length = 4 \wedge Gills = yes \wedge Beak = yes \wedge Teeth = many

N3 Length = 5 \wedge Gills = yes \wedge Beak = no \wedge Teeth = many

N4 Length = 4 \wedge Gills = yes \wedge Beak = no \wedge Teeth = many

N5 Length = 4 \wedge Gills = no \wedge Beak = yes \wedge Teeth = few

Si possono presentare, a partire da questo dataset, dei conteggi circa la copertura dello spazio delle istanze per ciascun valore di ciascuna feature, come si vede dalla Figura 6.1a. Tre di questi conteggi portano ad una purezza massima e, nel grafico in Figura 6.1b si sovrappongono rispettivamente all’asse x e y: sono ovviamente questi quelli a cui siamo interessati.

Altri due letterali coprono 2 esempi positivi e due negativi: hanno quindi la stessa impurità dell’intero dataset e vengono rappresentati quindi lungo la diagonale del grafico di copertura. Il punto evidenziato dalla freccia in violetto indica un segmento in cui vengono individuati correttamente tutti gli esempi positivi e un esempio negativo (è la scelta Gills = yes che permette la più ampia copertura per la miglior purezza).

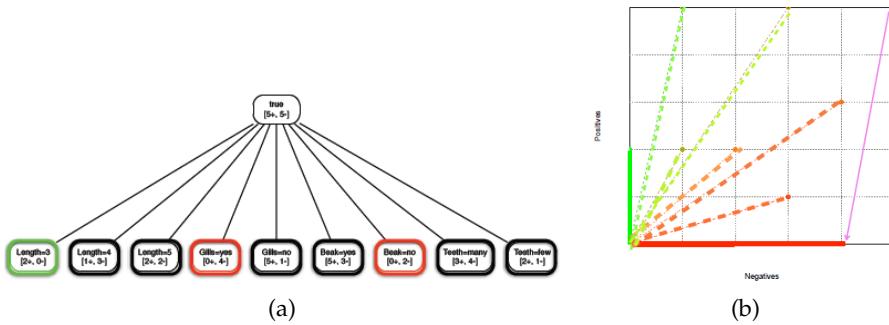


Figura 6.1: Gli esempi in rosso sono puri per la classe negativa, mentre quello in verde lo è per la classe positiva. Le altre linee sono di colore tendente al verde se $\hat{p} > 1/2$ e tendenti al rosso se $\hat{p} < 1/2$ (incontrandosi sull'arancione dove $\hat{p} = 1/2$)

Aggiungendo un letterale al corpo della regola si suddivide ideologicamente lo spazio delle istanze in segmenti. E fin qua, nulla di diverso rispetto a quanto avveniva nei decision tree se non che in questi ultimi si usavano misure di purezza di tutti i nodi figli (su cui poi applicare una media pesata per una ricerca euristica sulla bontà dello split) nell'apprendimento dei modelli a regole si è interessati sulla purezza (usando una qualsiasi misura) di un solo figlio (quello su cui il letterale aggiunto è verificato).

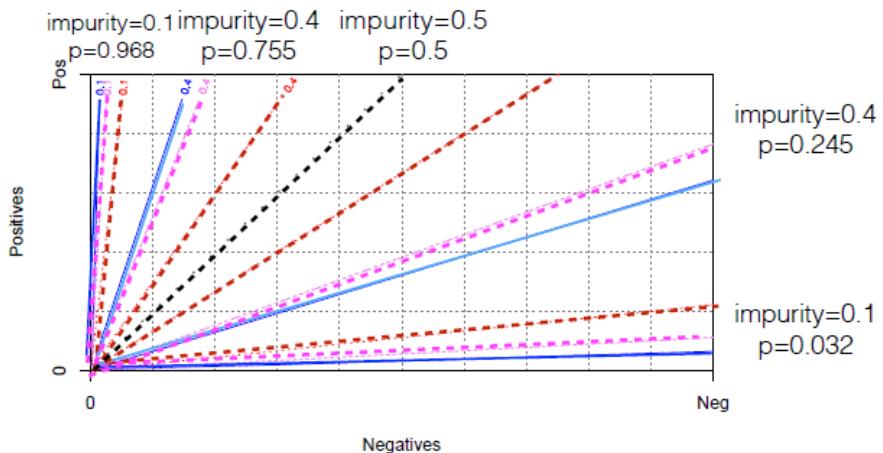


Figura 6.2: Misure di impurità applicate all'esempio presentato.

Si può verificare in Figura 6.2 (con valori normalizzati sul valore massimo di $\frac{1}{2}$) le misurazioni corrispondenti effettuate con l'entropia (in blu), con il Gini index (in violetto) e con la classe minoritaria (in rosso). La linea in nero è definita per il valore $\hat{p} = \frac{1}{2}$ e ciascuna linea divide il piano in due parti, una sopra la simmetria (dove l'impurità decresce con l'aumentare della probabilità empirica) e una simmetrica dal punto di vista della misurazione al di sotto (dove l'impurità è invece proporzionale a \hat{p}).

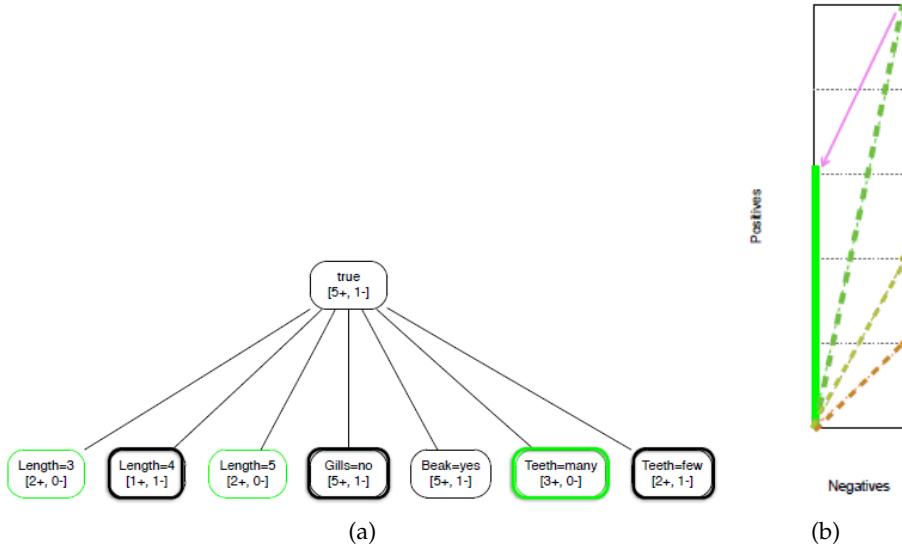


Figura 6.3: Selezione della seconda regola per l'esempio. Come si nota, i 4 esempi negativi individuati dalla prima regola $\text{Gills} = \text{yes}$ non vengono più considerati.

Come annunciato, si seleziona come primo letterale della regola $\text{Gills} = \text{yes}$ e si rimuovono i 4 esempi che tale clausola permette di individuare. Si ripete quindi il procedimento ottenendo la situazione in Figura 6.3a. Il punto del coverage plot che andiamo a selezionare si trova lungo l'asse y (Figura 6.3b) e corrisponde al letterale $\text{Teeth} = \text{many}$.

Spingendosi infine nella selezione della terza (e come vedremo ultima regola) si otterrà la situazione rappresentata nelle Figure 6.4a e 6.4b.

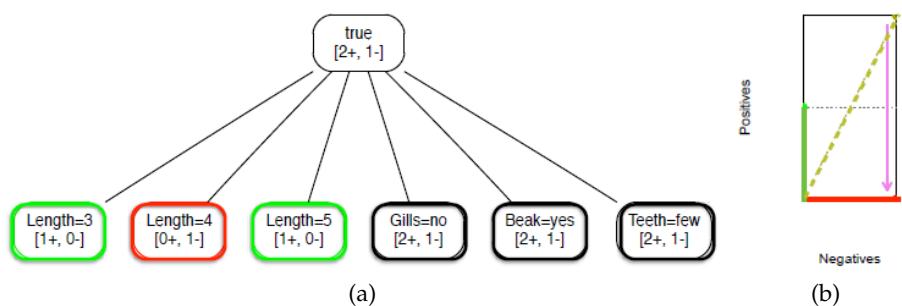


Figura 6.4: Terzo step.

Se si prende in considerazione il letterale $\text{Length} = 5$ ci si accorge che questo riesce a coprire l'ultimo esempio appartenente alla classe negativa, così che i rimanenti esempi (tutti appartenenti alla classe positiva) possano essere riconosciuti con una regola comune. Possiamo allora presentare il modello a regole (ordinate) generato in

questo esempio con le seguenti condizioni (da verificarsi in cascata):

1. if **Gills=yes** then **class= '-'**
2. else if **Teeth='many'** then **class='+'**
3. else if **Lenght=4** then **class = '-'**
4. else **class='+'** \leftarrow default

In questo caso la classe di default in questo esempio copre esattamente tutte le istanze, ma si potrebbe anche decidere di commettere un errore al fine di evitare overfitting.

6.2.3 Algoritmo di generazione delle regole ordinate

Si può allora presentare lo pseudocodice dell'algoritmo di generazione della liste di regole (algoritmo 9).

Algorithm 9 LearnRuleList(D) - Learn an ordered list of rules

Input: labelled training data D
Output: rule list R

```

R  $\leftarrow \emptyset$ 
while D  $\neq \emptyset$  do            $\triangleright$  or a little number of examples are left
    r  $\leftarrow$  LearnRule(D);           $\triangleright$  LearnRule: algorithm 10
    append r to the end of R;
    append r to the end of R;
    D  $\leftarrow x \in D \mid x \text{ is covered by } r$ ;  $\triangleright$  This is why the rule covering
algorithms are called: separate and conquer
end while
return R

```

6.2.4 L'analogia con i feature tree

Si nota che le regole apprese possono essere rappresentate come il percorso di un feature tree, rappresentato in figura 6.5a, e questo, riordinando le foglie rispetto alle loro probabilità empiriche, porta alla costruzione della curva di copertura che si può vedere in blu nel grafico in figura 6.5b. La precision, data dall'area sotto la curva, è ovviamente massima, esattamente come lo era per il modello a regole. Possiamo quindi stabilire che le liste di regole ereditano la proprietà dei decision tree per cui la curva data dal training set è sempre convessa.

6.2.5 Ranking con liste di regole

Utilizzando ancora l'esempio ripreso all'inizio della sezione 6.2.2 si considerano i seguenti concetti: A : Lenght = 4 e B : Beak = yes.

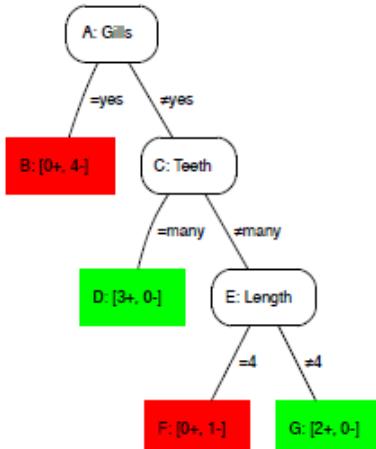
Algorithm 10 LearnRule(D) - Learn a single rule

Input: labelled training data D
Output: rule r

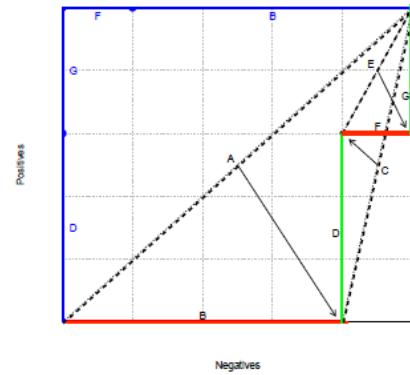
```

b ← true;                                ▷ The initial body of the rule
L ← set of available literals;
while not Homogeneous(D) do ▷ Keep searching for the best path
    l ← BestLiteral(D, L);                ▷ e.g., highest purity;
    b ← b ∧ l;
    D ←  $x \in D \mid x \text{ is covered by } b;$ 
    L ← L  $l' \in L \mid l' \text{ uses same feature as } l;$ 
end while
C ← Label(D);                            ▷ e.g., majority class
r ←
if b then
    Class=C;
return r

```



(a)



(b)

Figura 6.5: Rappresentazione tramite feature tree e rispettivo coverage plot del modello a regole

Il primo concetto copre gli esempi p2, n2, n4, n5 mentre il secondo copre gli esempi p1, p2, p3, p4, p5, n1, n2, n5. Usando questi due concetti come lista di regole nell'ordine AB in cui sono presentate, è possibile costruire la lista seguente:

1. if Length=4 then class='-' $\leftarrow A$
2. else if Beak=yes then class='+' $\leftarrow B \setminus A$
3. else class='-' $\leftarrow \text{default} \leftarrow -$

Si vuole, con questo modello, effettuare un ranking valutando l'errore commesso (in un ranking si commette un errore se un esempio positivo è posto dopo un esempio negativo).

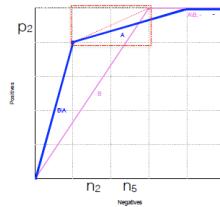


Figura 6.6: Ranking curve prodotto dall'applicazione dei due modelli a regole AB e BA

Se calcoliamo $\dot{p}_A = 1/4$ (probabilità empirica sulla foglia in cui $\text{Length} = 4$), mentre per la seconda regola avremo $\dot{p}_{B \setminus A} = 4/5$. Infine $\dot{p}_- = 0/1$.

Il ranking prodotto dal modello sarà: $\{[p_1, p_3, p_4, p_5, n_1][p_2, n_2, n_4, n_5,]n_3\}$ dove si nota un errore nel punto in cui n_1 è posto prima di p_2 . Voiendo quindi valutare questo modello per l'applicazione del ranking, si applicherà la formula

$$\text{rank_err} = \sum_{x \in T^+, x' \in T^-} \left(\frac{I(s(x) < s(x')) + 0.5 \cdot I(s(x) = s(x'))}{\text{Pos} \cdot \text{Neg}} \right)$$

L'errore totale corrisponde a 7.5.

La curva risultante (riordinando le regole in base alla probabilità empirica, esattamente come è stato spiegato nel capitolo 5 per i decision tree) è quella in figura 6.6: in particolare, in questa figura si vede in realtà la curva prodotta dal modello AB (in blu) e dal modello BA (in viola).

La parte racchiusa al di sotto del segmento in rosso identifica lo spazio di $A \wedge B$ che nessuno dei due modelli riesce a raggiungere.

Possiamo poi comparare i due modelli tenendo conto che il modello migliore è quello che mostrerà la miglior precision, ovvero che ha l'area sotto la curva maggiore.

Vediamo ora meglio il modello in cui l'ordine dei concetti considerato è BA:

1. if **Beak=yes** then **class= '+'** \leftarrow B
2. else if **Length=4** then **class='-'** \leftarrow A \ B
3. else **class='-'** \leftarrow default \leftarrow -

È già stata presentata la curva prodotta da questo modello: questa volta si ha che il primo segmento è prodotto dalla prima regola e cioè che l'ordine delle regole rispetta l'ordine (crescente) delle probabilità empiriche, infatti $\dot{p}_B = 5/8$, $\dot{p}_{A \setminus B} = 0/1$ e $\dot{p}_- = 0/1$. Il ranking che ne consegue è $\{[p_1, p_2, p_3, p_4, p_5, n_1, n_2, n_5][n_4]n_3\}$.

Per quanto riguarda il punteggio (score) del primo segmento relativo alla regola Beak = yes è 1 e per questo si commettono $3 \cdot 5 \cdot 0.5 = 7.5$ errori totali (come il precedente modello). Qual'è quindi il modello migliore? Dipende ovviamente dal peso associato all'errore: se gli

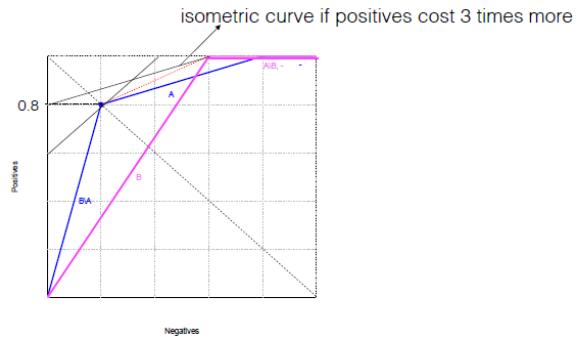


Figura 6.7: Le due curve di copertura a confronto: diversi pesi per gli errori sulle due classi porteranno a scelte diverse

errori sulle due classi peseranno uguali si preferà il primo modello che avrà un true positive rate (precision) dell'80%, altrimenti, se ad esempio si suppone un peso maggiore per gli errori commessi sulla classe positiva, si preferirà il secondo modello (la curva isometrica risulta più inclinata e la operating condition sarà data dal vertice in alto (figura 6.7).

6.3 MODELLI A REGOLE NON ORDINATE

6.3.1 Apprendere insiemi di regole

Verranno ora presentati i modelli basati su regole apprese in modo simile a quanto visto nella sezione 6.2, ma in cui non sono ordinate bensì inserite semplicemente in insiemi.

Per l'apprendimento di insiemi di regole si segue il seguente schema:

1. si seleziona una classe;
2. si seleziona il corpo della regola che copre il maggior numero di classi selezionate al punto 1 in modo che tale questa abbia la maggior precision sulla classe selezionata: $TP/(TP + FP)$; l'ordinamento non è necessario perché ogni regola è "molto precisa" sulla classe che considera.

I due punti verranno ripetuti più volte generando altre regole per altre classi. Non essendoci ordinamento tra le regole, più regole possono coprire esempi di una stessa classe.

6.3.2 Un esempio

Consideriamo ancora una volta l'esempio ricordato alla sezione 6.2.2 e prendiamo in considerazione la regola

if Length = 3 then Class = ⊕

che corrisponde ad una precision di 1 (gli esempi con Lenght = 3 sono 2 e sono solo presenti nella classe positiva, ovvero precision = $\{p_1, p_3\} | /2$).

Si procede quindi non considerando più i due esempi selezionati da questa regola e procediamo a cercarne un'altra: purtroppo però non esistono altre regole che selezionano in modo puro un certo numero di esempi. Si può però sfruttare un secondo livello nello spazio della ricerca dei concetti, utilizzando una congiunzione tra due letterali come antecedente della regola:

```
if Gills = no  $\wedge$  Lenght = 5 then Class =  $\oplus$ 
```

Infine, per coprire anche i rimanenti esempi positivi utilizziamo la regola

```
if Gills = no  $\wedge$  Teeth = many then Class =  $\oplus$ 
```

Si nota, che anche se queste regole possono coprire gli stessi esempi, questi ultimi appartengono tutti alla classe positiva e non è quindi necessario ordinarli.

L'insieme di regole che ricoprono tutti gli esempi della classe positiva è quindi:

- **if** Lenght=3 **then** class='+'
- **if** Gills='no' & Length=5 **then** class='+'
- **if** Gills='no' & Teeth='many' **then** class='+'

6.3.3 L'algoritmo di apprendimento di un insieme di regole

Traendo le conclusioni da quanto visto nella sottosezione 6.3.2 presentiamo lo pseudocodice dell'algoritmo di apprendimento di un insieme non ordinato di regole (algoritmo 11).

Algorithm 11 LearnRuleSet(D) - Learn an unordered set of rules

Input: labelled training data D

Output: rule set R

$R \leftarrow \emptyset$

for every class C_i **do**

$D_i \leftarrow D$;

while D_i contains examples of class C_i **do**

$r \leftarrow \text{LearnRuleForClass}(D_i, C_i)$ \triangleright LearnRuleForClass:

 algorithm 12

$R \leftarrow \bigcup\{r\}$

$D \leftarrow D_i \setminus \{x \in C_i \mid x \text{ is covered by } r\}$; \triangleright Remove only positive

end while

end for

return R

Algorithm 12 LearnRuleSetForClass(D, C_i) - Learn a single rule for a given class

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

```

b ← true
L ← set of available literals;
while not Homogeneous( $D$ ) do
    l ← BestLiteral( $D, L, C_i$ ) ▷ e.g., maximising precision on class
     $C_i$ 
    b ← b ∧ l
    D ←  $x \in D \mid x$  is covered by  $b$ ;
    L ←  $L \setminus l' \in L \mid l'$  uses same feature as  $l$ ;
end while
r ←
if b then
    Class= $C_i$ ; return r

```

6.3.4 L'introduzione di una valutazione probabilistica

Il procedimento visto fin'ora, nonostante come visto sembri infallibile, tende a focalizzarsi un po' troppo sulla precision e sulla ricerca di regole "pure" (considerando quindi anche segmenti molto piccoli), mancando e non considerando regole che potrebbero comunque essere valide in una generalizzazione del problema.

Ad esempio, il procedimento presentato selezionerà la regola Length = 3 prima rispetto alla regola Gills = no, nonostante quest'ultima porterebbe alla regole pura Gills = no \wedge Teeth = many.

Un modo per gestire questo accorgimento è quello di applicare la *Laplace Correction*, che stabilisce l'incremento di un valore su entrambi i calcoli di entrambe le classi, in modo da preferire regole più generali. Un altro modo per ridurre questo fenomeno è quello di utilizzare algoritmi basati su ricerca "beam" (che selezionano k candidati) piuttosto che sulla tecnica greedy vista finora (che seleziona solamente la regola migliore).

Applicando la Laplace Correction all'esempio che si porta avanti, otterremmo la valutazione in Figura 6.8a.

In generale, la formula che permette di valutare la regola i -esima applicando la Laplace Correction sarà:

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

6.3.5 Il ranking tramite un insieme non ordinato di regole

Consideriamo il seguente insieme di regole:

- **if** Length = 4 **then** Class = \ominus $\leftarrow A \rightarrow [1+, 3-] \rightarrow [p2, n2, n4, n5]$

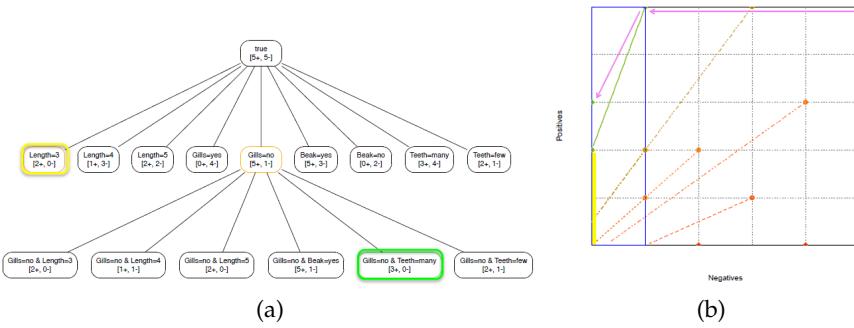


Figura 6.8: In figura 6.8a si mostra il risultato dell'applicazione della Laplace correction all'esempio. Il grafico risultante (figura 6.8b mostra i segmenti a partire dalla nuove origini (-1,-1))

- if **Beak = yes** then Class = $\oplus \leftarrow B \rightarrow [5+, 3-] \rightarrow [p1, p2, p3, p4, p5, n1, n2, n5]$
- if **Length = 5** then Class = $\ominus \leftarrow C \rightarrow [2+, 2-] \rightarrow [p4, p5, n1, n3]$

È ora possibile calcolare la probabilità empirica per ogni regola (a questo punto decidiamo di bypassarla) e possiamo inoltre calcolare le probabilità empirica di una regola senza considerare gli esempi già coperti da altre regole (evidenziando che le regole A e C sono mutualmente esclusive e che quindi possono esistere sovrapposizione solamente nei sottoinsiemi di regole {B, A} e {B, C}). Ovvero:

- $B \setminus \{A + C\} \leftarrow [[p1, p3], \{\}]$
- $A \setminus \{B + C\} \leftarrow [\{\}, \{n4\}]$
- $C \setminus \{A + B\} \leftarrow [\{\}, \{n3\}]$
- $AB \leftarrow [[p2], \{n2, n5\}]$
- $BC \leftarrow [[p4, p5], \{n1\}]$

Nella figura 6.9 è possibile vedere un confronto tra il modello a liste di regole ordinate prodotto nella sottosezione 6.2.5 (in blu) con quello appena ottenuto (in arancione): quest'ultimo si nota che si comporta meglio, in quanto i modelli a insiemi di regole permettono di raggiungere porzioni dello spazio delle istanze che i modelli a liste di regole non permettono di raggiungere.

In un modello con N regole, si possono presentare 2^N possibili sovrapposizioni, e questo rende ovviamente molto complicato individuare quali regole riguardano.

Solitamente si valutano le sovrapposizioni in maniera approssimata. Considerando ad esempio le due regole A e B, possiamo dire che la copertura della regola AB è data dai valori medi di copertura delle due regole, ovvero $[(1+5)/2, ((3+3)/2)]$, che porterà alla probabilità empirica stimata $\hat{p}(AB) = 3/6 = 0.5$. In modo analogo potremmo ottenere $\hat{p}(BC) = 3.5/6 = 0.58$.

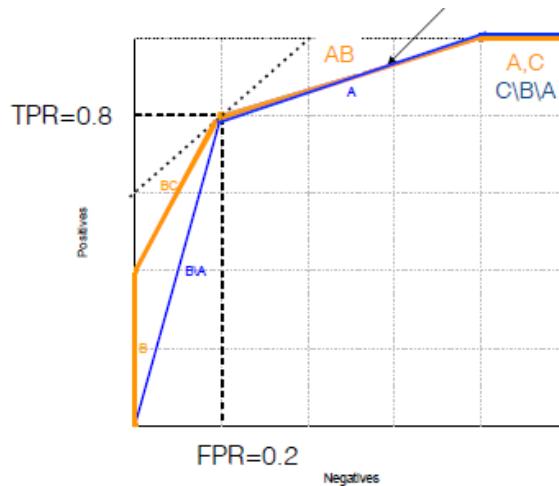


Figura 6.9: Confronto tra un ranker prodotto da un modello a regole ordinate (in blu) ed un modello a regole non ordinate (in arancione)

Il ranking che si ottiene da questa approssimazione è [B, BC, [AB, C], A]. Utilizzando però tali valori approssimati la concavità della curva di copertura non può essere garantita. La forza del modello a regole non ordinate è però che permette di spezzare lo spazio delle ipotesi in 2^N segmenti, mentre un modello a liste di regole con N regole è in grado di costruire solamente N segmenti.

6.3.6 *La predizione*

Così come in fase di training, anche in fase di test ad un esempio può corrispondere più di una regola, ciascuna delle quali porta a concludere una classe diversa. Come selezionare la classe migliore? Intuitivamente possiamo decidere di selezionare la classe conseguente alla regole con la maggiore copertura (eventualmente considerando la stima di copertura media): una copertura più ampia permette una migliore generalizzazione.

6.4 APPRENDIMENTO DI REGOLE DESCRITTIVE

6.4.1 *Subgroup discovery*

La prima applicazione descrittiva che si vuole introdurre è particolarmente popolare in molti ambiti, in particolare in quello medico: si tratta della cosiddetta *Subgroup discovery*.

Si tratta della produzione di ipotesi, verificate in sottoinsiemi dello spazio delle istanze (*Subgroups*), che si spera siano poi utilizzate per effettuare analisi relazionali tra il sottogruppo e l'intera popolazione (per esempio da una analisi dei dati sulla popolazione può derivare

una relazione tra il sottogruppo degli ammalati e coloro che consumano carne e che fumano).

Si è quindi interessati nella descrizione, nella produzione di numerose ipotesi valide in molti sottogruppi della popolazione considerata. Più formalmente un sottogruppo è una funzione di mapping $\hat{g} : X \rightarrow \{\text{true}, \text{false}\}$ appresi da un insieme di esempi $(x_i, l(x_i))$ dove l è una funzione che associa all'esempio la vera label della classe a cui appartiene.

Sono possibili molti sottogruppi, ma i migliori sono quelli in cui la distribuzione delle classi è statisticamente differente dalla popolazione d'origine.

Dall'utilizzo di sottogruppi si riesce a discostarsi dalla semplice definizione di impurità: si può ad esempio considerare il complementare di un sottogruppo. Riprendendo l'esempio della classificazione dei delfini, si nota che per i valori Gills = yes si hanno 4 esempi negativi e nessun positivo, mentre il suo complementare Gills = no copre tutti i positivi e solamente un esempio negativo.

Ma se non si considerano più le misure di impurità occorre cercare altre misure.

Come prima, si è ora interessati all'identificazione di istanze di segmenti, ma i modelli descrittivi non si concentrano tanto sulla predizione della classe quanto sulla descrizioni di insiemi omogenei di istanze.

Nel grafico di copertura, si può (come per i concetti) disegnare gli esempi positivi riferendoci all'asse y e quelli negativi sull'asse x : i sottogruppi con esempi disposti lungo la diagonale sono poco interessanti poiché hanno deviazione nulla rispetto alla popolazione d'origine. Una misura di qualità di un sottogruppo è quella di considerare una euristica tra quelle già conosciute, ad esempio la precision, e misurarne lo scostamento dalla stessa misura applicata alla popolazione d'origine. In realtà la precision tende a valutare migliore il segmento che copre più positivi e meno negativi, ma questo non permetterebbe una valutazione paritaria dei sottogruppi complementari: meglio l'average recall!

L'average recall ricordiamo essere definita come $(\text{tpr} + \text{tnr})/2$ e nella diagonale assumerà valore 0.5 (valore dal quale dovremo, secondo quanto appena detto, calcolare la deviazione dei sottogruppi). In alcuni casi è possibile notare come l'average recall e la precision non propendono per la medesima scelta sul miglior sottogruppo.

Focalizzandosi ad esempio sulla figura 6.10 si nota come la precision a cui è stata applicata la correzione di Laplace (Figura 6.10a) spesso si discosta parecchio dall'average recall (figura 6.10b): mentre utilizzando la prima misura si ottiene che i sottogruppi Gills = no \wedge Teeth = many e Gills = yes sono i migliori, con l'average recall assume maggior rilevanza il sottogruppo identificato da Beak = yes (migliore del primo sottogruppo e buono quanto il secondo).

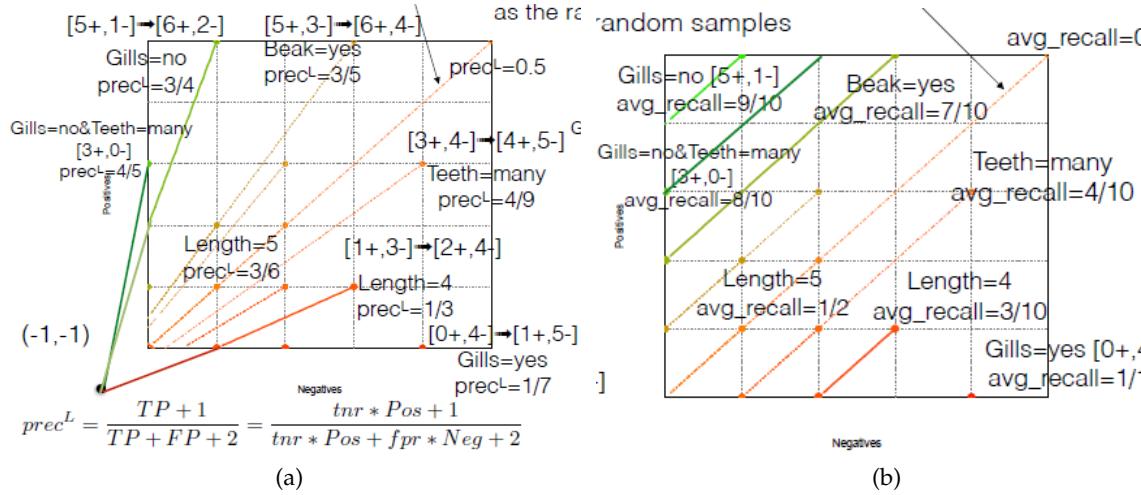


Figura 6.10: Confronto tra le valutazioni basate su average recall e precision

Si nota però, sempre dalla figura 6.10 che i sottogruppi posti nel mezzo non subiscono grosse variazioni in termini di ranking tra le due misure (la diagonale simboleggia per entrambe i sottogruppi meno interessanti).

Notiamo inoltre che con l'average recall abbiamo segmenti tutti paralleli alla diagonale: questo perchè avendo sull'asse delle x il tnr e sull'asse delle y il fpr. L'average recall può essere scritta come:

$$\text{avg_recall} = \frac{\text{tpr} + \text{tnr}}{2} = \frac{\text{tpr} + 1 - \text{fpr}}{2}$$

e quindi

$$\text{tpr} = \text{fpr} + (2 \cdot \text{avg_recall} - 1)$$

Le rette che connettono due punti che rappresentano sottogruppi con la stessa average recall hanno una pendenza di 1 (cioè average recall costante) e sono parallele alla diagonale.

6.4.2 Un esempio di valutazione dei sottogruppi

Immaginiamo di avere dieci sottogruppi creati a partire dal solito esempio, presentati in tabella 6.11, e valutati sia tramite la precision con Laplace correction che con average recall. La prima differenza che notiamo è che il ranking è differente nelle due misure: ad esempio $\text{Gills} = \text{no} \wedge \text{Teeth} = \text{many}$ ha un ranking maggiore rispetto a $\text{Gills} = \text{no}$ valutando la precision, mentre sono in ordine opposto rispetto all'average recall ($\text{Gills} = \text{no} \wedge \text{Teeth} = \text{many}$ ha però posizione corrispondente al complementare di $\text{Gills} = \text{no}$, cioè $\text{Gills} = \text{yes}$). Usando la precision con Laplace correction possiamo valutare la qualità del sottogruppo come $|prec^L - pos|$ (dove pos è la proporzione dei positivi sul totale degli esempi, nel nostro caso $5/10 = 0.5$)

Subgroup	Coverage	$prec^L$	Rank	avg-rec	Rank
Gills = yes	[0+, 4-]	0.17	1	0.10	1-2
Gills = no \wedge Teeth = many	[3+, 0-]	0.80	2	0.80	3
Gills = no	[5+, 1-]	0.75	3-9	0.90	1-2
Beak = no	[0+, 2-]	0.25	3-9	0.30	4-11
Gills = yes \wedge Beak = yes	[0+, 2-]	0.25	3-9	0.30	4-11
Length = 3	[2+, 0-]	0.75	3-9	0.70	4-11
Length = 4 \wedge Gills = yes	[0+, 2-]	0.25	3-9	0.30	4-11
Length = 5 \wedge Gills = no	[2+, 0-]	0.75	3-9	0.70	4-11
Length = 5 \wedge Gills = yes	[0+, 2-]	0.25	3-9	0.30	4-11
Length = 4	[1+, 3-]	0.33	10	0.30	4-11
Beak = yes	[5+, 3-]	0.60	11	0.70	4-11

Figura 6.11: Esempio di ranking prodotti su sottogruppi seguendo sia la misurazione della precision (con Laplace correction) che con average recall

mentre, con l'average recall, potremmo fare lo stesso con la formula $|avg_{rec} - 0.5|$.

6.4.3 Algoritmo per apprendere più regole

Essendo, come detto, interessati ad effettuare delle descrizioni, si sarà interessati ad avere più ipotesi che ricoprono un singolo esempio (e non solo un sottogruppo per esso): siamo cioè interessati ad avere più regole che ricoprono e ci permettono di descrivere "sotto diversi punti di vista" uno stesso esempio.

Un modo per apprendere più regole è quello di non eliminare gli esempi già coperti ma di decrementare un peso a loro associato mano a mano che vengono individuati (per esempio inizializzato ad 1 e dimezzato ogni volta che l'esempio è coperto da una nuova regola). Verrà inoltre utilizzata una euristica di ricerca valutata nei termini di una certa somma tra i pesi degli esempi piuttosto che sul loro numero.

Possiamo vedere lo pseudocodice dell'algoritmo che implementa quanto appena esplicato nel codice 13.

L'algoritmo LearnRule è la stessa vista nella sottosezione 6.2.3.

6.4.4 Items set

Si introdurranno ora regole che associano insiemi di oggetti diversi a feature booleane che ne identificano la presenza o meno.

Si porta subito un caso pratico in cui si vuole analizzare il comportamento dei clienti di un certo supermercato dal punto di vista della frequenza di acquisto di determinati oggetti (beni). Oggetto di studio in questo dominio saranno quindi le transazioni, di cui riportiamo un caso specifico in figura 6.12.

Ovviamente, per ogni oggetto è possibile elencare le transazioni che ne hanno riguardato l'acquisto: per esempio è possibile dire che sono stati acquistati pannolini nelle transazioni 1, 3, 4 e 7. È possibile

Algorithm 13 WeightedCovering(D) - Learn overlapping rules by weighting examples

Input: labelled training data D with instance weights initialised to 1
Output: rule list R
 $R \leftarrow \emptyset$
while some examples in D have weight 1 **do** ▷ Chiedendo che non esistono esempi con peso a 1 si chiede che tutti gli esempi siano coperti da almeno una regola: in realtà si potrebbe rilassare questa condizione di terminazione per evitare overfittig
 append r to the end of R
 decrease the weights of examples covered by r; ▷ La regola di decremento del peso è lasciata volontariamente libera
end while
return r

Algorithm 14 LearnRule(D) - Learn a single rule

Input: labelled training data D
Output: rule r
 $b \leftarrow \text{true}$; ▷ The initial body of the rule
 $L \leftarrow \text{set of available literals};$
while not Homogeneous(D) **do** ▷ Keep searching for the best path
 $l \leftarrow \text{BestLiteral}(D, L)$; ▷ e.g., highest purity;
 $b \leftarrow b \wedge l$;
 $D \leftarrow x \in D \mid x \text{ is covered by } b$;
 $L \leftarrow L \setminus l' \in L \mid l' \text{ uses same feature as } l$;
end while
 $C \leftarrow \text{Label}(D)$; ▷ e.g., majority class
 $r \leftarrow$
if b **then**
 Class=C;
return r

<i>Transaction</i>	<i>Items</i>
1	nappies
2	beer, crisps
3	apples, nappies
4	beer, crisps, nappies
5	apples
6	apples, beer, crisps, nappies
7	apples, crisps
8	crisps

Figura 6.12: Esempio di transazioni (identificate dal numero nella prima colonna), ciascuno con un insieme di beni acquistati (seconda colonna)

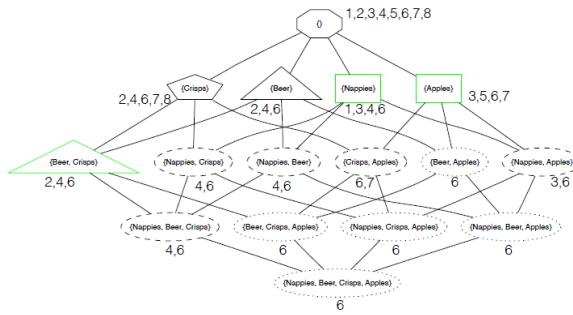


Figura 6.13: Esempio di grafo indicante il supporto per ogni insieme di oggetti

anche farlo per insiemi di oggetti: l'acquisto congiunto di birra e patatine ha riguardato le transazioni 2, 4 e 6 e si dirà allora che l'insieme {beer, crisps} copre l'insieme di transazioni {2, 4, 6}. La cardinalità dell'insieme di transazioni coperte è chiamato *supporto*, che è appunto una misura di valutazione della copertura di un insieme di oggetti. È possibile rappresentare l'insieme di oggetti in un grafo in cui gli insiemi di oggetti che coprono una singola transazione sono indicati con ovali puntinati, quelli che ne coprono due in ovali tratteggiati, quelli che ne ricoprono tre sono evidenziati con un triangolo e in generale quelli che ne coprono n con poligoni con n lati.

Gli insiemi con cardinalità massima e con supporto 3 o più sono indicati in verde nella figura di esempio 6.13: spesso un supporto minimo è dato dall'analista per effettuare un pruning su un problema di ricerca altrimenti enorme.

Si aggiunge ora un po' di formalismo, introducendo un po' di notazione.

- Un **item set** I è un insieme di oggetti $\{i_1, \dots, i_n\}$ tale che ogni $i_x \in I$ appartiene alla stessa transazione t .
- Un **item set frequente** I è un item set $\{i_1, \dots, i_n\}$ tale che ogni $i_x \in I$ appartiene ad un insieme di transazioni $T = \{t_1, \dots, t_m\}$ con $|T| \geq f_0$ (supporto minimo stabilito).
- Un **item set chiuso** I è un item set tale che non esiste un altro item set $I_2 \subseteq I$ con lo stesso supporto ($f(I_2) = f(I)$).
- Un **item set massimale** I è un item set frequente e chiuso tale che non esiste un altro item set I_2 t.c. $I \subset I_2$ con un supporto sufficiente.

In altre parole gli item set massimali sono i vertici posti al bordo inferiore del grafo, e quindi separano gli item set frequenti da quelli non frequenti (in alto del bordo si trovano solo vertici con frequenza sufficiente e sotto gli altri).

Nell'esempio in figura 6.13 si nota i vertici in verde essere i massimali: l'item set {Beer} non è massimo pur avendo un supporto

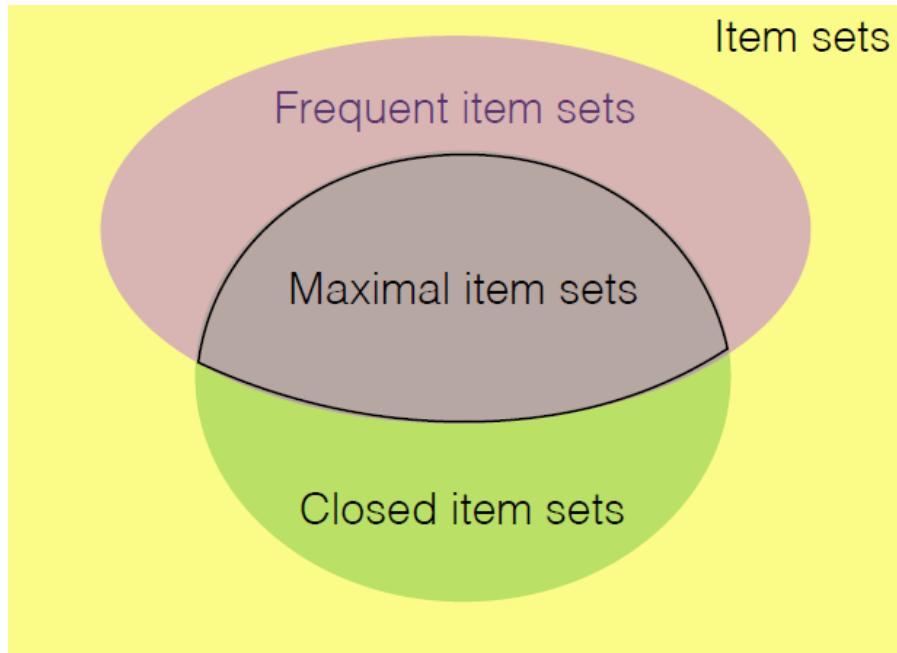


Figura 6.14: Rappresentazione con diagramma di Venn delle varie sottocategorie di item set

pari a 3 come {Beer, Crisps} ed essendo quindi un item set frequente: {Beer, Crisps} ha un item in più e quindi ha la massima cardinalità. Discorso simile per {Crisps}.

Si può meglio evidenziare la relazione tra queste sottocategorie di item set grazie al diagramma di Venn di in figura 6.14.

Si evidenziano inoltre, guardando sempre al grafo, le seguenti proprietà:

- scendendo di livello il supporto decresce: intuitivamente considerando meno item è meno probabile che compaiano tutti insieme dello stesso insieme di transazioni.
Il supporto è allora *monotono*, ovvero muovendosi verso il basso nel grafo il suo valore non cresce mai;
- l'insieme degli item set frequenti è *convesso*¹;
- la proprietà precedente permette di cercare tutti gli item set frequenti tramite un algoritmo di ricerca in ampiezza o in profondità che attraversa il grafo.

Si è interessati all'item set massimale perché esso rappresenta una sorta di generalizzazione degli item set, proprio perché posto come confine.

¹ Dati due membri S_1 ed S_2 di un insieme (convesso) tali che $S_1 \subseteq S_2$ ogni altro insieme S_x tale che $S_1 \supseteq S_x \supseteq S_2$ appartiene all'insieme (convesso).

Si può presentare allora l'algoritmo 15 che consente di cercare l'item set massimale (ovvero l'item set frequente con cardinalità massima). È quindi chiaro che il grafo cresce esponenzialmente nel numero dei

Algorithm 15 FrequentItems(D, f_0) - find all maximal item sets exceeding a given support threshold

Input: data $D \subseteq X$; support threshold $f_0 \triangleright$ Gli elementi di D non sono labelled poichè non occorre un apprendimento "supervisionato" ma serve solo individuare item set frequenti

Output: set of maximal frequent item sets M

$M \leftarrow \emptyset$

Initialise priority queue Q to contain the empty item set $\triangleright Q$

contiene l'item set vuoto, quindi la coda non è vuota

while Q is not empty **do**

$I \leftarrow$ next item set deleted from front of Q

 max \leftarrow true; \triangleright flag to indicate whether I is maximal

for each possible superset I' of I **do**

if $\text{Supp}(I') \geq f_0$ **then**

 max \leftarrow false; \triangleright // frequent superset found, so I is not maximal

 add I' to back of Q ;

end if

end for

if max = true **then**

$M \leftarrow M \cup \{I\}$

end if

end while

return M

nodi e l'obiettivo è fornire l'insieme di item set frequenti. Se si considerano solo gli item set massimali, applicando la proprietà della monotonicità del supporto, si può dedurre quali sono gli item set frequenti a partire dai massimali: non si arriverà però a conoscere l'esatta copertura (e quindi il supporto) degli item set frequenti.

Se si considerano gli item set chiusi è invece possibile conoscere il loro supporto. Si evidenzia allora meglio le differenze tra item set frequenti, chiusi e massimali.

- Gli item set chiusi rappresentano classi di equivalenza di item set con lo stesso supporto.
- L'estrazione degli item set frequenti non produce perdita di informazione per quanto riguarda il numero e la frequenza degli item set con supporto sufficiente ma causa un volume di risultati molto elevato (il più grande possibile)

- L'estrazione degli item set massimali riduce di molto la mole, ma al prezzo di una perdita di informazione sulla frequenza degli item frequenti ma non massimali
- Considerando gli item set chiusi può spesso risultare il giusto compromesso. Così facendo è possibile recuperare l'informazione di un item set I_1 frequente ma non incluso tra i risultati (perchè non è chiuso) cercando tra gli item set chiusi trovati l'item set $I_2 \supseteq I_1$ con supporto maggiore: I_2 e I_1 avranno lo stesso supporto.

6.4.5 Association rules

Gli item set frequenti possono essere sfruttati per la costruzione di *association rules*, ovvero regole della forma

if B then H

dove B e H sono item set che appaiono frequentemente assieme nelle transazioni.

Se si considera ad esempio l'arco che connette {beer} e {nappies, beer} del grafo di esempio (figura 6.13) si può vedere che i rispettivi supporti corrispondono a 3 e 2. Si dirà che la *confidenza* della regola di associazione **if beer then nappies** equivale a $2/3$.

In pratica questa è la misura di quanto si può essere fiduciosi circa la presenza del conseguente in transazioni che contengono l'antecedente.

Analogamente la confidenza della regola **if nappies then beer** equivale a $2/4$.

Alcune regole hanno confidenza piena, cioè equivalente a 1 (come la regola **if beer then crisps**) e sono esattamente le regole ottenute a partire da insiemi chiusi perchè il sottoinsieme ha lo stesso supporto del superset e quindi il rapporto sarà uguale ad 1 (questo suggerisce un'altra motivazione di importanza degli item set chiusi).

Altre regole possono avere un corpo vuoto (come ad esempio **if true then crisps** che ha confidenza $5/8$, cioè 5 transazioni su 8 riguardano le patatine).

6.4.5.1 L'algoritmo di generazione delle association rules

Come detto, l'interessato è rivolto alle regole di associazione tra item frequenti: la regola **if beer \wedge apple then crisps** ha confidenza pari a 1 ma solo in una transazione considera tutti e tre gli item set, quindi non è una regola interessante. Si userà allora l'algoritmo 15 per cercare gli item set frequenti e poi si selezionerà il corpo e la testa delle regole da ciascun insieme frequente m, escludendo le regole con confidenza al di sotto di una certa soglia. Questa procedură è schematizzata dallo pseudocodice dell'Algoritmo 16.

Algorithm 16 AssociationRules(D, f_0, c_0) - find all association rules exceeding given support and confidence threshold

Input: data $D \subseteq X$; confidence threshold c_0
Output: set of association rules R

$$R \leftarrow \emptyset$$

$$M \leftarrow \text{FrequentItems}(D, f_0)$$

for each $m \in M$ **do**

for each $H \subseteq m$ and $B \subseteq m$ such that $H \cap B = \emptyset$ **do**

if $\text{Supp}(B \cap H) / \text{Supp}(B) \geq c_0$ **then**

$R \leftarrow R \cup \text{if } B \text{ then } H$

end if

end for

end for

return R

6.4.5.2 Una possibile ottimizzazione dell'algoritmo

Si può pensare ad una ottimizzazione dell'algoritmo appena introdotto (Algoritmo 16) basata sulla seguente osservazione.

Invece di considerare la valutazione di ogni possibile sottoinsieme H accoppiato con ogni altro possibile sottoinsieme B con intersezione nulla in ogni ordine, si può considerare un ordine sulla cardinalità di H (prima i sottoinsiemi con la cardinalità minore). Così facendo si pone nel corpo della regola l'insieme con cardinalità massima: dati due teste $H_2 \subset H_1$ (H_1 ha cardinalità minore di H_2) originate a partire dallo stesso item set frequente I , conviene prima valutare la confidenza della regola $B_1 \rightarrow H_1$ con $B_1 = I - H_1$ e poi se questa confidenza è sufficiente si può estrarre $B_2 \rightarrow H_2$ con $B_2 = I - H_2$. Questo perché B_1 e B_2 hanno cardinalità diverse (B_2 ha cardinalità minore e quindi un supporto maggiore che siccome è a denominatore darà una confidenza minore). Quindi conviene cominciare valutando le regole con confidenza maggiore e non appena la confidenza comincia a scendere al di sotto della soglia stabilita si può arrestare la ricerca, in modo da non creare association rule inutili.

6.4.5.3 Post-processing

Molto spesso non è possibile effettuare un filtro sulla bontà delle regole ma si vuole ugualmente valutare la association rule ottenuta e una misura molto utile a questa valutazione è la *lift*, definita come segue:

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

con n che indica il numero di transazioni.

Alcune proprietà interessanti di questa misurazione sono, per esempio:

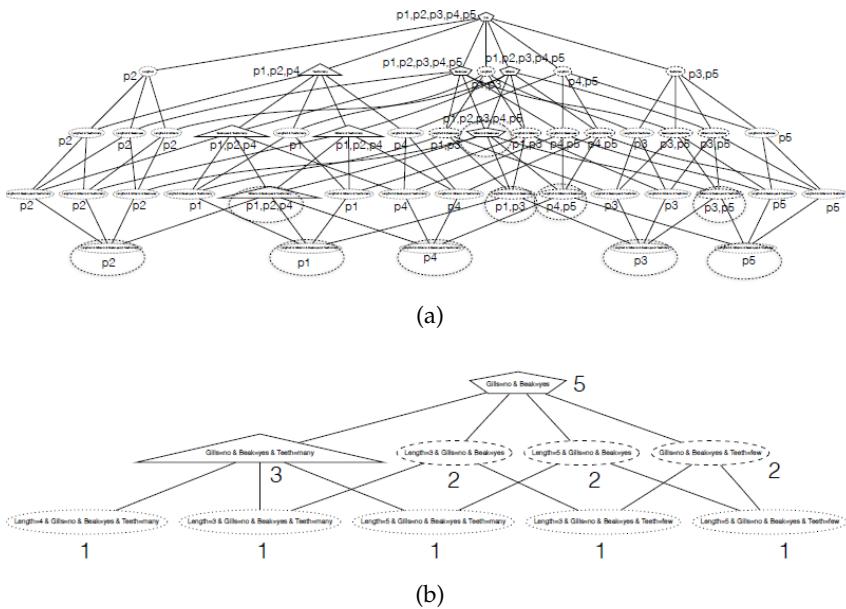


Figura 6.15: Confronto tra grafi ottenuti con e senza tener conto dei soli item set chiusi

- $\text{Lift}(\text{if } B \text{ then } H) = \text{Lift}(\text{if } H \text{ then } B)$
- $\text{Lift}(\text{if } \emptyset \text{ then } H) = \frac{n \cdot \text{Supp}(\emptyset \cup H)}{\text{Supp}(\emptyset) \cdot \text{Supp}(H)} = \frac{n \cdot \text{Supp}(H)}{n \cdot \text{Supp}(H)} = 1$

Riguardo al secondo punto, possiamo dire più in generale che una lift = 1 significa che $\text{Supp}(B \cup H)$ è interamente determinato dalle frequenze marginali $\text{Supp}(B)$ e $\text{Supp}(H)$ (cioè B e H sono statisticamente indipendenti). Si è solitamente interessati a lift maggiori di uno.

Si vuole ancora mostrare che, se si utilizzano item set chiusi nel corpo e nella testa delle association rules si ottengono regole più significative (ricordiamo inoltre che gli item set chiusi possono rappresentare una sorta di riduzione dell'intero insieme di item set).

Ad esempio, si preferirà la regola **if {nappies, beer, crisps} then {apples}** alle regole **if {nappies, beer} then {apples}** e **if {nappies, crisps} then {apples}** nonostante abbiano tutte e tre support = 1 e confidence = 0.5, poiché è più informativa. È quindi possibile non considerare le due regole meno informative.

Nella figura 6.15a si vede un grafo prodotto considerando gli esempi positivi dell'esempio della classificazione dei delfini (ogni item è un letterale del tipo Feature = valore). Il grafo in figura 6.15b è equivalente, ma filtrato solo sugli item set chiusi.

7

MODELLO LINEARI

I diversi modelli lineari che vengono affrontati in questo capitolo sono principalmente tre:

- regressione;
- Support Vector Machine ([SVM](#));
- kernel.

A questi si aggiunge anche il percepitrone, che però non verrà affrontato in questi appunti.

7.1 REGRESSIONE

Un modello lineare di regressione, definito da un punto di vista geometrico, è una retta su un piano che cerca di approssimare nel migliore dei modi un insieme di punti di coordinate (x, y) .

Il problema che si cerca di risolvere, quindi, è una regressione nella sua versione più semplice dove si ha:

- una singola variabile $X \in \mathbb{R}$ in input;
- una singola variabile $Y \in \mathbb{R}$ in output.

Questi due vettori non sono altro che le coordinate (x, y) di tutti i punti che il modello lineare deve includere; il compito della regressione quindi è trovare i valori C e D per definire l'equazione della retta:

$$Cx + D = y \quad (7.1)$$

Per fare un esempio si faccia riferimento alla Figura [7.1](#) dove si identifica facilmente la retta che meglio approssima tutti i punti elencati nella Tabella [7.1](#).

Nel tentativo di trovare la retta che tocca tutti i punti presenti sul piano si ricorre all'equazione sopra riportata, la quale viene applicata a ciascun punto per ottenere una sistema come segue:

$$\begin{cases} Cx_1 + D = y_1 \\ Cx_2 + D = y_2 \\ \vdots \\ Cx_n + D = y_n \end{cases} \quad (7.2)$$

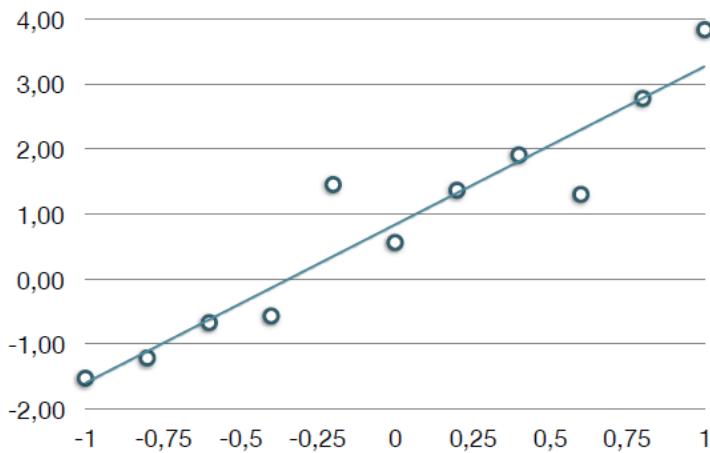


Figura 7.1: Un esempio di modello lineare, ovvero la retta che meglio approssima tutti i punti sul piano.

x	y
-1	-1.52
-0.8	-1.21
-0.6	-0.67
-0.4	-0.56
-0.2	1.46
0	0.57
0.2	1.37
0.4	1.91
0.6	1.31
0.8	2.78
1	3.84

Tabella 7.1: Coordinate (x, y) dei punti riportati nell'esempio in Figura 7.1.

Quasi sempre è praticamente impossibile che un sistema con un numero di equazioni notevolmente superiore al numero di incognite sia risolvibile, tuttavia questo sistema di equazioni può essere formalizzato in notazione matriciale, più compatta ed omogenea:

$$\mathbf{X}\mathbf{w} = \mathbf{y} \quad (7.3)$$

Nella matrice \mathbf{X} sono riportati i coefficienti di C e D; \mathbf{w} rappresenta i parametri che meglio approssimano la retta, ovvero proprio C e D, e la sua moltiplicazione per la matrice \mathbf{X} deve fornire come risultato il vettore \mathbf{y} .

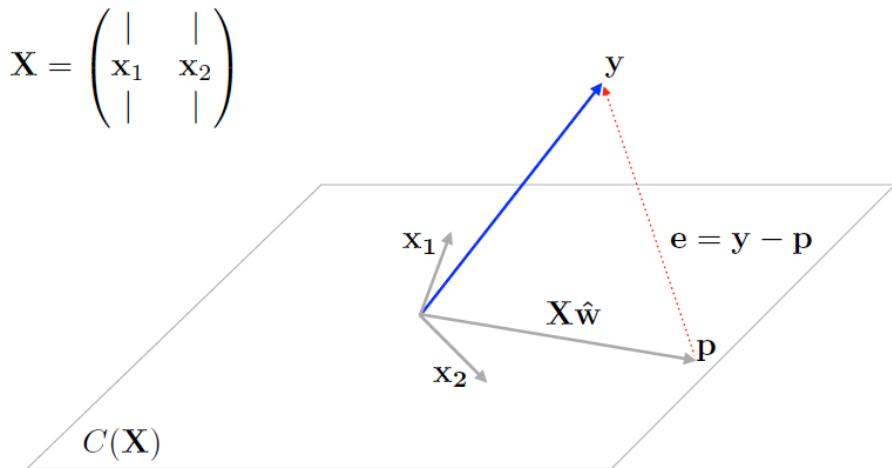


Figura 7.2: Interpretazione geometrica del prodotto $\mathbf{X}\hat{\mathbf{w}}$ e dell'errore \mathbf{e} commesso rispetto all'output atteso \mathbf{y} .

Perciò il sistema dell'esempio diventa:

$$\begin{bmatrix} -1 & 1 \\ -0.8 & 1 \\ \vdots & \vdots \\ 1 & 1 \end{bmatrix} \begin{bmatrix} C \\ D \end{bmatrix} = \begin{bmatrix} -1.52 \\ -1.21 \\ \vdots \\ 3.84 \end{bmatrix} \quad (7.4)$$

Per trovare i valori in \mathbf{w} si potrebbe provare la strada più semplice e immediata:

$$\mathbf{w} = \mathbf{X}^{-1} \mathbf{y} \quad (7.5)$$

L'operazione \mathbf{X}^{-1} , ovvero l'inversione della matrice, non si può applicare perché \mathbf{X} non è invertibile¹. Lo sarebbe se ci fossero tante equazioni quante variabili.

L'impossibilità di trovare una soluzione può essere visualizzata geometricamente in Figura 7.2 dove si nota, appunto, che i vettori x_1 , x_2 e \mathbf{p} , quest'ultimo frutto del prodotto $\mathbf{X}\hat{\mathbf{w}}$ ed approssimazione della soluzione \mathbf{y} :

$$\mathbf{p} = \mathbf{X}\hat{\mathbf{w}} \quad (7.6)$$

appartengono al piano $C(\mathbf{X})^2$, mentre il vettore \mathbf{y} al quale ci si vuole avvicinare per ottenere il modello con la migliore approssimazione non giace sullo stesso piano.

Sempre in riferimento alla Figura 7.2 si noti come una variazione di $\hat{\mathbf{w}}$ produca sempre vettori che appartengono al piano $C(\mathbf{X})$, quindi si evince chiaramente che è impossibile ottenere un vettore uguale ad

¹ Una matrice deve essere quadrata per poter essere invertibile.

² Il vettore \mathbf{p} appartiene al piano $C(\mathbf{X})$ perché è una combinazione lineare dei vettori x_1 e x_2 .

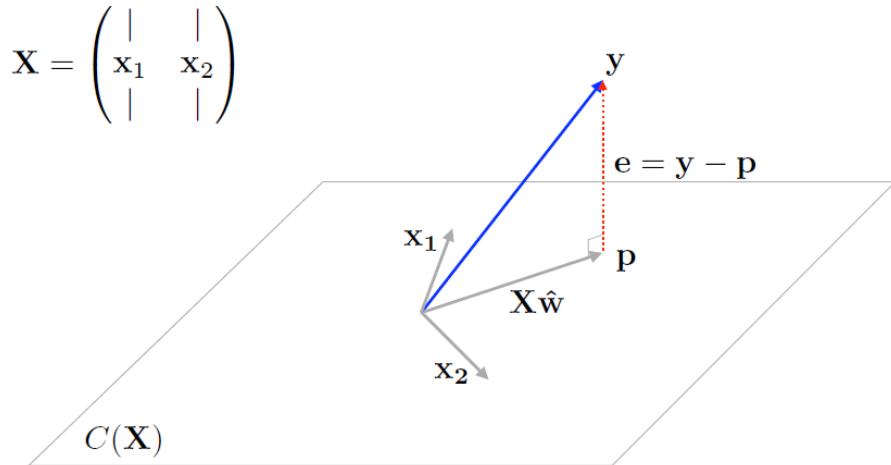


Figura 7.3: Interpretazione geometrica della minimizzazione dell'errore \mathbf{e} , che deve quindi essere ortogonale al piano $C(\mathbf{X})$, commesso rispetto all'output atteso \mathbf{y} .

\mathbf{y} , ma solo un risultato che vi si avvicini. A tale scopo si vuole quindi misurare la distanza fra i vettori \mathbf{p} e \mathbf{y} , questa viene identificata come:

$$\mathbf{e} = \mathbf{y} - \mathbf{p} \quad (7.7)$$

7.1.1 *Metodo dei minimi quadrati*

A questo punto l'obiettivo è trovare il vettore \mathbf{p} tale per cui la distanza dal vettore \mathbf{y} , ovvero \mathbf{e} , è minima. Tale distanza corrisponde alla norma 2 di \mathbf{e} :

$$\|\mathbf{e}\|_2 = \|\mathbf{y} - \mathbf{p}\|_2 = \sqrt{\sum_{i=1}^n (y_i - p_i)^2} \quad (7.8)$$

A questo punto il problema di trovare il modello lineare per l'insieme di punti può essere formulato come la minimizzazione della norma 2 di \mathbf{e} , che è equivalente alla minimizzazione della somma delle differenze dei quadrati delle componenti di \mathbf{y} e \mathbf{p} ³.

$$\text{minimize}_{\hat{\mathbf{w}}} \|\mathbf{X}\hat{\mathbf{w}} - \mathbf{y}\|_2^2 \quad (7.9)$$

Da un punto di vista geometrico questo vuol dire che il vettore \mathbf{e} sia perpendicolare al vettore \mathbf{p} , proprio come si può vedere nella Figura 7.3. Questo si traduce nell'imposizione della seguente condizione,

³ Da qui il nome del metodo dei minimi quadrati.

ovvero che il prodotto scalare dei due vettori sia nullo, con una serie di implicazioni dalle quali si ottiene $\hat{\mathbf{w}}$:

$$\mathbf{X}^T \mathbf{e} = 0 \quad (7.10)$$

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

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

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

$$\Rightarrow \hat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \quad (7.11)$$

Il risultato ottenuto con la Formula 7.11 rappresenta la soluzione del metodo dei minimi quadrati per ottenere il miglior vettore dei pesi $\hat{\mathbf{w}}$ per approssimare efficacemente l'output desiderato \mathbf{y} .

Differentemente da quanto avveniva nella Formula 7.5, dove era impossibile fare l'operazione di inversione della matrice \mathbf{X} , qua invece è possibile farlo perché $\mathbf{X}^T \mathbf{X}$ è quadrata.

7.1.2 Overfitting e regressione regolarizzata

Contrariamente alle aspettative, anche un modello semplice come quello lineare ottenuto dalla regressione può soffrire di problemi legati all'overfitting. In questo caso la causa da ricercare risiede in alcuni punti del dataset che si discostano notevolmente dalla media del dataset stesso.

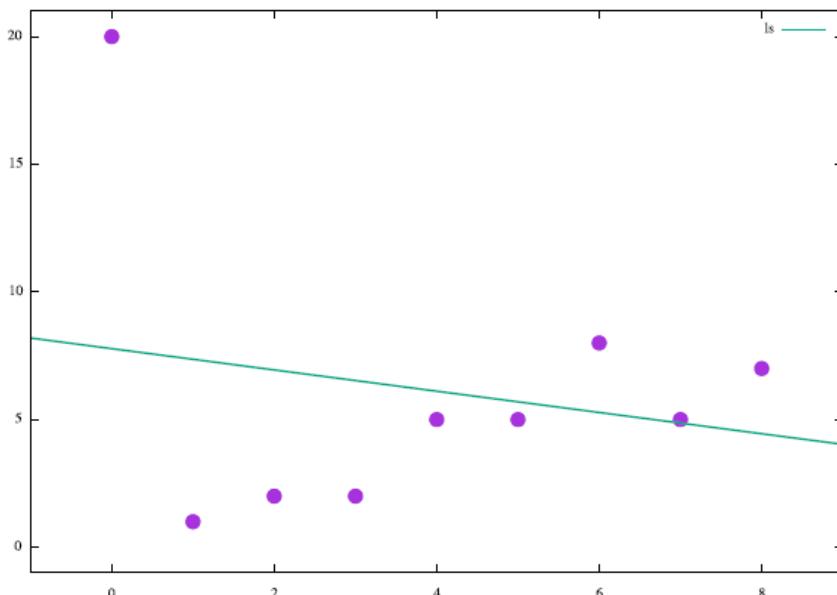


Figura 7.4: Un esempio di modello lineare ottenuto con il metodo dei minimi quadrati. Si nota come il punto di coordinate (0, 20), che è evidentemente discostato da tutti gli altri, influenza negativamente la regressione la quale finisce con il generare un modello distorto rispetto alla distribuzione della quasi totalità dei punti.

Tali punti particolarmente distanti da tutti gli altri vengono chiamati *outliers* e se ne può vedere un esempio nella Figura 7.4, dove la coppia $(0, 20)$ è effettivamente più lontana dalla maggior parte degli altri punti presenti sul piano.

Ovviamente si desidera che il modello sia robusto rispetto all'eventuale presenza di outliers, quindi che ne tenga meno conto al fine di ottenere un risultato finale che generalizzi adeguatamente, che non si discosti dalla maggior parte dei punti per cercare di includere quelli più distanti.

Il metodo dei minimi quadrati non è robusto rispetto a questa problematica. Se si va a considerare la funzione di loss utilizzata per la regressione, una funzione quadratica, è chiaro come gli outliers, avendo un errore che cresce in modo quadratico in proporzione alla distanza dal modello, influiscono pesantemente sul modello stesso, molto più di tutti i punti che invece non mostrano anomalie nella loro distribuzione.

La soluzione che si adotta per contrastare il fenomeno dell'overfitting nel metodo dei minimi quadrati è l'adozione della *regressione regolarizzata*. La regolarizzazione è un metodo generale per evitare l'overfitting rendendo robuste le funzioni di regressione nei confronti degli outliers e può essere declinata in diversi modi in base alle esigenze e al dataset.

7.1.2.1 Regressione ridge

Questo nuovo approccio che applica la regolarizzazione alla regressione con il metodo dei minimi quadrati parte dalla Formula 7.9 e la riscrive nel seguente modo:

$$\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2 \quad (7.12)$$

Il fattore $\|\mathbf{X}\widehat{\mathbf{w}} - \mathbf{y}\|_2^2$ della Formula 7.9 viene riscritto in modo equivalente come $(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})$; a questa riformulazione viene solo aggiunto il termine $\lambda \|\mathbf{w}\|_2^2$ che penalizza la soluzione al crescere di \mathbf{w} . Questa nuova versione prende il nome di *regressione ridge*.

La regolarizzazione viene controllata dal fattore moltiplicativo λ che ne determina la sensibilità alle variazioni di $\|\mathbf{w}\|_2^2$. Se questo viene impostato a 0 allora si torna alla Formula 7.9, se invece si sceglie di assegnargli ∞ si ottiene l'effetto opposto in cui si ignorano completamente i dati. Tutti i valori compresi fra 0 e ∞ sono potabili.

Il problema della regressione ha una soluzione in forma chiusa:

$$\widehat{\mathbf{w}} = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y} \quad (7.13)$$

Questa soluzione è quasi uguale alla Formula 7.11, dalla quale differisce solo per l'aggiunta di $\lambda \mathbf{I}$ che controlla la regolarizzazione, il termine \mathbf{I} rappresenta la matrice identità. Questa modifica permette di rendere più robusto e stabile il processo di inversione di una

matrice, la regolarizzazione quindi consiste nell'aggiunta di λ alla diagonale di $\mathbf{X}^\top \mathbf{X}$.

7.1.2.2 Regressione lasso

La regressione ridge, caratterizzata dall'aggiunta del fattore $\lambda \|\mathbf{w}\|_2^2$, non è l'unica possibile regolarizzazione.

Una seconda funzione alternativa alla precedente è la *regressione lasso* (least absolute shrinkage and selection operator), dove invece del quadrato della norma 2, il termine che si aggiunge è la norma 1:

$$\|\mathbf{w}\|_1 = \sum_i |w_i| \quad (7.14)$$

Questa soluzione riduce a 0 numerosi pesi, quindi vengono favorite soluzioni sparse⁴.

7.1.2.3 Regolarizzare con la minimizzazione della norma del vettore \mathbf{w}

Una prima spiegazione per motivare la scelta dell'euristica adottata per regolarizzare la regressione la si trova partendo dall'assunzione che la matrice \mathbf{X} sia affetta da un errore, specificato dalla matrice \mathbf{D} , che causa gli outliers, pertanto si ha:

$$(\mathbf{X} + \mathbf{D})\mathbf{w} = \mathbf{X}\mathbf{w} + \mathbf{D}\mathbf{w} \quad (7.15)$$

Quindi maggiore è la norma di \mathbf{w} , maggiore è l'impatto del fattore $\mathbf{D}\mathbf{w}$ che descrive l'errore puro sul risultato finale. Da qui si deduce come la minimizzazione della norma del vettore \mathbf{w} sia una scelta efficace come euristica per la riduzione dell'errore.

In riferimento alla Formula 7.12 si noti in particolare che i due fattori $(\mathbf{y} - \mathbf{X}\mathbf{w})^\top (\mathbf{y} - \mathbf{X}\mathbf{w})$ e $\lambda \|\mathbf{w}\|_2^2$ debbano essere bilanciati: si deve ridurre il secondo senza influire troppo sul primo, ovvero la funzione obiettivo vera e propria.

Una seconda spiegazione è basata sul principio del *Novacula Occami*⁵, per cui vettori dei pesi più piccoli possono essere visti come modelli più semplici e quindi da preferirsi, e questa preferenza viene espressa matematicamente con l'introduzione della norma come nelle funzioni di regolarizzazione ridge e lasso.

Una ulteriore spiegazione la si può ricercare da un'analisi dell'impostazione del vincolo della minimizzazione della norma del vettore dei pesi \mathbf{w} . Tale vincolo può essere visto come un bias per l'algoritmo di apprendimento con lo scopo di ridurre efficacemente la varianza dell'errore.

In altre parole sapendo che l'errore si può descrivere come nella Formula 7.16, quindi come la somma del quadrato del bias con la varianza, più altri elementi che possono essere omessi nel modello, as-

⁴ Un vettore sparso infatti contiene molti 0.

⁵ Altrimenti noto come rasoio di Occam.

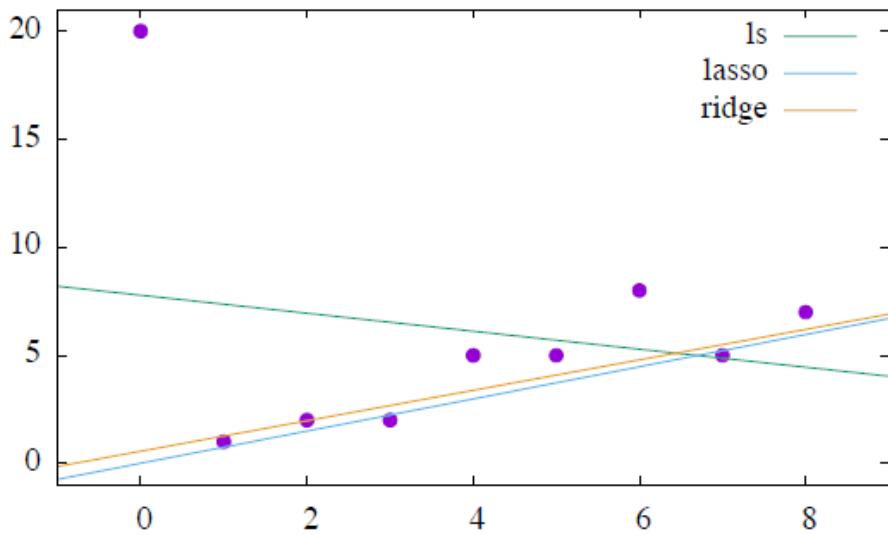


Figura 7.5: Rappresentazione dei tre modelli lineari ottenuti con diverse funzioni di regolarizzazione a confronto sullo stesso dataset di punti.

sumendo che l'errore sia costante, all'aumentare del bias la varianza si riduce.

$$e = b^2 + v + \dots \quad (7.16)$$

Utilizzando una funzione di loss quadratica, come il metodo dei minimi quadrati, si ha un regressore abbastanza instabile: anche variazioni modeste dei dati vengono enfatizzate, incrementando la varianza nell'errore di cui poc'anzi. Per questo nel caso delle funzioni ridge e lasso la minimizzazione della norma del vettore \mathbf{w} si traduce in un aumento del quadrato del bias b^2 che, per costruzione della Formula 7.16, si rivela una scelta efficace per ridurre la varianza v .

7.1.2.4 Confronto tra funzioni di regressione

Ricapitolando si hanno a disposizione tre funzioni per la regressione:

1. la regressione con il metodo dei minimi quadrati;
2. la regressione ridge, ovvero la regressione del punto 1 regolarizzata con il quadrato della norma 2 di \mathbf{w} ;
3. la regressione lasso, ovvero la regressione del punto 1 regolarizzata con la norma 1 di \mathbf{w} .

Si possono confrontare questi diversi metodi per valutare il miglioramento apportato dall'introduzione della regolarizzazione, e come diverse funzioni di regolarizzazione producano a loro volta risultati che si adattano meglio a diverse situazioni.

Facendo riferimento alla Figura 7.5 si possono vedere gli effetti delle tre diverse funzioni di regressione affrontate, e come la regolarizzazione apporti i miglioramenti attesi a dispetto della presenza di outliers.

Per completezza vengono riportati nelle Formule 7.17, 7.18 e 7.19 i valori dei pesi nel vettore \mathbf{w} calcolati per l'esempio di cui sopra:

$$\mathbf{w}_{ls} = \begin{bmatrix} -0.41663 \\ 7.77770 \end{bmatrix} \quad (7.17)$$

$$\mathbf{w}_{ridge} = \begin{bmatrix} 0.70500 \\ 0.55758 \end{bmatrix} \quad (7.18)$$

$$\mathbf{w}_{lasso} = \begin{bmatrix} 7.4465e-01 \\ 3.8439e-06 \end{bmatrix} \quad (7.19)$$

In primo luogo si notava già in precedenza come il modello lineare ottenuto dalla regressione con il metodo dei minimi quadrati non generalizzasse adeguatamente a causa dell'eccessiva influenza di un punto particolarmente distante. Con l'introduzione di due diverse soluzioni che regolarizzano la regressione per contrastare l'overfitting si apprezzano notevoli miglioramenti in tal senso, i modelli prodotti infatti appaiono molto più sensati rispetto alla distribuzione dei punti nel dataset.

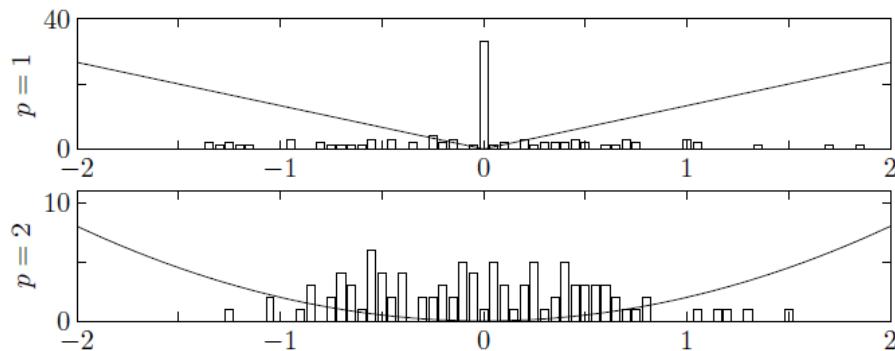


Figura 7.6: Confronto tra le funzioni di regressione regolarizzata lasso (in alto) e ridge (in basso) in termini di minimizzazione della norma e dispersione dei vettori dei pesi ottenuti.

In riferimento alla Figura 7.6 si possono confrontare le due funzioni di regressione regolarizzata affrontate in questo capitolo, con particolare attenzione alle differenze fra l'utilizzo della norma 1 e della norma 2 rispettivamente nella funzione lasso e nella funzione ridge.

Sull'asse delle ascisse si trovano tutti i valori che possono assumere le singole features, ovvero dei valori all'interno del vettore dei pesi \mathbf{w} . Analizzando i due grafici si nota come nel primo in alto c'è una spiccata densità di valori corrispondenti a 0, che come detto in precedenza porta alla definizione di vettore sparso; nel secondo invece

vi è una maggiore distribuzione dei valori all'interno dell'intervallo compreso fra -1 e 1 .

Tale differenza nella distribuzione che si ottiene nei due diversi vettori di pesi \mathbf{w} si spiega con la funzione di regolarizzazione rappresentata nei due grafici. In particolare nel primo in alto la norma 1 corrisponde alla funzione lineare lasso che ha valore nullo nel punto di coordinate $(0, 0)$ e per valori di x che crescono o decrescono cresce linearmente. Questo si traduce in una costante penalizzazione per tutti i valori sull'asse delle ascisse, pertanto l'algoritmo di apprendimento finisce per ritenere sempre più conveniente fissare a 0 buona parte delle features e lasciarne poche anche con valori più alti, che nell'esempio in Figura 7.6 corrispondono a valori maggiori di 1 e minori di -1 .

Nel secondo grafico invece si può distinguere la caratteristica curva di una funzione quadratica, nello specifico del quadrato della norma 2 , che ha vertice di coordinate $(0; 0)$ e cresce per valori di x decrescenti o crescenti. Differentemente dalla funzione lineare lasso discussa poc'anzi, in questo caso la distribuzione nel vettore dei pesi risultante mostra non un picco di zeri, bensì valori più distribuiti nell'intervallo compreso fra -1 e 1 . Tale risultato si spiega ovviamente con il tipo di funzione di regolarizzazione adottata. Se infatti si pone l'attenzione sulla curva rappresentata nel grafico si capisce come la penalizzazione per valori nell'intervallo $[-1, 1]$ sia nettamente inferiore rispetto a valori esterni allo stesso intervallo. L'algoritmo di apprendimento quindi ritiene più conveniente ridurre le features nell'intervallo fra -1 e 1 per minimizzare la penalizzazione della funzione quadratica, e all'interno di questo intervallo non reputa molto conveniente concentrare a 0 i valori compresi fra -1 e 1 come invece faceva la funzione lasso.

7.1.3 Classificazione con il metodo dei minimi quadrati

Fino a questo momento il metodo dei minimi quadrati è stato impiegato per risolvere problemi di regressione, tuttavia risulta particolarmente facile adattare tale approccio per affrontare problemi di classificazione. Per semplicità si parlerà di classificazione binaria:

$$\hat{c}(\mathbf{x}) = \begin{cases} 1 & \text{se } \mathbf{x}^T \hat{\mathbf{w}} - t > 0 \\ 0 & \text{se } \mathbf{x}^T \hat{\mathbf{w}} - t = 0 \\ -1 & \text{se } \mathbf{x}^T \hat{\mathbf{w}} - t < 0 \end{cases} \quad (7.20)$$

Con la Formula 7.20 si codifica in maniera molto semplice il risultato della regressione: la classe positiva viene identificata con il valore 1 , quella negativa con -1 . Inoltre è possibile, non obbligatoriamente, introdurre una terza classe identificata con 0 con il preciso scopo di

rappresentare una sorta di astensione dallo specificare una delle due precedenti classi, ovvero quando *nun saccio che pisce aggi'a piglia'*.

Si noti inoltre che non si utilizza più la formulazione omogenea utilizzata precedentemente in questo capitolo, pertanto si introduce il fattore t che prima corrispondeva alla colonna di 1 introdotta nella matrice \mathbf{X} .

7.2 SUPPORT VECTOR MACHINE

Quando si ha un problema di classificazione, come nell'esempio riportato in Figura 7.7, l'obiettivo è quello di trovare l'iperpiano, a.k.a. *decision boundary*, che divide correttamente le due classi di esempi; due classi linearmente separabili possono essere classificate da infiniti iperpiani. Particolarmenete interessante è l'iperpiano che divide nel modo migliore suddette classi.

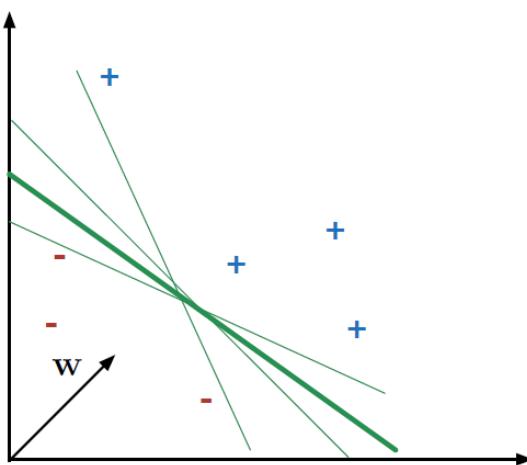


Figura 7.7: Ricerca di un classificatore lineare fra i diversi (e infiniti) possibili classificatori lineari per gli esempi positivi e negativi proposti.

Il concetto di "iperpiano migliore" va quindi indagato e definito con particolare attenzione, graficamente lo si può visualizzare con la Figura 7.8. Intuitivamente lo si può esprimere come il decision boundary che massimizza la distanza dagli esempi più vicini di entrambe le classi, in questo modo si auspica che possa generalizzare nel migliore dei modi.

Lo scopo di una **SVM** è quindi quello di trovare proprio questo iperpiano che separa nel migliore dei modi le due classi. Per cominciare a formalizzare questa idea si deve introdurre il concetto di *margine*, ovvero la distanza fra un esempio e l'iperpiano. Ogni esempio a disposizione viene classificato con la Formula 7.21:

$$y_i(\mathbf{w}\mathbf{x}_i - t) > 0 \quad (7.21)$$

Prendendo in considerazione i due fattori della moltiplicazione y_i , l'output atteso dalla classificazione, e $(\mathbf{w}\mathbf{x}_i - t)$, il risultato della clas-

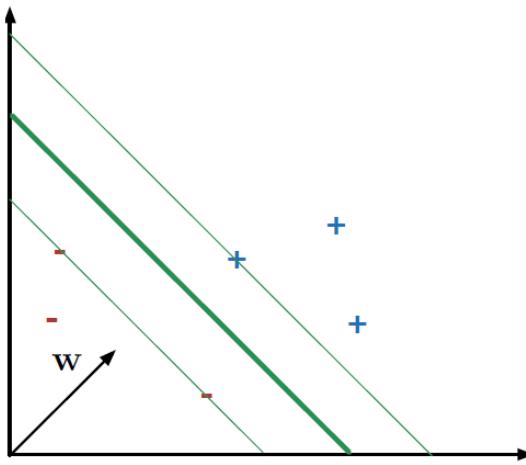


Figura 7.8: Il classificatore lineare ideale, equidistante dagli esempi più vicini di ambedue le classi.

sificazione, per risultare strettamente maggiori di 0 è necessario che siano concordi in segno. Questo equivale a dire che la soluzione della classificazione (il fattore tra parentesi tonde) e l'output atteso y_i , devono indicare la stessa classe, positiva o negativa che sia, per soddisfare la disequazione.

Assumendo che suddetta proprietà sia valida per tutti gli esempi, quindi che la classificazione non sbagli mai, allora esiste un valore ϵ tale che sia possibile riscrivere la disequazione precedente come segue:

$$y_i(\mathbf{w}\mathbf{x}_i - t) \geq \epsilon \quad (7.22)$$

A questo punto è interessante notare che se da entrambi i lati di questa nuova disequazione si moltiplica per una stessa quantità positiva e grande a piacere non si altera la sua veridicità. Si noti inoltre che l'esempio più vicino all'iperpiano soddisfa l'uguaglianza:

$$M y_i(\mathbf{w}\mathbf{x}_i - t) = M\epsilon \quad (7.23)$$

Il termine M , che indica il fattore moltiplicativo positivo introdotto poc'anzi, può essere considerato come una confidenza da poter scegliere con valori grandi a piacimento. Quindi si ottiene una sorta di grado di libertà nella soluzione che permette di spaziare fra infiniti iperpiani, i quali dividono tutti nel migliore dei modi le due classi, e si differenziano solo per la scelta del fattore moltiplicativo M .

Facendo nuovamente riferimento alla Figura 7.8 si può immaginare il decision boundary evidenziato come l'intersezione degli infiniti iperpiani che classificano gli esempi con il piano al quale appartengono gli esempi stessi. L'intersezione rimane invariata, ciò che cambia è il coefficiente angolare fra gli iperpiani delle soluzioni e il piano degli esempi, che varia secondo la scelta del fattore moltiplicativo M .

7.2.1 I Support Vector

A questo punto si decide di fissare per tutti gli esempi una distanza dall'iperpiano definita come segue:

$$y_i(\mathbf{w}\mathbf{x}_i - t) \geq 1 \quad (7.24)$$

Questo vale a dire che non ci possono essere esempi con una distanza dall'iperpiano inferiore ad 1, valore scelto a piacere. Di conseguenza gli esempi più vicini all'iperpiano soddisferanno l'equazione:

$$y_i(\mathbf{w}\mathbf{x}_i - t) = 1 \quad (7.25)$$

Tali esempi vengono definiti *support vectors*, dai quali deriva il nome Support Vector Machine ([SVM](#)).

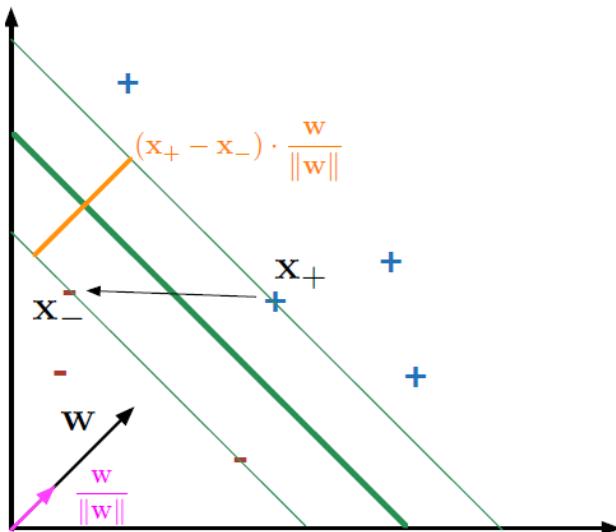


Figura 7.9: Interpretazione matematica del calcolo del margine fra i support vector, gli esempi più vicini alla soluzione.

Si prenda ora in considerazione la Figura 7.9 dove sono evidenziati un esempio positivo, \mathbf{x}_+ , ed un esempio negativo, \mathbf{x}_- , i quali sono i più vicini alla soluzione, il decision boundary. L'intento è quello di massimizzare la distanza evidenziata in arancione e calcolata come:

$$(\mathbf{x}_+ - \mathbf{x}_-) \cdot \frac{\mathbf{w}}{\|\mathbf{w}\|} \quad (7.26)$$

Calcolare la distanza fra i due esempi con la sola differenza $(\mathbf{x}_+ - \mathbf{x}_-)$ sarebbe scorretto perché non si otterrebbe il segmento arancione, bensì la freccia nera che va dall'esempio positivo a quello negativo. Per ovviare a questo problema si fa il prodotto scalare di detta differenza per il vettore unità di \mathbf{w} , indicato nell'immagine con il colore viola.

Si può riassumere con la seguente formula il margine μ :

$$\mu = (\mathbf{x}_+ - \mathbf{x}_-) \cdot \frac{\mathbf{w}}{\|\mathbf{w}\|} \quad (7.27)$$

$$= \frac{\mathbf{x}_+ \cdot \mathbf{w}}{\|\mathbf{w}\|} - \frac{\mathbf{x}_- \cdot \mathbf{w}}{\|\mathbf{w}\|} \quad (7.28)$$

Poiché sono stati scelti come esempi i support vector, ovvero gli esempi più vicini all'iperpiano, richiamando la Formula 7.25 e assumendo che y_i valga +1 o -1 rispettivamente per output attesi positivi o negativi, allora si ha:

$$\mathbf{x}_+ \cdot \mathbf{w} - t = 1 \Rightarrow \mathbf{x}_+ \cdot \mathbf{w} = 1 + t \quad (7.29)$$

$$\mathbf{x}_- \cdot \mathbf{w} - t = -1 \Rightarrow \mathbf{x}_- \cdot \mathbf{w} = t - 1 \quad (7.30)$$

Sostituendo questi risultati ai numeratori nella Formula 7.28 si ottiene come lunghezza del margine μ :

$$\mu = \frac{1+t}{\|\mathbf{w}\|} - \frac{t-1}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|} \quad (7.31)$$

7.2.2 Il problema di ottimizzazione

La massimizzazione del margine ottenuto con la Formula 7.31 è equivalente a minimizzare $\|\mathbf{w}\|$, che a sua volta è equivalente a $\|\mathbf{w}\|^2$.

Il problema di ottimizzazione quindi viene formulato come:

$$\begin{aligned} & \text{minimize}_{\mathbf{w}, t} \quad \frac{1}{2} \|\mathbf{w}\|^2 \\ & \text{subject to} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i - t) \geq 1; \quad 0 \leq i \leq n \end{aligned} \quad (7.32)$$

Il problema di minimizzazione è soggetto al vincolo ripreso dalla Forumla 7.24, il quale impone che ogni esempio sia classificato correttamente e con una distanza dall'iperpiano minima pari a 1. Il coefficiente anteposto al quadrato della norma di \mathbf{w} si rivelerà utile nei passaggi successivi in cui sarà necessario derivare tale espressione.

7.2.2.1 La bellezza del problema di ottimizzazione

Questa breve digressione ha l'importante scopo di rimarcare un aspetto non di poco conto e che sovente non viene considerato in ambito scientifico.

Si vuole infatti far presente come un problema che si è presentato apparentemente ostico e difficile, sia stato poi risolto con un'espressione tanto semplice quanto bella come la Formula 7.32.

In queste particolari occasioni si rende necessario fermarsi a riflettere e contemplare quella che si può definire *arte*.

7.2.3 Formulazione del problema duale

Con lo scopo di approfondire la soluzione trovata con la Formula 7.32 e soprattutto per cercare di ottenere un ulteriore vantaggio con l'aiuto dei kernel, è interessante affrontare l'utilizzo della teoria della dualità, la quale per problemi di ottimizzazione permette di trovare limiti tutt'altro che triviali.

La formulazione duale del problema è una funzione di massimizzazione, possibilmente più semplice e facile da risolvere, di un limite inferiore per il problema iniziale di minimizzazione; nel caso di un problema convesso come quello della Formula 7.32 la soluzione del problema duale corrisponde esattamente a quella del primale, ottenendo quindi la *dualità forte*. Le condizioni per la dualità forte sono note come condizioni di Karush-Kuhn-Tucker.

Si prenda in considerazione il seguente esempio di problema generico di ottimizzazione:

$$\begin{aligned} & \text{minimize}_x \quad f_0(x) \\ & \text{subject to} \quad f_i(x) \leq 0, \quad i = 1, \dots, m \\ & \quad g_i(x) = 0, \quad i = 1, \dots, p \end{aligned} \tag{7.33}$$

Da questa forma sufficientemente generale si possono ottenere numerose varianti applicabili a qualsiasi caso. Ad esempio: si può cambiare il segno della funzione obiettivo per passare da un problema di minimizzazione ad uno di massimizzazione, cambiare il segno di un vincolo inverte il verso della disequazione, a 0 si può sostituire un qualsiasi altro valore t .

La formulazione duale di Lagrange si definisce come:

$$\begin{aligned} g(\alpha, \nu) &= \inf_x \Lambda(x, \alpha, \nu) \\ &= \inf_x \left(f_0(x) + \sum_{i=1}^m \alpha_i f_i(x) + \sum_{i=1}^p \nu_i g_i(x) \right) \end{aligned} \tag{7.34}$$

La funzione ottenuta con l'ultimo passaggio è chiaramente un limite inferiore alla formulazione del problema primale di minimizzazione. La funzione lagrangiana Λ è costituita dalla somma di tre fattori: la funzione obiettivo originale, la sommatoria ottenuta dal primo vincolo moltiplicato per il coefficiente α_i , una analoga sommatoria per il secondo vincolo moltiplicato per il coefficiente ν_i ; per ogni punto ammissibile con valori negativi di $f_i(x)$, che quindi rispettano il vincolo imposto nel problema primale, e α_i positivo si ha un risultato negativo; un discorso simile vale per l'ultimo termine della somma, il quale però risulta nullo. Il risultato di queste somme porta quindi ad una quantità sicuramente minore di $f_0(x)$.

Applicando l'approccio generale della dualità alle **SVM** la funzione Λ ha come parametri un coefficiente moltiplicativo $\alpha_i \geq 0$ per ogni

vincolo del problema primale, oltre ai due parametri originari \mathbf{w} e t , per ottenere la funzione lagrangiana seguente:

$$\begin{aligned}\Lambda(\mathbf{w}, t, \alpha_i) &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i (y_i (\mathbf{w} \cdot \mathbf{x}_i - t) - 1) \\ &= \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i y_i (\mathbf{w} \cdot \mathbf{x}_i) + \sum_{i=1}^n \alpha_i y_i t + \sum_{i=1}^n \alpha_i \\ &= \frac{1}{2} \mathbf{w} \cdot \mathbf{w} - \mathbf{w} \cdot \left(\sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \right) + t \left(\sum_{i=1}^n \alpha_i y_i \right) + \sum_{i=1}^n \alpha_i\end{aligned}\quad (7.35)$$

Si noti l'assenza di coefficienti moltiplicativi ν rispetto alla formulazione generale, questo perché nel problema primale delle SVM non ci sono vincoli della forma $= 0$.

In questa prima formulazione (Formula 7.35) del problema duale si possono riconoscere tutti gli elementi già introdotti con la versione generale dei problemi duali:

- $\frac{1}{2} \|\mathbf{w}\|^2$ è la funzione obiettivo nel problema primale di minimizzazione;
- $\sum_{i=1}^n \alpha_i (y_i (\mathbf{w} \cdot \mathbf{x}_i - t) - 1)$ è la somma (per tutti gli n esempi nel dataset) del vincolo della forma ≤ 0 presente nella versione primale del problema di minimizzazione, moltiplicato per il coefficiente positivo α_i .

Con i due passaggi che portano alla Formula 7.36 viene distribuito il coefficiente α_i all'interno della sommatoria e spezzata la sommatoria stessa; tenendo presente che $\alpha_i y_i$ è un numero, di conseguenza può essere moltiplicato con \mathbf{x}_i per isolare \mathbf{w} e portare quest'ultimo termine all'esterno della sommatoria. Un analogo raccoglimento viene fatto per il termine t nella seconda sommatoria. L'ultima operazione rilevante alla quale fare attenzione è la trasformazione della norma $\|\mathbf{w}\|^2$ nella funzione obiettivo nel prodotto $\mathbf{w} \cdot \mathbf{w}$. Lo scopo di quest'ultima trasformazione è utile per evidenziare il quadrato di \mathbf{w} nell'ottica delle derivate che si calcoleranno nei successivi passaggi.

Per calcolare l'estremo inferiore rispetto a \mathbf{w} e t (la funzione inf nella Formula 7.34) di questa funzione lagrangiana Λ si deve cominciare facendo la derivata parziale proprio rispetto a questi due termini.

$$\frac{\partial \Lambda}{\partial t} = \sum_i \alpha_i y_i \quad (7.37)$$

$$\frac{\partial \Lambda}{\partial \mathbf{w}} = \mathbf{w} - \sum_i \alpha_i y_i \mathbf{x}_i \quad (7.38)$$

La derivata rispetto a t porta a considerare l'unico termine nella Formula 7.36 dove t compare, ovvero la seconda sommatoria, ottenendo pertanto quanto riportato nella Formula 7.37⁶.

Analogo discorso per la derivata rispetto al termine \mathbf{w} , il cui risultato viene riportato nella Formula 7.38. In questo caso il primo termine della sottrazione ottenuta dalla derivazione è il frutto della derivata della funzione obiettivo⁷; il secondo termine invece si ottiene dalla derivata della prima sommatoria, quella dalla quale era stato raccolto \mathbf{w} , e ciò che rimane è proprio il coefficiente dell'oggetto della derivazione.

Imponendo che le due derivate appena calcolate siano uguali a 0 si ottiene:

$$\sum_i \alpha_i y_i = 0 \quad (7.39)$$

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \quad (7.40)$$

A questo punto si deve far presente un risultato sorprendente e meraviglioso. Con la Formula 7.40 si scopre che il vettore dei pesi \mathbf{w} che si desidera trovare si può ottenere calcolando la somma di tutti gli esempi iniziali \mathbf{x}_i moltiplicati per un coefficiente α_i e per y_i . La soluzione risulta essere una combinazione lineare degli esempi originari. Fino ad ora questo non era noto ed è una scoperta molto interessante che giustifica la formulazione duale del problema iniziale.

Sostituendo le Formule 7.39 e 7.40 nella funzione lagrangiana Λ riportata nella Formula 7.36 si elimina la seconda delle tre sommatorie perché il prodotto risulterebbe nullo, mentre il vettore \mathbf{w} viene sostituito in tutte le sue tre occorrenze con quello ottenuto dalla derivazione. Per semplicità vengono presentati degli ulteriori passaggi non presenti nelle slide utilizzate a lezione, al fine di esplicitare tutti i calcoli fatti per arrivare ai risultati che verranno presentati successivamente. Per rendere più facile seguire suddetti passaggi si adotterà una valida semplificazione: invece di sostituire \mathbf{w} con quanto ottenuto dalla

⁶ Si ricorda che essendo il fattore $\alpha_i y_i$ un numero, il risultato della sommatoria sarà a sua volta un numero, ed in quanto unico coefficiente di t diventa l'unico risultato della derivata rispetto a t .

⁷ Si ricorda che nella Formula 7.32 era stato introdotto il coefficiente $\frac{1}{2}$ nell'ottica di una successiva semplificazione della derivata. Ecco, questo è il momento. Grazie a quel coefficiente si ottiene \mathbf{w} invece di $2\mathbf{w}$.

derivazione, si farà esattamente la sostituzione opposta, pertanto la sommatoria $\sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$ viene sostituita con \mathbf{w} per ottenere:

$$g(\alpha) = \frac{1}{2} \mathbf{w} \cdot \mathbf{w} - \mathbf{w} \cdot \mathbf{w} + \sum_{i=1}^n \alpha_i \quad (7.41)$$

$$= -\frac{1}{2} \mathbf{w} \cdot \mathbf{w} + \sum_{i=1}^n \alpha_i \quad (7.42)$$

$$= -\frac{1}{2} \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \sum_{j=1}^n \alpha_j y_j \mathbf{x}_j + \sum_{i=1}^n \alpha_i \quad (7.43)$$

$$= -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j + \sum_{i=1}^n \alpha_i \quad (7.44)$$

Per riassumere i passaggi:

- nella Formula 7.41 è stata sostituita la prima sommatoria con il termine \mathbf{w} come accennato poc'anzi, ed è stata eliminata la seconda sommatoria di valore nullo;
- nella Formula 7.42 è stata fatta la somma tra i due coefficienti del fattore $\mathbf{w} \cdot \mathbf{w}$, ovvero $\frac{1}{2} - 1$;
- nella Formula 7.43 è stata effettuata la sostituzione delle due occorrenze del vettore dei pesi \mathbf{w} con la Formula 7.40 ottenuta dalla derivazione;
- nella Formula 7.44 è stato svolto il prodotto fra le due sommatorie per ottenere il risultato finale.

Questa formulazione finale ottenuta è in generale un limite inferiore della soluzione parametrizzata su α , la cui massimizzazione porta alla seguente formulazione duale del problema primale delle SVM:

$$\begin{aligned} & \text{maximize}_{\alpha} \quad -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j + \sum_{i=1}^n \alpha_i \\ & \text{subject to} \quad \alpha_i \geq 0, \quad i = 1, \dots, n \\ & \quad \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned} \quad (7.45)$$

7.2.3.1 Conclusioni finali sulla formulazione duale

Si cerca ora di riassumere che cosa si può concludere dopo tutta la fatica dedicata per formalizzare adeguatamente il problema duale. In particolare si pone l'accento su diversi vantaggi ottenuti.

In primo luogo si è scoperto che il vettore dei pesi \mathbf{w} che risolve il problema di classificazione è una combinazione lineare dei support vector.

In aggiunta a questo ora si sa che i moltiplicatori lagrangiani positivi (strettamente maggiori di zero, non nulli) sono associati con i support vector, implicando di conseguenza che tutti gli esempi che non sono support vector sono associati a moltiplicatori di Lagrange nulli, quindi gli esempi di cui si tiene conto sono solo i support vector, i quali di solito non sono molti.

A questo proposito tale proprietà che mette in relazione i moltiplicatori lagrangiani positivi con i support vector è una conseguenza di una delle condizioni di Karush-Kuhn-Tucker:

$$\alpha_i(y_i(\mathbf{w} \cdot \mathbf{x}_i - t) - 1) = 0 \quad (7.46)$$

Il prodotto tra il coefficiente lagrangiano α_i e il vincolo espresso nella formulazione del problema primale di ottimizzazione nella Formula 7.32 deve essere nullo. Il vincolo appena presentato, in quanto tale, dovrà essere necessariamente rispettato ed essendo noi interessati ad individuare i support vector siamo certi che per questi punti il secondo fattore assumerà valore 0 (dal vincolo 7.25) e quindi distinguiamo per α_i i seguenti due casi:

$$\left\{ \begin{array}{ll} \alpha_i = 0, & \text{ma allora il prodotto 7.46 sarà } = 0 \\ & \text{a prescindere dal fatto che il secondo} \\ & \text{fattore sia } 0, \text{ cioè che si tratti effettivamente} \\ & \text{di un support vector} \\ \alpha_i > 0, & \text{allora perchè il vincolo 7.46 sia } = 0 \text{ il secondo} \\ & \text{fattore è necessariamente } = 0, \\ & \text{cioè } \mathbf{x} \text{ è un support vector} \end{array} \right.$$

Quindi, come sarà ormai chiaro, gli esempi che avranno un coefficiente α_i strettamente positivo saranno un sottoinsieme (eventualmente esattamente lo stesso insieme) dei support vector; tutti gli altri esempi sono ininfluenti e il calcolo del vettore dei pesi \mathbf{w} , ovvero l'apprendimento, come anche la classificazione con l'iperpiano ottenuto, possono essere svolti in termini di prodotto scalare dei soli support vector.

Quest'ultima ancora più importante conclusione porta ad introdurre l'argomento successivo. La formulazione duale del problema delle SVM permette infatti di svolgere anche task di classificazione non lineare grazie all'introduzione del metodo Kernel, nel quale il prodotto scalare viene sostituito da un prodotto lineare in spazio multi dimensionale per ottenere funzioni più complesse di quelle lineari utilizzate fino ad ora.

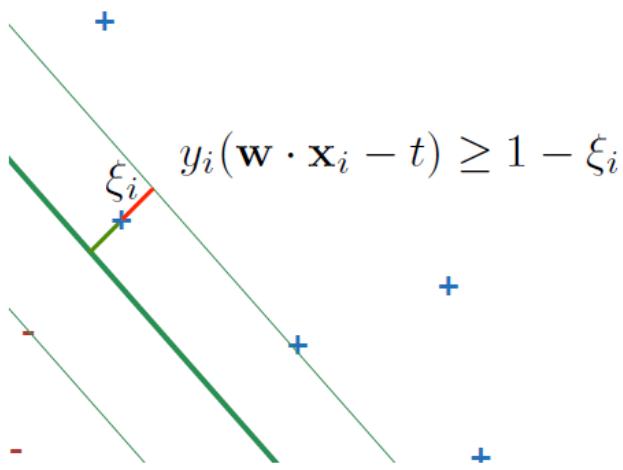


Figura 7.10: Esempio di SVM con un margine di tolleranza per gli errori con un esempio che giace fra la soluzione ed un support vector.

7.2.4 Margini di errore

Il metodo delle **SVM** presentato funziona egregiamente quando gli esempi sono linearmente separabili, se si dovesse introdurre uno o più esempi che non rendono più vera tale affermazione allora tutto quanto non funziona più.

La formulazione fino ad ora elaborata era soggetta al forte vincolo del margine fra esempio e piano pari a 1. L'intenzione ora è di rilassare questa limitazione per ovviare alla problematica appena presentata e tollerare quindi alcuni errori nella soluzione.

Tramite l'introduzione del termine ξ_i si introduce quindi una tolleranza verso due tipologie di errori.

$$y_i(\mathbf{w} \cdot \mathbf{x}_i - t) \geq 1 - \xi_i \quad (7.47)$$

In questo modo si permette la presenza di esempi più vicini alla soluzione, ovvero che giacciono fra quest'ultima e i support vector, come è raffigurato nella Figura 7.10. Inoltre per valori sufficientemente grandi di ξ_i si può arrivare a tollerare esempi che si trovano oltre il confine segnato dalla soluzione, quindi errori di classificazione, come per esempio un positivo nel bel mezzo dei negativi. Poverino.

La variabile ξ_i , altrimenti nota come *slack variable*, viene associata ad ogni esempio, permettendo così ad alcuni esempi di potersi trovare più vicini all'iperpiano rispetto ai support vector, o addirittura oltre quest'ultimo, ovvero classificati male.

La formulazione del problema di ottimizzazione della Formula 7.32 diventa:

$$\begin{aligned} & \text{minimize}_{\mathbf{w}, t, \xi} \quad \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i \\ & \text{subject to} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i - t) \geq 1 - \xi_i; \quad 1 \leq i \leq n \\ & \quad \xi_i \geq 0; \quad 1 \leq i \leq n \end{aligned} \quad (7.48)$$

Nella funzione obiettivo si nota l'aggiunta di una penalizzazione con il termine $C \sum_{i=1}^n \xi_i$. Questo vuol dire che non si cerca solo di minimizzare il quadrato della norma del vettore dei pesi \mathbf{w} , ma anche la somma di tutte le slack variable ξ_i , che vuol dire minimizzare il numero di esempi che si trovano oltre ai support vector o addirittura oltre al decision boundary. La costante C , noto anche come parametro della complessità, ha lo scopo di regolare l'importanza che viene attribuita al margine di errore introdotto, ergo la complessità della [SVM](#).

Trovare la soluzione ottimale implica quindi un buon bilanciamento fra la massimizzazione del margine fra esempi e iperpiano e la tolleranza di errori di classificazione, che implica sia la quantità degli errori che la loro grandezza.

A tale scopo la scelta di un adeguato valore per la costante C si rivela molto importante: valori particolarmente alti per questo parametro infatti aumentano la penalizzazione per ogni di errore concesso, al contrario valori molto bassi concedono più flessibilità nell'accettare esempi classificati in maniera scorretta. Ovviamente C può anche essere pari a 0, in questo caso il secondo termine della funzione da minimizzare si annulla e il risultato è che il processo di apprendimento sarà libero di classificare scorrettamente degli esempi senza alcun vincolo. Questo è possibile perché nel vincolo è rimasto invece il valore ξ_i sottratto ad 1. Il risultato in questo particolare caso è quindi un iperpiano che cerca di minimizzare la norma senza più alcun interesse nel classificare correttamente gli esempi, il vettore dei pesi \mathbf{w} risulterà allora essere un vettore nullo, le cui componenti sono tutte pari a 0.

Pari attenzione va posta anche al caso in cui si sceglie un valore per la costante C molto grande, troppo grande, grandemente grande. In questo caso vuol dire che gli errori praticamente non sono tollerati, infatti la penalizzazione che ne conseguirebbe sarebbe altissima. In questo frangente il processo di addestramento finirebbe per scegliere i coefficienti ξ_i nulli, con la conseguenza che nella funzione da minimizzare scomparirebbe il secondo termine (prodotto nullo), mentre a destra della disequazione che costituisce il vincolo rimarrebbe solo 1. Ma questa versione del problema è esattamente quella originaria, in cui non erano contemplati gli errori, la Formula [7.32](#).

Più in generale se si permette un minore margine di errore basteranno meno support vector per risolvere la classificazione.

Analogamente a quanto fatto per la prima formulazione del problema delle [SVM](#), anche in questo caso si affronterà la riformulazione duale, pertanto si costruisce subito la funzione lagrangiana Λ . Per completezza si ricorda che gli elementi che la compongono sono, in

ordine: i primi due addendi vengono ripresi dalla funzione obiettivo, gli altri due sono i due vincoli del problema primale.

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

I calcoli svolti in questo passaggio sono semplici: la seconda delle tre sommatorie viene scomposta dopo che al suo interno è stato svolto il prodotto fra i termini α_i e ξ_i .

Si noti a questo punto che il primo e il terzo addendo sono in realtà la funzione lagrangiana precedente, ovvero quella riportata nella Formula 7.35, pertanto si può riscrivere:

$$\begin{aligned}\Lambda(\mathbf{w}, t, \xi_i, \alpha_i, \beta_i) &= \Lambda(\mathbf{w}, t, \alpha_i) + \sum_{i=1}^n C \xi_i - \sum_{i=1}^n \alpha_i \xi_i - \sum_{i=1}^n \beta_i \xi_i \\ &= \Lambda(\mathbf{w}, t, \alpha_i) + \sum_{i=1}^n \xi_i (C - \alpha_i - \beta_i)\end{aligned}\quad (7.49)$$

Oltre ad aver riportato la funzione lagrangiana del problema originario delle SVM è stato raccolto il fattore comune alle tre sommatorie, ξ_i , per ottenere la Formula 7.49.

A questo punto per trovare l'estremo inferiore della funzione appena ottenuta si dovrebbe fare la derivata rispetto alle variabili \mathbf{w} , t e ξ_i . Poiché le prime due variabili compaiono solo nel primo addendo, ovvero la precedente lagrangiana di cui avevamo già calcolato l'estremo inferiore, non c'è bisogno di ricalcolarne la derivata. Nella sommatoria invece compare ξ_i , la derivata di questa parte viene imposta uguale a 0 e risulta:

$$C - \alpha_i - \beta_i = 0 \quad (7.50)$$

A questo punto se vale quanto appena ottenuto, ovvero che il fattore tra parentesi nella sommatoria della Formula 7.49 sia sempre nullo, l'intera sommatoria ha valore pari a 0. Di conseguenza non rimane altro che:

$$\Lambda(\mathbf{w}, t, \xi_i, \alpha_i, \beta_i) = \Lambda(\mathbf{w}, t, \alpha_i) \quad (7.51)$$

Quindi la nuova funzione lagrangiana è esattamente la stessa funzione lagrangiana che era stata calcolata per il precedente problema delle SVM.

La riformulazione del problema duale pertanto può ereditare la funzione obiettivo dalla Formula 7.45, ed aggiornare solamente i vincoli cui è soggetta con l'introduzione della costante C .

$$\begin{aligned} \text{maximize}_{\alpha} \quad & -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j + \sum_{i=1}^n \alpha_i \\ \text{subject to} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \dots, n \\ & \sum_{i=1}^n \alpha_i y_i = 0 \end{aligned} \quad (7.52)$$

L'introduzione della costante C all'interno del vincolo si spiega partendo dalla Formula 7.50 e svolgendo i seguenti passaggi:

$$\begin{aligned} C - \alpha_i - \beta_i &= 0 \\ -\alpha_i &= -C + \beta_i \\ \alpha_i &= C - \beta_i \end{aligned} \quad (7.53)$$

Tenendo presente che β_i è una variabile duale e quindi sempre maggiore o uguale a 0, allora la quantità a destra dell'ultima equazione rivela che α_i è sempre sicuramente più piccola di C , o al più uguale a C .

Il vincolo $\beta_i \geq 0$ è stato omesso poiché β_i non compare nella funzione obiettivo da massimizzare.

Un'altra proprietà interessante sulla quale soffermarsi deriva dalle condizioni di Karush-Kuhn-Tucker. In particolare una di queste afferma $\beta_i \xi_i = 0$, implicando di conseguenza che se un esempio viene classificato con un certo margine di errore $\xi_i \geq 0$ per rispettare la condizione citata poc'anzi deve necessariamente valere $\beta_i = 0$. Questo ha a sua volta un'implicazione collegata alla Formula 7.53 per cui si ha $\alpha_i = C$.

A questo punto è possibile riassumere nel seguente modo come vengono classificati gli esempi:

- $\alpha_i = 0$ per tutti gli esempi classificati correttamente e che non sono support vector;
- $0 < \alpha_i < C$ per tutti gli esempi che sono support vector;
- $\alpha_i = C$ per tutti gli esempi classificati in maniera errata.

Con la Figura 7.11 è possibile vedere come le slack variable sono legate alla posizione degli esempi rispetto all'iperpiano che svolge la classificazione.

7.2.4.1 La bellezza del problema duale finale

Vale la pena fermarsi nuovamente qualche istante per ammirare la bellezza del risultato appena ottenuto.

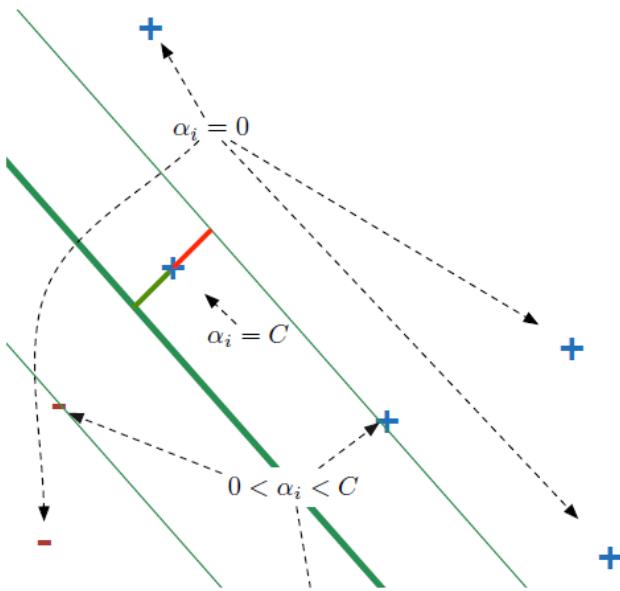


Figura 7.11: Interpretazione grafica del ruolo che giocano le slack variable nella classificazione di esempi in base alla loro posizione rispetto al decision boundary.

L'introduzione del margine di errore e la conseguente riformulazione del problema delle **SVM** ha portato a numerosi e quasi spaventosi passaggi matematici che sembravano complicare notevolmente la soluzione ottenuta in precedenza. Invece si scopre che il risultato che si arriva ad ottenere è praticamente uguale al problema duale precedente, salvo per l'introduzione della costante C nei vincoli, semplice e bellissimo.

7.3 KERNEL

Punti a questo giunto si sa come risolvere problemi linearmente separabili con una **SVM**, come affrontare problemi non-linearmente separabili con l'introduzione dei margini di errore, ma rimane un ultimo nodo: la multidimensionalità. A questo proposito si introduce proprio il metodo Kernel, in inglese *kernel trick*, il quale permette di sfruttare algoritmi per problemi lineari per risolvere problemi non-lineari, che è proprio ciò che si farà con le **SVM**.

L'idea alla base del metodo Kernel è la riformulazione dello spazio delle ipotesi, ovvero trasformare un dataset di esempi non separabili in un equivalente dataset con gli stessi esempi ma in uno spazio multidimensionale che permetta possibilmente di trovare con una funzione lineare (un classificatore **SVM**, ad esempio) una soluzione lineare, la quale corrisponderebbe ad una soluzione non lineare per lo spazio delle ipotesi originario. Un esempio di tale mapping è quello riportato in Figura 7.12, questo è particolarmente semplice perché si passa da 2 dimensioni a 3 dimensioni ma al contempo chiarificatore.

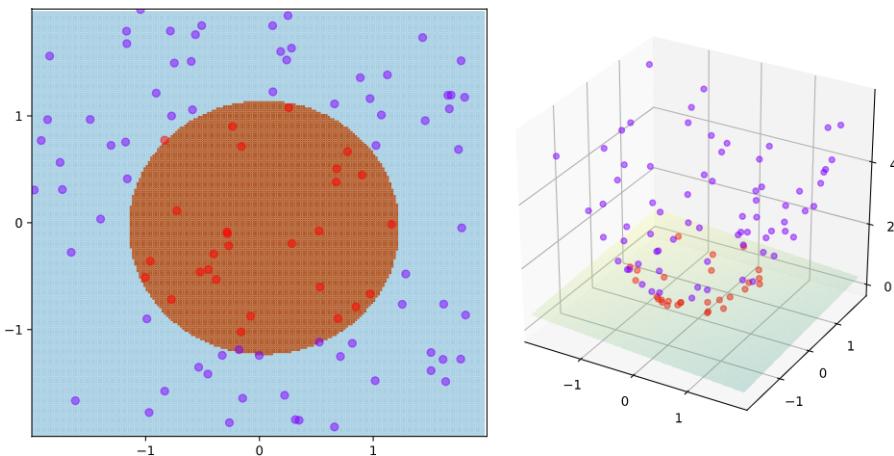


Figura 7.12: Esempio di come si applica l’idea del metodo Kernel per rimappare esempi di uno spazio bidimensionale (a sinistra) in uno spazio tridimensionale (a destra) che permetta di trovare una soluzione lineare con una SVM.

Si può inoltre fare un esempio che segue in approccio più matematico e formale nella presentazione di questo metodo. Si prenda in considerazione un vettore di due elementi \mathbf{x} , un mapping ϕ ed una funzione di classificazione lineare $\hat{c}(\mathbf{x})$ della SVM:

$$\begin{aligned}\mathbf{x} &= (x_1, x_2) \\ \phi(\mathbf{x}) &= (x_1^2, \sqrt{2}x_1x_2, x_2^2, c) \\ \hat{c}(\mathbf{x}) &= \text{sign}(\mathbf{w} \cdot \mathbf{x} - t)\end{aligned}\tag{7.54}$$

7.3.1 La funzione Kernel

La funzione Kernel è il prodotto interno tra $\phi(\mathbf{x})$ e $\phi(\mathbf{y})$, quindi il prodotto interno di \mathbf{x} e \mathbf{y} mappati in uno spazio di Hilbert multidimensionale.

$$K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle\tag{7.55}$$

Si tenga presente che questo metodo è generale, quindi può essere applicato a qualsiasi prodotto scalare, a prescindere da specifici valori. Di seguito viene ripreso l’esempio introdotto in precedenza, si utilizzerà il Kernel per sostituire il prodotto fra gli esempi \mathbf{x} e i pesi \mathbf{w} , tuttavia si potrebbe applicare anche ad un ipotetico prodotto fra esempi $\mathbf{x}_i \cdot \mathbf{x}_j$.

Se l’idea di Kernel $K(\mathbf{w}, \mathbf{x})$ la si applica al prodotto $\mathbf{w} \cdot \mathbf{x}$ della Formula 7.54 si ottiene:

$$\begin{aligned}\hat{c}(\mathbf{x}) &= \text{sign}(K(\mathbf{w}, \mathbf{x}) - t) = \text{sign}(\phi(\mathbf{w}) \cdot \phi(\mathbf{x}) - t) \\ &= \text{sign}((w_1^2, \sqrt{2}w_1w_2, w_2^2, c) \cdot (x_1^2, \sqrt{2}x_1x_2, x_2^2, c) - t) \\ &= \text{sign}(w_1^2x_1^2 + 2w_1w_2x_1x_2 + w_2^2x_2^2 + c^2 - t)\end{aligned}\tag{7.56}$$

Come si nota dall'ultima espressione appena ottenuta si ha ora a che fare con una funzione non più lineare, bensì quadratica se considerata nello spazio delle ipotesi originario, che invece nel nuovo spazio di Hilbert ottenuto con il mapping corrisponde ad un iperpiano lineare.

7.3.1.1 Prodotto interno e spazio di Hilbert

Il prodotto interno è una generalizzazione del prodotto scalare in spazi vettoriali arbitrari, una funzione che associa ad ogni coppia di vettori uno scalare, ovvero è un mapping ϕ come l'esempio fatto in precedenza con la Formula 7.54. Le condizioni che vanno soddisfatte affinché una funzione si possa considerare un prodotto interno sono le seguenti:

SIMMETRIA $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle;$

LINEARITÀ NEL PRIMO ARGOMENTO⁸ $\langle a\mathbf{x} + b\mathbf{y}, \mathbf{z} \rangle = a\langle \mathbf{x}, \mathbf{z} \rangle + b\langle \mathbf{y}, \mathbf{z} \rangle;$

DEFINITO POSITIVO $\langle \mathbf{x}, \mathbf{x} \rangle \geq 0; \langle \mathbf{x}, \mathbf{x} = 0 \rangle \Leftrightarrow \mathbf{x} = 0.$

Si introduce l'idea di Kernel come una funzione $K : \mathbb{V} \times \mathbb{V} \rightarrow \mathbb{R}$, dove \mathbb{V} è uno spazio vettoriale complesso o reale, per la quale esiste il mapping $\phi : \mathbb{V} \rightarrow \mathbb{F}$, dove \mathbb{F} è uno spazio di Hilbert, ovvero uno spazio vettoriale completo su cui è definito un prodotto interno.

Gli spazi di Hilbert si rivelano particolarmente vantaggiosi per le esigenze del problema che si sta affrontando, è infatti lo spazio ideale in cui poter operare perché dispone di tutte le proprietà utili legate al prodotto scalare: calcolo delle distanze, angoli, e ortogonalità (anche come utile misura di similarità fra vettori⁹). Con la garanzia della completezza¹⁰.

7.3.1.2 Kernel come funzioni di similarità

Introducendo il concetto di spazio di Hilbert è stato fatto un accenno a come il prodotto interno ad esso associato permetta di disporre di molte proprietà utili al calcolo di distanze tra vettori, angoli e ortogonalità. Si può interpretare lo spazio di Hilbert in cui vengono mappati gli esempi come uno spazio in cui è possibile valutare in modo efficace la similarità tra i vettori associati agli esempi stessi, grazie alle operazioni che lo spazio di Hilbert mette a disposizione.

Utilizzare i Kernel quindi diventa un modo per mappare gli esempi in un nuovo spazio vettoriale in cui è possibile calcolare la distanza

⁸ Linearità nel secondo argomento applicando la proprietà della simmetria.

⁹ Si ricorda a tal proposito che la similarità del coseno fornisce per vettori ortogonali (o quasi) valori nulli (o quasi), mentre per vettori con un angolo molto ridotto si ottengono risultati alti.

¹⁰ Per un approfondimento della nozione di completezza topologica in matematica si invita a consultare Wikipedia.

tra i vettori associati ed ottenere quindi una misura della loro similità. A tal proposito se due vettori sono simili, ovvero "puntano" nella stessa direzione, allora il loro prodotto scalare sarà molto alto. Al contrario per vettori dissimili si otterrà un risultato del prodotto scalare molto basso, nullo quando i vettori sono ortogonali.

7.3.2 Kernel importanti

Fra tutte le possibili funzioni Kernel che si possono costruire, alcune vanno tenute particolarmente in conto. In particolare la prima funzione Kernel che si vuole ricordare è quella del Kernel polinomiale in entrambe le sue versioni:

$$\begin{aligned} K(\mathbf{x}, \mathbf{y}) &= (\mathbf{x} \cdot \mathbf{y})^d \\ K(\mathbf{x}, \mathbf{y}) &= (\mathbf{x} \cdot \mathbf{y} + 1)^d \end{aligned}$$

Il grado d è determinante per la scelta del giusto Kernel polinomiale, si ha:

- con $d = 1$ si ottiene un volgarissimo e trivialissimo Kernel lineare;
- con $d = 2$ si ottiene un Kernel quadratico che trova molte applicazioni nei problemi legati al linguaggio naturale;
- con un generico valore per d si ottiene un polinomio di grado d .

Si noti in particolare la differenza fra la formulazione appena fornita per la funzione Kernel e quella introdotta precedentemente con la Formula 7.55. Se prima si parlava sempre di fare il mapping e poi calcolare il prodotto interno, ora per una maggiore efficienza prima si calcola il prodotto interno e poi il mapping.

Una seconda funzione rilevante è il Kernel gaussiano:

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

Tale Kernel viene utilizzato principalmente per il calcolo di Radial Basis Function (RBF) e lo spazio di Hilbert in cui vengono mappate le feature è di infinite dimensioni. Il parametro σ ha lo scopo di regolare l'ampiezza della funzione radiale, viene determinato tramite cross validation su un validation set indipendente.

Oltre ai Kernel presentati se ne possono costruire molti altri in base alle diverse esigenze dettate dai problemi che si desidera risolvere con questi modelli lineari. Tali nuovi Kernel possono anche essere applicati a tipi di dati quali stringhe o alberi, come ad esempio succede per alcuni problemi in ambito biologico. Un esempio nel campo dell'analisi delle sequenze di DNA (quindi dove si tratta con stringhe) è il seguente:

$$\phi(x) = (5, 6, 0, \dots, 3)$$

La funzione di mapping in questo caso associa a sottostringhe di una certa lunghezza, 3 ad esempio (AAA, AAC, AAG, AAT, ACA, ACC, ..., TTT¹¹), il numero di volte che queste occorrono all'interno di una stringa x . Una funzione Kernel di questo tipo quindi fa un mapping in un vettore di occorrenze per tutte le possibili sottostringhe utilizzando algoritmi basati sulla programmazione dinamica; con i vettori che si ottengono in questo modo è possibile calcolare molto più agevolmente la loro similarità, la quale corrisponde direttamente alla similarità delle stringhe¹² dalle quali questi sono stati ottenuti. Questo è quindi un altro esempio di come un mapping dello spazio degli esempi di un certo problema in un nuovo spazio di diversa e decisamente maggiore dimensionalità permette di trovare molto più agevolmente le soluzioni del suddetto problema.

7.3.2.1 Combinare e costruire nuovi Kernel

In generale le funzioni Kernel possono essere combinate fra di loro per ottenerne di nuove. Alcune delle combinazioni più facili sono il prodotto e la somma di Kernel:

$$\begin{aligned} K(\mathbf{x}, \mathbf{y}) &= K_1(\mathbf{x}, \mathbf{y})K_2(\mathbf{x}, \mathbf{y}) \\ K(\mathbf{x}, \mathbf{y}) &= K_1(\mathbf{x}, \mathbf{y}) + K_2(\mathbf{x}, \mathbf{y}) \end{aligned}$$

Il prodotto fra due Kernel è all'incirca equivalente ad una operazione di AND, la somma invece ad un OR. Se ad esempio si considerano i Kernel K_1 e K_2 come funzioni di similarità, il comportamento del prodotto rispecchia l'operazione logica di AND, mentre la somma rispecchia quello dell'OR logico.

$$K(\mathbf{x}, \mathbf{y}) = aK(\mathbf{x}, \mathbf{y}), a > 0$$

Si può moltiplicare un Kernel per un qualsiasi valore scalare $a > 0$ e ottenere sempre un Kernel.

$$K(\mathbf{x}, \mathbf{y}) = f(\mathbf{x})f(\mathbf{y})$$

Un Kernel può anche essere il risultato del prodotto di due funzioni reali applicate ad \mathbf{x} e \mathbf{y} .

Ovviamente questi sono solo alcuni esempi, i Kernel si possono ottenere in molti modi, sia dalla combinazione di altri Kernel, sia dalla combinazione di altre funzioni. Tutto purché si rispettino determinate condizioni che garantiscono la validità del Kernel ottenuto.

¹¹ Tutte le stringhe di lunghezza 3 che si possono ottenere dalla combinazione delle quattro lettere utilizzate per identificare i nucleotidi: A, C, G, T.

¹² Il concetto di similarità in questo caso è definito proprio dalle frequenze delle sottostringhe nelle stringhe messe a confronto. Se in entrambe queste compaiono con frequenze analoghe le stesse sottostringhe allora si potranno considerare somiglianti, viceversa nel caso opposto.

7.3.3 Condizioni per le funzioni Kernel

Per verificare che un nuovo Kernel sia effettivamente valido bisogna rispettare delle condizioni, le condizioni di Mercer¹³. Verificare che un Kernel sia valido vuol dire assicurarsi che esista uno spazio di Hilbert nel quale il nuovo Kernel valuta il prodotto interno al suo interno, con il vantaggio che non è necessario conoscere esattamente lo spazio di Hilbert nel quale si effettua il mapping.

Una funzione K è un Kernel valido se e solo se per ogni insieme finito di punti $\{x_1, \dots, x_n\}$ la matrice $\mathbf{K}_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$ con $i, j = 1, \dots, n$, dove \mathbf{x}_i e \mathbf{x}_j sono esempi presi dall'insieme di punti appena introdotto, è simmetrica e semidefinita positiva.

A questo fine si vuole dimostrare che se esiste una funzione di mapping ϕ tale che $K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$, allora per un insieme di punti $\{x_1, \dots, x_n\}$ la matrice \mathbf{K} è simmetrica e semidefinita positiva, ovvero $\mathbf{K} \succeq 0$, ed implica $\forall \mathbf{z} : \mathbf{z}^T \mathbf{K} \mathbf{z} \geq 0$.

Si dimostra facilmente che la matrice \mathbf{K} è simmetrica poiché il prodotto interno è esso stesso simmetrico. Ci si concentra quindi sulla dimostrazione della restante parte: la matrice è semidefinita positiva.

$$\begin{aligned}
 \mathbf{z}^T \mathbf{K} \mathbf{z} &= \sum_i \sum_j z_i z_j \mathbf{K}_{i,j} \\
 &= \sum_i \sum_j z_i z_j K(\mathbf{x}_i, \mathbf{x}_j) \\
 &= \sum_i \sum_j z_i z_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle \\
 &= \sum_i \sum_j \langle z_i \phi(\mathbf{x}_i), z_j \phi(\mathbf{x}_j) \rangle \\
 &= \sum_i \langle z_i \phi(\mathbf{x}_i), \sum_j z_j \phi(\mathbf{x}_j) \rangle \\
 &= \langle \sum_i z_i \phi(\mathbf{x}_i), \sum_j z_j \phi(\mathbf{x}_j) \rangle \\
 &= \langle \sum_i z_i \phi(\mathbf{x}_i), \sum_i z_i \phi(\mathbf{x}_i) \rangle \geq 0
 \end{aligned}$$

Nei primi passaggi si fanno alcune semplici operazioni di sostituzione. La matrice $\mathbf{K}_{i,j}$ viene sostituita con la funzione Kernel $K(\mathbf{x}_i, \mathbf{x}_j)$, che a sua volta viene sostituita con il prodotto interno $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$. A questo punto si applicano le proprietà del prodotto interno introdotte in precedenza. Il prodotto interno è lineare in entrambi gli argomenti, quindi si può scegliere di distribuire z_i e z_j al suo interno. Per lo stesso

¹³ James Mercer fu un matematico inglese noto proprio per aver elaborato l'omonimo teorema di Mercer, nel quale dimostrava che un Kernel definito positivo può essere espresso come un prodotto scalare in uno spazio multi dimensionale, come appunto uno spazio di Hilbert.

motivo nei due passaggi successivi anche le due sommatorie vengono portate nel prodotto interno. Infine nell'ultimo passaggio è possibile sostituire l'indice j con i , ottenendo due termini identici. Poiché il prodotto interno è definito positivo il risultato è sicuramente maggiore o uguale a 0. *Quod era demonstrandum.*

8

MODELLO BASATI SULLA DISTANZA

8.1 SIMILARITÀ E DISTANZA

Questi modelli sono molto utilizzati poiché ragionare sulle distanze è molto semplice.

Molte forme di apprendimento di basano sulle similarità tra gli esempi, cercando in fase di test un esempio simile a quello sottoposto nel training set su cui il modello è stato addestrato.

Nei modelli che tendono al raggruppamento delle istanze (come i decision tree) queste similarità rappresentano un perno su cui effettuare il partizionamento dello spazio delle istanze.

Si introdurrà ora un modello che utilizza una forma più avanzata del concetto di similarità.

Si andrà a misurare la similarità tra due punti tramite il concetto di distanza (due punti vicini saranno considerati molto simili), ma esistono modi diversi per valutare la distanza.

8.1.1 Distanza di Minkowski

La prima valutazione della distanza che viene presentata è quella basata sulla *distanza di Minkowski*, definita come segue.

Sia $\mathbb{X} = \mathbb{R}^d$, la distanza di Minkowski di ordine $p > 0$ è definita come

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

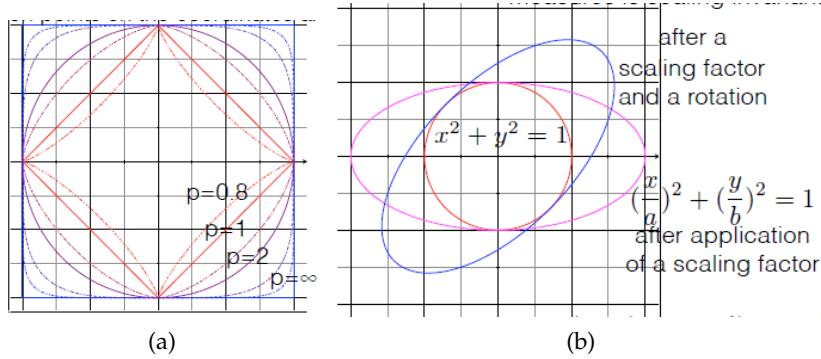
Si nota che questa misura può anche essere vista sotto un altro punto di vista: $\|z\|_p = \left(\sum_{j=1}^d |z_j|^p \right)^{\frac{1}{p}}$ altro non è che la norma p del vettore z .

La norma 2 si nota essere la ormai nota e familiare *distanza Euclidea*, così come la norma 1 denota la *distanza di Manhattan* (distanza da percorrere seguendo solamente le direzioni degli assi x e y).

Al crescere di p , la distanza viene "arricchita" da una precisione maggiore data dalle coordinate, fino ad arrivare a definire la *distanza di chebyshev*:

$$\text{Dis}_{\infty}(x, y) = \max_j |x_j - y_j|$$

Potrà inoltre tornare utile la norma 0, che conta il numero di elementi non nulli in un vettore, e può essere utilizzata quindi per valutare il vettore differenza di x e y per verificare i punti in cui questi sono



uguali. Questa non è in realtà una vera distanza di Minkowski, ma può essere definita come

$$\text{Dis}_0(x, y) = \sum_{j=1}^d (x_j - y_j)^0 = \sum_{j=1}^d I[x_j \neq y_j]$$

stabilendo quindi che $x^0 = 0 \text{ se } x = 0$ (cioè i due elementi sono esattamente gli stessi, quindi l'elemento zero non è preso in considerazione) e $x^0 = 1 \text{ se } x \neq 0$.

Se x e y stringhe binarie, la norma 0 è chiamata anche *distanza di Hamming* (spesso usata per indicare il vettore di bit che occorre sommare al vettore x per ottenere il vettore y).

Per stringhe non binarie di lunghezza non uguale, lo stesso concetto della distanza di Hamming è dato dalla generalizzazione del *distanza di Levenshtein* (o *edit distance*, date due stringhe x e y tale misura è il numero minimo di operazioni di inserimenti, cancellazioni e sostituzioni di un carattere necessari per modificare x in y).

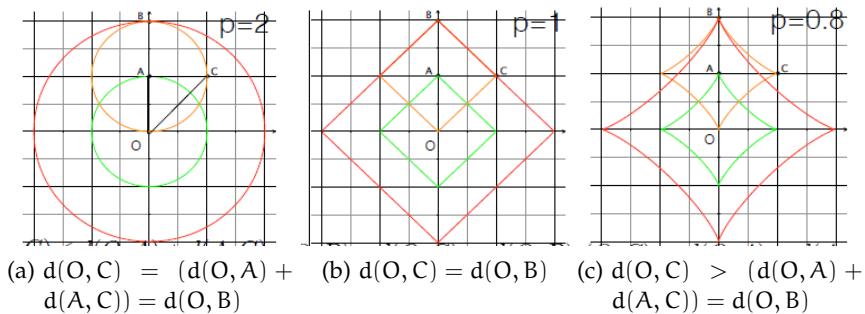
In alcuni casi, le features sono sparse (si pensi alla rappresentazione dei documenti per l'information retrieval, in cui alcune informazioni sono mancanti). In questi casi gli zeri possono rappresentare l'assenza di informazione e non devono quindi essere tenuti in considerazione della valutazione della funzione distanza.

A tale scopo si introduce la *distanza di Jaccard*, di cui possiamo vedere un confronto con il codice di Hamming grazie ad un esempio. Consideriamo le stringhe binarie $p = \{1000000000\}$ e $q = \{0000001001\}$. Indicando con M_{ij} il numero di attributi di p a 0 o dove in q è a 1, calcoliamo $M_{01} = 2$, $M_{10} = 1$, $M_{00} = 7$ e $M_{11} = 0$.

Con il codice di Hamming si ottiene una misura di similarità pari a $(M_{11} + M_{00}) / (M_{11} + M_{00} + M_{01} + M_{10})$ mentre per Jaccard, non considerando M_{00} per quanto detto, si ottiene $(M_{11}) / (M_{11} + M_{01} + M_{10})$.

È quindi chiaro che la scelta del parametro p debba essere accurata poiché incide parecchio nella valutazione della distanza: in particolare, tramite la rappresentazione grafica dei punti equidistanti dall'origine, possiamo vedere come incide questo parametro (figura 8.1a).

Nella figura ?? mostriamo la rappresentazione grafica della distanza



Euclidean che si può notare avere la particolarità di rimanere invariata all'applicazione di un fattore di scaling e di una rotazione.

Vedremo poi come questa osservazione ci sarà utile per introdurre la distanza di Mahalanobis, utile a tener conto di differenti correlazioni tra diverse features (inizialmente x e y non sono correlate, ma a seguito di queste trasformazioni avranno un coefficiente di correlazione in comune).

8.1.1.1 Distanza metrica

Dato uno spazio delle istanze \mathbb{X} , una distanza metrica è una funzione $\mathbb{X} \times \mathbb{X} \rightarrow \mathbb{R}$ tale che per ogni $x, y, z \in \mathbb{X}$:

- la distanza tra un punto e se stesso è 0: $\text{Dis}(x, x) = 0$
- tutte le altre distanze sono maggiori di 0: se $x \neq y$ allora $\text{Dis}(x, y) > 0$
- le distanze sono simmetriche: $\text{Dis}(y, x) = \text{Dis}(x, y)$
- una strada che passa per punti intermedi non può essere più corta del percorso diretto tra i due punti (disegualanza triangolare): $\text{Dis}(x, z) \leq \text{Dis}(x, y) + \text{Dis}(y, z)$

Se il vincolo posto dal secondo punto viene rilassato (ponendo che $\text{Dis}(x, y)$ può assumere valore zero anche se $x \neq y$) la funzione Dis è chiamata *pseudo-metrica*. In relazione all'ultimo punto, possiamo vedere diversi comportamenti a seconda della scelta del parametro critico p (figura ??). Si nota che nel caso in cui $p < 1$ la disegualanza triangolare non è verificata.

8.1.1.2 La distanza Ellittica

Si spiegherà ora meglio le trasformazioni fatte al cerchio per ottenere la forma ellittica. La distanza ellittica infatti permette di considerare differentemente le direzioni nella valutazioni della distanza.

Consideriamo le seguenti matrice:

$$R = \begin{bmatrix} \cos(-45^\circ) & -\sin(-45^\circ) \\ \sin(-45^\circ) & \cos(-45^\circ) \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

$$S = \begin{bmatrix} 1/2 & 0 \\ 0 & 1 \end{bmatrix}$$

$$M = \begin{bmatrix} 5/8 & -3/8 \\ -3/8 & 5/8 \end{bmatrix}$$

La matrice R descrive la rotazione in senso orario di 45 gradi, la matrice S descrive lo scaling dell'asse x di un fattore $\frac{1}{2}$. L'equazione

$$(SRx)^T(SRx) = x^T R^T S^T S Rx = x^T \underbrace{R^T S^2 R}_M x = x^T M x = 1/4$$

descrive una forma, ottenuta dalla rotazione di 45° e dallo scaling come sopra descritto, che è un cerchio con raggio $\frac{1}{2}$, ovvero l'*ellisse ascendente* che avrà equazione $\frac{5}{8}x^2 + \frac{5}{8}y^2 - \frac{3}{4}xy = (\frac{1}{2})^2$

8.1.2 La distanza di Mahalanobis

Questo ci porta a discutere del fatto che spesso le features degli esempi sono correlate tra di loro.

La matrice M dell'ellisse è spesso stimata dai dati a partire dalla matrice di covarianza invertita: $M = \Sigma^{-1}$. In particolare, si usa la definizione della *distanza di Mahalanobis* (dal nome di chi la propose):

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

dove la matrice di covarianza ha l'effetto di decorrelare e normalizzare le features.

Nella realtà potremmo fare un paragone con il caso in cui dobbiamo effettuare un viaggio secondo un dato percorso, ma dobbiamo tener conto ad esempio della presenza di vento (fattore su cui effettuare lo scaling).

Si fa inoltre notare come la distanza Euclidea è una applicazione particolare della distanza di Mahalanobis in cui la matrice di covarianza è la matrice identità:

$$\text{Dis}_2(x, y) = \text{Dis}_M(x, y | I).$$

8.1.2.1 Computazione della matrice di covarianza

Data la matrice dei dati X costruita da n oggetti ciascuno con d features

$$X = \begin{matrix} x_{11} & x_{12} & x_{13} & \cdots & x_{1d} \\ x_{21} & x_{22} & x_{23} & \cdots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & x_{n3} & \cdots & x_{nd} \end{matrix}$$

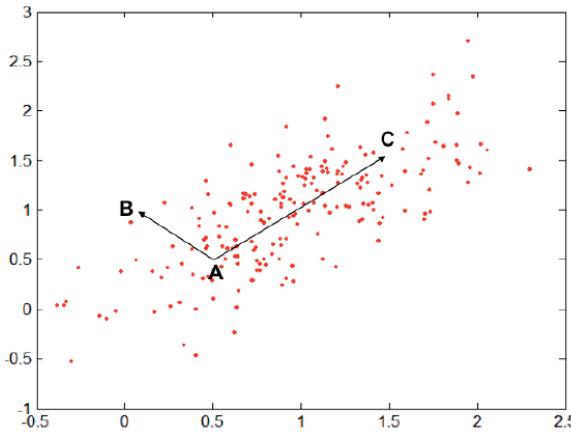


Figura 8.1: La distanza di Mahalanobis prende in considerazione diverse correlazioni tra le features.

la matrice di covarianza Σ è quadratica e d-dimensionale e ogni suo elemento σ_{lj} è la covarianza tra la features l-esima e la j-esima su tutti gli oggetti:

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

Questi σ_{lj} rappresentano una misura di quanto siano legati i cambiamenti di l e j. Se i due elementi non sono concordi il loro prodotto sarà negativo. Se sono indipendenti il valore σ_{lj} sarà approssimativamente 0. Se $l = j$ (la diagonale) la covarianza equivale alla varianza dell'attributo l. La matrice è ovviamente simmetrica.

$$\Sigma = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} & \cdots & \sigma_{1d} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} & \cdots & \sigma_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sigma_{d1} & \sigma_{d2} & \sigma_{d3} & \cdots & \sigma_{dd} \end{bmatrix}$$

8.1.2.2 Esempio della distanza di Mahalanobis

Supponiamo di avere la seguente matrice di covarianza:

$$\Sigma = \begin{bmatrix} 0.3 & 0.2 \\ 0.2 & 0.3 \end{bmatrix}$$

E gli esempi riportati come in figura 8.1 Con i seguenti punti: A : (0.5, 0.5), B : (0, 1), C : (1.5, 1.5) si otterranno le seguenti distanze di Mahalanobis $Mahal(A, B) = 5$, $Mahal(A, C) = 4$. Possiamo vedere come normalizzando tramite la matrice di covarianza Σ si riduce la distanza lungo la direzione di maggior dispersione dei dati, mentre se avessi utilizzato la distanza Euclidea avremmo ottenuto che la distanza tra A e C è molto maggiore rispetto a quella tra A e B.

8.2 VICINANZA E ESEMPLARI

8.2.1 Media e distanza

The arithmetic mean minimises squared Euclidean distance

1. La media aritmetica μ (centroide) di un insieme di dati D in uno spazio Euclideo è il punto (unico) che minimizza la somma delle distanze quadratiche dai punti.

Dimostrazione. Consideriamo

$$\operatorname{argmin}_y \sum_{x \in D} \|x - y\|^2 = \mu$$

dove $\|\cdot\|$ denota la norma 2.

Questo minimo può essere trovato sfruttando il gradiente della somma e cercando dove questo si annulla:

$$\nabla_y \sum_{x \in D} \|x - y\|^2 = -2 \sum_{x \in D} (x - y) = -2 \sum_{x \in D} x + 2 |D| y = 0$$

Da cui si ottiene derivando

$$y = \frac{1}{|D|} \sum_{x \in D} x = \mu$$

□

Questo punto è conosciuto come la *media geometrica*. Mentre per il caso univariato corrisponde al valor medio, nel caso multivariato non esiste una espressione in forma chiusa per la mediana geometrica, ma questa deve essere calcolata tramite approssimazioni successive.

In determinate situazioni ha senso individuare un esempio all'interno del dataset: in questo caso si parlerà di *medoide*, per distinguerlo dal centroide che è un esemplare che non compare necessariamente nel dataset (poiché vettore che ha come feature i-esima la media di tutti i valori della feature i-esima assunti dagli altri esempi).

La ricerca del medoide prevede di valutare per ogni elemento in input la distanza totale da tutti gli altri punti, al fine di scegliere il punto che minimizza tale somma (per n punti si tratta di una operazione con complessità $O(n^2)$).

Nella figura 8.2 è rappresentato un dataset di dieci punti in cui si riconoscono, per varie distanze metriche, i rispettivi medoidi e centroidi. Si nota inoltre come gli outlier "tirano" il centroide lontano: possiamo quindi o eliminare gli outliers oppure utilizzare misure differenti non influenzate dagli outliers, come la mediana geometrica oppure il medoide.

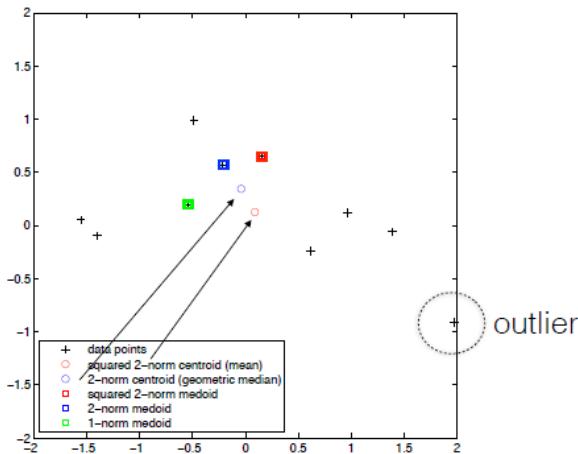
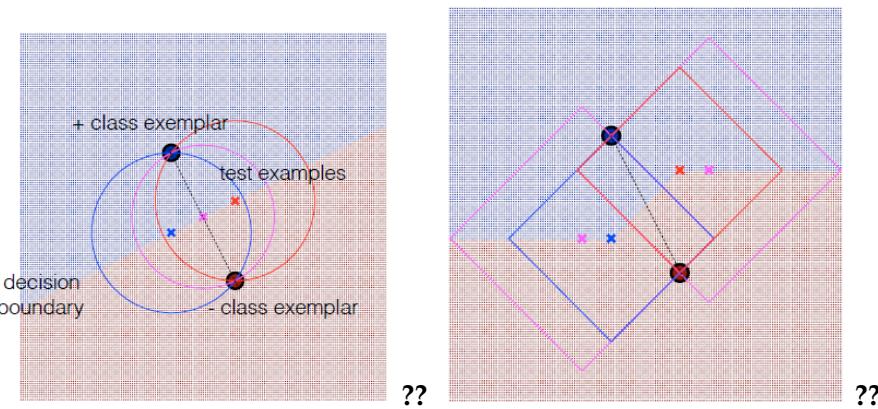


Figura 8.2: Rappresentazione dei centroidi e dei medoidi



- (a) Applicazione della regole di decisione basata sulla distanza Euclidea e confronto con il classificatore lineare di base
- (b) Misurazione basata su distanza di Manhattan

8.2.2 Il classificatore lineare di base

Il classificatore lineare di base costruisce il decision boundary come la retta perpendicolare e passante al centro del segmento che unisce i due esemplari della classe.

Un altro metodo per classificare gli esempi si basa sull'applicazione della seguente regola di decisione: *se x è più vicino all'esemplare della classe positiva μ^+ allora classifica x come positivo, altrimenti come negativo*. Se si usasse la distanza Euclidea per valutare la distanza dall'esemplare si otterebbe esattamente la stessa classificazione del classificatore lineare di base (come si vede dalla figura ??) quindi, quest'ultimo, può essere interpretato dal punto di vista della distanza come un metodo che costruisce esemplari che mirano a minimizzare il loro quadrato della distanza Euclidea da ciascuna classe, per poi applicare la regola di decisione basato sull'esemplare più vicino. Usando la distanza di Manhattan al posto della distanza Euclidea (figura ??) si

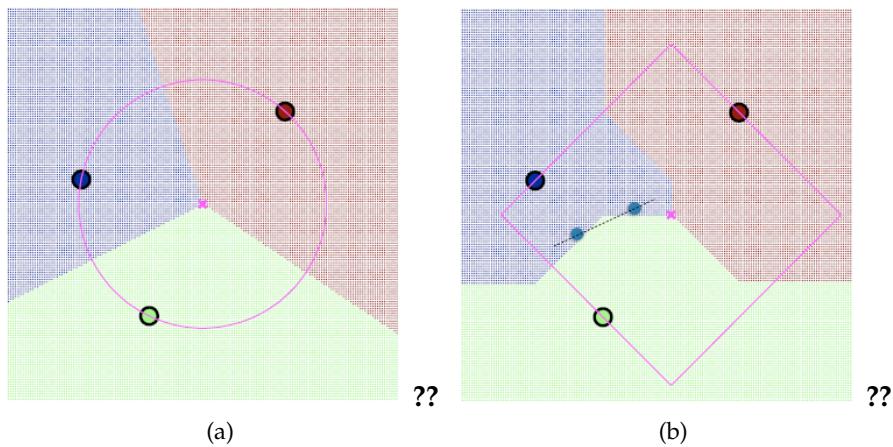


Figura 8.3: Regione di decisione definita da una regola di decisione basata sulla vicinanza dall'esemplare per tre esemplari

ottengono aree delimitate da rombi, piuttosto che da cerchi. Inoltre, la figura 8.3 si nota cosa comporta l'aumentare degli esemplari: con la distanza di Manhattan (a destra) la regione che si ottiene sarà *non convessa*, cioè dati due punti al suo interno la loro interpolazione è ancora contenuta all'interno della regione stessa. Per certi punti di vista avere una regione convessa ha determinati vantaggi.

8.2.2.1 Two esemplari is meglio che un

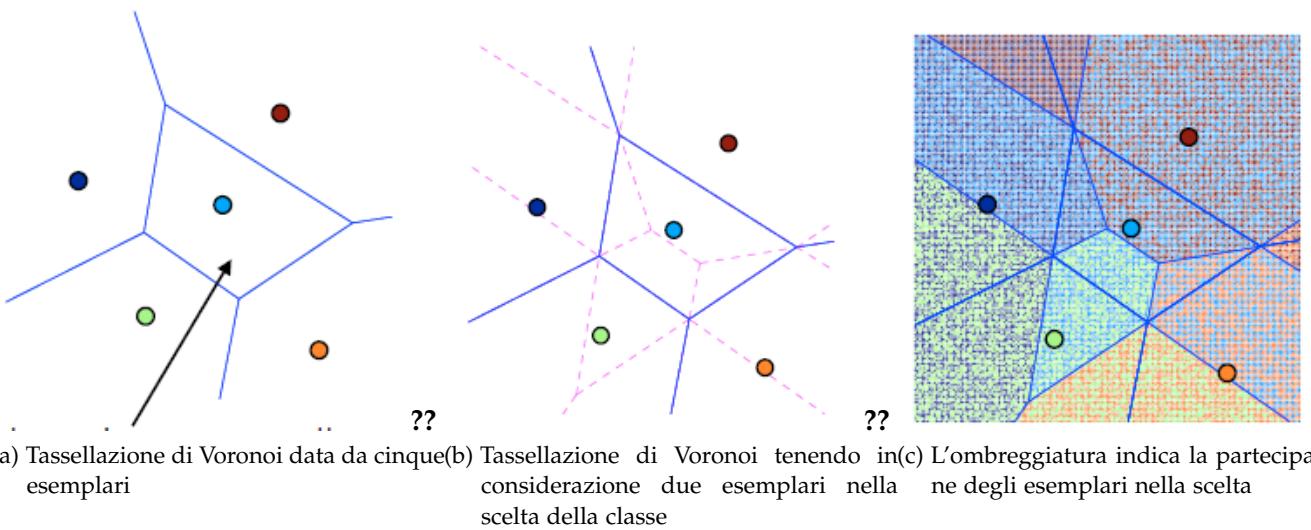
Basando la decisione della classe a cui attribuire l'esempio non su un singolo esemplare ma su più di uno si ottiene una tassellazione di Voronoi in cui si hanno più decision boundary nella linea che divide lo spazio secondo i diversi esemplari. I poligoni della tassellazione di Voronoi sono ragioni convesse. Ma si usassero due nearest-neighbor per la valutazione della classificazione (e questo è molto vantaggioso poichè nel training set sono presenti errori che potrebbero portare ad una classificazione errata ma riducibili da una seconda considerazione) la tassellazione di Voronoi verrebbe a sua volta divisa in sotto-regioni.

Si può vedere graficamente quanto spiegato nella figura ??.

8.2.3 Il punto della situazione

Riassumendo, possiamo dire che gli elementi fondamentali di un modello basato su distanza sono:

- Una distanza metrica (distanza Euclidea, di Manhattan, di Minkowski, Mahalanobis ecc)
 - Gli esemplari (centroidi o medoidi)



- Una regola basata sulla distanza, che prende in considerazione i k esemplari più vicini

A breve introdurremo diversi mix di questi elementi che ci porteranno a risolvere task supervisionati o meno.

8.3 CLASSIFICAZIONE (E CLUSTERING) NEAREST-NEIGHBOUR

8.3.1 Classificatore Nearest-neighbour

L'idea è di scegliere la classe dell'esempio di test basandoci sulla classificazione di k vicini dell'esempio di training ("più vicini"). È preferibile utilizzare un k dispari in modo da essere certi da poter selezionare la classe di maggioranza.

Esistono diverse varianti dell'algoritmo che andremo ad introdurre. Si può ad esempio tener conto della distanza stessa dei k vicini per pesare la valutazione della classe di maggioranza. Si può inoltre applicare l'algoritmo per effettuare regression.

KNN usa tutti i dati del training set, quindi il training ha complessità temporale $\mathbf{O}(n)$, così come sarà $\mathbf{O}(n)$ la procedura che porta alla predizione.

Inoltre applicare l'algoritmo a problemi multidimensionali non è così scontato (le distanze non riescono più ad effettuare una sufficiente separazione) ed è quindi prevista una selezione di range ristretto di features al fine di rendere le distanze sufficientemente utili.

La scelta di K non è scontata e non esiste a priori una regola per scegliere questo parametro.

$1NN$ separa perfettamente le classi, quindi avrà basso bias ma alta varianza. Questo è il caso in cui la predizione è più variabile, e non è rassicurante poiché basterebbe la presenza di rumore o dati del training set non rappresentativi a sbagliare di brutto la predizione. Questo

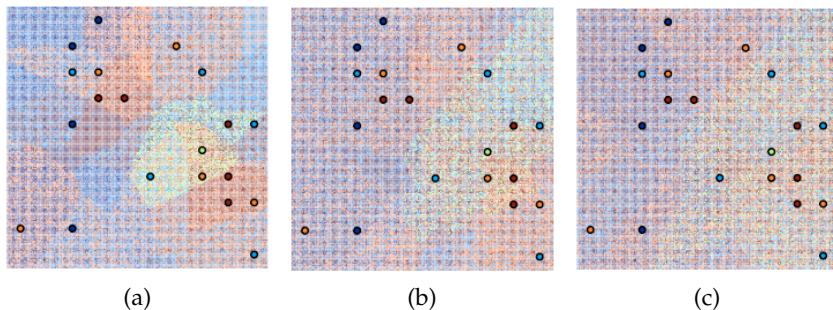


Figura 8.4: L'ombreggiatura rappresenta il distribuzione di probabilità prevista su cinque classi.

ci porta a comprendere come aumentando il parametro k si aumenta il bias e si riduce la varianza, fino a poter considerare il caso $k=n$ in cui ogni esempio sottoposto avrà lo stesso numero di vicini e riceverà la predizione dettata esattamente dalla probabilità a priori delle classi del training set. Discuteremo su questo "dilemma bias-varianza" nella sottosezione 8.3.2.

Il modello nearest-neighbour può essere facilmente adattato a valori reali, a oggetti strutturati (nearest-neighbour retrieval) e può anche trattare le probabilità (se $k > 1$, i conteggi normalizzati degli elementi sulle classi rappresentano la distribuzione delle classi stesse). Una possibilità è allora quella di applicare un peso alle distanze: ad esempio il peso può essere dato dal reciproco della distanza del test dall'esemplare.

La classificazione è quindi data dalla combinazione di raggruppamenti (*Voronoi tessellation*) e applicazione di pesi (*distanze pesate*): poiché i pesi decrescono esponenzialmente in confronto alla distanza, il conseguente aumento di k è molto più piccolo rispetto al caso in cui il voto non è ponderato. Nel machine learning c'è un acceso dibattito tra la generazione di modelli locali e modelli globali: mentre i primi usano una funzione obiettivo ottimizzata su una porzione locale del training set, gli altri utilizzano una funzione valida globalmente nel training set. Per valori piccoli di K possiamo dire che KNN si comporta come la combinazione di molti modelli locali (in ciascuna porzione del training set agirà localmente un modello) mentre con K grande avremo un unico (o pochi) modelli globali.

Se KNN è usato per la regression, possiamo aggregare le k predizioni dai k vicini per effettuarne una media, eventualmente pesando i valori sulla distanza degli esemplari.

Si chiede al lettore di non confondere questo aggregatore "k-means" con l'algoritmo k-means per il clustering che vedremo a breve. Similmente ai decision tree anche i modelli basati su distanza sono quindi generalizzabili a task differenti (regression, classificazione, clustering ecc) semplicemente cambiando la funzione di aggregazione.

8.3.2 Il dilemma bias-variance

Si ritorna ora sull'argomento della stabilità e correttezza del modello predittivo.

Con valori piccoli di k abbiamo una varianza alta (le predizioni sul test set sono più suscettibili ai piccoli cambiamenti in termini di posizione degli elementi del training set, poiché il numero di elementi considerati è piccolo) e un bias basso (l'errore sistematico commesso nelle predizioni è più basso), mentre con valori grandi di k si avrà una bassa varianza (le predizioni sono più stabili nei confronti dei rumori presenti nel training set poiché i piccoli cambiamenti possono essere compensati dal grande numero di vicini considerati) ma un bias maggiore (l'errore sistematico nelle predizioni è maggiore poiché, le predizioni sono più resistenti ai cambiamenti di porzioni consistenti nel training set).

Il suggerimento è quello di iniziare con $k=1$ e, iterando, incrementare questo parametro valutandolo man mano tramite cross-validation.

8.3.3 La densità dei vicini: *dbscan*

Dando per terminata la parte su KNN, e volendo vedere un esempio di applicazione della distanza non supervisionato, introdurremo un approccio differente che utilizza i vicini non per una predizione, ma per scopi descrittivi.

DBSCAN è un algoritmo (più tipico del data mining rispetto al machine learning) basato sulla densità, dove con questa si intende il numero di punti entro un certo raggio (*Eps*). DBScan itera su tutto il dataset e valuta la densità presente in una area unitaria (data da un raggio *Eps* appunto) in esso riconoscibile. Un punto (tra tutti quelli da valutare nel test set) è selezionato come centrale se ha più di un certo numero (*MinPts*) di punti entro l'*Eps*. Un punto di bordo ha meno *MinPts* ma è nella vicinanza del punto di centro. Un punto di rumore è un qualunque punto che non è né un punto di bordo né un punto di centro.

Nella figura ?? è riportato un esempio di punti labellate per l'applicazione di DBSCAN. Vediamo per completezza lo pseudo-codice di DBSCAN (algoritmo 17). Possiamo vedere in figura ?? un esempio

Algorithm 17 LearnRuleList(D) - Learn an ordered list of rules

Labellare tutti i punti come "centri", "bordi" o "rumore"

Eliminare i punti di rumore

Porre un arco tra i punti di centro entro *Eps* e gli altri ▷ Il fatto che il grafo sarà connesso figura 8.6 ci permetterà di definire il cluster (punto 4)

Porre ogni gruppo di punti di centro connessi in un cluster separato
Assegnare i punti di bordo ai cluster dei rispettivi punti di centro

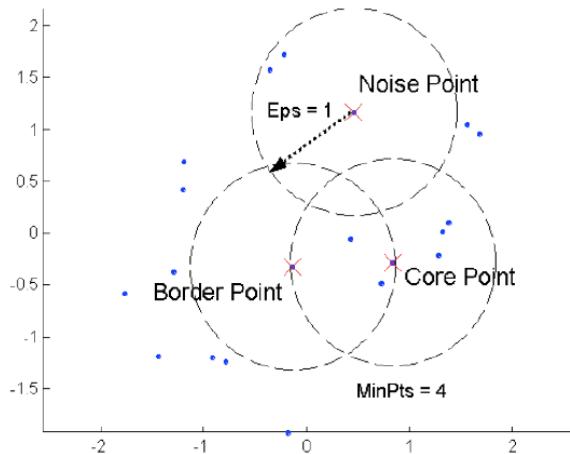


Figura 8.5: Esempio di individuazione dei punti di centro, di bordo e di rumore per DBSCAN

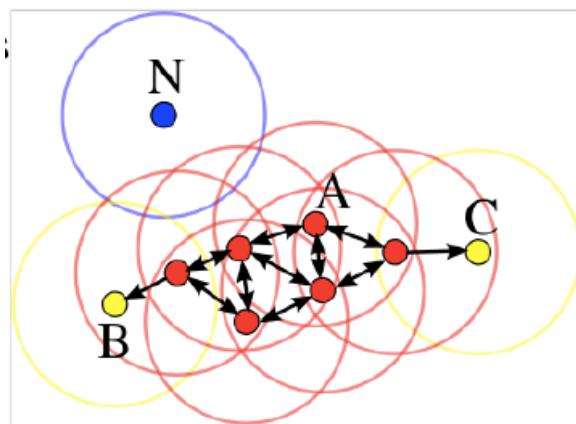


Figura 8.6: Rappresentazione grafica del punto 3 dell'algoritmo DBScan

dell'operato dell'algoritmo. Si nota che la forma dei cluster ottenuti tende a seguire la distribuzione dei dati e questo non è un fenomeno comune agli algoritmi. In figura ?? vediamo invece un funzionamento non corretto dell'algoritmo. Questo capita poiché DBScan considera una densità standard generica e comune, e non è in grado di riconoscere cluster di densità differente ma solamente quelli riconoscibile dai parametri utilizzati. Settare i parametri per l'algoritmo DBScan non è un passaggio scontato.

Un consiglio può essere dato dall'osservazione che ogni punto in un cluster ha i suoi k vicini all'incirca alla stessa distanza. I punti rumorosi hanno i k vicini alla distanza maggiore. Quindi, nel grafo in figura 8.9 si possono ordinare le distanze di ogni punto secondo il loro k -esimo vicino. Questo ci suggerisce che se la distanza è elevata l'esempio è un outlier (lontano dagli altri esempi) mentre se la distanza è bassa il punto si trova in una regione densa.

Ci aspettiamo che la scelta dei parametri rispetti questo ragionamento.

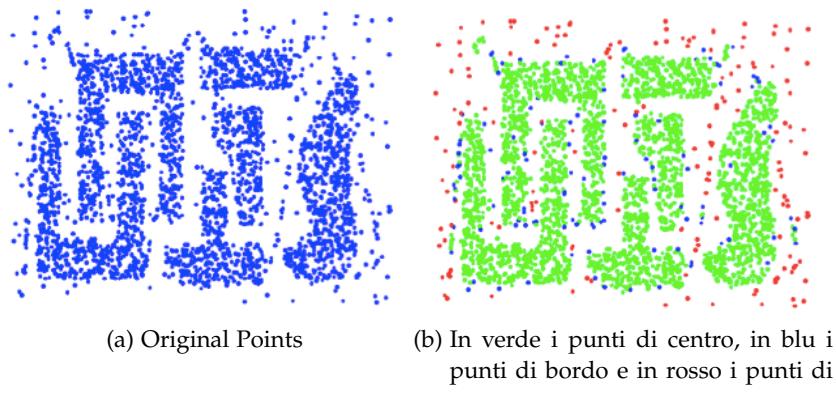


Figura 8.7: Eps=10, MinPts = 4

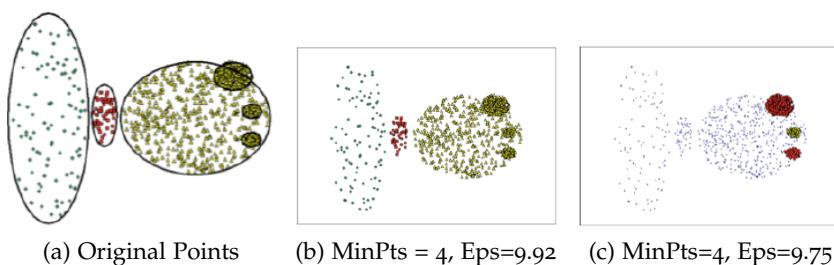


Figura 8.8: In questo caso varia la desità tra i cluster e il numero di dati è particolarmente elevato

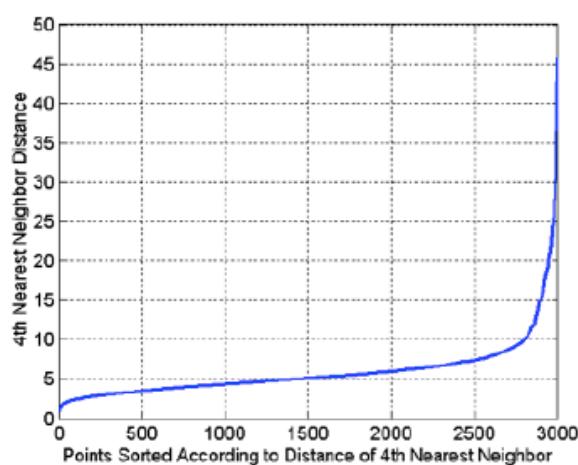


Figura 8.9: Andamento della distanza

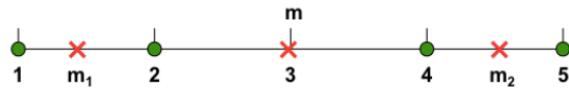


Figura 8.10: Esempio di dataset per la valutazione di SSE al variare di K = numero di cluster

8.4 CLUSTERING BASATO SU DISTANZA

Vediamo ora un altro algoritmo non-supervisionato per effettuare del clustering descrittivo basato sulla distanza.

Una volta generati i cluster abbiamo il problema di effettuarne una valutazione (ricordiamo, senza avere un test set) e a tal fine iniziamo ad utilizzare il SSE (come già visto per i decision tree utilizzati per clustering) per la definizione di:

- **Cluster cohesion:** misura quanto strettamente collegati sono gli elementi in un cluster. Un esempio di questa misura è dato dalla somma degli quadrati (SSE) entro un (singolo) cluster: $WSS = \sum_i \sum_{x \in X_i} (x - m_i)^2$, se il cluster è molto denso ci aspettiamo (e speriamo) una bassa cohesion.
- **Cluster separation:** misura quanto distinto o ben separato è un cluster da un altro. La separazione è misurata dalla somma dei quadrati tra i cluster: $BSS = \sum_i |C_i| (m - m_i)^2$ dove $|C_i|$ è la cardinalità del cluster i , e si spera che la separazione sia alta.

L'SSE totale, dato dalla somma di BSS e WSS è costante in tutto il dataset (anche perchè altro non è che la varianza su tutto il dataset). Possiamo confermare questa affermazione grazie ad un semplice esempio: supponiamo di avere il dataset rappresentato dalla figura 8.10 e valutiamo SSE totale variando K (dove ora con K si intende il numero di cluster e non il numero di vicini come nelle sezioni precedenti).

$$K = 1 \text{ cluster : } \begin{cases} WSS = (1 - 3)^2 + (2 - 3)^2 + (4 - 3)^2 + (5 - 3)^2 = 10 \\ BSS = 4 \cdot (3 - 3)^2 = 0 \\ \text{Total} = 10 + 0 = 10 \end{cases}$$

$$K = 2 \text{ cluster : } \begin{cases} WSS = (1 - 1.5)^2 + (2 - 1.5)^2 + (4 - 4.5)^2 + (5 - 4.5)^2 = 1 \\ BSS = 2 \cdot (3 - 1.5)^2 + 2 \cdot (4.5 - 3)^2 = 9 \\ \text{Total} = 1 + 9 = 10 \end{cases}$$

8.4.1 Matrice di scatter

Nel caso in cui ci si trovi in esempi multivariati si può computare L'SSE tramite la *matrice di scatter*.

Data una matrice X , la matrice scatter è la matrice

$$S = (X - 1\mu)^T(X - 1\mu) = \sum_{i=1}^n (X_i - \mu)^T(X_i - \mu)$$

dove $\mu = (\mu_1, \dots, \mu_d)$ e $\mu_j = \frac{1}{n} \sum_{i=1}^n x_{ij}$. Quindi μ è il vettore contenente i centroidi delle colonne di X .

Lo scatter di X è definito come $\text{Scat}(X) = \sum_{i=1}^n \|X_i - \mu\|^2$ che è l'equivalente di sommare tutti gli elementi sulla diagonale della matrice scatter. $\text{Scat}(X)$ equivale anche alla somma degli errori quadratici (SSE) tra ciascun esempio (riga di X) e il centroide μ . Ricordiamo qui quanto visto nel capitolo 5 circa la relazione tra la varianza totale (somma delle varianze delle singole features) e la distanza euclidea quadratica totale dal centroide:

$$= \sum_{i=1}^n \|x_i - \mu\|^2 = \sum_{i=1}^n \sum_{j=1}^d (x_{ij} - \mu_j)^2 = n \sum_{j=1}^d \sigma_j^2 = n \cdot \sigma^2$$

Se si immagina di suddividere il dataset in K partizioni $D_1 \cup \dots \cup D_K = D$ e stabilendo μ_j il centroide di D_j e sia S_j la matrice di scatter della partizione D_j allora

- $S = \sum_{j=1}^K S_j + B$
- B è la matrice di scatter tra i cluster che si ottiene rimpiazzando ciascun punto in D col corrispondente centroide μ_j . Questa matrice descrive lo spread dei centroidi.
- Ciascuna matrice S_j è chiamata matrice di scatter entro il cluster e descrive la compattezza del j -esimo cluster.
- Ne consegue che

$$\text{Scat}(D) = \sum_{j=1}^K \text{Scat}(D_j) + \sum_{j=1}^K |D_j| \|\mu_j - \mu\|^2$$

8.4.2 Algoritmo K-means

Il problema K – means è quello di trovare una partizione che minimizza il primo termine (o massimizza il secondo).

È un problema NP-completo (esponenziale nel numero di punti) e non esiste una soluzione efficiente per la ricerca del minimo globale (per questo nell'algoritmo si introduce una euristica che porta ad una complessità lineare nel numero di oggetti e di features).

L'algoritmo *K-means* (o *Lloyd's algorithm*) itera partizionando il dataset secondo una regola di decisione basata sul centroide più vicino (ad ogni centroide individuato si associa una porzione di dataset), per poi ricalcolare il centroide per ogni partizione così ottenuta.

L'algoritmo converge ad uno stato stabile in un numero finito di iterazioni (ciascuna infatti riduce gli errori nei cluster), ma non c'è garanzia circa il fatto che la soluzione trovata sia un minimo globale: nella pratica è utile eseguire l'algoritmo più volte e selezionare la soluzione che riduce l'errore quadratico (SSE) entro i cluster.

8.4.2.1 Esempio di riduzione dello scatter partizionando il dataset

Si consideri i cinque seguenti punti centrati attorno all'origine $(0, 0)$: $\{(0, 3), (3, 3), (3, 0), (-2, -4), (-4, -2)\}$. La matrice di scatter sarà:

$$S = \begin{pmatrix} 0 & 3 & 3 & -2 & -4 \\ 3 & 3 & 0 & -4 & -2 \end{pmatrix} \begin{pmatrix} 0 & 3 \\ 3 & 3 \\ 3 & 0 \\ -2 & -4 \\ -4 & -2 \end{pmatrix} = \begin{pmatrix} 38 & 25 \\ 25 & 38 \end{pmatrix}$$

dove $\text{Scat}(D) = \text{sommadeglielementisulladiagonale} = 38+38=76$.

Se posizioniamo in un cluster i primi due punti e lasciamo i tre punti rimanenti in un secondo cluster otterremo i centroidi $\mu_1 = (1.5, 3)$ e $\mu_2 = (-1, -2)$ e le matrici di scatter entro i centroidi saranno:

$$S_1 = \begin{pmatrix} 0 - 1.5 & 3 - 1.5 \\ 3 - 3 & 3 - 3 \end{pmatrix} \begin{pmatrix} 0 - 1.5 & 3 - 3 \\ 3 - 1.5 & 3 - 3 \end{pmatrix} = \begin{pmatrix} 4.5 & 0 \\ 0 & 0 \end{pmatrix}$$

$$S_2 = \begin{pmatrix} 0 - (-1) & -2 - (-2) & -4 - (-4) \\ 0 - (-2) & -4 - (-2) & -2 - (-2) \end{pmatrix} \begin{pmatrix} 3 - (-1) & 0 - (-2) \\ -2 - (-1) & -4 - (-2) \\ -4 - (-1) & -2 - (-2) \end{pmatrix} = \begin{pmatrix} 26 & 10 \\ 10 & 8 \end{pmatrix}$$

Due copie di μ_1 e tre copie di μ_2 hanno, per definizione, lo stesso centro del dataset completo $(0, 0)$. Si può allora calcolare la matrice di scutter tra i cluster come

$$B = \begin{pmatrix} 1.5 & 1.5 & -1 & -1 & -1 \\ 3 & 3 & -2 & -2 & -2 \end{pmatrix} \begin{pmatrix} 1.5 & 3 \\ 1.5 & 3 \\ -1 & -2 \\ -1 & -2 \\ -1 & -2 \end{pmatrix} = \begin{pmatrix} 7.5 & 15 \\ 15 & 30 \end{pmatrix}$$

Con traccia 37.5.

Alternativamente, se trattiamo i primi tre punti come un cluster e gli

ultimi due in un altro si otterrebbe $\mu'_1 = (2, 2)$ e $\mu'_2 = (-3, -3)$, con la matrice di scatter entro i cluster:

$$S'_1 = \begin{pmatrix} 0-2 & 3-2 & 3-2 \\ 3-2 & 3-2 & 0-2 \end{pmatrix} \begin{pmatrix} 0-2 & 3-2 \\ 3-2 & 3-2 \\ 3-2 & 0-2 \end{pmatrix} = \begin{pmatrix} 6 & -3 \\ -3 & 6 \end{pmatrix}$$

$$S'_2 = \begin{pmatrix} -2-(-3) & -4-(-3) \\ -4-(-3) & -2-(-3) \end{pmatrix} \begin{pmatrix} -2-(-3) & -4-(-3) \\ -4-(-3) & -2-(-3) \end{pmatrix} = \begin{pmatrix} 2 & -2 \\ -2 & 2 \end{pmatrix}$$

con $\text{Scat}(D'_1) = 12$ e $\text{Scat}(D'_2) = 4$. La matrice di scatter tra cluster è

$$B' = \begin{pmatrix} 2 & 2 & 2 & -3 & -3 \\ 2 & 2 & 2 & -3 & -3 \end{pmatrix} \begin{pmatrix} 2 & 3 \\ 2 & 3 \\ 2 & 3 \\ -3 & -3 \\ -3 & -3 \end{pmatrix} = \begin{pmatrix} 30 & 30 \\ 30 & 30 \end{pmatrix}$$

con traccia 60.

Il secondo caso produce cluster più piccoli con centroidi più distanti.

8.4.2.2 Pseudo-codice dell'algoritmo

Algorithm 18 KMeans(D,K) - K-means clustering using Euclidean distance Dis_2 (funziona ovviamente anche con altre distanze)

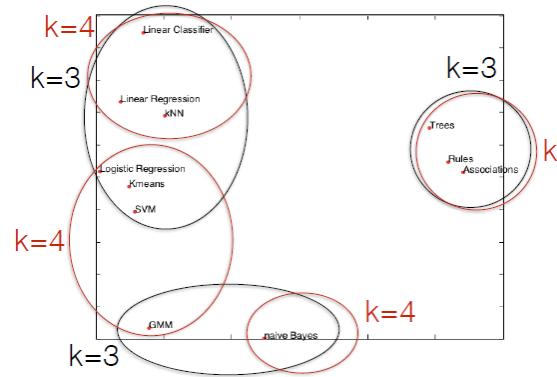
Input: dataset $D \subseteq \mathbb{R}^d$, il numero di cluster voluti $K \in \mathbb{N}$
Output: k centroidi $\{\mu_1, \dots, \mu_K \in \mathbb{R}^d\}$ rappresentanti i cluster;
randomly initialise K vectors $\{\mu_1, \dots, \mu_K \in \mathbb{R}^d\}$;
while $\{\mu_1, \dots, \mu_K\}$ has been changed **do**
 assign each $x \in D$ to $\text{argmin}_j \text{Dis}_2(x, \mu_j)$;
 ▷ l'assegnazione della label dipende dal task (classificazione,
 ranking ecc)
 for $j = 1$ to K **do**
 $D_j \rightarrow \{x \in D \mid x \text{ assigned to cluster } j\}$;
 $\mu_j = \frac{1}{|D_j|} \sum_{x \in D_j} x$
 end for
end while
return $\{\mu_1, \dots, \mu_K\}$

Questo algoritmo è semplice ed efficiente ma un suo grosso punto di debolezza di è che il parametro k deve essere stabilito a priori dall'utente, che è quindi tenuto a dover valutare attentamente questa scelta. Se un esempio è multivariato questa operazione di individuazione di k è ancora più difficile.

Sarebbe sicuro scegliere $k=n$ ma avremmo n cluster in cui ciascun dato è un centroide ed è ovviamente poco utile.

Model	Geom	Stats	Logic	Group	Grad	Disc	Real	Sup	Unsup	Multi
Trees	1	0	3	3	0	3	2	3	2	3
Rules	0	0	3	3	1	3	2	3	0	2
Naive Bayes	1	3	1	3	1	3	1	3	0	3
Knn	3	1	0	2	2	1	3	3	0	3
Linear Classifier	3	0	0	0	3	1	3	3	0	0
Linear Regression	3	1	0	0	3	0	3	3	0	1
Logistic Regression	3	2	0	0	3	1	3	3	0	0
SVM	2	2	0	0	3	2	3	3	0	0
Kmeans	3	2	0	1	2	1	3	0	3	1
GMM	1	3	0	0	3	1	3	0	3	1
Associations	0	0	3	3	0	3	1	0	3	1

(a) Rappresentazione del dataset dei modelli presenti nel machine learning



(b) Mapping risultante dalla rappresentazione dei modelli del machine learning con KMeans con K = 4

Inoltre, come già detto, nonostante la convergenza non è detto che il minimo trovato sia globale e occorre eseguire più volte al fine di selezionare la soluzione migliore (vedremo un esempio nella sottosezione 8.4.2.4).

8.4.2.3 Meta Learning usando clustering

Vogliamo effettuare del clustering descrittivo sul dataset dei modelli presenti nel machine learning. Il dataset è descritto dalla tabella in figura 8.11a in cui abbiamo una riga per ciascun modello, ciascuno descritto da un punteggio rappresentante il fatto che il modello ha determinate abilità nel rappresentare le caratteristiche relazionali (presenti nelle colonne) tra i punti. Eseguendo un algoritmo di clustering (KMeans) sperando di riuscire a descrivere le similarità tra i modelli si può ottenere il mapping in figura 8.11b. Modificando il parametro k su questo algoritmo il mapping cambia.

8.4.2.4 I punti stazionari nel clustering

Consideriamo di avere un insieme di punti $\{8, 44, 50, 58, 84\}$ e di volerlo suddividere in due cluster. Esistono 4 possibili partizioni che l'algoritmo 2-Means può trovare:

- $\{8\}, \{44, 50, 58, 84\}$
- $\{8, 44\}, \{50, 58, 84\}$
- $\{8, 44, 50\}, \{58, 84\}$
- $\{8, 44, 50, 58\}, \{84\}$

Tutte e queste quattro partizioni conducono ad un punto stazionario per 2-Means, ma solamente la prima è ottima e minimizza il WSS (le

altre sono sub-optimali).

Il suggerimento (già presentato) è quello di eseguire più volte l'algoritmo con diverse inizializzazioni e selezionare la soluzione migliore.

8.4.3 Clustering attorno ai medoidi

E se usassimo i medoidi invece dei centroidi? È sufficiente modificare l'inizializzazione e considerare dei punti del dataset al posto dei centroidi. Questo è utile poiché il centroide è affetto da modifiche errate a causa della presenza di outlier, mentre il medoide è un dato interno del dataset riconosciuto come "il più vicino" agli altri dati.

Vediamo quindi questa variante dell'algoritmo, nello pseudocodice ??.

8.4.3.1 Algoritmo K-medoids Clustering

Algorithm 19 KMedoids(D,K,Dis) - K-medoids clustering using arbitrary distance metric Dis

```

1: Input: dataset D ⊆ X, il numero di cluster voluti K ∈ N
2: Output: k medoidi {μ1, ..., μK ∈ D} rappresentanti i cluster; una
distanza Dis : XxX → R
3: randomly pick K data points {μ1, ..., μK ∈ D};
4: while {μ1, ..., μK} has been changed do
5:   assign each x ∈ D to argminjDis(x, μj);
6:
7:   for j = 1 to K do
8:     Dj → {x ∈ D | x assigned to cluster j};
9:     μj = argminx ∈ Dj ∑x' ∈ Dj Dis(x, x')
10:  end for
11: end while
12: return {μ1, ..., μK}
```

Esiste anche una versione in cui viene esplicitata una funzione che computa l'errore quadratico medio (WSS) rispetto al medoide corrente (algoritmo ??). Calcolata questa funzione prova ad invertire il ruolo del medoide con gli oggetti e valuta quale combinazione si comporta meglio. La complessità dell'algoritmo 20 è $\mathcal{O}(n^2)$ ma può essere eseguito su un campione alla volta (ad ogni iterazione).

8.4.3.2 Svantaggi di K-means

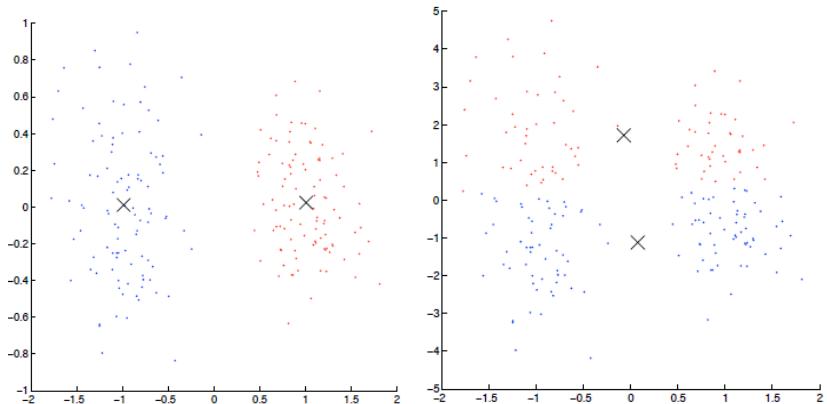
L'algoritmo K-means ha parecchie caratteristiche positive (linearità, convergenza ecc) ma purtroppo ha lo svantaggio che tende a creare cluster di forma globulare (poiché tende a minimizzare SSE) e nel caso di un rescaling rispetto ad un asse (come mostrato in figura ??) tende a spezzare i cluster in metà.

Algorithm 20 PAM(D, K, Dis) - Partitioning around medoids clustering using arbitrary distance metric Dis

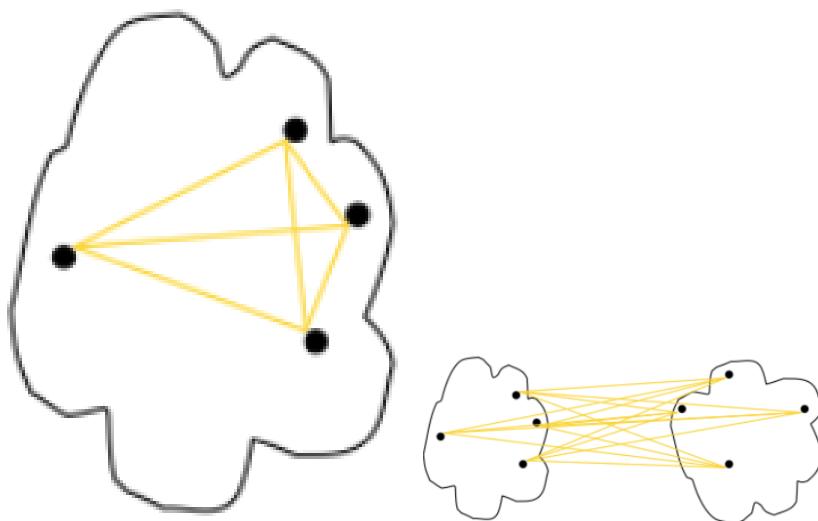
```

1: Input: dataset  $D \subseteq \mathbf{X}$ , il numero di cluster voluti  $K \in \mathbf{N}$ 
2: Output:  $k$  medoidi  $\{\mu_1, \dots, \mu_K \in D\}$  rappresentanti i cluster; una
distanza  $\text{Dis} : \mathbf{X} \times \mathbf{X} \leftarrow \mathbf{R}$ 
3: randomly pick  $K$  data points  $\{\mu_1, \dots, \mu_K \in D\}$ ;
4: while no further improvement possible do
5:   assign each  $x \in D$  to  $\arg\min_j \text{Dis}(x, \mu_j)$ ;
6:   for  $j = 1$  to  $K$  do
7:      $D_j \rightarrow \{x \in D \mid x \text{ assigned to cluster } j\}$ ;
8:   end for
9:    $Q \rightarrow \sum_j \sum_{x \in D_j} \text{Dis}(x, \mu_j)$ ;
10:   $\triangleright$  compute  $Q$  for clustering quality (on WSS)
11:  for each medoid  $\mu_j$  and each non-medoid  $\mathbf{o}$  do
12:    calculate the improvement in  $Q$  resulting from swapping
 $\mu_j$  with  $\mathbf{o}$ ;
13:  end for
14:  select the pair with maximum improvement and swap;
15: end while
16: return  $\{\mu_1, \dots, \mu_K\}$ 

```



(a) Dataset non riscalato: 2-Means seleziona correttamente i cluster
(b) Dataset riscalato rispetto all'asse y: 2-Means perde la sua efficacia



- (a) La cohesion è la somma dei pesi dei link all'interno di un cluster
(b) La separation è la somma dei pesi tra i nodi di cluster e i nodi al di fuori del cluster

8.4.4 Silhouettes

La cluster cohesion e la cluster separation possono anche essere rappresentate e intuite tramite una rappresentazione a grafo, come mostrato in figura ???. Questa intuizione ci porta ad introdurre una nuova misura di valutazione: la *silhouette*.

È una misura molto popolare nel clustering poichè è in grado allo stesso tempo di ridurre la somma dei pesi tra nodi dello stesso cluster (cohesion) e di massimizzare la separation.

Per ogni esempio nel dataset x_i , sia $d(x_i, D_j)$ la distanza media di x_i dai punti nel cluster D_j e sia $j(i)$ l'indice del cluster a cui x_i appartiene. Sia inoltre $a(x_i) = d(x_i, D_{j(i)})$ la distanza media di x_i dal punto del suo cluster $D_{j(i)}$ e sia $b(x_i) = \min_{k \neq j(i)} d(x_i, D_k)$ la distanza media dai punti nel suo cluster k vicino.

Ci si aspetta $a(x_i) < b(x_i)$ ma ciò non è garantito.

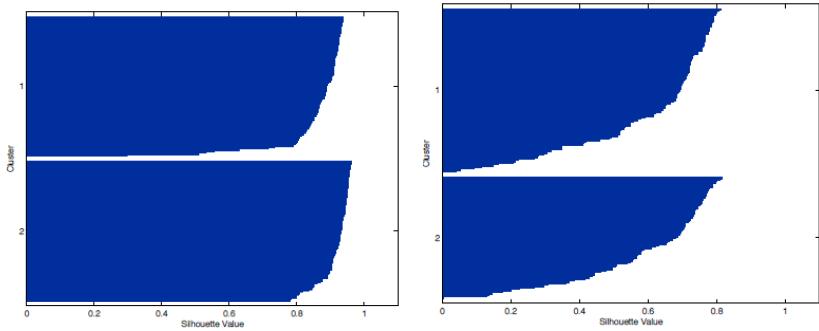
Quindi possiamo considerare la differenza $b(x_i) - a(x_i)$ come un indice di quando è "ben clusterizzato" x_i , e dividendo questa differenza per $b(x_i)$ si ottiene un indice compreso tra 0 e 1.

Nel caso in cui $a(x_i) > b(x_i)$ si deduce che, in media, i membri nei cluster vicini sono più vicini a x_i rispetto ai membri dello stesso cluster di x_i . In questo caso dividendo per $a(x_i)$ si ottiene la seguente definizione:

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

Più alta è la silhouette $s(x_i)$, migliore è la clusterizzazione di x_i . La silhouette ordina e traccia il grafico $s(x)$ per ogni istanza, raggruppata nei cluster.

Nelle figure 8.11a e 8.11b i grafici delle silhouette $s(x_i)$ degli esempi



(a) Grafici di silhouette riferiti agli esempi presentati sopra. Viene usata la distanza Euclidea e si nota che tutti i punti hanno una $s(x)$ molto alta, a rappresentanza che gli esempi sono molto più vicini, in media, agli altri membri del cluster piuttosto che ai membri dei cluster vicini

(b) La silhouette del clustering in questa figura è meno convincente rispetto a quello della figura 8.11a ma comunque tutti i valori sono positivi

introdotti precedentemente: le barre orizzontali dei punti x_i nei cluster ordinati secondo i valori della silhouette $s(x_i)$.

La silhouette 8.11a è più convincente rispetto a quella della figura a fianco, poiché a barra più lunghe corrisponde un miglior clustering (1 è il valore massimo della silhouette).

8.5 CLUSTERING GERARCHICO

Ci spostiamo ora dal clustering mirato ad effettuare un partizionamento in favore di cluster mirato a creare gerarchie: si usano cluster che creano suddivisioni in termini di Diagramma di Venn, o alberi (questi ultimi più informativi circa le relazioni tra gli esempi).

Ripartiamo dal dataset dei modelli del machine learning. Nella sottosezione 8.4.2.3 avevamo lanciato k-means con diversi valori di k ottenendo diversi risultati e questo non ci ha lasciato particolarmente soddisfatti. Proveremo dunque ad approcciare lo stesso problema con il clustering gerarchico, che non richiede di fissare il valore dei cluster a priori, ma tende a formare un albero e ad individuare in esso gli oggetti con foglie raggruppabili. Si può vedere un esempio di questo albero in figura 8.11. I cluster vengono poi scelti successivamente in base ai raggruppamenti che si preferiscono.

Il dendogramma, riesce inoltre così a suggerire un distanza tra i vari punti del dataset (punti più vicini vengono raggruppati anche con k più grande) poiché dati molto diversi vengono separati (o uniti, a seconda che si usi un algoritmo divisivo o agglomerativo) a livelli alti dell'albero.

Più formalmente, dato un set di elementi D, un *dendrogramma* è un albero binario con gli elementi di D nelle sue foglie e con sottoinsiemi di D nei nodi intermedi. Il livello del nodo rappresenta la distanza tra

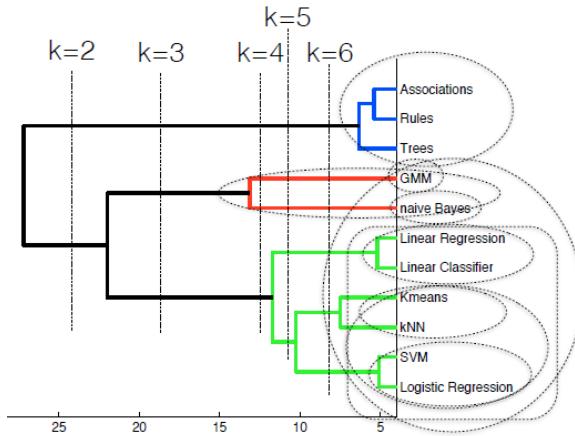


Figura 8.11: Dendrogramma raffigurante l'albero gerarchico dei possibili clustering dei modelli. I punti neri e quelli rossi rappresentano rispettivamente l'appartenenza a due diverse classi.

i due cluster rappresentati dai figli dei nodi (le foglie hanno livello 0). Questo ci fa capire come questo albero ci possa dare già parecchie informazioni circa il dataset. Si consideri ad esempio l'applicazione alle specie animali per studiarne l'evoluzione secondo caratteristiche comuni.

8.5.1 Le funzioni di collegamento

Una *funzione di collegamento* (*linkage function*) $L : 2^{\mathbb{X}} \times 2^{\mathbb{X}} \rightarrow \mathbb{R}$ calcola la distanza tra due sottoinsiemi arbitrari dello spazio delle istanze, data una metrica di distanza $\text{Dis} : \mathbb{X} \times \mathbb{X} \rightarrow \mathbb{R}$. Le funzioni di collegamento di maggiore uso sono (ne abbiamo anche una rappresentazione grafica in figura 8.12):

- Single linkage (visione ottimistica, la distanza tra due cluster è la minore esistente tra due dei loro elementi)
- Complete linkage (visione pessimistica, la distanza tra due cluster è la maggiore esistente tra due dei loro elementi)
- Average linkage (la distanza tra due cluster è la media esistente tra tutti i collegamenti esistenti tra i loro elementi)
- Centroid linkage (la distanza tra due cluster è la distanza esistente tra i centroidi dei cluster)

Chiaramente tutte queste funzioni coincidono per i cluster singleton, mentre all'aumentare della cardinalità i valori divergono.

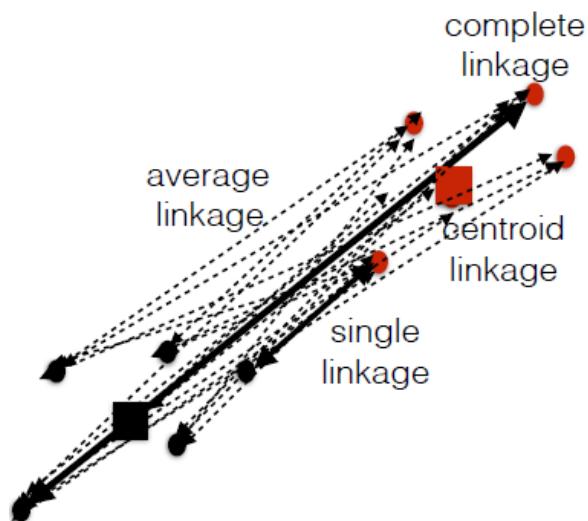
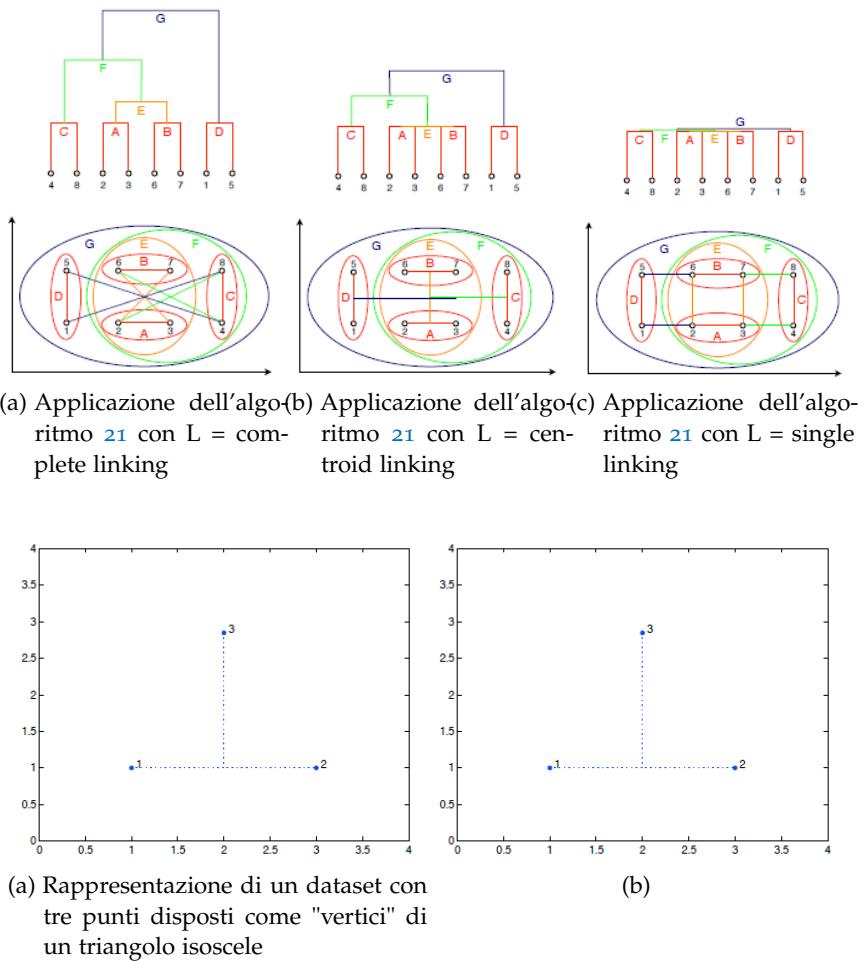


Figura 8.12: Rappresentazione grafica delle principali linkage function

Algorithm 21 HAC(D, L) - Hierarchical agglomerative clustering

Input: dataset $D \subseteq X$, funzione di collegamento L
Output: un dendrogramma rappresentante un clustering descrittivo di D

Inizializza i clusters con singleton dei punti del dataset;
while Tutti i punti del dataset sono in un cluster **do**
 trovare la coppia di clusters (X, Y) con il più basso linkaggio 1 e
 farne il merge;
 Creare un nodo figlio di (X, Y) al livello 1
end while
return L'albero binario creatosi



8.5.2 Algoritmo per il cluster gerarchico agglomerativo

8.5.3 Il dibattito sul linkaggio

Presentiamo ora un dataset che come vedremo pone difficoltà alla funzione di collegamento.

Consideriamo una tabella regolare di 8 punti divisi in due righe da quattro e assumiamo che i punti vicini siano alla stessa distanza tra di loro.

Applichiamo quindi sia complete linkage (figura 8.13a), sia centroid linkage (figura 8.13b), sia single linkage (figura 8.13c) e analizziamone le rispettive figure. Vogliamo in particolare far notare come se i punti sono posti in una griglia regolare e la distanza dei punti è esattamente la stessa non è possibile rappresentare una particolare distanza tra i cluster e quindi non può esistere un vero e proprio dendrogramma (questo infatti "collappa"). Inoltre, un'altra problematica è distinguibile dal seguente esempio. Supponiamo di avere tre punti disposti come in figura ?? Se supponiamo di usare Centroid Linkage, essendo i punti 1 e 2 più vicini l'algoritmo porta ad effettuare prima un merge su

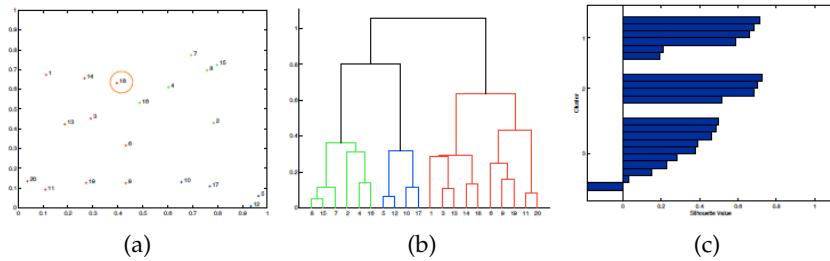


Figura 8.13: Valutazione di una distribuzione generata random a partire da campionamento uniforme di 20 elementi

essi in un primo cluster con livello pari alla distanza tra i due punti. Se ora si riusa il centroid linkage per calcolare la distanza tra il cluster (1,2) e il singleton 3 si ottiene una distanza inferiore rispetto a quella tra il punto 1 e il 2 (valutata precedentemente e quindi il livello del terzo cluster sarà inferiore rispetto a quello del cluster (1,2), definendo un *dendrogramma non monotono* in cui i livelli dei clustering trovati non hanno andamento monotono. Scegliendo differenti funzioni di collegamento (per esempio average linkage) questo problema non si manifesterebbe e questo introduce una difficoltà nella scelta corretta di questa funzione.

8.5.3.1 Spurious clustering

Applicando un approccio spesso utilizzato nelle tecniche di data mining si potrebbe pensare di verificare l'applicabilità del dataset al clustering confrontando il clustering ottenuto con un clustering ottenuto a partire da una distribuzione random (figura 8.13a): se sono in qualche modo simili non è un buon segnale poiché significa che il clustering effettuato non è poi tanto più informativo rispetto ad un clustering randomico.

Anche una silhouette (figura ??) con valori decrescenti possono confermare l'assenza di una struttura di cluster robusta, e valori negativi (punto 18) significano che il punto è in media più vicino ai punti di un altro cluster.

8.6 DAI KERNEL ALLE DISTANZE

Vedremo come usare i Kernel introdotti nel capitolo ?? per computare le distanze.

Un kernel è una funzione di similarità tra i punti del dataset:

$$K(x_i, x_j) = \varphi(x_i) \cdot \varphi(x_j)$$

Computando questa funzione un prodotto scalare nello spazio delle features (senza costruire il vettore delle feature $\varphi(x)$ esplicitamente),

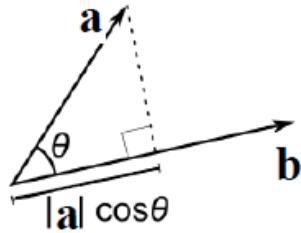


Figura 8.14: Rappresentazione geometrica del prodotto scalare

possiamo dire che un qualsiasi metodo di apprendimento che può essere definito in termini di prodotto scalare è riconducibile ad una "kernelizzazione".

8.6.1 La kernelizzazione

Si può applicare, ad esempio, lo stesso "Kernel Trick" a molti metodi di apprendimento basati su distanza e questo grazie al fatto che la distanza Euclidea può essere riscritta in termini di prodotto scalare.

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

I termini $x \cdot x$ e $y \cdot y$ hanno l'effetto di rendere l'intera espressione *translation-invariant* (cosa che il prodotto scalare non è). Quindi il prodotto scalare è una misura di similarità, ed in particolare al suo aumentare la distanza decresce.

Ricordiamo che il prodotto scalare $a \cdot b$ è definito come $\sum_{i=1}^n a_i b_i$, mentre in termini geometrici il prodotto scalare tra due vettori Euclidei è definito come $a \cdot b = \|a\| \|b\| \cos(\Theta)$, come spiegato dalla figura 8.14. A questo punto dovremmo essere maggiormente sicuri del background necessario a proseguire.

Sostituendo il prodotto scalare on una funzione kernel k si ottiene la seguente distanza kernelizzata:

$$\text{Dis}_k(x, y) = \sqrt{k(x, x) - 2k(x, y) + k(y, y)}$$

8.6.2 Algoritmo K-Means Kernelizzato

8.6.3 Cosine Similarity

Algorithm 22 Kernel-KMeans(D, K) - K-means clustering using kernelised distance Dis_k

Input: dataset $D \subseteq X$, numero di $K \in \mathbb{N}$
Output: K-fold partition $D_1 \cup \dots \cup D_k = D$
randomly initialise K clusters $D_1 \cup \dots \cup D_k$;
while $D_1 \cup \dots \cup D_k = D$ has no changed **do**
 assign each $x \in D$ to $\operatorname{argmin}_j \frac{1}{|D_j| \sum_{y \in D_j} Dis_k(x, y)}$;
 for $j = 1$ to K **do**
 $D_j \leftarrow \{x \in D \mid x \text{ assigned to cluster } j\}$
 end for
end while
return $D_1 \cup \dots \cup D_k$

MODELLI PROBABILISTICI

In questo capitolo verranno presentate due tipologie diverse di modelli probabilistici: i modelli *discriminativi* (di cui si possono già citare le **SVM** viste nel Capiolo 7) che si concentrano a modellare la zona dello spazio delle istanze in cui è presente la separazione delle classi, e i modelli *generativi* che mirano non a modellare il bordo, bensì la legge che governa la classe e che sarà sperabilmente replicabile altrove in quanto dedotta dagli esempi che hanno generato una distribuzione di probabilità su quella classe. Per questi ultimi si porrà particolare attenzione al caso della distribuzione Gaussiana e si vedrà come il classificatore di base lineare potrà essere interpretato in modo geometrico e quando soddisfa la proprietà ottima di Bayes che porta a distinguere quali classificatori sono ottimali secondo la legge di Bayes.

9.1 MODELLI DESCRITTIVI

I modelli descrittivi si focalizzano sul definire le differenze tra classi apprendendo una funzione discriminativa che modella il decision boundary tra la zona degli esempi di una classe e quelli dell'altra. Le classi corrispondono a valori di variabili indipendenti (*target*) restituiti dalla funzione d -dimensionale dove d è il numero di feature rappresentate poi nel vettore $V = v_1, \dots, v_d$. Si indica inoltre con C_k la classe k -esima ($k \in \{1, \dots, L\}$).

Si introduce quindi una definizione della funzione discriminante. Per ogni classe k la funzione discriminante $g_k(v) = g_k(v_1, \dots, v_d)$ mappa ogni esempio descritto d -dimensionalmente in un numero reale.

Il confine tra due classi C_i e C_k è determinato dalla differenza $g_i(V) - g_k(V) = 0$. Tale differenza deve essere nulla perché qua le due classi si incontrano e non si è in grado di decidere tra di esse in quanto i punti si trovano esattamente a cavallo.

La funzione è discreta e se le feature non lo fossero occorrerebbe una fase di preprocessing per discretizzarle. La funzione discriminante determina quindi $\frac{L(L-1)}{2}$ confini tra coppie di classi.

Lo scopo è quello di trovare una funzione g_k più semplice possibile in modo che per ogni esempio t_i (supponendo essere C_i la classe positiva) avremo che $g_i(t_i) > g_k(t_i), \forall k \neq i$.

Si definiscono inoltre:

FUNZIONE DI REGRESSIONE una funzione numerica che mappa un esempio t_i con una approssimazione $g(t_i)$ del valore reale della

variabile continua. Tanto migliore è questa funzione, tanto minore è l'errore che commetto con l'approssimazione. Non è in realtà una funzione discriminativa ma dal punto di vista della struttura sono spesso simili.

FUNZIONE LINEARE una combinazione lineare dei valori delle feature $g_i(V) = w_1 \cdot v_1 + w_2 \cdot v_2 + \dots + w_d \cdot v_{d+1}$.

Una funzione discriminante lineare definisce un confine $G_{ij}(V) = g_i(V) - g_j(V) = 0$ tale che $(w_{i1} - w_{j1}) \cdot v_1 + (w_{i2} - w_{j2}) \cdot v_2 + \dots + (w_{id} - w_{jd}) \cdot v_d = 0$. L'obbiettivo è trovare, per ogni classe C_i il vettore pesi $[w_{i1}, w_{i2}, \dots, w_{id}]$.

Altro non è che la combinazione lineare di ciascun esempio con il vettore dei pesi w che restituisce il risultato della discriminazione.

Queste funzioni possono essere viste dal punto di vista probabilistico come delle funzioni che associano una classe al vettore x nello spazio delle feature degli esempi. Dal punto di vista probabilistico si tratta quindi di una probabilità condizionata, a posteriori, della classe sapendo di essere in un certo spazio degli esempi descritto dai valori delle features.

Queste funzioni possono essere viste dal punto di vista probabilistico come delle funzioni che associano una classe al vettore x nello spazio delle feature degli esempi. Dal punto di vista probabilistico si tratta quindi di una probabilità condizionata, a posteriori, della classe sapendo di essere in un certo spazio degli esempi descritto dai valori delle features.

9.2 I MODELLI GENERATIVI

Si introducono ora i modelli generativi e lo si fa per differenza rispetto ai modelli discriminativi.

Mentre i modelli discriminativi modellano la distribuzione di probabilità a posteriori $P(Y|X)$ dove Y è la variabile target e X sono le feature (cioè, dato X vogliamo la distribuzione di probabilità su Y) i *modelli generativi* modellano la probabilità congiunta $P(Y,X)$ del target Y e delle feature X . Ottenendo la probabilità congiunta è possibile derivare ogni probabilità marginale.

In particolare:

$$P(X) = \sum_y P(Y=y, X)$$

e la probabilità a posteriori

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

Inoltre, è possibile descrivere un modello generativo attraverso il concetto di *omiglianza* o *likelihood*, ovvero tramite la funzione $P(X|Y)$,

che permette di ottenere facilmente $P(Y|X) = P(X|Y)P(Y)$ e la distribuzione a priori.

Questi modelli sono chiamati generativi poiché dalla probabilità congiunta si possono ottenere nuovi esempi che si suppongono correttamente classificati: è possibile usare $P(Y)$ per campionare una classe oppure, a partire da $P(X|Y)$, per generare una istanza di una classe.

9.2.1 *Vantaggi e svantaggi di un modello generativo*

Un modello generativo necessita di memorizzare la probabilità congiunta, e lo spazio di memoria richiesto per questa attività aumenta esponenzialmente nel numero di feature: questo porta molto spesso a semplificare il modello effettuando determinate ipotesi, come assumere che le feature siano indipendenti.

È inoltre possibile che l'accuratezza nel modellare $P(X)$ non sia coerente con quella della modellazione di $P(Y|X)$, quindi ci potrebbe essere qualche zona dello spazio delle feature in cui la probabilità non è stata modellata correttamente. Tuttavia potrebbe non essere così grave: supponiamo che in una zona $P(X)$ è bassa, un modello di $P(Y|X)$ assumerà una minore importanza (e quindi l'occorrenza di una classificazione non corretta di queste istanza sarà rara).

L'apprendimento può essere visto come un processo continuo di riduzione dell'incertezza ad ogni iterazione:

1. per prima cosa si assume una distribuzione a priori uniforme sulle classi (derivante da una conoscenza del dominio e nulla di più);
2. dopo aver osservato la classificazione delle istanze dei primi esempi (e quindi delle sue feature X), si può adottare una distribuzione di classi a posteriori $P(Y|X)$: qualora quest'ultima sia meno uniforme rispetto a quella a priori siamo inclini ad accettare la classe più probabile. Abbiamo così ridotto l'incertezza sulla classe osservando l'istanza della feature X ;
3. questa probabilità a posteriori può essere usata in uno step successivo come se fosse la "nuova" probabilità a priori;
4. si ripete il processo fino a quando non si ottengono più nuove informazioni.

È proprio questo processo il cardine dei modelli di tipo Bayesiano: si parte da una conoscenza derivante dall'esperienza del progettista (i Belief) e ad ogni nuovo esempio si calcola (tramite teorema di Bayes) la probabilità a posteriori incorporando nel nuovo modello la nuova informazione e utilizzando poi questa nuova probabilità a posteriori come probabilità a priori per gli esempi successivi che si vogliono inglobare.

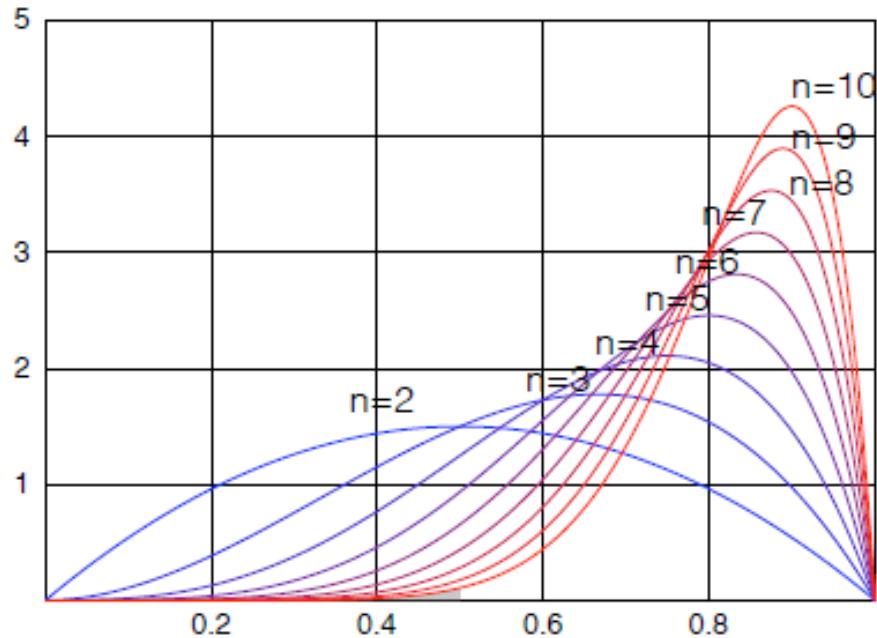


Figura 9.1: La distribuzione β ha la forma $f()$, dove $\beta - 1$ è il numero di fallimenti e $\alpha - 1$ è il numero di successi negli esempi osservati.

9.2.2 Riduzione dell'incertezza nella stima: distribuzione β

Supponiamo di voler stimare la probabilità Θ che una email arbitraria sia spam, quindi occorre utilizzare una distribuzione a priori adeguata.

La cosa più naturale da fare è quella di ispezionare n e-mails, determinare il numero di email che sono spam (d) e dire che $\hat{\Theta} = \frac{d}{n}$. Anche se la stima di Θ è la migliore, ovvero è la c.d. *MAP, Maximum A Posteriori*, non significa che altri valori di Θ siano completamente sballati: concentrarsi solo sul valore MAP pur sapendo che i dati sono leggermente rumorosi e senza ammettere nessun margine di errore potrebbe risultare troppo riduttivo.

Per ovviare questo problema è possibile modellarlo tramite una distribuzione di probabilità incentrata su Θ (in questo caso una Beta-distribution) che è aggiornata ogni volta che si ottengono nuove informazioni (si vede l'avanzare dell'aggiornamento nella Figura 9.1 in cui per ogni curva il bias sullo spam è dato dall'area sotto la curva e a destra di $\Theta = \frac{1}{2}$).

Per costruire questa curva si può riprendere la distribuzione di Bernoulli: se d sono gli esempi che hanno successo per una classe, $n-d$ sono gli esempi appartenenti all'altra classe.

Nella funzione β si lascia semplicemente libero il valore della probabilità Θ (non è più una costante che assume valore MAP) e lo si fa variare sull'asse x .

Nel grafico la probabilità di spam è data dalla coordinata x , mentre il

valore più probabile è il massimo della curva.

È possibile vedere come si modifica la curva al variare del numero n degli esempi considerati.

Ogni volta che si ispeziona una e-mail, si introduce l'incertezza riguardo la probabilità a priori che sia spam. Dopo aver osservato due e-mail di cui una è spam, il valore possibile di Θ è caratterizzato da una ovvia distribuzione simmetrica incentrata sul valore $\frac{1}{2}$. Se, dopo averne ispezionate altre, diciamo dieci, tutte risulteranno essere spam tranne la prima, questa distribuzione si sposterà inevitabilmente verso destra, ottenendo la conformazione in Figura 9.1: la varianza della curva rappresenta l'incertezza e la distribuzione per n e-mail raggiunge il massimo quando $\hat{\Theta}_{MAP} = \frac{n-1}{n}$ (ad esempio per n = 5, $\hat{\Theta}_{MAP} = 0.8$).

9.3 LA PROSPETTIVA BAYESIANA

Questo quadro suggerisce come lavorano gli statistici Bayesiani, ovvero facendo delle inferenze sulle probabilità stesse e modellandole con delle curve che ammettono una certa incertezza attorno alle stime fatte sulle probabilità.

Pensadoci, anche con gli intervalli di confidenza potevano fare delle inferenze su quanto era probabile che una certa ipotesi nulla fosse vera tramite il p-value. È possibile ottenere un risultato simile con quanto appena visto: cioè è possibile da un lato usare lo spread di questa curva per avere l'incertezza riguardo alle probabilità e poi usarla per generare degli esempi in base a quella che è la nostra credenza per quel dominio (come abbiamo visto fare per i modelli generativi).

È inoltre possibile rifiutare certe ulteriori inferenze: si supponga di avere 9 spam e 1 ham, quindi la probabilità di ham è bassissima, la curva è molto spostata verso valori alti della probabilità e l'area sotto la curva in un intervallo da 0 a 0.5 è molto piccolo. Il valore di probabilità per cui una email sia ham è data integrando la curva sotto la parte ham.

Per concludere questa parte introduttiva sulla statistica Bayesiana diciamo che lo statista, con i suoi belief, può decidere come modellarli scegliendo la distribuzione di probabilità che corrisponde alla nostra credenza.

Se per esempio è convinto che avrà una distribuzione 50-50 occorre scegliere la funzione beta corrispondente. Ecco che possiamo usare questi modelli a qualsiasi oggetto, persino ai modelli stessi (poiché al variare del training set possiamo generare diversi modelli).

Se si conoscessero la probabilità degli esempi e della classe (a priori), si potrebbe usarle assieme e stimare la classe corretta dato l'esempio, che è proprio quella che massimizza la probabilità di ottenere la classe giusta. Un classificatore che assegna questo valore soddisfa l'ottimalità di Bayes.

9.4 L'OTTIMALITÀ DI BAYES

Si introduce ora un principio teorico che confronta le predizioni che farebbe un classificatore basato sul teorema di Bayes per predire la classe (si assume conosciuta la probabilità della classe osservati i vari esempi). Un classificatore che si basa su questi principi soddisfa la proprietà di *ottimalità di Bayes*. Più formalmente, un classificatore è *Bayes-optimal* se assegna sempre $\operatorname{argmax}_y P^*(Y = y | X = x)$ a un'istanza x , dove P^* è la distribuzione reale a posteriori.

Questa nozione può anche essere utilizzata come criterio più generale: ipotizzando una certa probabilità, diciamo Gaussiana, si può generare artificialmente un dataset che soddisfa quella distribuzione (per costruzione lo farà, perché è generato basandosi su di essa) e poi si può andare a verificare se un modello di classificazione addestrato con quel dataset soddisfa le predizioni della classe (cioè se soddisfa l'ottimalità di Bayes e quindi il modello collima con quello bayes-ottimo).

Ecco che questa proprietà può essere usata per i modelli probabilistici come base di confronto che restituisce il valore ottimale a cui dovrebbe tendere un qualunque modello probabilistico.

9.5 MODEL SELECTION

Vedremo ora il ragionamento Bayesiano applicato alla selezione del modello: fino ad ora è stato detto che si possono produrre diversi modelli a partire dallo stesso dataset (uno basato su alberi, un altro su SVM e così via) e poi confrontarli a livello di accuratezze e selezionare quello che si comporta meglio. Si vuole ora selezionare il modello sul principio dell'ottimalità di Bayes e vedere se questo ci darà la soluzione ottima. In generale la soluzione di un singolo modello, secondo il ragionamento bayesiano, non è la soluzione migliore in quanto la cosa migliore da fare sarebbe avere una collezione di modelli e la stima da farsi su di essi è data dalla media pesata delle loro predizione con la probabilità che il modello sia corretto per quell'esempio (è ciò che fanno i modelli ensemble).

Si seguirà ora l'applicazione delle leggi di probabilità condizionata di bayes-ottimo che è quella che si vorrebbe avere per un classificatore che rispetta questa proprietà.

Facendo l'opposto delle operazioni di marginalizzazione, si può specializzare ulteriormente la probabilità di Y , sapendo di aver osservato X , per tutti i possibili modelli M .

Sia M l'insieme di tutti i possibili modelli applicabili m per fare la stima, si considerano (e si sommano) tutte le probabilità di stima delle classi per ogni modello m sullo stesso esempio x .

$$P(Y | X = x) = \sum_{m \in M} P(Y, M = m | X = x)$$

Si utilizza poi la chain rule per mettere alla parte di condizionamento ciò che stava nella parte principale della priorità, cioè:

$$P(Y | X = x) = \sum_{m \in M} P(Y | M = m, X = x)P(M = m | X = x)$$

E infine, siccome il modello e l'esempio sono variabili casuali indipendenti (l'esempio è del training e il modello è scelto ed è indipendente rispetto all'esempio da testare) si ottiene

$$P(Y | X = x) = \sum_{m \in M} P(Y | M = m, X = x)P(M = m)$$

A questo punto, confrontando quest'ultima formula con quella da cui siamo partiti notiamo che se anche si scegliesse un certo modello m^* che è quello che nel complesso si è comportato meglio su tutti gli esempi del dataset ($\text{argmax}_y P(Y = y | M = m^*, X = x)$) non è detto che il criterio bayes-ottimo sia rispettato (non andiamo a prendere tutti i modelli prendendo le loro probabilità e pesandole sulle probabilità di applicare quello specifico modello). Si vede infatti che le due formule non sono uguali (in realtà basterebbe avere il massimo sullo stesso valore della classe y stimata uguale e non è necessario che le due formule siano effettivamente uguali ma è sufficiente che conducano alla stessa scelta della classe).

9.6 LA DISTRIBUZIONE NORMALE

9.6.1 Gli obiettivi

Si considera ora un caso teorico in cui è presente un dataset distribuito secondo funzioni gaussiane e in cui si vuole riconoscere i casi in cui il classificatore di base lineare (semplice, facile da calcolare poiché si basa sul valore dei centroidi nelle due classi e costruisce un boundary lineare a metà tra le classi) segue il principio bayes-ottimo e, andando a vedere la distribuzione delle classi, si effettueranno delle interpretazioni di tipo geometrico.

Si vuole poi introdurre il principio di *maximum likelihood* (ottenuta rendendo massima la likelihood su ciascuno degli esempi del dataset), in parte legato col principio bayes-ottimo poiché trova anch'esso un valore di massimo ma non sulla probabilità condizionata a posteriori, bensì sulla probabilità a priori (likelihood, la probabilità dell'esempio x sapendo la classe y). Questo permette di individuare i parametri incogniti della legge di distribuzione di probabilità che hanno generato il dataset stesso (tale legge non è conosciuta ma è inferibile dai dati). Si applicherà poi il principio della maximum likelihood al problema della regressione lineare.

9.6.2 Una distribuzione normale multivariata

Si darà ora un insieme di formule (non tutte utili al mero scopo del superamento dell'esame) utili a definire la distribuzione normale multivariata, ovvero operante in uno spazio delle esempi con d feature dove ogni esempio è rappresentato da un vettore di d elementi. Cominciamo col definire semplicemente la formula della distribuzione normale univariata (il vettore che descrive l'esempio è in realtà in questo caso composto da una sola entry):

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

dove ovviamente $E = \sqrt{2\pi}\sigma$.

Se vogliamo generalizzare nel caso multivariato non avremo più il valore della deviazione standard (che ricordiamo identifica l'ampiezza della campana attorno al centroide) bensì la matrice di covarianza (contenenti le varianze di tutte le feature sulla diagonale e le coovarianze di coppie di feature simmetricamente nelle due parti della diagonale) e invece di avere lo scarto quadratico ad esponente avremo una certa moltiplicazione che vediamo nella formula e in cui riconosciamo la distanza di Mahalonobis:

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

dove $E_d = (2\pi)^{\frac{d}{2}} \sqrt{|\Sigma|}$, d è il numero di entries del vettore che descrive il singolo esempio, μ è il vettore target e Σ^{-1} è l'inverso della matrice di covarianza e $|\Sigma|$ è il suo determinante.

Ci troviamo in questo caso in un cosiddetto *mixture model*, cioè un modello fatto con due funzioni gaussiane, una per ciascuna classe. Nel caso univariato, potremmo ritrovarci con una rappresentazione simile a quella in figura 9.2.

Nel caso multivariato possiamo invece avere una rappresentazione come quella in figura 9.3 dove a sinistra abbiamo un caso semplice in cui si vede subito che il classificatore lineare è esattamente ortogonale alla bisettrice e si comporta in modo bayes-ottimo poiché separa perfettamente gli esempi delle due aree ed effettua la scelta più probabile secondo un classificatore che rispetta la proprietà e che conosce le due gaussiane (dove sono centrate e le loro covarianze). Tale soglia creata dal classificatore è esattamente il luogo dei punti in cui le due gaussiane si intersecano (ovvero le due probabilità sulle due classi hanno uguale valore). Come detto il primo caso è molto semplificativo e infatti abbiamo che le due deviazioni standard delle due distribuzioni delle due classi si equivalgono, il decision boundary si pone esattamente a metà della distanza tra i due centroidi ma le feature non sono fra loro correlate (infatti si distribuiscono lungo tutte le direzioni). Nel caso centrale invece esiste una certa correlazione tra

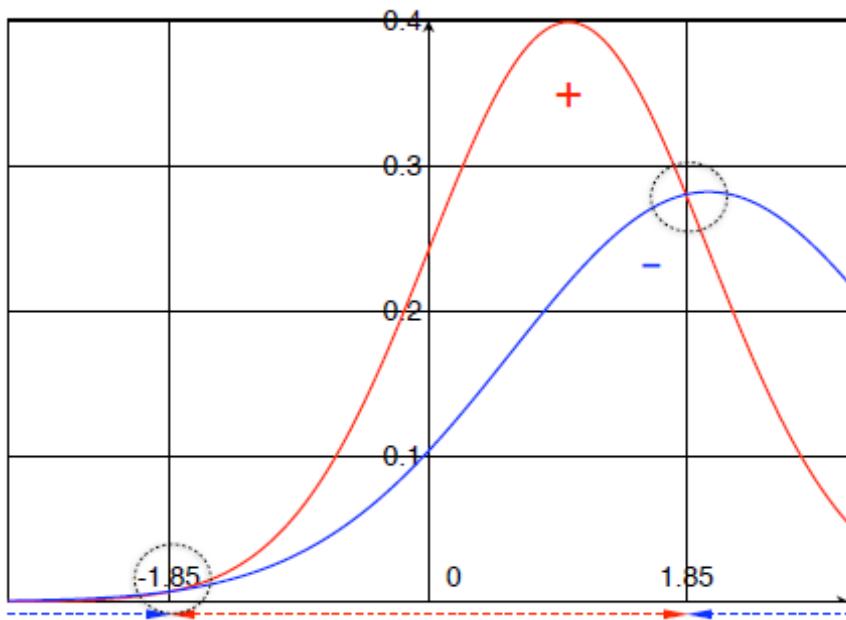


Figura 9.2: Rappresentazione del Mixture Model nel caso univariato

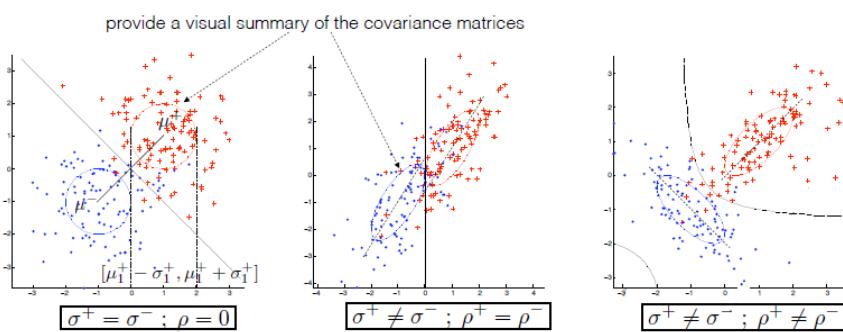


Figura 9.3: Rappresentazione del Mixture Model nel caso bivariato

la prima e a seconda feature e infatti i punti si distribuiscono secondo un asse principale di due elissoidi: in questo caso il classificatore di base lineare non è più bayes-ottimo e quindi il modello discriminativo secondo cui le due probabilità di classe si equivalgono rimane sempre lineare ma non coincide con il classificatore di base lineare. Il terzo caso è quello più generale in cui si hanno due decision boundary.

9.7 LIKELIHOOD RATIO

A inizio Capiolo si era partiti trattando i modelli discriminativi (ovvero modelli basati su funzione continua che opera nello spazio delle feature) e tramite differenze tra i valori da essi predetti riuscivamo a capire a quale classe appartiene l'esempio.

In realtà si applicavano la differenze tra due funzioni discriminative (una per ogni classe) per avere una sorta di confronto relazionale. Si avrà quindi

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

e

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

Nell'esempio che vedremo ora faremo invece un rapporto fra le due likelihood, utilizzandole come funzioni indicatrici della classe, ottenendo così il *likelihood ratio* definito come

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

Questo indice ha una relazione diretta con la probabilità di appartenenza di un esempio ad una classe:

- Se per un esempio x questo rapporto ha un valore > 1 allora $P(x | \oplus) > P(x | \ominus)$ e ciò significa che la classe positiva è più probabile rispetto a quella negativa (decideremo quindi per la prima).
- Viceversa, se il rapporto è < 1 ... (confidiamo nel vostro intelletto Signori Lettori ;-)
- Se il rapporto è $= 1$ siamo nel punto di definizione del decision boundary tra le due classi (equiprobabili) e non potremo decidere.

Il caso più interessante per definire il decision boundary è ovviamente quello in cui il likelihood ratio è uguale a 1. Ma quando si verifica questo caso? Matematicamente parlando, la risposta intuitiva riguardando la formula, è "quando l'argomento della funzione

esponenziale è uguale a 0". Possiamo distinguere tre casi, via via più generali.

- Come primo caso supponiamo di avere $\sigma^{\oplus} = \sigma^{\ominus} = \sigma$, cioè di avere due campane di ampiezza σ uguale ma distribuite in modo diverso secondo i loro (diversi) centroidi μ^{\oplus} e μ^{\ominus} . Possiamo allora semplificare la parte di esponente della formula $LR(x)$ come segue:

$$\begin{aligned} & -\frac{1}{2} \left[\left(\frac{x - \mu^{\oplus}}{\sigma^{\oplus}} \right)^2 - \left(\frac{x - \mu^{\ominus}}{\sigma^{\ominus}} \right)^2 \right] = \\ & = -\frac{1}{2\sigma^2} \left[(x - \mu^{\oplus})^2 - (x - \mu^{\ominus})^2 \right] = \dots = \frac{\mu^{\oplus} - \mu^{\ominus}}{\sigma^2} \left[x - \frac{\mu^{\oplus} + \mu^{\ominus}}{2} \right] = \\ & = \frac{\mu^{\oplus} - \mu^{\ominus}}{\sigma^2} \left[x - \frac{\mu^{\oplus} + \mu^{\ominus}}{2} \right] \end{aligned}$$

Vediamo allora che avremo una differenza tra x e una quantità μ valor medio tra i due centroidi, moltiplicati per un valore γ che è la differenza dei due centroidi normalizzata rispetto al quadrato della deviazione standard. Diremo allora che il luogo di massima likelihood (cioè quel luogo di punti x in cui il likelihood ratio è uguale a 1) è il punto $x_{ML} = \mu$, ed è quindi graficamente rappresentata da una retta verticale e coincide con il classificatore lineare di base (che ricordiamo è semplice da ottenere, perché basta tracciare la retta ortogonale al segmento che congiunge i centroidi delle due classi).

- Nel secondo caso generalizziamo, e supponiamo che non è più vera il vincolo per cui $\sigma^{\oplus} \neq \sigma^{\ominus}$. Introduciamo un esempio nel caso di distribuzione normale univariata. Abbiamo $\sigma^{\oplus} = 1$ e $\sigma^{\ominus} = 2$, e quindi notiamo che $\sigma^{\ominus} = 2\sigma^{\oplus}$ e dunque $LR(x) = 2\exp \left(-[(x-1)^2 - (x-2)^2/4]/2 \right) = 2\exp(3x^2/8)$. Sviluppato si ottiene che $LR(x) = 1 \iff x = \pm(8/3)\ln 2 = \pm 1.85$, come visibile nel grafico in figura 9.2. Questo ci suggerisce che il decision boundary non è più semplice come nel primo caso, ma sarà dato da due aree separate (la regione della classe negativa è separata a metà da un'area centrale che è l'area positiva, quest'ultima identificabile poiché compresa fra le due radici).
- Come terzo caso, riportiamo quanto assunto nei casi precedenti nel caso bivariato. Avremo ovviamente centroidi diversi per le due feature di x ma noi assumiamo $\mu_1^{\oplus} = \mu_2^{\oplus} = 1$ e $\mu_1^{\ominus} = \mu_2^{\ominus} = -1$. La formula della Gaussiana nel caso bivariato definisce le correlazioni tra le feature degli esempi ed è la seguente:

$$P(x_1, x_2 | \mu_1, \mu_2, \sigma_1, \sigma_2, \rho) = \frac{1}{E_2} \exp \left(-\frac{1}{2(1-\rho^2)} (z_1^2 + z_2^2 - 2\rho z_1 z_2) \right)$$

con

$$E_2 = 2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}$$

con z il numero di deviazione standard fra x e la media

$$z_i = \frac{x_i - \mu_i}{\sigma_i}$$

dove ρ è in pratica una normalizzazione rispetto alle deviazioni standard delle due feature in oggetto. Vediamo i sottocasi:

- Nel primo, assumiamo $\sigma^{\oplus} = \sigma^{\ominus} = 1$ (ovvero deviazioni standard uguali per le due classi) e assumiamo anche che le correlazioni tra le due feature di ciascun esempio sia uguale a $\rho = 0$ (se non vi è correlazione tra le due feature gli esempi si disporranno ad uguale distanza dal centroide pari alla deviazione standard, altrimenti assumeranno una forma ad elissoide). In questo caso il luogo di punti in cui si ha massima likelihood è dato da:

$$(x_1 - 1)^2 + (x_2 - 1)^2 - (x_1 + 1)^2 - (x_2 + 1)^2 = -2x_1 - 2x_2 - 2x_1 - 2x_2 = 0 \leftrightarrow x_1 + x_2 = 0$$

Ne vediamo il significato rappresentato in figura 9.3 in cui si nota il che il decision boundary è una retta ortogonale al segmento che collega i due centroidi e coincide quindi con il classificatore di base lineare

- Nel secondo caso rilassiamo l'ipotesi che le due deviazioni standard siano uguali, mantenedo però il vincolo di uguale correlazione tra le due feature. In particolare assumeremo un caso specifico in cui $\sigma_1^{\oplus} = \sigma_1^{\ominus} = 1$, $\sigma_2^{\oplus} = \sigma_2^{\ominus} = \sqrt{2}$ ed infine ancora $\rho^{\oplus} = \rho^{\ominus} = \sqrt{2}/2$ (l'asse portante dell'elissoide è inclinato a 45°). Allora il decision boundary sarà dato da (x) Vediamo dalla figura 9.3 che il decision boundary sarà sempre lineare, ma non coinciderà più con il classificatore di base poiché la rettă non è più ortogoneale alla retta che congiunge i due centroide.
- Nel caso più generale avremo anche il parametro ρ diverso per le due classi e questo ci porterebbe ad avere un decision boundary che non è un unico luogo di punti bensì un'area divisa in mezzo dall'area dell'altra classe (il decision boundary diventerebbe iperbolico, come si vede nella figura 9.3 in cui si assume $\rho = 0.7$). In particolare, si assume come esempio $\rho^{\oplus} = -\rho^{\ominus} = \rho$, allora il decision boundary sarà dato da

$$\begin{aligned} & (x_1 - 1)^2 + (x_2 - 1)^2 - 2\rho(x_1 - 1)(x_2 - 1) - (x_1 + 1)^2 + \\ & - (x_2 + 1)^2 - 2\rho(x_1 + 1)(x_2 + 1) = -4x_1 - 4x_2 - 4\rho x_1 x_2 - 4\rho = 0 \\ & \leftrightarrow \\ & x_1 + x_2 + \rho x_1 x_2 + \rho = 0 \end{aligned}$$

9.8 CONCLUSIONI

Per completezza riportiamo una generalizzazione della Gaussiana nel caso d-variato (in cui gli esempi sono espressi nella forma $x = (x_1, \dots, x_d)^T \in \mathbb{R}^d$):

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

con

$$E_d = (2\pi)^d / 2\sqrt{|\Sigma|} \text{ e } \mu = (\mu_1, \dots, \mu_d)^T$$

Come detto, stiamo cercando in questo capitolo di trovare dei collegamenti tra l'interpretazione probabilistica e l'interpretazione geometrica e, sofferemandoci sulla forma di questa Gaussiana, riconosciamo nella formula la *distanza di Mahalanobis* che ricordiamo essere una possibile generalizzazione della distanza euclidea che tiene però conto, valutando la distanza tra il punto e il centroide, della varianza con il quale è distribuito tutto il dataset.

In particolare, in questa versione della Gaussiana si usa la covarianza per normalizzare la distanza tra i punti ed il centroide (ponendo tale quantità a denominatore).

Quando il classificatore di base coincide con il classificatore di Bayes-Ottimo per il caso (più generale in assoluto) in cui abbiamo una matrice di covarianza Σ ? Quando, ritornando al likelihood ratio nel caso più generale, ovvero

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

Assumendo però che le matrici di covarianza per le due classi siano uguali ed unitarie (cioè che le feature di entrambe le classi non abbiano correlazione fra loro), avremo nuovamente che il likelihood ratio è uguale all'esponenziale di una quantità pari alla differenza dello scarto al quadrato tra il vettore dell'esempio x e il centroide della classe positiva e lo stesso scarto considerando la classe negativa:

$$\begin{aligned} LR(x) &= \exp\left(-\frac{1}{2} [(x - \mu^\oplus)^T (x - \mu^\oplus) - (x - \mu^\ominus)^T (x - \mu^\ominus)]\right) = \\ &= \exp\left(-\frac{1}{2} [|x - \mu^\oplus|^2 - |x - \mu^\ominus|^2]\right) \end{aligned}$$

Ricordando che l'interesse è quello di identificare il luogo dei punti che definiscono il decision boundary, saremo interessati ad ottenere $LR(x) = 1$, e cioè il suo esponente uguale a 0. Assumendo matrici di covarianza entrambi uguali alla matrice identità saremo nel caso appena descritto per ogni x equidistante dai due centroidi μ^\oplus e μ^\ominus . Siamo nuovamente ricaduti nel primo caso, in cui il classificatore di base è bayes-ottimo quando le due classi hanno la stessa deviazione

standard e non hanno alcuna correlazione.

In generale, assumendo di sapere solamente che le due matrici di covarianza sono uguali, il decision boundary è ancora lineare e intersecherà ancora il segmento $\mu^\oplus - \mu^\ominus$ nel mezzo, ma non sarà ad esso ortogonale. Per verificare che sia esattamente il classificatore Bayes-Ottimo occorre decorrelare le feature e normalizzarle (moltiplicando la matrice dei dati X per l'inverso della matrice di covarianza, ovvero da $X(X^T X)^{-1}$).

Generalizzando ancora e non avendo matrici di covarianza uguali, il decision boundary sarà iperbolico.

Possiamo quindi dire che le tre immagini viste prima (che riunifichiamo nella figura 9.3) generalizzano le tre casistiche per le distribuzioni multivariate.

9.9 DALLA DISTANZA ALLA PROBABILITÀ

Arriviamo finalmente al legame tra l'interpretazione geometrica e quella probabilistica, riconoscendo che esiste un legame tra la distanza dei punti adottando una certa metrica e una certa covarianza dei dati Σ e la distribuzione: abbiamo già visto la distanza di Mahalanobis è messa ad esponente nella formula della Gaussiana.

Spesso, in statistica, la likelihood viene usata come argomento di un logaritmo (negato, perché esiste una costante negata che moltiplica la distanza) uguagliato a zero, proprio perché ci si concentra sul solo termine a potenza. Questo vedremo come semplificherà parecchio le nostre vite quando ci imbatteremo (a breve) nella considerazione della likelihood dell'intero dataset.

9.10 STIMA DELLA MAXIMUM-LIKELIHOOD

Supponiamo ora di voler stimare i parametri (μ e Σ) di una distribuzione Gaussiana multivariata. Il criterio di maximum-likelihood ci permette di partire dai dati e da questi stimare quale può essere il valore dei parametri che rende massima la verosimiglianza. Usaremos, come anticipato nella sezione precedente, il valore del logaritmo negato con la distanza ad argomento. Vediamo come avviene (matematicamente) questa stima dei parametri incogniti della distribuzione degli esempi. Supponiamo di voler stimare il parametro μ . Assumiamo che gli esempi del dataset X sono indipendenti, e di conoscere la matrice di covarianza Σ (calcolata a partire dal dataset stesso X). Avendo l'informazione circa l'indipendenza degli esempi del dataset, la likelihood congiunta (dell'intero dataset) può essere scritta come prodotto su ogni singolo $x \in X$ e la stima della massima likelihood può essere trovata come segue:

$$\hat{\mu} = \arg \max_{\mu} \prod_{x \in X} P(x | \mu, \Sigma) =$$

volendo però trovare il valore massimo di μ che rende massima la verosomiglianza dell'intero dataset. Per farlo, seguiamo i seguenti passi algebrici.

$$\begin{aligned} \hat{\mu} &= \arg \max_{\mu} \prod_{x \in X} P(x | \mu, \Sigma) = \\ &= \arg \max_{\mu} \prod_{x \in X} \frac{1}{E_d} \exp \left(-\frac{1}{2} (\text{Dis}_M(x, \mu | \Sigma))^2 \right) = \\ &= \arg \max_{\mu} \sum_{x \in X} \left[\ln E_d + \frac{1}{2} (\text{Dis}_M(x, \mu | \Sigma))^2 \right] = \text{rimuovendo i termini costanti} \\ &= \arg \max_{\mu} \sum_{x \in X} (\text{Dis}_M(x, \mu | \Sigma))^2 \end{aligned}$$

Abbiamo in pratica generalizzato il teorema visto per gli algoritmi di clustering dove il centroide era proprio il punto che rendeva minime le distanze quadratiche tra tutti i punti del dataset e se stesso. In generale il criterio che permette di partire dai dati per ottenere stime sui parametri incogniti è uno dei Sacri Graal del machine learning e si vedrà che verrà adottato anche in altre occasioni.

9.11 MINIMIZZARE LO SCARTO QUADRATICO PER FARE REGRESSIONE LINEARE

Continuiamo ora ad usare il principio di maximum likelihood applicandolo ora alla minimizzazione degli scarti quadratici per la risoluzione dei problemi di regressione lineare.

Abbiamo già visto una soluzione di tipo geometrico, mentre adotteremo ora un criterio di massima somiglianza.

In questa applicazione avremo n esempi x_i e si vuole effettuare una regressione tramite una funzione che restituisce il valore y_i .

La regressione guarda tutto il dataset e calcola il luogo dei punti che rende minimo lo scarto al quadrato tra ogni singola osservazione dei valori delle w e il valore calcolato della funzione di regressione sopra introdotta.

Trattandosi di una generalizzazione su tutto il dataset, potrebbero verificarsi dei piccoli errori ϵ_i per qualche esempio i . Se supponiamo che questi errori si distribuiscono in modo Gaussiano con una media nulla e una certa varianza σ (incognito) possiamo calcolare i parametri (coefficiente angolare e termine noto) della regressione.

Essendo gli errori ϵ_i e gli y_i indipendenti la loro probabilità congiunta è data dal prodotto delle n Gaussiane degli errori:

$$P(y_1, \dots, y_n | a, b\sigma) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left(\frac{\epsilon_i^2}{2\sigma^2} \right), \text{ dove } \epsilon_i = y_i - (a + bx_i)$$

9.11.1 Un esempio pratico

Vediamo subito un esempio di applicazione in un caso univariato, in cui abbiamo supponiamo di voler investigare sulla relazione tra l'altezza delle persone e il loro peso: avremo n misurazioni della forma $(h_i, w_i), 1 \leq i \leq n$.

La funzione di regressione assume la funzione lineare $w = a + bh$. e si vorranno scegliere i parametri a e b in modo che la somma degli scarti quadratici $\sum_{i=1}^n (w_i - (a + bh_i))^2$ sia minima.

Per rendere matematicamente minima tale quantità si considera la derivata parziale dello scarto quadratico rispetto all'incognita, la si egualia a 0 e si risolve tale equazione per trovare l'incognita stessa (nel nostro caso gli a e b desiderati). Vediamo prima la derivata parziale rispetto ad a che ci aiuterà a trovare il termine noto della funzione di regressione lineare:

$$\frac{\partial}{\partial a} \sum_{i=1}^n (w_i - (a + bh_i))^2 = -2 \sum_{i=1}^n (w_i - (a + bh_i)) = 0 \Rightarrow \hat{a} = \bar{w} - \hat{b}\bar{h}$$

dove \bar{w} è il valore medio di tutti i w_i , così come \bar{h} lo è per gli h_i . E vediamo anche la derivata parziale rispetto b , utilizzata per stimare il coefficiente angolare:

$$\frac{\partial}{\partial b} \sum_{i=1}^n (w_i - (a + bh_i))^2 = -2 \sum_{i=1}^n (w_i - (a + bh_i))h_i = 0 \Rightarrow \hat{b} = \frac{\sum_{i=1}^n (h_i - \bar{h})(w_i - \bar{w})}{\sum_{i=1}^n (h_i - \bar{h})^2} =$$

Quindi la soluzione della regressione lineare che minimizza lo scarto quadratico (e rende massima la verosimiglianza) è

$$w = \hat{a} + \hat{b}h = \bar{w} + \hat{b}(h - \bar{h})$$

9.12 DUE CRITERI COINCIDENTI

Si può notare che massimizzare la verosimiglianza e ridurre lo scarto quadratico (senza considerare la Gaussiana, bensì solamente ogni singola osservazione) sono criteri che coincidono ai nostri fini. Infatti, focalizzandoci sul fatto che stiamo interpretando delle likelihood (e volendo trovare il suo massimo sapendo di essere nel caso di una distribuzione Gaussiana ora assunta, a differenza di prima che era una conoscenza trascurabile) possiamo seguire i seguenti passaggi:

$$\begin{aligned} P(y_1, \dots, y_n | a, b, \sigma) &= \prod_{i=1}^n \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - (a + bx_i))^2}{2\sigma^2}\right) = \\ &= \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n \exp\left(-\frac{\sum_{i=1}^n (y_i - (a + bx_i))^2}{2\sigma^2}\right) \end{aligned}$$

E considerando il logaritmo per valutare l'esponente (come visto in precedenza):

$$-\ln(P(y_1, \dots, y_n | a, b, \sigma^2)) = \frac{n}{2} \ln(2\pi) + \frac{n}{2} \ln(\sigma^2) + \frac{\sum_{i=1}^n (y_i - (a + bx_i))^2}{2\sigma^2}$$

Occorrerà poi anche qui calcolare le derivate parziale di questa quantità rispetto ad a , b e a σ^2 per ottenere i rispettivi valori che massimizzino la likelihood:

$$\sum_{i=1}^n (y_i - (a + bx_i)) = 0 \Rightarrow \hat{a} = \bar{y} - \hat{b}\bar{x}$$

$$\sum_{i=1}^n (y_i - (a + bx_i))x_i = 0 \Rightarrow \hat{b} = \frac{\sigma_{xy}}{\sigma^2}$$

$$\frac{n}{2} \frac{1}{\sigma^2} - \frac{\sum_{i=1}^n (y_i - (a + bx_i))^2}{2(\sigma^2)^2} = 0 \Rightarrow \text{la somma degli scarti quadratici è uguale a } n\sigma^2$$

Siamo arrivati (come era sperato) agli stessi parametri a e b del calcolo precedente, mentre la terza equazione ci permette di controllare la varianza dell'errore (in questo caso $n\sigma^2$ e infatti abbiamo n osservazioni indipendenti fra loro e quindi la varianza complessiva è il prodotto delle singole varianze).

9.13 MODELLI PROBABILISTICI PER DATI NON NUMERICHE

Vogliamo ora continuare con le possibili applicazioni del teorema di Bayes per esempi con feature non numeriche ma di tipo categorico (ad esempio nominale, come può essere il colore degli occhi, con cui non è possibile ordinare gli esempi in modo significativo).

Esempio classico che verrà presentato sarà quello dei testi, dove le parole compaiono.

Tratteremo queste feature categoriche in due modi diversi:

- indicando con una variabile booleana se una data parola è assente o meno nel documento (sottosezione 9.13.1)
- oppure come variabile continua che indica la frequenza di una parola in un documento (sottosezione 9.13.2).

Vediamo quindi alcuni casi differenti per trattare questa tipologia di dati.

9.13.1 Modello basato su distribuzione di Bernoulli

Questo primo caso applica un modello della distribuzione di Bernoulli in cui viene indicato per ciascuna feature categorica se è presente o

meno in un esempio.

Applicato all'esempio dei documenti, essendo le feature descrittive di ciascuno le parole, diremo se una parola è presente o meno nel documento, avendo tante feature quante sono le parole del linguaggio (diciamo k), ciascuna espressa con una variabile Bernoulliana dedicata.

In altre parole, ogni parola i -esima del vocabolario si comporta come una variabile aleatoria x_i che segue una distribuzione di Bernoulli governata da un parametro Θ_i che indica con che probabilità la parola è presente in un generico documento osservabile.

Dovendo gestire per ogni esempio un vettore di k feature diremo di trovarci in una *distribuzione di Bernoulli Multivariata*.

9.13.1.1 In ricordo della distribuzione di Bernoulli

Vogliamo brevemente ricordare che la distribuzione di Bernoulli modella la probabilità che un evento binario si verifichi. L'output di questo modello è il *successo* o il *fallimento* dell'evento, ed è governato dal parametro fissato Θ (che è il valore atteso della variabile aleatoria):

$$\begin{cases} \Theta = P(X = 1) \\ 1 - \Theta = P(X = 0) \end{cases}$$

Calcolando la varianza, data come il valore atteso dello scarto tra il valore x di ciascuna variabile e il suo valore atteso, avremo che questa è data da:

$$\begin{cases} E(X) = \Theta \\ E[(X - E(X))^2] = \Theta(1 - \Theta) \end{cases}$$

Generalizzando questo caso a n esperimenti (e non a una singola osservazione del comportamento della variabile in un caso) saremo nel caso di una *distribuzione binomiale*.

Volendo visualizzare il numero di successi sugli n esperimenti, supponiamo essere k , la probabilità di successo di quella variabile cambia seguendo k :

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

La varianza sarà semplicemente data da:

$$\begin{cases} E(X) = n\Theta \\ E[(X - E(X))^2] = n\Theta(1 - \Theta) \end{cases}$$

9.13.2 Modello basato su distribuzione Multinomiale

Nel secondo modello, vogliamo sfruttare l'informazione della frequenza di comparsa di una parola e non come un semplice evento aleato-

rio.

Il vettore che identifica l'esempio è ora un vettore di conteggi in cui l'entry i -esima indica quante volte la feature i -esima occorre.

Avremo in tal senso una distribuzione basato su una distribuzione multinomiale, così descrivibile:

$$P(X = (x_1, \dots, x_d)) = n! \frac{\Theta_1^{x_1}}{x_1} \dots \frac{\Theta_k^{x_k}}{x_k}, \text{ con } \sum_{i=1}^d x_i = n$$

dove Θ_i corrisponde alla probabilità di comparsa dell' i -esima parola e x_i corrisponde alla frequenza di questa parola in un documento di n parole.

Il termine x_i a denominatore serve a tenere conto di tutte le possibili disposizioni della parole e nel caso estremo in cui si hanno solamente 2 parole e $k = x_1$ ricadiamo nel caso della legge binomiale.

9.14 NAIVE BAYES

Applicheremo i due modelli visti nelle sottosezioni precedenti tramite il metodo "Naive Bayes" che applica il teorema di Bayes per predire la classe del documento nell'ipotesi che le feature siano indipendenti. Se pensiamo ai testi (documenti) questo rilassamento è più che verosimile, in quanto vista la mole di parole possibile è inverosimile disegnare la probabilità congiunta: sicuramente lo scrittore segue una logica nella stesura del testo e alcune parole rendono molto più probabile la presenza di altre (come ad esempio è per le parole "viagra" e "pillola") ma i classificatori funzionano molto bene anche se basati su questa tecnica.

Nel caso multivariato vedevamo già che questa assunzione valeva, in quanto ciascuna probabilità di apparizione Θ_i veniva elevata alla sua frequenza x_i senza che venissero tenute in considerazione le altre parole.

Questo vincolo è però importante poiché ci permette di ottenere la probabilità congiunta delle feature semplicemente moltiplicando le loro probabilità singole.

9.14.1 Il modello Naive Bayes formalmente

Vedremo ora di formalizzare quanto appena introdotto.

Assumiamo un dataset con esempi descritti da vettore di d feature ciascuno: $X = x_1, \dots, x_d$.

Ricordiamo che il teorema di Bayes permette di trovare il valore della classe Y che massimizza

$$P(Y | x_1, \dots, x_d) = \frac{P(x_1, \dots, x_d | Y)P(Y)}{P(x_1, \dots, x_d)} =$$

tralasciando il denominatore poiché fissato una volta scelta la classe

$$= P(x_1, \dots, x_d | Y)P(Y)$$

Vogliamo ora stimare correttamente $P(x_1, \dots, x_d | Y)$ (aka likelihood) a partire dai dati e assumendo l'ipotesi di indipendenza delle singole feature dell'esempio X (data la classe Y) possiamo dire che

$$P(x_1, \dots, x_d | Y) = P(x_1 | Y)P(x_2 | Y) \cdots P(x_d | Y)$$

Dove stimare i singoli $P(x_1 | Y), P(x_2 | Y), \dots, P(x_d | Y)$ è un problema più facile. Se ci pensiamo, una volta che abbiamo stimato bene queste probabilità possiamo anche generare dei testi che verosimilmente seguono la stessa legge di distribuzione e in tal senso le probabilità sono tanto più utili quanto più differiscono tra i vari esempi per le stesse classi (la parola viagra è utilissima per capire se un'email è spam, poiché la comparsa di viagra in un'email spam è molto probabile, mentre è molto poco probabile in una email ham). Per generare nuovi esempi si può ricorrere a criteri di tipo Bayesiano, come i tre sotto elencati:

MAXIMUM LIKELIHOOD (ML) In questo criterio stabiliamo che essendo le classi equiprobabili la probabilità a priori del teorema di Bayes non influisce e quindi andremo semplicemente a massimizzare la likelihood: $\operatorname{argmax}_y P(X = x | Y = y)$

MAXIMUM A POSTERIORI (MAP) Se non possiamo assumere equiprobabilità delle classi dovremo calcolare il MAP: $\operatorname{argmax}_y P(X = x | Y = y)P(Y = y)$

RECALIBRATED LIKELIHOOD questo caso rappresenta un'ulteriore generalizzazione in cui ci si concentra solamente sulla likelihood, ma pesata con dei pesi rappresentativi della priorità a priori di comparsa degli oggetti delle diverse classi: $\operatorname{argmax}_y w_y P(X = x | Y = y)$

9.14.2 Un esempio di applicazione

Applicheremo ora il modello multivariato ad un esempio concreto, in cui consideriamo un vocabolario contenente 3 parole {a,b,c}. Considereremo due classi rappresentative del caso di presenza (successo) o assenza della parola, con i seguenti parametri (per ogni parola):

$$\Theta^\oplus = (0.5, 0.67, 0.33)$$

$$\Theta^\ominus = (0.67, 0.33, 0.33)$$

Macroscopicamente vediamo che è la parola b a comportarsi in modo molto diverso per le due classi (e avevamo già accennato al fatto che questo aiuta molto ad effettuare la distinzione).

Applicando il principio di indipendenza delle parole, applichiamo la likelihood, ovvero valutiamo quanto è possibile che un oggetto x con tutte e 2 le parole $\{a,b\}$ (rappresentato quindi da un vettore di feature binario del tipo $x = [1, 1, 0]$) appartenga alla classe positiva e quanto invece è probabile appartenga a quella negativa:

$$P(x | \oplus) = 0.5 \cdot 0.67 \cdot (1 - 0.33) = 0.222$$

$$P(x | \ominus) = 0.67 \cdot 0.33 \cdot (1 - 0.33) = 0.148$$

Notiamo che la probabilità di appartenere alla classe positiva è maggiore e quindi (volendo noi applicare il teorema di Bayes-ottimo e propendendo al massimizzare la verosimiglianza) diremo che l'oggetto x appartiene a questa classe.

Si nota inoltre che anche l'assenza delle parole ha un impatto sul calcolo della verosimiglianza.

Volendo potremmo anche arrivare a conclusioni analoghe ma attraverso il calcolo del likelihood ratio (soprattutto poiché nel caso di due classi questo passaggio alternativo può risultare molto conveniente). Il likelihood ratio può essere calcolato come

$$\frac{P(x | \oplus)}{P(x | \ominus)} = \frac{0.5}{0.67} \cdot \frac{0.67}{0.33} \cdot \frac{1 - 0.33}{1 - 0.33} = 1.5 (> 1)$$

Nella valutazione di questo ratio otteniamo ovviamente la stessa conclusione ottenuta in precedenza, ma che ci permette di valutare anche quanto può variare la probabilità a priori delle varie parole permettendo di rimanere invariata la scelta della classe effettuata con una valutazione MAP. Infatti, in questa metodologia avremo:

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

Dove ricordiamo che il secondo rapporto è il prior odds, e per permettere la stessa scelta della classe dovrà essere maggiore di 2/3.

Questo è particolarmente importante essendo l'esempio rappresentativo di una classe di esempi vasta in cui i valori possono variare seppur, ci si aspetti, leggermente.

9.14.3 Un esempio multinomiale

Ci baseremo ora sullo stesso esempio presentato nella sottosezione [9.14.2](#) ma applicandolo ad un caso multinomiale. Supponiamo di avere le seguenti (nuove) probabilità di comparsa delle tre parole:

$$\Theta^{\oplus} = (0.3, 0.5, 0.2)$$

$$\Theta^{\ominus} = (0.6, 0.2, 0.2)$$

Dove questa volta è la parola "a" ad essere molto discriminativa (vedremo che questo impatterà molto).

Il documento considerato contiene 3 occorrenze della parola "a", 1 occorrenza della parola "b" e nessuna occorrenza della parola "c", ed è quindi descrivibile dal vettore $x = (3, 1, 0)$ con ovviamente $n = 4$.

Valutiamo le likelihood:

$$P(x | \oplus) = 4! \frac{0.3^3}{3!} \frac{0.5^1}{1!} \frac{0.2^0}{0!} = 0.054, \text{ si precisa che } \frac{0.2^0}{0!} = 1$$

$$P(x | \ominus) = 4! \frac{0.6^3}{3!} \frac{0.2^1}{1!} \frac{0.2^0}{0!} = 0.1728$$

Valutiamo anche il likelihood ratio:

$$\left(\frac{0.3}{0.6}\right)^3 \left(\frac{0.5}{0.2}\right)^1 \left(\frac{0.2}{0.2}\right)^0 = 5/16$$

Arriviamo dunque alla conclusione di scelta della classe negativa.

9.14.4 Un terzo esempio con probabilità ricalibrate

Se sappiamo che due parole come "viagra" e "pills" sono fortemente correlate l'assunzione della indipendenza porta ad una likelihood falsata.

Questo potrebbe essere un problema sanabile attraverso la ricalibrazione della likelihood, che, come visto, richiede di calcolare i pesi w_y con tecniche euristiche. Ricordiamo:

$$\text{RECALIBRATED LIKELIHOOD} = \text{argmax}_y w_y P(X = x | Y = y)$$

Consideriamo quindi un nuovo esempio in cui abbiamo documenti composti da 5 parole {a,b,c,d,e}. Dal punto di vista del numero degli esempi le due classi considerate sono equiprobabili, infatti abbiamo quattro esempi per ciascuna che sono

- $e_1: b\ d\ e\ b\ b\ d\ e$
- $e_2: b\ c\ e\ b\ b\ d\ d\ e\ c\ c$
- $e_3: a\ d\ a\ d\ e\ a\ e\ e$
- $e_4: b\ a\ d\ b\ e\ d\ a\ b$

per la classe positiva, e

- $e_5: a\ b\ a\ b\ a\ b\ a\ e\ d$
- $e_6: a\ c\ a\ c\ a\ c\ a\ e\ d$
- $e_7: e\ a\ e\ d\ a\ e\ a$
- $e_8: d\ e\ d\ e\ d$

Per prima cosa procediamo eliminando le stop words, che individuiamo in "d" e "e". Le parole rimanenti (<{a,b,c}) costituiscono il vocabolario.

Andremo ad applicare ora degli aggiustamenti sui pesi (che ricordano vagamente quelli fatti con la Laplace correction, in cui si aggiunge una occorrenza per ogni classe anche per evitare probabilità nulle) per cercare di rendere più robuste le stime sulla probabilità di comparsa delle parole.

Se siamo in un modello multinomiale, possiamo sommare per ciascuna classe la presenza delle parole su tutti gli esempi e otterremo i vettore 5, 9, 3 per la classe positiva e 11, 3, 3 per la classe negativa.

Al fine di stimare meglio i parametri della multinomiale aggiungiamo degli pseudo-conteggi che ci portano ad avere un numero totale di occorrenze del vocabolario di 20 parole per classe:

$$5 + 9 + 3 + 3 \text{pseudo - counts} = 20 \text{ per la classe positiva.}$$

$$11 + 3 + 3 + 3 \text{pseudo - counts} = 20 \text{ per la classe negativa.}$$

I valori di probabilità saranno quindi:

$$\hat{\Theta}^+ = (6/20, 10/20, 4/20) = (0.3, 0.5, 0.20)$$

$$\hat{\Theta}^- = (12/20, 4/20, 4/20) = (0.6, 0.2, 0.2)$$

Nel caso multivariato (in cui ci preoccupiamo solamente di stabilire se una parola è presente o meno in una classe ma senza preoccuparci della sua occorrenza) ci si muove in modo leggermente differente.

Anzitutto si forma il vettore che conta le occorrenze per ciascuna classe: saranno i vettori (2, 3, 1) per la classe positiva e (3, 1, 1) la quella negativa. Dopodiché si aggiungono due pseudo documenti, uno contenente tutte le parole ed uno che non ne contiene nessuna: si conteranno quindi $n + 2 = 4 + 2 = 6$ documenti totali e le probabilità per ciascuna parola in ciascuna classe saranno individuate dai vettori (anche qui si aggiunge un valore per parola):

$$\hat{\Theta}^+ = (3/6, 4/6, 2/6) = (0.5, 0.67, 0.33)$$

$$\hat{\Theta}^- = (4/6, 2/6, 2/6) = (0.67, 0.33, 0.33)$$

9.14.5 La stima della densità

Vediamo ancora come esiste una similitudine di quanto introdotto con le variabili categoriche (che hanno un comportamento discreto). In generale, usando Naive Bayes, occorre stimare probabilità di eventi discreti: se le feature sono continue per poter applicare questo modello occorre fare una operazione di discretizzazione.

Possiamo in particolare comportarci in tre modi differenti:

- Possiamo costruire degli istogrammi considerando intervalli di valori che ricopre la legge generativa degli oggetti (quella tratteggiata in figura 9.4a)
- Si può usare una likelihood che tiene in conto sia la frequenza dei dati che la probabilità generativa Gaussiana (la curva che si ottiene (in rosso) si posiziona a metà tra i due valori considerati). Questo metodo si base su un *kernel density estimator*. Se si usa una legge uniforme anziché una gaussiana (figura 9.4b) si nota come l'assunzione Gaussiana di questo metodo sbilancia molto la stima, tanto che il metodo dell'istogramma funziona meglio.
- *Solid bell curve* (in blu)

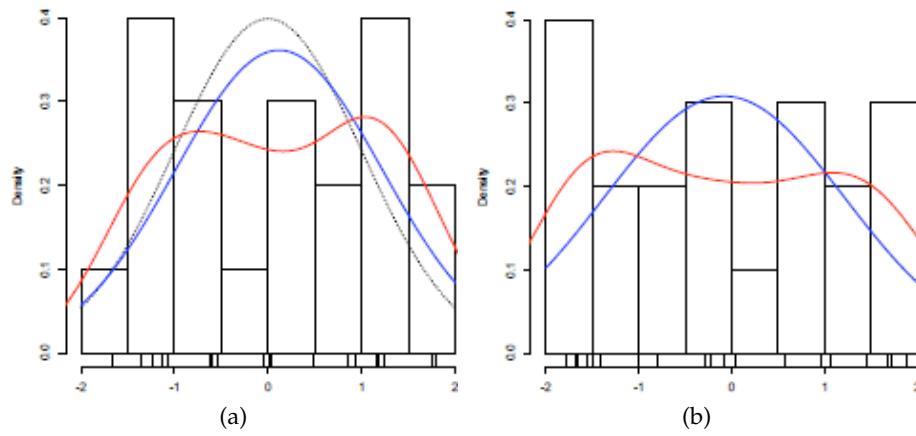


Figura 9.4: Varie possibilità di discretizzazione di legge a feature continue (tratteggiata)

Y_i	$P(Y_i X_i)$
1	$P(Y_i=1 x_i)=p$
0	$1-P(Y_i=1 x_i)=1-p$

Figura 9.5: Probabilità della classe modellata in modo dicotomico

9.15 LOGISTIC REGRESSION

Vedremo ora il modello discriminativo della logistic regression.

I modelli generativi visti finora (come Naive Bayes) hanno il pregio che permettono di poter essere usati per generare istanze verosimili rispetta alla legge che regola la distribuzione degli esempi e permettono inoltre di trovare il decision boundary tra due classi.

Vedremo invece ora un modello discriminativo che mira nell'immediato a trovare il decision boundary tra le due classi (nel caso di più classi che non vedremo occorrerà trattare tutte le possibili combinazioni di classi binarie).

alla fine di questa sezione faremo un confronto vedendo che i decision boundary trovati con un modello generativo e con un modello discriminativo sono diversi.

Il modello della regressione lineare combina in maniera lineare le feature descrittive degli esempi (x_i) utilizzando dei pesi (β_i).

Il modello della regressione lineare non può però essere applicato su valori non continui. Con la *logistic regression* vogliamo invece predire uno score dicotomico (nel caso in cui consideriamo due classi). Per farlo possiamo modellare la probabilità di trovarci in una classe con una regressione lineare.

In particolare, modelliamo la probabilità della classe con una variabile Bernoulliana Y_i come descritto in figura 9.5: Come vediamo dalla tabella si tratta di una probabilità condizionata dall'esempio i -esimo. Indicando con k il numero di successi sul numero totale di esperimenti (che noi limiteremo a 1), possiamo rappresentare la funzione di distribuzione con la seguente

$$f(k | x_i) = \begin{cases} p & \text{if } k = 1 \\ 1 - p & \text{if } k = 0 \end{cases} = p^k [1 - p]^{1-k}$$

Quello che faremo è far giocare il ruolo che nella formula precedenza giocava k , alla variabile Y_i , cioè se $Y_i = 1$ vuol dire che l'esempio osservato appartiene alla classe positiva con probabilità p . Sintetizzato:

$$f(Y_i | x_i) = p_i^Y [1 - p_i]^{1 - Y_i}$$

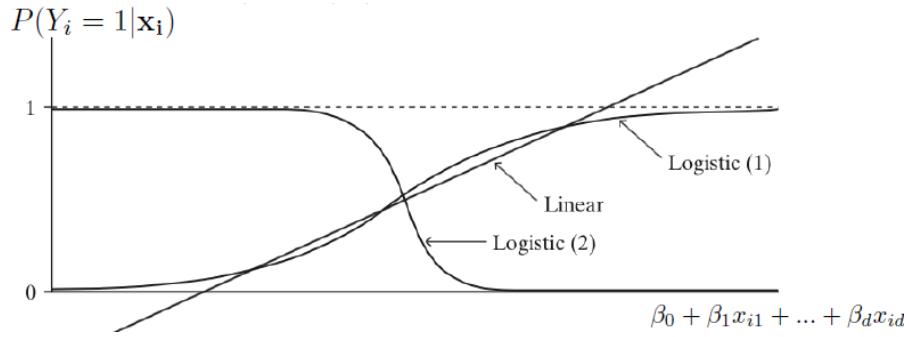


Figura 9.6: Applicazione della funzione sigmoide al risultato della regressione lineare.

Quindi l'idea è di predire la probabilità della classe positiva una volta osservato l'esempio i-esimo con una regressione lineare, combinando cioè in maniera lineare tutte le feature dell'esempio i-esimo:

$$P(Y_i = 1 | x_i) = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_d x_{id}$$

Purtroppo però il risultato della regressione lineare restituisce un valore non limitato nell'intervallo

0, 1

, bensì lo è tra

$-\infty, +\infty$

, e quindi non può essere considerata come probabilità. Occorre quindi utilizzare una funzione sigmoide che limiti il risultato (figura 9.6) e allora:

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

Per i valori per cui $P(Y_i = 1 | x_i) \approx 0$ la classe da predire è quella negativa, mentre per i valori per cui $P(Y_i = 1 | x_i) \approx 1$ la classe predetta sarà quella positiva: è interessante focalizzarsi sul momento di transizione tra le classi.

Per semplificare i conti rispetto alla forma sigmoidale possiamo rappresentare esplicitare l'esponente come l'*Odds-ratio*, cioè il rapporto tra le due probabilità opposte:

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

e si avrà che

$$\ln(\text{Odds}(Y_i | x_i)) = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_d x_{id}$$

Quindi, dulcis in fundo, si utilizzano gli strumenti della regressione lineare per calcolare una funzione della probabilità di essere nella classe positiva osservato l'esempio i-esimo.

9.15.1 I modelli lineari generalizzati

Il modello di regressione logistica usa quindi la regressione per pre-dire il logaritmo dell'Odds-ratio. Esso è un caso particolare di *modelli lineari generalizzati* che, dopo aver calcolato la funzione di regressione tramite una funzione lineare (utilizzando dei pesi) che produce un target Y_i , applicano a questo una funzione f :

$$f(Y_i) = g(x_i) = w_0 + w_1 x_{i1} + \cdots + w_d x_{id} = w^T x_i$$

dove $f(Y_i)$ può essere:

- $f(Y_i) = Y_i$ (caso più semplice)
- $f(Y_i) = \frac{1}{Y_i}$
- $f(Y_i) = \log(Y_i)$
- $f(Y_i) = \log(Y_i/1 - Y_i)$ (ovvero il caso della regressione logistica appena visto)

Se, a partire dalla funzione, vogliamo ottenere il target dobbiamo quindi poter invertire la funzione f .

9.15.2 Il modello di regressione logistica applicata a feature categoriche

In realtà il modello di regressione logistica può essere applicato anche nel caso in cui le feature assumano valori categorici (nominali).

Per farlo si può ricorrere agli Odds, che ricordiamo essere il rapporto tra la probabilità di un evento e il suo opposto, e sostituirli nella formula della regressione. Supponiamo di avere una variabile Y che può assumere valori (nominali) A e B e che si comporta secondo la tavola di contingenza in Figura 9.7, allora procederemo al seguente calcolo.

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 9.7: ...

$$\begin{cases} P(Y = 1 | X = A) = \frac{n_{A1}}{n_A} \\ P(Y = 0 | X = A) = \frac{n_{A0}}{n_A} \end{cases} \Rightarrow \text{Odds}(A) = \frac{P(Y = 1 | X = A)}{P(Y = 0 | X = A)} = \frac{n_{A1}}{n_{A0}}$$

e

$$\begin{cases} P(Y = 1 | X = B) = \frac{n_{B1}}{n_B} \\ P(Y = 0 | X = B) = \frac{n_{B0}}{n_B} \end{cases} \Rightarrow \text{Odds}(B) = \frac{P(Y = 1 | X = B)}{P(Y = 0 | X = B)} = \frac{n_{B1}}{n_{B0}}$$

Sostituendo la Odds (al posto dei valori x_{ij}) nella formula si ottiene la seguente:

$$\ln(\text{Odds}(Y_i | x_i)) = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_d x_{id}$$

9.15.3 Un piccolo cambio di notazione

Ponendo le nostre scuse ai lettori, vogliamo ora uniformare la notazione usata nel corso con quelle del libro, e perciò se prima usavamo la formula di regressione $\beta_0 + \beta_1 x_{i1} + \cdots + \beta_d x_{id}$ indicheremo ora con il termine $-t$ il termine noto (ai secoli beta₀), con w il vettore dei pesi e con x quello dei dati. Avremo quindi la nuova formula della regressione lineare: $-t + w \cdot x$.

Così, riscriveremo la probabilità della classe positiva osservato l'esempio i-esimo $P(Y_i = 1 | x_i)$ come

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

E riscriveremo anche la distribuzione di bernoulli per ogni esempio x_i come

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

Si evidenzia come i parametri della regressione lineare w e t siano nascosti nella formula di $\hat{p}(x)$, e quindi esiste un parametro per ciascuna feature moltiplicata per ogni istanza i-esima.

Quasi a non essere soddisfatti, riscriviamo anche la formula della likelihood, che battezzeremo ora *conditional likelihood* ad evidenziare il fatto che si basa sulla probabilità condizionata $P(y_i | x_i)$ invece che su $P(x_i)$ come nei modelli generativi:

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

Ovviamente per permetterci di fare quella produttoria siamo a conoscenza di indipendenza dei valori di y fissato x (che non è una assunzione forte come quella di naive Bayes in cui si assume che ogni esempio è dato indipendente).

La forma logaritmica della likelihood sarà quindi:

$$LCL(w, t) = (\sum_i y_i \ln \hat{p}(x_i)) + (1 - y_i) \ln (1 - \hat{p}(x_i))$$

Ricordando che y_1 assume valore 1 quando l'esempio appartiene alla classe positiva e 0 quando non è così, possiamo vedere come, nel

primo caso (classe positiva) rimane solo la prima sommatoria, mentre nel secondo caso rimane solamente il logaritmo. Possiamo allora riscrivere

$$LCL(w, t) = \sum_{x^\oplus \in Tr^\oplus} \ln \hat{p}(x^\oplus)) + \sum_{x^\ominus \in Tr^\ominus} \ln \hat{p}(x^\ominus))$$

dove Tr^\oplus e Tr^\ominus rappresentano rispettivamente la parte di training set dei soli esempi positivi e di quelli negativi.

9.15.4 Ritorno alla likelihood

Ora che abbiamo concluso la rivisitazione delle notazioni, arrivando a definire la likelihood nei termini logaritmici possiamo procedere con quanto fatto in precedenza con questa formula: massimizzarne il valore al fine di stimare i parametri della legge di distribuzione che rende più verosimile il dataset.

Per poter far ciò, andremo ad egualare a zero le d derivate parziali rispetto a w (in realtà tratterò ad uno ad uno i parametri w_1, \dots, d) e trovare così il massimo del logaritmo della likelihood (essendo il logaritmo una funzione monotona, dove è massimo il logaritmo della likelihood lo è anche la likelihood stessa):

$$\begin{aligned} \nabla_w LCL(w, t) &= 0 \\ \frac{\partial}{\partial t} LCL(w, t) &= 0 \Leftrightarrow \\ \Leftrightarrow \sum_{x^\oplus \in Tr^\oplus} (\hat{p}(x) - 1) + \sum_{x^\ominus \in Tr^\ominus} \hat{p}(x^\ominus) &= \sum_{x_i \in Tr} (\hat{p}(x_i) - y_i) \end{aligned}$$

Se guardiamo il risultato ottenuto, stiamo sommando per ogni esempio, la sua probabilità di appartenere alla classe positiva ridotta del valore reale del target per l'esempio. In altre parole, altro non è che l'errore commesso nella stima della classe per gli esempi: se è o vuol dire che il modello predice al meglio (ecco che anche sotto questo punto di vista uguagliarlo a 0 ha un senso).

Ma se poniamo a zero questa somma, stiamo dicendo che, in media, la probabilità predetta sommando le $\hat{p}(x_i)$ è uguale alla proporzione di esempi positivi nel dataset.

Questo modello di regressione logistica fornisce allora un valore della probabilità per un esempio di appartenere alla classe positiva del tutto simile a quello che facevano altri modelli (ad esempio gli alberi costruivano segmentazioni dello spazio degli esempi e nelle foglie mostravano la probabilità di appartenenza ad una classe sotto le spoglie dell' empirical probability).

L'altra derivazione parziale della likelihood logaritmica che possiamo fare è quella rispetto ai pesi:

$$\frac{\partial}{\partial w_j} = \sum_{x_j \in Tr} (y_i - \hat{p}(x_i)) x_{ij}$$

Ponendo a zero questa derivata otteniamo ovviamente il valore ottimo.

Ma la derivata sarà uguale a zero esattamente quando y_i è uguale alla probabilità stimata della classe per tutti gli esempi.

Si nota però che l'errore commesso nella stima ($y_i - \hat{p}(x_i)$) viene pesato con il valore della j -esima feature: se tale feature non esiste (o vale zero) l'errore non viene considerato e quindi questa formulazione risulta più rilassata rispetto alla precedente.

Ponendo a zero queste due derivate parziali rendiamo massimi i parametri w e t :

$$w^*, t^* = \underset{w,t}{\operatorname{argmax}} \text{CL}(w, t) = \underset{w,t}{\operatorname{argmax}} \text{LCL}(w, t)$$

Di altro non si tratta se non di un *problema di ottimizzazione convesso*, per cui per definizione esiste un solo massimo e per cui possono essere applicati diverse tecniche di risoluzione.

Un approccio possibile è quello ispirato all'algoritmo di apprendimento dei pesi di un, iterando sugli esempi (ciascuno apporterà un contributo alla discesa del gradiente) e applicando la seguente regola di aggiornamento dei pesi:

$$w = w + \eta(y_i - \hat{p}_i)x_i$$

Ci teniamo a sottolineare la continua vicinanza che troviamo in questo argomento con i principi base delle reti neurali.

9.15.5 La regressione logistica a confronto

Vediamo ora un confronto finale, applicando il modello di apprendimento linguistici a esempi pratici.

Mostriamo in figura 9.8a un dataset con due classi (positivi e negativi). In blu viene mostrato l'iperpiano di separazione tra le due classi ottenuto tramite regressione logistica, e si nota che questo è molto più inclinato rispetto ad altri iperpiani ottenuti con altri modelli lineari (classificatore di base lineare in rosso e classificatore basato su riduzione degli scarti quadratici in giallo) e si comporta meglio poiché si concentra di più a modellare la separazione tra gli esempi delle classi che sono più vicini tra di loro (hanno un impatto maggiore gli esempi vicino al bordo), mentre gli altri due non tengono (troppo) in considerazione degli esempi lontani dal centro.

Nella figura 9.8b abbiamo invece un altro dataset con esempi distribuiti in modo differente. In questo la regressione logistica modella una retta quasi orizzontale e commette un numero maggiore di errori rispetto alle altre due rette poiché si concentra troppo nella parte di transizione tra le due classi senza guardare la restante distribuzione della classe.

Nella figura 9.9 rappresentiamo invece un dataset con due classi. Gli

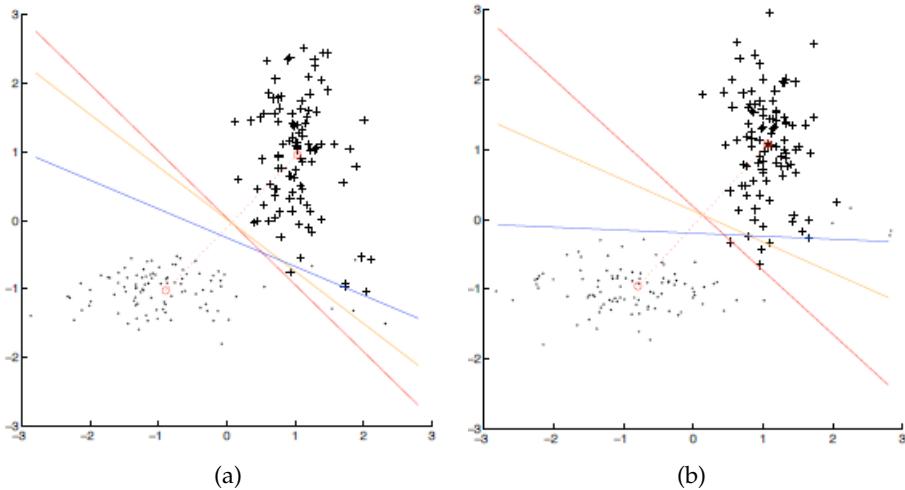


Figura 9.8: Due esempi di dataset in cui si pone un confronto tra la regressione logistica (in blu), il classificatore lineare di base (in rosso) e il classificatore basato sulla riduzione degli scarti quadratici (in giallo).

esempi della classe negativa sono rappresentati lungo l'ascissa, mentre sulla retta $y = 1$ troviamo gli esempi positivi (i due pallini rappresentano i valori medi per le due classi). Il modello in rosso è quello che rappresenta la regressione logistica ed è quello che meglio rappresenta la transizione tra le due classi. In blu vediamo invece la rappresentazione di un modello generativo di tipo Gaussiano calibrato con funzione sigmoidale (calibrazione logistica).

Nonostante gli esempi siano stati generati secondo una legge gaussiana, la regressione logistica, spingendosi vicino al bordo, commette un errore minore (poiché la transizione è molto ripida).

Ottimi risultati li ottiene anche il modello isotonico (in verde, vedi capitolo 10) che riesce a slegarsi anch'esso dalla legge che ha generato il dataset ed arrivare velocemente al bordo.

9.16 ALGORITMO EM

9.16.1 Obiettivi

Vedremo ora una generalizzazione dell'algoritmo di clustering delle k medie con un setting di tipo probabilistico.

Questo algoritmo di clustering ha il vantaggio di produrre una legge generativa del cluster, fornendo i parametri della distribuzione di probabilità che potrebbe aver generato i cluster.

Questo algoritmo associa ad ogni esempio un peso, una probabilità, di appartenenza ai cluster.

Dal punto di vista probabilistico, altro non facciamo che guardare gli esempi e trovare la legge generativa dei cluster che è quella più vero-

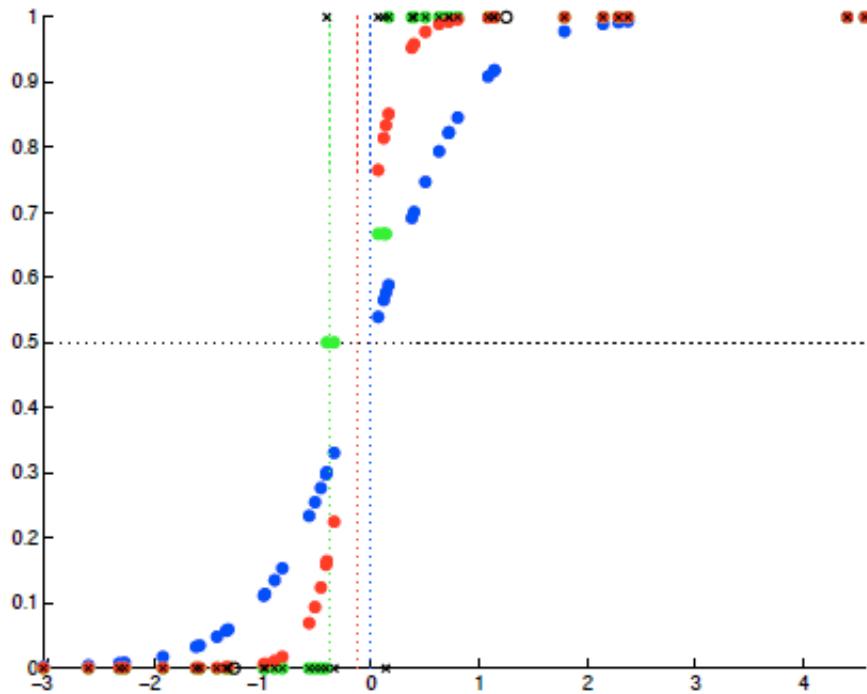


Figura 9.9: Confronto della regressione lineare con esempi univariati.

simile per quei dati (analogamente a quanto fatto massimizzando la likelihood).

Parliamo di mistura finita di funzioni probabilità poiché si ipotizza di avere un numero finito di cluster (k) e si vogliono trovare le k funzioni di distribuzione ciascuna generativa del corrispondente cluster. Parliamo di mistura poiché un unico punto dello spazio degli esempi potrebbe far riferimento k funzioni di probabilità diverse.

Supponiamo che tutte le k distribuzioni saranno di tipo Gaussiano, governate quindi dai parametri μ (valore medio) e σ (deviazione standard), saranno così rappresentabili:

$$f_i(x; \mu_i, \sigma_i) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}}$$

Ogni cluster è stato sicuramente generato da una distribuzione ed il nostro modello associa ad ogni distribuzione una probabilità di appartenenza del cluster, quindi

$$\sum_i P(C_i) = 1$$

La distribuzione risultante sarà quindi data pesando le k distribuzioni dei cluster sulle loro probabilità, e si ottiene, sommandole, la legge complessiva (*mistura finita*) che governa lo spazio:

$$f(x) = \sum_i P(C_i) f_i(x; \mu_i, \sigma_i)$$

È importante non confondersi tra questi pesi $P(C_i)$ e i pesi che ogni cluster attribuisce agli esempi (ogni esempio potrebbe infatti appartenere a più di un cluster) Se noi conoscessimo a quale cluster appartiene ciascun punto, potremmo molto facilmente ottenere i parametri ignoti della distribuzione:

$$\mu_{C_i} = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

$$\sigma_{C_i}^2 = \frac{(x_1 - \mu_{C_i})^2 + (x_2 - \mu_{C_i})^2 + \cdots + (x_n - \mu_{C_i})^2}{n-1} \text{ si usa } n-1 \text{ poiché l'ultimo grado di libertà è governato}$$

Se viceversa conoscessimo le distribuzioni dei cluster (con loro parametri), potremmo facilmente stimare i pesi di appartenenza di ciascun esempio a ciascun cluster (con il teorema di Bayes):

$$\hat{P}(C_i | x) = \frac{f(x | C_i)P(C_i)}{f(x)}$$

e potremmo assegnare l'esempio x al cluster con probabilità massima.

Il problema è quando non conosciamo nulla: ne le etichette dei cluster per ciascun dato, ne i parametri delle distribuzioni generative. Vediamo solo come i dati si sono distribuiti nello spazio.

9.16.2 Introduzione all'algoritmo

L'algoritmo di *Expectation-Maximisation* risolve tale problema con modalità iterativa, ovvero ciclando tra due fasi fino a raggiungere una stabilità. In particolare:

EXPECTATION STEP in cui vengono fissati i parametri dei cluster ($\mu_{C_i}, \sigma_{C_i}, P(C_i)$) e determinato in quale cluster ciascuna istanza potrebbe appartenere.

MAXIMIZATION STEP Una volta associata a ciascuna istanza un certo peso di appartenenza ad un cluster, si usano questi pesi per riaggiornare i parametri dei cluster.

In un certo senso riconosciamo l'operato di k-means che inizializza i centroidi (inizialmente random), valuta la appartenenza di ogni punto al cluster e poi ricalcola il centroide.

Assumendo di voler trattare k cluster (k deciso a monte), per determinare i parametri delle loro distribuzioni possiamo utilizzare la likelihood del dataset dato un cluster (la verosomiglianza dei parametri osservati i dati), e cercare un suo massimo locale tramite un algoritmo.

Questa likelihood sarà data da una produttoria (assumendo gli esempi indipendenti) applicata alla mistura finita

$$L(X; \mu_i, \sigma_i) = \prod_j \sum_i P(C_i) f(x_j | C_i)$$

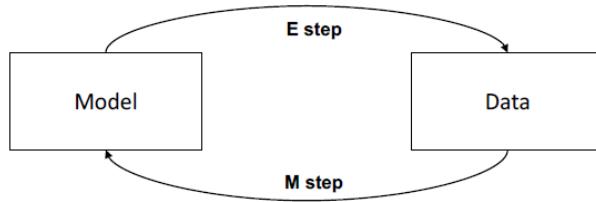


Figura 9.10: Rappresentazione del susseguirsi delle fasi di Expectation e Maximisation nell'algoritmo EM

Ma ricordiamo che

$$f(x_j | C_i) = f(x_j; \mu_i, \sigma_i) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{(x_j - \mu_i)^2}{2\sigma_i^2}}$$

Allora

$$L(x_j; \mu_i, \sigma_i) = \prod_j \sum_i P(C_i) \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{(x_j - \mu_i)^2}{2\sigma_i^2}}$$

A livello pratico, possiamo ovviamente considerare il logaritmo ottenendo due sommatorie.

9.16.3 *L'algoritmo*

La figura 9.10 riprende la rappresentazione dell'algoritmo EM già parzialmente discussa nella sottosezione 9.16.2. Scendiamo ora nel dettaglio delle due fasi. In parte si tratterà di riprendere quanto già visto nella fase introduttiva, ma riteniamo utile mettere in ordine gli elementi visti.

9.16.3.1 *Expectation Step*

In questa fase viene attribuita ad ogni esempio x_j la sua probabilità di appartenenza al cluster C_i fermo restando i parametri attuali della distribuzione. Tale probabilità sarà data da

$$\hat{P}(C_i | x_j) = \frac{f(x_j | C_i)P(C_i)}{f(x_j)} = w_{ij}$$

Dove ricordiamo ancora che

$$f(x_j | C_i) = f(x_j; \mu_i, \sigma_i) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{(x_j - \mu_i)^2}{2\sigma_i^2}}$$

Potrebbe sembrare strano, nella formula di Bayes, vedere a numeratore una moltiplicazione tra una funzione di distribuzione di probabilità e una probabilità: ci si aspetterebbe una integrazione di quella f (in un intervallo compreso tra 0 ed 1), al fine di avere una reale probabilità. La formula funziona poiché anche a denominatore si ha una funzione (in qualche modo si annullano) e quindi il teorema di Bayes

vale ancora.

Il valore di w_{ij} indica la probabilità per l'esempio j -esimo di appartenere al cluster i -esimo.

9.16.3.2 Maximisation Step

Una volta fissata l'appartenenza dei dati a determinati cluster si stima i parametri della distribuzione secondo la attribuzione ai cluster degli esempi di tutto il dataset.

Questo si fa applicando il principio di *Maximum Likelihood Estimation* che già conosciamo, e che mira a massimizzare la likelihood sul dataset X (con istanze x_j indipendenti) così definita. In particolare, essendo la likelihood definita dalla formula

$$L(X; \mu_i, \sigma_i) = \prod_j f(x_j) = \prod_j \sum_i P(C_i) \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{(x_j - \mu_i)^2}{2\sigma_i^2}}$$

La massimizzazione del valore $L(X; \mu_i, \sigma_i)$ corrisponde alla applicazione dei seguenti stimatori pesati:

LA PROBABILITÀ DEL CLUSTER

$$P(C_i) = \frac{1}{n} \sum_{j=1}^n P(C_i | x_j) = \frac{1}{n} \sum_{i=1}^n w_{ij}$$

IL CENTROIDE PER IL CLUSTER i -ESIMO Come visto prima, il valore μ_i non è altro che la media campionaria dei dati osservati pesati con le loro probabilità di appartenenza al clustering predetto:

$$\mu_i = \frac{w_{i1}x_1 + w_{i2}x_2 + \dots + w_{in}x_n}{w_{i1} + w_{i2} + \dots + w_{in}}$$

LO SCARTO QUADRATICO PER IL CLUSTER i -ESIMO

$$\sigma_i^2 = \frac{w_{i1}(x_1 - \mu_{C_i})^2 + w_{i2}(x_2 - \mu_{C_i})^2 + \dots + w_{in}(x_n - \mu_{C_i})^2}{w_{i1} + w_{i2} + \dots + w_{in}}$$

9.16.3.3 Condizione di terminazione

Per stabilire quando l'algoritmo può terminare, ad ogni iterazione avviene un confronto tra il logaritmo della likelihood attuale e quello all'iterazione precedente: quando non avviene nessun miglioramento (la likelihood è cioè *satura*) l'algoritmo ha raggiunto una convergenza e può terminare. Il logaritmo della likelihood (per la sola funzione di distribuzione del cluste i -esimo) sarà:

$$\log(L(X; \mu_{C_i}, \sigma_{C_i})) = \sum_j \frac{(x_j - \mu)^2}{2\sigma^2} - \frac{1}{k} n \log(2\pi) - n \log(\sigma)$$

Se confrontato con k-means, si noterà che questo algoritmo ha una convergenza generalmente più lenta.

9.17 MODELLI BASATI SU COMPRESSIONE

Parleremo oggi su modelli basati sul principio della compressione, cardine della teoria dell'informazione.

Avevamo già parlato dell'entropia, accennando al suo utilizzo ai fini di modellare la quantità di informazione contenuta in un messaggio: rappresenta quindi il limite teorico di comprimibilità di un contenuto informativo.

Questa quantità può essere ottenuta dall'inverso del logaritmo della probabilità massima a posteriori (MAP).

Vediamo come procedere:

$$y_{MAP} = \operatorname{argmin}_y -\log P(X = x | Y = y) - \log P(Y = y)$$

Se un evento ha probabilità p di capitare, il logaritmo negato di p quantifica il contenuto di informazione del messaggio che l'evento ha scatenato.

Scrivremo (IC = information content)

$$IC(X | Y) = -\log_2 P(X | Y) \text{ and } IC(Y) = -\log P(Y)$$

Se usiamo il logaritmo in base 2 presupponiamo che il bit è l'unità di misura del contenuto del messaggio: l'unità di misura dell'informazione dipende dalla base del logaritmo.

Ad esempio, il lancio di una moneta (due facce) causerà la comunicazione dell'esito che sarà un messaggio contenente una informazione misurabile in $-\log_2(1/2) = 1$ bit.

Il messaggio di esito del lancio di un dado a 6 facce conterrà invece una informazione misurabile in $-\log_2(1/6) = 2.6$ bit.

9.17.1 Esempio di applicazione del principio

Vogliamo ora presentare un esempio di applicazione del principio compressione sul classico esempio delle email.

Rappresentiamo nella tabella in figura 9.11 la probabilità dell'evento di comparsa di una parola (Viagra) in una classe Y che può essere spam o ham. Se la classe Y ha valori equiprobabili allora $IC(Y) =$

Y	$P(Viagra = 1 Y)$	$IC(Viagra = 1 Y)$	$P(Viagra = 0 Y)$	$IC(Viagra = 0 Y)$
spam	0.40	1.32 bits	0.60	0.74 bits
ham	0.12	3.06 bits	0.88	0.18 bits

Figura 9.11: Probabilità dell'evento di comparsa della parola Viagra in una classe Y che può essere spam o ham.

$\text{spam}) = IC(Y = \text{ham}) = -\log_2(1/2) = 1$ e allora

$$\operatorname{argmin}_y (IC(Viagra = 1 | Y = y) + IC(Y = y)) = \text{spam}, \text{ poiché } 1.32 < 3.06$$

$$\underset{y}{\operatorname{argmin}} (\text{IC}(\text{Viagra} = 0 \mid Y = y) + \text{IC}(Y = y)) = \text{ham}, \text{ poiché } 0.18 < 0.74$$

Se assumo invece un caso in cui i valori per la classe Y non sono equiprobabili ma il valore ham è quattro volte più probabile (ho un filtro anti spam molto valido) allora $\text{IC}(Y = \text{spam}) = -\log_2(0.2) = 2.32$ e $\text{IC}(Y = \text{ham}) = -\log_2(0.8) = 0.32$ e quindi

$$\underset{y}{\operatorname{argmin}} (\text{IC}(\text{Viagra} = 1 \mid Y = y) + \text{IC}(Y = y)) = \text{ham}(3.38), \text{ poiché } 3.06 + 0.32 < 1.32 + 2.32$$

Come confermano i numeri, nonostante la presenza della parola Viagra, la classe ham conviene poiché la probabilità della classe ha peso maggiore.

9.17.2 Il principio della "minimum description lenght"

Quello che vogliamo fare ora è applicare il principio dell'information content alla scelta dei modelli.

In generale, nel machine learning, abbiamo infatti un dataset dato, un certo task da risolvere (ad esempio di tipo predittivo) e un certo insieme di modelli da applicare per risolverlo: ogni modello avrà una complessità diversa (da quelli molto complessi fino a fare overfitting) e da questa dipende la lunghezza del messaggio che comunicano (più è complesso, più il messaggio sarà grande).

Questa complessità la possiamo vedere composta da due elementi:

- $L(m)$ che quantifica la complessità del modello
- $L(D \mid m)$ che quantifica gli errori commessi su un certo dataset D applicando il modello m

Un modello molto complesso avrà probabilmente il primo termine elevato e il secondo ridotto. Occorre quindi scegliere il modello (secondo il principio di *Minimum description lenght*) secondo la regola

$$m_{MDL} = \underset{m \in M}{\operatorname{argmin}} (L(m) + L(D \mid m))$$

Sia $L(m)$ la lunghezza in bit della descrizione di un modello m (information content del modello m) e sia $L(D \mid m)$ la lunghezza in bit della descrizione di un dato D in un modello m.

In altre parole il principio MDL bilancia nella scelta del modella la complessità del modello stesso con la sua accuratezza.

10

FEATURES

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

10.1 DIVERSI TIPI DI FEATURE

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

10.2 TRASFORMAZIONE DELLE FEATURE

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

APPRENDIMENTO ENSEMBLE

L'apprendimento *ensemble*, altrimenti noto in inglese come *ensemble learning*, può trovare la sua traduzione letterale in italiano nel termine *apprendimento dell'insieme*.

L'ensemble learning è un nome che fa da cappello ad un insieme di tecniche rivolte alla risoluzione di problemi di classificazione o regressione combinando i risultati di altri algoritmi di apprendimento. In particolare si sceglie un determinato algoritmo di apprendimento da fornire in input ad un algoritmo di ensemble learning, il quale ripete la fase di apprendimento più volte per combinarne i risultati e ottenere delle performance migliori rispetto a quelle del singolo algoritmo usato in input.

Alla base della ricerca su questi argomenti vi è una fondamentale domanda teorica: l'apprendimento forte è equivalente all'apprendimento debole?

APPRENDIMENTO DEBOLE Su una certa classe di problemi di apprendimento si riesce a fare meglio di una scelta casuale (50 : 50), ovvero superare anche se di poco la precisione di 0.5.

APPRENDIMENTO FORTE Su una certa classe di problemi di apprendimento si riesce a raggiungere quasi una precisione di 1.

Da qui la seconda domanda: avendo a disposizione per una certa classe di problemi un algoritmo di apprendimento debole, si può riuscire a costruire un algoritmo di apprendimento forte? Se questo fosse possibile allora si può sempre riuscire a risolvere tale classe di problemi anche in modo forte.

11.1 FUNZIONAMENTO DELL'APPRENDIMENTO ENSEMBLE

Con la Figura 11.1 si vogliono percorrere i passi principali del funzionamento di tale tecnica di apprendimento. In alto c'è il dataset originale di partenza D , da cui si ottengono T nuovi dataset¹ dove T è un parametro a scelta dell'utente. I nuovi dataset così ottenuti diventano l'input per l'algoritmo di apprendimento (solitamente debole) che viene scelto, e questo fornirà in output esattamente T modelli che vengono infine combinati per ottenere un modello M finale e definitivo con il quale risolvere il problema di apprendimento.

A fronte di questo schema però bisogna ancora comprendere come prendere due importanti decisioni che riguardano l'apprendimento ensemble:

¹ Il modo in cui tali dataset vengono creati a partire da quello originale può variare.

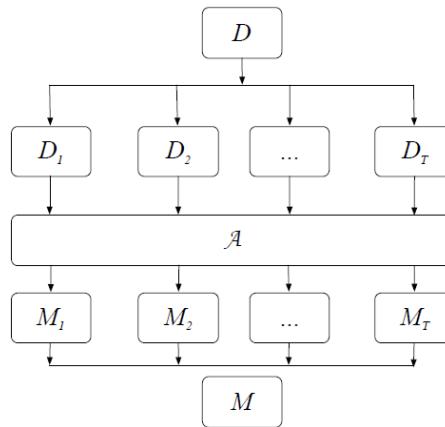


Figura 11.1: Schema del funzionamento dell’ensemble learning. In alto si ha il dataset di partenza D che viene successivamente utilizzato per ottenere T (parametro a scelta dell’utente) nuovi dataset (la ripartizione delle istanze può avvenire in modi diversi). Tali nuovi dataset sono l’input dell’algoritmo di apprendimento prescelto \mathcal{A} , dal quale si ottengono lo stesso numero T di modelli che vengono combinati per ottenere un unico modello finale M .

- come generare i classificatori di base e quindi come generare i relativi T dataset;
- come si integrano, come si combinano i classificatori di base fra di loro per ottenerne uno unico finale.

11.1.1 Come combinare diversi modelli

L’ultimo passaggio visto in Figura 11.1 vedeva la combinazione dei diversi modelli ottenuti dall’algoritmo di apprendimento, diventa quindi rilevante capire come avviene questa integrazione.

La quasi totalità delle tecniche di ensemble learning si basano sul seguente approccio comune:

$$M(x) = f \left(\sum_{i=1}^T w_i M_i(x) \right) \quad (11.1)$$

dove si ha:

- x è un esempio del dataset;
- $M_i(x)$ è la risposta del singolo classificatore i per l’esempio x ;
- w_i è il “peso” associato al modello M_i ;
- $w_i > w_j$ vuol dire che il modello M_i è più affidabile del modello M_j ;
- solitamente tutti i pesi sono strettamente positivi, ovvero $w_i > 0$, altrimenti un valore negativo implicherebbe che il modello sbagli più sovente di quanto non sia giusto.

11.1.2 Come generare diversi modelli di base

La differenza maggiore fra le diverse tecniche di apprendimento ensemble risiede però nel modo in cui vengono ottenuti i dataset e di conseguenza i modelli di base. Gli approcci sono molteplici:

BAGGING Abbreviazione di *Bootstrap Aggregation*, verrà affrontato più avanti nella sezione 11.2;

BOOSTING verrà affrontato più avanti nella sezione 11.3, in particolare la versione AdaBoost (ma è anche famosa la versione Logi-Boost);

ECOC Abbreviazione di *Error-Correcting Output Codes*;

STACKING

L'elenco sopra riportato ovviamente non è esaustivo, sono stati riportati solamente i principali approcci. I primi due verranno affrontati nelle seguenti sezioni.

11.2 BOOTSTRAP AGGREGATION

L'idea alla base di questa tecnica, quella che caratterizza effettivamente il bagging è come vengono costruiti i T dataset a partire da quello originale D . Ed in secondo luogo come vengono successivamente combinati i modelli ottenuti dall'algoritmo di bagging.

Come si può facilmente notare dall'Algoritmo 23 il codice è molto semplice e l'operazione cardine è proprio la prima all'interno del ciclo `for`. Si tratta appunto della costruzione dei nuovi dataset per addestrare i T modelli. Tali dataset vengono ottenuti come repliche di bootstrap² dal dataset di partenza D , ovvero con un campionamento della stessa cardinalità $|D|$ del dataset, quindi includendo eventuali ripetizioni di uno stesso esempio e senza rimuovere lo stesso dall'insieme D originale.

Come è già stato detto, la seconda caratteristica che contraddistingue questa tecnica è il modo in cui vengono combinati i modelli ottenuti in output dall'algoritmo in un unico modello finale. Tale combinazione è un semplice meccanismo di maggioranza, per ricondurci ad una formulazione matematica più chiara è come se si avessero tutti i pesi $w_i = 1$, pertanto il modello risultante applicando tali pesi alla Formula 11.1 sarà:

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

² La tecnica della replica di bootstrap applicata numerose volte permette di ottenere una distribuzione che tende ad essere simile a quella del dataset D di partenza, che è cosa buona e giusta perché non si alternano le caratteristiche del problema mettendo in evidenza esempi che nella realtà non lo erano.

dove f è la funzione segno, pertanto il risultato è $+1$ se la sommatoria risulta positiva, -1 se al contrario la sommatoria risulta negativa.

Di seguito viene riportato lo pseudocodice dell'algoritmo:

Algorithm 23 Bagging(D, T, \mathcal{A}) - Addestrare un insieme di modelli partendo da dataset costruiti con il metodo *bootstrap*.

Input: dataset D ; dimensione dell'insieme T ; algoritmo di apprendimento di base \mathcal{A} .

Output: insieme di modelli le cui predizioni vengono combinate per ottenere un unico modello finale.

for $t = 1$ to T **do**

 Si costruisce una replica di bootstrap del dataset D ;

 Si lancia l'algoritmo di apprendimento \mathcal{A} per produrre il modello M_t ;

end for

return $\{M_t | 1 \leq t \leq T\}$

Si vuole a questo punto tornare a parlare delle repliche di bootstrap per focalizzarsi sull'importanza chiave che rivestono per la buona riuscita dell'ensemble learning.

L'apprendimento ensemble si basa infatti su tale metodo di generazione dei dataset per fare in modo che siano tutti diversificati. Se fossero tutti uguali, infatti, si otterrebbero modelli che classificano in modo identico e la loro combinazione finale non potrebbe nessun miglioramento alle prestazioni all'algoritmo di apprendimento debole utilizzato. La diversificazione dalle repliche di bootstrap viene garantita dall'estrazione (senza rimozione) già descritta in precedenza.

Volendo capire quanto sono diversi questi dataset ottenuti con la replica di bootstrap, si può andare a valutare la percentuale di esempi che vengono esclusi in ogni replica: 37% circa non fanno parte della replica. Questo è molto facile da dimostrare: la probabilità che un esempio non faccia parte della replica di bootstrap è $(1 - \frac{1}{n})^n$, il limite di questa espressione per valori di n che tendono ad infinito è $\frac{1}{e} = 0.3679$.

11.2.1 Esempio applicativo del Bagging

A questo punto si desidera vedere l'algoritmo del Bagging in azione per verificare con i propri occhi che un sistema che sembra abbastanza semplice riesce a produrre effettivamente dei risultati apprezzabili.

Si prenda quindi in considerazione la Figura 11.2 nella quale è stato riportato un grafico dove i puntini sono tutti gli esempi che compongono il dataset; tali esempi sono classificati con due colori, blu e rosso. Le due classi identificate si possono dividere molto bene con una ellisse.

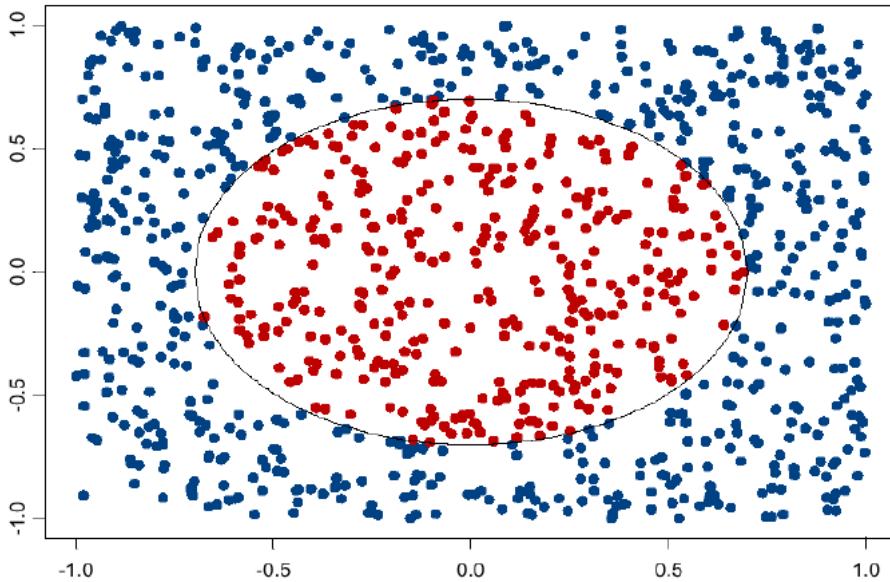


Figura 11.2: Dataset di esempio per l'algoritmo di Bagging con due classi identificate dai colori blu e rosso.

Se si cerca di risolvere questo task di classificazione utilizzando un singolo [CART](#), degli alberi di decisione che possono essere utilizzati sia per task di classificazione che per task di regressione, proprio come suggerisce il nome stesso, il risultato è mostrato in rosso nella Figura 11.3. Risulta evidente come la superficie ottenuta dall'albero di decisione sia una approssimazione molto grezza dell'ellisse corrispondente alla corretta suddivisione delle due classi di esempi, ciò è dovuto alle difficoltà di questo tipo di modelli nell'approssimare decision boundary curvilinei o diagonali.

Tuttavia se si utilizza un gran numero di volte questi alberi di decisione si ottengono ogni volta delle superfici con un contorno spezzato in modo diverso, e che avranno in comune sempre una buona parte della porzione centrale dell'ellisse. Il risultato della media viene riportato graficamente in Figura 11.4, ivi si può notare come ora il contorno ottenuto mediando su un numero di 100 [CART](#) sia una approssimazione molto più apprezzabile dell'ellisse. Quest'ultimo infatti viene riempito maggiormente con il rosso che corrisponde al modello finale, mentre in bianco sono evidenziati i punti in cui il modello è più incerto, e queste zone sono limitate, come ci si aspetterebbe, solo ai contorni dell'approssimazione dell'ellisse.

11.2.2 Random forest

L'ultimo argomento collegato al Bagging è quello che tratta delle *random forest*, le foreste casuali. Fra tutti gli algoritmi di apprendimento utilizzabili nel Bagging ve ne è uno che spicca in particolare perché tutt'oggi rappresenta lo stato dell'arte: tale classificatore viene consi-

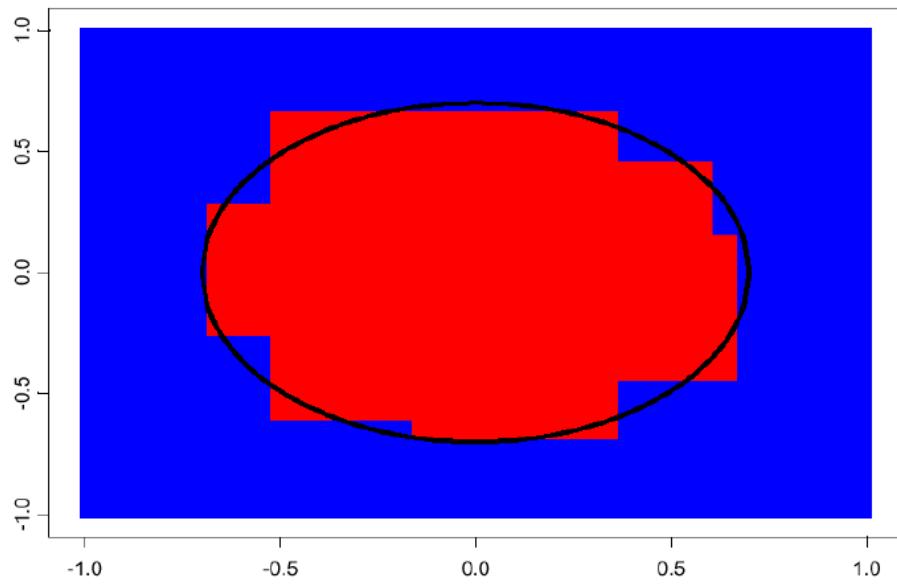


Figura 11.3: In rosso è evidenziata la superficie corrispondente al risultato ottenuto dall'applicazione di un albero di decisione [CART](#) per la risoluzione del problema di classificazione introdotto con il dataset in Figura 11.2. In nero l'ellisse corrispondente alla classificazione corretta.

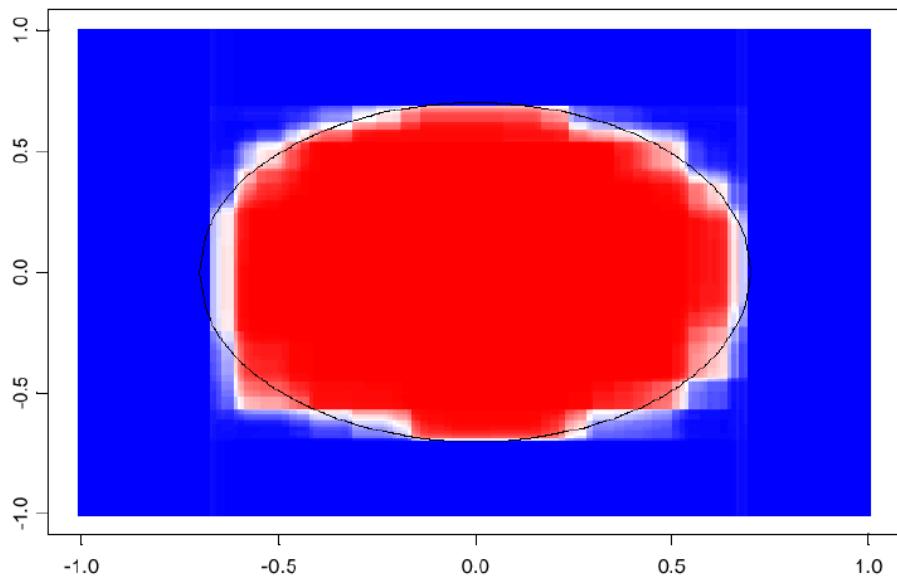


Figura 11.4: Approssimazione grafica della media di 100 alberi di decisione [CART](#) utilizzati per classificare gli esempi rossi e blu divisi dall'ellisse nera. Il colore bianco evidenzia le zone in cui il modello è più incerto nella classificazione.

derato fra i più robusti ed è ampiamente utilizzato. Le random forest non sono altro che l'applicazione nel Bagging di alberi di decisione non potati come algoritmo di apprendimento debole.

A questo punto sembrerebbe proprio trattarsi dell'esempio che è stato fatto poco fa. Tuttavia vi è una differenza che contraddistingue questo algoritmo di apprendimento: nel momento in cui viene creato l'albero di decisione non potato, come insieme delle feature utilizzate per guidare l'apprendimento di tale albero non si usano tutte quelle di cui si dispone, ma solo un sottoinsieme molto ridotto scelto casualmente. Empiricamente il dimensionamento di questo sottoinsieme F' viene valutato come $|F'| = \sqrt{|F|}$.

11.3 ADAPTIVE BOOSTING

Per avvicinarsi a questa secondo approccio all'ensemble learning si rende prima di tutto necessario un approfondimento dei concetti introdotti in precedenza e riguardanti l'apprendimento forte e l'apprendimento debole.

Si vuole quindi definire una classe di concetti come fortemente apprendibile se esiste un algoritmo che in tempo polinomiale ottenga un basso errore³ con una alta confidenza di tale risultato per tutti i concetti appartenenti a detta classe.

Un modello ottenuto attraverso un processo di apprendimento debole, al contrario, elimina la necessità di raggiungere una arbitraria accuratezza alta, accontentandosi di risultati poco migliori di scelte casuali, ergo con un errore strettamente minore di 0.5.

L'algoritmo di boosting, alla base della versione migliorata AdaBoost, permette di affermare che una classe di concetti C può essere appresa debolmente se e solo se questa può essere appresa fortemente.

In altre parole si può dire che dato un algoritmo di apprendimento A in grado di fornire una soluzione appena migliore di una scelta casuale per ogni istanza del problema, allora esiste un algoritmo di boosting in grado di fornire una soluzione accurata a piacimento che si basa proprio sull'algoritmo di apprendimento di base A. Inoltre il miglioramento della accuratezza cresce esponenzialmente rispetto al numero di volte che viene utilizzato A. Questo miglioramento esponenziale può essere visto nel grafico in Figura 11.5.

11.3.1 L'intuizione di AdaBoost

Nel momento in cui si utilizza l'algoritmo di apprendimento di base A, di tutti gli esempi presenti nel dataset alcuni saranno più facili e altri invece più difficili. Man mano che si utilizza tale algoritmo

³ Un errore basso a piacimento.

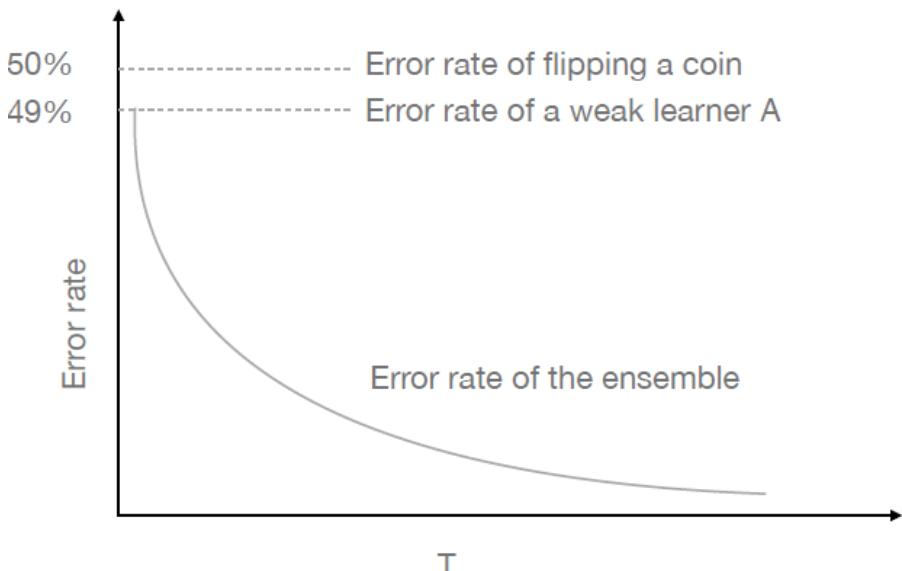


Figura 11.5: Grafico che mostra la diminuzione esponenziale dell'errore commesso in relazione al numero di volte T che si utilizza l'algoritmo di apprendimento debole A.

si individuerà una sacca di esempi su cui questo fa particolarmente fatica.

L'idea fondamentale che c'è dietro AdaBoost quindi consiste nell'assegnazione di pesi ad tutti gli esempi⁴ al fine di guidare l'algoritmo verso la correzione degli esempi classificati non correttamente. Seguendo questo approccio si cerca di fare in modo che tutti gli esempi vengano classificati correttamente da almeno uno dei modelli ottenuti con l'algoritmo di base A, con la speranza che poi unendo tutti questi modelli si riesca a minimizzare, se non addirittura azzerare, l'errore commesso.

Entrando nel dettaglio di tale idea, va specificato che:

- i pesi inizialmente sono uguali per tutti gli esempi (solitamente il valore di partenza è $\frac{1}{n}$);
- i pesi degli esempi vengono aggiornati in modo tale da tenere conto delle predizioni fatte, ovvero:
 - per esempi la cui predizione è corretta, il relativo peso viene diminuito;
 - al contrario per esempi la cui predizione non è corretta, il relativo peso viene aumentato.

Se l'algoritmo di apprendimento di base A è debole e ci si aspetta che commetta un errore strettamente minore di 0.5, a fronte dell'idea

⁴ Si noti come questo assegnamento di pesi modifichi la distribuzione originaria degli esempi, pertanto va contro l'assunzione fatta con le repliche di bootstrap dove invece si tendeva a mantenere la distribuzione del dataset iniziale.

appena presentata tale errore va calcolato con il dataset pesato, quindi dando importanza maggiore agli esempi con un peso più alto.

11.3.2 L'algoritmo AdaBoost

Di seguito viene riportato il codice dell'algoritmo AdaBoost:

Algorithm 24 AdaBoost(D, T, \mathcal{A})

```

1: Input: dataset  $D = (x_1, y_1), \dots, (x_n, y_n)$ ; dimensione dell'insieme  $T$ ;
   algoritmo di apprendimento di base  $\mathcal{A}$ .
2: Output: modello finale risultante dalla combinazione dei  $T$ 
   modelli ottenuti con l'algoritmo di base  $\mathcal{A}$ .
3:  $\mathbf{w}^1 = \left[ \frac{1}{|D|}, \dots, \frac{1}{|D|} \right]$ 
4: for  $t \in \{1, \dots, T\}$  do
5:    $m_t = \mathcal{A}(D, \mathbf{w}^t)$ 
6:    $\epsilon_t = \sum_{i=1}^n w_i^t I[y_i \neq m_t(x_i)]$ 
7:    $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$ 
8:   for  $i \in \{1, \dots, n\}$  do
9:      $w_i^{t+1} = w_i^t \exp(-\alpha_t y_i m_t(x_i))$ 
10:  end for
11:   $\mathbf{w}^{t+1} = \text{normalize}(\mathbf{w}^{t+1})$ 
12: end for
13: return  $M(x) = \text{sign}\left(\sum_{t=1}^T \alpha_t m_t(x)\right)$ 

```

A fronte dell'Algoritmo 24 si fanno presenti i seguenti punti per commentare lo pseudocodice di AdaBoost:

- le etichette y_n associate agli esempi facenti parte del dataset in input appartengono al dominio $\{-1, +1\}$;
- il vettore \mathbf{w}^1 contiene i pesi iniziali associati agli esempi normalizzati, pertanto valgono tutti $\frac{1}{|D|}$;
- si itera T volte per addestrare i T modelli con l'algoritmo di apprendimento di base, più precisamente:
 - l'algoritmo di apprendimento debole \mathcal{A} genera un modello m_t minimizzando l'errore pesato con il vettore dei pesi \mathbf{w}^t ;
 - viene calcolato ϵ_t , una misura dell'errore pesato commesso dal modello m_t appena addestrato⁵, come la somma dei soli pesi associati ad esempi classificati non correttamente⁶;

⁵ Poiché il modello ottenuto dall'addestramento debole deve commettere un errore strettamente minore di 0.5, allora ϵ_t sarà anch'esso strettamente minore di 0.5.

⁶ La funzione I restituisce 0 se per un certo esempio x_i l'etichetta y_i è uguale alla predizione del modello $m_t(x_i)$, 1 se invece la predizione non è corretta. Così facendo la sommatoria non tiene conto dei pesi degli esempi classificati correttamente perché tutti moltiplicati per 0.

- viene calcolato α_t , questo valore è il peso associato al t -esimo modello che verrà utilizzato quando alla fine si dovranno combinare tutti i T modelli in uno unico⁷;
- a questo punto vengono ripresati gli esempi del dataset, quindi per ogni peso w_i del vettore che verrà utilizzato alla successiva iterazione $t+1$ tale formula riduce i pesi degli esempi classificati correttamente ed aumenta quelli associati ad esempi la cui predizione era errata⁸;
- l'ultimo passo del ciclo è la normalizzazione ad 1 del vettore dei pesi appena aggiornati⁹;
- l'algoritmo AdaBoost restituisce un modello che prende il segno della combinazione di tutti i modelli m_t pesati con i pesi a loro associati α_t .

11.3.3 Correttezza di AdaBoost

L'algoritmo AdaBoost può essere visto come un grezzo minimizzatore di una funzione di penalizzazione esponenziale pesata dell'intero sistema ensemble. L'algoritmo seleziona in modo *greedy* ad ogni passo, ovvero ad ognuna delle $t \in T$ iterazioni, le ipotesi deboli e i pesi α_t che hanno proprio lo scopo di minimizzare la funzione di penalizzazione esponenziale di M_t .

Si assuma ora di aver già costruito una parte dei modelli che compongono l'ensemble, l'insieme. Si è quindi al punto in cui è stato aggiunto il penultimo modello, il $T-1$ -esimo:

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

⁷ Si può facilmente dimostrare che α_t è un valore strettamente maggiore di 1: poiché $\epsilon_t < 0.5$, si ha nella frazione all'interno del logaritmo un numeratore $N > 0.5$ ed un denominatore $D < 0.5$, pertanto il rapporto risulta essere necessariamente strettamente maggiore di 1.

⁸ Analizzando con attenzione la formula con cui viene svolta questa operazione di ricalcolo dei pesi se ne può cogliere il senso. Se l'esempio x_i è stato classificato correttamente il prodotto $y_i m_t(x_i)$ è positivo perché i due fattori sono concordi in segno (entrambi positivi o entrambi negativi), tuttavia la moltiplicazione con il termine $-\alpha_t$ rende il risultato negativo. Se l'argomento della funzione esponenziale è negativo allora il suo risultato è un valore compreso fra 0 e 1, pertanto moltiplicando il peso attuale per un valore minore di 1 si ottiene l'effetto di ridurlo. Al contrario per gli esempi classificati male l'argomento della funzione esponenziale è positivo (poiché i segni del risultato del modello e del reale output sono concordi) risultando in un valore maggiore di 1 che moltiplicato per il peso dell'esempio avrà l'effetto di incrementarlo.

⁹ Questo passo è fondamentale per mantenere ad ogni iterazione i pesi nell'intervallo compreso fra 0 e 1.

Bisogna a questo punto focalizzare l'attenzione sulla scelta di m_T e α_T . In particolare l'errore commesso dall'ultimo modello $M_T(x) = M_{T-1}(x) + \alpha_T m_T(x)$ nella funzione di penalizzazione esponenziale:

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

con:

$$w_i^T = \begin{cases} e^{-y_i M_{T-1}(x_i)} & \text{se } T > 1 \\ 1 & \text{se } T = 1 \end{cases} \quad (11.2)$$

Poiché si utilizza una funzione di penalizzazione esponenziale, l'errore totale è calcolato come la somma dei singoli errori, calcolati a loro volta con $e^{-y_i M_T(x_i)}$. Come già spiegato nel dettaglio in precedenza se per un certo esempio x_i la predizione è corretta, ovvero i due fattori y_i e $M_T(x_i)$ sono concordi in segno, allora l'esponente di e sarà negativo (positivo invertito di segno), quindi il risultato della funzione esponenziale sarà un valore compreso fra 0 e 1 per una penalizzazione molto bassa. Al contrario se la classificazione non è corretta l'esponente è positivo (negativo invertito di segno), e la penalizzazione che viene assegnata con la funzione esponenziale diventa maggiore di 1 e cresce esponenzialmente.

Nel secondo passaggio viene espanso l'esponente applicando la definizione fornita poc'anzi di $M_T(x_i)$, e con un ulteriore passaggio viene distribuito il termine y_i sui due addendi all'interno delle parentesi con i quali è moltiplicato.

Nel quarto passaggio si applica una semplice proprietà delle potenze che permette di isolare uno dei due termini, $e^{-y_i M_{T-1}(x_i)}$, che nel passaggio successivo può quindi essere sostituito con w_i^T secondo la definizione data con la Formula 11.2.

Ma ora che è stata tirata in ballo la Formula 11.2 ci si può accorgere che questa differisce da quella applicata nell’Algoritmo 24, dove invece si legge:

$$w_i^{t+1} = w_i^t \exp(-\alpha_t y_i m_t(x_i))$$

Tale differenza in realtà è solo apparente, perché si può dimostrare che queste due formulazioni sono effettivamente del tutto equivalenti. Pertanto partendo dalla Formula 11.2 si potrà arrivare alla versione utilizzata nell’Algoritmo 24:

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

Dopo questa precisazione si può tornare al sesto passaggio del calcolo dell’errore. A questo punto vengono separati gli esempi classificati correttamente nella prima sommatoria, da quelli invece classificati non correttamente nella seconda sommatoria. In particolare si noti come vengono semplificati gli esponenti:

- considerando nella prima sommatoria solo gli esempi che sono stati classificati correttamente, il prodotto dei termini y_i e $m_T(x_i)$ ¹⁰ è +1, pertanto si possono omettere e mantenere solo il termine $-\alpha_T$;
- considerando nella seconda sommatoria solo gli esempi che sono stati classificati non correttamente, il prodotto dei termini y_i e $m_T(x_i)$ è -1, pertanto si possono omettere e invertire di segno il termine α_T .

Con un ulteriore passaggio, il settimo, si aggiunge e si sottrae una identica quantità con l’obbiettivo di ottenere una formulazione che renda più facile vedere come minimizzare la funzione esponenziale dell’errore. Ponendosi questo obiettivo si intende riscrivere il primo termine (si la prima sommatoria) senza che questo dipenda dalla correttezza dell’attuale classificatore.

L’ottavo passaggio vede solo riordinare gli elementi in vista del nono passaggio nel quale le prime due sommatorie vengono accorpate: l’argomento è identico, gli indici sono complementari¹¹. Anche le altre due sommatorie vengono accorpate in una unica. Queste hanno in comune lo stesso indice che tiene in considerazione solo gli esempi la cui predizione è sbagliata, ma poiché l’argomento differisce per un segno nell’esponente da una all’altra, si può solo fare un raccoglimento.

¹⁰ Si ricorda che la classe positiva corrisponde a +1 e la classe negativa a -1.

¹¹ La prima considera gli esempi che sono stati classificati correttamente, la seconda quelli classificati non correttamente.

Con questa formulazione la prima delle due sommatorie è un termine costante che non dipende più dalla correttezza o meno degli esempi perché li considera tutti, mentre la seconda sommatoria è dipendente dagli esempi classificati male dal modello. Proprio questo secondo termine si rivela indicativo di come procede l'ottimizzazione e diventerà quindi oggetto della minimizzazione dell'errore.

Nell'ottica delle minimizzazioni dell'errore risulta quindi chiaro come il peso α_T diventi il fattore da ottimizzare, ovvero quello che annulla la seguente derivata¹²:

$$\frac{\partial E}{\partial \alpha_T} = \sum_{\{i|y_i \neq m_t(x_i)\}} w_i^T e^{\alpha_T} - \sum_{\{i|y_i = m_t(x_i)\}} w_i^T e^{-\alpha_T} \quad (11.3)$$

Si riprenda la definizione data nell'Algoritmo 24¹³ per la somma di tutti i pesi degli esempi classificati non correttamente:

$$\epsilon_T = \sum_{\{i|y_i \neq m_t(x_i)\}} w_i^T \quad (11.4)$$

Punti a questo giunto si impone che la derivata calcolata nella Formula 11.3 sia uguale a 0, pertanto si possono svolgere i seguenti passaggi per ottenere il termine α_T :

$$\begin{aligned} & \left(e^{\alpha_T} \sum_{\{i|y_i \neq m_t(x_i)\}} w_i^T \right) - \left(e^{-\alpha_T} \sum_{\{i|y_i = m_t(x_i)\}} w_i^T \right) = 0 \\ & e^{\alpha_T} \epsilon_T - e^{-\alpha_T} (1 - \epsilon_T) = 0 \\ & e^{\alpha_T} \epsilon_T = e^{-\alpha_T} (1 - \epsilon_T) \\ & \frac{e^{\alpha_T}}{e^{-\alpha_T}} \frac{\epsilon_T}{\epsilon_T} = \frac{e^{-\alpha_T}}{e^{-\alpha_T}} \frac{1 - \epsilon_T}{\epsilon_T} \\ & e^{\alpha_T - (-\alpha_T)} = \frac{1 - \epsilon_T}{\epsilon_T} \\ & e^{2\alpha_T} = \frac{1 - \epsilon_T}{\epsilon_T} \\ & 2\alpha_T = \ln \left(\frac{1 - \epsilon_T}{\epsilon_T} \right) \\ & \alpha_T = \frac{1}{2} \ln \left(\frac{1 - \epsilon_T}{\epsilon_T} \right) \end{aligned} \quad (11.5)$$

Di nuovo si ripercorrono i tre passaggi appena svolti:

¹² Ottenuta dalla derivazione della formulazione al sesto passaggio di due pagine fa.

¹³ Si noti che nell'algoritmo si usa una funzione indicatrice I , mentre ora il suo ruolo viene sostituito dall'indice della sommatoria che considera direttamente i soli esempi di interesse, ovvero quelli la cui classificazione non è corretta.

- nel primo passaggio è stata riportata la Formula 11.3 posta uguale a 0, perché trovando lo zero della derivata si trova il minimo della funzione, inoltre l'esponenziale è stato portato all'esterno della sommatoria in ambedue i termini;
- nel secondo passaggio si applica la sostituzione delle due sommatorie come da Formula 11.4, ricordando che i pesi w_i^T sono normalizzati ad 1, pertanto se la somma dei pesi associati agli esempi classificati male è proprio ϵ_T allora la somma dei pesi corrispondenti agli esempi classificati correttamente è esattamente il complementare $1 - \epsilon_T$;
- nel terzo passaggio viene semplicemente spostato il secondo esponenziale a destra dell'uguale;
- nel quarto passaggio si divide da ambedue le parti per $e^{-\alpha_T}$ e per ϵ_T , questo per permettere due importanti semplificazioni;
- nel quinto passaggio si può apprezzare il risultato delle semplificazioni, inoltre il rapporto fra i due esponenziali a sinistra dell'equazione è stato riscritto in forma contratta con la differenza fra esponenti;
- nel sesto passaggio si è semplicemente svolta la somma nell'esponente della funzione esponenziale;
- nel settimo passaggio viene introdotto il logaritmo per isolare l'esponente della funzione esponenziale;
- nell'ottavo ed ultimo passaggio si divide semplicemente per 2 da entrambi i lati dell'equazione ottenendo la soluzione per il termine di interesse α_T , la Formula 11.5 corrisponde esattamente a quella utilizzata nell'Algoritmo 24.

L'unico aspetto dell'Algoritmo 24 che non verrà approfondito è l'aggiornamento dei pesi con la formula alla riga 9 all'interno del secondo ciclo for.

Si vuole inoltre specificare che non è scontato che tutti gli algoritmi di apprendimento debole che potenzialmente si potrebbero utilizzare siano in grado di gestire un dataset pesato come richiesto (sempre all'interno dell'Algoritmo 24, riga 5), tuttavia il problema è limitato poiché esistono delle soluzioni generalmente abbastanza semplici per i diversi tipi di algoritmi di apprendimento. Tuttavia ci possono essere talvolta dei casi meno semplici in cui si deve ricorrere alla tecnica del *dataset resampling*.

11.3.3.1 *Dataset resampling*

La tecnica del *dataset resampling* si basa su un'idea abbastanza semplice: del dataset originale viene fatta una copia dove gli esempi vengono ripetuti in proporzione al loro peso w associato. Questa operazione

viene svolta ogni volta che vengono aggiornati i pesi dall'algoritmo AdaBoost.

Di seguito viene riportato il codice dell'algoritmo che genera la copia del dataset.

Algorithm 25 Resample(D, \mathbf{w}, n)

```

1:  $D' = \emptyset$ 
2: for  $i \in \{1, \dots, n\}$  do
3:    $v = \text{random}()$   $\triangleright$  Estrae a caso un numero compreso fra 0 e 1
4:    $D' = D' \cup \{x_k\} \mid \sum_{i=1}^{k-1} w_i < v \leq \sum_{i=1}^k w_i$ 
5: end for
6: return  $D'$ 
```

Il fulcro di tale implementazione è nelle righe 3 e 4, ovvero il ciclo for, il quale per un certo numero di volte n^{14} definito a monte e passato per parametro inserisce con una probabilità data dal peso di ciascun esempio detto esempio nel nuovo dataset. Con un numero sufficientemente alto di iterazioni è possibile ottenere una distribuzione del nuovo dataset molto simile a quella del dataset originario.

Particolare attenzione va posta sulla scelta di n , perché questo determina:

- la dimensione del nuovo dataset, poiché se si vuole costruire dataset molto più grandi sarà necessario più tempo e più spazio;
- più il nuovo dataset è grande, maggiore sarà lo sforzo computazionale dell'algoritmo di apprendimento ensemble per utilizzarlo nella sua interezza;
- maggiore è n , maggiore è la precisione con cui il nuovo dataset rispecchia la distribuzione determinata dai pesi.

Alla luce di queste osservazioni si tende a scegliere di creare nuovi dataset di dimensioni analoghe a quello di partenza, pertanto si ha $|D'| = |D|$. Questo approccio si rivela ragionevolmente soddisfacente nonostante possa escludere gli esempi che hanno associati dei pesi particolarmente piccoli, tuttavia sono di minore importanza rispetto a quelli con pesi più alti, per i quali invece è certa una buona presenza.

In conclusione si può affermare che questo approccio è computazionalmente più oneroso rispetto all'adattamento degli algoritmi di apprendimento per aggiornare i pesi, pertanto va utilizzato solo quando si ha la certezza che possa risultare più vantaggioso, oppure quando effettivamente non si può utilizzare il primo approccio.

¹⁴ Da non confondersi con il numero n di istanze che compongono il dataset.

11.3.4 Come fa a funzionare 'sta roba

Per capire come fanno gli algoritmi di apprendimento ensemble a funzionare si possono addurre diverse spiegazioni di carattere euristico che vengono ivi riassunte:

- decomposizione bias-variance;
- benefici di carattere statistico;
- benefici nelle rappresentazioni dei concetti;
- benefici computazionali.

In riferimento ai concetti legati al problema bias-variance già presentato nella sottosezione 3.2.2, gli algoritmi di apprendimento ensemble risultano efficaci nella riduzione sia del bias, che della varianza. In particolare grazie a prove di carattere empirico il Bagging risulta più efficace nella riduzione della varianza che del bias, pertanto può essere un approccio molto efficace con classificatori come gli alberi di decisione che soffrono particolarmente di tale problema.

Per capire come fa il Bagging ad essere così efficace contro la varianza si può pensare alla distribuzione di probabilità della variabile random che conta il numero di errori commessi dai modelli base dell'ensemble. Ci si focalizza su un singolo esempio e si va a verificare se i singoli classificatori che vanno successivamente a comporre il modello finale commettono o meno un errore nell'attribuzione della classe corretta a detto esempio.

Si decide di chiamare X la variabile stocastica che conta il numero di errori commessi dai T modelli che compongono l'ensemble, e si ricorda inoltre l'assunzione per cui l'algoritmo di apprendimento debole deve necessariamente essere migliore di un modello random, pertanto il suo errore p deve essere strettamente minore di 0.5 (ricordando che si deve collocare questo discorso sempre in termini di classificazione binaria).

Alla luce di queste ultime due osservazioni si può affermare che il numero di errori X si distribuisce come una binomiale¹⁵ con parametri p e T :

$$X \sim Bin(p, T)$$

Ora si vuole vedere con che probabilità l'intero ensemble che utilizza il Bagging commette un errore, ovvero quando la maggioranza dei modelli che lo compone classifica erroneamente:

$$P(X > \lfloor \frac{T}{2} \rfloor) = \sum_{x=\lfloor \frac{T}{2} \rfloor + 1}^T P(X = x) \quad (11.6)$$

¹⁵ Tale distribuzione binomiale può essere vista come una somma di bernulliane.

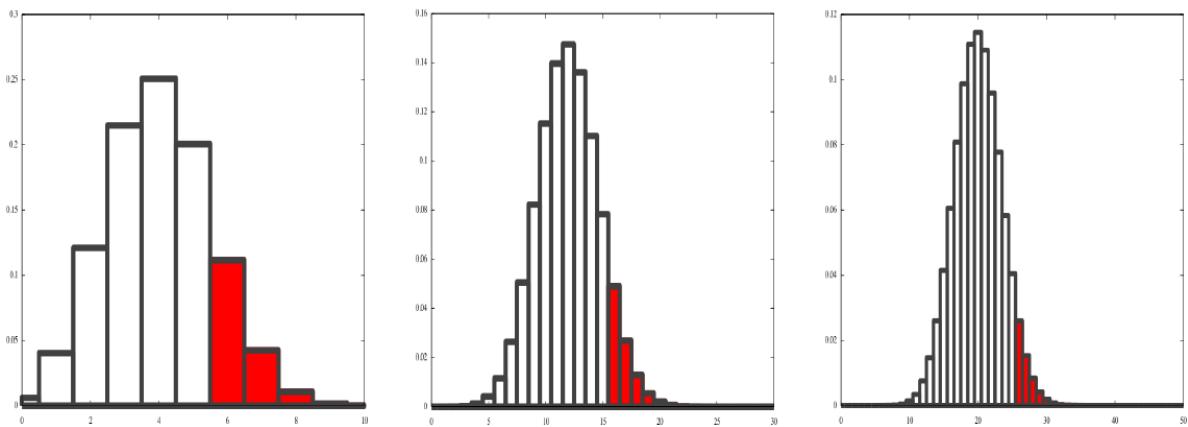


Figura 11.6: Tre grafici rappresentanti la distribuzione binomiale dei modelli rispetto all'errore di classificazione che commettono. Sull'asse delle ordinate vi è la probabilità di commettere un errore, sull'asse delle ascisse il numero dei classificatori (nei tre grafici, da sinistra a destra, rispettivamente 10, 20 e 30). All'aumentare del numero T di modelli utilizzati, diminuisce la probabilità che l'insieme di tutti questi modelli commetta un errore (area evidenziata in rosso).

La Formula 11.6 quindi somma solo da $\lfloor \frac{T}{2} \rfloor + 1$ a T le probabilità P che la variabile aleatoria X sia proprio uguale ad x . Tale sommatoria tende a 0 quando T tende ad infinito, questo si può vedere graficamente nella Figura 11.6. Sull'asse delle ascisse è riportato il numero T di classificatori utilizzati, che aumenta da sinistra a destra nei tre diversi grafici riportati, mentre sull'asse delle ordinate rimane fissa la probabilità P . Nei tre grafici si nota come all'aumentare del valore T la campana si abbassa¹⁶ e si stringe, pertanto la porzione evidenziata in rosso della campana stessa che ricade nella metà destra del grafico è sempre più piccola. Questa porzione rossa corrisponde esattamente alla sommatoria della Formula 11.6 e quando T diventa infinito la sommatoria diventa una delta di Dirac¹⁷.

Questo è uno dei motivi per cui l'apprendimento ensemble funziona bene: con un maggior numero di modelli a disposizione diventa meno probabile che la maggioranza di questi si sbagli nella classificazione.

Se l'algoritmo Bagging risulta quindi essere più portato alla riduzione della varianza piuttosto che del bias, AdaBoost invece si rivela portato per entrambi, anche in questo caso a fronte di prove di carattere empirico. Più precisamente è particolarmente efficace su classificatori con alto bias come ad esempio i modelli lineari, che sono molto semplici.

Rimangono a questo punto da analizzare gli altri tre benefici introdotti nell'elenco all'inizio di questa sottosezione, cominciando da

¹⁶ Attenzione alla scala riportata sull'asse delle ordinate!

¹⁷ Per approfondimenti matematici sulla funzione delta di Dirac rivolgersi a Wikipedia.

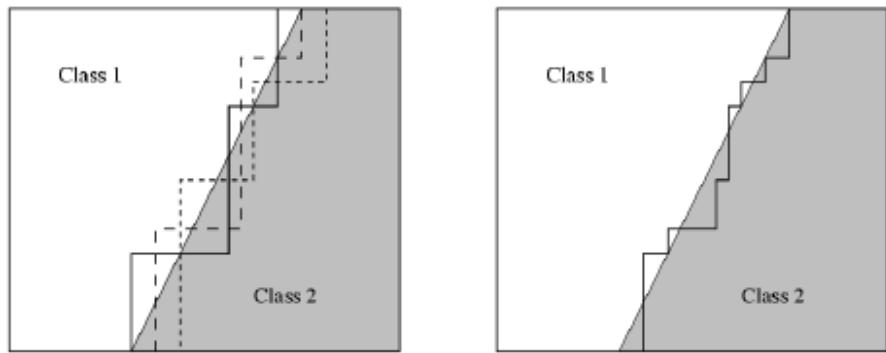


Figura 11.7: A sinistra si possono distinguere tre diversi modelli che cercano di classificare le istanze divise nelle due classi, mentre a destra c'è il modello finale ottenuto mediando i risultati dei tre diversi modelli.

quelli di tipo statistico.

I benefici statistici fanno riferimento ad una approssimazione del classificatore ottimo di Bayes¹⁸: si prendono un certo numero di classificatori "buoni" e si fa una media dei loro risultati. Questo è proprio ciò che viene fatto dagli algoritmi di apprendimento ensemble.

I benefici derivanti dalla rappresentazione dei concetti invece sono già stati introdotti e si possono facilmente intuire ponendo l'attenzione sulla Figura 11.7. Il modello a destra, ottenuto mediando i tre diversi modelli a sinistra, è evidentemente più preciso nel tentativo di classificare le due classi di istanze. Infatti i singoli modelli non sono molto precisi, perché è difficile approssimare un decision boundary obliqui con una funzione a gradini, però combinando i risultati di tante funzioni di questo tipo è possibile approssimare sempre meglio il risultato che si vorrebbe raggiungere.

Infine la motivazione computazionale. Sovente gli algoritmi di apprendimento automatico possono essere visti come algoritmi di ricerca di una approssimazione in uno spazio delle ipotesi, spazio che può essere anche molto complesso e non convesso, per cui possono essere presenti minimi locali nei quali ci si può imbattere durante la sua esplorazione. Utilizzando più algoritmi in questo spazio è possibile trovare diverse soluzioni, diversi minimi, e scegliere quella ottima per ottenere una soluzione finale che sia migliore, ovviamente, delle singole.

¹⁸ Il classificatore ottimo di Bayes combina tutti i classificatori possibili e li pesa con la probabilità a posteriori di ciascuno di essi, ha delle performance che si dimostrano ottime, non si può mai ottenere un classificatore migliore di questo (pare ci sia pure un teorema che lo dice). Unico problema: non è computabile.

12

LA MISURAZIONE DEGLI ESPERIMENTI

In questo capitolo si introdurranno concetti utili a progettare e organizzare esperimenti utili al fine di misurare la bontà dei modelli predittivi introdotti precedentemente.

Molto spesso si ha infatti la necessità di misurare, all'introduzione di un nuovo algoritmo, la bontà di quest'ultimo rispetto a quello usato in precedenza (in particolare si vuole dimostrare che le migliorie ottenute siano significative).

Esistono due strade per ottenere questo tipo di risultato: la prima strada è quella teorica ed è quella introdotta nel Capitolo 9, con cui si vuole confrontare l'algoritmo con uno ideale e dimostrarne quindi la tendenza all'ottimo, mentre la seconda è quella presentata di seguito e più sperimentale.

Si tratta di lanciare lo stesso esperimento con i due (o più) algoritmi che si vogliono confrontare e verificarne i risultati di performance.

12.1 ACCURATEZZA ATTESA

Nel Capitolo 1 abbiamo introdotto le tabelle di contingenza, ovvero tavole che riportano per riga gli esempi veramente positivi e veramente positive e per colonna le rispettive predizioni. Riportiamo di seguito un esempio.

Quale misura si può utilizzare per confrontare questo modello con

	<i>Predicted</i> \oplus	<i>Predicted</i> \ominus	
<i>Actual</i> \oplus	60	20	80
<i>Actual</i> \ominus	0	20	20
	60	40	100

Figura 12.1: Esempio di Tabella di Contigenza

altri? Al fine di tracciare la curva roc abbiamo visto che si considera il true positive rate nell'asse delle ordinate, ovvero $60/80 = 0.75$, mentre sull'asse delle ascisse avremo il true negative rate, ovvero $20/20$. L'accuratezza è invece data dal rapporto tra i valori predetti correttamente e la totalità degli esempi, ovvero $(60 + 20)/100$. Ma sono misurazioni attendibili? In realtà, il dataset ha una prevalenza di esempi positivi e quindi la alta capacità di riconoscere esempi negativi è anacqua dal numero esiguo. Se gli esempi negativi e positivi fossero distribuiti uniformemente la media tra il tpr e il tnr (ovvero l'*average recall* sarebbe una misurazione attendibile (0,88).

Essendoci una prevalenza di esempi una classe, l' *average recall* è leggermente fuorviante e occorrerebbe tenere conto di questa discrepanza nel dataset. Tuttavia questo non si può fare in modo così semplice. Il dubbio naturale che ci si pone è quindi se è meglio considerare più attendibile l'*average recall* oppure l'*accuracy*. La risposta è che l'*accuracy* dovrebbe essere usata (nonostante tende a favorire la classe predominante) se ci si aspetta una distribuzione affidabile per il dominio che si considera e non sono per il test set del caso particolare che stiamo trattando (nel caso di dataset di campioni di persone, il numero di malati di slà è verosimilmente in netta minoranza).

Se non sappiamo nulla circa la frequenza degli esempi nelle altre istanze del dominio, calcolando l'*average recall* fa una media senza soppesare in alcun modo la maggior o minor presenza di esempi nelle due classi potrebbe essere la soluzione migliore in quanto la misura predittiva è idealmente più stabile.

Supponiamo di avere due classificatori costruiti sullo stesso dataset e che si comportano l'uno prediligendo l'*average recall* come misura maggiore in valore assoluto e la seconda prediligendo invece l'*accuracy* in quanto ha un forte squilibrio nel numero di esempi positivi. Introducendo ora un caso in cui abbiamo ancora un dataset in cui il (forte) sbilanciamento è a favore del numero di esempi della classe negativa, ed in particolare sale vertiginosamente il numero di true negative. Abbiamo già precedentemente introdotto i concetti di *precision*, cioè la capacità del classificatore di classificare correttamente l'insieme degli esempi positivi, e la *recall*, che considera solo la riga dei true positivi e misura quanto sia corretto il classificatore nel "richiamare" come positivi gli esempi tali.

Sia la precisione che la *recall* non considerano quindi gli esempi correttamente predetti negativi, che in questo caso abbiamo visto essere però particolarmente numerosi.

La F-measures è la media armonica tra queste ultime due misure, tenendole ugualmente in conto.

Si nota in generale che è molto difficile per un classificatore ottenere una buona precisione e contemporaneamente una buona *recall*. I classificatori sono spesso, in tal senso, polarizzati verso una delle due misure in quanto se vuole tenere alta la precisione tenterà a voler predire talmente tanti esempi come positivi da sbagliare e predirne alcuni come tali quando non lo sono, oppure per mantenere la *recall* alta richiamando molti esempi positivi sbaglia abbassando la precisione.

Quindi la F-measure riesce a considerare contemporaneamente questi due aspetti, a scapito di dar merito all'algoritmo di classificare correttamente gli esempi negativi (è "polarizzata" verso i positivi).

Se volessimo invece valutare la capacità di un algoritmo di fare ranking corretto, quindi di ordinare gli esempi predicendone la classe di appartenenza occorrono altre misure. Consideriamo il rank in figu-

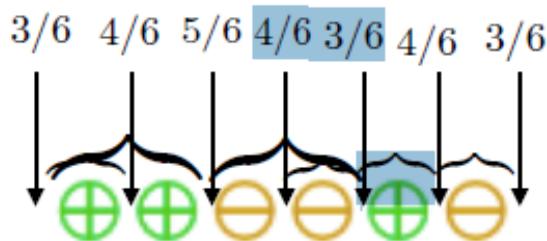


Figura 12.2: Example of ranking

ra 12.2 in cui prima vengono predetti due esempi positivi, poi due esempi negativi, poi ancora un esempio positivo ed infine ancora un esempio negativo.

In questo caso il ranker commette due errori in quanto ciascuno degli esempi predetti negativi vengono predetti prima del terzo esempio positivo (il terzo esempio negativo non provoca invece nessun errore). Quindi, su un totale di 9 possibili posizioni dei 3 esempi per entrambe le due classi, avrà un AUC (Area Sotto La Curva che afferma la bontà della predizione del ranker) di 7/9. Introduciamo ora delle relazioni rispetto alla capacità del ranker, per ciascuna delle possibili alternative, di avere un certo rate di predizione della classe positiva. Dato il ranking, vi sono diverse possibilità per il classificatore di agire. Potrebbe ad esempio fissare la soglia sul primo esempio, considerando come negativi tutti i successivi e in tal caso otterrebbe un accuratezza di 3/6 e un tpr è 0 poiché nessun esempio positivo viene predetto. Se invece la soglia venisse spostata, l'accuratezza salirebbe a 4/6 così come anche il tpr sarebbe uguale ad 1/6 e spostando ancora si ottiene l'accuratezza a 5/6 e un tpr di 2/6 e così via.

Esiste intuitivamente un legame tra l'accuratezza e la AUC. L'accuratezza aspettata da tutte le possibili soglie è

$$(3 + 4 + 5 + 4 + 3 + 4 + 3) / (6 \cdot 7) = 26/42$$

Questo legame è data dalla formula

$$\text{Accuracy} = \frac{n}{n+1} \cdot \frac{2\text{AUC} - 1}{4} + \frac{1}{2} = (\text{nel nostro caso}) \frac{5}{42} + \frac{1}{2} = \frac{26}{42}$$

Quindi riusciamo ad ottenere una valutazione (pesata sulla sua probabilità) dell'accuratezza grazie all'AUC.

Quindi in generale:

- Abbiamo già detto come l'accuratezza sia una misura attendibile qualora il nostro dataset è rappresentativo del caso più generale
- L'average recall è in vece adatta qualora non conosciamo nulla riguardo il caso più generale, che facendo una media ugualmente pesata è più adatta quando alle classi vogliamo attribuire uguale peso.

- Precision e Recall funzionano bene quando vogliamo ignorare i valori classificati correttamente negativi
- Il TPR e AUC (collegata all'accuracy aspettata come visto) sono utili se siamo interessati in un contesto di ranking

12.2 COME EFFETTUARE LE MISURAZIONI

Abbiamo finora visto che non si può prendere una misura e usarla a prescindere, ma esiste in realtà una pluralità di misure e occorre valutare quale sia la migliore da considerare per il nostro scopo. Ma come valutare le misure di performance viste? Come progettare i nostri esperimenti?

È chiaro che qualunque misura è soggetta ad errore e quindi, effettuate diverse misurazioni, possiamo ottenere e discutere la varianza (σ^2) delle misurazioni e ottenere così il valore medio dell'insieme di misurazioni, che sarà sicuramente più stabile rispetto alla successione delle semplici misure (poichè soggette ad errore).

Su queste, supponiamo k , misure, il calcolo del valor medio ci consente di abbassare la varianza di un fattore k poichè per un k sufficientemente alto la varianza della stima si aspetta essere più bassa rispetto alla varianza σ^2/k , infatti con n campioni indipendenti presi da quei k , dalla *legge dei grandi numeri*

$$\text{Var}(\bar{X}_n) = \text{Var}\left(\frac{1}{n}(X_1 + \dots + X_n)\right) = (\text{applicheremo qui le proprietà della varianza}) \frac{1}{n^2} V$$

Ed ecco che ci accorgiamo che la varianza campionaria è più bassa rispetto alla varianza della popolazione (sotto l'assunzione che tutti i campioni siano indipendenti).

Trasponiamo allora l'esempio sulle misure della accuratezza di un classificatore che considera molti esempi. Abbiamo allora un dataset formato da n esempi e vogliamo capire se il classificatore predirà correttamente su questi esempi tutti indipendenti. Essendo tali gli esempi, la correttezza della predizione è una variabile aleatoria, rappresentabile con una distribuzione bernoulliana (variabile binaria casuale con probabilità di successo/fallimento incognita). Consideriamo come successo il caso in cui il classificatore predica correttamente sull'esempio. Avendo n esempi la miglior stima è data dal rapporto $\hat{a} = A/n$ dove A è il numero di esempi classificati correttamente. Essendo nel caso di una variabile bernoulliana, la varianza è data da $a(1 - a)$. Se mediamo rispetto a tutte le possibili istanze n la varianza (della stima di accuratezza che è incognita) sarà intuitivamente $a(1 - a)/n$. Siccome abbiamo visto come la varianza si riduce proporzionalmente all'aumentare dei campioni, quello che vogliamo fare al fine di migliorare la nostra valutazione è ripetere la stima dell'accuratezza tante volte (ovvero per k campioni indipendenti) e la varianza

così risultante sarà più stretta di quella originaria. In tale ottica la formula sarà (dividendo per $k-1$ invece di k perchè il campione è ridotto, solitamente 10):

$$\frac{1}{k-1} \sum_k^{i=1} (\hat{a}_i - \bar{a})^2, \text{ con } \bar{a} = \frac{1}{k} \sum_k^{i=1} \hat{a}_i = \text{valor medio delle stime}$$

Purtroppo però, spesso non abbiamo un dataset grande a sufficienza per selezionare al suo interno k partizioni ciascuna con un numero discreto di esempi indipendenti. In questi casi occorre applicare la procedura della *Cross-Validation*: si formano k partizioni random di cui una viene usata come test set indipendente, su cui testare il modello, e le rimanenti $k-1$ saranno i training set per produrre il modello. I training set non sono realmente indipendenti poichè condivino $k-2$ partizioni, ma il test set è realmente indipendente rispetto al training set considerato (e i rispettivi esempi vengono quindi mantenuti separati). Così facendo stiamo praticamente calcolando la bontà di generazione del modello da parte dell'unico algoritmo di induzione usato durante le singole fasi di training (che produrranno appunto modelli diversi fra loro).

Il suggerimento è quindi quello di utilizzare la Cross Validation per valutare diversi modelli e per decidere eventuali iper parametri, e poi dopo i vari tentativi selezionare i tratti distintivi migliori (iper parametri appunto) per costruire un unico modello valido su tutto il training set, la cui accuratezza sarà verosimilmente quella ottenuta mediando sulle k partizioni durante la fase di Cross-Validation.

Per decidere il numero di k partizioni bisogna considerare che ognuno deve contenere un numero sufficiente di esempi, poichè altrimenti la legge dei grandi numeri che ci permette di effettuare un numero sufficiente di stime indipendenti non vale più.

Se non si hanno un numero adatto di esempi (centinaia) si può sfruttare il caso estremo della cross validation *leave-one-out* (o statistica jackknife) in cui il numero delle partizioni viene messo uguale al numero degli esempi, avendo a questo punto $n-1$ esempi in ciascuna delle $k-1$ partizioni e un esempio singolo che funge da test set.

Ogni volta che stimiamo l'accuratezza avremo uno o un 1 (l'unico valore di verità possibile) che verrà poi mediato sulle k ($=n$) partizioni, rendendo particolarmente applicabile questa tipologia di cross validation quando supponiamo una distribuzione normale.

Nel caso in cui il dataset sia invece sbilanciato per qualcuna delle classi si può pensare di applicare la cross validation *stratificata* in cui la selezione degli esempi per ciascuna delle partizioni è casuale ma il numero contenuto è rappresentativo del dominio (altrimenti si rischierebbe di avere partizioni nulla o comunque non significative).

Fold	Naive Bayes	Decision tree	Nearest neighbour
1	0.6809	0.7524	0.7164
2	0.7017	0.8964	0.8883
3	0.7012	0.6803	0.8410
4	0.6913	0.9102	0.6825
5	0.6333	0.7758	0.7599
6	0.6415	0.8154	0.8479
7	0.7216	0.6224	0.7012
8	0.7214	0.7585	0.4959
9	0.6578	0.9380	0.9279
10	0.7865	0.7524	0.7455
avg	0.6937	0.7902	0.7606
stdev	0.0448	0.1014	0.1248

Figura 12.3: Esempi di dati estrapolati dall'applicazione di tre algoritmi di apprendimento

12.3 ESEMPI DI CROSS VALIDATION

Abbiamo una tabella rappresentante i risultati di tre algoritmi e supponiamo di applicare una *10-folds* cross validation.

Le due ultime righe della tabella indicano la media delle accuratezze ottenute per ciascun algoritmo e la deviazione standard (radice quadrata della varianza).

Guardando la tabella potremmo concludere che il Decision Tree vada meglio degli altri, seguito da nearest neighbour e infine Naive Bayes. Tuttavia, sebbene l'accuracy sia inferiore, la deviazione standar è più piccola per Naive Bayes rispetto ad esempio a Nearest Neighbour (non ci sorprende dato che una predizione basata solo sui vicini può essere deviante).

12.4 L'INTERPRETAZIONE DELLE MISURAZIONI

Non sappiamo quindi fino a che punto dei dati di questo tipo possono essere esaustivi per trarre le conclusioni a cui siamo interessati. Occorre più che altro effettuare analisi statistiche al fine di confermare o meno che le differenze osservate siano realistiche o meno.

Quello che facciamo è riprendere la teoria degli intervalli di confidenza, in particolare prenderemo una stima \hat{a} ottenuta dalla cross-validation e ne studieremo la distribuzione rispetto ad un ipotetico valore a noi sconosciuto ma che andremo ad approssima calcolando la deviazione standard σ calcolata rispetto alla formula della binomiale vista nella sezione precedente.

Dagli intervalli di confidenza sappiamo che supponendo di conoscere la stima dell'accuracy e la deviazione standard σ , andiamo a vedere l'attendibilità della stima tramite la sua appartiene ad un in-

tervallo di confidenza.

Per esempio possiamo andare a stimare la verosomiglianza (la *likelihood*) che una stima ricava intorno ad un certo valor medio con una ampiezza dell'intervallo pari un certo numero di volte la deviazione standard. Nel nostro esempio, con una deviazione standard pari a ± 1 otterremmo un 68%, che indica che 68 volte su 100 le stime ricadono in quell'intervallo. Ampliando l'intervallo (e accettando quindi una minor accuratezza nella stima) ci aspettiamo che il numero di stime incluse cresca ed infatti con un intervallo pari a due volte la deviazione standard il *livello di confidenza* sale al 95%.

Se il numero degli esempi è sufficientemente elevato possiamo approssimare graficamente la binomiale con una funzione gaussiana costruita attorno a un certo valore medio a dell'accuratezza (incognita) con una certa deviazione standard σ . Se calcoliamo l'integrale intorno al valor medio e considerando una ampiezza di σ l'area sotto la curva ci darà la likelihood (che abbiamo appena visto essere la probabilità con cui una stima ridice in quella zona), mentre, essendo l'integrale sull'intera curva massimo uguale ad 1 (poichè è una probabilità) si ottiene che la probabilità che la stima cada al di fuori dell'area dell'intervallo di confidenza è $1 - C$, ovvero $\frac{1-C}{2}$ a sinistra e a destra di a . Ampliando σ , il valore $\frac{1-C}{2}$ scende: l'obiettivo per uno studio coerente è proprio fissare un sigma corretto.

Mentre nell'immagine ?? a destra troviamo raffigurato quanto abbiamo appena provato a riassumere, nell'immagine a destra troviamo i valori della confidenza per gli intervalli più comuni. In particolare il 95,4% di confidenza è già un buon risultato che lascia fuori dall'intervallo una quantità esigua di stime.

Se le stime che vogliamo valutare riguardano solamente una coda della gaussiana possiamo facilmente valutare l'integrale sotto questa parte della funzione rappresentata.

Ricapitolando quanto appena detto, visto che per il calcolo dell'intervallo di confidenza utilizziamo la varianza, quest'ultima la possiamo calcolare tramite l'applicazione della binomiale data da $\hat{\alpha}(a - \hat{\alpha})$. Per il resto, se il numero degli esempi è sufficientemente alto, possiamo approssimare la binomiale con la curva gaussiana, che la preferiamo in quanto simmetrica e quindi ci permette di effettuare calcoli semplificati. Ma questa assunzione è valida solamente se vale la disuguaglianza $na(1 - a) \geq 5$, altrimenti occorrerà usare realmente la binomiale.

.

12.5 ESEMPIO DI VALUTAZIONE DELL'INTERVALLO DI CONFIDENZA

Supponiamo di aver eseguito un esperimento verificando che 80 esempi su 100 sono correttamente classificati. La nostra stima dell'accuratezza è ovviamente $\hat{a} = 0.80$. La varianza sarà invece

$$\hat{a}(1 - \hat{a})/n = 0.80 \cdot 0.20/100 = 0.0016$$

da cui la deviazione standard $\sqrt{0.0016} = 0.04$.

Verifichiamo poi che $n\hat{a}(1 - \hat{a}) \geq 5$ ed effettivamente lo è ($= 16$), quindi possiamo calcolare l'intervallo di confidenza utilizzando come approssimazione la gaussiana. Impostiamo l'intervallo attorno al valore supponendo che la stima sia il vero valore dell'accuratezza, allora avremo ad esempio i seguenti intervalli di confidenza:

- [0.76, 0.84] considerando un intervallo $\mu \pm \sigma = 0.8 \pm 0.04$, ottendendo una percentuale di circa 68%
- [0.72, 0.88] considerando un intervallo $\mu \pm 2\sigma = 0.8 \pm 0.08$, ottendendo una percentuale di circa 95%

Se però riduciamo n , ad esempio perché cambiano dataset considerandone uno con solo 50 esempi di cui 40 correttamente classificate, da cui un'accuratezza sempre del 80% ma una deviazione standard di 0.06 gli intervalli di confidenza considerando intorni con distanza sempre di σ e 2σ sarebbero:

- [0.74, 0.86] con distanza σ
- [0.68, 0.92] con distanza 2σ e confidenza pari al 95%

Si nota inoltre che con un dataset di soli 30 esempi la approssimazione tramite gaussiana non è più possibile e occorrerebbe usare i valori tabulati della binomiale al posto della gaussiana.

12.6 CONCLUSIONI

Ma come usare correttamente gli intervalli di confidenza? Gli intervalli di confidenza sono delle stime per fare delle approssimazioni sulle stime di accuratezza che possiamo osservare durante le misurazioni. Non sono affermazioni sul valore vero del parametro che ricordiamo essere incognito. In pratica quello che facciamo è fissare una accuratezza e sperare che sia quella reale, costruendo un intorno con questo valore al centro. Se le stime che escono dall'area considerata all'interno dell'intervallo di confidenza sono (probabilmente) molto poche allora la nostra valutazione era corretta, altrimenti l'accuratezza stimata era errata. In generale, comunque, non possiamo dire che ci aspettiamo che il valore reale dell'accuratezza sia all'interno di quell'intorno, poiché non sarebbe corretto senza conoscere le probabilità a priori delle stime e delle accuratezze vere. Possiamo solamente dire

che assumendo una accuratezza reale pari a 0.80, la probabilità che una misurazione m ricada nell'intervallo [0.72,0.88] è 0.95.

12.7 LE IPOTESI NULLE E IL P-VALUE

Possiamo tuttavia utilizzare un ragionamento simile per testare una particolare ipotesi nulla che abbiamo su a (il valore vero del parametro).

Con questo metodo si effettua una ipotesi nulla derivante dall'andamento della curva, si calcola una certa deviazione standard e con questa si può calcolare l'ampiezza dell'intervallo di confidenza.

Per esempio, supponiamo che la nostra ipotesi nulla sia che la accuratezza reale sia 0.5 e che la deviazione standard derivante dalla distribuzione sia quindi $\sqrt{0.5(1-0.5)/100} = 0.05$. Data una nostra stima del 0.80 possiamo calcolare il *p-value*, che è la probabilità di ottenere una misurazione dello 0.8 o più alta data l'ipotesi nulla.

Il p-value è poi confrontato con una data soglia solitamente molto bassa (pensiamo per ora ad un $\alpha = 0.05$, a cui corrisponde una confidenza dello 0.95).

L'ipotesi nulla sarà rigettata nel caso in cui il p-value è al di sotto della soglia α , e nel nostro caso $p = 1.9732 \cdot 10^{-9}$

12.8 PAIRED T-TEST

La complicazione sta nel fatto che nella realtà spesso non disponiamo della deviazione standard (e a volte nemmeno della sua stima). Questo significa che l'utilizzo di una distribuzione gaussiana potrebbe risultare troppo incerta. La distribuzione che si utilizza avrà sempre una rappresentazione a campana, simmetrica e centrata intorno al valor medio come la distribuzione normale ma leggermente (vedremo a breve in base a cosa) più alta nelle due code in modo da accettare una maggiore incertezza negli estremi: la chiameremo *t-distribution*.

Il *Paired t-tests* è un test che assume parecchia rilevanza nella Cross-Validation.

Per una coppia di algoritmi viene calcolata la differenza di accuratezza in ciascuna partizione. La nostra ipotesi nulla è che la differenza sia 0, in modo che ogni differenza nella performance è attribuita alla scelta. Si calcola quindi un p-value usando la distribuzione normale, e rifiutiamo l'ipotesi nulla se il p-value è sotto la nostra soglia α .

La t (di Student) è una distribuzione con valori tabulati nel numero delle osservazioni che decide il numero di *gradi di libertà*, parametro che determina in quale misura avviene questo prolungamento delle code: nel nostro caso equivale a 1 – il numero di partizioni poichè la partizione finale è completamente determinata dalle precedenti

Fold	NB-DT	NB-NN	DT-NN
1	-0.0715	-0.0355	0.0361
2	-0.1947	-0.1866	0.0081
3	0.0209	-0.1398	-0.1607
4	-0.2189	0.0088	0.2277
5	-0.1424	-0.1265	0.0159
6	-0.1739	-0.2065	-0.0325
7	0.0992	0.0204	-0.0788
8	-0.0371	0.2255	0.2626
9	-0.2802	-0.2700	0.0102
10	0.0341	0.0410	0.0069
avg	-0.0965	-0.0669	0.0295
stdev	0.1246	0.1473	0.1278
p-value	0.0369	0.1848	0.4833

Figura 12.4: Esempio di applicazione del t-paired test

12.8.1 Esempio di applicazione del Paired t-test

Vediamo un esempio in, riportato in figura 12.4, in cui l'ipotesi nulla è che la differenza venga da una distribuzione normale e deviazione standard sconosciuta.

Il p-value, indicato nell'ultima riga è calcolato dall'uso della t-distribution on $k - 1 = 9$ gradi di libertà e solamente la differenza tra naive bayes e i decision tree è significante con $\alpha = 0.05$.

Si precisa che stiamo solamente valutando coppie di algoritmi.

12.9 WILCOXON'S SIGNED-RANK TEST

Il t-test non è appropriato per dataset multipli poichè non è possibile misurare le performance tra più dataset alla volta (le differenze ottenute non potrebbero essere confrontate, ad esempio perchè i dataset potrebbero avere cardinalità diverse e non sarebbero osservazioni omogenee).

Per poter effettuare test di questo tipo è necessario utilizzare test appositi, come il *Wilcoxon's signed-rank test*. L'idea di quest'ultimo è di effettuare un rank (in ordine crescente) sulle difference di performance in valore assoluto.

Si calcola quindi la somma algebrica tra i rank positivi e negativi separatamente e si considera come statistica di test la sommatoria minore. Come accade spesso, per un ampio numero di dataset (almeno 25) questa statistica può essere considerata come una distribuzione normale, altrimenti si dovrà ricorrere alle tabelle.

Il *Wilcoxon signed-rank test* assume che una ampia differenza di performance è migliore rispetto ad una minore, ma d'altra parte non fa assunzioni circa la loro commensurabilità.

<i>Data set</i>	<i>NB-DT</i>	<i>Rank</i>
1	-0.0715	4
2	-0.1947	8
3	0.0209	1
4	-0.2189	9
5	-0.1424	6
6	-0.1739	7
7	0.0992	5
8	-0.0371	3
9	-0.2802	10
10	0.0341	2

Figura 12.5: Ranking ottenuto dall'applicazione di un certo algoritmo su 10 dataset

Inoltre il test è "non-parametrico" che, nella terminologia della statistica, significa che non c'è alcuna assunzione circa la distribuzione (mentre nel caso precedente assumevamo la t-distribution) e possiamo dire che il test è meno sensibile agli *outliers*. I test parametrici (come il t-test) sono in genere più forti ma possono essere molto fuorviati quando la distribuzione assunta non è corretta.

12.9.1 Esempio di applicazione del Wilcoxon's signed-rank test

Supponiamo di aver ottenuto, dall'applicazione di due algoritmi su 10 dataset, i rank riportati in figura 12.5, tramite ordinamento dei valori assoluti delle differenze di accuratezza. La somma dei rank per le differenze positive è $1 + 5 + 2 = 8$, mentre per quelle negative è $4 + 8 + 9 + 6 + 7 + 3 + 10 = 47$. Il valore critico per i 10 dataset, tabulato e associato ai diversi valori di confidenza (per noi $\alpha = 0.05$) è 8, che devrà essere confrontato con la somma minore tra le due ottenute prima (nel nostro caso sarà la somma dei rank con valori positivi). Se la più piccola delle due sommatorie è minore o uguale ad 8 (valore critico), l'ipotesi nulla secondo cui i rank sono distribuiti ugualmente per le differenze positive e negative non è valida. In questo esempio ricadiamo proprio in questa casistica.

12.9.2 Confrontare diversi algoritmi su più dataset

Se vogliamo invece valutare k algoritmi su n dataset occorre utilizzare test specifici per evitare che il livello di confidenza decada ad ogni

<i>Data set</i>	<i>Naive Bayes</i>	<i>Decision tree</i>	<i>Nearest neighbour</i>
1	0.6809 (3)	0.7524 (1)	0.7164 (2)
2	0.7017 (3)	0.8964 (1)	0.8883 (2)
3	0.7012 (2)	0.6803 (3)	0.8410 (1)
4	0.6913 (2)	0.9102 (1)	0.6825 (3)
5	0.6333 (3)	0.7758 (1)	0.7599 (2)
6	0.6415 (3)	0.8154 (2)	0.8479 (1)
7	0.7216 (1)	0.6224 (3)	0.7012 (2)
8	0.7214 (2)	0.7585 (1)	0.4959 (3)
9	0.6578 (3)	0.9380 (1)	0.9279 (2)
10	0.7865 (1)	0.7524 (2)	0.7455 (3)
avg rank	2.3	1.6	2.1

Figura 12.6: Esempio di applicazione del Friedman test

confronto tra coppie di algoritmi.

Il test di cui stiamo parlando è il *Friedman test*.

Così come il Wilcoxon's signed-rank test, anche questo test è basato su un ranking e non fa alcuna assunzione riguardo la distribuzione delle misurazioni (cioè il test è non-parametrico).

L'idea è quella di effettuare un ranking sulle performance dei diversi algoritmi applicati a ciascun dataset, da quello che si è comportato meglio al peggiore.

Sia R_{ij} il ranking dell'algoritmo j -esimo per l' i -esimo dataset, e sia $R_j = \sum_i R_{ij}/n$ il ranking medio dell'algoritmo j -esimo. Sotto l'ipotesi nulla che tutti gli algoritmi mantengano una certa performance media, R_j dovrebbe attenersi a questa performance. Per questo, calcoliamo le seguenti quantità:

- Il rank medio $\bar{R} = \frac{1}{nk} \sum_{ij} R_{ij} = \frac{k+1}{2}$
- La somma degli scarti quadratici $n \sum_j (R_j - \bar{R})^2$
- La somma degli scarti quadratici $\frac{1}{n(k-1)} \sum_{ij} (R_{ij} - \bar{R})^2$

Si nota un'analogia col clustering in quanto, la terza misura rappresenta una sorta di distanza da un rank "centroide" (che vogliamo essere ampia), mentre la terza misura rappresenta la distanza su tutti i rank.

La statistica di Friedman è data dal rapporto tra il secondo valore a numeratore e il terzo valore a denominatore.

12.9.3 Esempio di applicazione del Friedman's test

Riportiamo una tabella di esecuzione dalla quale abbiamo dato esauriva spiegazione nella sottosezione precedente. Calcoliamo quindi

- $\bar{R} = \frac{k+1}{2} = 2$

- $n \sum_j (R_j - \bar{R})^2 = 10 \cdot [(2.3 - 2)^2 + (1.6 - 2)^2 + (2.1 - 2)^2] = 10 \cdot 0.26 = 2.6$
- $\frac{1}{n(k-1)} \sum_{ij} (R_{ij} - \bar{R})^2 = 1$

La statistica di Friedman sarà quindi $\frac{2.6}{1} = 2.6$. Il valore critico per $k = 3$ e $n = 3$ con $\alpha = 0.05$ è 7.8, quindi l'ipotesi nulla è valida e ciò significa che tutti gli algoritmi mantengono una performance standard.

Viceversa, se immaginassimo che i rank medi ottenuti fossero stati 2.7, 1.3 e 2.0 avremmo dovuto asserire che l'ipotesi nulla non sarebbe stata valida.

12.10 POST-HOC TESTS

Il test di Friedman ci suggerisce una conclusione globale circa la similarità dei k algoritmi considerati, ma una volta effettuato, occorrerà applicare un *post-hoc test* per capire quale sia effettivamente il migliore confrontante le performance a coppie.

Anche questo test non è parametrico.

L'idea è di calcolare la *differenza critica* (CD) data dalla formula dettata dal *Nemenyi test*:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6n}}, \text{ dove } q_\alpha \text{ dipende da } \alpha \text{ e da } k$$

Se supponiamo, per esempio, di aver ottenuto dall'applicazione di 3 algoritmi su 10 dataset, rank medi pari a 2.7, 1.3 e 2.0 allora otterremmo una CD pari a 1.047.

Se andassimo quindi a calcolare $2.7 - 1.3 = 1.4 > 1.047$ arriveremmo alla conclusione che questi due algoritmi sono sufficientemente diversi fra di loro (grazie a questo confronto a coppie effettuato mediante la critical difference).

12.10.1 Bonferroni-Dunn test

Variante del Nemenyi test per il calcolo della differenza critica è il *Bonferroni-Dunn test* in cui il termine q_α è diverso, in relazione al fatto che si effettuano $k - 1$ confronti di coppie e non $k(k - 1)/2$. In definitiva questo test rifiuterà ipotesi un po' più frequentemente rispetto al test di Nemenyi.

Per esempio se $\alpha = 0.05$ e $k = 3$ avremo $q_\alpha = 2.241$.

12.10.2 Rappresentazione grafica del post-hoc test

Graficamente possiamo rappresentare i risultati di questi test andando a calcolare la differenza critica e rappresentandola come un segmento ampio quanto il suo valore stesso e andando a confrontare le

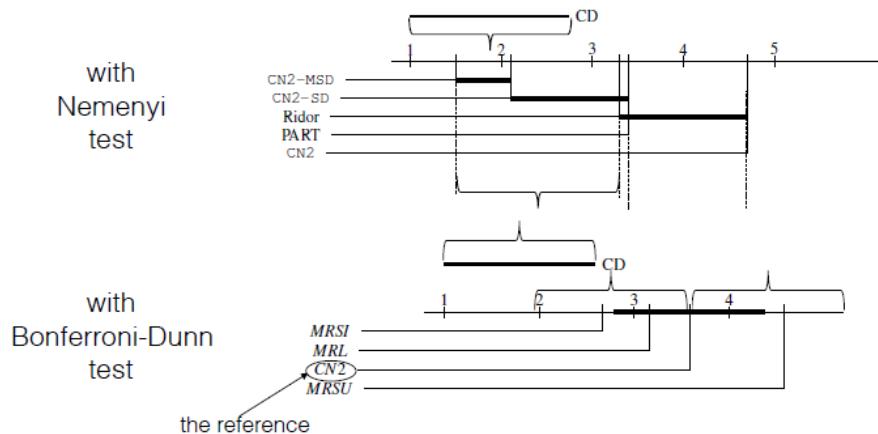


Figura 12.7: Rappresentazione grafica della CD utilizzata graficamente.

differenze di rank medi osservati confrontandoli con l'ampiezza del segmento.

Nel primo caso, con Nemenyi test, in pratica si individua nell'asse delle ascisse il punto rappresentante l'average rank di ciascuno degli algoritmi che vogliamo confrontare, e si collegano due punti riferiti a due algoritmi con un segmento, evidenziandone la distanza che dovrà essere minore rispetto al segmento campione rappresentante proprio la CD.

Nel secondo caso si individua il punto dell'average rank di un algoritmo selezionato, si disegna simmetricamente rispetto a questo due CD, e si vede quali altri algoritmi non eccedono il segmento della CD, ovvero quelli che non saranno significativamente diversi rispetto all'algoritmo "pivot".

La figura 12.7 si vede riportato un esempio di quanto appena spiegato.

INDICE ANALITICO

A

- accuracy, 21
- apprendimento
 - non supervisionato, 1
 - supervisionato, 1
- apprendimento automatico, 1
- association rule discovery, 4

B

- Bayes
 - assunzione naïve, 11

C

- classificatore
 - lineare, *vedi anche* modello lineare
- classificazione
 - binaria, 20
 - binaria, 2
 - multiclasse, 2
- clustering
 - descrittivo, 4
 - predittivo, 2
- contingenza, *vedi* tabella di contingenza
- coverage plot, 26

F

- feature, 13
- funzione di penalizzazione, 33
- funzione loss, *vedi* funzione di penalizzazione

H

- Hilbert, *vedi* spazio di Hilbert

K

- Kernel, 152
 - combinazione di, 156
 - gaussiano, 155
 - polinomiale, 155

quadratico, 155

M

- Maximum A Posteriori, 8
- Maximum Likelihood, 8
- minimi quadrati, 132
- modello, 4
 - geometrico, *vedi* modello lineare
 - geometrico (esempio), 4
 - lineare, 129
 - logico, 4
 - logico (esempio), 12
 - probabilistico, 4, 7
 - probabilistico (esempio), 8

O

- overfitting, 3, 133

P

- performance, 3
 - misura delle, 21
- precisione, 25
- prodotto interno, 154

R

- ranking, 35
- ranking error rate, 37
 - esempi, 38
- recall, 25
- regression, 2, 129
 - lasso, 135
 - regolarizzata, 133
 - ridge, 134
- ROC plot, 29

S

- scoring, 32
- spazio di Hilbert, 154
- subgroup discovery, 3
- Support Vector Machine, 139
 - margine di errore, 148

Support Vector, [141](#)
SVM, *vedi* Support Vector Ma-
chine

T

tabella di contingenza, [21](#)

task, [1](#)

descrittivi, [3](#)

predittivo, [1, 2](#)

test set, [3](#)

training set, [3](#)