UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI SCIENZE ECONOMICHE E STATISTICHE

Corso di Laurea in STATISTICA PER I BIG DATA
Tesi di Laurea in STATISTICAL LEARNING

Support Vector Machines & Bankruptcy Prediction

Relatore:
Ch.mo Prof.
Pietro CORETTO

Candidato:
Dario FALCHETTA
Mat. 0212800916

ANNO ACCADEMICO 2022/2023

Abstract

Le Macchine a Vettori di Supporto (*Support Vector Machines*, SVM, Vapnik et al. [12]) sono un metodo di classificazione per problemi a classi binarie che consentono di individuare decision boundary arbitrariamente non lineari. La separazione, mediante un iperpiano, di classi non linearmente separabili, è ottenuta attraverso una gamma di funzioni kernel che proiettano i dati in uno spazio vettoriale di dimensione maggiore (infinitamente grande in alcuni casi) in modo da poter individuare separazioni precedentemente impercettibili. Siccome l'iperpiano dipende solo da specifici data points, detti appunto *Support Vector Machines*, il metodo presenta un certo livello di robustezza e tolleranza a dati sparsi, valori anomali e classi sbilanciate. Questo lavoro studia il meccanismo alla base del metodo SVM e la sua applicazione al problema di previsione di bancarotta per imprese sulla base delle loro condizioni finanziarie.

Indice

1 Introduzione

Il problema della *classificazione* trova svariati campi di applicazione pratici e fa riferimento alla necessità di assegnare un soggetto/oggetto ad una tra due o più classi, note a priori, sulla base di una serie di caratteristiche che sono state misurate su di esso. Nella sezione (2) viene descritto e formalizzato il problema della classificazione dal punto di vista statistisco e vengono presentati gli steps da condurre per ottenere risultati validi ed utilizzabili. Dopodiché, vengono presentati i dati oggetto di studio, analizzati con l'obiettivo di prevedere, e potenzialmente prevenire, il fallimento di un'impresa.

Nella sezione (3) viene presentato il metodo di classificazione *Support Vector Machines* ideato ai AT&T Bell Laboratories da Vladimir Vapnik ed i suoi colleghi, focus principale di questo lavoro. La descrizione di questo metodo è condotta in maniera incrementale, partendo dalla teoria su cui è basata la classificazione prodotta, per poi arrivare alla versione più evoluta del metodo passando prima per alcuni step intermedi, il Maximal Margin Classifier ed il Support Vector Classifier, e descrivendo i problemi e le soluzioni che hanno determinato la sua evoluzione.

Nella sezione (4) si descrivono le operazioni di pre-processing applicate ad i dati, necessarie per il loro utilizzo. Infine, nella sezione (5) si presentano le decisioni ed i risultati relative all'analisi dei dati, riassunti poi nelle conclusioni in sezione (6).

2 Background

2.1 Classificazione

Il concetto di *classe* può essere definito come una caratteristica qualitativa che accomuna tutti gli oggetti/soggetti appartenenti ad essa e, data la presenza di una "ground truth" a cui fare riferimento, cioè la suddetta classe a cui appartengono gli esemplari osservati, si parla in questo caso di *supervised learning*, a differenza del più complesso caso dell'*unsupervised learning* in cui tali informazioni non sono disponibili.

La risoluzione del problema della classificazione, ovvero l'assegnazione delle unità osservate alla classe corretta, consiste nella ricerca di regole decisionali che presentino opportune garanzie: nonostante la possibilità di commettere

errori sia imprescindibile, si cerca di costruire classificatori con tassi di errore prossimi allo zero.

Formalmente, osservato un vettore di features $X = (X_1, \ldots, X_p)^T$ per le varie unità statistiche soggette ad analisi, si desidera costruire un classificatore, cioè una funzione $\hat{Y}(X) \in \mathcal{Y}$, dove $\mathcal{Y}$ rappresenta l'insieme delle possibili class labels, tale che $\hat{Y}(x) = y_k$ se l'osservazione x appartiene alla classe $y_k \in \mathcal{Y}$.

In un problema di classificazione binaria, la costruzione del classificatore parte da un insieme di osservazioni di training

$$\mathcal{D} = \{(x_i, y_i) | x_i \in \mathbb{R}^p, y_i \in \{-1, +1\}\}_{i=1}^n$$

la cui la classe di appartenenza y_i è nota. L'errore commesso assegnando un'osservazione alla classe sbagliata viene quantificato attraverso una funzione di perdita $L(.)$ ed una serie di metriche possono essere calcolate per valutare le performance del modello costruito. Queste metriche possono poi essere utilizzate per selezionare il valore più opportuno di eventuali iperparametri che controllano la complessità del modello ed il suo adattamento ai dati osservati, oppure per confrontare tra loro modelli diversi e scegliere quello che garantisce la migliore performance. Una volta ottenuto un classificatore desiderabile attraverso opportuni approcci di training, è possibile implementare tale modello per la classificazione di osservazioni unseen delle quali non si conosce la vera classe di appartenenza.

2.2 Addestramento di un modello

Il tipico approccio di addestramento di un modello consiste in un iniziale partizionamento dei dati osservati in *training set* e *test set*: il primo è costituito dalle osservazioni che verranno utilizzate per costruire il modello, le cui performance vengono poi valutate sul secondo e considerate come proxy della sua capacità di trattare i dati unseen.

2.2.1 Bias-Variance trade-off

La sfida principale da affrontare in qualsiasi applicazione di machine learning è la gestione del livello ottimale di complessità per il modello, ovvero il suo adattamento ai dati osservati detto performance *in-sample*, ed il conseguente comportamento su nuovi data points, la performance *out-of-sample*.

È possibile dimostrare che l'errore commesso da una qualsiasi tecnica di learning sia funzione del suo bias, della sua varianza e della variabilità dei dati osservati: le prime due prendono il nome di *errore riducibile* e sono quantità non negative che dipendono dal modello considerato; la terza, invece, è detta *errore irriducibile* ed è fissa per un certo set di dati osservati. È facile capire che, indipendentemente dalla variabilità dei dati osservati, la qualità del modello dipende dalla gestione del *bias-variance trade-off*, meccanismo intrinseco del machine learning che controbilancia la capacità di rispettare la ground truth osservata con l'abilità del modello di generalizzare su dati unseen: con *bias* si intende l'errore introdotto approssimando un problema reale, che potrebbe essere estremamente complesso, con un modello eccessivamente semplice; con *varianza*, invece, si fa riferimento alla possibilità che il modello produca in output risultati molto diversi in seguito a leggere variazioni dei dati su cui è costruito. In aggiunta, un ulteriore punto di vista da considerare, è che l'errore commesso dal modello stimato sarà sempre fortemente legato alla variabilità dei dati osservati la cui, se dovesse essere alta, porterebbe anche il miglior modello a produrre risultati incerti.

Complessità e flessibilità, che può essere vista come la capacità del modello di interpolare strutture geometriche arbitrariamente complesse, sono due termini utilizzati per illustrare le sfaccettature del bias-variance trade-off:

- Modelli dall'elevata complessità sono molto flessibili e sono in grado di adattarsi molto bene ai dati su cui sono stati costruiti, cosa che può, tuttavia, portare il modello ad avere una scarsa performance sui dati non osservati. In termini di bias e varianza si parla di *overfitting*, cioè il modello presenta un bias molto basso, perché i dati vengono approssimati con una relazione pressoché perfetta, ma una varianza più elevata perché l'eccessivo adattamento ai dati osservati provoca una perdita di capacità di generalizzare che lo porta a produrre risultati molto diversi sul test set in seguito a leggere variazioni nei dati di training.

- Modelli dalla bassa complessità sono poco flessibili ed incapaci di cogliere la struttura dei dati in-sample, caso in cui si parla di *underfitting*. Lo scarso adattamento si traduce in elevato bias, perché si vanno ad approssimare i dati con una relazione pessima, e probabilmente anche elevata varianza dal momento che non è stato colto il meccanismo alla base dei dati.

È quindi evidente come la scelta del modello più opportuno per il problema in questione dipenda dal bilanciamento tra bias e varianza. Quale delle due quantità prevale sull'altra, tuttavia, dipende anche da quale è l'utilizzo inteso per il modello.

2.2.2 Selezione e Validazione

Le applicazioni in questione sono la *selezione* e la *validazione*.

Con selezione si fa riferimento alla ricerca del valore più opportuno per un particolare iperparametro che permetta di selezionare un particolare membro dalla famiglia di modelli da esso definita. Un *iperparametro* è una componente che un modello può avere per gestire alcune delle caratteristiche inerenti alla sua natura ed il suo comportamento. Non esiste un "valore giusto" per un dato iperparametro perché questo dipende sempre dai dati e dalle circostanze del problema da affrontare. Per questo motivo, piuttosto che "stimare" il suo valore, si parla di "selezionare" quello più opportuno. Tale ricerca è condotta costruendo vari modelli a partire da diversi settaggi per l'iperparametro e confrontando la loro capacità previsiva attraverso la stima di bias e varianza di previsione: siccome si è interessati al modello in grado di produrre i risultati più consistenti, cioè dal tasso d'errore meno variabile, la quantità di principale interesse è la varianza. Tra i diversi modelli stimati, la scelta del migliore è condotta sulla base di un principio di parsimonia attraverso la regola *one-standard-error* o *two-standard-error*, che consistono nell'approssimare l'intervallo di confidenza della stima dell'errore come

$$\mathrm{err}_{\mathrm{CV}} \pm const \times \mathrm{SE}_{\mathrm{CV}}$$

dove

- $\mathrm{err}_{\mathrm{CV}}$ è la stima dell'errore ottenuta dalla Cross Validation;
- $\mathrm{SE}_{\mathrm{CV}}$ è lo standard error della stima dell'errore ottenuta dalla Cross Validation;
- const è una costante che definisce l'ampiezza dell'intervallo di confidenza.

Se le stime $\bar{L}_s$, a partire dalle quali è calcolato $\mathrm{err}_{\mathrm{CV}}$, fossero normalmente distribuite, fissare $const = 1.96$ permetterebbe di ottenere degli intervalli di confidenza al 95% per la stima dell'errore. Siccome questa ipotesi non è quasi mai fondata, solitamente si fissa $const = 1$ o $const = 2$ anche se il livello

di confidenza non è noto: fare ciò è possibile dal momento che tale intervallo avrà la stessa ampiezza per tutti i diversi settaggi del modello. Il principio di parsimonia è finalmente implementato individuando il modello che produce la stima dell'errore più bassa e costruendo per esso l'intervallo di confidenza appena descritto: siccome tutti i modelli che ricadono in questo intervallo possono essere considerati indistinguibili l'uno dall'altro, la sceltà ricadrà su quello il cui livello di complessità, regolato dal valore dell'iperparametro, è minore.

Con validazione si fa invece riferimento alla necessità di scegliere il modello che garantisce la migliore performance per il task di learning in questione a partire da un range di alternative. In questo caso si stanno confrontando modelli di natura diversa e ciò a cui si è interessati è la loro capacità previsiva, ovvero al modello che è in grado di commettere meno errori out-of-sample. Per questo motivo, la quantità di interesse è il bias ed il modello scelto sarà quello che presenta il bias più basso.

2.2.3 Cross Validation

Accontentarsi delle performance ottenute dal modello su un unico partizionamento dei dati in training e test set potrebbe non essere la scelta migliore dal momento che la casualità coinvolta in tale partizionamento potrebbe aver generato circostanze più che favorevoli, creando una percezione eccessivamente ottimistica della qualità del modello o, al contrario, circostanze inutilmente intricate e complesse da gestire che lo hanno portato a performare male ma che in realtà non rispecchiano la sua performance media sui dati in questione.

La *Cross Validation* è lo strumento statistico utilizzato per valutare le capacità predittive dei modelli sfruttando la potenza di calcolo per produrre molteplici split dei dati in training e test set e facendo media sulle loro prestazioni. Esistono varie possibili configurazioni alternative a seconda di quale è il task di interesse ma, in linea di massima, l'algoritmo che permette di implementare la Cross Validation è il seguente:

Algorithm 1: Cross Validation in generale

input: data set $\mathbb{X}_n = \{(y_1, x_1), (y_2, x_2), \ldots, (y_n, x_n)\}$

for $s = 1, \ldots, S$ **do**

split: partiziona casualmente i dati in $\mathbb{X}_{(s)}^{train}$ e $\mathbb{X}_{(s)}^{test}$

train: stima $\hat{f}_s(.)$ su $\mathbb{X}_{(s)}^{train}$

predict: prevedi l'outcome y in $\mathbb{X}_{(s)}^{test}$ usando $\hat{f}_s(.)$

test: calcola $\bar{L}_s \leftarrow$ media sul $\mathbb{X}_{(s)}^{test}$

end

return

$$\text{err}_{\text{CV}} \leftarrow \sum_{s=1}^{S} \bar{L}_s$$

Ciò che si fa è partizionare i dati casualmente in train e test set S volte, costruire il modello su ogni training set e usarlo per fare previsioni sui dati del corrispondente test set, ottenendo così S stime dell'errore di cui si calcola la media per ottenere la valutazione finale prodotta dalla Cross Validation.

Nonostante risulti essere molto intuitivo, la Cross Validation è uno strumento molto complesso e potenzialmente instabile perché soggetto a varie fonti di casualità:

- la variabilità intrinseca del popolazione da cui provengono i dati;
- la variabilità del campione $\mathbb{X}_n$ estratto;
- la variabilità dovuta all'aver scelto una specifica configurazione per ogni $\hat{f}_s(.)$;
- la variabilità di ogni training set $\mathbb{X}_{(s)}^{train}$ a partire dal quale è stata costruita $\hat{f}_s(.)$, dal fatto che questa rispecchia solo una parte della variabilità totale della popolazione e dalla sua dipendenza dalla tecnica di campionamento utilizzata.

Il modo più immediato per avere un minimo controllo sul peso attribuito a queste varie fonti di incertezza, oltre che al numero di split S da considerare ed oltre al metodo di partizionamento, è la scelta della dimensione relativa delle partizioni. La logica alla base della scelta è che una maggiore numerosità per il training set consente di ottenere un modello più stabile, costruito a partire da una maggiore quantità di informazioni, il che gli permette di raggiungere un migliore adattamento ai dati ma portandolo tuttavia ad avere stime meno

consistenti, cioè dalla variabilità più alta, data la dimensione ridotta del test set. Al contrario, un training set meno numeroso introduce distorsione nelle stime del modello, che tuttavia preserverà una maggiore capacità di generalizzazione dato il maggior numero di osservazioni assegnate al test set, risultando in stime meno variabili.

Nel prendere questa decisione bisogna considerare anche che qualsiasi sovrapposizione delle partizioni, tra uno degli S partizionamenti e l'altro, introduce correlazione nelle stime dell'errore, inflazionando la varianza e facendola esplodere ad infinito molto velocemente, sfalsando così i risultati.

Come visto finora, non c'è una configurazione applicabile a tutte le possibili circostanze ma queste vanno analizzate, capite e valutate, in modo da poter prendere la decisione più opportuna.

2.2.4 Metriche di performance

Per i problemi di classificazione, la funzione di perdita solitamente considerata è la *loss 0-1*, definita come

$$L\left(y_i, \hat{Y}(x_i)\right) = I\left\{y_i \neq \hat{Y}(x_i)\right\} = \begin{cases} 1 & se\ y_i \neq \hat{Y}(x_i) \\ 0 & altrimenti \end{cases}$$

cioè una funzione che assume valore 1 quando il classificatore costruito assegna l'osservazione alla classe sbagliata e 0 in caso contrario. Il confronto tra classi osservate e previste è la base per la valutazione delle performance di un classificatore. A partire dalla matrice di confusione ottenuta da tale confronto è possibile costruire una serie di metriche utili per descrivere le qualità e le capacità previsive del metodo di learning considerato.

Una *matrice di confusione* è una tabella in cui vengono confrontate le classi osservate nei dati con quelle previste dal classificatore e contati il numero di match e mismatch tra di esse:

Classi Vere \ Classi Previste	**Negativo (-)**	**Positivo (+)**	**Somma**
Negativo (-)	TN	FP	N
Positivo (+)	FN	TP	P
Somma	N^*	P^*	n

La misura di accuratezza più intuitiva da considerare è il *Misclassification Rate*, dato da

$$ERR = \frac{FN + FP}{n}$$

e rappresenta la percentuale di osservazioni erroneamente classificate. Alternativamente, è possibile riscrivere questo indice anche in termini di funzione di perdita come media campionaria delle loss 0-1 calcolate sulle osservazioni del campione

$$ERR = \frac{1}{n} \sum_{i=1}^{n} L\left(y_i, \hat{Y}(x_i)\right).$$

Nonostante sia un indice molto facile da interpretare e utile, dato che permette di considerare teoremi asintotici, il misclassification rate potrebbe non essere la misura di errore più opportuna per una determinata applicazione perché conferisce lo stesso peso agli errori di I^o e II^o tipo, definiti rispettivamente come assegnare alla classe dei positivi un'osservazione appartenente alla classe dei negativi e viceversa.

Due delle metriche più importanti ricavabili dalla confusion matrix sono il *True Positive Rate* (TPR) ed il *False Positive Rate* (FPR), definite come

$$TPR = \frac{TP}{P}$$

e

$$FPR = \frac{FP}{N},$$

rispettivamente uguali alla proporzione di veri positivi correttamente classificati e veri negativi erroneamente classificati. I due indici sono legati da un trade-off sistematico, una relazione che fa in modo che il valore di uno aumenti al diminuire dall'altro.

L'utilizzo di queste metriche coninvolge solitamente una griglia di valori soglia su cui basare la classificazione e la visualizzazione grafica della curva *ROC* (*Receiver Operating Curve*), una curva che mostra come il classificatore riesce a gestire il trade-off tra TPR e FPR al variare del valore soglia: un buon classificatore è uno il cui valore dell'area sotto la curva *ROC* (detta *AUC*) è vicino ad 1. L'andamento della curva *ROC* dipende chiaramente dai dati su cui è costruita quindi è da considerare come una curva casuale, soggetta alla variabilità del campione. Di conseguenza, per rendere i risultati più informativi, è possibile considerare intervalli di confidenza per la curva stessa.

Alcune finali metriche di performance da tenere in considerazione nei pro-

blemi di classificazione sono l'indice *F-score* ed il *Coefficiente di Correlazione di Matthews* (MCC). Il primo è definito media armonica tra *Precision* (TP/P^*) e *Recall* (TP/P), ovvero

$$F_1 = 2\frac{Precision \times Recall}{Precision + Recall}.$$

Una particolarità di questo indice è che le quantità utilizzate per calcolarlo considerano solo parte della matrice di confusione, ovvero quella relativa ai positivi, ignorando completamente la classe dei negativi, il che può essere poco opportuno quando si è particolamente interessati alla corretta classificazione di questi ultimi. Per colmare queste mancanze è possibile fare riferimento all'indice MCC, una misura di dipendenza tra le classi vere e quelle previste

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN}}.$$

Tale indice produce valori desiderabili quando la classificazione rispetta alle classi osservate, assumendo valori $+1$ o -1 in presenza di perfetta classificazione e 0 quando la performance valutata non può essere considerata migliore del random guessing, cioè l'assegnazione del 50% delle osservazioni ad ogni classe.

2.3 Dataset

La costruzione di modelli bassati su misure econometriche per analizzare la situazione finanziaria, prevedere e anticipare lo stato di bancarotta di un'impresa è un topic di forte interesse dal momento che le performance di imprese, grandi o piccole che siano, hanno un diretto effetto sulle comunità locali, sui concorrenti del settore, sugli investitori ma anche sull'economia globale.

I dati oggetto di studio di questo lavoro, riguardanti la condizione finanziaria di imprese del settore manifatturiero polacco, sono stati trattati da Maciej Zięba, Sebastian K. Tomczak e Jakub M. Tomczak della Wrocław University of Science and Technology e prelevati dal database *Emerging Markets Information Service* (EMIS), una compagnia che si occupa di reportaggio in prima linea in una moltitudine di mercati. I dati relativi alle imprese fallite sono registrati per gli anni dal 2007 al 2013, mentre per quelle non fallite vanno dal 2000 al 2012. Questi dati sono suddivisi in cinque file a seconda dell'orizzonte previsivo analizzato per la previsione. Di questi cinque, considererò il dataset contenente

i dati relativi al quinto anno del periodo di previsione in cui la variabile di risposta *class* indica se l'impresa è fallita o meno nell'anno successivo.

Gli indicatori misurati sulle osservazioni sono 64 in totale, alcuni dei quali sono presentati come *synthetic features* ovvero misure calcolate come combinazione aritmetica di altri indici econometrici [14, p.1]. Il tutto è descritto nella seguente tabella:

ID	**Descrizione**	**ID**	**Descrizione**
X1	net profit / total assets	X2	total liabilities / total assets
X3	working capital / total assets	X4	current assets / short-term liabilities
X5	[(cash + short-term securities + receivables - short-term liabilities) / (operating expenses - depreciation)] * 365	X6	retained earnings / total assets
X7	EBIT / total assets	X8	book value of equity / total liabilities
X9	sales / total assets	X10	equity / total assets
X11	(gross profit + extraordinary items + financial expenses) / total assets	X12	gross profit / short-term liabilities
X13	(gross profit + depreciation) / sales	X14	(gross profit + interest) / total assets
X15	(total liabilities * 365) / (gross profit + depreciation)	X16	(gross profit + depreciation) / total liabilities
X17	total assets / total liabilities	X18	gross profit / total assets
X19	gross profit / sales	X20	(inventory * 365) / sales
X21	sales (n) / sales (n-1)	X22	profit on operating activities / total assets
X23	net profit / sales	X24	gross profit (in 3 years) / total assets
X25	(equity - share capital) / total assets	X26	(net profit + depreciation) / total liabilities
X27	profit on operating activities / financial expenses	X28	working capital / fixed assets
X29	logarithm of total assets	X30	(total liabilities - cash) / sales
X31	(gross profit + interest) / sales	X32	(current liabilities * 365) / cost of products sold
X33	operating expenses / short-term liabilities	X34	operating expenses / total liabilities
X35	profit on sales / total assets	X36	total sales / total assets
X37	(current assets - inventories) / long-term liabilities	X38	constant capital / total assets

X39	profit on sales / sales	X40	(current assets - inventory - receivables) / short-term liabilities
X41	total liabilities / ((profit on operating activities + depreciation) * (12/365))	X42	profit on operating activities / sales
X43	rotation receivables + inventory turnover in days	X44	(receivables * 365) / sales
X45	net profit / inventory	X46	(current assets - inventory) / short-term liabilities
X47	(inventory * 365) / cost of products sold	X48	EBITDA (profit on operating activities - depreciation) / total assets
X49	EBITDA (profit on operating activities - depreciation) / sales	X50	current assets / total liabilities
X51	short-term liabilities / total assets	X52	(short-term liabilities * 365) / cost of products sold)
X53	equity / fixed assets	X54	constant capital / fixed assets
X55	working capital	X56	(sales - cost of products sold) / sales
X57	(current assets - inventory - short-term liabilities) / (sales - gross profit - depreciation)	X58	total costs / total sales
X59	long-term liabilities / equity	X60	sales / inventory
X61	sales / receivables	X62	(short-term liabilities *365) / sales
X63	sales / short-term liabilities	X64	sales / fixed assets

Tabella 1: Variabili dataset

2.4 Sbilanciamento tra classi

Come presentato nella sezione (4) relativa al pre-processing dei dati, il dataset soggetto ad analisi presenta classi di numerosità molto diversa. Lo sbilanciamento tra classi è un fenomeno molto comune nei problemi di classificazione ed è facile intuirne il perché semplicemente pensando, ad esempio, al numero di banconote false che possono essere in circolazione, al numero di email "spam" che vengono ricevute ogni giorno, al numero di transazioni fraudolente rilevate o, ancora, al numero di pazienti affetti da una determinata rara patologia che si desidera studiare. Solitamente, lo sbilanciamento è a sfavore della classe dei positivi, cioè quella usata per codificare la caratteristica in esame.

I modelli prodotti a partire da datasets che rientrano in questa categoria potrebbero risultare erroneamente performanti a causa, ad esempio, degli indici usati per valutare la loro performance. È sicuramente il caso del misclassification rate (ERR) e, analogamente dell'indice di accuratezza ($ACC = 1 - ERR$) che danno lo stesso peso agli errori FP ed FN quando, in realtà, uno dei due potrebbe chiaramente essere più grave dell'altro come, ad esempio, considerare sano (assegnandolo alla classe dei negativi) un individuo che in realtà è affetto da una patologia (cioè appartiene alla classe dei positivi), il che lo renderebbe un falso negativo, piuttosto che considerare malato un individuo che non lo è, identificandolo come falso positivo.

Ottenere questo tipo di risultato è un'immediata conseguenza del fatto che il modello è stato addestrato principalmente su dati appartenenti ad una delle categorie, rendendolo incapace di sapere come assegnare con sicurezza le osservazioni alle altre classi. In questi casi, paradossalmente, sarebbe addirittura possibile ottenere performance migliori assegnando tutte le osservazioni alla classe più rappresentata nei dati, ottenendo così un'accuratezza molto elevata.

In alcune circostanze, la presenza di classi sbilanciate non ostacola troppo i modelli, che riescono comunque a svolgere la classificazione con buoni risultati: la scelta più semplice ed immediata sarebbe semplicemente ignorare gli indici ACC o ERR e considerare diverse misure di performance come l'indice *F-score* oppure la curva ROC e relativa AUC.

In altri casi, tuttavia, i modelli non riescono ad apprendere abbastanza informazioni per poter assegnare adeguatamente alla classe meno numerosa. Alcune delle soluzioni a cui è possibile ricorrere vanno ad agire sul metodo di campionamento.

2.4.1 Oversampling e Undersampling

Oversampling e *undersampling* sono alcune delle numerose possibili tecniche utilizzabili per regolare la distribuzione delle classi in un campione. Considerato un fenomeno in cui il 20% delle osservazioni appartengono alla classe dei positivi e l'80% a quella dei negativi, condurre oversampling significherebbe campionare con reinserimento dalla classe dei positivi quattro volte di più, fino ad ottenere un campione in cui entrambi le classi sono ugualmente rappresentate. Nell'undersampling invece si fa il contrario: ipotizzando la stessa situazione appena presentata, la classe dei negativi verrebbe campionata un quarto delle volte, ottenendo un campione in cui le classi hanno la stessa numerosità.

Tipicamente, tra questi due metodi, l'oversampling è quello preferibile dal momento che con l'undersampling si va a scartare informazione campionaria che potrebbe essere utile ad ottenere modelli di migliore qualità. Un punto a favore dell'undersampling potrebbero essere i minori costi pratici e relativi al campionamento ma questi si sono rivelati triviali con le nuove tencologie dell'era dei Big Data.

2.4.2 SMOTE

Oversampling e undersampling si pongono ai due estremi delle tecniche di campionamento utilizzate nei problemi con classi sbilanciate ma, tra di esse, esistono in letteratura un gran numero di alternative che vanno a compensare alcune delle loro rispettive mancanze. *SMOTE* (*Synthetic Minority Oversampling Technique*) [1] è la più comune di queste in cui, invece di fare oversampling con reinserimento, questo è condotto attraverso la creazione di osservazioni "artificiali". Il procedimento è il seguente:

1. si considera un certa osservazione x_m appartenente alla minority class, quella meno rappresentata nei dati;
2. si calcola $x_d = x_m - x_j$, il vettore differenza tra x_m ed uno dei suoi k-nearest neighbors x_j;
3. si calcola $x_d^* = \alpha x_d$ con $\alpha \in [0, 1]$, il che consiste nel selezionare un determinato punto appartenente al segmento che unisce x_m e x_j;
4. si somma il risultato dello step precedente all'osservazione iniziale, ottenendo così un data point artificiale come $x_m^* = x_m + x_d^*$;
5. gli step 2, 3 e 4 vengono ripetuti con i diversi k-nearest neighbors in base alla quantità di oversampling necessaria. Ad esempio, se è necessario un oversampling del 200%, il procedimento viene ripetuto per 2 dei k-nearest neighbors.

La tecnica SMOTE è presentata con la possibilità di essere affiancata ad un undersampling della majority class [1, p.331], scartando osservazioni da essa finché la minority class non rappresenta una certa percentuale della precedente. Si dimostra che, così facendo, l'iniziale distorsione del metodo di learning verso la classe più numerosa cambia di direzione a favore di quella meno numerosa, ottenendo così migliori risultati rispetto al semplice undersampling o all'oversampling con reinserimento. Il motivo che spiega ciò risulta essere abbastanza

intuitivo [1, p.352]: ricampionando la stessa osservazione più volte, la regione attorno ad essa diventa molto "densa" e piccola all'aumentare delle ripetizioni perché concentrata in un solo punto, rimpicciolendo lo spazio attribuito alla minority class provocando così l'effetto opposto a quello desiderato; al contrario, generando osservazioni artificiali attorno ad essa, tale regione viene ingrandita, migliorando le capacità della regola decisionale di trattare la minority class.

3 Metodologia

Per capire il meccanismo alla base del classificatore SVM bisogna partire da ciò su cui è basato, il *Maximal Margin Classifier*, per poi passare al *Support Vector Classifier*, una sua generalizzazione, ed arrivare infine al *Support Vector Machine*.

3.1 Iperpiano

Il Maximal Margin Classifier è basato sul concetto di iperpiano separatore ottimo, cioè quell'iperpiano che garantisce la separazione, detta margine, più grande tra le due classi. In uno spazio p-dimensionale, un iperpiano L è un suo sottoinsieme piatto e affine, cioè possiede $p-1$ dimensioni e nessuno dei punti appartenenti ad esso è privilegiato rispetto agli altri. I punti appartenenti a questo sottoinsieme rispettano la condizione

$$L = \{x : f(x) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p = 0\},$$

o in forma vettoriale

$$L = \{x : f(x) = \beta_0 + \beta^T x = 0\}, \tag{1}$$

dove:

- β_0 è l'intercetta dell'iperpiano L;
- $\beta = (\beta_1, \beta_2, \ldots, \beta_p)^T$ è un vettore colonna di dimensione $p \times 1$;
- $x = (x_1, x_2, \ldots, x_p)^T$ è un vettore colonna di dimensione $p \times 1$.

In $p = 2$ dimensioni, i punti appartenenti a tale insieme costituiscono una retta, in $p = 3$ dimensioni un piano mentre, in $p > 3$ dimensioni, la struttura

geometrica definita prende il nome di iperpiano la quale, nonostante sia difficile da immaginare, non è altro che una generalizzazione delle precedenti.

L'iperpiano L divide lo spazio in due parti: un sottoinsieme di punti per i quali

$$f(x) = \beta_0 + \beta^T x > 0,$$

ed uno per i quali

$$f(x) = \beta_0 + \beta^T x < 0.$$

Alcune proprietà dell'iperpiano L sono le seguenti:

- Per ogni punto x che giace su L, si ha che $\beta^T x = 0$ ovvero β è ortogonale ad L ed a tutti i vettori che giacciono su di esso. Questo fa sì che attraverso $\beta^T x_i$ sia possibile calcolare la distanza ortogonale di ogni osservazione x_i da L;

- I parametri β_0 e β vengono normalizzati rispetto al magnitudo di β, ovvero $(\beta_0, \beta^*) = \left(\frac{\beta_0}{\|\beta\|}, \frac{\beta}{\|\beta\|}\right)$, il che rende β^* un vettore unitario, cioè tale che $\|\beta\|^1 = 1$.

La normalizzazione dei parametri β_0 e β è un passaggio necessario dal momento che, altrimenti, l'iperpiano non sarebbe *scale invariant*: moltiplicandoli per una certa costante sarebbe possibile individuare un iperpiano dal margine più ampio senza, in realtà, cambiare nulla di significativo. Siccome le distanze sono fondamentali per l'affidabilità della classificazione, è necessario garantire risultati che abbiano una certa integrità.

Ricavato β^*, questo può essere usato per calcolare la distanza euclidea m_i di un certo punto x_i dalla sua proiezione x_i' sull'iperpiano L. Infatti, considerato un vettore $\gamma_i = x_i - x_i'$ di magnitudo $\|\gamma_i\| = m_i$ che giace sull'iperpiano ed ha la stessa direzione di β^*, questo può essere definito come β^* riscalato per la costante m_i, ovvero

$$\gamma_i = m_i \beta^* = m_i \frac{\beta}{\|\beta\|}.$$

A questo punto è possibile vedere come il punto x_i' sia la proiezione di x_i su L

[1] $\|\beta\| = \sum_{j=1}^{p} \beta_j^2$ è la norma euclidea del vettore β.

a partire dalla definizione di γ_i

$$\begin{aligned} \gamma_i &= x_i - x_i' \\ x_i' &= x_i - \gamma_i \\ x_i' &= x_i - m_i \frac{\beta}{\|\beta\|} \end{aligned}$$

Siccome x_i' giace su L, allora rispetterà la condizione

$$\begin{aligned} f(x_i') &= \beta_0 + \beta^T x_i' = 0 \\ &= \beta_0 + \beta^T \left(x_i - m_i \frac{\beta}{\|\beta\|} \right). \end{aligned}$$

Risolvendo per m_i è possibile ricavare la distanza del punto x_i dalla sua proiezione x_i' su L

$$\begin{aligned} \beta_0 + \beta^T x_i - m_i \frac{\beta^T \beta}{\|\beta\|} &= 0 \\ \beta_0 + \beta^T x_i - m_i \frac{\|\beta\|^2}{\|\beta\|} &= 0 \\ \beta_0 + \beta^T x_i - m_i \|\beta\| &= 0 \\ m_i &= \frac{\beta_0 + \beta^T x_i}{\|\beta\|} = \frac{f(x_i)}{\|\beta\|} \end{aligned}$$

È possibile vedere come m_i, la distanza di x_i da L sia chiaramente legata alla definizione stessa dell'iperpiano:

- alcune osservazioni presenteranno $y_i f(x_i) = 1$ e di conseguenza $m_i = \frac{1}{\|\beta\|}$ che, successivamente, si dimostrerà essere pari alla semi-ampiezza del *geometric margin*;
- le osservazioni che presenteranno $y_i f(x_i) > 1$ saranno quelle soggette ad una corretta classificazione.

Considerato che β_0 è semplicemente una costante, viene fissato $\beta_0 = \frac{\beta_0}{\|\beta\|}$. Allora

$$m_i = \beta_0 + \beta^{*T} x_i$$

e, aggiungendo informazioni relative alla classe di appartenenza di x_i

$$m_i = y_i \left(\beta_0 + \beta^{*T} x_i \right).$$

3.2 Maximal Margin Classifier

Operativamente, la selezione dell'iperpiano ottimale, tra gli infiniti iperpiani che rispettano l'equazione (1), è basata sugli n punti osservati $x_i \in \mathcal{D}$. Immaginando un iperpiano L che permetta di separare perfettamente le classi, ogni osservazione x_i goderebbe della proprietà che

$$f(x_i) = \begin{cases} \beta_0 + x_i^T\beta > 0 & se\ y_i = +1 \\ \beta_0 + x_i^T\beta < 0 & se\ y_i = -1 \end{cases}$$

il che può essere riscritto in forma compatta come

$$f(x_i) = y_i \left(\beta_0 + x_i^T\beta\right) > 0 \tag{2}$$

Regole decisionali come questa, che calcolano una trasformazione lineare delle features e ne restituiscono il segno, senza calcolare alcun tipo di probabilità di appartenenza alle classi, sono dette *percettroni* (Rosenblatt, 1958) e sono basate sulla minimizzazione della distanza dei punti misclassificati dall'iperpiano scelto come decision boundary. La funzione obiettivo da minimizzare è

$$\mathcal{D}(\beta_0, \beta) = -\sum_{i \in \mathcal{M}} m_i$$

dove $\mathcal{M}$ è l'insieme degli indici corrispondenti alle osservazioni misclassificate. Siccome tutti i data points $i \in \mathcal{M}$ presentano $m_i < 0$, allora $\mathcal{D}(\beta_0, \beta)$ sarà una quantità non negativa, proporzionale alla distanza totale dei punti misclassificati dall'iperpiano L.

La disequazione (2) è anche detta *functional margin* relativo all'osservazione x_i: più è grande questo valore, più si è fiduciosi della classificazione effettuata dato che ciò implica che l'osservazione x_i è molto lontana dal sottospazio delle osservazioni appartenenti alla classe opposta. Il functional margin relativo all'intero training set è definito come

$$F = \min_{\beta_0, \beta} f(x_i), \quad i = 1, \ldots, n$$

ed è una sorta di "worst case scenario" in cui si fissa l'ampiezza del margine pari alla distanza tra l'iperpiano e l'osservazione più vicina ad esso. Tale valore viene utilizzato come misura per confrontare tra loro i diversi possibili iperpiani definiti da β_0 e β, portando a scegliere quello per cui F è più grande.

Scelto l'iperpiano desiderato, è possibile utilizzarlo per classificare un'osservazione unseen $x_i^* = (x_{i1}^*, \ldots, x_{ip}^*)^T$: se la disequazione (2) è verificata, il data point x_i^* è stato correttamente classificato. Al contrario, si commette un errore di classificazione quando $f(x_i^*) < 0$, cioè quando:

- l'osservazione appartiene alla classe $y_i = +1$ ma $\beta_0 + x_i^T\beta < 0$ quindi viene assegnata alla classe $y_i = -1$;
- l'osservazione appartiene alla classe $y_i = -1$ ma $\beta_0 + x_i^T\beta > 0$ quindi viene assegnata alla classe $y_i = +1$.

Come detto in precedenza, l'iperpiano definito da β_0 e β non è scale invariant. Questo problema è risolto normalizzando tali parametri, ottenendo così m_i che è la distanza dell'osservazione x_i dalla sua proiezione x_i' su L, detta *geometric margin*. Così come per il functional margin, è possibile definire il geometric margin relativo all'intero training set come

$$M = \min_{\beta_0, \beta^*} m_i, \quad i = 1, \ldots, n.$$

Da questo punto in poi, per alleggerire la notazione, si considererà $\beta = \beta^*$.

In caso di classi linearmente separabili, la possibilità di ruotare o spostare un iperpiano di un infinitesimo determina l'esistenza di infinite possibili decision boundaries. La scelta più intuitiva per selezionare una di esse è scegliere l'iperpiano che presenta il geometric margin più grande, chiamato *Maximal Margin Hyperplane*. L'iperpiano così ottenuto costituisce la regola decisionale detta *Maximal Margin Classifier*, un classificatore lineare non probabilistico il quale, applicato ad osservazioni unseen, rende possibile la loro classificazione.

Le osservazioni la cui distanza da L è minima, cioè pari a $m_i = \frac{1}{\|\beta\|}$, giacciono esattamente sul margine e prendono il nome di *support vectors*. I support vectors saranno almeno due, uno per ogni classe ed il margine dipende esclusivamente da queste osservazioni speciali: un loro spostamento determinerebbe uno spostamento dell'iperpiano mentre lo spostamento di qualsiasi altra osservazione non avrebbe alcun effetto sulla sua struttura (a patto che questa non lo attraversi).

3.2.1 Problema di ottimizzazione

Considerando la specificazione (2) dell'iperpiano L ed un insieme osservazioni di training $\mathcal{D}$, la stima dei parametri β_0 e β è condotta con la risoluzione del

seguente problema di ottimizzazione in cui si cerca l'iperpiano dal geometric margin più ampio siccome, generalmente, più è largo è il margine, più basso sarà l'errore di classificazione commesso out of sample

$$\begin{aligned} \max_{\beta_0, \beta} \quad & M \\ \text{s.t.} \quad & m_i \geq M, \quad \forall i = 1, \ldots, n. \end{aligned} \tag{3}$$

Attraverso il vincolo si impone che ogni osservazione abbia distanza dall'iperpiano maggiore di M ed esattamente uguale ad M solo per i support vectors. M rappresenta l'ampiezza del margine da ogni lato dell'iperpiano ed è pari ad $M = \frac{1}{\|\beta\|}$, per un'ampiezza totale per il margine pari a $M = \frac{2}{\|\beta\|}$ la quale, essendo appunto una distanza, sarà necessariamente una quantità positiva. La formulazione (3) rappresenta un problema di ottimo non convesso, piuttosto complessa da risolvere. Per questo motivo, attraverso alcuni passaggi, è possibile trasformare (3) in un problema di ottimo convesso. Esiste una relazione tra functional e geometric margin, per la quale

$$M = \frac{F}{\|\beta\|},$$

quindi è possibile riscrivere il problema di ottimo come

$$\begin{aligned} \max_{\beta_0, \beta} \quad & \frac{F}{\|\beta\|} \\ \text{s.t.} \quad & \frac{f(x_i)}{\|\beta\|} \geq \frac{F}{\|\beta\|}, \quad \forall i = 1, \ldots, n. \end{aligned}$$

Moltiplicando entrambi i lati della disequazione per il magnitudo di β, si ottiene

$$\begin{aligned} \max_{\beta_0, \beta} \quad & \frac{F}{\|\beta\|} \\ \text{s.t.} \quad & f(x_i) \geq F, \quad \forall i = 1, \ldots, n. \end{aligned}$$

Dal momento che i parametri β_0 e β sono stati normalizzati, è possibile riscalarli senza modificare il margine. Per questo motivo si fissa $F = 1$, ottenendo così

$$\begin{aligned} \max_{\beta_0, \beta} \quad & \frac{1}{\|\beta\|} \\ \text{s.t.} \quad & f(x_i) \geq 1, \quad \forall i = 1, \ldots, n. \end{aligned}$$

A questo punto, la ricerca del massimo è trasformata in una ricerca del minimo dal momento che, come è possibile vedere, più piccolo è il magnitudo del vettore β ottimale, più grande sarà il geometric margin

$$\begin{aligned} \min_{\beta_0, \beta} \quad & \frac{1}{2}\|\beta\|^2 \\ \text{s.t.} \quad & y_i \left(\beta_0 + x_i^T \beta\right) \geq 1, \quad \forall i = 1, \ldots, n. \end{aligned} \tag{4}$$

Per convenienza, la funzione obiettivo è:

- moltiplicata per $\frac{1}{2}$ per semplificare la notazione delle derivate;
- elevata al quadrato per rimuovere la radice prevista dalla norma euclidea.

Così facendo, si è arrivati ad un problema di ottimizzazione quadratico convesso con n vincoli lineari, uno per ogni data point, più semplice da risolvere rispetto alla formulazione (3). In questo problema si va a minimizzare il magnitudo di β, inversamente proporzionale all'ampiezza M del margine, assicurandosi che ogni data point si trovi sul margine, con valore del vincolo pari ad 1 per i support vectors, oppure si trovi al di fuori di esso, caso in cui il vincolo assume valori maggiori di 1.

3.2.2 Formulazione lagrangiana

Il problema di ottimo vincolato (4), che può essere visto come

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & g(x) = 0, \end{aligned}$$

viene affrontato con il *metodo dei moltiplicatori di Lagrange*. Questo metodo individua l'ottimo della funzione obiettivo $f(x)$ nel punto in cui il suo gradiente $\nabla f(x)$ punta nella stessa direzione del gradiente $\nabla g(x)$ del vincolo $g(x)$: si assume quindi che tali gradienti siano proporzionali per una certa costante α detta *moltiplicatore di Lagrange*, ovvero che

$$\nabla f(x) = \alpha \nabla g(x).$$

La funzione detta *lagrangiana* $\mathcal{L}(x, \alpha) = f(x) - \alpha g(x)$ permette di individuare il punto di ottimo in presenza di condizioni di stazionarietà, ovvero

trovando i punti in cui il suo gradiente è nullo

$$\mathcal{L}(x, \alpha) = \nabla f(x) - \alpha \nabla g(x) = 0.$$

Applicando quanto detto al problema (4), si ha che

$$\begin{aligned} \min_{\beta_0, \beta} \quad & f(\beta) = \frac{1}{2}\|\beta\|^2 \\ \text{s.t.} \quad & g(\beta_0, \beta) = y_i \left(\beta_0 + x_i^T \beta\right) - 1, \quad \forall i = 1, \ldots, n. \end{aligned}$$

Siccome la costruzione della funzione lagrangiana prevede l'introduzione di un moltiplicatore per ogni vincolo, che in questo problema è legato alle distanze delle osservazioni dall'iperpiano, dato che le osservazioni trattate sono n, la lagrangiana presenterà n moltiplicatori. La funzione ottenuta è la seguente

$$\mathcal{L}(\beta_0, \beta, \alpha) = \frac{1}{2}\|\beta\|^2 - \sum_{i=1}^{n} \alpha_i \left[y_i \left(\beta_0 + x_i^T \beta\right) - 1\right]. \tag{5}$$

3.2.3 Ricerca dell'ottimo

Il problema di ottimo (5) è solitamente risolto nella sua formazione duale. Il *principio di dualità* enuncia che i problemi di ottimizzazione possono essere visti da due punti di vista, detti *primale* e *duale*: se il primo è un problema di minimizzazione, il secondo sarà un problema di massimizzazione, e viceversa. Ogni soluzione al problema primale sarà almeno grande quanto la soluzione del problema duale. Di conseguenza, la soluzione del problema primale agirà come limite superiore per la soluzione del duale mentre, la soluzione del duale, agirà come limite inferiore al problema del primale. Le soluzioni ai due problemi non devono necessariamente essere uguali ma, quando lo sono, si parla di *dualità forte*. Siccome il problema SVM rispetta particolari condizioni, si è in presenza di dualità forte.

Il principio di dualità nel caso di un problema di minimizzazione lagrangiano è ottenuto utilizzando esclusivamente moltiplicatori positivi per aggiungere i vincoli alla funzione obiettivo. Il problema primale è definito come

$$\begin{aligned} \max_{\alpha} \quad & \min_{\beta_0, \beta} \mathcal{L}(\beta_0, \beta, \alpha) \\ \text{s.t.} \quad & \alpha_i \geq 0 \quad \forall i = 1, \ldots, n \end{aligned}$$

La soluzione è perseguita verificando le condizioni di stazionarietà

$$\nabla \mathcal{L}(\beta_0, \beta, \alpha) = \nabla f(\beta) - \alpha \nabla g(\beta_0, \beta) = 0$$

rispetto a β

$$\frac{\partial L}{\partial \beta} = \beta - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \quad \rightarrow \quad \hat{\beta} = \sum_{i=1}^{n} \alpha_i y_i x_i = 0$$

e rispetto a β_0

$$\frac{\partial L}{\partial \beta_0} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \quad \rightarrow \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

Come menzionato precedentemente, le osservazioni x_i che assumono il ruolo di support vectors sono le uniche che contribuiscono nella definizione dell'iperpiano del margine. Siccome il vettore di coefficienti $\hat{\beta}$ che definiscono l'iperpiano è funzione di x_i, è possibile fare in modo che quanto appena detto sia vero permettendo solo ai moltiplicatori α_i relativi ai support vectors di essere non nulli.

Individuato $\hat{\beta}$ è possibile sostituirlo nella funzione obiettivo

$$W(\beta_0, \alpha) = \frac{1}{2} \left(\sum_{i=1}^{n} \alpha_i y_i x_i \right) \left(\sum_{j=1}^{n} \alpha_j y_j x_j \right) - \left(\sum_{i=1}^{n} \alpha_i y_i x_i \right) \left(\sum_{j=1}^{n} \alpha_j y_j x_j \right) - \beta_0 \sum_{i=1}^{n} \alpha_i y_i + \sum_{i=1}^{n} \alpha_i$$

Siccome $\sum_{i=1}^{n} \alpha_i y_i = 0$, tale termine si annulla e, per quanto riguarda i prodotti interni tra le quantità tra parentesi, questi sono uguali quindi possono essere riscritti come un'unica sommatoria. Riordinando i termini, si ottiene

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \\ \text{s.t.} \quad & \alpha_i \geq 0 \quad \forall i = 1, \ldots, n \end{aligned} \tag{6}$$

Questa formulazione prende il nome di *Wolfe dual* ed implementa le condizioni di *Karush-Kuhn-Tucker* (KKT), condizioni necessarie che sono una generalizzazione dell'approccio lagrangiano e permettono di avere disuguaglianze nei vincoli. Le condizioni KKT sono condizioni del primo ordine, necessarie

per avere che la soluzione al problema sia effettivamente un ottimo e, siccome si dimostra la presenza di alcune condizioni di regolarità, le condizioni KKT sono anche sufficienti nel problema SVM. Tra queste condizioni, oltre a quella di stazionarietà del punto di ottimo e oltre ai vincoli sulla funzione primale e duale, la più importante è detta di *complementary slackness*. È definita come

$$\alpha_i \left[y_i \left(\beta_0 + x_i^T \beta - 1 \right) \right] = 0 \quad \forall i = 1, \ldots, n$$

ed è proprio la condizione che porta i support vectors ad essere gli unici ad influire sul margine e sull'iperpiano.

L'importanza della formulazione (6) è legata al fatto che questa è esclusivamente funzione dei moltiplicatori di Lagrange α_i, che la soluzione del problema, ovvero l'ampiezza ottimale per il margine, dipende esclusivamente dalle osservazioni accoppiate a due a due e a come questa sia una loro trasformazione lineare. Questo sarà utile successivamente con l'introduzione delle funzioni kernel: la presenza di decision boundaries non lineari potrebbe limitare l'applicazione del classificatore; tuttavia, attraverso le funzioni kernel, è possibile sostituire il prodotto interno $x_i^T x_j$ per tutte le coppie di osservazioni con una nuova funzione che proietti i dati in uno spazio di dimensione maggiore, in cui si cercherà una separazione lineare che, riportata nello spazio di partenza, potrà anche risultare non lineare.

Una volta individuati moltiplicatori di Lagrange non nulli, è possibile utilizzarli per ricavare i valori di β_0 e β ricordando che

$$\hat{\beta} = \sum_{i=1}^{n} \alpha_i y_i x_i.$$

Una serie di valori per β_0 sono ricavati a partire dai support vectors, la cui distanza dall'iperpiano è pari ad 1

$$y_i \left(\beta_0 + \beta x_i^T \hat{\beta} \right) = 1.$$

La precedente espressione viene moltiplicata per y_i sfruttando il fatto che $y_i^2 = 1$. A questo punto si risolve per β_0, l'unica quantità incognita, e si ottengono S valori, tanti quanti sono i support vectors

$$\hat{\beta}_{0s} = y_s - x_s^T \beta, \quad \forall s = 1, \ldots, S.$$

Piuttosto che scegliere uno degli S valori a caso, è possibile:

- considerarne la media

$$\hat{\beta}_0 = \frac{1}{S} \sum_{s=1}^{S} \left(y_s - x_s^T \hat{\beta} \right);$$

- considerare la media dei support vector più vicini delle due classi [8, p.72]

$$\hat{\beta}_0 = -\frac{1}{2} \left[\max_{y_i=-1} \left(x_i^T \hat{\beta} \right) + \min_{y_i=+1} \left(x_i^T \hat{\beta} \right) \right].$$

Stimati i parametri $\hat{\beta}_0$ e $\hat{\beta}$, si ottiene la funzione $\hat{f}(x) = \hat{\beta}_0 + x_i^T \hat{\beta}$ che permette di definire il Maximal Margin Classifier $\hat{G}(x)$ come

$$\hat{G}(x) = \text{sign} \left(\hat{f}(x) \right),$$

dove la funzione sign(.) restituisce il segno del suo argomento. $\hat{G}(x)$ è stata costruita in modo tale che nessuna delle osservazioni di training cada nel margine ma questo non sarà necessariamente vero per le osservazioni unseen da classificare. È facile intuire quindi come un margine abbastanza ampio sia in grado di separare le classi, commettendo alcuni errori sul test set ma producendo comunque una buona classificazione.

3.2.4 Il problema del Maximal Margin Classifier

Il problema di quanto detto finora è che il funzionamento del Maximal Margin Classifier è basato sulla presenza di classi perfettamente separabili da una decision boundary lineare, cioè si assume l'effettiva esistenza dell'iperpiano separatore ottimo e la possibilità di separare tutte le osservazioni di una classe da quelle dell'altra attraverso un margine "abbastanza" largo. Tale assunzione, nella maggior parte dei casi, rappresenta un vincolo eccessivamente stringente che renderebbe impossibile utilizzare il classificatore. Se, nonostante ciò, questo venisse comunque implementato, l'iperpiano individuato potrebbe, in alcuni casi, presentare un margine eccessivamente piccolo il che sarebbe un problema in quanto, come detto precedentemente, la sua ampiezza è legata alla fiducia che si ha nella classificazione effettuata. In più, un modello che lascia aperta solo una piccola finestra d'errore è probabilmente un modello che è stato ecces-

sivamente adattato ai dati osservati, collocandosi così in uno degli estremi del problema del bias-variance tradeoff.

Per ovviare a questo problema, si definisce una separazione meno vincolante detta Soft Margin che caratterizza il Support Vector Classifier, generalizzazione della precedente regola decisionale.

3.3 Support Vector Classifier

Allontanarsi dalla ricerca di un iperpiano che separa perfettamente le classi permette di conferire al modello una maggiore robustezza: la possibilità di avere poche osservazioni sul lato sbagliato dell'iperpiano è controbilanciata dalla capacità di classificare le rimanenti osservazioni di training anche quando le classi non sono linearmente separabili. Tale tradeoff dà il nome a questa generalizzazione, il *Soft Margin Classifier* (o *Support Vector Classifier*). Il classificatore descritto può essere ottenuto modificando il problema di ottimizzazione descritto per il Maximal Margin Classifier

$$\begin{aligned} \min_{\beta_0,\beta} \quad & \frac{1}{2}\|\beta\|^2 \\ \text{s.t.} \quad & y_i\left(\beta_0 + \beta x_i^T \hat{\beta}\right) \geq 1 - \varepsilon_i \quad \forall i = 1, \ldots, n, \\ & \varepsilon_i \geq 0, \quad \sum_{i=1}^{n} \varepsilon_i \leq C \end{aligned} \tag{7}$$

dove ε_i è una variabile di *slack* che allenta l'effetto del vincolo sull'i-esima osservazione. In particolare:

- se $\varepsilon_i = 0$, il vincolo si riduce ad essere uguale al caso del Maximal Margin Classifier e l'osservazione i si trova sul lato corretto del margine;
- se $\varepsilon_i > 0$, l'osservazione "viola il margine";
- se $\varepsilon_i > 1$, l'osservazione si trova sul lato sbagliato dell'iperpiano e, di conseguenza, verrà classificata come appartenente alla classe sbagliata;
- $\sum_{i \in \mathcal{M}} \varepsilon_i$ è proporzionale alla distanza totale delle osservazioni misclassificate dall'iperpiano.

Siccome valori elevati di ε_i porterebbero il vincolo ad essere sempre soddisfatto, c'è bisogno di penalizzare la loro scelta. C è l'iperparametro usato per fare ciò, una costante positiva che rappresenta un tetto massimo di violazioni

del margine concesse ai data points, una sorta di budget da spendere per poter commettere errori:

- Il Maximal Margin Classifier non è altro che il caso particolare del Support Vector Classifier quando $C = 0$, caso in cui non sono concesse violazioni del margine, limitando l'applicazione del classificatore a situazioni in cui i dati sono linearmente separabili.

- Per valori piccoli di C si sarà meno tolleranti ad errori di classificazione e il margine sarà poco ampio. In questo caso, dato che si permette a poche osservazioni di violare il margine, quest'ultimo dipenderà da pochi support vectors. Un margine ristretto è sinonimo di un modello "stretto" che si concentra sull'ottenere un basso bias di previsione ma perde la capacità di generalizzare.

- Più C cresce, meno forte sarà il vincolo imposto sulla regola decisionale e maggiore sarà il numero di support vectors, arrivando ad un margine più ampio che è più indicativo di un modello "largo", che si concentra principalmente sulla capacità di classificare correttamente le osservazioni unseen, introducendo bias nella classificazione dei dati osservati.

Il problema di ottimo definito in (7) può essere espresso in maniera più conveniente dal punto di vista computazionale come

$$\begin{aligned} \min_{\beta_0, \beta} \quad & \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n} \varepsilon_i \\ \text{s.t.} \quad & \varepsilon_i \geq 0, \quad y_i\left(\beta_0 + \beta x_i^T \hat{\beta}\right) \geq 1 - \varepsilon_i \quad \forall i = 1, \ldots, n \end{aligned} \tag{8}$$

Tuttavia, così facendo, l'interpretazione di C cambia dal momento ora va ad influenzare il reciproco dell'ampiezza del margine:

- $C = +\infty$ corrisponde al caso del Maximal Margin Classifier in cui non si ammettono violazioni del margine;

- valori grandi di C portano il margine ad essere poco ampio, ammettendo poche violazioni;

- per C che si avvicina a zero, il margine si allarga sempre di più, ammettendo un numero crescente di violazioni.

C assume quindi il ruolo di parametro di tuning che gestisce il trade-off tra bias e varianza e, come tale, non ha un "valore corretto" da stimare, anche perché dipende dalla scala dello spazio delle features in cui si sta lavorando. Piuttosto, il valore più opportuno per i dati viene selezionato mediante cross-validation a partire da una griglia di valori arbitraria.

Ancora una volta, la robustezza del metodo dipende dai support vectors che, in questo caso, sono definiti come le osservazioni che si trovano esattamente sul margine oppure sul lato sbagliato rispetto a quello della loro classe di appartenenza (ma sempre sul lato corretto dell'iperpiano).

Il problema di ottimo è risolto come nel caso del Maximal Margin Classifier, cioè attraverso la *Wolfe dual function* che produce il problema

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C \quad \forall i = 1, \ldots, n \end{aligned}$$

in cui l'unica differenza è l'aggiunta del limite superiore al valore dei moltiplicatori.

3.4 Support Vector Machines

Il Support Vector Classifier affronta il problema riscontrato implementando il Maximal Margin Classifier, che si vedeva incapace di classificare i dati se questi non erano perfettamente separabili in maniera lineare. Questo problema viene risolto allentando il vincolo sulla perfetta linearità della separazione ed ammettendo la possibilità di commettere errori di classificazione.

Spesso, tuttavia, questa soluzione potrebbe portare ad un eccessivo numero di errori di classificazione: si tratta del caso in cui i dati sono separabili esclusivamente da decision boundaries fortemente non lineari per le quali un iperpiano semplificherebbe troppo la reale separazione tra i data points. In realtà, il vero problema è che la separazione tra i data points non è visibile a causa della dimensione p dello spazio in cui sono situati i dati, dimensione che potrebbe essere troppo piccola per evidenziare il confine tra le classi. Un'idea potrebbe essere quella di espandere lo spazio delle features osservate con loro trasformazioni arbitrarie, portando i dati in uno spazio di dimensione maggiore in cui cercare la decision boundary lineare che, riportata nello spazio originale, potrà apparirà non lineare.

Il procedimento è il seguente:

- si scelgono R trasformazioni $h_r(x),\ \forall r = 1, \ldots, R$;
- ognuna di esse è calcolata per ogni osservazione, ossia si calcola $h(x_i) = (h_1(x_i), h_2(x_i), \ldots, h_R(x_i)),\ \forall i = 1, \ldots, n$;
- si produce la funzione non lineare $\hat{f}(x_i) = \hat{\beta}_0 + h\left(x_i^T\right)\hat{\beta}$;
- la classificazione è data da $\hat{G}(x_i) = \text{sign}\left(\hat{f}(x_i)\right)$.

La criticità sta quindi nella scelta delle trasformazioni da applicare ai dati.

Considerata come trasformazione il semplice prodotto interno tra due vettori x_i e x_j, definito come

$$\langle x_i, x_j \rangle = x_i^T x_j = \sum_{l=1}^{p} x_{il} x_{jl},$$

il Support Vector Classifier è dato da

$$\hat{G}(x_i) = \text{sign}\left(\hat{\beta}_0 + \sum_{i=1}^{n} \alpha_i y_i \langle x_i, x_j \rangle\right),$$

corredato da n parametri $\alpha_1, \ldots, \alpha_n$, uno per ogni osservazione, e da $\binom{n}{2}$ prodotti interni in totale.

Dato che, come menzionato precedentemente, si dimostra che $\alpha_i = 0$ per tutte le osservazioni che non sono support vectors, in realtà il numero di prodotti interni da calcolare è ancora più basso. Considerato quindi l'insieme $\mathcal{S}$ di indici relativi alle osservazioni che assumono il ruolo di support vectors, il Support Vector Classifier può essere definito come

$$\hat{G}(x_i) = \text{sign}\left(\hat{\beta}_0 + \sum_{i \in \mathcal{S}} \alpha_i y_i \langle x_i, x_j \rangle\right).$$

3.5 Funzioni kernel

Un *kernel* $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ è una funzione tale che, considerata una trasformazione $\phi : \mathcal{X} \to \mathcal{V}$,

$$K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle_{\mathcal{V}}.$$

Le funzioni kernel promuovono una modalità di calcolo più efficiente: trasformare i dati originali e poi calcolare il prodotto interno tra i dati trasformati

potrebbe richiedere l'utilizzo di una grande quantità di risorse, sia dal punto di vista del tempo impiegato per le computazioni, dato l'elevato numero di osservazioni con cui solitamente si lavora, sia dal punto di vista della memoria necessaria, dal momento che queste trasformazioni possono far esplodere velocemente il numero di dimensioni dello spazio in cui si collocano i dati trasformati. Attraverso le funzioni kernel è possibile aggirare questi problemi. In parole povere, le funzioni kernel restituiscono uno scalare uguale al prodotto interno tra le proiezioni di due vettori nello spazio definito da ϕ. Il modus operandi che prevede questa trasformazione "on the fly", senza il bisogno di passare esplicitamente per questo nuovo spazio, è detto *kernel trick*.

Il semplice prodotto interno tra due vettori, chiamato *kernel lineare* in questo ambito, è solo una delle possibili funzioni kernel che possono essere utilizzate nel Support Vector Classifier. Infatti, è possibile generalizzare l'espressione di $\hat{G}(x_i)$ al caso della generica funzione kernel $K(.)$

$$\hat{G}(x_i) = \text{sign}\left(\hat{\beta}_0 + \sum_{i \in \mathcal{S}} \alpha_i y_i K(x_i, x_j)\right).$$

Le funzioni kernel più utilizzate sono il kernel lineare, il kernel polinomiale di grado d e il kernel RBF.

3.5.1 Kernel lineare

Il prodotto interno

$$K_L(x_i, x_j) = \langle x_i, x_j \rangle,$$

se utilizzato come funzione kernel, è detto *kernel lineare* ed è proporzionale alla similarità tra le osservazioni calcolata in termini di *Correlation Similarity*, detta anche *Cosine Similarity*. Si dimostra che tale misura calcola la distanza tra due osservazioni in termini di coordinate polari, legate all'angolo formato dai due vettori nello spazio euclideo: più è ampio l'angolo formato dai due vettori più gli oggetti sono considerati dissimili.

Formalmente, dati i vettori $x_i, x_j \in \mathbb{R}^p$, la *Cosine Similarity* $s_c(x_i, x_j)$ è calcolata come

$$s_c(x_i, x_j) = \frac{\sum_{l=1}^{p} x_{il} x_{jl}}{\|x_i\| \|x_j\|},$$

ed il suo punto di forza è la capacità di considerare come simili osservazioni molto distanti tra loro in termini di distanza euclidea oppure come dissimili

osservazioni in realtà vicine, cosa possibile dal momento il prodotto interno viene diviso per il prodotto delle loro lunghezze.

3.5.2 Kernel polinomiale di grado d

Quando il Support Vector Classifier viene usato in combinazione ad un kernel non lineare si parla di *Support Vector Machines*. Il kernel polinomiale di grado d è definito come

$$K_{poly(d)}(x_i, x_j) = (c + \langle x_i, x_j \rangle)^2$$

- quando $d = 1$ e $c = 0$, il classificatore SVM si riduce ad essere uguale al Support Vector Classifier che usa un kernel lineare;
- quando $d > 1$, i dati vengono proiettati in uno spazio di dimensione maggiore utilizzando polinomi di grado d.

All'aumentare del valore di d si ottiene una decision boundary più flessibile e, di conseguenza, un adattamento più stretto del modello ai dati, rischiando di generare overfitting e perdere la capacità di generalizzare.

3.5.3 Kernel RBF

Il *Radial Basis Function kernel* è definito come

$$K_{RBF}(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}$$

Dalla definizione di questa funzione kernel è possibile notare la sua relazione con la distanza euclidea tra due osservazioni x_i e x_j ed il fatto che presenta una forma "bell shaped", il che le conferisce anche il nome di *Gaussian kernel*. Tenendo conto che γ è una costante positiva il cui crescere determina una maggiore non linearità della regola decisionale, dal momento che γ è inversamente proporzionale all'ampiezza della curva a campana:

- se le osservazioni x_i e x_j sono vicine in termini di distanza euclidea, tale distanza sarà piccola e di conseguenza $K(x_i, x_j)$ sarà grande;
- se le osservazioni x_i e x_j sono lontane in termini di distanza euclidea, tale distanza sarà grande e di conseguenza $K(x_i, x_j)$ sarà piccolo.

Il valore più opportuno per γ dipende dai dati quindi viene individuato mediante cross-validation.

Come menzionato precedentemente, le funzioni kernel sono utilizzate per quantificare la similarità tra coppie di osservazioni perché sono costruite in modo che il valore che assumono sia tanto piccolo quanto più le osservazioni sono simili. Il kernel RBF utilizza solo le osservazioni più vicine ad una certa test observation x^* per determinare la sua classe di appartenenza, assumendo quindi un comportamento molto locale rispetto ad ogni data point. Data questa concentrazione locale, il kernel RBF è in grado di definire decision boundaries fortemente non lineari, molto di più rispetto ad una "semplice" separazione quadratica.

Le funzioni kernel sono necessarie al raggiungimento dell'obiettivo del classificatore SVM: aumentare la dimensionalità dei dati per cogliere separazioni precedentemente nascoste. Questo può tuttavia essere concettualmente e computazionalmente molto intensivo dal momento che si richiederebbe la considerazione di tutte le possibili trasformazioni delle features e di tutte le loro interazioni. Il kernel RBF è particolarmente adatto a tale compito dal momento che è in grado di considerare le infinite interazioni tra le features senza effettivamente portare i dati nello spazio delle features espanso, cosa che in questo caso sarebbe addirittura impossibile dal momento che si tratta di spazi con infinite dimensioni. Data questa versatilità, il kernel RBF è solitamente la miglior scelta "out of the box".

Per vedere come questo sia possibile basta considerare la definizione stessa del kernel RBF ed espanderla [4], fissando $\gamma = \frac{1}{2}$ senza perdità di generalità

$$\begin{aligned} K_{RBF}(x_i, x_j) &= e^{-\gamma \|x_i - x_j\|^2} \\ &= \exp\left\{-\frac{1}{2}\langle (x_i - x_j), (x_i - x_j)\rangle\right\} \\ &= \exp\left\{-\frac{1}{2}\left[x_i^T x_i - 2x_i^T x_j + x_j^T x_j\right]\right\} \\ &= \exp\left\{-\frac{1}{2}\left[x_i^T x_i + x_j^T x_j\right]\right\} \exp\left\{x_i^T x_j\right\} \\ &= \exp\left\{-\frac{1}{2}\left[\|x_i\|^2 + \|x_j\|^2\right]\right\} \exp\left\{\langle x_i, x_j\rangle\right\} \end{aligned}$$

Per evidenziare la quantità all'esponente, si somma e si sottrae c senza influenzare la formula, ottenendo

$$K_{RBF}(x_i, x_j) = \exp\left\{-\frac{1}{2}\left[\|x_i\|^2 + \|x_j\|^2\right]\right\} \exp\left\{c + \langle x_i, x_j\rangle\right\} \exp\left\{-c\right\}.$$

Fissato

$$C = \exp\left\{-\frac{1}{2}\left[\|x_i\|^2 + \|x_j\|^2\right]\right\} \exp\{-c\},$$

si ottiene

$$K_{RBF}(x_i, x_j) = Ce^{(c+\langle x_i, x_j\rangle)}$$

la cui espansione in serie di Taylor è

$$C\sum_{m=0}^{\infty} \frac{(c + \langle x_i, x_j\rangle)^m}{m!} = C\sum_{m=0}^{\infty} \frac{K_{poly(m)}(x_i, x_j)}{m!}.$$

Il kernel RBF è quindi definito come una somma infinita di kernel polinomiali la cui complessità aumenta all'aumentare di m. Tale proprietà è proprio ciò che consente a questo kernel di definire decision boundaries fortemente non lineari in circostanze in cui la struttura dei dati è molto complessa.

4 Data pre-processing

Dopo aver scaricato i dati dal *UCI Machine Learning Repository* e convertiti in formato `csv` attraverso opportuni tools online, questi sono stati importati in R. In seguito ad un'ispezione preventiva del dataset, ho codificato le variabili con il tipo di dato opportuno: la variabile di risposta `class` in tipo `factor` e tutti i predittori in tipo `numeric`.

Una superficiale esplorazione dei dati ha poi rivelato la presenta di valori "?", probabilmente dovuti al fatto che non tutti gli indici sono stati calcolati per tutti gli anni del periodo di previsione per tutte le imprese analizzate. Per questo motivo, in assenza di ulteriori inforamzioni, sono stati considerati come valori mancanti ed eliminati tutti i record che ne presentassero anche solo uno tra i predittori.

4.1 Riduzione PCA

Trattare dati dall'alta dimensionalità può risultare particolarmente oneroso per i metodi di machine learning quindi potrebbe essere utile applicare tecniche di riduzione al fine di migliorarne l'efficacia. I dati presentano una struttura di correlazione piuttosto forte, probabilmente dovuta al fatto che molti degli indici sono costruiti come combinazioni algebriche delle medesime quantità [14, p.1], quindi ricorro alla riduzione tramite Componenti Principali per indivi-

duare il minor numero di dimensioni che permettano di preservare la maggiore informazione possibile.

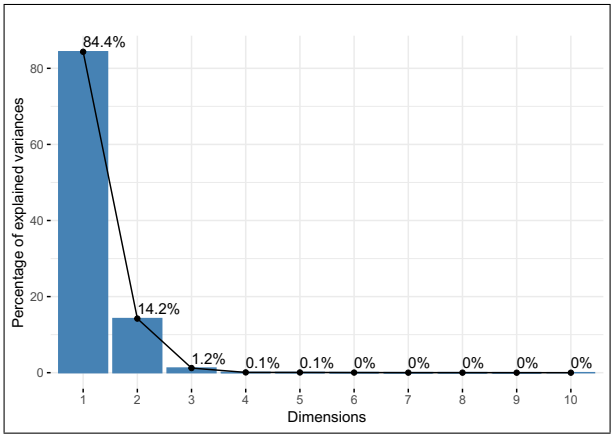

Figura 1: Screeplot componenti principali

Come è possibile dedurre dallo screeplot in Figura 1, per cumulare il 99% della variabilità osservata basta mantenere solo le prime tre componenti principali, il che è un ottimo risultato in quanto se ne stanno scartando la maggior parte. Per questo motivo, a partire dalle componenti preservate, i dati vengono riportati nello spazio di partenza costruendone una versione ridotta su cui lavorare da questo punto in poi.

5 Risultati dell'analisi

Trattandosi di un problema di tipo supervisionato, la selezione degli iperparametri può essere condotta attraverso Cross Validation massimizzando il valore di determinati indici di performance. Uno dei metodi per fare ciò in R è attraverso l'utilizzo della libreria `caret` che mette a disposizione apposite funzioni per controllare i vari aspetti dell'addestramento.

5.1 Dettagli sull'implementazione software

In questa sezione viene spiegata la procedura di addestramento condotta attraverso la libreria `caret`, un pacchetto volto alla creazione di modelli predittivi. La funzione `trainControl()` è la funzione di questo pacchetto che permette di definire gli aspetti computazionali del training vero e proprio, che sarà a carico della funzione `train()`: la sua corretta specificazione è fondamentale e necessaria per ottenere un buon addestramento.

La tipologia di Cross Validation condotta per ottenere stime consistenti dall'addestramento di un modello e selezionare i valori più performanti per i suoi iperparametri è la *Repeated k-fold CV*. Il suo funzionamento è il seguente:

1. si dividono i dati in `k` folds e se ne seleziona uno come test set, mentre i rimanenti `k`-1 saranno usati come training set;

2. sul training set viene addestrato il modello per ogni combinazione dei suoi iperparametri;

3. sul test set viene valutato il modello sulla base di una metrica di performance scelta;

4. i punti precedenti vengono ripetuti finché ognuno dei `k` folds non viene usato come test set, dopodiché si calcola la media dei `k` valori della metrica di performance;

5. l'intera procedura è ripetuta `times` volte.

È facile vedere come un valore di `times` più grande permetta di ottenere stime più consistenti, al prezzo di un addestramento più oneroso. Terminate le iterazioni, la combinazione di valori degli iperparametri che massimizza la metrica di performance scelta sarà quella che definisce il modello ottimale. Questa procedura è implementata attraverso il parametro `method = "repeatedcv"`.

Per avere un minimo controllo sui folds utilizzati dalla Cross Validation, questi vengono preventivamente costruiti con la funzione `createMultiFolds()`, con parametri `k=10`, `times=10`, dopo aver fissato un seed per rendere riproducibile la procedura randomica. Tali folds sono poi passati al parametro `index` di `trainControl()`.

Come evidenziato nella sezione (2.4), i dati osservati sono afflitti dal problema delle classi sbilanciate. La tecnica di oversampling *SMOTE*, presentata in tale sezione, è implementata in combinazione con un undersampling della majority class attraverso il parametro `sampling`. Inoltre, come suggerito precedentemente, l'indice *ACC*, utilizzato di default da `caret`, non è il più idoneo per confrontare le performance di modelli costruiti su dati con classi sbilanciate; piuttosto, è preferibile far riferimento alla curva *ROC* con relativa *AUC*. Per fare in modo che questa sia la procedura seguita, è necessario specificare l'opzione `classProbs = TRUE` così che le posterior class probabilities, ovvero le probabilità che un'osservazione appartenga alle rispettive classi dati i valori

osservati per le features, vengano conservate al fine di poterle usare come valori soglia per costruire la curva *ROC*.

Il cuore dell'addestramento con `caret` è la funzione `train()` che permette di definire una serie di settaggi incentrati sulle caratteristiche del modello vero e proprio. Siccome questo lavoro si concentra sullo studio delle capacità del metodo SVM, tutte le variabili presentate nella sezione (2.3) verranno usate indiscriminatamente al fine di permettere al modello di disporre della maggiore quantità di informazione possibile per produrre una buona classificazione.

Il parametro `method` è utilizzato per dichiarare il tipo di modello da addestrare: per l'addestramento di Support Vector Machines è possibile ricorrere alle impostazioni `"svmLinear"`, `"svmPoly"` e `"svmRadial"` per specificare il tipo di funzione kernel. Ognuna di esse è corredata dai suoi iperparametri presentati nella sezione (3.5):

- il costo C, comune a tutte le specificazioni;
- il grado d e `scale` per il kernel polinomiale;
- il parametro γ per il kernel RBF (chiamato `sigma` in `caret`);

I valori utilizzati per gli iperparametri sono basati su quelli che sarebbero stati usati automaticamente specificando `tuneLength = 5`. Tuttavia, ho preferito elencarli a mano con una combinazione di `tuneGrid` e della funzione `expand.grid()` in modo da definire tutte le possibili combinazioni ottenute considerando `C=0.25,0.50,1,2,4` come costo per tutti i kernel e `d=2,3,4` per il kernel polinomiale, evitando `d=1` che non è altro che il kernel lineare. Inoltre, i parametri `scale` del kernel polinomiale e `sigma` del kernel RBF sono scelti automaticamente da `caret`.

Essendo il classificatore SVM un metodo "distance based", esso dipende dalla scala di misurazione dei dati: siccome le variabili incluse potrebbero essere misurate su scale diverse, i dati vengono standardizzati specificando `preProcess = c("center", "scale")`.

Infine, le impostazioni definite con la funzione `trControl()` sono gestite dal parametro `trControl` e la metrica di performance è opportunamente specificata con `metric = "ROC"`.

5.2 Interpretazione risultati

I risultati dell'addestramento condotto sono riportati nelle seguenti tabelle.

Cost	***AUC***	**Sens**	**Spec**
0.25	**0.6847318**	**0.5443383**	**0.7484848**
0.50	0.6115629	0.4601672	0.7394707
1	0.5760355	0.4171945	0.7530655
2	0.5877106	0.4079509	0.7813017
4	0.6025085	0.4155424	0.8013304

Tabella 2: Training kernel lineare

La miglior configurazione del kernel lineare in termini di AUC è ottenuta con $Cost = 0.25$. La *sensitivity* relativa a questa specificazione è, tuttavia, piuttosto bassa, il che indica che il modello non è molto idoneo alla classificazione delle osservazioni appartenenti alla minority class.

Degree	**Cost**	***AUC***	**Sens**	**Spec**
2	0.25	0.7179283	0.5016284	0.8074380
2	0.50	0.7163909	0.4706288	0.8142584
2	1	0.7142556	0.4407657	0.8208182
2	2	0.7156873	0.4468743	0.8258953
2	4	0.7158923	0.4532194	0.8309458
3	0.25	0.7121640	0.4930317	0.7904453
3	0.50	0.7122227	0.4407593	0.8158627
3	1	0.7155001	0.4346982	0.8210575
3	2	0.7151145	0.4305659	0.8278237
3	4	0.7092223	0.4200820	0.8343018
4	0.25	0.7134301	0.5097446	0.7786042
4	0.50	0.7167958	0.4939008	0.7883117
4	**1**	**0.7208806**	**0.4945981**	**0.7934803**
4	2	0.7100047	0.5055045	0.7782931
4	4	0.7061745	0.5307630	0.7658627

Tabella 3: Training kernel polinomiale, $scale = 0.001$

In combinazione ad un costo pari a 1, usare un kernel di quarto grado permette alla formulazione polinomiale di ottenere un adattamento più stretto ai dati grazie alla maggiore complessità, giungendo così ad un'area pari a 0.72 sotto la curva ROC. Nonostante ciò, anche questa specificazione soffre del problema della precedente, cioè anch'essa presenta una *sensitivity* piuttosto bassa, fattore determinante nel caso di classi sbilanciate.

Cost	*AUC*	Sens	Spec
0.25	**0.7595303**	**0.8230470**	**0.5802727**
0.50	0.7577746	0.8278614	0.5627273
1	0.7527355	0.8314457	0.5390909
2	0.7445180	0.8315480	0.4995455
4	0.7255061	0.8308975	0.4813544

Tabella 4: Training kernel RBF, $\gamma = 5.526677$

Il kernel RBF con costo pari a 0.25 si presenta come il più performante sia in termini di *AUC* che di *sensitivity*, riuscendo a classificare correttamente l'82.3% delle osservazioni appartenenti alla classe dei positivi.

I risultati presentati nelle tabelle possono essere graficamente visualizzati in figura (2), considerando il valore dell'indice *AUC* al variare del parametro di costo.

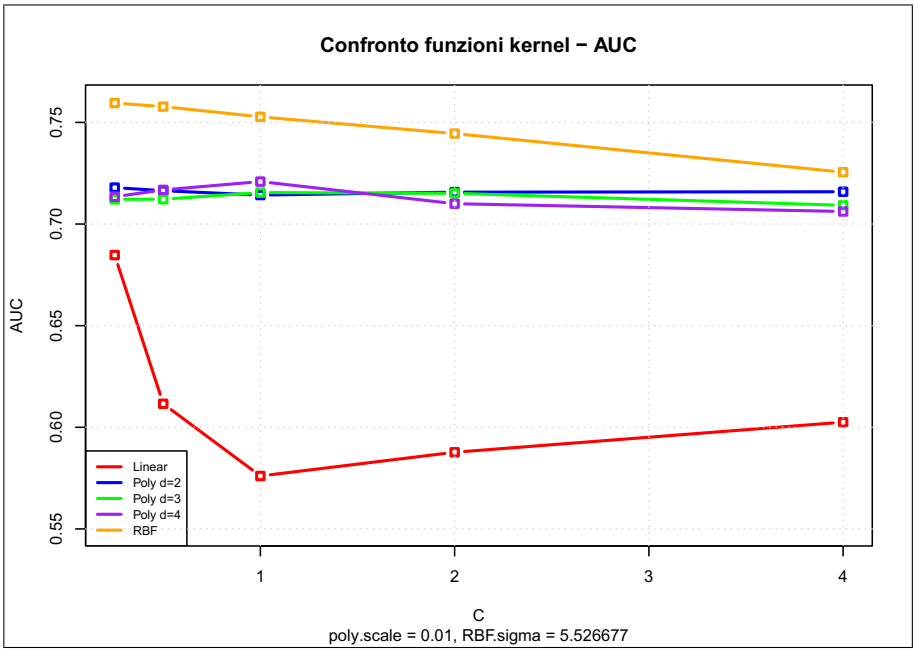

Figura 2: Performance *AUC* funzioni kernel

Da questa visualizzazione si evince la superiorità del kernel RBF, solitamente considerato come la migliore alternativa "out of the box". Prevedibilmente, la performance peggiore è raggiunta dal kernel lineare che è probabilmente incapace di cogliere a pieno la complessa struttura della decision boundary tra le classi dovuta alla presenza di un elevato numero di predittori. Nonostante questo sia vero, anche la prestazione più scadente di questa specificazione riesce

ad ottenere un valore AUC maggiore di 0.5, il che permette di considerare tale classificatore come almeno preferibile al semplice "random guessing".

Le curve ROC per le migliori specificazioni di ogni kernel sono visibili in figura (3): l'area più grande è sottesa dalla curva ROC del kernel RBF ma anche il kernel polinomiale di quarto grado (la miglior alternativa tra i polinomiali) e quello lineare riescono ad ottenere una buona performance.

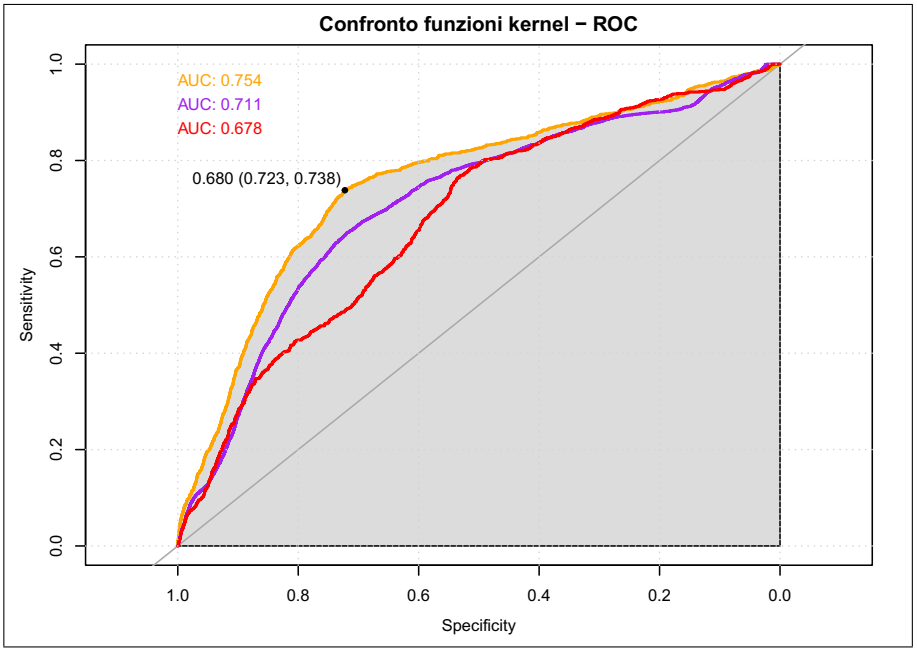

Figura 3: Curve ROC funzioni kernel

Il valore soglia per la curva ROC che permette di ottenere la migliore combinazione di $specificity$ e $sensitivity$ è 0.680: in corrispondenza di questo valore soglia, il 72.3% dei positivi ed il 73.8% dei negativi sono correttamente classificati.

6 Conclusioni

Prevedere lo stato di bancarotta delle imprese è un problema di classificazione di primaria importanza nel mondo della finanza dato il suo forte impatto sulla società e sui suoi membri. Questo lavoro considera uno dei numerosi metodi di classificazione presenti in letteratura, il metodo SVM per la classificazione binaria, ed evidenzia il suo funzionamento, i suoi punti di forza e le sue carenze, applicandolo ai dati prelevati dal database EMIS. Nonstante la sua semplicità, il metodo SVM si dimostra efficace anche in presenza di classi non linearmente

separabili grazie ad una gamma di funzioni kernel le cui proprietà sono facilmente regolabili attraverso opportuni iperparametri, a differenza di altri metodi che possono risultare più onerosi da addestrare. Inoltre, con opportune specificazioni è anche possibile estendere il campo di applicazione di questo metodo a problemi multiclass in cui le osservazioni campionarie risultano appartenere a più di due categorie. Come molte situazioni in natura, le classi osservate sono fortemente sbilanciate, con solo il 3% rappresentanti imprese in stato di bancarotta. Questo forte sbilanciamento influenza molto le capacità predittive del metodo SVM, portandolo ad ottenere una precisione anche maggiore del 90% ma classificando erroneamente tutte le imprese che risultano in bancarotta. Di conseguenza, viene adottata la tecnica di ricampionamento SMOTE per bilanciare tale effetto, in aggiunta alla scelta di metriche considerate più idonee rispetto alla semplice accuratezza. L'utilizzo del classificatore SVM in questo lavoro ha previsto l'addestramento e il confronto delle principali specificazioni per le funzioni kernel: il kernel lineare, il kernel polinomiale di grado d ed il kernel RBF. L'addestramento condotto per il tuning degli iperparametri evidenzia la versatilità del kernel RBF, che ottiene i migliori risultati in termini di AUC quando impostato con parametri $\gamma = 5.526677$ e $C = 0.25$, seguito poi dal kernel polinomiale, che raggiunge la migliore performance con grado $d = 4$ con $C = 1$, ed infine dal kernel lineare con $C = 0.25$, prestazioni probabilmente giustificate dalla presenza di dati dalla struttura piuttosto complessa, dovuta all'elevato numero di predittori.

Riferimenti bibliografici

[1] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[2] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2, pages 129–135, 417–426. Springer, 2009.

[3] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An introduction to statistical learning*, volume 112, pages 367–383. Springer, 2013.

[4] A. Kowalczyk. *RBF Kernel proof*. https://pages.cs.wisc.edu/~matthewb/pages/notes/pdf/svms/RBFKernel.pdf, last accessed: May 2023.

[5] A. Kowalczyk. *SVMs - An overview of Support Vector Machines*. https://www.svm-tutorial.com/2017/02/svms-overview-support-vector-machines/, last accessed: May 2023.

[6] A. Kowalczyk. *Support vector machines succinctly*, Oct 2017. https://www.syncfusion.com/succinctly-free-ebooks/support-vector-machines-succinctly, last accessed: May 2023.

[7] MIT-OpenCourseWare. *16. Learning: Support Vector Machines*. https://youtu.be/8NYoQiRANpg, last accessed: May 2023.

[8] A. Ng. *CS229 Lecture Notes*. http://cs229.stanford.edu/notes2022fall/main_notes.pdf, Chapter 6 - Support vector machines, last accessed: May 2023.

[9] B. T. Smith. *Lagrange Multipliers Tutorial in the Context of Support Vector Machines*. PhD thesis, Faculty of Engineering and Applied Science Memorial University of Newfoundland, St. John's, Newfoundland, Canada, June 2004.

[10] Stanford-Online. *Lecture 6 - Support Vector Machines | Stanford CS229: Machine Learning Andrew Ng (Autumn 2018)*. https://youtu.be/lDwow4aOrtg, last accessed: May 2023.

[11] Stanford-Online. *Lecture 7 - Kernels | Stanford CS229: Machine Learning Andrew Ng (Autumn 2018)*. https://youtu.be/8NYoQiRANpg, last accessed: May 2023.

[12] V. N. Vapnik. *The nature of statistical learning theory*. Statistics for Engineering and Information Science. Springer-Verlag, New York, second edition, 2000.

[13] P. H. Winston. *MIT Artifial Intelligence Lecture Notes*. https://ocw.mit.edu/courses/6-034-artificial-intelligence-fall-2010/resources/mit6_034f10_svm/, last accessed: May 2023.

[14] M. Zikeba, S. K. Tomczak, and J. M. Tomczak. Ensemble boosted trees with synthetic features generation in application to bankruptcy prediction. *Expert Systems with Applications*, pages 2–4, 11–14, 2016.