Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

REGRESSIONE LINEARE E LOGISTICA PER LA PREDIZIONE DI LOW ACHIEVEMENT SCOLASTICO

Relatore:
Chiar.mo Prof.
Gabbrielli Maurizio
Correlatore:
Dottor
Zanellati Andrea

Presentata da:
Tozzi Lorenzo

Sessione II
Anno Accademico 2021/2022

Alla mia famiglia

Introduzione

L'obiettivo che si pone questa tesi è quello di creare dei modelli di machine learning tradizionali per la predizione di low achievement scolastico, da paragonare con i modelli più complessi utilizzati da A.Zanellati et al.[19] nel loro paper. Si cercherà di capire se un aumento di complessità nel modello implichi o meno un miglioramento di performance in questo tipo di problema.

Un fenomeno molto diffuso a scuola con delle conseguenze nel lungo periodo è il low achievement. Secondo uno studio condotto dalle INVALSI[1],il 20% degli studenti italiani ha avuto delle performance inferiori rispetto ai livelli sufficienti e alcuni di essi hanno poi anche abbandonato la scuola.[1]

Verranno implementati dei modelli basati su logistic regression e linear regression per la previsione di low achievement nelle prove INVALSI di matematica di seconda superiore, a partire da quelle di quinta elementare. I risultati di questi modelli saranno poi comparati con i risultati dei modelli basati su random forest e reti neurali, ottenendo delle performance paragonabili, quindi soddisfacenti.

Struttura della tesi

Nel primo capitolo vengono introdotte le basi del machine learning, ponendo enfasi su linear regression e logistic regression, su alcune tecniche utilizzate per lo sviluppo dei modelli e sulle principali problematiche che si possono riscontrare.

[1]Istituto nazionale per la valutazione del sistema educativo di istruzione e di formazione.

Nel secondo capitolo viene riassunto il lavoro e i risultati ottenuti da A.Zanellati et al.[19] oltre ad introdurre il dataset INVALSI.

Nel terzo capitolo vengono riportati tutti i modelli sviluppati per questa tesi, con relativi risultati.

Infine, nelle conclusioni verranno paragonati i risultati da me ottenuti con quelli ottenuti da A.Zanellati et al.[19] seguiti da alcune considerazioni.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione al Machine Learning

Per la realizzazione di questa tesi sono state utilizzate alcune tecniche di machine learning. Nel primo capitolo verrà fatta una breve introduzione dell'argomento e degli algoritmi utilizzati in questa tesi.

Murphy, nel suo libro *Machine Learning: A Probabilistic Perspective* dà una chiara definizione di machine learning: "Un insieme di metodi che riescono in automatico a riconoscere dei pattern nei dati e ad utilizzarli per predirre dati futuri o per effettuare altri tipi di decisione". [1] [2]
Da ciò si deduce che i metodi di machine learning sono capaci di imparare a riconoscere dei pattern.

1.1 Apprendimento

Che cosa significa che un algoritmo è in grado di imparare? Secondo Mitchell: "Si dice che un algoritmo impara da un'esperienza E a compiere un certo task T con un misuratore di performance P, se le performance nel

[1] a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty

compiere il task T, misurate con P, migliorano con l'esperienza E." [3] [2]

1.1.1 Task

I task che possono essere appresi da algoritmi di Machine Learning sono tantissime e di seguito ne verranno elencati alcuni:

- **Classification**: Il programma cerca di capire la classe di appartenenza di un determinato input; per esempio, riconoscere se un tumore è benigno o maligno.
- **Regression**: L'algoritmo cerca di predire un valore continuo; ad esempio, potrebbe indovinare il prezzo di mercato di una casa.
- **Clustering**: Il programma cerca di raggruppare insieme i dati simili; per esempio, trovare pazienti che hanno condizioni cliniche simili.
- **Association**: L'algoritmo trova eventi o elementi che spesso avvengono in contemporanea; tipo gli oggetti che spesso vengono comprati assieme nei supermercati.
- **Anomaly detection**: Il programma cerca di scoprire anomalie e casistiche non ricorrenti; come nel caso di frode della carta di credito.
- **Sequence Mining**: L'algoritmo cerca di prevedere il prossimo evento, come indovinare che cosa si cliccherà a un certo punto su un sito internet.
- **Reccomendation systems**: Il programma associa le preferenze di una persona con quelle di altre con simili gusti, in modo da poter raccomandare nuovi oggetti nel modo più preciso possibile; come succede

[2] A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

nelle principali piattaforme di streaming video quando consigliano dei nuovi contenuti da vedere.

1.1.2 Esperienze

Gli algoritmi di machine learning, nella maggior parte dei casi, fanno esperienza tramite dataset. In base a com'è composto il dataset, possiamo suddividere i programmi in due grandi categorie:

1. **Supervised Learning**: L'apprendimento da parte dei programmi appartenenti a questa categoria viene supervisionato. In che modo? Tramite una fase di "allenamento" in cui si insegna all'algoritmo a compiere un determinato task. Successivamente, l'algoritmo sarà in grado di replicare il task con una certa accuratezza.
   Ma come si insegna all'algoritmo un determinato task? Si utilizza un dataset etichettato e composto da esempi di possibili input del programma affiancati (o etichettati) dai risultati che ci si aspetta in output. Riprendendo l'esempio dato precedentemente per il task di classification, avremo un dataset composto da foto di tumori, etichettati come o benigni o maligni.

2. **Unsupervised Learning**: L'apprendimento da parte di questi programmi, invece, non viene supervisionato, quindi il modello deve cercare da solo le informazioni. In questo caso il dataset non ha alcuna etichetta. Per esempio, in un task di clustering, l'algoritmo impara da solo a dividere gli input e a raggrupparli per determinate caratteristiche.

1.1.3 Misurazioni

Poichè possono esistere performance di diverso tipo, ogni task ha i propri metodi quantitativi per misurare le performance. Ad esempio, per un task di classificazione binaria abbiamo quattro misure comunemente usate, tutte basate sulle definizioni riportate in tabella 1.1di cui discutiamo di seguito.

Quando un esempio viene predetto come vero ed è davvero vero, lo defininiamo come true positive(tp); quando un valore viene predetto vero, ma in realtà è falso lo definiamo come false positive (fp); quando un valore viene predetto falso ma in realtà è vero, lo definiamo come false negative (fn); infine, quando un valore viene predetto come falso e lo è davvero, allora lo definiamo come true negative (tn). tp, fp, fn, tn sono tutti dei valori assoluti. La somma di tp e fn sono i valori realmente veri (rp), mentre la somma tra fp e tn sono i valori realmente falsi (rn). La somma di tp e fp sono i valori predetti veri (pp), mentre la somma di fn e tn sono i valori predetti falsi (pn). La somma tra pp e pn deve essere 1, come anche la somma tra rp e rn. [4]

Possiamo ora introdurre le quattro principali metriche per la misurazione delle performance in un problema di classificazione binaria:

1. $Accuracy = tp + tn$ è la percentuale delle classificazioni avvenute correttamente.

2. $Precision = tp/pp$ è la percentuale dei valori predetti realmente veri, sul totale di tutti i valori predetti come veri.

3. $Recall = tp/rp$ è la percentuale dei valori predetti realmente veri sul totale di tutti i valori davvero veri.

4. $f1 = 2 * \frac{precision*recall}{precision+recall}$ ed è la media armonica di precision e recall.

	Realmente Veri	**Realmente Falsi**	
Predetti Veri	tp	fp	pp
Predetti Falsi	fn	tn	pn
	rp	rn	1

Tabella 1.1: Tabella di contingenza per misure di problemi di classificazione [4]

1.2 Linear Regression

La linear regression è un algoritmo di supervised learning che serve a risolvere un problema di regressione. Il programma :

- Prende in input una matrice $\mathbf{X} \in \mathbb{R}^{m \times n}$,dove n rappresenta il numero di esempi su cui verranno effettuate previsioni, mentre m rappresenta l'insieme delle feature che verranno utilizzate per compiere il task.

- Restituisce in output un vettore $\hat{y} \in \mathbb{R}^n$ calcolato nel seguente modo:

$$\hat{y} = \mathbf{w}^T \mathbf{X} \tag{1.1}$$

  Dove $\mathbf{w} \in \mathbb{R}^m$ è un vettore di parametri.

Ponendo $n = 1$ e svolgendo la moltiplicazione scalare otteniamo:

$$\hat{y} = \sum_i w_i x_i \tag{1.2}$$

I parametri sono dei valori che vengono utilizzati dal modello per controllare l'output. Come possiamo osservare in 1.2 ,ogni parametro w_i è moltiplicato per una feature x_i. Quindi il valore di ogni parametro w_i determina quanto la feature x_i influisce sulla predizione. Infatti se a una feature x_j viene assegnato un parametro w_j di valore zero, la feature x_j non avrà alcun impatto sulla predizione finale. Viceversa, se il valore assoluto del parametro w_j è elevato, allora la feature x_j avrà una grande influenza sul risultato finale.[5]
Per questo motivo ci si riferisce al vettore $\mathbf{w}$ come al vettore dei pesi (weights).

L'algoritmo di linear regression ha anche un'interpretazione geometrica. Nel caso in cui $n = 1$ e $m = 1$ l'equazione 1.1 diventa:

$$\hat{y} = \mathbf{w} * \mathbf{x} \tag{1.3}$$

La formula 1.3 è uguale alla formula della retta. Questo perchè quello che l'algoritmo di linear regression fa, nel caso più semplice, è approssimare

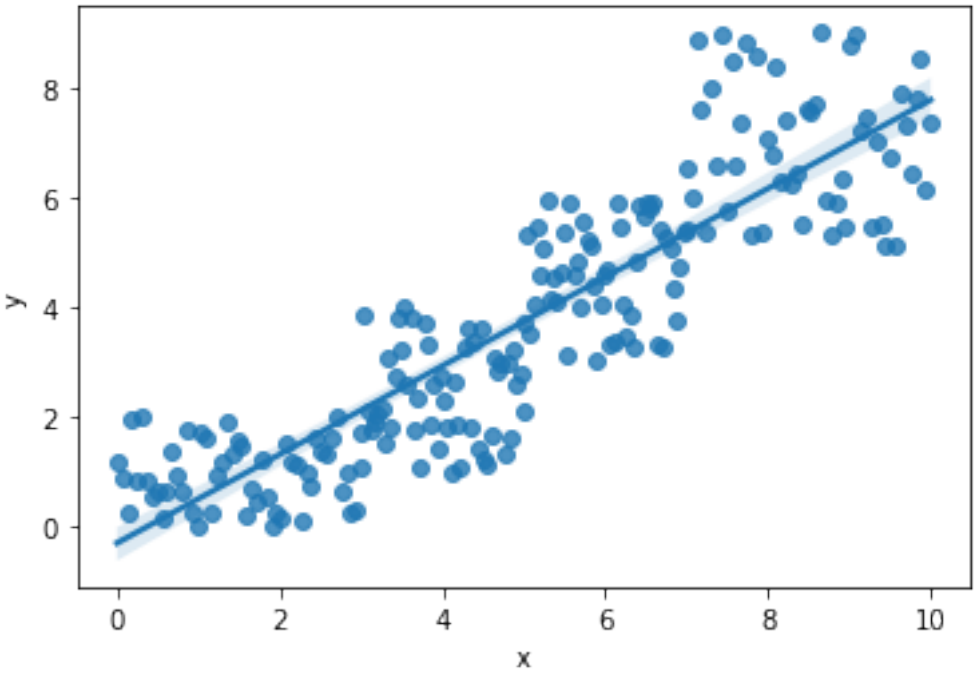


Figura 1.1: Esempio di linear regression su un dataset con una feature

i nostri dati in input **X** con una retta passante per il centro, come si nota in figura 1.1

I parametri w_i determinano l'inclinazione della retta e sono calcolati in automatico dall'algoritmo per individuare la migliore approssimazione possibile.

Come fa l'algoritmo a scegliere i pesi in modo da avere la migliore approssimazione dei dati possibile? Serve una misura che ci permetta di paragonare le predizioni una con l'altra.

1.2.1 MSE

Supponiamo di avere una matrice di n possibili input, un vettore target y contenente le predizioni esatte per tutti gli input e di utilizzare entrambi per la valutazione delle performance del modello. Ci riferiremo alla matrice

come $\mathbf{X}^{(test)}$, al vettore dei valori esatti come $\mathbf{y}^{(test)}$ e al vettore dei valori predetti dal modello come $\hat{\mathbf{y}}^{(test)}$.

Un modo per valutare le performance del modello è quello di calcolare l'errore quadratico medio (in inglese mean squared error, MSE):

$$MSE_{test} = \frac{1}{n}\sum_{i}(\hat{\mathbf{y}}_i^{(test)} - \mathbf{y}_i^{(test)})^2 \tag{1.4}$$

L'errore quadratico medio può anche essere visto come la distanza euclidea tra la predizione e il target:

$$MSE_{test} = \frac{1}{n} \parallel (\hat{\mathbf{y}}^{(test)} - \mathbf{y}^{(test)}) \parallel_2^2 \tag{1.5}$$

Più $\hat{\mathbf{y}}^{(test)}$ sarà simile a $\mathbf{y}^{(test)}$ e più il valore del mean squared error sarà basso. Quindi, più una predizione è accurata e più il MSE sarà prossimo allo zero; MSE sarà minimo (quindi zero) quando $\hat{\mathbf{y}}^{(test)} = \mathbf{y}^{(test)}$. [5]

1.2.2 Equazioni Normali

Quando e come vengono scelti i pesi da utilizzare durante la predizione? Il vettore dei parametri w verrà calcolato durante la fase di allenamento del modello. In questa fase all'algoritmo vengono passati in input una matrice $\mathbf{X} \in \mathbb{R}^{m \times n}$ e un vettore target y contenente le predizioni esatte per tutti gli n campioni. Per avere le migliori predizioni possibili, l'algoritmo dovrà scegliere i pesi $\mathbf{w}_i$ in modo da minimizzare il MSE. Per minimizzarlo, è sufficiente porre il suo gradiente a zero:

$$\nabla_w MSE_{train} = 0 \tag{1.6}$$

Sostituendo a MSE_{train} la definizione data in 1.5 si trova:

$$\nabla_w \frac{1}{n} \parallel (\hat{\mathbf{y}}^{(train)} - \mathbf{y}^{(train)}) \parallel_2^2 = 0 \tag{1.7}$$

Sostituendo a $\hat{y}^{(train)}$ la definizione data in 1.1 si ottiene:

$$\frac{1}{n}\nabla_w \parallel \mathbf{X}^{(train)} w - \mathbf{y}^{(train)}) \parallel_2^2 = 0 \tag{1.8}$$

Risolvendo la norma euclidea, si consegue:

$$\nabla_w(\mathbf{X}^{(train)}w - \mathbf{y}^{(train)})^T(\mathbf{X}^{(train)}w - \mathbf{y}^{(train)}) = 0 \tag{1.9}$$

Distribuendo la trasposizione e applicando il prodotto scalare si ottiene:

$$\nabla_w(w^T\mathbf{X}^{(train)T}\mathbf{X}^{(train)}w - 2w^T\mathbf{X}^{(train)T}\mathbf{y}^{(train)} + \mathbf{y}^{(train)T}\mathbf{y}^{(train)} = 0) \tag{1.10}$$

Risolvendo il gradiente si trova:

$$2\mathbf{X}^{(train)T}\mathbf{X}^{(train)}w - 2\mathbf{X}^{(train)T}\mathbf{y}^{(train)} = 0 \tag{1.11}$$

Isolando poi w si ottiene:

$$w = (\mathbf{X}^{(train)T}\mathbf{X}^{(train)})^{-1}\mathbf{X}^{(train)T}\mathbf{y}^{(train)} \tag{1.12}$$

L'equazione in 1.12 è la soluzione di un sistema di equazioni chiamato "equazioni normali" e permette di trovare il vettore dei pesi w in modo da minimizzare l'errore quadratico medio. [5]

1.2.3 Implementazione

L'algoritmo di linear regression può essere utilizzato semplicemente tramite la sua implementazione all'interno della libreria python: scikit-learn. Tutti gli algoritmi di supervised learning sono offerti dalla libreria come oggetti e sono implementati con una determinata interfaccia:

1. Un metodo chiamato "fit" che riceve come input dei dati e li utilizza per l'allenamento del modello. [6]

2. Un metodo chiamato "predict" che prende come input dei dati e produce per essi delle predizioni, basate sui parametri scelti in fase di allenamento. [6]

Il metodo fit deve essere chiamato prima del metodo predict, sennò verrà segnalato errore.

Allo stesso modo funziona quindi l'implementazione dell'algoritmo di linear regression. Quest'ultima ha un metodo "fit", con input $\mathbf{X} \in \mathbb{R}^{n \times m}$ e $\mathbf{y} \in \mathbb{R}^n$, dove $\mathbf{X}$ sono i dati per l'allenamento e $\mathbf{y}$ i dati target. Questi due input vengono utilizzati insieme all'equazione 1.12 per calcolare il migliore set di pesi w. Il metodo restituisce in output l'oggetto stesso, che avrà salvato il vettore dei parametri calcolati.
L'oggetto ha un metodo "predict" con input $\mathbf{X} \in \mathbb{R}^{n \times m}$ che rappresenta i dati su cui effettuare previsioni; inoltre restituisce in output un vettore $\hat{y} \in \mathbb{R}^n$, che è il vettore dei valori predetti per i dati in input.

1.2.4 Bias

Spesso quando ci si riferisce all'algoritmo di linear regression si intende un modello leggeremente più complesso dove è presente anche un termine b chiamato bias o di intercettazione. L'equazione 1.1 diventa:

$$\hat{y} = \mathbf{w}^T\mathbf{X} + \mathbf{b} \tag{1.13}$$

Dove $\mathbf{b} \in \mathbb{R}^n$ dove $\forall i, \mathbf{b}_i = b$

Si può anche decidere di vedere il vettore del bias b come un parametro aggiuntivo e aggiungerlo semplicemente modificando le dimensioni di $\mathbf{X}$ e $\mathbf{w}$:

- La matrice in input diventa $\mathbf{X} \in \mathbb{R}^{(m+1)*n}$; Si aggiunge quindi una riga, che deve essere composta da soli 1.

- Il vettore dei pesi diventa $\mathbf{w} \in \mathbb{R}^{m+1}$ dove l'elemento $w_0 = b$.

L'aggiunta del termine di intercettazione cambia quella che era l'interpretazione geometrica del nostro algoritmo. Come si nota in figura 1.2, con questa modifica i dati verranno sempre approssimati con una retta, ma quest'ultima non sarà più costretta a passare per il centro [5].

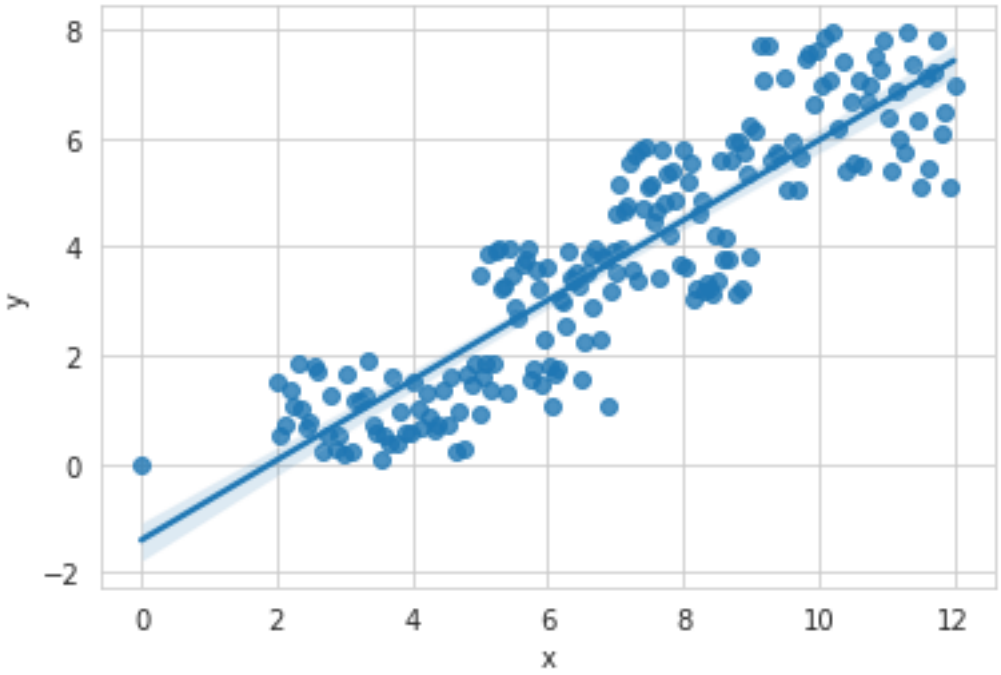


Figura 1.2: Esempio di linear regression con bias, su un dataset con una feature

1.3 Logistic Regression

La logistic regression è un algoritmo di supervised learning che serve a risolvere un problema di classificazione binaria (dove il vettore target y può assumere solo valore 0 o 1). Il programma :

- Prende in input una matrice $\mathbf{X} \in \mathbb{R}^{m+1 \times n}$,dove n rappresenta il numero di esempi da classificare, mentre m rappresenta l'insieme delle feature che verranno utilizzate per compiere il task.

- Restituisce in output un vettore $\hat{y} \in \mathbb{R}^n | \ \forall i, \hat{y}_i \in [0,1]$ calcolato nel seguente modo:

$$\hat{y} = \frac{1}{1 + e^{-\theta^T \mathbf{x}}} \tag{1.14}$$

  Dove $\theta \in \mathbb{R}^{m+1}$ è un vettore di parametri.

L'output della classificazione viene calcolato attraverso una funzione logistica. La particolarità di questo tipo di funzioni, è che il valore da loro calcolato risulta sempre compreso tra 0 e 1 come si nota nella figura 1.3

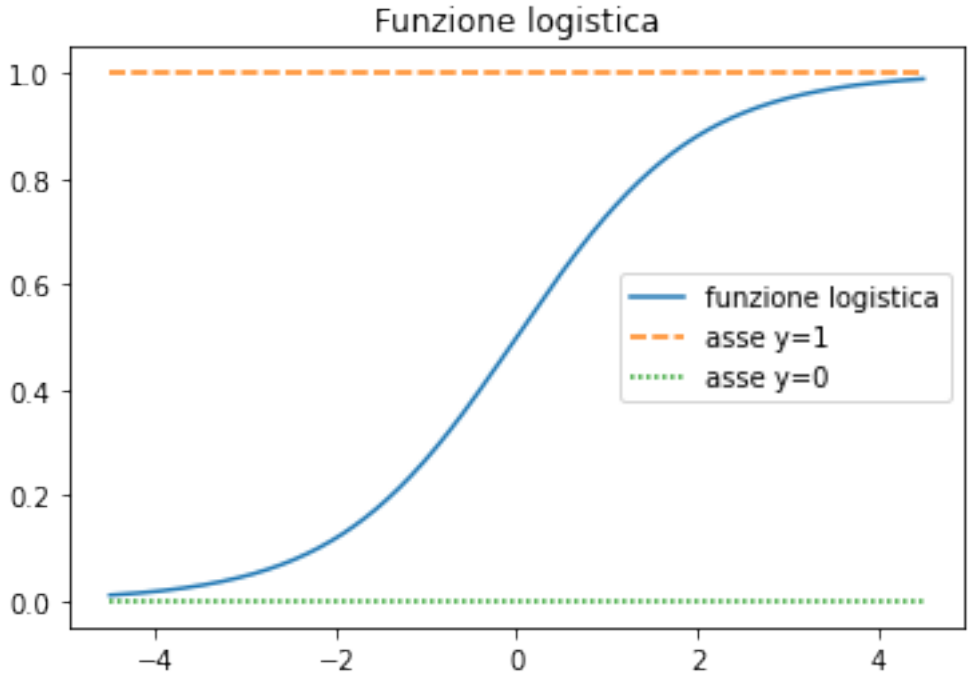


Figura 1.3: Grafico della funzione logistica: $g(z) = \frac{1}{1+e^{-z}}$

L'elemento i-esimo $\hat{y}_i$ che viene restituito dall'algoritmo di logistic regression può essere interpretato come la probabilità che l'elemento i-esimo nel vettore target y sia uguale a 1, quindi:

$$\hat{y}_i = P(y_i = 1) \tag{1.15}$$

Vale anche la seguente proprietà:

$$P(y_i = 1) + P(y_i = 0) = 1 \tag{1.16}$$

Tutte le probabilità maggiori o uguale a 0.5 vengono classificate nella classe 1 (perchè c'è una probabilità sufficientemente alta che gli elementi

corrispondenti nel vettore target siano 1), mentre tutti quelli minori di 0.5 vengono classificati nella classe 0.

Dire che $\hat{y}_i$ è maggiore o uguale a 0.5 significa dire:

$$\frac{1}{1+e^{-\theta^T \mathbf{X}_i}} \geq 0.5 \tag{1.17}$$

Facendo riferimento alla figura 1.3, si evince come la funzione logistica sia ≥ 0.5 quando l'input z è ≥ 0. Da cui:

$$\hat{y}_i \geq 0.5 \Leftrightarrow \frac{1}{1+e^{-\theta^T \mathbf{X}_i}} \geq 0.5 \Leftrightarrow \theta^T \mathbf{X}_i \geq 0 \tag{1.18}$$

Quindi, in fase di allenamento del modello, il vettore dei parametri θ deve essere scelto in modo che quando un elemento i-esimo del vettore target y è 1, la i-esima colonna della matrice X moltiplicata scalarmente con θ^T dia un risultato ≥ 0.

Diamo anche un'interpretazione geometrica alla ricerca dei parametri ottimali.

Sia $\mathbf{X}$ una matrice di campioni(con la prima riga tutta di 1) con solo due features ($m = 2$), sia $\mathbf{y}$ il vettore target contente la classe esatta per ogni campione; sia poi la figura 1.4 il grafico della matrice $\mathbf{X}$ con classi $\mathbf{y}$.

Supponiamo infine di avere un modello già allenato per la logistic regression. Abbiamo quindi già calcolato il vettore dei parametri $\theta = \begin{bmatrix} -15 \\ 1 \\ 1 \end{bmatrix}$. Definiamo anche un generico vettore $x_j = \begin{bmatrix} 1 \\ x1 \\ x2 \end{bmatrix}$. Basandoci sull'equazione 1.18 :

$$\theta^T x_j \geq 0 \Rightarrow -15 + x1 + x2 \geq 0 \tag{1.19}$$

L'equazione appena trovata individua la regione di spazio in cui risiede la classe 1 e che è "sopra" la retta rossa nella figura 1.5. La retta rossa è invece individuata dall'equazione $-15 + x1 + x2 = 0$; la stessa è anche chiamata decision boundary ed è il luogo dove $\hat{y}_j$ ha valore esattamente 0.5.

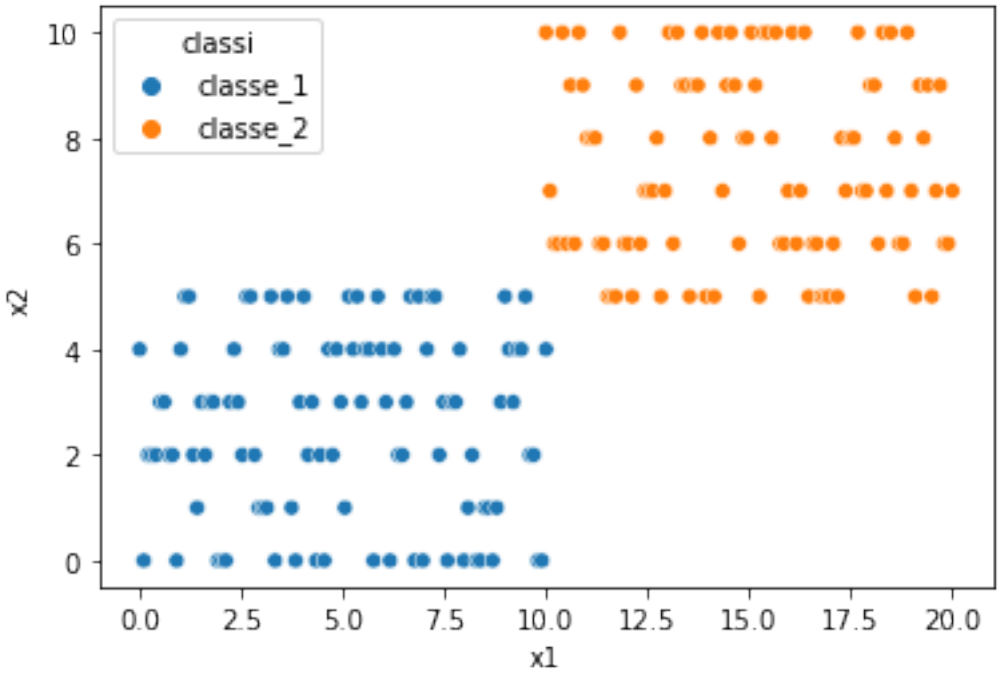


Figura 1.4: Grafico rappresentante un dataset generato casualmente, con due feature x1,x2. Due sono i possibili valori target: "classe_1" e "classe_2"

Quindi, ricercare i parametri ottimali significa ricercare il decision boundary che meglio separa le nostre classi.

Come fa l'algoritmo a scegliere i parametri in modo da avere un decision boundary ottimale? Anche in questo caso ci serve una misura che ci permetta di paragonare una classificazione con l'altra.

1.3.1 Cross-entropy

Supponiamo di avere una matrice di n possibili input, un vettore target y contenente le classi di appartenenza per ognuno degli n input e di utilizzare entrambi per la valutazione delle performance del modello. Ci riferiremo alla matrice come $\mathbf{X}^{(test)}$, al vettore dei valori esatti come $\mathbf{y}^{(test)}$ e al vettore output del modello come $\hat{\mathbf{y}}^{(test)}$.

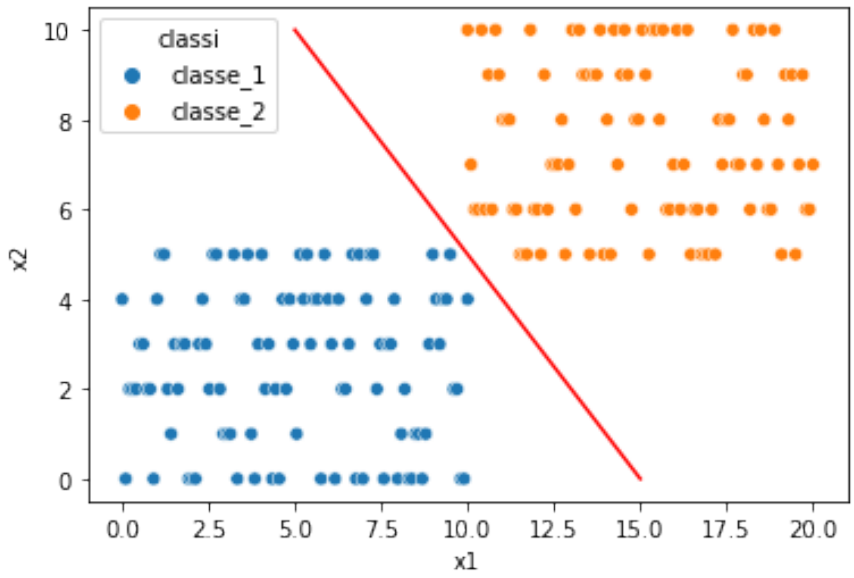


Figura 1.5: Grafico della 1.4 con aggiunta del decision boundary

Un modo per valutare le performance del modello è quello di calcolare la perdita logistica (anche chiamata cross-entropy):

$$CE = -y^{(test)} * ln(\hat{y}^{(test)}) - (1 - y^{(test)}) * ln(1 - \hat{y}^{(test)}) \tag{1.20}$$

Questa formula viene ricavata applicando il logaritmo naturale con cambio segno all'equazione della likelihood. [7]

Quando $y_i^{(test)} = 1$ l'equazione 1.20 diventa:

$$CE = -ln(\hat{y}_i^{(test)}) \tag{1.21}$$

Mentre quando $y_i = 0$ l'equazione 1.20 diventa:

$$CE = -ln(1 - \hat{y}_i^{(test)}) \tag{1.22}$$

Sia nel caso di $y_i^{(test)} = 1$ che di $y_i^{(test)} = 0$, la cross-entropy avrà sempre un valore non negativo. Questo perchè abbiamo il logaritmo naturale di un valore sempre compreso tra 0 e 1, che sarà sempre un numero non positivo moltiplicato per -1.

Anche in questo caso, come per il MSE, la cross-entropy avrà valore minimo(zero) quando $\hat{y}^{(test)} = y^{(test)}$.

1.3.2 Gradient Descent

Quando e come vengono scelti i parametri da utilizzare durante la classificazione? Anche in questo caso il vettore dei parametri θ verrà calcolato durante la fase di allenamento del modello. In questa fase, all'algoritmo vengono passati in input una matrice $\mathbf{X} \in \mathbb{R}^{(m+1)\times n}$ e un vettore target y contenente la classe per ognuno degli n campioni. Per avere la miglior classificazione possibile, l'algoritmo dovrà scegliere i parametri θ_i in modo da minimizzare la somma di tutti i valori calcolati dalla cross-entropy, quindi:

$$SumCE = \sum_i (CE(\hat{y}_i^{(train)}, y_i^{(train)}) \tag{1.23}$$

Per minimizzare la somma dei valori calcolati dalla cross-entropy, si utilizzano metodi basati su gradient descent. Il gradient descent è un algoritmo che serve per trovare un punto di minimo locale di una funzione differenziabile e funziona in questo modo:

1. Supponiamo di avere una funzione $f(x)$ differenziabile da minimizzare.
2. Vengono scelti dei valori iniziali per x casuale e calcoliamo la $f(x)$.
3. Si calcola il gradiente della funzione, sostituendo ad x i suoi valori e trovando la direzione verso un punto di massimo.
4. Vengono aggiornati i valori di x, muovendoci in direzione contraria a quella ottenuta dal gradiente (quindi verso un punto di minimo), di un passo di lunghezza arbitraria.
5. Si valuta nuovamente la funzione $f(x)$ sui nuovi valori di x.
6. Tornare al punto 3 fino a chè non ci sarà più un decremento del valore di $f(x)$. Questo significherà che x corrisponderà al valore di minimo locale della funzione.

Alcuni dei metodi utilizzati per minimizzare la formula 1.23 si basano su una versione ottimizzata dell'algoritmo di gradient descent, chiamata stochastic gradient descent (SGD). La differenza principale tra i due algoritmi è che nello SGD, la funzione SumCE non viene valutata usando tutto l'insieme del training set , bensì usandone solo una parte di esso. Nei casi di training set molto grandi, l'algoritmo di stochastic gradient descent porta ad un costo computazionale inferiore rispetto all'algoritmo di gradient descent.[8]

1.3.3 Implementazione

L'algoritmo di logistic regression può essere utilizzato semplicemente tramite la sua implementazione all'interno della libreria python: scikit-learn. Come già discusso nel paragrafo 1.2.3, anche la logistic regression viene implementata come oggetto con la medesima interfaccia di quella vista nella linear regression.

L'implementazione ha quindi un metodo "fit", con input $\mathbf{X} \in \mathbb{R}^{n \times m+1}$ e $\mathbf{y} \in \mathbb{R}^n$, dove $\mathbf{X}$ sono i dati per l'allenamento e $\mathbf{y}$ le classi target. Questi due input vengono utilizzati insieme a un solver per calcolare il migliore set di parametri θ. Il metodo restituisce in output l'oggetto stesso, che avrà salvato il vettore dei parametri calcolati.

Che cos'è un solver? Un solver è un metodo che viene utilizzato per minimizzare un'equazione, in questo caso la 1.23. Il solver viene scelto quando l'oggetto viene istanziato per la prima volta. Scikit-learn permette di scegliere tra cinque solver differenti:

1. **SAG**: stochastic average gradient, funziona come SGD, solo che ad ogni iterazione si ricorda il risultato del valore di gradiente precedente e questo gli permette di convergere al minimo più velocemente. Una delle migliori scelte per dataset molto grandi.[9]

2. **SAGA**: é un miglioramento di SAG. Converge al minimo più velocemente e supporta anche regularizzazione di tipo "l1" (di cui parleremo più avanti).[10]

3. **L-BFGS**: é un metodo quasi-Netwon che ottimizza l'algoritmo BFGS, essendo più veloce e consumando meno risorse ad ogni iterazione[11]; é il solver selezionato di default da scikit-learn.

4. **Liblinear**: Si basa su un algoritmo di coordinate descent, che funziona in modo analogo al gradient descent, solo che calcola la direzione in cui eseguire i "passi" non con il gradiente della funzione, ma in un modo diverso. [12]

5. **Newton-cg**:é un algoritmo che utilizza le informazioni date dal gradiente e dall'hessiana di una funzione, per convergere più velocemente. [13]

L'oggetto ha un metodo "predict" con input $\mathbf{X} \in \mathbb{R}^{n \times m+1}$, che rappresenta i dati su cui effettuare classificazioni. Restituisce in output un vettore $\hat{y} \in \mathbb{R}^n$, che è il vettore dei valori classificati per i dati in input.

1.3.4 Multiclass Logistic regression

Fino ad ora è stato illustrato l'algoritmo della logistic regression binaria, che però può essere generalizzato per risolvere un problema di classificazione a K classi.

Anche in questo caso vale che la somma delle probabilità che l'i-esimo input appartenga a una delle K classi è pari a 1 (generalizzazione della formula 1.16).

Nella multiclass logistic regression a K classi, una di esse viene scelta come "pivot" . Vengono poi creati K-1 modelli distinti di logistic regression binaria, dove ognuno di essi viene allenato per riconoscere una delle K-1 classi rimanenti, mentre tutte le altre sono trattate come un'unica classe sconosciuta. La combinazione di questi modelli, ci permette di scoprire la probabilità assegnata ad ognuna delle K-1 classi. Utilizzando la generalizzazione dell'equazione 1.16, possiamo poi scoprire anche la probabilità della classe pivot. Conoscendo ora le probabilità associate ad ogni classe, è possibile effettuare la classificazione. [14]

1.4 Underfitting e Overfitting

Linear regression e logistic regression sono algoritmi allenati per minimizzare l'errore sul training set (vedi formule 1.12, 1.23). Quello che effettivamente interessa quando si utilizza uno dei due algoritmi, sono le performance su dati nuovi e mai visti. Performare bene su dati nuovi si chiama generalizzazione. [5]

L'errore di generalizzazione, anche detto errore di test, è l'errore che il modello compie su dati che non ha mai visto. Anche l'errore di test deve essere il più piccolo possibile.

Quando possiamo affermare che un modello di machine learning performa bene?

- Quando l'errore di allenamento è basso.
- Quando l'errore di test e quello di allenamento sono paragonabili.

Definiamo che un modello soffre di underfitting se ottiene un errore troppo grande in fase di allenamento. Definiamo che un modello soffre di overfitting se ha un errore di test molto superiore rispetto a quello di allenamento. [5]

Per risolvere questo tipo di problema, si può scegliere di cambiare il numero di feature da utilizzare. Nel caso di underfitting bisogna scegliere un numero maggiore di feature per rendere il modello più preciso, mentre nel caso di overfitting bisogna diminuire il numero di feature da utilizzare.

Non sempre però vogliamo decidere di togliere delle feature dal nostro modello. Come si agisce in quei casi?

1.4.1 Regolarizzazione

La regolarizzazione è una tecnica per ridurre l'overfitting. Nel caso di linear e logistic regression, la regolarizzazione consiste nell'aggiungere un termine chiamato termine di regolarizzazione(RT)alle funzioni da minimizzare (rispettivamente 1.4, 1.23). Il termine da aggiungere dipende dal tipo di regolarizzazione che si sceglie di utilizzare:

1. Regolarizzazione *l1*: $RT = \alpha \parallel w \parallel_1$

2. Regolarizzazione *l2*: $RT = \alpha \parallel w \parallel_2$

Prendiamo ad esempio la formula 1.4 e aggiungiamo il termine di regolarizzazione:

$$\begin{aligned} MSE_{train} &= \frac{1}{n}\sum_i (\hat{\mathbf{y}}_i^{(train)} - \mathbf{y}_i^{(train)})^2 + \alpha \parallel w \parallel_1 \\ &= \frac{1}{n}\sum_i (\hat{\mathbf{y}}_i^{(train)} - \mathbf{y}_i^{(train)})^2 + \alpha \sum_j |w_j| \end{aligned} \tag{1.24}$$

$$\begin{aligned} MSE_{train} &= \frac{1}{n}\sum_i (\hat{\mathbf{y}}_i^{(train)} - \mathbf{y}_i^{(train)})^2 + \alpha \parallel w \parallel_2 \\ &= \frac{1}{n}\sum_i (\hat{\mathbf{y}}_i^{(train)} - \mathbf{y}_i^{(train)})^2 + \alpha \sqrt{w_1^2 + ... + w_m^2} \end{aligned} \tag{1.25}$$

La minimizzazione della formula 1.24, essendo una somma, comporta la minimizzazione di entrambi i termini. La minimizzazione del primo termine comporta la creazione della formula 1.12, mentre la minimizzazione del secondo termine comporta (per quanto possibile) la minimizzazione ogni singolo parametro. Un discorso analogo può essere fatto per la formula 1.25.

Aggiungere il termine di regolarizzazione e minimizzarlo consiste quindi nel minimizzare i valori del vettore dei parametri. Perchè questo dovrebbe ridurre l'overfitting? Precedentemente è stato riportato che per combattere l'overfitting, una strategia è quella di ridurre il numero di features. Se un valore del vettore dei parametri diventa zero, la corrispondente feature sarà ininfluente nel calcolo dell'MSE (nel caso di linear regression), quindi minimizzare il vettore dei parametri significa effettuare una feature selection, fatta dall'algoritmo stesso, per massimizzare le performance dello stesso.

La principale differenza tra le due regolarizzazioni è che la regolarizzazione di tipo *l1* tende a creare vettori dei parametri sparsi. [15]

1.5 Hyperparameter tuning

La maggior parte degli algoritmi di machine learning prende come input degli hyperparametri, che sono dei parametri usati per controllare il comportamento dell'algoritmo. Questi parametri non vengono mai adattati dall'algoritmo stesso ed ogni algoritmo ha i suoi.

Noi discuteremo quelli di linear regression e di logistic regression, secondo l'implementazione che si trova nella libreria di python scikit-learn.

1.5.1 Linear Regression

L'algoritmo di linear regression base, discusso in 1.2.3 non presenta hyperparametri. La sua versione con regolarizzazione, invece, ha due hyperparametri principali:

1. **alpha** che è la costante per cui viene moltiplicato il termine di regolarizzazione nella funzione da minimizzare.

2. **solver** è il metodo che l'algoritmo utilizza per minimizzare la funzione 1.25.Funziona solo con regolarizzazione *l2* perchè non tutti i solver supportano la regolarizzazione di tipo *l1*.

1.5.2 Logistic Regression

L'algoritmo di logistic regression presenta invece tre hyperparametri principali:

- **penalty** che è un parametro che ci permette di scegliere quale tipo di regolarizzazione utilizzare. Anche None è un valore valido, e significa che non viene aggiunto alcun termine di regolarizzazione alla funzione da minimizzare.

- **C** è un numero reale che rappresenta l'inverso della forza di regolarizzazione. Più è piccolo e più è forte la regolarizzazione.

- **solver** è il metodo con cui decidiamo di minimizzare la funzione. Non tutti i solver sono compatibili con regolarizzazione di tipo *l1*.

1.5.3 Grid search

Il metodo utilizzato all'interno di questa tesi per effettuare hyperparameter tuning si chiama Grid search e viene implementato dalla libreria scikit-learn come un oggetto con la medesima interfaccia descritta nel paragrafo 1.2.3 [6].

L'oggetto grid search alla sua creazione riceve come input un modello, un dizionario che ha per chiavi gli hyperparametri e per valori uno spazio di possibili valori per ogni chiave e un valutatore di performance coerente con la task che imparerà il modello. L'implementazione del grid search ha anche un metodo "fit", con input $\mathbf{X} \in \mathbb{R}^{n \times m+1}$ e $\mathbf{y} \in \mathbb{R}^n$, dove $\mathbf{X}$ sono i dati per l'allenamento e $\mathbf{y}$ il vettore target. Quando il metodo fit viene chiamato, il modello passato precedentemente in input, viene allenato sui dati passati in input a fit con tutte le possibili combinazioni di hyperparametri definite nel dizionario e salva la combinazione di hyperparametri che genera la miglior performance.

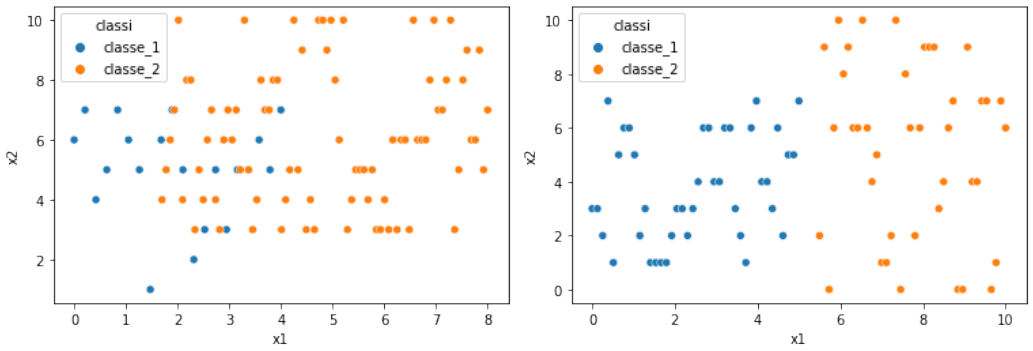
1.6 Bilanciamento del dataset

Uno dei principali problemi che può ridurre di molto la capacità di classificazione di un modello è l'allenamento su un dataset sbilanciato, specialmente se c'è anche sovrapposizione tra le classi.

Un dataset sbilanciato è un dataset contenente tanti esempi per una determinata classe e pochi per le altre classi rimanenti.

1.6.1 Perchè imparare da un dataset bilanciato è meglio

Imparare da un dataset sbilanciato è un task molto complesso. Questo perchè una situazione come quella in figura 1.6a, con la classe 2 con molti



(a) (b)

Figura 1.6: Classe 2 predomina rispetto alla classe 1 (a) dataset bilanciato rispetto alle due classi (b) [16]

più dati rispetto alla classe 1 e con presenza di sovrapposizione tra esse, può "confondere" un classificatore come la logistic regression. Per esempio, ci si rende conto che il decision boundary non riuscirà mai a separare perfettamente le due class; questo risulterà in un errore di classificazione per la classe minoritaria molto alto e non è accettabile. [16]

Di seguito discuteremo due delle tecniche più comuni per bilanciare un dataset.

1.6.2 Random under-sampling

Il random under-sampling è un metodo non-euristico che mira al bilanciamento delle classi eliminando randomicamente elementi dalla classe maggioritaria. Il principale difetto di questo algoritmo, è che rischia di eliminare dei dati potenzialmente fondamentali per l'algoritmo di machine learning. [17]

1.6.3 Random over-sampling

Il random over-sampling è un metodo non-euristico che mira al bilanciamento delle classi replicando randomicamente elementi dalla classe minoritaria. Si pensa che l'utilizzo di questo algoritmo possa aumentare l'overfitting.

Inoltre, se il dataset sbilanciato è già abbastanza grande, richiede un costo computazionale elevato. [17]

Capitolo 2

Modelli predittivi di low achievement con il dataset INVALSI

Come già menzionato nell'introduzione, questa tesi espande il lavoro fatto da A. Zanellati et al.[19] dove vengono allenati tre modelli di machine learning sul dataset INVALSI delle prove di matematica per predire il rischio di non raggiungimento di obiettivi minimi.

2.1 INVALSI

L'INVALSI è l'Istituto nazionale per la valutazione del sistema educativo di istruzione e di formazione che somministra agli studenti in determinati anni del percorso scolastico delle prove scritte per la valutazione del livello di apprendimento di alcune competenze fondamentali di Italiano, Inglese e matematica. [18]

Le prove INVALSI vengono proposte in modo censuario:

1. In seconda elementare (livello K-2).

2. In quinta elementare (livello K-5).

3. In terza media (livello K-8).

4. In seconda superiore (livello K-10).

5. In quinta superiore (livello K-13).

Le prove invalsi di matematica prevedono una votazione finale da 1 (minimo) a 5 (massimo) e le domande sono strutturate in modo da valutare diversi aspetti e competenze, in base al livello (ad esempio livello K-5, K-10) al quale vengono proposti. Per due prove dello stesso livello, ma di annate differenti, le competenze valutate sono sempre le stesse, ma la percentuale di domande a loro associate può avere delle piccole variazioni.

2.2 Metodologie

Nello short paper di A.Zanellati et al.[19] sono state selezionate due coorti di studenti al livello K-5: una per l'anno scolastico 2012/2013 e una per l'anno scolastico 2013/2014. Per gli stessi studenti sono stati presi anche i dati riguardanti le invalsi al livello K-10 per la definizione del target. In particolare il target avrà valore 1 (presenza di low achievement) se il voto della prova, conseguita in seconda superiore, è ≤ 2, viceversa avrà valore 0 (assenza di low achievement) nel caso in cui il voto sia ≥ 3. Le coorti al livello K-5 sono state unite con il target creato a a partire dai risultati delle coorti al K-10.

2.2.1 Dataset

Il dataset INVALSI per le prove di matematica, non è solamente composto dal voto finale ottenuto nella specifica prova da ciascun alunno. Presenta un valore booleano per ogni domanda del test, ma anche tantissime altre feature, non solo relative al contesto scolastico, ma anche socio-economico e demografico; ad esempio, la provenienza geografica dello studente, il livello di istruzione di entrambi i genitori, se il ragazzo abbia frequentato la materna, etc.

2.2.2 Encoding

Per rendere il modello trasferibile tra coorti dello stesso livello ma di annate diverse, è necessario raggruppare le domande in base alle competenze associate, poi creare una variabile per ognuno dei gruppi e calcolare il numero di risposte giuste sul numero totale (che può cambiare di anno in anno). Basandosi sul framework INVALSI per la costruzione dei test, A.Zanellati et al.[19] individuano aree, processi e macro-processi usati per l'encoding delle domande che troviamo in tabella 2.1.

Nella tabella 2.2 si può vedere il risultato dell'encoding legato alle feature strutturalmente legate agli aspetti didattici dell'apprendimento, fatto su due studenti.

2.2.3 Preprocessing

Prima di allenare i modelli di machine learning, vengono ultimati alcuni step di preprocessing sul database:

1. Eliminazione delle feature con molti valori mancanti.
2. "One-hot" encoding delle categorical features per renderle dei valori numerici.
3. Feature selection.
4. Bilanciamento del dataset, visto che i dataset INVALSI tendono a presentare uno sbilanciamento tra chi non ha low achievement e chi non li ha.

2.2.4 Algoritmi

Il primo modello è un random forest (RF), molto utilizzato all'interno del campo dell'educational data mining; il modello è stato allenato con alcune tecniche specifiche per ridurre overfitting ed aumentare la precisione. Il secondo modello è un categorical embeddings (CE) basato su reti neurali

con diversi tipi di layer in base al tipo di input. Il terzo modello è un feature tokenizer transformer (FTT) basato su reti neurali con meccanismi di attenzione. [19]

2.3 Risultati

I tre modelli di machine learning sono stati allenati usando la piattaforma di sviluppo Google Colaboratory Notebook, Python come linguaggio di programmazione e alcune importanti librerie come scikit-learn e pandas. Come si nota dai risultati riportati in tabella 2.3, il modello con le migliori performance è il feature tokenizer transformer che ottiene valori superiori in tutte e tre le metriche; mentre quello con le peggiori performance è il random forest, con addirittura uno scarto di 0.15 punti nella precision rispetto al FTT. Il modello che fa uso di categorical embedding ottiene dei risultati simili, anche se sempre leggermente inferiori rispetto al FTT, ma di gran lunga superiori al modello basato su random forest, con uno scarto di 0.14 per la precision e di 0.9 per la recall.

Le migliori performance sono ottenute tramite l'utilizzo di reti neurali che hanno come unico difetto una interpretabilità inferiore rispetto al random forest.

Queste performance, soprattutto quelle riportate dal random forest, possono essere simili o peggiori a quelle che si potrebbero ottenere con l'utilizzo di modelli statistici di apprendimento più tradizionali, come linear o logistic regression? Questa è la domanda a cui proveremo a rispondere con questa tesi.

Aree
(NU) Numeri
(SP) Spazi e figure
(DF) Dati e previsioni
(RF) Relazioni e funzioni
Processi
(P1) Conoscere e padroneggiare su specifici contenuti della matematica
(P2) Conoscere e usare di algoritmi e procedure
(P3) Conoscere le differenti forme di rappresentazione e muoversi dall'una all'altra
(P4) Risolvere problemi usando strategie in diversi campi
(P5) Riconoscere la misurabilità degli oggetti e fenomeni in differenti contesti e misurarne le quantità.
(P6) Acquisire progressivamente le forme tipiche di pensiero matematico
(P7) Utilizzare tool, modelli e rappresentazioni nel trattamento quantitativo dell'informazione nel campo scientifico, tecnologico, economico e sociale
(P8) Riconoscere forme in spazio e utilizzarle per risolvere problemi
Macro-processi
(MP1) Formulazione
(MP2) Interpretazione
(MP3) Utilizzo

Tabella 2.1: Tabella dell'encoding delle domande [19]

Id	**NU**	**SP**	**DF**	**RF**	**P1**	**P2**	**P3**	**P4**	**P5**	**P6**	**P7**	**P8**	**MP1**	**MP2**	**MP3**
1	0.86	0.75	0.90	0.80	0.71	0.80	1.00	0.89	1.00	0.67	0.91	0.75	0.81	0.73	0.94
2	0.50	0.25	0.50	0.53	0.29	0.60	0.50	0.22	1.00	0.33	0.73	0.25	0.50	0.47	0.44

Tabella 2.2: Tabella con esempio di encoding delle feature legate agli aspetti didattici [19]

Modello	**Accuracy**	**Precision**	**Recall**
Random Forest	0.77	0.62	0.67
CE	0.76	0.76	0.76
FTT	0.78	0.77	0.78

Tabella 2.3: Tabella dei risultati ottenuti [19]

Capitolo 3

Implementazione e Risultati

Per questa tesi sono stati sviluppati dei modelli di machine learning basati su logistic regression e su linear regression, allenati sul dataset INVALSI per il riconoscimento di low achievement scolastico. L'obiettivo era il poter paragonare i modelli di machine learning più tradizionali con i modelli più complessi utilizzati da A.Zanellati et al. nel loro paper (tabella 2.3) per verificare se un incremento di complessità implica anche un aumento delle performance.

3.1 Ambiente di sviluppo

I modelli di machine learning creati per questa tesi sono stati sviluppati utilizzando:

- La piattaforma di sviluppo Google Colaboratory Notebook.
- Python in versione 3.7.14 come linguaggio di programmazione.
- La libreria di python Pandas in versione 1.3.5.
- La libreria di python Numpy in versione 1.21.6.
- La libreria di python scikit-learn in versione 1.0.2.

3.2 Tipi di dataset utilizzato

Il dataset utilizzato per lo sviluppo dei modelli è sempre quello INVALSI delle prove di matematica presentato nel paragrafo 2.2.1. Anche in questo caso sono state selezionate le medesime coorti al livello K-5: quella dell'anno scolastico 2012/2013 e quella dell'anno scolastico 2013/2014. Per gli stessi studenti, sono stati presi anche i dati riguardanti le invalsi al livello K-10 per la definizione di due diversi target:

1. **Target binario**: lo stesso definito nella sezione 2.2. Gli elementi del target avranno valore 1 (presenza di low chievement) se il voto della prova, conseguita in seconda superiore, è ≤ 2; viceversa avranno valore 0 (assenza di low achievement) nel caso in cui il voto sia ≥ 3.

2. **Target non binario**. Il target assumerà esattamente i valori dei voti presi dagli studenti nella prova di matematica a livello K-10. Quindi i valori saranno sempre compresi tra 1 e 5 e l'obiettivo sarà classificare correttamente lo studente nella classe di voto corretta.

Dall'unione delle coorti a livello K-5 con il target binario otterremo il dataset binario di allenamento (DSB_{train}, coorte 2012/2013) e il dataset binario di test (DSB_{test}, coorte 2013/2014). Dall'unione delle coorti a livello K-5 con il target non binario otterremo invece il dataset non binario di allenamento ($DSNB_{train}$, coorte 2012/2013) e il dataset non binario di test ($DSNB_{test}$, coorte 2013/2014).

3.3 Preprocessing

Prima della definizione dei modelli di machine learning sviluppati in questa tesi, vengono sempre effettuati degli step di preprocessing sul database.

Visto che sia il DSB_{train}, che il $DSNB_{train}$ presentano un forte sbilanciamento delle classi, il primo step del preprocessing prevede l'utilizzo della tecnica di random under-sampling discussa nel paragrafo 1.6.

Come discusso nel paragrafo 2.2.1, il dataset INVALSI è ricco di feature quindi per cercare di evitare overfitting, il secondo step di preprocessing prevede l'utilizzo di tecniche per la rimozione delle feature meno significative all'apprendimento:

1. **SelectFpr**: é una tecnica per selezionare feature basata su test statistici univariati, che quindi dipendono da un singolo attributo o caratteristica. In questo caso, le feature vengono selezionate in base al false positive rate.

2. **SelectFromModel**: è una tecnica per selezionare feature da affiancare ad un qualsiasi modello che associa un coefficiente di importanza ad ogni feature. Le feature a cui viene assegnato un coefficiente più basso di una determinata threshold, vengono direttamente eliminate.

3.4 Modelli basati su logistic regression

Il task che in questa tesi ci poniamo di compiere è, indifferentemente dal target binario o non binario, un task di classificazione. Quindi i primi modelli sono stati creati utilizzando:

- logistic regression binaria per i dataset binari.

- multiclass logistic regression per i dataset non binari.

3.4.1 Modelli su dataset binario

Nel caso di dataset binario, il nostro target potrà soltanto avere valori 1 (presenza di low achievement) o 0 (assenza di low achievement). Sul DSB_{train} sono stati allenati due modelli di logistic regression che differiscono tra di loro per l'utilizzo di due diverse tecniche di feature selection.

Hyperparameter tuning

Per definire il miglior modello possibile, come primo step sono stati tunati gli hyperparametri, con tecnica grid search, descritta nel paragrafo 1.5. Ricordiamo che gli hyperparametri presenti in un modello di logistic regression sono: **C** e **solver**. Come sono stati scelti i valori da mettere nel dizionario input della grid search?

Per l'hyperparametro solver sono stati scelti i cinque diversi valori che può assumere. Invece, essendo l'hyperparametro C un parametro a valori reali, la scelta è descritta dal seguente algoritmo:

1. Inizialmente sono stati scelti una serie di valori equispaziati compresi tra 1/10 e 1*10, dove 1 è il valore di default assegnato da scikit-learn a C.

2. Si fa partire la grid search.

3. Si prende il valore ottenuto per C, e si selezionano dei valori a lui vicini.

4. Finchè si hanno dei miglioramenti sostanziali nello scoring, tornare al punto 2. Quando non si avrà più un miglioramento sostanziale, allora si potrà fermare la grid search.

Come metrica di scoring è stata selezionata la f1.

Sono riportati in tabella 3.1 i valori trovati in fase di hyperparameter tuning e utilizzati per la definizione dei modelli, che ora sono pronti per essere allenati.

Modelli	**solver**	**C**
Logistic regression con FPR	newton-cg	0.0022
Logistic regression con SelectFromModel	lbfgs	0.002

Tabella 3.1: Tabella degli hyperparametri per la logistic regression binaria

Risultati ottenuti

Entrambi i modelli così ottenuti vengono allenati sul DSB_{train} e testati sul DSB_{test}.

Come si evince dalla tabella 3.2, i valori per tutte e quattro le metriche sono identici. Non si ha quindi alcun guadagno in performance nell' utilizzare una tecnica di logistic regression rispetto all'altra; poichè la maggior parte delle feature selezionate sono identiche. Per questo motivo, da ora in avanti, si utilizzerà solo la selectFromModel come unica tecnica di feature selection.

Modelli	**Accuracy**	**Precision**	**Recall**	**F1**
Logistic regression con FPR	0.77	0.65	0.61	0.63
Logistic regression con SelectFromModel (LRB)	0.77	0.65	0.61	0.63

Tabella 3.2: Tabella dei risultati per la logistic regression binaria

3.4.2 Modello su dataset non binario

Nel caso di dataset non binario, il nostro target potrà avere valori compresi tra 1 e 5. Sul $DSNB_{train}$ è stato allenato un solo modello di multiclass logistic regression.

Hyperparameter tuning

Anche in questo caso, per definire il miglior modello possibile come primo step sono stati tunati gli hyperparametri, con tecnica grid search, descritta nel paragrafo 1.5. Gli hyperparametri presenti in un modello di multiclass logistic regression non sono solo **C** e **solver**, ma anche **multi_class**. Multi_class è un hyperparametro che permette di scegliere che algoritmo usare per la risoluzione di un problema a K classi.

Per l'hyperparametro solver sono stati scelti quattro dei cinque possibili valori che può assumere, perchè il solver liblinear non supporta tutti i valori dell'hyperparametro multi_class. Per l'hyperparametro multi_class

sono stati scelti i due possibili valori che può assumere. Il tuning dell'hyperparametro C è avvenuto con lo stesso algoritmo usato nella sezione 3.4.1.

In questo caso, la metrica di scoring utilizzata è la accuracy, perchè f1, essendo una metrica binaria, ha bisogno di specifici parametri per essere usata in una classificazione a K classi e il grid search non permette l'inserimento di tali parametri.

In tabella 3.3 sono riportati i valori trovati in fase di hyperparameter tuning che sono stati utilizzati per la definizione dei modelli e che ora sono pronti per essere allenati.

Modello	**solver**	**C**	**multi_class**
Multiclass logistic regression	newton-cg	13.5	multinomial

Tabella 3.3: Tabella degli hyperparametri per la multiclass logistic regression

Risultati ottenuti

Il modello viene poi allenato su $DSNB_{train}$ e testato su $DSNB_{test}$.

Come si nota dalla tabella 3.4, i risultati sono molto inferiori rispetto a quelli trovati allenando un modello su un dataset binario.

Modello	**Accuracy**	**Precision**	**Recall**	**F1**
Multiclass logistic regression (LRNB)	0.38	0.36	0.40	0.35

Tabella 3.4: Tabella dei risultati per la multiclass logistic regression

3.5 Modelli basati su linear regression

Perchè non si può usare la linear regression per i problemi di classificazione? Se pensiamo ad una classificazione binaria, la linear regression oltre a

restituire valori compresi tra 0 e 1, potrebbe restituire anche valori superiori a 1 o inferiori a 0. Perchè non cercare di approssimare le predizioni della linear regression tramite un modello di logistic regression e vedere se la concatenazione di questi due modelli riesce a classificare meglio i dati rispetto alla semplice logistic regression?

3.5.1 Chained Estimators

Nella figura 3.1 viene riportata la classe creata per la concatenazione di due modelli differenti.

__init__

Il metodo __init__ serve ad inizializzare la classe: prende come input i due modelli che vogliamo concatenare e li salva nelle variabile est1, est2.

tune

Il metodo tune prende come input:

- n_estimators che decide su quale dei due modelli fare hyperparameter tuning.
- grid che è il dizionario utilizzato dalla grid search.
- scoring che è il valutatore di performance usato in grid search.
- X che sono i dati di allenamento.
- y che è il vettore target di allenamento.

In caso di tuning degli hyperparametri del primo modello, X e y sono usati direttamente come input per la grid search. Nel caso, invece, di tuning del secondo modello, X e y vengono divisi in due parti della stessa dimensione e salvati in due nuove variabili, rispettivamente: X_train_1, X_train_2, y_train_1, y_train_2. Il primo modello verrà allenato su

X_train_1 e y_train_1, dopodichè gli input per la grid search saranno modello1.predict(X_train_2) e y_train_2; questo perchè il secondo modello deve imparare a lavorare sui risultati del primo modello. Perchè dividiamo in due parti sia X che y? Perchè se non lo facessimo, alleneremmo il primo modello su X e y, poi dovremmo utilizzare per la grid search modello1.predict(X), y. Utilizzare delle predizione sui dati di allenamento non è consigliato, perchè a causa dell'overfitting si otterranno sempre le migliori performance possibili e non quelle reali, andando in questo modo a rovinare il tuning del secondo modello.

Infine, i parametri trovati vengono prima salvati nel modello e poi ristituiti in output.

fit

Il metodo fit serve per allenare entrambi i modelli: prende come input i dati di allenamento X e il vettore target y. Sia X che y vengono divisi in due parti della stessa dimensione e salvati in due nuove variabili, rispettivamente: X_train_1, X_train_2, y_train_1, y_train_2. Il primo modello viene allenato direttamente su X_train_1 e y_train_1, mentre il secondo viene allenato su modello1.predict(X_train_2) e y_train_2.
X e y vengono divisi in due parti per lo stesso motivo esposto sopra.

predict

Il metodo predict serve per compiere il task imparato in allenamento su dei dati mai visti e senza un vettore target. Per come verrà utilizzato in questa tesi, il primo modello predice dei valori a partire dall'input X, mentre il secondo classifica i valori predetti dal primo. In output si restituisce il vettore dei valori classificati.

set_params

Il metodo set_params serve a configurare i parametri dei due modelli: prende in input il numero del modello da configurare(1 o 2) e un dizionario

avente come chiavi il nome dei parametri e come valori i valori scelti per quei parametri.

```
class ChainEstimators():
  def __init__(self,est1,est2):
    self.est1 = est1
    self.est2 = est2
  def tune(self,n_estimator,grid,scoring,X,y):
    if n_estimator == 1:
      est = self.est1
      pred = X
    else :
      est = self.est2
      X_train_1, X_train_2, y_train_1, y_train_2 = train_test_split(X, y, test_size=0.5)
      self.est1.fit(X_train_1,y_train_1)
      pred = self.est1.predict(X_train_2)
      y = y_train_2
    grid_search = GridSearchCV(estimator=est, param_grid=grid, n_jobs=-1, scoring=scoring)
    grid_result = grid_search.fit(pred,y)
    est.set_params(**grid_result.best_params_)
    return [grid_result.best_score_,grid_result.best_params_]
  def fit(self,X,y):
    X_train_1, X_train_2, y_train_1, y_train_2 = train_test_split(X, y, test_size=0.5)
    self.est1.fit(X_train_1,y_train_1)
    y_2 = self.est1.predict(X_train_2)
   #abbiamo bisogno di un array in colonna, quindi si utilizza il metodo reshape
    self.est2.fit(y_2.reshape(-1,1),y_train_2)
    return self
  def predict(self,X):
    mid_pred = self.est1.predict(X)
    #abbiamo bisogno di un array in colonna, quindi si utilizza il metodo reshape
    final_pred = self.est2.predict(mid_pred.reshape(-1,1))
    return final_pred
  def set_params (self,n_estimator,params):
    if n_estimator == 1:
      est = self.est1
    else:
      est = self.est2
    est.set_params(**params)
    return self
```

Figura 3.1: Codice per concatenare due modelli

3.5.2 Modello su dataset binario

Nel caso di dataset binario, il nostro target potrà soltanto avere valori 1 (presenza di low achievement) o 0 (assenza di low achievement). Sul DSB_{train}

è stato allenato un modello concatenato composto da linear regression e logistic regression.

Hyperparameter Tuning

Per definire i migliori modelli concatenati, come primo step sono stati tunati gli hyperparametri di entrambi, con tecnica grid search, descritta nel paragrafo 1.5. La linear regression base non ha alcun hyperparametro; invece la descrizione degli hyperparametri della logistic regression e di come vengono tunatisi trova al paragrafo 3.4.1.

L'utilizzo di f1 o accuracy come metrica di scoring non cambia i risultati ottenuti in fase di test.

Sono riportati in tabella 3.5 i valori trovati in fase di hyperparameter tuning e utilizzati per la definizione dei modelli, che ora sono pronti per essere allenati.

Modello	**solver**	**C**
Modello concatenato	saga	0.094

Tabella 3.5: Tabella degli hyperparametri per il modello concatenato su dataset binario

Risultati ottenuti

I modelli concatenati vengono poi allenati su DSB_{train} e testati su DSB_{test}.

Modello	**Accuracy**	**Precision**	**Recall**	**F1**
Modello concatenato	0.73	0.58	0.59	0.58

Tabella 3.6: Tabella dei risultati per il modello concatenato su dataset binario

Come si nota dalla tabella 3.6, i risultati sono inferiori rispetto a quelli trovati allenando un semplice modello di logistic regression sullo stesso da-

taset binario. Che cosa potrebbe aver intaccato le performance? Mentre la logistic regression viene implementata di default con regolarizzazione, la linear regression no. Potrebbe quindi essere un problema di eccessivo overfitting nel primo modello. Si decide quindi di implementare un secondo modello concatenato formato da linear regression regolarizzata e logistic regression.

Modelli concatenati con regolarizzazione

Sono stati creati due modelli concatenati: uno avente come primo modello una linear regression con regolarizzazione *l1* (MCB-L1), l'altro avente come primo modello una linear regression con regolarizzazione *l2* (MCB-L2).

Il primo step è sempre quello di fare il tuning degli hyperparametri con tecnica grid search. Per la linear regression con regolarizzazione *l1* c'è un solo parametro da ottimizzare: **alpha** descritto nel paragrafo 1.5.1. Come tutti i parametri a valori reali, viene utilizzato l'algoritmo nel paragrafo 3.4.1. Nella linear regression con regolarizzazione *l2* ci sono due parametri da ottimizzare: **alpha** e **solver**, anche loro descritti nel paragrafo 1.5.1. Per l'hyperparametro solver sono stati scelti sei dei sette possibili valori che può assumere, perchè il solver lbfgs ha bisogno di impostazioni aggiuntive per poter funzionare. Alpha, viene invece ottimizzato come sopra. Per entrambi i modelli di linear regression viene utilizzata il MSE come metrica di scoring. Le logistic regression di entrambi i modelli vengono tunate come nel paragrafo 3.4.1.

In tabella 3.7 troviamo i valori degli hyperparametri e i risultati di questi modelli, che sono decisamente migliori se paragonati ai risultati del modello concatenato senza regolarizzazione; si può affermare con certezza che era l'overfitting ad impedire che il modello concatenato performasse al suo meglio. Tra i due tipi di regolarizzazione utilizzati, si notano delle performance leggermente migliori con regolarizzazione di tipo *l2*.

Modello	solver linear reg	alpha	solver logistic reg	C	Accuracy	Precision	Recall	F1
MCB-L1		0.0004	saga	0.7	0.76	0.62	0.66	0.64
MCB-L2	lsqr	67.84	saga	5.3	0.76	0.62	0.67	0.64

Tabella 3.7: Tabella degli hyperparametri e dei risultati per i modelli concatenati con regolarizzazione su dataset binario

3.5.3 Modello su dataset non binario

Nel caso di dataset non binario, il nostro target potrà avere valori compresi tra 1 e 5. Sul $DSNB_{train}$ è stato allenato un modello concatenato composto da linear regression e multiclass logistic regression.

Hyperparameter tuning

Per definire il migliori modelli concatenati, come primo step sono stati tunati gli hyperparametri di entrambi, con tecnica grid search, descritta nel paragrafo 1.5. La linear regression base non ha alcun hyperparametro; invece la descrizione degli hyperparametri della multiclass logistic regression e di come vengono tunati si trova al paragrafo 3.4.2.

Sono riportati in tabella 3.8 i valori trovati in fase di hyperparameter tuning e utilizzati per la definizione dei modelli, che ora sono pronti per essere allenati.

Modello	solver	C	multi_class
Modello concatenato	lbfgs	0.094	multinomial

Tabella 3.8: Tabella degli hyperparametri per il modello concatenato su dataset non binario

Risultati ottenuti

I modelli concatenati vengono poi allenati su $DSNB_{train}$ e testati su $DSNB_{test}$.

Modello	Accuracy	Precision	Recall	F1
Modello concatenato	0.36	0.34	0.37	0.35

Tabella 3.9: Tabella dei risultati per il modello concatenato su dataset non binario

Come si nota dalla tabella 3.9, i risultati sono inferiori rispetto a quelli trovati allenando un semplice modello di multiclass logistic regression sullo stesso dataset. Anche in questo caso, il problema potrebbe essere un eccessivo overfitting nel primo modello. Si decide quindi di implementare un secondo modello concatenato formato da linear regression regolarizzata e multiclass logistic regression.

Modelli concatenati con regolarizzazione

In questo caso, è stato creato un solo modello concatenato, avente come primo modello una linear regression con regolarizzazione *l2* (MCNB-L2).

Il primo step è sempre quello di fare il tuning degli hyperparametri con tecnica grid search. Per il tuning delle linear regression, vedere il paragrafo 3.5.2 Le multiclass logistic regression del modello viene tunata come nel paragrafo 3.4.2.

In tabella 3.10 troviamo i valori degli hyperparametri e i risultati del modello, che sono decisamente migliori se paragonati ai risultati del modello concatenato senza regolarizzazione, tabella 3.9; si può affermare con certezza che era l'overfitting ad impedire che il modello concatenato performasse al suo meglio. I risultati rimangono comunque molto inferiori rispetto a quelli trovati allenando un modello sul dataset binario.

Modello	solver linear reg	alpha	solver logistic reg	C	Accuracy	Precision	Recall	F1
MCNB-L2	lsqr	1^{-10}	lbfgs	1.98	0.39	0.37	0.39	0.37

Tabella 3.10: Tabella degli hyperparametri e dei risultati per i modelli concatenati con regolarizzazione su dataset non binario

Conclusioni

La domanda a cui volevamo rispondere con questa tesi è: le performance riportate dai modelli di machine learning utilizzati da A.Zanellati et al.[19] per il riconoscimento di low achievement scolastico, possono essere simili o peggiori a quelle che si potrebbero ottenere con l'utilizzo di modelli statistici di apprendimento più tradizionali, come linear o logistic regression?

In tabella 3.11 sono riportati tutti i principali risultati del paper [19] e di questa tesi. I modelli con le migliori performance sono, per distacco, quelli che utilizzano reti neurali; nessun altro modello riesce a raggiungere la loro precision e recall. è noto però che i modelli neurali siano meno "spiegabili" e questo rappresenta un limite rispetto all'uso del modello anche in termini diagnostici. Prendendo invece in esame il random forest, MCB-L2 e LRB riescono ad avvicinarsi alle sue performance. I modelli che invece sono stati allenati sul dataset non binario hanno delle performance non ottimali. Quest'ultime possono esere giustificate dal fatto che le 5 classi avranno delle forti sovrapposizioni tra di loro, che porterà a sbagliare classificazione per diversi input.

Siamo quindi riusciti, tramite la combinazione di modelli statistici di apprendimento più tradizionale, ad avvicinarci ai risultati ottenuti dal random forest nel rilevamento di low achievement. In più abbiamo anche scoperto che la predizione del voto alle prove invalsi della seconda superiore, a partire dalla prova invalsi della quinta elementare è un problema non adatto ai metodo di classificazione tradizionali.

Modello	**Accuracy**	**Precision**	**Recall**	**F1**
FTT	0.78	0.77	0.78	0.77
CE	0.76	0.76	0.76	0.76
Random Forest	0.77	0.62	0.67	0.64
MCB-L2	0.76	0.62	0.67	0.64
LRB	0.77	0.65	0.61	0.63
MCNB-L2	0.39	0.37	0.39	0.37
LRNB	0.38	0.36	0.40	0.35

Tabella 3.11: Risultati finali della tesi e dell'articolo di A.Zanellati et al[19].

Appendice A

Elenco feature presenti nel dataset INVALSI

Nome feature	**Tipo feature**
CODICE_SCUOLA	int64
CODICE_CLASSE	int64
campione	int64
livello	int64
prog	int64
CODICE_STUDENTE	int64
pu_ma_gr	int64
Livelli Grado 10	int64
CODICE_PLESSO	float64
pu_ma_no	float64
Fattore_correzione_new	float64
Cheating	float64
PesoClasse	float64
PesoScuola	float64
PesoTotale_Matematica	float64
WLE_MAT	float64

WLE_MAT_200	float64
WLE_MAT_200_CORR	float64
pu_ma_no_corr	float64
n_stud_prev	float64
n_classi_prev	float64
Punteggio WLE grado 13	float64
DROPOUT	bool
macrotipologia	object
SIDI_Invalsi	object
sesso	object
mese	object
anno	object
luogo	object
eta	object
codice_orario	object
freq_asilo_nido	object
freq_scuola_materna	object
luogo_padre	object
titolo_padre	object
prof_padre	object
luogo_madre	object
titolo_madre	object
prof_madre	object
voto_scritto_ita	object
voto_orale_ita	object
voto_scritto_mat	object
voto_orale_mat	object
D1	object
D2	object
D3_a	object

D3_b	object
D4_a	object
D4_b	object
D4_c	object
D4_d	object
D5_a	object
D5_b	object
D6	object
D7_a	object
D7_b	object
D8	object
D9	object
D10_a	object
D10_b1	object
D10_b2	object
D10_b3	object
D11_a	object
D11_b	object
D12_a	object
D12_b	object
D13_a	object
D13_b	object
D13_c	object
D14	object
D15	object
D16_a	object
D16_b	object
D16_c	object
D16_d	object
D17_a	object

D17_b	object
D18	object
D19_a	object
D19_b	object
D20	object
D21	object
D22	object
D23_a	object
D23_b	object
D23_c	object
D23_d	object
D24_a	object
D24_b	object
D25	object
D26_a	object
D26_b	object
D26_c	object
D26_d	object
regolarità	object
cittadinanza	object
cod_provincia_ISTAT	object
sigla_provincia_istat	object
Nome_reg	object
Cod_reg	object
Areageo_3	object
Areageo_4	object
Areageo_5	object
Areageo_5_Istat	object
Pon	object

Tabella A.1: Tabella delle feature presenti nel dataset INVALSI prima dell'operazione di encoding.

Bibliografia

[1] Ricci, R. (2019). La dispersione scolastica implicita. La dispersione scolastica implicita, 41-48.

[2] Murphy, K. P. (2012). Machine learning: a probabilistic perspective. Cambridge, MA, MIT Press.

[3] Mitchell, T. M. (1997). Machine learning (Vol. 1). McGraw-hill New York.

[4] Powers, D. (2007). Evaluation: From precision, recall and F-factor to ROC, informedness, markedness & correlation (Tech. Rep.). Adelaide, Australia.

[5] Goodfellow, I., Bengio, Y.„ Courville, A. (2016). Deep Learning. MIT Press.

[6] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., Vanderplas, J., Joly, A., Holt, B. & Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project (cite arxiv:1309.0238)

[7] Bishop, C. M. (2006). Pattern recognition and machine learning. New York: Springer. ISBN: 0387310738 9780387310732

[8] Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In Proceedings of COMPSTAT'2010 (pp. 177-186). Physica-Verlag HD.

[9] Mark Schmidt, Nicolas Le Roux, Francis Bach. Minimizing Finite Sums with the Stochastic Average Gradient. Mathematical Programming, Springer Verlag, 2017, 162 (1-2), pp.83-112. 10.1007/s10107- 016-1030-6 . hal-00860051v2

[10] Defazio, A., Bach, F., & Lacoste-Julien, S. (2014). SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. Advances in neural information processing systems, 27.

[11] Liu, D. C., & Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. Mathematical programming, 45(1), 503-528.

[12] Wright, S. J. (2015). Coordinate descent algorithms. Mathematical Programming, 151(1), 3-34.

[13] Wright, S., & Nocedal, J. (1999). Numerical optimization. Springer Science, 35(67-68), 7.

[14] Wikipedia contributors. (2022, July 16). Multinomial logistic regression. In Wikipedia, The Free Encyclopedia. Retrieved 16:28, September 22, 2022, from https://en.wikipedia.org/w/index.php?title=Multinomial_logistic_regression&oldid=1098653

[15] Kim, S., Koh, K., Lustig, M., Boyd, S. & Gorinevsky, D. (2008). An Interior-Point Method for Large-Scale l1-Regularized Least Squares. Selected Topics in Signal Processing, IEEE Journal of, 1, 606–617.

[16] Batista, G. E., Prati, R. C., & Monard, M. C. (2004). A study of the behavior of several methods for balancing machine learning training data. ACM SIGKDD explorations newsletter, 6(1), 20-29.

[17] Kotsiantis, S., Kanellopoulos, D., & Pintelas, P. (2006). Handling imbalanced datasets: A review. GESTS international transactions on computer science and engineering, 30(1), 25-36.

[18] Prove nazionali INVALSI. (31 maggio 2022). Wikipedia, L'enciclopedia libera. Tratto il 24 settembre 2022, 09:23 da //it.wikipedia.org/w/index.php?title=Prove_nazionali_INVALSI&oldid=127663633.

[19] Zanellati, A., Zingaro, S. P., & Gabbrielli, M. (2022). Student Low Achievement Prediction. In International Conference on Artificial Intelligence in Education (pp. 737-742). Springer, Cham.

Ringraziamenti

Ringrazio il mio correlatore, il Dottor Andrea Zanellati per l'aiuto dato durante la stesura della tesi, la grande disponibilità e le belle parole sempre spese nei miei confronti.

Ringrazio i miei genitori per aver sempre creduto in me, per non avermi mai fatto mancare niente, per essere sempre stati lì per me e per avermi sempre dato un abbraccio nel momento giusto. Vi voglio bene.

Ringrazio mio fratello Andrea per essere da sempre il mio punto di riferimento e la mia ispirazione. Grazie per avermi fatto scoprire tutte quelle che sono le mie passioni più grandi. Sei il fratello maggiore migliore che potessi desiderare, ti voglio un mondo di bene.

Ringrazio la nonna Lola e il nonno Giovanni, per essermi sempre stati vicino, per tutti i pranzi, le chiacchere e le passeggiate al mare fatte insieme. Vi voglio bene.

Ringrazio il nonno Paolo e la nonna Franca, per essere sempre stati un esempio di forza e di gentilezza. Siete sempre con me nel mio cuore, vi voglio bene.

Ringrazio Mattia, Gabriele, Saverio e Gregorio i miei amici speciali. Grazie per tutte le belle avventure vissute insieme, per le battute e la spensieratezza donatami durante le nostre uscite. Vi voglio bene.

Ringrazio Luca e Andrea, compagni di mangiate e di giochi da tavolo. So di poter sempre contare su di voi e per questo vi ringrazio tantissimo. Vi voglio bene.

Ringrazio Sandro, Natascia, Matilda, Nerio, Liliana e Paolo per avermi

accettato nella loro famiglia e per farmi sempre sentire a casa. Vi voglio bene.

Ringrazio la dottoressa Erika Corigio per avermi fatto capire quanto io sia importante e per avermi fatto riacquisire la fiducia in me stesso.

Infine vorrei ringraziare Morghi, la mia dolce metà. Grazie per i mille covini che mi fai ogni giorno, grazie per avermi fatto sempre sentire al sicuro e protetto e grazie soprattutto per aver riempito il mio cuore di amore e per amarmi così come sono. Ti amo tantisssimo.