POLITECNICO DI TORINO

POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Matematica

Tesi di Laurea Magistrale

Metodi di Machine Learning per il Rischio di Credito

Relatore
Prof. Patrizia Semeraro
Prof. Edoardo Fadda

Candidato
Simone Lucio Cannavò

2025

Abstract

In questo elaborato vengono presentate diverse tecniche di machine learning applicate al problema del rischio di credito. In tale contesto, risulta particolarmente complesso effettuare inferenza sulla classe dei default, poiché essa è significativamente meno rappresentata rispetto a quella dei non default. Oltre all'impiego di modelli classici di machine learning, viene proposta una metodologia basata su metodi ensemble e ottimizzazioni mirate, con l'obiettivo di migliorare la *recall* della classe dei default. Tale metrica risulta infatti tra le più penalizzate nelle applicazioni di rischio di credito, pur rappresentando un indicatore fondamentale della cautela del modello nell'identificazione dei soggetti a rischio. Un ulteriore obiettivo del lavoro, valido sia per i modelli classici sia per quelli ensemble, è il calcolo della probabilità marginale di default. Il fine ultimo è la definizione di una pipeline robusta, in grado di affrontare efficacemente il problema delle classi sbilanciate e supportare l'analisi predittiva in ambito creditizio.

Contents

Chapter 1

Machine Learning per il rischio di credito

1.1 Introduzione

Nel mondo della finanza è ricorrente che enti pubblici e privati concedano crediti a clienti che necessitano somme di denaro in breve tempo, la restituzione del debito generalmente è articolata in più tranche periodiche, la durata del tempo di rimborso si decide a priori, dipendentemente dal tipo di contratto pattuito con il cliente. Tale fenomeno avviene da sempre, costituendo un ruolo essenziale nel mercato globale. Tuttavia un problema importante in questo contesto è l'incertezza sul comportamento del cliente, concedendo il prestito infatti, non si può essere certi che il cliente riesca a rimborsare il credito ricevuto. Il Machine Learning è considerato un approccio molto utile al fine di prevedere il comportamento dei clienti, l'obiettivo è quello di imparare a distinguere correttamente gli individui in default (non restituiscono il prestito) da i Non default (riescono a restituire il prestito), partendo da un dataset composto da caratteristiche finanziarie, comportamentali e in qualche modo rappresentative dell'atteggiamento dei clienti. Tra le avversità che si affrontano in questo ambito, vi è lo sbilanciamento tra le classi al quale si porrà molta attenzione durante lo svolgimento dell'elaborato. In generale infatti la classe dei default è meno numerosa rispetto a quella dei non default, rendendo molto difficile ai modelli di Machine Learning ottenere buone performance sulla classe degli individui in default. In questo capitolo verrà discusso l'articolo *"Machine Learning in Joint Default Assessment"* [4], in cui la ricerca si basa sul Machine Learning per il rischio di credito, tale elaborato infatti sarà la base della ricerca condotta in questa tesi.

1.2 Revisione delle Metodologie di Valutazione del Rischio di Credito

In questa sezione verranno trattati due dei più importanti paper per la costruzione del corrente elaborato. Nello specifico:

- *Machine Learning techniques in joint default assessment* [4]

- *Integration of unsupervised and supervised machine learning algorithms for credit risk assessment* [8]

Il primo dei due lavori è composto da due analisi. Nella prima i dati sono generati in vitro, i dataset vengono prodotti utilizzando una funzione logistica di parametri θ, in modo tale da ottenere il 20% dei clienti in default e la restante parte non in default. Si sono scelte tali proporzioni per essere in linea con le numeriche del dataset reale della seconda analisi. Vengono scelte 5 relazioni tra le feature:

- **Uniforme Indipendente (UI)**:
  Generata tramite un midello logit con due covariate X_1, X_2 indipendenti e identicamente distribuite secondo $\mathcal{U}[0,1]$. Il vettore dei parametri è:
  $$\theta = [-0.2,\ 1.5,\ -5.0].$$

- **Quadratica (2S)**:
  Generata tramite un modello logit con due covariate, dove $X_1 \sim \mathcal{U}[0,1]$ e $X_2 = X_1^2 + \mathcal{U}[-0.05,\ 0.05]$. I parametri del modello sono:
  $$\theta = [-0.2,\ 0.7,\ -5.5].$$

- **Copula Normale (NC)**:
  Generata tramite un modello logit con due covariate le cui distribuzioni marginali sono $\mathcal{U}[0,1]$, collegate tramite una copula Normale. La matrice di covarianza utilizzata è:
  $$\Sigma = \begin{bmatrix} 0.5 & -0.2 \\ -0.2 & 0.5 \end{bmatrix}.$$
  Il vettore dei parametri è:
  $$\theta = [-0.2,\ 0.5,\ -3.2].$$

- **Copula t (TC)**:
  Generata tramite un modello logit con due covariate uniformi $\mathcal{U}[0,1]$, collegate mediante una copula t di Student con 2 gradi di libertà. I parametri del modello sono:
  $$\theta = [-0.2,\ 0.5,\ -3.2].$$

- **Non Lineare (5NL)**:
  Generata tramite un modello logit con tre covariate $X_1, X_2, X_3 \sim \mathcal{U}[0,1]$. Le covariate utilizzate nel modello sono definite come:
  $$Z_1 = X_1 \cdot X_2 + \mathcal{U}[-0.05,\ 0.05], \quad Z_2 = X_1 \cdot X_3 + \mathcal{U}[-0.05,\ 0.1].$$
  Il vettore dei parametri è:
  $$\theta = [-0.1,\ -1,\ -0.5,\ -0.5,\ -1,\ -1].$$

Per ogni relazione vengono effettuate 10 simulazioni per 4 configurazioni di portafoglio. Dunque vengono generati 200 dataset. Dopo aver ottenuto i dataset vengono addestrati i seguenti modelli:

- Regressione Logistica
- Random Forest
- MLP
- Adaboost
- KNN

Dopo la valutazione dei modelli viene calcolata la probabilità di default e la correlazione, stimate tramite il metodo dei momenti. La ricerca continua con la valutazione di due misure di rischio: Var ed Expected Shortfall. In questo elaborato non viene discussa questa sezione poichè non inerente alla ricerca effetuata nella tesi.

Nella seconda analisi presente in *"Machine Learning in Joint Default Assessment"* [4], viene effetuato lo stesso processo analizzato nella prima fase ma con un dataset reale. Il dataset utilizzato è l'UCI Credit Risk, ampiamente descritto nei prossimi capitoli. L'approccio utilizzato è il medesimo della prima analisi. Vengono addestrati gli stessi modelli al fine di calcolare la probabilità di default, la correlazione e le due misure di rischio discusse in precedenza. Il lavoro di tesi che si sta proponendo è composto da due parti distinte. Nella prima parte sono state utilizzate le tecniche utilizzate in *"Machine Learning in Joint Default Assessment"* [4] , sullo stesso dataset (UCI Credit Risk) ottenendo gli stessi risultati. Dopodiché si è effetuata la stessa analisi su un nuovo dataset (German Credit Dataset) valutando le evidenze ottenute L'obiettivo della ricerca in *"Machine Learning in Joint Default Assessment"*, non è quella di ottenere ottime performance nella classificazione, bensì è quello di provare a stimare in modo corretto le misure di rischio discusse in precedenza. Tuttavia un problema riscontrato nel contesto del rischio di credito riguarda le performance ottenute sulla classe minoritaria. Nel contesto dei dataset sbilanciati infatti, la "recall" viene compromessa in modo particolare. Tale metrica indica quanti default sono stati classificati correttamente rispetto al numero effettivo di default presenti. Ottenere valori bassi dunque, indica che il modello assegna la maggior parte dei default alla classe errata. La perdita derivante dal classificare un cliente in default come non-default è generalmente molto più elevata rispetto all'errore inverso: nel primo caso si subisce l'intera perdita del credito concesso, mentre nel secondo si rinuncia a futuri guadagni (interessi) su un prestito che non sarebbe andato in default.
Un altro settore in cui lo sbilanciamento è un ostacolo importante per i modelli di classificazione è il contesto sanitario. Le numeriche riferite ai test sanitari positivi (virus, malattie sessualmente trasmissibili etc.), sono molto minori rispetto a quelle riferite ai test negativi. Come nel caso del rischio di credito, classificare erroneamente un'osservazione appartenente alla classe minoritaria (ad esempio, un default nel contesto finanziario o un test positivo in ambito sanitario) ed assegnarla a quella maggioritaria comporta un costo significativamente più elevato rispetto all'errore inverso. Nel caso finanziario, ciò può

tradursi in perdite economiche rilevanti, nel contesto sanitario, le conseguenze possono essere persino più gravi, arrivando a compromettere la salute del paziente.
Dunque nella seconda fase della tesi è stata discussa la costruzione di nuova struttura basata su modelli ensemble al fine di migliorare le performance. A supporto di questo obiettivo è stato consultato un altro paper nell'ambito del rischio di credito (Integration of unsupervised and supervised machine learning algorithms for credit risk assessment [8]) da cui è nata l'idea dell'implementazione dei modelli ensemble. L'analisi svolta nel paper appena citato viene svolta su tre dataset:

Credit dataset	**Total**	**Non-defaults / defaults**	**Default ratio**	**Feature dimension**
Chinese P2P	23 435	19 546 / 3 889	16.6%	81
German	1 000	700 / 300	30%	20
Australian	690	307 / 383	55.5%	14

Tabella 1.1: Descrizione dei dataset utilizzati in *"Integration of unsupervised and supervised machine learning algorithms for credit risk assessment"* [8]

Il Dataset cinese P2P è fornito da una piattaforma cinese di prestiti peer-to-peer(finanziamenti senza passare dalla banca) è costituito da 23.435 richiedenti, di cui 3.889 in default e 19.546 non in default. L'arco temporale dei dati raccolti si apre da gennaio 2014 chiudendosi a settembre 2017, i dati sono riportati al primo settembre 2018 così da garantire almeno un anno di storico a tutti i richiedenti. L'importo dei prestiti varia tra 10.000 e 200.000 RMB(Renminbi, valuta ufficiale della cina) , con una media di 50.000 RMB. Il dataset contiene 81 variabili descrittive tra cui: età, genere, stato civile, titolo di studio, assicurazione, mutuo mensile e limite della carta di credito.
Il Dataset tedesco invece contiene 1.000 campioni, di cui 700 non default e 300 default, descritti tramite 20 variabili (3 continue e 17 categoriche), relative a età, occupazione, storia creditizia, saldo dei conti, importo richiesto e finalità del prestito. Tale descrizione del dataset sarà più specifica nel capitolo 3 poiché questo database verrà utilizzato nell'analisi.
Il Dataset australiano è composto da 690 campioni, con 307 richiedenti non in default e 383 in default. Le istanze sono descritte da 14 variabili, di cui 6 continue e 8 categoriche.
Nel paper *"Integration of unsupervised and supervised machine learning algorithms for credit risk assessment"* [8] distinguiamo due fasi principali. Nella prima vengono utilizzati modelli supervisionati di machine learning singoli, per la classificazione dei clienti in default, rispettivamente:

- Regressione logistica
- Alberi decisionali
- Random Forest
- Support Vector Machines
- Gradient boosting decision trees
- k-Nearest Neighbors

- Artificial neural network

Segue una breve descrizione del support vector machines e dell' Artificial neural network, modelli non utilizzati nella tesi e dunque non presenti nel prossimo capitolo (Tecniche di Machine Learning)

Support vector machines è un modello di machine learning supervisionato molto utilizzato nell'ambito del rischio di credito. L'idea è quella di proiettare i dati in uno spazio delle caratteristiche ad elevata dimensionalità in modo da renderli separabili da un iperpiano. Si cerca infatti l'iperpiano con il margine massimo che possa separare i dati nelle classi di riferimento.

Le reti neurali sono uno strumento complesso, utilizzate in moltissimi contesti. IL modello presenta una struttura a livelli. Ogni livello è costituito da singoli neuroni, i livelli sono rispettivamente:

- Livello di input
- Livelli nascosti
- Livello di output

Il livello di input rappresenta il punto di ingresso della rete, in cui il modello riceve le caratteristiche. I livelli nascosti costituiscono la parte operativa della rete. Ciascun neurone combina le informazioni ricevute applicando dei pesi, sommandoli e passando il risultato attraverso una funzione di attivazione (come la sigmoide o la ReLU) [6], nel contesto del rischio di credito viene utilizzata la sigmoide.
Questo processo consente alla rete di catturare relazioni non lineari tra le variabili. Al termine del processo, il livello di output restituisce il risultato finale della rete. Tale valore viene calcolato a partire dalle elaborazioni dei livelli precedenti e riflette la sintesi delle informazioni apprese dalla rete.
La seconda fase di Integration of unsupervised and supervised machine learning algorithms for credit risk assessment [8] è fondamentale per la ricerca effettuata nella tesi. Al fine di migliorare le perfomance è stata costruita una pipeline che punti a far lavorare insieme i modelli invece di utilizzare un approccio individualistico. Alla base della struttura proposta dagli autori del paper è stato utilizzato il k-means per evidenziare la presenza di cluster nei dati di input. Subito dopo il k-means, sono stati costruiti i modelli individuali, discussi in precedenza. Le probabilità stimate dai modelli, diventano i dati di input per un altro modello di unsupervised learninig.
Tramite gli output dei modelli di supervised learning infatti viene costruito il SOM(Self-organizing maps). Quest ultimo modello infatti è una rete neurale non supervisionata. L'idea è quella di mappare i dati di input in una griglia bidimensionale. Ogni vettore di input viene proiettato sul neurone più simile ad esso, noto come Best Matching Unit. I neuroni nell'intorno del best match unit vengono aggiornati in modo da ottenere per ogni iterazione pesi sempre più simili ai dati di input. Al termine delle iterazioni tutti i vettori di input sono proiettati nei neuroni della griglia, (un neurone sicuramente sarà associato ad almeno un vettore di input), l'etichetta del neurone viene infatti assegnata tramite l'utilizzo del major voting.

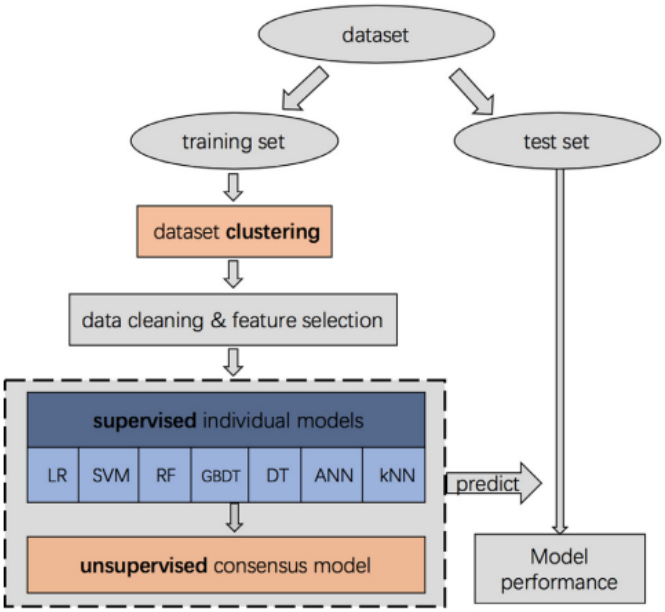

Figura 1.1: Struttura del modello ensemble utilizzato in Integration of unsupervised and supervised machine learning algorithms for credit risk assessment [8]

La struttura riassunta in figura 1.1 conduce a risultati più stabili rispetto ai modelli individuali, proprio per questo si è deciso di seguire la strada dei modelli ensemble nella tesi, seppur non utilizzando il Som (seppur efficiente, l'idea del som è di semplificare database molto ampi), ma provando diversi modelli supervisionati come metamodello finale, confrontandoli e determinando infine il più performante.

Chapter 2

Tecniche di Machine Learning

In questo capitolo vengono illustrate tutte le tecniche di machine learning supervisionato e non supervisionato utilizzate nel corso del lavoro. Come si vedrà in seguito saranno utilizzati due approcci distinti, nella prima fase verranno costruiti e testati i modelli individuali, nella seconda fase invece verrà costruita una pipeline strutturata in cui più modelli lavorano insieme al fine di migliorare le performance ottenute rispetto al primo approccio, focalizzandosi specialmente sull'aumento della recall della classe 1. In questo capitolo però non si farà distinzione tra i due approcci, l'obiettivo infatti sarà quello di presentare il funzionamento di tutte le techince utilizzate, sia nella prima che nella seconda fase.

2.1 Random Forest

In questa sezione viene illustrata l'idea del funzionamento della random forest.

Per un certo vettore di caratteristiche (features) $\mathbf{x}$, si considerino i valori di predizione $Z_b = g_{T_b}(\mathbf{x})$ per $b = 1,2,\ldots,B$, ottenuti da insiemi di addestramento indipendenti $T_1,\ldots,T_B$. L'idea chiave delle *Random Forest* è utilizzare il bagging per generare più insiemi di addestramento, ma ottenendo alberi decorrelati tra loro. Dunque per ogni insieme T_b^*, si costruisce un albero decisionale usando solo un sottoinsieme casuale di $m \leq p$ variabili (dove p è il numero totale di caratteristiche). Questa strategia riduce la probabilità che le variabili forti vengano selezionate ai livelli iniziali, diminuendo la correlazione tra alberi e migliorando così la performance predittiva dell'ensemble. Il processo si riassume come segue.
Dati un insieme di addestramento $\tau = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, il numero di alberi B e il numero di variabili da selezionare m, si generano B insiemi bootstrap, per ciascuno dei quali si selezionano m variabili casuali (senza ripetizione) e si costruisce un albero decisionale usando solo queste variabili. Infine si restituisce l'ensemble di alberi ottenuto, che costituisce la Random Forest. La predizione nel caso della classificazione si ottiene con la tecnica del major voting, la classe con più voti viene assegnata per ogni elemento. Le informazioni riportate di seguito sono tratte dal libro *[Data Science and Machine Learning]* [2], in particolare nel capitolo [8].

2.2 Balanced random Forest

Di seguito viene proposta una variante dell'algoritmo appena discusso, tale modello viene denominato "Balanced Random Forest". L'algoritmo è implementato nella libreria imblearn di phython- Il principio di funzionamento è uguale a quello discusso nella sezione precedente, la differenza che si riscontra con la Random Forest è la struttura dell'insieme di addestramento di ogni albero. Infatti ogni albero della random forest è addestrato su un insieme bootstrap bilanciato, permettendo così al modello di poter distinguere più facilmente tra le classi.

2.3 Regressione Logistica

La regressione logistica (logit) è un modello lineare generalizzato in cui, dato un vettore di caratteristiche $\boldsymbol{x} \in \mathbb{R}^p$, la risposta casuale Y segue una distribuzione di Bernoulli condizionata alla forma $Y \sim \text{Ber}(h(\boldsymbol{x}^\top \boldsymbol{\beta}))$, dove $h(u) = \frac{1}{1+e^{-u}}$ è la funzione logistica.

Il parametro $\boldsymbol{\beta}$ viene appreso dai dati di addestramento massimizzando la verosimiglianza delle risposte osservate, oppure, in modo equivalente, minimizzando la funzione di perdita di entropia incrociata:

$$-\frac{1}{n}\sum_{i=1}^{n} \ln g(y_i \mid \boldsymbol{\beta}, \boldsymbol{x}_i),$$

dove

$$g(y = 1 \mid \boldsymbol{\beta}, \boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{x}^\top \boldsymbol{\beta}}}, \quad g(y = 0 \mid \boldsymbol{\beta}, \boldsymbol{x}) = \frac{e^{-\boldsymbol{x}^\top \boldsymbol{\beta}}}{1 + e^{-\boldsymbol{x}^\top \boldsymbol{\beta}}}.$$

In particolare, il rapporto logaritmico tra le probabilità (log-odds ratio) è:

$$\ln \frac{g(y = 1 \mid \boldsymbol{\beta}, \boldsymbol{x})}{g(y = 0 \mid \boldsymbol{\beta}, \boldsymbol{x})} = \boldsymbol{x}^\top \boldsymbol{\beta}.$$

Pertanto, la superficie di decisione, ovvero l'insieme dei punti per cui le due classi hanno uguale probabilità, è il piano $\boldsymbol{x}^\top \boldsymbol{\beta} = 0$.

Il classificatore finale assegna l'osservazione alla classe con probabilità stimata maggiore:

$$\hat{y} = \arg \max_{j \in \{0,1\}} g(y = j \mid \boldsymbol{\beta}, \boldsymbol{x}).$$

Il modello può essere esteso al caso multiclasse. In questa sezione è stato discusso solo quello binario poichè coerente con il caso studio analizzato nella tesi. Anche per la regressione logistica le informazioni sono state apprese da *[Data Science and Machine Learning]* di [Dirk P. Kroese, Zdravko I. Botev, Thomas Taimre, Radislav Vaisman].

2.4 KNN

Sia $\tau = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ il set di addestramento, con $y_i \in \{0, \ldots, c-1\}$, e sia $\boldsymbol{x}$ un nuovo vettore di caratteristiche da classificare. Si definiscano $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(n)}$ i vettori ordinati

in base alla loro vicinanza a $\boldsymbol{x}$ secondo una distanza, ad esempio la distanza euclidea $\|\boldsymbol{x} - \boldsymbol{x}_i\|$.

Sia quindi:

$$\tau(\boldsymbol{x}) := \{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(K)}, y^{(K)})\}$$

l'insieme dei K punti del set di addestramento più vicini a $\boldsymbol{x}$.

La regola di classificazione dei *K-nearest neighbors* assegna $\boldsymbol{x}$ alla classe più frequente tra quelle contenute in $\tau(\boldsymbol{x})$. In caso di parità tra due o più classi, si seleziona una di esse in modo casuale con probabilità uguale. Il principio di funzionamento di tale algoritmo è stato appreso da *[Data Science and Machine Learning]* di [Dirk P. Kroese, Zdravko I. Botev, Thomas Taimre, Radislav Vaisman].

Boosting: Idea Generale

Nel presente lavoro di tesi vengono implementati e analizzati diversi algoritmi basati sulla tecnica del boosting, tra cui:

- XGBoost
- LightGBM
- CatBoost
- AdaBoost

Tutti questi modelli appartengono alla famiglia dei metodi di **boosting**, una tecnica di ensemble learning ampiamente utilizzata per migliorare le prestazioni predittive dei modelli deboli. Di seguito, in accordo con quanto riportato in *Data Science and Machine Learning* [2] viene fornita una trattazione generale del boosting e una descrizione degli algoritmi che lo adottano.

Il **boosting** è un'idea potente che mira a migliorare l'accuratezza di qualsiasi algoritmo di apprendimento, specialmente se si usano *weak learners*, cioè modelli semplici. Gli alberi decisionali poco profondi sono esempi tipici di weak learners.

Il boosting, inizialmente sviluppato per la classificazione binaria, può essere esteso facilmente anche alla classificazione multiclasse e alla regressione. Come il bagging, il boosting usa un ensemble di predittori. Tuttavia, a differenza del bagging dove i modelli sono addestrati in parallelo su campioni bootstrappati, nel boosting i modelli vengono addestrati in sequenza: ogni modello corregge gli errori del precedente.

Boosting per la Regressione

Nel caso della regressione con perdita quadratica $\text{Loss}(y, \hat{y}) = (y - \hat{y})^2$, il boosting segue la seguente procedura:

Input: Set di addestramento $\tau = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, numero di iterazioni B, parametro di shrinkage γ.

Output: Funzione di predizione potenziata $g_B(x)$.

1. Inizializza $g_0(x) \leftarrow \frac{1}{n} \sum_{i=1}^n y_i$.

2. Per $b = 1$ a B:

   (a) Calcola i residui: $e_i^{(b)} = y_i - g_{b-1}(\boldsymbol{x}_i)$.

   (b) Costruisci il dataset $\tau_b = \{(\boldsymbol{x}_i, e_i^{(b)})\}_{i=1}^n$.

   (c) Adatta una funzione h_b ai dati τ_b.

   (d) Aggiorna: $g_b(x) \leftarrow g_{b-1}(x) + \gamma h_b(x)$.

3. Restituisci $g_B(x)$.

Interpretazione come Discesa del Gradiente

Il parametro γ è interpretabile come il passo nella direzione del gradiente negativo della perdita quadratica. Infatti, il gradiente negativo è:

$$-\frac{\partial \text{Loss}(y_i, z)}{\partial z}\bigg|_{z=g_{b-1}(x_i)} = 2(y_i - g_{b-1}(x_i)) = 2e_i^{(b)}.$$

Gradient Boosting Generale

Input: Set $\tau = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, numero di iterazioni B, funzione di perdita differenziabile $\text{Loss}(y, \hat{y})$, parametro di passo γ.

Output: Funzione potenziata $g_B(x)$.

1. Inizializza $g_0(x) \leftarrow 0$.

2. Per $b = 1$ a B:

   (a) Per ogni $i = 1, \ldots, n$, calcola:

   $$r_i^{(b)} \leftarrow -\frac{\partial \text{Loss}(y_i, z)}{\partial z}\bigg|_{z=g_{b-1}(x_i)}.$$

   (b) Approssima il gradiente negativo:

   $$h_b = \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^n (r_i^{(b)} - h(x_i))^2.$$

   (c) Aggiorna:

   $$g_b(x) \leftarrow g_{b-1}(x) + \gamma h_b(x).$$

3. Restituisci $g_B(x)$.

AdaBoost per la Classificazione Binaria

AdaBoost considera una variabile risposta binaria $y_i \in \{-1,1\}$ e costruisce una sequenza di predittori:

$$g_B(x) = \sum_{b=1}^{B} \alpha_b c_b(x),$$

dove $c_b(x) \in \{-1,1\}$ è un classificatore debole e $\alpha_b \in \mathbb{R}^+$.
Input: Set $\tau = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, numero di iterazioni B.
Output: Funzione predittiva $g_B(x)$.

1. Inizializza $g_0(x) \leftarrow 0$.
2. Inizializza i pesi: $w_i^{(1)} \leftarrow \frac{1}{n}$ per ogni i.
3. Per $b = 1$ a B:
   (a) Trova $c_b \in \mathcal{C}$ che minimizza la perdita zero–uno pesata:
   $$c_b = \arg\min_{c \in \mathcal{C}} \ell_\tau^{(b)}(c) = \arg\min_{c \in \mathcal{C}} \frac{\sum_{i=1}^n w_i^{(b)} \cdot \mathbf{1}\{c(x_i) \neq y_i\}}{\sum_{i=1}^n w_i^{(b)}}.$$
   (b) Calcola il peso α_b:
   $$\alpha_b = \frac{1}{2} \ln\left(\frac{1 - \ell_\tau^{(b)}(c_b)}{\ell_\tau^{(b)}(c_b)}\right).$$
   (c) Aggiorna i pesi:
   $$w_i^{(b+1)} = w_i^{(b)} \cdot \exp(-y_i \alpha_b c_b(x_i)).$$
4. Restituisci la funzione:
   $$g_B(x) = \sum_{b=1}^{B} \alpha_b c_b(x).$$

Regola di Classificazione Finale

Dopo aver ottenuto $g_B(x)$, la classificazione finale viene effettuata tramite:

$$\hat{y} = \text{sign}\left(\sum_{b=1}^{B} \alpha_b c_b(x)\right).$$

Nei primi passi, i pesi $w_i^{(1)}$ sono uguali per ogni osservazione. A ogni iterazione, i pesi degli esempi classificati in modo errato aumentano, mentre quelli corretti diminuiscono. Ciò permette ai classificatori successivi di concentrarsi sugli esempi difficili.

XGBoost: Estensione del Gradient Boosting

XGBoost (Extreme Gradient Boosting) è una versione ottimizzata e regolarizzata del gradient boosting, progettata per essere estremamente veloce, efficiente e accurata.

- Usa una **funzione obiettivo di secondo ordine**, sfruttando il gradiente e la derivata seconda (hessiana) della perdita.
- Integra **regolarizzazione L1 e L2** per prevenire overfitting.
- Supporta il **parallelismo**, l'ottimizzazione tramite histogrammi, e la gestione nativa di **valori mancanti**.
- Utilizza una strategia di **potatura post-order** per la costruzione degli alberi, invece del pre-pruning.
- Include funzioni di **early stopping** e **gestione della memoria** efficienti.

Differenze principali rispetto ad AdaBoost e Gradient Boosting:

- Rispetto ad **AdaBoost**, XGBoost è molto più flessibile, supporta regressione, classificazione e ranking, e non si basa sulla perdita esponenziale.
- Rispetto al **Gradient Boosting** tradizionale, XGBoost è più efficiente (grazie a ottimizzazioni ingegneristiche) e robusto (grazie alla regolarizzazione e gestione dei dati).

XGBoost: Funzionamento e Approccio tramite Istogrammi

Il principio di funzionamento del Xgboost è stato riportato da *"XGBoost: A Scalable Tree Boosting System"* [7].
XGBoost (Extreme Gradient Boosting) è un algoritmo di boosting sviluppato per massimizzare le prestazioni predittive riducendo al minimo i costi computazionali, e costituisce una delle implementazioni più utilizzate e potenti del gradient boosting. Viene ottimizzata una funzione obiettivo composta da due termini: un termine di perdita (ad esempio log-loss per classificazione o square-loss per regressione) e un termine di regolarizzazione $\Omega(f)$ che penalizza la complessità del modello. Questo consente di bilanciare l'accuratezza del fit con la capacità di generalizzazione.

Ad ogni iterazione, il nuovo albero $f_t(x)$ viene costruito per approssimare la funzione obiettivo tramite una espansione di secondo ordine (Taylor) della perdita. Per ciascun campione, si calcolano il gradiente g_i e la hessiana h_i della funzione di perdita rispetto alla predizione corrente. L'albero viene quindi costruito per massimizzare il guadagno regolarizzato del criterio di splitting, definito in termini di g_i e h_i (cfr. **Sezione 2.2**).

Una delle innovazioni tecniche più rilevanti introdotte da XGBoost è la strategia di apprendimento tramite **istogrammi**. Nei metodi tradizionali, per ogni feature continua è necessario valutare tutti i possibili punti di split, rendendo il processo computazionalmente costoso. XGBoost migliora questo processo raggruppando i valori delle feature in

un numero limitato di intervalli (bin), costruendo così un istogramma. Ogni bin aggrega statistiche come la somma dei gradienti e delle hessiane dei campioni che vi ricadono. Gli split vengono poi valutati solo tra i bordi dei bin adiacenti, riducendo notevolmente il numero di operazioni.
In sintesi, XGBoost combina un approccio di boosting avanzato con una costruzione di alberi ottimizzata tramite istogrammi, regolarizzazione esplicita, parallelismo e gestione efficiente della memoria, risultando una delle scelte più affidabili per applicazioni di machine learning su larga scala. Come vedremo in seguito nel capitolo dei risultati sperimentali infatti, XgBoost si rivela uno degi modelli più performanti.

2.5 LightGBM

LightGBM (Light Gradient Boosting Machine) èun'implementazione avanzata del *Gradient Boosting Decision Tree* (GBDT), sviluppata da Microsoft Research per affrontare le limitazioni di scalabilità e prestazioni degli approcci tradizionali, in particolare quando si lavora con dataset di grandi dimensioni o ad alta dimensionalità. I dataset di riferimento in questa analisi (Uci Credit Risk e German Data) non sono ad alta dimensionalità ma le tecniche boosting conducono alle migliori performance, motivo per il quale è stata adottata pure questa tecnica.

Innovazioni principali

LightGBM introduce due tecniche fondamentali:

- **Gradient-based One-Side Sampling (GOSS)**: durante il training, tutte le istanze con gradienti elevati (dunque più informative) vengono mantenute, mentre solo una porzione casuale delle istanze con gradienti bassi viene selezionata. Per compensare la distorsione nella distribuzione, viene applicato un moltiplicatore correttivo nel calcolo del guadagno informativo.
- **Exclusive Feature Bundling (EFB)**: sfruttando la sparsità dei dati, LightGBM raggruppa le *feature* mutuamente esclusive in un'unica variabile, riducendo così la dimensionalità effettiva del problema e migliorando la velocità del training senza perdita significativa di informazione.

LightGBM adotta inoltre una strategia di crescita degli alberi *leaf-wise*, selezionando la foglia con il massimo guadagno informativo ad ogni iterazione, a differenza del classico approccio *level-wise*. Tale principio conduce ad effettuare lo split non su tutti i nodi, ma solo sulla foglia con più guadagno, tale approccio quindi potrebbe rivelarsi più accurato, tuttavia aumenta il rischio di overfitting specialmente in dataset non molto grandi.

2.5.1 Confronto con GBDT classico e XGBoost

Le informazioni riguardanti la struttura del lightgbm sono state apprese da **"LightGBM: A Highly Efficient Gradient Boosting Decision Tree"** [4].

Caratteristica	GBDT Classico	XGBoost	LightGBM
Algoritmo di split	Pre-sorting	Pre-sorting / Istogramma	Istogramma
Strategia di crescita	Level-wise	Level-wise / Leaf-wise	Leaf-wise
Campionamento delle istanze	Nessuno	Subsampling casuale	GOSS
Riduzione della dimensionalità	Nessuna	Nessuna	EFB (bundling feature esclusive)
Gestione dati sparsi	Limitata	Buona	Ottimizzata (EFB)
Efficienza computazionale	Bassa su big data	Alta	Molto alta
Supporto GPU	Limitato	Sì	Sì
Scalabilità	Limitata	Buona	Eccellente

Tabella 2.1: Confronto tra GBDT, XGBoost e LightGBM

2.5.2 CatBoost

L'ultima implementazione ottenuta dal gradient boosting decision tree che viene proposta è il CatBoost. Catboost è un algoritmo sviluppato da Yandex. Come lightgbm e Xgboost l'implementazione è opensource. La novità di tale algoritmo è di gestire in maniera autonoma ed efficiente le variabili categoriche. Il funzionamento generale è lo stesso del gradient boosting, le innovazioni essenziali sono:

Ordered Target Statistics

La tecnica per trattare autonomamente le variabili categoriche viene chiamata Ordered Target Statistics, viene sostituita ogni categoria della variabile con una statistica. Vengono effettuate una o più permutazioni del dataset, e per ogni riga del database viene calcolata la statistica utilizzando le informazioni delle righe precedenti. In questo modo il Catboost simula un ambiente di apprendimento casuale, e la statistica calcolata sfrutta solo le informazioni delle righe precedenti evitando dunque problemi di data leakage, in particolare la formula utilizzata è la seguente:

Per un esempio x_i con categoria c_i, la statistica ordinata è calcolata come:

$$\mathrm{TS}_i = \frac{\sum_{j<i} \mathbf{1}_{\{c_j = c_i\}} \cdot y_j + a \cdot \mu}{\sum_{j<i} \mathbf{1}_{\{c_j = c_i\}} + a}$$

Dove:

- TS_i: valore assegnato alla feature categorica nella riga i
- c_i: categoria dell'esempio i
- y_j: target dell'esempio j

- $\mathbf{1}_{\{c_j=c_i\}}$: funzione indicatrice (vale 1 se $c_j = c_i$, 0 altrimenti)
- a: parametro di smoothing (iperparametro)
- μ: prior (media globale del target nel dataset)

In questo modo la statistica viene calcolata usando le informazioni delle righe che si trovano prima dell'esempio di cui vogliamo calcolare le variabili.

Ordered Boosting

Nel GBDT, ogni albero viene addestrato sui residui calcolati rispetto alla previsione attuale del modello, che è costruita utilizzando l'intero dataset. Questo introduce una dipendenza : ogni punto contribuirà al calcolo del suo stesso errore, conducendo potenzialmente a fenomeni di overfitting. *CatBoost* affronta questo problema introducendo il meccanismo di **Ordered Boosting**, questo approccio simula un addestramento "temporale", viene sfruttata l'informazione precedente come se i dati arrivassero in sequenza. Più precisamente, viene generata una permutazione casuale delle osservazioni, e i residui sono calcolati in modo incrementale: per ogni osservazione x_i, la previsione è prodotta da un modello addestrato solo sulle osservazioni che precedono x_i nella permutazione. Nel caso della classificazione binaria, i modelli di gradient boosting aggiornano le predizioni in modo iterativo minimizzando una funzione di perdita differenziabile, tipicamente la *log-loss*, definita come:

$$L(y, \hat{p}) = -\left[y \log(\hat{p}) + (1 - y) \log(1 - \hat{p})\right],$$

dove $\hat{p}$ rappresenta la probabilità predetta per la classe positiva. A ogni iterazione, si calcola lo *pseudo-residuo*, ovvero il gradiente della funzione di perdita rispetto alla funzione di decisione del modello. Per la log-loss, lo pseudo-residuo è dato da:

$$r_i = y_i - \hat{p}_i.$$

2.6 K-means Clustering

I modelli discussi in precedenza sono stati utilizzati nella fase di predizione, alcuni come modelli singoli altri nella pipeline dei modelli ensemble. Adesso però viene descritto un modello non supervisionato utilizzato nella fase di feature engineering nell'esperimento dei modelli ensemble. Tale tecnica è stata utilizzata, come si vede nella sezione della metodologia, per la creazione di una nuova feature, utile per i modelli supervisionati nella fase di inferenza. Il **K-means clustering** è un metodo semplice ma potente, il cui obiettivo è quello di dividere in cluster ben separati il dataset. È necessario specificare a priori il numero di cluster K.
L'algoritmo provvederà poi ad assegnare ogni punto del dataset un solo gruppo. Trovare K ottimale è una sfida complessa, tuttavia esistono dei metodi utili alla scelta del miglior numero di cluster. Tornando al princio di funzionamento del K-means, si mostra in

seguito la formulazione matematica. si chiamino $C_1, \ldots, C_K$ gli insiemi che raccolgono gli indici delle osservazioni appartenenti a ciascun gruppo, con le seguenti proprietà:

1. L'unione degli insiemi copre tutte le osservazioni, ovvero: $C_1 \cup C_2 \cup \cdots \cup C_K = \{1, \ldots, n\}$.
2. I cluster sono disgiunti: $C_k \cap C_{k'} = \emptyset$ per ogni $k \neq k'$.

L'obiettivo è individuare una partizione che minimizzi la **variabilità interna ai cluster**. Questo si traduce nel problema di ottimizzazione seguente:

$$\min_{C_1,\ldots,C_K} \sum_{k=1}^{K} W(C_k)$$

dove $W(C_k)$ rappresenta la dispersione all'interno del cluster k.

Come si misura la variazione interna

Una delle scelte più comuni per calcolare $W(C_k)$ consiste nel sommare le distanze euclidee al quadrato tra tutte le coppie di punti nello stesso cluster:

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

In questa formula, $|C_k|$ indica il numero di osservazioni contiene il cluster C_k.

Combinando questa misura con l'obiettivo generale, si ottiene la formulazione generale del problema:

$$\min_{C_1,\ldots,C_K} \left\{ \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 \right\}$$

Si cercano i cluster in cui i punti all'interno dei singoli insiemi siano più vicini tra loro.

Procedura dell'algoritmo

Poiché esplorare tutte le possibili partizioni è computazionalmente impossibile per n grande, si utilizza una strategia iterativa per trovare una soluzione soddisfacente. Ecco come funziona:

1. Si assegna in modo casuale a ciascun dato un'etichetta da 1 a K, fornendo così una partizione iniziale.
2. Si ripete il seguente ciclo fino a quando le etichette non cambiano più:
   (a) Per ciascun gruppo, si calcola il **centroide**, ovvero la media di ogni variabile su tutte le osservazioni del cluster:

$$\bar{x}_{kj} = \frac{1}{|C_k|} \sum_{i \in C_k} x_{ij}$$

(b) Ogni osservazione viene riassegnata al gruppo il cui centroide è più vicino, secondo la distanza euclidea.

Garanzia di convergenza

L'algoritmo garantisce che il valore della funzione obiettivo diminuisca ad ogni iterazione. Questo è dimostrabile con la seguente identità:

$$\frac{1}{|C_k|}\sum_{i,i'\in C_k}\sum_{j=1}^{p}(x_{ij}-x_{i'j})^2 = 2\sum_{i\in C_k}\sum_{j=1}^{p}(x_{ij}-\bar{x}_{kj})^2$$

Da questa espressione si deduce che l'uso della media di ogni feature nel cluster minimizza la somma delle deviazioni quadrate. La riassegnazione, invece, può solo migliorare (o lasciare invariata) la qualità del clustering. Quando le assegnazioni non cambiano più, si è raggiunto un **ottimo locale**. Per valutare il numero di cluster ottimale è stato utilizzato il metodo di silhouette. Tuttavia il numero di cluster scelto non corrisponde al valore con il coefficiente più alto trovato. Seppure non è stato discriminante nell'analisi, tale metodo ha fornito informazioni utili e dunque ne viene riportato brevemente il funzionamento:

2.6.1 Metodo del coefficiente di silhouette.

Il coefficiente di silhouette è una misura utilizzata per valutare la coerenza interna del clustering e supportare la scelta del numero ottimale di cluster k. Per ciascun punto del dataset, si calcola quanto esso sia ben assegnato al proprio cluster rispetto agli altri, confrontando la distanza media tra il punto e gli altri elementi del proprio cluster con quella rispetto al cluster più vicino. Si calcola quindi $a(i)$, ovvero la distanza media tra l'osservazione i e tutte le altre osservazioni appartenenti allo stesso cluster. Formalmente:

$$a(i) = \frac{1}{|C_i|-1}\sum_{\substack{j\in C_i\\ j\neq i}} d(i,j)$$

dove C_i indica il cluster a cui appartiene i e $d(i,j)$ rappresenta la distanza tra le osservazioni i e j.

Successivamente, si calcola $b(i)$, ovvero la distanza media tra l'osservazione i e tutte le osservazioni del cluster più vicino, ovvero quello diverso da C_i con distanza media minima:

$$b(i) = \min_{C\neq C_i}\frac{1}{|C|}\sum_{j\in C} d(i,j)$$

Si definisce quindi il coefficiente di silhouette per l'osservazione i come:

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

Il coefficiente assume valori compresi tra -1 e 1: valori prossimi a 1 indicano un buon raggruppamento, valori prossimi a 0 suggeriscono punti al confine tra due cluster, mentre

valori negativi indicano una possibile assegnazione errata. Se il coefficiente è maggiore di 0, $a(i)$ è minore di$b(i)$, ciò indica che i valori sono in media più vicini al centroide d'appartenenza che a qualsiasi punto del centroide più vicino. infatti Per identificare il valore ottimale di k, si può costruire un grafico che mostra il valore medio del coefficiente di silhouette al variare di k; il numero di cluster ottimale corrisponde a quello per cui tale media risulta massima, segnalando una struttura di clustering ben definita.

2.7 Metodi di calibrazione

Calibrazione Platt (Platt Scaling)

Platt Scaling [5] è una tecnica di calibrazione che trasforma gli *score* grezzi $f(x)$ di un classificatore in probabilità a posteriori mediante una funzione sigmoide parametrica. In particolare, si assume che

$$P(y = 1 \mid f) \;=\; \sigma(A\, f + B) \;=\; \frac{1}{1 + e^{A\, f + B}},$$

dove i parametri A e B sono ottenuti minimizzando la *log-loss* su un dataset di calibrazione indipendente:

$$(A^*, B^*) \;=\; \arg\min_{A,B} \; -\sum_{i=1}^{N} \Big[y_i \ln p_i + (1 - y_i) \ln(1 - p_i) \Big], \quad p_i = \frac{1}{1 + e^{A f_i + B}}.$$

Per ridurre l'overfitting sul piccolo insieme di calibrazione, Platt sostituisce i target binari {0,1} con valori "smussati"

$$y^+ = \frac{N^+ + 1}{N^+ + 2}, \quad y^- = \frac{1}{N^- + 2},$$

dove N^+ e N^- sono il conteggio di esempi positivi e negativi. L'ottimizzazione viene eseguita con algoritmi numerici di discesa del gradiente o Newton Raphson, calcolando le derivate

$$\frac{\partial L}{\partial A} = \sum_{i=1}^{N} (p_i - y_i)\, f_i, \quad \frac{\partial L}{\partial B} = \sum_{i=1}^{N} (p_i - y_i),$$

e aggiornando iterativamente A e B.

Regressione Isotonica

La regressione isotonica [9] è un metodo non parametrico di calibrazione che richiede solo che la funzione m sia monotona non decrescente. La funzione di mapping m è la trasformazione che porta gli score "grezzi" f_i prodotti dal modello alle probabilità calibrate

$$\hat{p}_i \;=\; m(f_i).$$

Dato un insieme di score grezzi $\{f_i\}$ e i corrispondenti target $\{y_i\}$, cerchiamo

$$\hat{m} = \arg \min_{m:\ m \text{ monotona}} \sum_{i=1}^{N} \big(y_i - m(f_i)\big)^2.$$

Per risolvere questo problema si usa il *Pool-Adjacent-Violators Algorithm (PAV)*:

1. Ordina le coppie (f_i, y_i) in base a f_i.
2. Inizia con ogni coppia come blocco separato.
3. Se due blocchi adiacenti violano la monotonicità (media successiva $<$ media corrente), si uniscono e si ricalcola la media.
4. Le iterazioni si ripetono finché tutti i blocchi sono non decrescenti

Per evitare bias, la stima di m viene fatta su un set di validazione indipendente rispetto al training originale del modello .

Chapter 3

Esplorazione dei dataset e analisi della Metodologia

In questo capitolo viene mostrata la struttura dei database utilizzati nel lavoro di tesi, e la nuova metodologia introdotta, descrivendo i passi fondamentali seguiti per la costruzione dei metodi di ensemble. I dataset utilizzati sono due, il primo è *"UCI Credit Risk"* disponibile su Default of Credit Card Clients Dataset, il secondo invece è *"German Credit Data"*, un dataset ampiamente utilizzato nel contesto della valutazione del rischio di credito. Il lavoro di tesi si distingue in due fasi principali:

- Fase di riproduzione delle tecniche di Machine Learning mostrate in *Machine Learning techniques in joint default assessment* [4] su i due dataset.
- Costruzione di una nuova metodologia, facendo riferimento alla struttura mostrata in *"Integration of unsupervised and supervised machine learning algorithms for credit risk assessment"* [8] al fine di migliorare le performance predittive dei modelli.

Nella prima fase della tesi come già discusso vengono riprodotte le tecniche utilizzate nel paper di riferimento *"Machine Learning techniques in joint default assessment"* [4], al fine di calcolare la probabilità di default. Dunque vengono costruiti i modelli individuali, valutate le performance dei modelli ed infine stimate la probabilità di default e la correlazione tra i creditori, con il metodo dei momenti. I risultati ottenuti in questa prima fase con l' "UCI Credit Risk" (discussi nel capitolo 4 della tesi), sono uguali a quelli ottenuti in *Machine Learning techniques in joint default assessment* [4], essendo lo stesso database utilizzato. Per il German Credit Data otteniamo dei nuovi risultati, visibili anch'essi nel capitolo 4.

Nel corrente capitolo non viene riportata la metodologia utilizzata nella prima fase della tesi perché già discussa nel capitolo 1. L'obiettivo della sezione della metodologia 3.3 è dunque mostrare la nuova procedura costruita, il motivo per cui è stata implementata questa nuova *pipeline* è quello di migliorare le performance ottenute dai modelli singoli di classificazione (fase 1), al momento della discriminazione tra le due classi (default e non default). Tra le metriche di valutazione, quella su cui si vorrebbe ottenere il più grande

miglioramento è la *recall* della classe dei default. In seguito sarà inoltre motivata la scelta di questa metrica come principale oggetto di analisi confrontandola con le altre.

La struttura del capitolo dunque prevede, in primo luogo, l'analisi descrittiva dei dataset utilizzati, seguita dallo studio della metodologia proposta.

3.1 UCI CREDIT RISK

Come anticipato, in questa sezione si discuterà della struttura dell' UCI CREDIT RISK, analizzando le variabili che lo compongono e le maggiori criticità che lo carraterizzano.

Questo dataset contiene informazioni sui pagamenti, fattori demografici, dati di credito, storico dei pagamenti e rendiconti delle fatture dei clienti di carte di credito a Taiwan, dal mese di aprile del 2005 a settembre dello stesso anno. È composto da 30.000 record e 25 variabili, quantitative, categoriche e ordinali. Non presenta valori mancanti, e non è stato inoltre ritenuto necessario eliminare osservazioni, poiché non si riscontrano valori anomali. Di seguito viene mostrata una tabella riassuntiva, in cui vengono descritte le features presenti:

Variabile	Descrizione	Tipo
ID	Identificativo univoco del cliente (Variabile che sarà esclusa nella fase di preprocessing)	Categorica
LIMIT_BAL	Limite massimo di credito concesso	Quantitativa
SEX	Sesso (1 = maschio, 2 = femmina)	Categorica
EDUCATION	Livello di istruzione (1 = laurea, 2 = master, 3 = liceo, ecc.)	Categorica
MARRIAGE	Stato civile (1 = sposato, 2 = single, 3 = altri)	Categoriale
AGE	Età del cliente	Quantitativa
PAY_1	Stato del pagamento a settembre 2005 (-1 = pagato in tempo, 1 = 1 mese di ritardo, ecc.)	Ordinale
PAY_2	Stato del pagamento ad agosto 2005	Ordinale
PAY_3	Stato del pagamento a luglio 2005	Ordinale
PAY_4	Stato del pagamento a giugno 2005	Ordinale
PAY_5	Stato del pagamento a maggio 2005	Ordinale
PAY_6	Stato del pagamento ad aprile 2005	Ordinale
BILL_AMT1	Importo della fattura a settembre 2005	Quantitativa
BILL_AMT2	Importo della fattura ad agosto 2005	Quantitativa
BILL_AMT3	Importo della fattura a luglio 2005	Quantitativa
BILL_AMT4	Importo della fattura a giugno 2005	Quantitativa
BILL_AMT5	Importo della fattura a maggio 2005	Quantitativa
BILL_AMT6	Importo della fattura ad aprile 2005	Quantitativa
PAY_AMT1	Importo pagato a settembre 2005	Quantitativa
PAY_AMT2	Importo pagato ad agosto 2005	Quantitativa
PAY_AMT3	Importo pagato a luglio 2005	Quantitativa
PAY_AMT4	Importo pagato a giugno 2005	Quantitativa
PAY_AMT5	Importo pagato a maggio 2005	Quantitativa
PAY_AMT6	Importo pagato ad aprile 2005	Quantitativa
default. payment. next. month	1 = il cliente è andato in default il mese successivo, 0 = no (Variabile target)	Categorica

Tabella 3.1: Descrizione delle variabili del dataset UCI Credit Card Default

Come si evince dalla tabella 3.1 , le variabili presenti risultano molto differenti tra loro. Si inizia l'analisi esplorativa mostrando la distribuzione dell'età dei creditori, così da osservare eventuali differenze tra la popolazione dei default e dei non default.

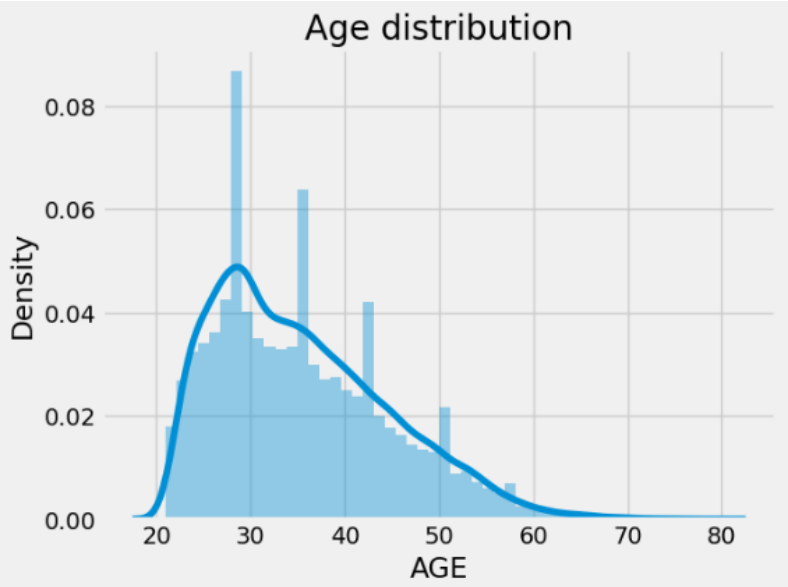

Figura 3.1: Distribuzione dell'età per l'UCI CREDIT RISK

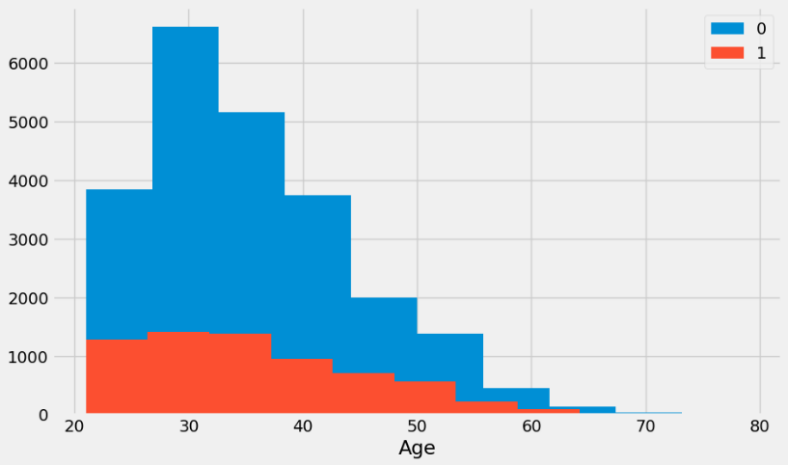

Figura 3.2: Distribuzione dell'età discriminando per default e non default nell'UCI Credit RISK

Come si evince dalla 3.2, la maggior parte dei default si osservano tra i 20 e i 45 anni. In generale però sia le osservazioni della classe dei creditori insolventi (classe 1), che quella dei buoni creditori (classe 0), sono più numerose in questo range, tuttavia le numeriche tra le due classi sono molto diverse. I modelli infatti avranno difficoltà a fare inferenza su i default, assegnando con più semplicità la classe dei non default ai campioni di test. Lo sbilanciamento delle classi non è un problema solo per il database in questione, ma in generale nel contesto del rischio di credito. Tornando al caso studio in questione valutiamo la numerosità per ogni classe:

- Classe dei non default: 23.364
- Classe dei default: 6.636

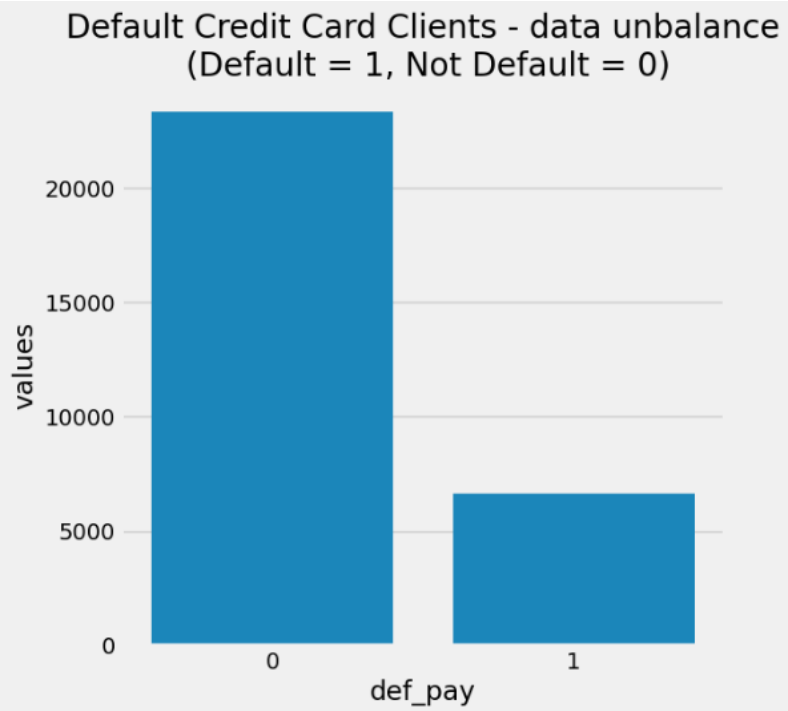

Figura 3.3: Numero di clienti per le due classi

Come si diceva tale differenza nelle numeriche conduce ad una scarsa capacità previsionale dei modelli sulla classe 1. Tale problematica, come vedremo in seguito sarà la principale causa delle scarse performance ottenute per i modelli individuali sulla classe dei default. Si continua l'analisi esplorativa analizzando un'altra variabile rilevante nel rischio di credito. La variabile in questione è "limit bal", essa rappresenta il limite massimo assegnato ad ogni cliente. L'unità di misura è il nuovo dollaro di Taiwan. Un limite più alto suggerisce maggiore fiducia da parte della banca, mentre un limite più basso rappresenta più sfiducia verso il creditore. Osserviamo la distribuzione:

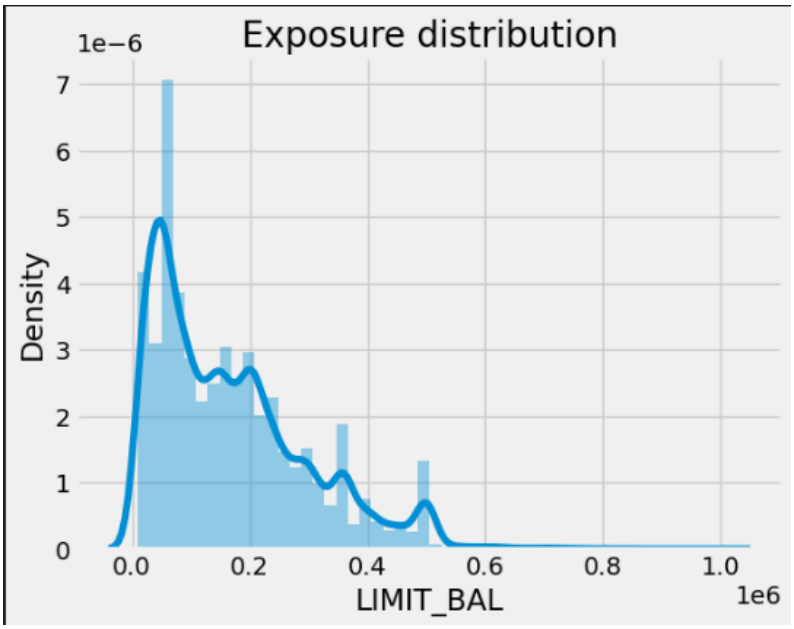

Figura 3.4: Distribuzione del limite massimo di credito

La media corrisponde a 167.484 NT, corrisponderebbe (con il tasso di cambio del 2005) a circa 4300 euro. Vediamo adesso come sono distribuiti i limiti rispetto alle due classi, utilizzando i boxplot. Effettivamente come ci si aspettava, dalla figura 3.5 si vede

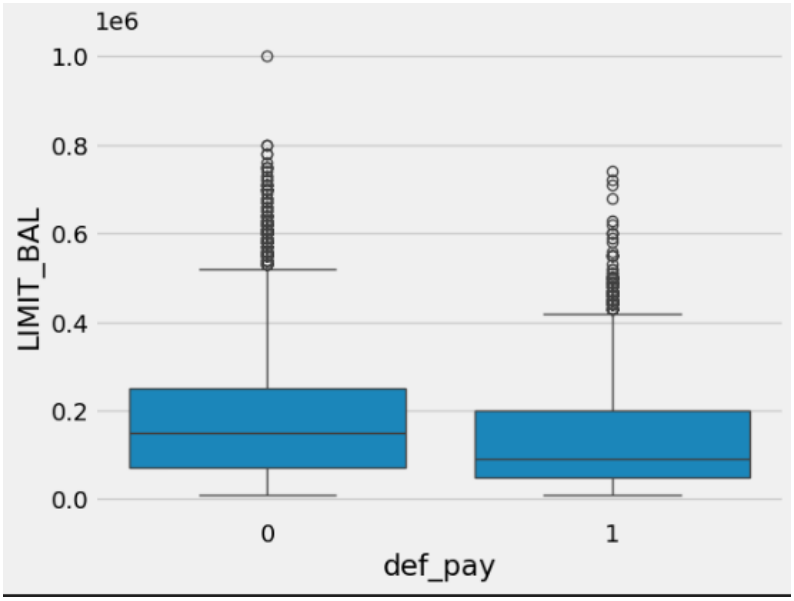

Figura 3.5: Box plot della variabile "Limit Bal" su entrambe le classi

che la mediana dei default (secondo quartile) risulta più bassa rispetto ai non default. I boxplot sono uno strumento fondamentale per l'individuazione degli outliers, i valori

fuori dai "whiskers"(estensione massima dei dati), calcolati come 1.5 per l'interquartile range (Q3-Q1) sono infatti dei valori critici, che potrebbero essere scartati dell'analisi. Tuttavia i valori sembrano tutti ammissibili, considerando infatti che il valore di limite massimo corrisonde a 1.000.000 Nt circa 25.640 euro. Si è deciso quindi di tenere tutti i record. Dopo aver analizzato le variabili quantitative è utile osservare il comportamento dei creditori rispetto alle variabili "PAY". Come è riportato sopra, tali variabili sono ordinali e rappresentano lo stato del pagamento:

- -1 Pagato in tempo
- 1 Pagato con un mese di ritardo
- 2 Pagato con 2 mesi di ritardo
- 3 Pagato con 3 mesi di ritardo
- 4 Pagato con 4 mesi di ritardo
- 5 Pagato con 5 mesi di ritardo
- 6 Pagato con 6 mesi di ritardo
- 7 Pagato con 7 mesi di ritardo
- 8 Pagato con 8 mesi di ritardo
- 9 Pagato con 9 mesi di ritardo o più

Il database contiene le informazioni inerenti a 6 mesi di analisi, quindi si hanno 6 variabili di questo tipo. Nel contesto del rischio di credito il comportamento del cliente è una variabile fondamentale. Generalmente infatti, pagamenti in ritardo indicano un comportamento instabile da parte del debitore che potrebbe condurre al default. È interessante dunque osservare con l'ausilio dei box plot l'impatto di tali variabili sul default dei clienti.

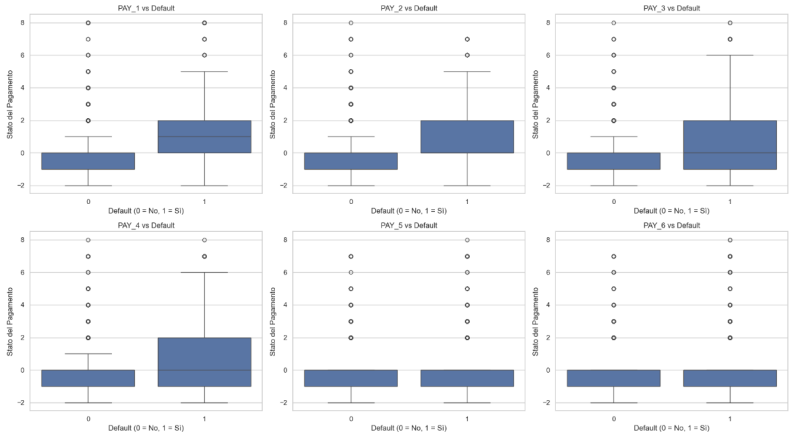

Figura 3.6: Box plot dello stato del pagamento nell'UCI Credit Risk

Si nota che le due distribuzioni sono ben separate soprattutto per i primi 4 mesi. Come si nota in figura 3.6, la prima variabile, quindi lo stato del pagamento al primo mese, risulta la feature più discriminante. I clienti che andranno in default infatti ritardano i pagamenti, mentre i non default risultano sempre puntuali. Nei mesi finali invece, anche i default si rivelano precisi. Quest'ultima osservazione rivela maggiore rilevanza delle prime 4 variabili rispetto alle ultime due in termini di separabilità dei dati. Dopo aver notato che lo stato del pagamento risulta una delle variabili maggiormente discriminanti, si vuole analizzare l'importo associato ad ogni pagamento, il nome della variabile in questione è "PAY AMT" ed anche in questo caso distinguiamo 6 variabili, una per ogni mese analizzato. Come ci si aspettava gli importi dei default sono molto più bassi rispetto

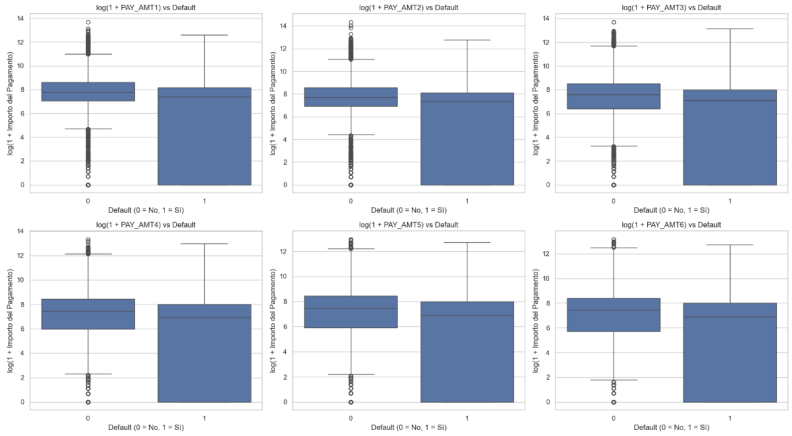

Figura 3.7: Box plot dell'importo del pagamento per ogni mese

a quelli della classe dei non default. Mostrando anche in questo caso la capacità di tali variabili di separare i dati. Le variabili categoriche invece non sono molto utili alla distinzione delle due classi. Inoltre nella replica dell'esperimento effetuato in in *Machine Learning techniques in joint default assessment* [4], (fase 1 del lavoro di tesi), vengono utilizzate solo le variabile numeriche. A supporto di tale osservazione come ultimo step

per la data exploration si è voluta calcolare la feauture importance per ogni variabile.

Feature importance

La feature importance rappresenta una misura dell'influenza che ciascuna variabile esercita sulla capacità predittiva di un modello. In altre parole, indica quanto una variabile contribuisce alla previsione dell'output.

Il calcolo della feature importance è stato ottenuto tramite la Random Forest. Nel modello Random Forest, l'*importanza delle variabili* (*feature importance*) viene calcolata analizzando quanto ciascuna variabile contribuisce alla riduzione dell'impurità nei nodi decisionali degli alberi. Nei problemi di classificazione, l'impurità è solitamente misurata tramite l'indice di Gini.

Ogni volta che una feature viene utilizzata per effettuare una suddivisione (*split*) in un nodo, si calcola la riduzione di impurità tra il nodo padre e i figli. Questa riduzione viene accumulata per ciascuna variabile su tutti i nodi in cui è stata utilizzata e su tutti gli alberi della foresta. Alla fine, i valori vengono normalizzati per ottenere un punteggio di importanza relativa.

Formalmente, l'importanza di una feature X_j è calcolata come:

$$\text{Importance}(X_j) = \frac{\sum_{t=1}^{T} \sum_{n \in \text{splits}(X_j)} \Delta I_{t,n}}{\sum_{t=1}^{T} \sum_{n \in \text{all splits}} \Delta I_{t,n}}$$

dove:

- T è il numero totale di alberi nella foresta;
- $\Delta I_{t,n}$ è la riduzione di impurità nel nodo n dell'albero t;
- la somma al numeratore è limitata ai nodi in cui viene usata la variabile X_j.

Per l'Uci Credit Risk vengono mostrati gli istogramma per mostrare l'importanza delle variabili:

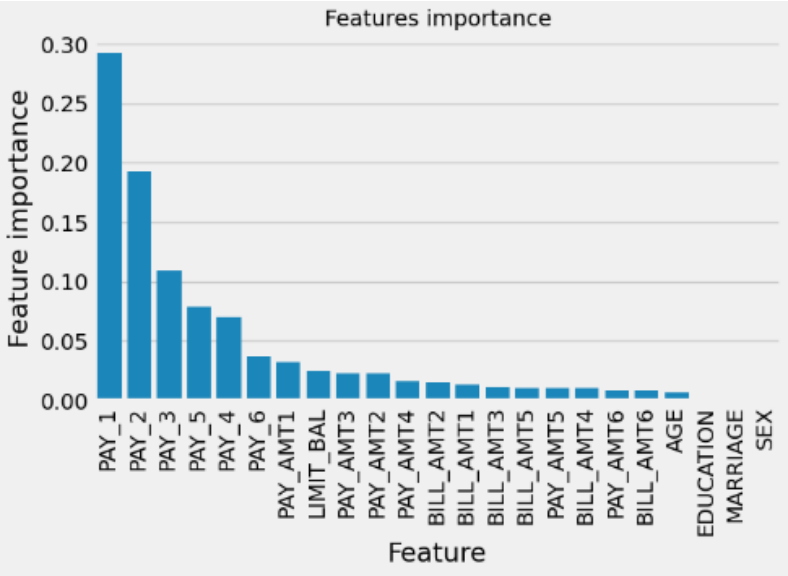

Figura 3.8: Feature Importance calcolata con la Random Forest per l'UCI Credit Risk

Come si vedeva dai boxplot precdentemente mostrati le variabili maggiormente impattanti effettivamente risultano:

- Lo stato del pagamento mensile
- Il limite del credito asssegnabile
- L'importo del pagamento mensile

Si procederà di seguito all'analisi del german dataset allo stesso modo.

3.2 German Credit Dataset

IL dataset adesso presentato é *Statlog German Credit Data*, disponibile su Kaggle. Questo dataset è ampiamente utilizzato nell'analisi del rischio di credito e contiene informazioni dettagliate su 1000 clienti, ciascuno caratterizzato da venti variabili. L'obiettivo è quello di classificare i clienti in "default" o "non-default" dipendentemente dalle loro caratteristiche. Nel dataset analizzato non sono presenti missing values, si è inoltre deciso di non escludere alcuna osservazione non notando valori molto diversi rispetto alle loro distribuzioni di riferimento. Le caratteristiche incluse nel dataset sono descritte di seguito:

Variabile	Descrizione	Tipo
Account status	Stato del conto corrente del cliente	Categorica
Duration	Durata del prestito in mesi	Quantitativa
Credit history	Cronologia creditizia del cliente (es. prestiti precedenti, debiti)	Categorica
Purpose	Scopo del prestito (es. acquisto di auto, elettrodomestici, ecc.)	Categorica
Credit amount	Importo del prestito richiesto	Quantitativa
Savings account/bonds	Stato dei risparmi e obbligazioni del cliente	Categorica
Employment status	Durata dell'impiego attuale del cliente	Ordinale
Installment rate	Tasso di rata rispetto al reddito disponibile	Quantitativa
Personal status and sex	Stato civile e sesso del cliente	Categorica
Other debtors/guarantors	Presenza di altri debitori o garanti	Categorica
Residence since	Anni di residenza nell'attuale abitazione	Quantitativa
Property	Tipo di proprietà posseduta (es. immobili, risparmi, ecc.)	Categorica
Age	Età del cliente	Quantitativa
Other installment plans	Presenza di altri piani di pagamento rateale	Categorica
Housing	Tipo di abitazione (es. affitto, di proprietà, ecc.)	Categorica
Number of existing credits	Numero di prestiti esistenti presso altre banche	Quantitativa
Job	Tipo di lavoro del cliente	Categorica
Number of people liable	Numero di persone a carico	Quantitativa
Telephone	Disponibilità di un telefono fisso	Categorica
Foreign worker	Indica se il cliente è un lavoratore straniero	Categorica

Tabella 3.2: Descrizione delle variabili del dataset German Credit

Queste variabili forniscono un quadro dettagliato delle caratteristiche finanziarie e personali dei clienti, consentendo un'analisi accurata del rischio di credito. La presenza di variabili sia numeriche che categoriche rende questo dataset ideale per l'applicazione di tecniche di *machine learning* e *data mining*. Per una migliore comprensione della struttura del dataset, si presenta di seguito un'immagine rappresentativa che mostra un'anteprima della struttura. La maggiore criticità del database analizzato (come nell'UCI CREDIT RISK) è lo sbilanciamento delle classi. I creditori sono distribuiti tra le due classi in questo modo:

- 700 Non default
- 300 Default

Sono già state analizzate nel caso dell'UCI Credit Risk le problematiche per i modelli di machine learning causate dallo sbilanciamento. Per ogni variabile continua nel dataset,

	id	duration	cred_hist	Purpose	Cred_am	st_sav	pr_empl	inst_rate_per	personal_status	oth_deb	res_years	Property	Age	other_installment	Housing	Numb_of_people	Job	Numb_people_maintenance	Telephone	for_worker	def_pay
0	1	6	1	6	1169	4	3	4	3	2	4	3	67	1	1	2	1	1	1	1	0
1	2	48	3	6	5951	2	0	2	0	2	2	3	22	1	1	1	1	1	0	1	1
2	3	12	1	2	2096	2	1	2	3	2	3	3	49	1	1	1	3	2	0	1	0
3	4	42	3	3	7882	2	1	2	3	1	4	1	45	1	0	1	1	2	0	1	0
4	5	24	2	4	4870	2	0	3	3	2	4	2	53	1	0	2	1	2	0	1	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	996	12	3	3	1736	2	1	3	0	2	4	3	31	1	1	1	3	1	0	1	0
996	997	30	3	9	3857	2	0	4	1	2	4	1	40	1	1	1	0	1	1	1	0
997	998	12	3	6	804	2	3	4	3	2	4	0	38	1	1	1	1	1	0	1	0
998	999	45	3	6	1845	2	0	4	3	2	4	2	23	1	0	1	1	1	1	1	1
999	1000	45	1	9	4576	0	4	3	3	2	4	0	27	1	1	1	1	1	0	1	0

1000 rows × 21 columns

Figura 3.9: German Credit Data.

sono stati creati dei grafici per esaminare la distribuzione dei valori. Questi plot hanno permesso di visualizzare la forma della distribuzione, identificando eventuali asimmetrie, valori estremi o tendenze particolari. Inizieremo mostrando come è distribuita l 'età tra i creditori. Questa variabile riveste un ruolo significativo nella previsione del default, a differenza di quanto visto nel dataset analizzato in precedenza: un creditore giovane, infatti, tende ad avere in generale una maggiore probabilità di insolvenza.

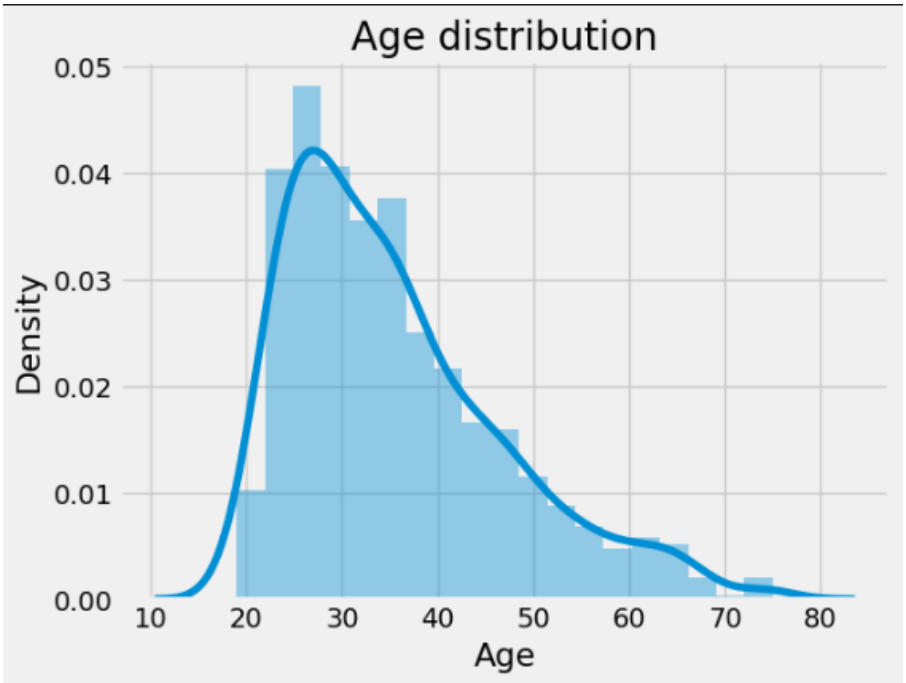

Figura 3.10: Distribuzione del'età del German Credit Data

Come viene mostrato in figura 3.10 la maggior parte dei creditori si aggira attorno ai 30 anni. Questa caratteristica renderà difficile la separabilità dei dati, poiché avremo la maggior parte dei creditori in un età critica dal punto di vista del rischio di default, ma una netta supremazia al livello numerico di creditori non-default in tutte le fasce d'età compresa quella giovanile. A questo punto è interessante osservare la struttura della

distribuzione dei default e dei non default nello stesso grafico, così da poter visualizzare eventuali differenze o analogie: Le distribuzioni dei dati sembrano essere molto simili, con

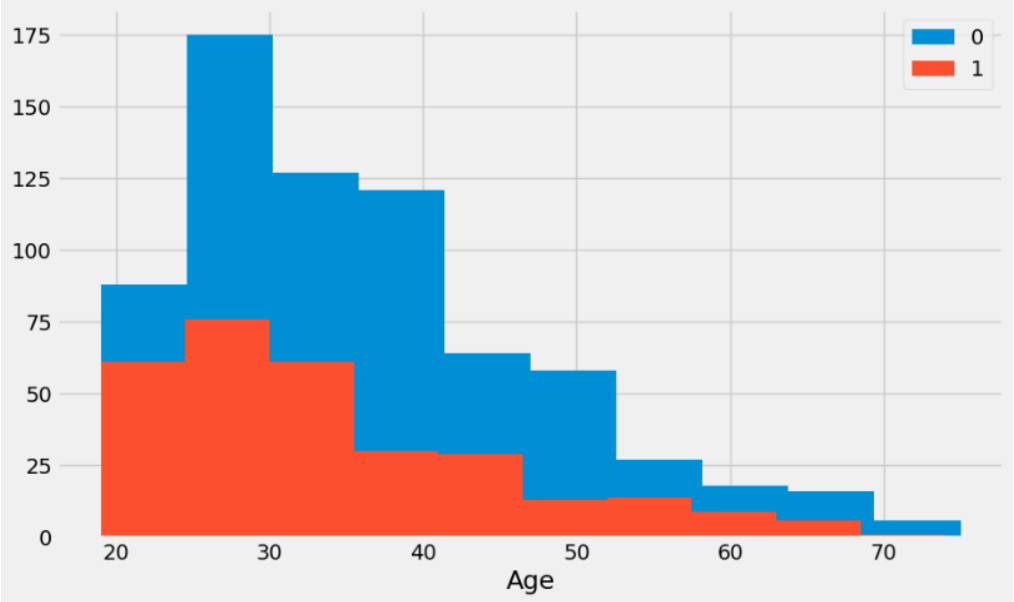

Figura 3.11: Distribuzione dell'età per i default e i non default nel German Credit Data.

una netta maggioranza dei non default rispetto ai default. Come visto anche nell'analisi dell' UCI CREDIT RISK, anche nel caso studio corrente si nota come lo sbilanciamento tra le classi sia l'aspetto maggiormente critico. Poiché anche se la maggior parte dei default stanno tra i 20 e i 40 anni, la maggioranza in generale dei non default indurrà i modelli a favorire la classe 0 rispetto alla classe 1. Non si riesce ancora ad osservare la capacità di tale variabile nella sperazione tra le classi, allora vengono utilizzati i boxplot per migliorare la visualizzazione. Questo tipo di grafico rende più immediata l'osservazione di eventuali scostamenti o tendenze.

Figura 3.12: Box plot dell' età nel German Credit Dataset per entrambe le classi

Dalla figura 3.12 si vede come la popolazione dei default si concentra in un range d'età leggermente più basso rispetto alla popolazione dei non default, riflettendo una maggiore dose di rischio per i creditori più giovani. Si continua l'analisi, con una variabile che senza dubbio è tra le più importanti, sia nel contesto specifico (si vedrà con la Feature Importance in seguito), che nel contesto generale del rischio di credito. La variabile in questione è l'entità dell'importo richiesto (Credit Amount), essa rappresenta un fattore rilevante nella valutazione del rischio di credito. Da un punto di vista economico è noto che richieste di prestiti molto elevate possono essere associate a una maggiore difficoltà nel rimborso. Tuttavia, è importante sottolineare che un importo elevato non implica necessariamente un profilo rischioso: potrebbe anche indicare un soggetto economicamente solido, che richiede un credito significativo per finanziare un progetto strutturato o un investimento pianificato. Per visualizzare nel concreto le informazioni derivate dal dataset analizzato vengono mostrati i boxplot del credit amount rispetto alla variabile target.

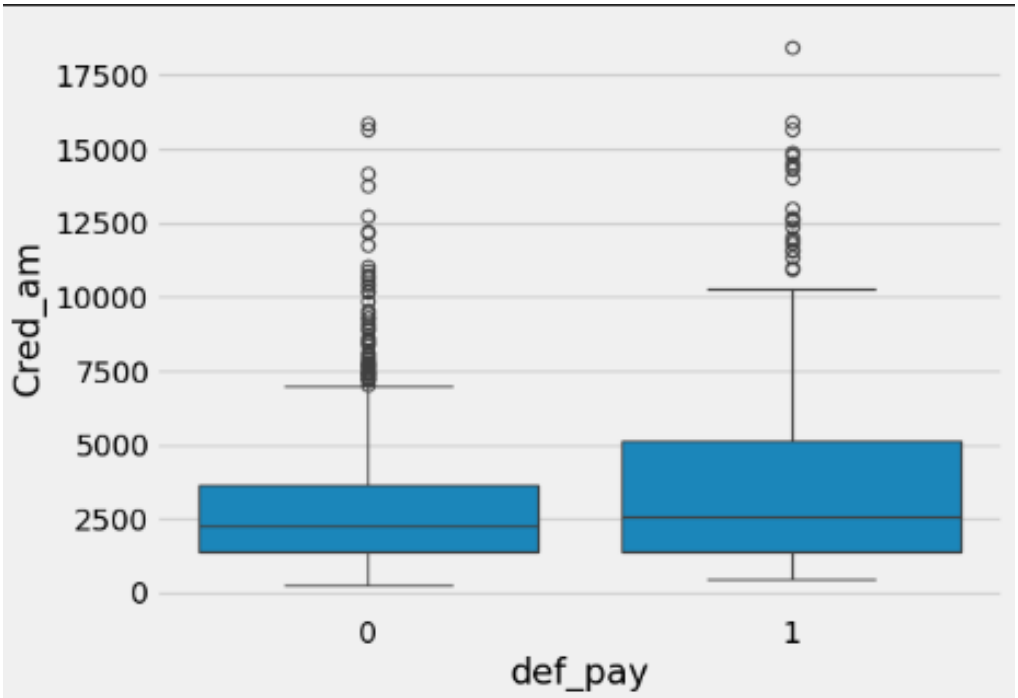

Figura 3.13: Box plot dell'importo di credito per le due classi nel German Credit Dataset.

Dalla figura 3.13 si osserva come la distribuzione del "credit amount" differisca tra i default e i non default. In generale, la popolazione dei default richiede infatti un importo di credito maggiore rispetto ai non default, rappresentando di conseguenza un profilo più rischioso.

La variabile duration, che rappresenta la durata del prestito in mesi, è una caratteristica fondamentale nell'analisi del rischio di credito, come sarà confermato in seguito dal calcolo della feature importance. Di seguito come fatto per l'età verrà mostrata la distribuzione di tale variabile.

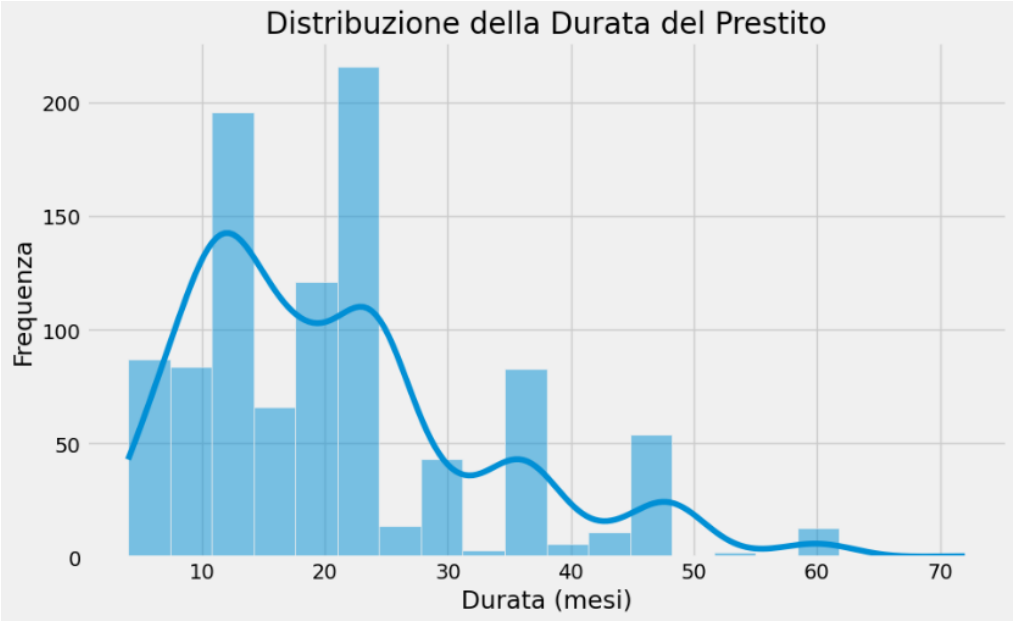

Figura 3.14: Distribuzione della Duration nel German Credit Dataset

La maggior parte dei clienti riceve un credito di circa 20 mesi. Si vuole capire se un debito a lungo termine induce ad un rischio di default più elevato, rispetto ad uno a breve termine. Una durata più lunga infatti implica un'esposizione prolungata al rischio per l'istituto finanziario, in quanto aumenta il tempo di attività e dunque la probabilità che eventi imprevisti possano compromettere la capacità del cliente di rimborsare il debito. Allo stesso tempo, i prestiti a lungo termine generano un maggiore accumulo di interessi, aumentando il costo complessivo per il cliente e il profitto potenziale per la banca. Tuttavia, proprio per questo, i clienti con una situazione finanziaria meno solida tendono a richiedere durate più estese. Nel German credit dataset si vede come i debitori con durata del prestito maggiore sono più predisposti al default, di seguito dunque mostriamo ancora una volta i box plot per osservare le differenze tra le due popolazioni.

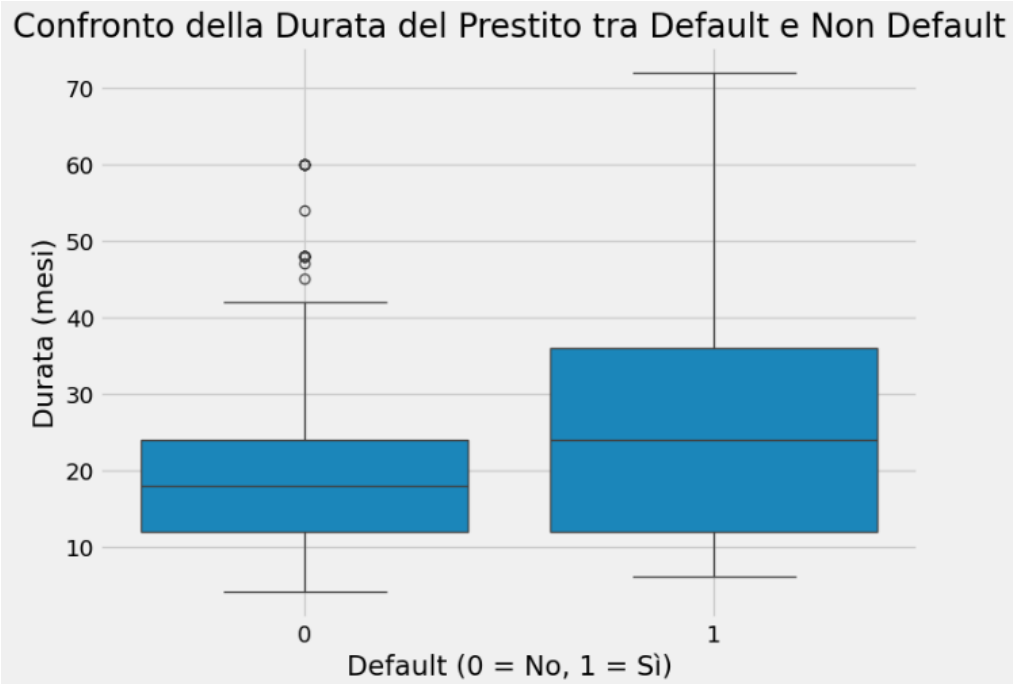

Figura 3.15: Box plot della durata del prestito per il German Credit Data

Come si vede dalla figura 3.15 le due distribuzioni sono molto diverse, si nota dunque che la popolazione dei default tende a richiedere crediti dalla durata maggiore rispetto ai non default. Tale differenza delle due distribuzioni rende la duration la seconda variabile più importante tra le caratteristiche analizzate(dopo L'importo di Credito visto in precedemza)

Dopo aver analizzato le variabili continue, risultate nel calcolo della feature importance le più rilevanti, è interessante concludere l'analisi esplorativa mostrando una variabile categorica, anch'essa molto informativa per l'inferenza dei modelli successivamente addestrati. Tale caratteristica è la "Credit History" dei clienti. Tale variabile rappresenta il passato finanziario dei debitori, le categorie della feature sono le seguenti:

- **existing paid** Crediti esistenti pagati regolarmente fino ad oggi.
- **critical** Conto critico o problemi gravi nello storico creditizio.
- **other existing credit** – Altri crediti esistenti (non specificati), attivi.
- **delayed previously** Ritardi nei pagamenti di prestiti precedenti.
- **all paid** Tutti i crediti precedenti sono stati pagati regolarmente.
- **no credits | all paid** Nessun credito precedente oppure tutti i crediti precedenti sono stati saldati.

Vengono mostrati degli istogrammi per ogni categoria così da visualizzare come sono distribuite le due popolazioni per ogni categoria. Circa la metà dei default (300 in totale)

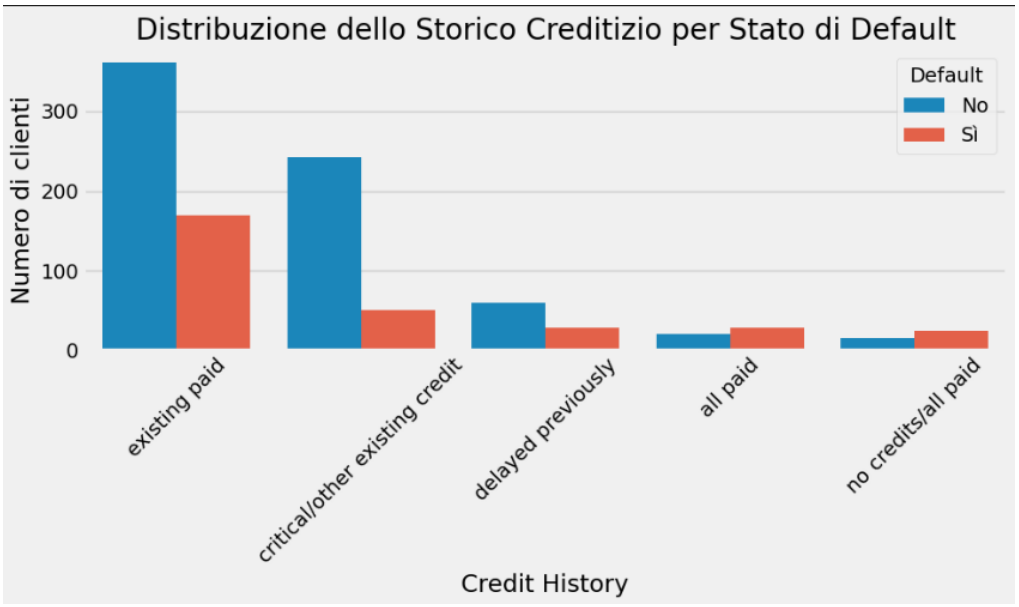

Figura 3.16: Istogrammi che evidenziano il numero di clienti per entrambe le classi, e tutte le categorie della Credit History.

in passato ha richiesto dei crediti. Tale caratteristica dunque aggiunge un importante contenuto informativo per l'analisi.

Come già visto nell'analisi esplorativa dell'UCI CREDIT RISK, per ogni variabile presente nel dataset è stata calcolata la feature importance. Le analisi riportate in precedenza si riferiscono alle 4 variabili più importanti. Di seguito dunque vengono mostrati degli istogrammi che indicano l'importanza per ogni feature:

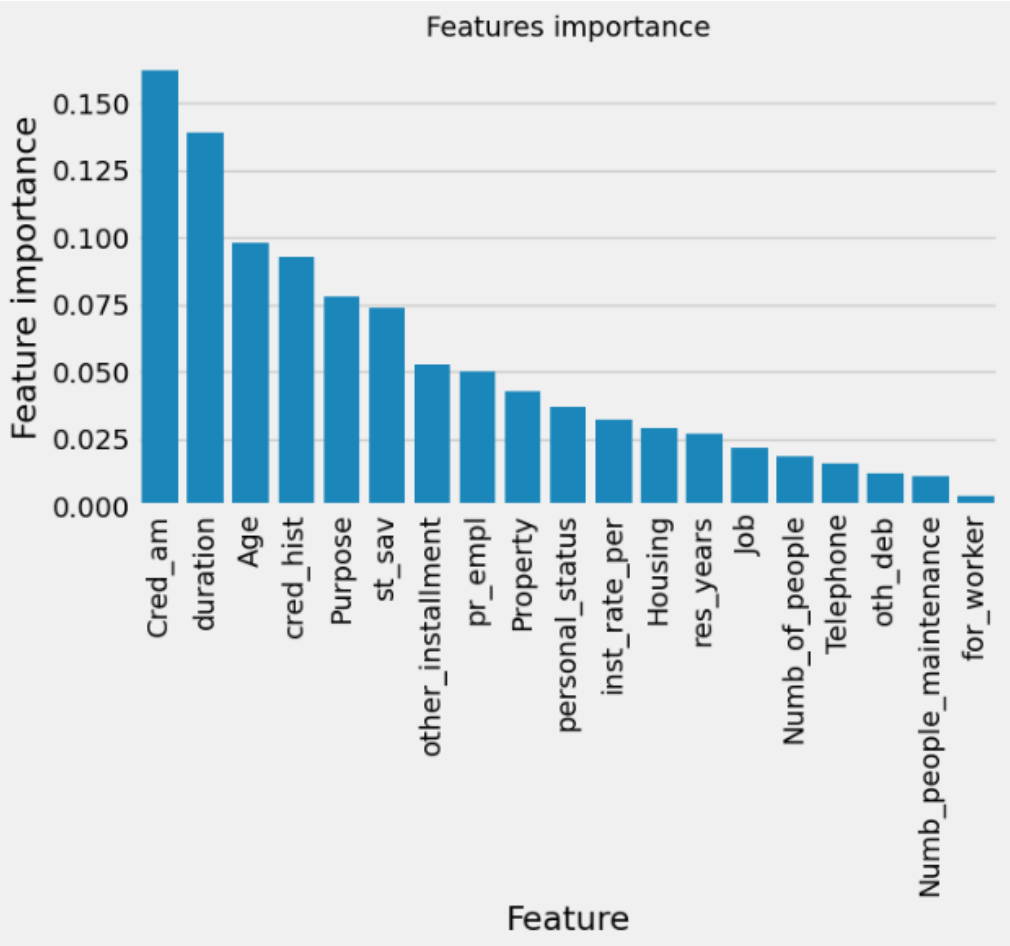

Figura 3.17: Feature Importance per il German Credit Data

Si procede l'analisi con la presentazione della nuova metodologia utilizzata, al fine di migliorare la recall della classe 1 ottenuta dai modelli singoli.

3.3 Metodologia

3.3.1 Modelli ensemble

Nel campo del Machine Learning per il rischio di credito, la sfida maggiore è riuscire a prevedere correttamente gli individui che andranno in default rispetto ai non default. Le classi su cui si deve fare inferenza sono 2 : 1 (default) e 0 (non default), con l'introduzione della seguente struttura, basata sui modelli ensemble l'obiettivo sarebbe migliorare le metriche riferite alla classe dei default e stimare correttamente la probabilità marginale di default. La difficoltà di ottenere performance elevate per la classe 1(default) deriva dal netto sbilanciamento del dataset come ampiamente visto nei capitoli precedenti, e dalla presenza di dati difficilmente separabili. Proprio per questo si è voluta esplorare un' altra strada rispetto all'utilizzo dei soli modelli individuali implementati nella fase complilativa

dell'analisi. Di seguito verrà discussa l'idea e i metodi utilizzati per provare ad affontare questo problema. Si è partiti dall'analisi dell'articolo *"Integration of unsupervised and supervised machine learning algorithms for credit risk assessment"* [8], esaminato nel Capitolo 1 della tesi. In questo studio viene proposta una pipeline in cui diversi modelli supervisionati costituiscono il primo livello di una struttura, in cui al termine vi è un modello non supervisionato (SOM) che utilizzando gli output dei modelli posti al livello inferiore individua due cluster. L'obiettivo di tale approccio è quello di combinare più modelli all'interno dello stesso flusso di lavoro, al fine di migliorare le prestazioni rispetto all'utilizzo di un singolo modello.

Nel presente lavoro di tesi si adotta un'impostazione molto simile: si utilizza un modello non supervisionato all'inizio della pipeline, con l'obiettivo di aumentare la dimensionalità informativa del dataset. Successivamente vengono costruiti diversi modelli supervisionati i cui output, a loro volta, vengono forniti in input a un ulteriore modello supervisionato finale. In generale vengono testati diversi modelli finali così da valutare quale combinazione tra quelle testate fosse la più performante.

Tra le metriche della classe dei default, si è discusso dell'importanza della recall.

Un modello con una buona recall in questo ambito condurrebbe ad una decisione più cauta da parte di chi concede il credito. La recall della classe 1 infatti è il rapporto tra i true positive (quindi i default predetti correttamente dal modello) e la somma dei default effettivi. Avere quindi una recall bassa indica che il modello ha assegnato molti clienti in default alla classe dei non default. Nel contesto del rischio di credito il più grande problema è proprio questo. È sicuramente preferibile assegnare un non-default alla classe dei default, quindi non concedere un prestito ad un cliente che restituirà con successo l'importo, che perdere l'intera somma considerando un profilo di default come non default.

Un ulteriore obiettivo dell'analisi resta stimare in modo accurato la probabilità di default per ogni individuo, al fine di calcolare in lavori futuri indicatori di rischio finanziario come il Value at Risk (VaR) ed Expected Shortfall utilizzando ad esempio le tecniche mostrate nella seconda parte dell'analisi presente in in *"Machine Learning in Joint Default Assessment"* [4]. L'esperimento anche in questo caso è stato svolto con entrambi i dataset, per valutare effettivamente la validità della struttura proposta e il comportamento dell'algoritmo con dataset come visto diversi per molte caratteristiche.

3.3.2 Struttura del metodo

L'approccio utilizzato per affrontare il problema è il medesimo per entrambi i dataset, con qualche differenza nella fase di feature engineering, e nelle tecniche di oversampling utilizzate. Si procede quindi brevemente mostrando i passi effetuati per poi descriverli in maniera più approfondita:

- **Preprocessing iniziale:** scaling temporaneo delle feature originali per applicare il clustering solo ai clienti in default.

- **Feature engineering:** Si aggiungono delle feature per arricchire il contenuto informativo del dataset. (Solo nel caso dell'UCI Credit Risk)

- **Clustering sui clienti in default:** applicazione di `KMeans` con $k = 2$ per identificare sottogruppi di default e calcolo della distanza di ciascun punto dal proprio centroide.
- **Arricchimento delle feature:** aggiunta al dataset delle nuove feature `DEFAULT_CLUSTER` e `CLUSTER_DISTANCE` sia per il training che per il test, così da associare ad ogni creditore il cluster di default più simile.
- **Data split:** suddivisione del dataset in training e validation set.
- **Feature scaling:** scaling dell'intero dataset con `MinMaxScaler`, comprese le nuove feature ingegnerizzate.
- **Bilanciamento del training set:** utilizzo della tecnica `SMOTETomek` per riequilibrare le classi del training set.
- **Tuning degli Iperparametri :** ottimizzazione dei parametri per i modelli supervisionati: (`XGBoost`, `CatBoost`, `LightGBM`, `BalancedRandomForest`) tramite `Optuna` con cross-validation.
- **Stacking ensemble:** creazione di un meta-dataset combinando le predizioni dei modelli base e le feature originali.
- **Ottimizzazione dei Meta-Modelli** Ottimizzazione dei meta-modelli utilizzando anche in questo caso il metodo optuna.
- **Training del meta-modello:** addestramento del classificatore finale. In questa fase si procede eseguendo diversi esperimenti, i modelli supervisionati finali addestrati sono: Random Forest, Light Gradient Boosting Machine, Gradient boosting Classifier, XgBoost sul metadataset. Per il German Dataset è stata utilizzata anche la regressione logistica.
- **Calibrazione dei Meta-Modelli:** Si è proceduto inoltre alla calibrazione dei modelli finali, ricordando infatti che l'obiettivo oltre a migliorare della recall della classe dei default è quello di calcolare la probabilità di default.
- **Ottimizzazione della soglia di decisione:** Si è scelta la probabilità di soglia decisionale ottimale validando i modelli sul Meta-validation-Set, in modo da ottenere le migliori: `F1-score` e `balanced accuracy`
- **Valutazione finale:** calcolo delle performance finali su test set (F1-score, classification report) e stima della probabilità media e varianza del modello per il calcolo di misure di rischio.

3.3.3 Feature engineering

In questa fase solo per l'UCI Credit Risk si decide di ampliare il contenuto informativo del dataset. Dunque vengono aggiunte al database di partenza alcune variabili, derivate dalle caratteristiche già presenti nela database, le feature aggiuntive sono mostrate di seguito:

Variabile	**Descrizione e Motivazione**	**Tipo**
MAX_UTIL_RATIO	Rapporto massimo tra l'importo di fattura e il limite di credito. Serve a catturare episodi di uso elevato del credito, potenzialmente associati a rischio di default.	Quantitativa
MEAN_UTIL_RATIO	Rapporto medio di utilizzo del credito nei sei mesi. Una media elevata indica un utilizzo costante e intenso della linea di credito, possibile segnale di stress finanziario.	Quantitativa
STD_BILL	Deviazione standard delle fatture mensili. Misura la variabilità della spesa: alta variabilità può indicare comportamento instabile o imprevedibile nei consumi.	Quantitativa
TOTAL_PAYMENT	Somma totale dei pagamenti effettuati. Utile per distinguere clienti che tendono a ripagare attivamente il debito da quelli che accumulano solo fatture.	Quantitativa
AVG_PAY_RATIO	Rapporto medio tra pagamento e fattura. Indica la propensione del cliente a saldare le fatture: valori bassi possono segnalare rischio di insolvenza.	Quantitativa
PAY_TREND	Differenza tra PAY_6 e PAY_3. Misura l'evoluzione della puntualità nei pagamenti: un trend peggiorativo può essere un forte indicatore di rischio futuro.	Ordinale
LATE_COUNT	Numero di mesi (su 4) in cui il cliente ha avuto ritardi (PAY_i $\geq$ 1). Sintetizza la frequenza delle situazioni di ritardo, utile come indicatore diretto del comportamento scorretto.	Discreta / Ordinale

Tabella 3.3: Nuove variabili aggiunte e loro motivazioni informative

Nel contesto del rischio di credito la creazione di nuove variabili è molto comune. Ad esempio in *"Feature engineering in credit risk models: key to optimization"* [3] si discute al livello generale dell'importanza di trend, minimi e massimi e statistiche varie in questo ambito. Nel lavoro di Huang et al. [1], il modello classico RFM (Recency, Frequency, Monetary) viene esteso introducendo la componente "S" (Standard Deviation), che misura la volatilità degli importi spesi. Tale estensione, nota come RFMS, migliora la capacità predittiva del credit scoring. In modo analogo, la variabile `STD_BILL` introdotta in questo lavoro rappresenta la deviazione standard delle fatture mensili, utile per cogliere comportamenti instabili o anomali nei consumi. Le variabili ottenute, quelle originali e quelle create artificialmente sono tra di loro correlate. Per questo i modelli di Machine Learning che verranno costruiti in seguito sono tutti di tipo non parametrico poché stabili contro le problematiche della correlazione tra variabili.

3.3.4 K-Means e Clustering a supporto dei modelli Supervisionati

Come visto in *"Integration of unsupervised and supervised machine learning algorithms for credit risk assessment"* [8], l'utilizzo di tecniche di clustering come input iniziale del lavoro, può essere discriminante nell'operato dei modelli supervisionati. Per questo è stata introdotta nell'analisi l'utilizzo di tale algoritmo così da ottenere due nuove features:

- `Default Cluster`
- `CLUSTER_DISTANCE`

Nel tecnico si utilizza il K-Means su un sottinsieme formato da soli clienti in default , in questo modo si riescono a distinguere due cluster, quindi a determinare due diversi profili di default. Una volta determinati i cluster si assegnano tutti i punti del dataset ai due insiemi, considerando la distanza tra i centroidi. Dunque otteniamo una nuova feature: `Default Cluster`. Tale variabile indica per i default la tipologia di creditore insolvente a cui appartengono, e per i non default il profilo di debitore inadempiente a cui assomigliano maggiormente. Dovendo assegnare a tutti i punti nel dataset la variabile, anche gli elementi che si trovano distanti da entrambi i centroidi saranno assegnati a uno dei due cluster, difatti anche gli elementi lontani da entrambi i cluster. Dunque per determinare la forza del legame tra l'elemento e il cluster di appartenenza è stata aggiunta un'ulteriore variabile: `CLUSTER_DISTANCE` che indica la distanza dal punto al centroide di appartenenza.

SMOTETomek

Il problema principale del dataset in questione, o comunque di quasi tutti i dataset nel contesto del rischio di credito è lo sbilanciamento tra le classi. Sono presenti in generale, molti più clienti non in default rispetto a quelli in default, tale disuguaglianza conduce alle maggiori problematiche per i modelli supervisionati. Il fenomeno che accade infatti è che i modelli si addestrano bene per la previsione della classe maggioritaria e falliscono l'inferenza per la classe in minoranaza. Proprio per questo vengono utilizzate tecniche di oversampling per ovviare a questo problema. Nei due casi studio le tecniche utilizzate sono le medesime. Sia per l'UCI-CREDIT-RISK che per il German Credit Data si è deciso di utilizzare *SMOTETomek*. Di seguito si discuterà del metodo utilizzato. SMOTETomek è una tecnica utilizzata per gestire dataset sbilanciati, combinando due approcci distinti:*SMOTE* (Synthetic Minority Over-sampling Technique) e i *Tomek Links*. SMOTE agisce generando nuovi esempi sintetici della classe minoritaria. Per ciascun esempio minoritario x_i, vengono individuati i k vicini più prossimi della stessa classe (tipicamente $k = 5$ anche in questo lavoro è stato utilizzato questo valore). Uno di questi vicini (x_{nn}) viene selezionato randomicamente e un nuovo punto sintetico viene generato tramite interpolazione lineare secondo la formula:

$$x_{\text{new}} = x_i + \delta \cdot (x_{nn} - x_i),$$

dove δ è un numero casuale $\in[0,1]$ Questo processo consente di arricchire la distribuzione della classe minoritaria in modo più realistico, senza duplicazioni.

Successivamente, i Tomek Links vengono utilizzati per ripulire il dataset. Una *Tomek Link* è una coppia di esempi (x_i, x_j) appartenenti a classi diverse, tali che x_i è il vicino più prossimo di x_j e viceversa. Queste coppie rappresentano punti vicini al confine tra le classi e quindi ambigui. Rimuovendo uno o entrambi gli esempi della classe maggioritaria coinvolti in una Tomek Link, si migliora la separabilità delle classi, riducendo il rumore e migliorando l'efficacia dei modelli predittivi.

Classe	**Prima di SMOTETomek**	**Dopo SMOTETomek**
Classe 0 (non-default)	14,018	13,111
Classe 1 (default)	3,982	6,102

Tabella 3.4: Distribuzione delle classi prima e dopo l'applicazione di SMOTETomek

3.3.5 Optuna

Per l'ottimizzazione dei modelli si procede facendo validazione incrociata sull'insieme di training con 5 fold. La cross Validation è una tecnica utilizzata per validare i modelli di machine learning. Si suddivide il dataset in un numero k di insiemi, denominati fold. Si addestra ad ogni iterazione il modello su l'intero set di dati a meno di un fold che sarà l'insieme su cui verrà valutato il modello. In ciascuna iterazione ogni osservazione è esclusa dal training set e usata come test almeno una volta, quindi il modello produce una predizione per ogni osservazione senza che questa abbia mai contribuito all'addestramento nella stessa iterazione in cui viene predetta. Come si discuteva in precedenza il nostro obiettivo è quello di ottenere un miglioramento delle metriche della classe 1, specialmente nella recall.

Nell'applicazione dei singoli modelli nella fase compilativa del lavoro di tesi, la selezione degli iperparametri ottimali è stata effettuata tramite il metodo `GridSearch`, che esegue la ricerca valutando tutte le combinazioni possibili all'interno di uno spazio predefinito di parametri.

In alternativa, nell'ambito dei metodi ensemble si propone l'utilizzo di `Optuna` , un approccio di ottimizzazione iperparametrica basato su strategie di tipo bayesiano. A differenza di `GridSearch`, che esplora in modo sistematico l'intero spazio, `Optuna` adotta un approccio più intelligente e dinamico, in grado di convergere più rapidamente verso combinazioni di iperparametri promettenti, riducendo significativamente il tempo computazionale necessario per la ricerca. Di seguito mostriamo il funzionamento di tale metodo e come è stato adattato al lavoro corrente.

Sia

$$f : \mathcal{X} \to \mathbb{R}$$

Una funzione obiettivo di costo, (nell' esperimento proposto è stato scelto di utilizzare l'f1-score per la classe di default, quindi l'obiettivo non sarà trovare il minimo ma il massimo). Si prosegue nella definizione del metodo:

$\mathcal{X} \subseteq \mathbb{R}^d$ rappresenta lo spazio di ricerca degli iperparametri.

L'obiettivo è trovare:

$$x^* = \arg\min_{x \in \mathcal{X}} f(x)$$

L'algoritmo Optuna con TPE si articola nei seguenti passaggi:

1. **Raccolta delle osservazioni**
   Dopo n valutazioni, n=10 nel nostro caso studio.

$$\mathcal{D} = \{(x_i, y_i) \mid i = 1, \ldots, n\}, \quad \text{con } y_i = f(x_i)$$

2. **Definizione della soglia y^***
   Si fissa un quantile $\gamma = (0{,}2)$ (tipico in questo ambito) e si definisce la soglia:

$$y^* = \inf \{y \in \mathbb{R} \mid \mathbb{P}(f(x) < y) \geq \gamma\}$$

   Si suddividono quindi le osservazioni nei due sottoinsiemi:

$$\mathcal{D}_{\text{good}} = \{x_i \mid y_i < y^*\}, \qquad \mathcal{D}_{\text{bad}} = \{x_i \mid y_i \geq y^*\}$$

3. **Stima delle distribuzioni**
   Si stimano le seguenti densità tramite Kernel Density Estimation (KDE):

$$l(x) = p(x \mid y < y^*) \approx \hat{p}_{\text{good}}(x)$$

$$g(x) = p(x \mid y \geq y^*) \approx \hat{p}_{\text{bad}}(x)$$

4. **Funzione di acquisizione**
   Il prossimo candidato viene scelto massimizzando il rapporto tra le due distribuzioni:

$$x_{t+1} = \arg\max_{x \in \mathcal{X}} \frac{l(x)}{g(x)} = \arg\max_{x \in \mathcal{X}} \frac{p(x \mid y < y^*)}{p(x \mid y \geq y^*)}$$

   In pratica, si campionano vari punti da $l(x)$ e si seleziona quello che massimizza il rapporto rispetto a $g(x)$. In questo modo il candidato viene scelto tra i valori migliori.

5. **Valutazione e aggiornamento**
   Il candidato selezionato x_{t+1} viene valutato:

$$y_{t+1} = f(x_{t+1})$$

   e la base di dati viene aggiornata:

$$\mathcal{D} \leftarrow \mathcal{D} \cup \{(x_{t+1}, y_{t+1})\}$$

   Il processo viene quindi ripetuto.

L'Optuna si fonda su *"Kernel Density Estimation"* (KDE), una tecnica non parametrica utilizzata per stimare la densità di probabilità di una variabile aleatoria a partire da un campione finito di osservazioni. A differenza dei metodi parametrici, che assumono una forma prestabilita per la distribuzione (come ad esempio la distribuzione normale), KDE

costruisce una stima meno rigida, sommando delle funzioni *kernel* centrate su ciascuna osservazione.

Nell'utilizzo di `Optuna`, KDE stima due distribuzioni conizionate, quella delle "buone osservazioni" e quella delle "cattive osservazioni", rispettivamente:

$$l(x) = p(x \mid y < y^*), \quad g(x) = p(x \mid y \geq y^*)$$

Le stime delle distribuzioni ottenute servono ad *Optuna* per scegliere la prossima configurazione di parametri da valutare nel modo descritto precedentemente.

Nel capitolo dei risultati sperimentali mostreremo i parametri ottenuti per ogni modello supervisionato validato.

3.3.6 Creazione del modello finale

Dopo aver perfezionato i cinque modelli di base (XGBoost, CatBoost, Random Forest Bagging, LightGBM e solo per il german credit dataset la Regressione Logistica) sui dati di training bilanciato, vengono sottoposti a una cross-validation a 5 fold per ricavare, "out-of-fold", le probabilità di default di ciascun campione. Queste predizioni vengono salvate su disco e utilizzate come nuove feature nel metadataset di training, affiancandole alle variabili originali. Successivamente, viene addestrato sul training set di partenza (senza le probabilità) e vengono calcolate le probabilità di default anche sui set di validazione e di test; i vettori risultanti vengono impilati fra loro e poi concatenati alle rispettive feature scalate, ottenendo così i metadataset di validation e di test pronti per alimentare il meta-learner nello schema di stacking.

Nella fase finale vengono scelti 4 modelli da utilizzare come modelli finali:

- Xgboost
- Random Forest
- Light Gradient Boosting Machine
- Gradient Boosting Decision Trees
- Regressione Logistica (Sono con il German Credit Data)

Vengono ottimizzati i modelli finali come fatto per i modelli supervisionati in precedenza. Utilizzando il metodo *Optuna* facendo cross validation sul nuovo insieme di training creato. Il nuovo insieme di validazione ottenuto, viene utilizzato per calcolare la probabilità di soglia decisionale.

3.3.7 Probabilità di soglia

Quando il dataset è sbilanciato come si è detto i modelli riescono a fare inferenza molto bene sulla classe maggioritaria, ma non performano in maniera ottimale sulla classe minoritaria. Questo bias conduce in generale ad assegnare probabilità basse nel caso specifico (nel nostro caso ottenere probabilità alte equivale ad appartenere alla classe di default

quindi quella minoritaria). Utilizzare quindi come soglia di probabilità 0.50 potrebbe condurre ad un netto abbassamento della *recall* della classe 1. Proprio per questo sono stati validati i modelli finali in due modi, massimizzando la f-1 score, una metrica dipendentente sia dalla recall che dalla precision:

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

massimizzando la balanced accuracy nell'altro caso:

$$\text{Balanced Accuracy} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right)$$

Viene utilizzata la balanced accuracy perché tiene conto della recall delle due classi. La probabilità di soglia ottenuta ottimizzando queste due metriche è solo indicativa. Infine vengono testati i modelli sull'insieme di test, invece per il calcolo della probabilità e la correlazione vengono utilizzati i modelli calibrati.

Nel prossimo capitolo verranno mostrati e discussi i risultati ottenuti su entrambi i dataset, per le due fasi del lavoro, quella di riproduzione e quella sperimentale.

Chapter 4

Risultati Sperimentali

4.1 UCI CREDIT RISK

4.1.1 Introduzione

In questocapitolo vengono presentati e confrontati i risultati ottenuti, mettendo in evidenza le differenze tra le prestazioni ottenute nella prima fase (Riproduzione delle tecniche di machine learning utilizzate in *"Machine Learning in Joint Default Assessment"* [4]) e quelle ottenute nella fase 2 (Costruzione di una nuova pipeline basata su i modelli ensemble al fine di migliorare le performance). Si inizia presentando le evidenze ottenute nella prima fase sull'UCI Credit Risk. Inoltre prima di mostrare tutti i risultati ottenuti, si ricorda che il rapporto tra i clienti in default e i clienti totali per l'UCI Credit Risk è 0.22.

4.1.2 Modelli singoli

Tuning dei modelli

Per ogni modello, come descritto nel capitolo della replica dell'esperimento è stata utilizzata *gridsearch* (a meno della regressione logistica) per ottimizzare i parametri degli algoritmi, di seguito quindi (per ogni algoritmo) vengono mostrati gli iperparametri ottenuti.

Regressione Logistica

Vengono mostrati di seguito gli iperparametri ottenuti:

Parametro	**Default**	**Descrizione**
`penalty`	`"l2"`	Tipo di penalizzazione per la regolarizzazione. Può essere `"l1"`, `"l2"`, `"elasticnet"` o `"none"`.
`C`	`1.0`	Inverso della forza di regolarizzazione. Valori più piccoli implicano una regolarizzazione più forte.
`fit_intercept`	`True`	Specifica se calcolare l'intercetta del modello. Se `False`, si assume che i dati siano centrati.
`tol`	`1e-4`	Tolleranza per i criteri di arresto dell'ottimizzazione.
`n_jobs`	`None`	Numero di core da utilizzare per il calcolo. Funziona solo con i solver `liblinear` e `saga`.

Tabella 4.1: Principali iperparametri del modello `LogisticRegression` di `scikit-learn`

Gli iperparametri scelti per la regressione logistica sono quelli di default della libreria scikit-learn. Si prosegue adesso mostrando le metriche di valutazione e la curva ROC del modello:

Classe	Precision	Recall	F1-score	Support
0	0.78	1.00	0.88	4673
1	0.00	0.00	0.00	1327
Accuracy		0.78		6000
Macro avg	0.39	0.50	0.44	6000
Weighted avg	0.61	0.78	0.68	6000

Tabella 4.2: Classification report con precision, recall, F1-score e support per ciascuna classe

Come si evince dalle metriche di valutazione la regressione in questo caso non riesce a lavorare correttamente con la classe dei default, mentre ottiene performance altissime per la classe dei non default, visualizziamo adesso la curva ROC ottenuta.

Figura 4.1: curva ROC

L'AUC è 0.64. Osservare il valore dell'AUC è sempre interessante perché indica una valutazione riassuntiva del modello. Nello specifico come si sottolineava in precedenza l'obiettivo è ottenere un modello cauto nella classificazione dei default.

Si prosegue l'analisi dei risultati mostrando la probabilità di default e la correlazione tra i creditori stimata. La regressione logistica non necessita calibrazione per il calcolo

Parametro	**Valore stimato**
β_1	2.419
β_2	6.788
π_1 (Probabilità di default)	0.263
ρ (Correlazione tra i creditori)	0.098

Tabella 4.3: Stime dei parametri del modello: coefficienti β, probabilità di default π_1 e correlazione ρ

della probabilità. Vengono visualizzate le evidenze ottenute in tabella ?? Inoltre, viene osservato il delta tra la probabilità di default emprica e la probabilità di default stimata in figura 4.2.

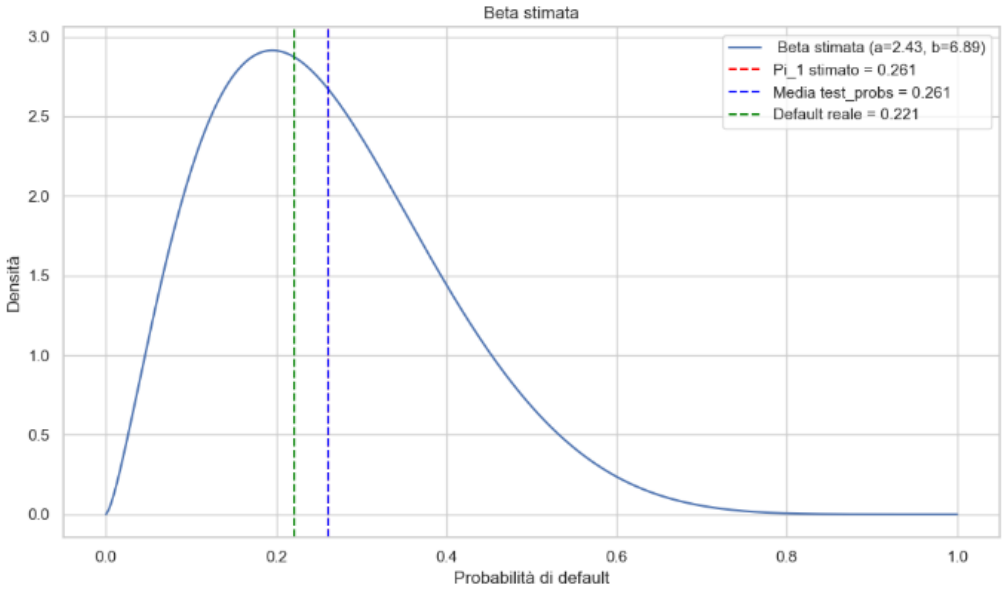

Figura 4.2: Beta con i parametri stimati dalla regressione logistica

Random Forest

Si procede con i risultati allo stesso modo. Per i modelli non parametrici è stata fatta *gridsearch* per valutare gli iperparametri ottimi per la random forest risultano i seguenti:

Parametro	**Valore**	**Descrizione**
`max_depth`	6	Profondità massima degli alberi. Limita la crescita degli alberi per evitare overfitting.
`min_samples_split`	5	Numero minimo di campioni richiesto per suddividere un nodo interno. Maggiore è il valore, più conservativa è la crescita dellâalbero.
`n_estimators`	70	Numero di alberi . Aumentarlo può migliorare la performance, ma anche il costo computazionale.

Tabella 4.4: Iperparametri utilizzati per il modello `RandomForestClassifier`

Le performance ottenute dal modello invece:

Classe	**Precision**	**Recall**	**F1-score**	**Support**
0	0.83	0.95	0.89	4673
1	0.67	0.34	0.41	1327
Accuracy	0.81			6000
Macro avg	0.75	0.63	0.65	6000
Weighted avg	0.79	0.81	0.78	6000

Tabella 4.5: Classification report: precision, recall, F1-score e support per ciascuna classe

Come si vede dalla tabella riportata, le metriche ottenute sono nettamente migliori rispetto alla regressione logistica, infatti per la classe dei non default il modello riesce bene a fare inferenza. Inoltre si può notare una buona "precision" sulla classe dei default. Di seguito viene mostrata la curva ROC ottenuta.

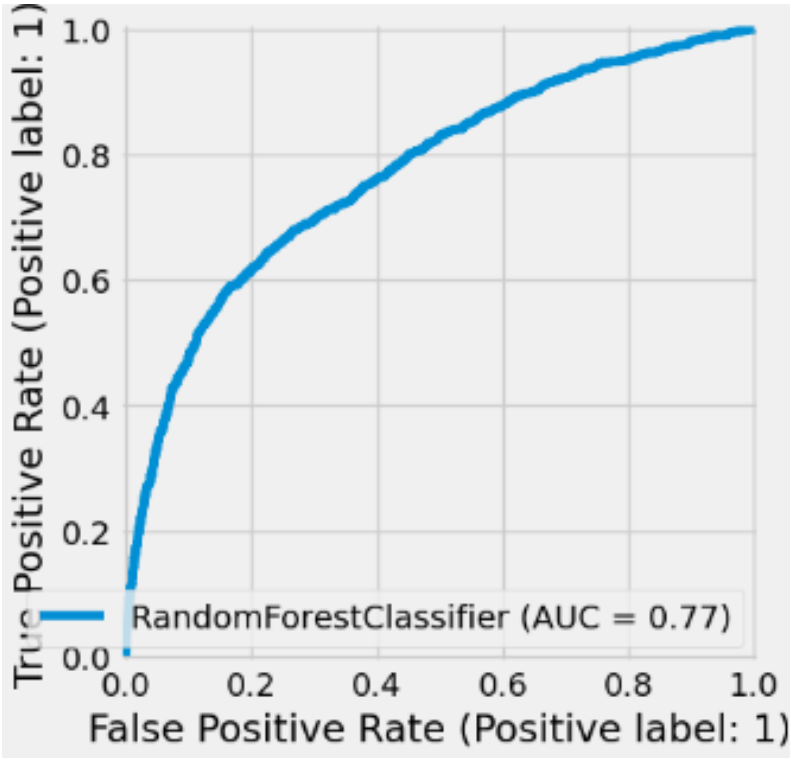

Figura 4.3: curva ROC Random Forest

Come si vede in figura il coefficiente AUC risulta di 0.77 quindi da un punto di vista generale il modello funziona molto bene. La Random Forest e tutti i modelli che verranno mostrati in seguito saranno calibrati, in particolare il modello viene calibrato mediante la regressione isotonica. Dopo la calibrazione otteniamo:

Parametro	**Valore stimato**
β_1	0.687
β_2	2.394
π_1 (Probabilità di default)	0.22
ρ (Correlazione tra i creditori)	0.24

Tabella 4.6: Stime aggiornate dei parametri del modello: coefficienti β, probabilità di default π_1 e correlazione ρ

Viene mostrata la beta simulata con i parametri stimati e il delta tra la probabilità reale e quella stimata risultano

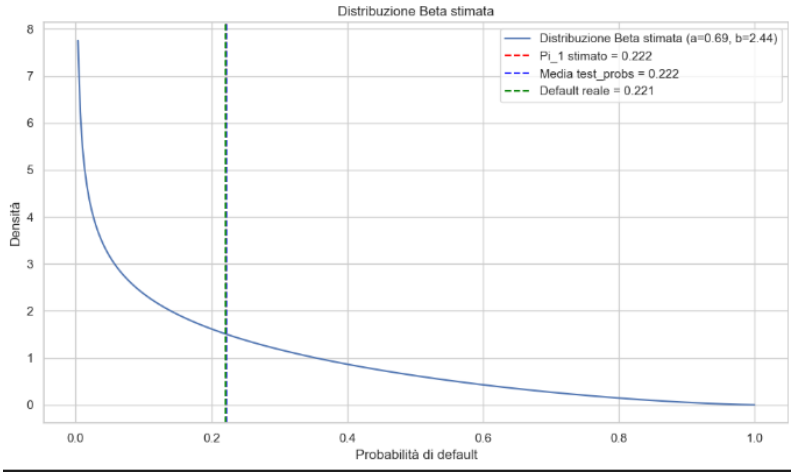

Figura 4.4: Beta con i parametri stimati dalla random forest

AdaBoost

In seguito si procede con l'Adaptive Boosting, gli iperparametri ottenuti e le metriche corrispondenti sono mostrate di seguito: Tali metriche risultano molto simili a quelle ottenute con la random forest.

Parametro	Valore	Descrizione
`learning_rate`	0.75	Fattore di apprendimento che ridimensiona il contributo di ciascun classificatore debole. Valori più piccoli rendono l'addestramento più conservativo, ma potenzialmente più accurato.
`n_estimators`	100	Numero di classificatori deboli da addestrare in sequenza. Un numero maggiore può migliorare la performance, ma aumenta anche il rischio di overfitting.

Tabella 4.7: Iperparametri utilizzati per il modello `AdaBoostClassifier`

Classe	Precision	Recall	F1-score	Support
0	0.84	0.95	0.89	4673
1	0.67	0.35	0.46	1327
Accuracy	0.82			6000
Macro avg	0.75	0.65	0.67	6000
Weighted avg	0.80	0.82	0.79	6000

Tabella 4.8: Classification report: precision, recall, F1-score e support per ciascuna classe

La curva ROC del modello è la seguente:

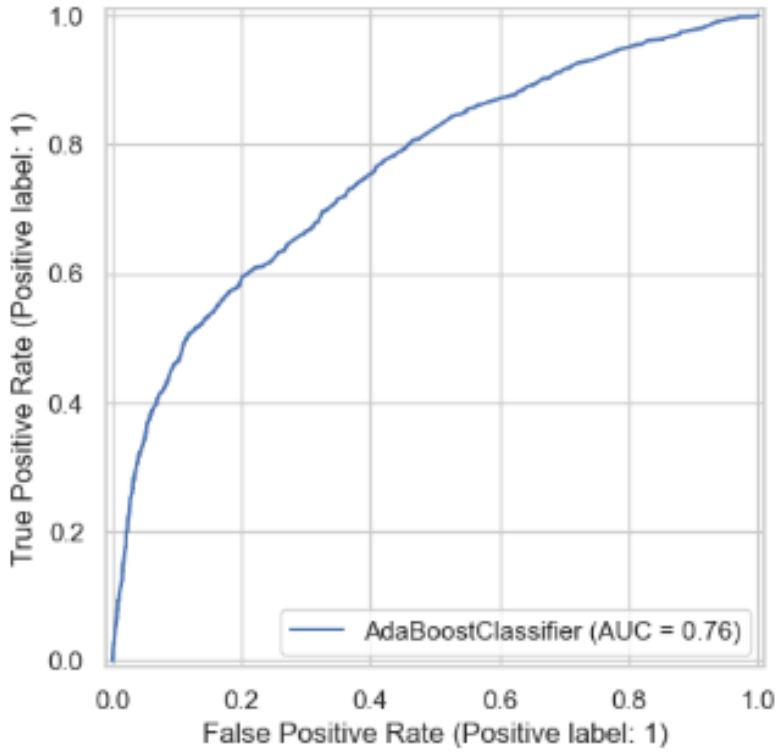

Figura 4.5: curva ROC Adaboost

Proseguiamo ancora mostrando la probabilità di default e la correlazione calcolata:

Parametro	**Valore stimato**
β_1	0.795
β_2	2.801
π_1 (Probabilità di default)	0.221
ρ (Correlazione tra i creditori)	0.21

Tabella 4.9: Stime dei parametri del modello: coefficienti β, probabilità di default π_1 e correlazione ρ

Anche i valori dei parametri stimati risultano molto simili a quelli ottenuti con la Random Forest infatti la forma della curva ottenuta è molto simile:

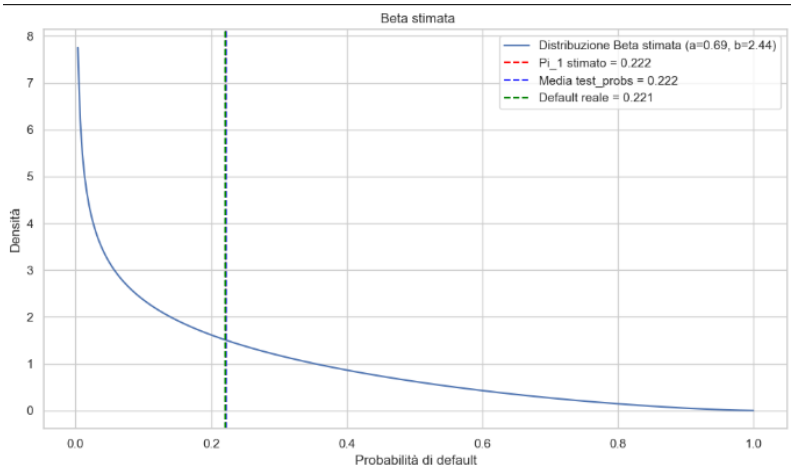

Figura 4.6: Beta con i parametri stimati dall'Adative Boosting

Infine l'ultimo modello proposto è il KNN.

KNN

Infine vengono mostrati:

- iperparametri ottenuti
- metriche ottenute dal modello
- probabilità e correlazione con i parametri della beta stimati.

A differenza degli altri modelli per il modello k-Nearest Neighbors (k-NN) è stata adottata la calibrazione tramite regressione sigmoidea (sigmoid calibration), anche nota come Platt scaling. Come ci aspettiamo i modelli singoli hanno ottenuto buone performance in

Parametro	Valore	Descrizione
`n_neighbors`	9	Numero di vicini più prossimi considerati per assegnare l'etichetta alla nuova istanza. Un numero dispari è spesso usato per evitare pareggi nella classificazione binaria.
`metric`	`'euclidean'`	Tipo di metrica di distanza utilizzata per calcolare la vicinanza tra i punti. La distanza euclidea è la più comune e misura la distanza "lineareâ nello spazio delle feature.

Tabella 4.10: Iperparametri utilizzati per il modello `KNeighborsClassifier`

generale, ma non sono riusciti ad ottenere una recall della classe 1 maggiore di 0.5. Concludiamo mostrando la probabilità e la correlazione stimate. Seguite dal grafico della beta.

Tabella 4.11: Classification Report

Classe	Precision	Recall	F1-Score	Support
0	0.84	0.94	0.88	4673
1	0.61	0.35	0.45	1327
Accuracy	0.81			
Macro Avg	0.72	0.64	0.67	6000
Weighted Avg	0.79	0.81	0.79	6000

Tabella 4.12: Parametri stimati della distribuzione Beta

Parametro	Valore
a	0.5211
b	1.8083
π_1 (media)	0.2237
ρ (varianza ridotta)	0.3004

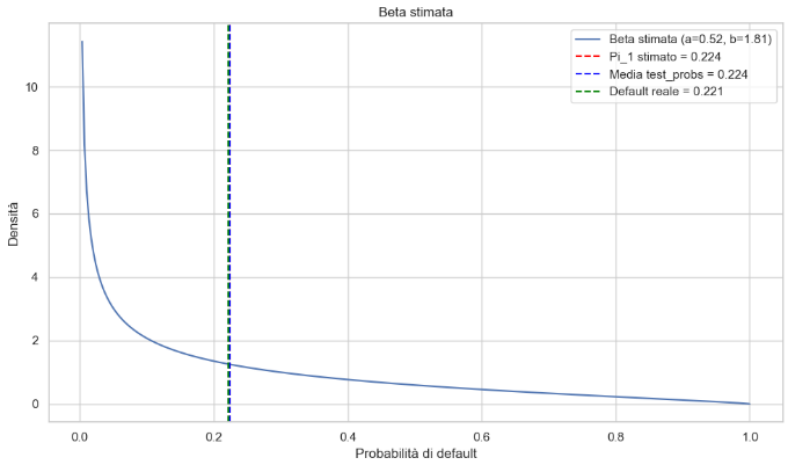

Figura 4.7: Curva Beta Knn

4.1.3 Modelli ensemble

Si procede con i risultati sperimentali, ottenuti con la nuova procedura discussa nel capitolo della metodologia. Dunque vengono mostrati gli iperparametri dei modelli ottimizzati con *Optuna*, e i risultati ottenuti per ogni modello. I meta-modelli, utilizzati come modelli finali del metodo ensemble per l'UCI Credit Risk sono:

- Random Forest
- Xgboost
- Light Gradient Boosting Machine
- Gradient Boosting

Come è stato mostrato nel capitolo della metodologia per ogni modello è stata ricercata la probabilità di soglia ottimizzando le metriche:

- Balanced Accuracy
- F1-Score

Si procede dunque con l'analisi dei risultati

Random Forest

Gli iperparametri ottenuti, utilizzando la Random Forest come meta-modello finale sono:

Parametro	Valore	Descrizione
`class_weight`	`"balanced"`	Regola automaticamente i pesi delle classi in base alla loro frequenza inversa nel dataset di addestramento. Utile in caso di classi sbilanciate.
`max_depth`	`6`	Profondità massima di ciascun albero nella foresta. Limita l'overfitting e controlla la complessità del modello.
`n_estimators`	`44`	Numero di alberi nella foresta. Un numero maggiore tende a migliorare la performance, ma aumenta anche il tempo di addestramento.
`n_jobs`	`-1`	Numero di job da eseguire in parallelo. (`-1`) utilizza tutti i core disponibili.
`random_state`	`42`	garantisce la riproducibilità dei risultati.

Tabella 4.13: Parametri ottimi della `random Forest`

La probabilità di soglia ottenuta ottimizzando la Random Forest sull'insieme di validazione, e la Balanced accuracy associata risultano rispettivamente:

- 0.33 Probabilità
- 0.68 Balanced Accuracy

Le metriche ottenute sull'insieme di test invece sono mostrate in tabella 4.14 Otteniamo

Classe	Precision	Recall	F1-score	Support
0	0.87	0.72	0.79	4673
1	0.46	0.60	0.46	1327
Accuracy	0.76 (su 6000 campioni)			
Macro avg	0.62	0.66	0.63	6000
Weighted avg	0.76	0.70	0.72	6000

Tabella 4.14: Classification report della Random Forest con probabilità di soglia 0.33

una recall della classe dei default di 0.60 che seppur avendo abbasato la soglia a 0.33 è un buon risultato. Vediamo dunque le metriche ottenute con la probabilità calcolata massimizzando F1-Score. Probabilità di soglia e score associato sono rispettivamente:

- 0.34 Probabilità di soglia
- 0.49 F1 Score

Le metriche ottenute sull'insieme di test sono rispettivamente:

Configurazione del modello	**Recall classe 1**
Random Forest singola	0.34
Random Forest (ensemble senza validazione)	0.46
Random Forest (ensemble, validazione su balanced accuracy)	0.60
Random Forest (ensemble, validazione su F1-score)	0.60

Tabella 4.15: Valori di recall per la classe 1 in 4 diverse configurazioni del modello Random Forest

Si mostra un riepilogo delle performance ottenute confrontando il modello ensemble con il modello singolo.

Configurazione del modello	**Recall classe 1**
Random Forest singola	0.34
Random Forest (ensemble, validazione su balanced accuracy)	0.46
Random Forest (ensemble, validazione su F1-score)	0.61

Tabella 4.16: Valori di recall per la classe 1 in 3 diverse configurazioni del modello Random Forest

Dopo la calibrazione infatti la probabilità stimata risulta molto simile a quella effettiva. Dunque i valori risultanti e la beta ottenuta sono:

Parametro	**Valore**
Intervallo	[0.52 1.87]
π_1 (probabilità di default)	0.22
ρ (parametro di correlazione)	0.30

Tabella 4.17: Parametri stimati del modello

La differenza tra i due valori di probabilità risulta molto basso, il modello calibrato riesce a stimare correttamente le probabilità.

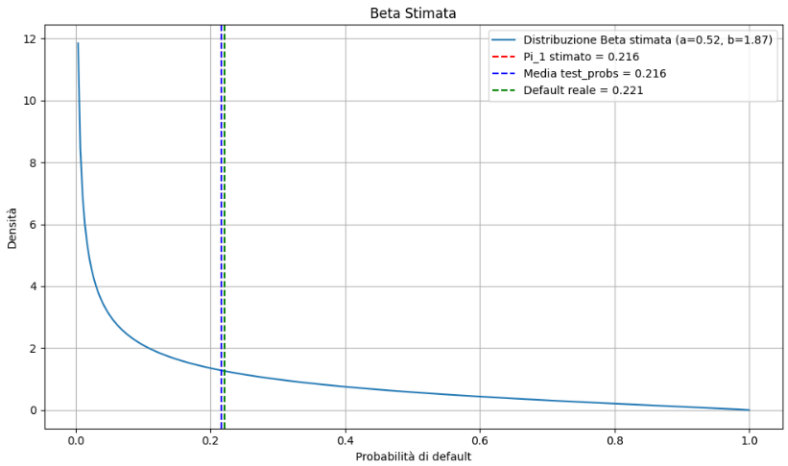

Figura 4.8: Beta con i parametri stimati dalla random forest come meta-modello finale

XgBoost

L'Xgboost è un modello molto potente essendo già singolarmente un modello ensemble. Gli iperparametri ottenuti risultano i seguenti:

Parametro	**Valore**	**Descrizione**
`eval_metric`	`"logloss"`	Funzione di valutazione usata durante l'addestramento, qui la log-loss, tipica per problemi di classificazione binaria.
`learning_rate`	0.003	Tasso di apprendimento: controlla quanto ogni albero corregge l'errore del precedente.
`max_depth`	5	Profondità massima degli alberi
`n_estimators`	109	Numero di alberi da addestrare.
`n_jobs`	-1	Numero di thread usati per l'addestramento. Il valore -1 utilizza tutti i core disponibili.

Tabella 4.18: Parametri ottimizzati del modello `XGBClassifier`

La probabilità di soglia e la balanced accuracy associata, ottenuti ottimizzando quest'ultima sul set di validazione risultano rispettivamente:

- 0.30 Probabilità
- 0.67 Balanced Accuracy

La probabilità di soglia ottenuta è nuovamente 0.30. Con tale valore di soglia le metriche ottenute sono molto simili a quelle calcolate nell'esperimento precedente, in cui viene testata la Random Forest. Mostriamo le metriche ottenute sull'insieme di test:

Classe	**Precision**	**Recall**	**F1-score**	**Support**
0	0.87	0.75	0.80	4673
1	0.39	0.57	0.46	1327
Accuracy	0.71 (su 6000 campioni)			
Macro avg	0.63	0.66	0.63	6000
Weighted avg	0.76	0.71	0.73	6000

Tabella 4.19: Classification report per il modello XGBoost con probabilità di soglia a 0.33

Per l'F1-Score i valori ottenuti sull'insieme di validazione risultano simili a quelli calcolati durante l'ottimizzazione della balanced accuracy infatti:

- 0.29 Probabilità
- 0.49 F1-Score

Classe	**Precision**	**Recall**	**F1-score**	**Support**
0	0.85	0.78	0.81	4673
1	0.40	0.59	0.45	1327
Accuracy	0.72 (su 6000 campioni)			
Macro avg	0.63	0.65	0.63	6000
Weighted avg	0.75	0.72	0.73	6000

Tabella 4.20: Classification report con soglia di probabilità 0.29

Probabilità di default

Anche per l'Xgboost è stata necessaria la calibrazione isotonica, di seguito mostriamo i valori calcolati con il metodo dei momenti e la beta ottenuta con i parametri stimati dal modello:

Parametro	Valore
Parametri Beta	[0.60, 2.32]
Probabilità di default (π_1)	0.21
Correlazione (ρ)	0.25

Tabella 4.21: Parametri stimati: distribuzione Beta, probabilità di default e correlazione

Anche in questo caso il modello stima bene la probabilità di default, il delta con il valore reale è praticamente nullo:

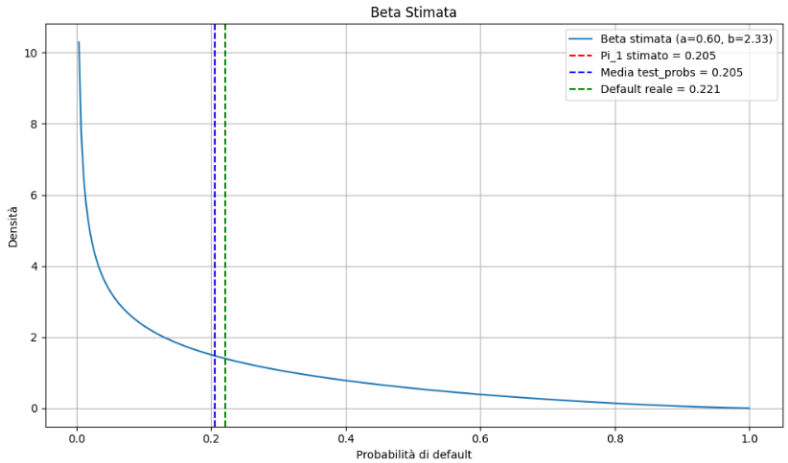

Figura 4.9: Beta con i parametri stimati dal Xgboost come meta-modello finale

Si procede adesso con gli ultimi due modelli.

Ligh Gradient Boosting Machine

Gli iperparametri ottenuti per Ligh Gradient Boosting Machine sono i seguenti:

Parametro	**Valore**	**Descrizione**
`class_weight`	`"balanced"`	Bilancia automaticamente le classi in base alla loro frequenza.
`learning_rate`	0.0142	Tasso di apprendimento: controlla la velocità con cui il modello apprende.
`max_depth`	7	Profondità massima degli alberi per limitare l'overfitting.
`n_estimators`	115	Numero totale di alberi da costruire.
`n_jobs`	-1	Utilizza tutti i core della CPU disponibili.
`random_state`	42	Fissa la casualità per riproducibilità dei risultati.

Tabella 4.22: Parametri ottimi del modello LGBMClassifier

Proseguiamo con le metriche ottenute con i due metodi di validazione utilizzati.

La soglia decisionale e il valore della balanced accuracy ottenuti ottimizzando tale metrica sono i seguenti:

- 0.33 Probabilità
- 0.68 Balanced Accuracy

Dopo aver testato il modello otteniamo:

Classe	Precision	Recall	F1-score	Support
0	0.86	0.73	0.79	4673
1	0.38	0.59	0.46	1327
Accuracy	0.70			
Macro avg	0.62	0.65	0.62	6000
Weighted avg	0.75	0.70	0.72	6000

Tabella 4.23: Classification report del modello LGBMClassifier con probabilità di soglia 0.33

La recall ottenuta risulta circa di 0.60 con la probabilità di soglia decisionale di 0.33.

Otimizzando l'F1-Score si ottiene:

- Probabilità di soglia 0.33
- F1-Score 0.49

Dunque le metriche ottenute sono identiche a quelle stimate ottimizzando la balanced accuracy. Si procede con la calibrazione del modello per il calcolo della probabilità di default e la visualizzazione della beta ottenuta con in parametri stimati.

Parametro	**Valore**
Parametri Beta	[0.57, 2.19]
π (Probabilità di Default)	0.21
ρ (Correlazione)	0.2660

Tabella 4.24: Valori stimati: parametri Beta, probabilità di default π e correlazione ρ

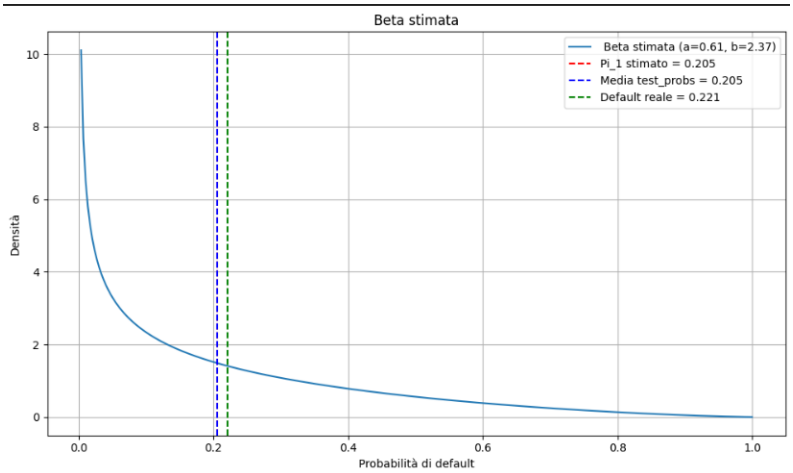

Figura 4.10: Beta con i parametri stimati dal Lgbm come meta-modello finale

Gradient Boosting

Viene introdotto l'ultimo meta-modello utilizzato, il Gradient Boosting. I parametri ottenuti dopo il metodo optuna:

Parametro	Valore	Descrizione
`learning_rate`	0.035	Tasso di apprendimento: determina quanto ciascun albero contribuisce al modello finale.
`max_depth`	3	Profondità massima degli alberi.
`n_estimators`	155	Numero di alberi da costruire nella sequenza di boosting.
`random_state`	42	Inizializzazione casuale controllata per garantire la riproducibilità.

Tabella 4.25: Parametri ottimizzati del modello GradientBoostingClassifier

La probabilità di soglia e il valore di balanced accuracy ottenuti risultano:

- 0.30 Probabilità
- 0.65 Score associato

Di seguito mostriamo le metriche ottenute: Le metriche risultanti per la classe dei de-

Classe	Precision	Recall	F1-Score	Support
0	0.85	0.83	0.84	4673
1	0.44	0.48	0.46	1327
Accuracy	0.75			
Macro Avg	0.64	0.62	0.63	6000
Weighted Avg	0.74	0.76	0.75	6000

Tabella 4.26: Report di classificazione con probabilità di soglia a 0.30

fault risultano più basse rispetto agli altri modelli. Il modello grb è alla base dei modelli mostrati in precedenza, dunque ci si aspetta un comportamento simile e performance meno elevate. Per quanto riguarda l'ottimizzazione del F1-Score sull'insieme di validazione si ottengono rispettivamente:

- 0.20 Probabilità
- 0.48 F1-Score

La soglia calcolata in questo caso risulta ancora più bassa. Il bound sulla probabilità come detto è fondamentale per la classificazione delle istanze di test. In questo caso dunque è necessario che il modello stimi probabilità di default maggiore di 0.21 per associare la classe di default al creditore. Di seguito vengono mostrate le metriche ottenute. Non si ottengono ottime performance per la classe dei default, nonostante la soglia sia particolarmente bassa. Dunque infine viene mostrata la probabilità di default stimata del modello calibrato:

Classe	Precision	Recall	F1-Score	Support
0	0.86	0.74	0.80	4673
1	0.39	0.58	0.47	1327
Accuracy	0.72			
Macro Avg	0.63	0.65	0.63	6000
Weighted Avg	0.75	0.72	0.73	6000

Tabella 4.27: Report di classificazione con probabilità di soglia a 0.20

Parametro	Valore
Valori stimati	[0.57, 2.17]
π_1	0.21
ρ	0.27

Tabella 4.28: Parametri stimati del modello

In conclusione si visualizza la beta ottenuta dai parametri stimati, e il delta rispetto alla probabilità reale:

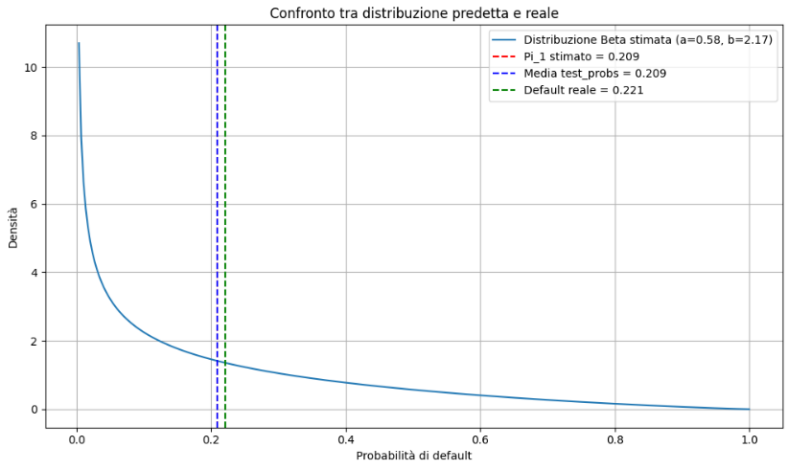

Figura 4.11: Beta con i parametri stimati dal grb come meta-modello finale

4.2 German Dataset

In questa sezione vengono mostrati i risultati ottenuti sul German dataset tramite l'utilizzo dei modelli singoli, nella prima fase del lavoro.

4.2.1 Modelli Singoli

Regressione Logistica

I parametri della regressione logistica sono:

Parametro	Valore	Descrizione
`C`	1.0	Inverso della regolarizzazione L2. Valori più piccoli implicano una regolarizzazione più forte.
`fit_intercept`	True	Indica se calcolare l'intercetta nel modello.
`intercept_scaling`	1	Utile solo quando `solver='liblinear'` e `fit_intercept=True`.
`max_iter`	100	Numero massimo di iterazioni per la convergenza dell'ottimizzatore.
`multi_class`	deprecated	Parametro deprecato; in passato usato per specificare la strategia multiclass (`ovr`, `multinomial`).
`solver`	lbfgs	Algoritmo utilizzato per l'ottimizzazione (adatto a problemi di classificazione con L2).
`tol`	0	Tolleranza per il criterio di arresto dell'ottimizzazione.

Tabella 4.29: Parametri del modello `LogisticRegression` con descrizione

Le metriche ottenute tramite l'utilizzo della regressione logistica sul German Credit dataset sono riportate in tabella ?? Il modello performa molto bene sulla classe dei non

	precision	recall	f1-score	support
0	0.75	0.93	0.83	140
1	0.62	0.27	0.37	60
accuracy			0.73	200
macro avg	0.68	0.60	0.60	200
weighted avg	0.71	0.73	0.69	200

Tabella 4.30: Classification report del modello

default, mostrando valori elevati di precision e recall. Tuttavia, i risultati sulla classe dei default sono decisamente più deboli La recall è molto bassa, indicando una difficoltà del modello nel riconoscere correttamente i soggetti realmente a rischio.

Si prosegue mostrando la curva ROC del modello, utile in questa fase per valutare le capacità predittive complessive. Nell'analisi dedicata ai modelli di Stacking, invece, l'attenzione sarà rivolta principalmente alla recall della classe 1, senza focalizzarsi sulla curva ROC, in quanto questa riflette le performance globali del modello e non quelle specifiche per la classe di interesse.

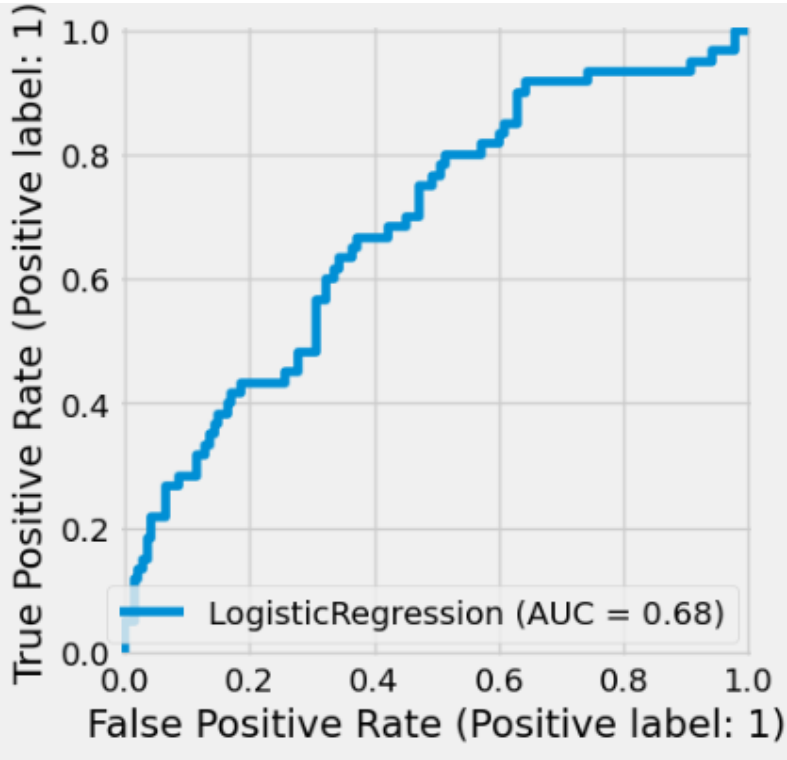

Figura 4.12: curva ROC della Regressione Logistica

Probabilità di default

I valori di probabilità e correlazione ottenuti con il metodo dei momenti sono i seguenti:

Parametro	Valore
Intervallo stimato	[1.92392929, 4.50684811]
π_1	0.30
ρ	0.13457542134114153

Tabella 4.31: Valori stimati del modello

Il Grafico della Beta ottenuta con i parametri stimati è il seguente:

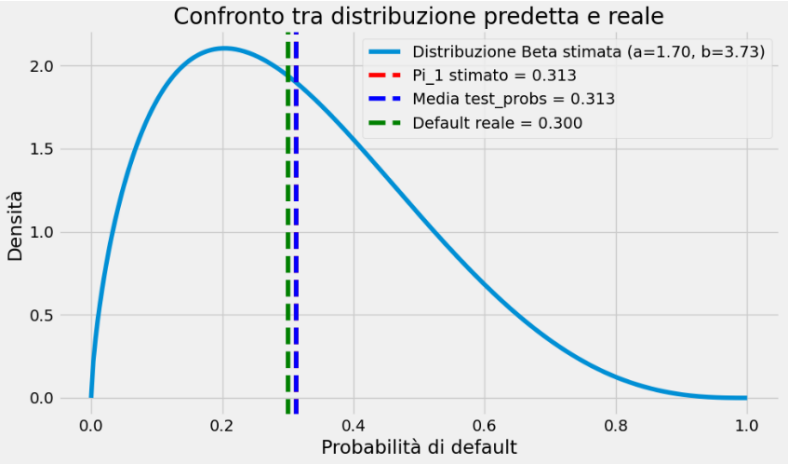

Figura 4.13: Beta della beta con i parametri stimati dalla regressione logistica

Random Forest

I parametri della Random Forest ottenuti facendo gridsearch su f1-score sono:

Parametro	Valore	Descrizione
`bootstrap`	True	Indica se utilizzare il bootstrap durante la costruzione degli alberi.
`ccp_alpha`	0.0	Parametro di complessità per la potatura degli alberi (valore 0 = nessuna potatura).
`criterion`	gini	Funzione per misurare la qualità dello split: indice di Gini.
`max_depth`	6	Profondità massima degli alberi nella foresta.
`max_features`	sqrt	Numero massimo di feature considerate per ogni split (radice quadrata del totale).
`min_impurity_decrease`	0.0	Soglia minima di riduzione dell'impurità per effettuare uno split.
`min_samples_leaf`	1	Numero minimo di campioni richiesti in una foglia.
`min_samples_split`	5	Numero minimo di campioni richiesti per dividere un nodo interno.
`min_weight_fraction_leaf`	0.0	Frazione minima di peso totale richiesta in una foglia.
`n_estimators`	50	Numero di alberi nella foresta.
`oob_score`	False	Se usare i campioni fuori borsa (out-of-bag) per la validazione interna.
`verbose`	0	Livello di verbosità durante il fitting.

Tabella 4.32: Parametri non nulli del modello con descrizione

Le metriche ottenute dal modello sull'insieme di test sono mostrate in tabella ?? La

	precision	recall	f1-score	support
0	0.74	0.97	0.84	140
1	0.75	0.20	0.32	60
accuracy			0.74	200
macro avg	0.74	0.59	0.58	200
weighted avg	0.74	0.74	0.68	200

Tabella 4.33: Classification report del modello

random forest è uno dei modelli che performa meglio in termini di prestazioni complessive, si mostra infatti la curva ROC del modello:

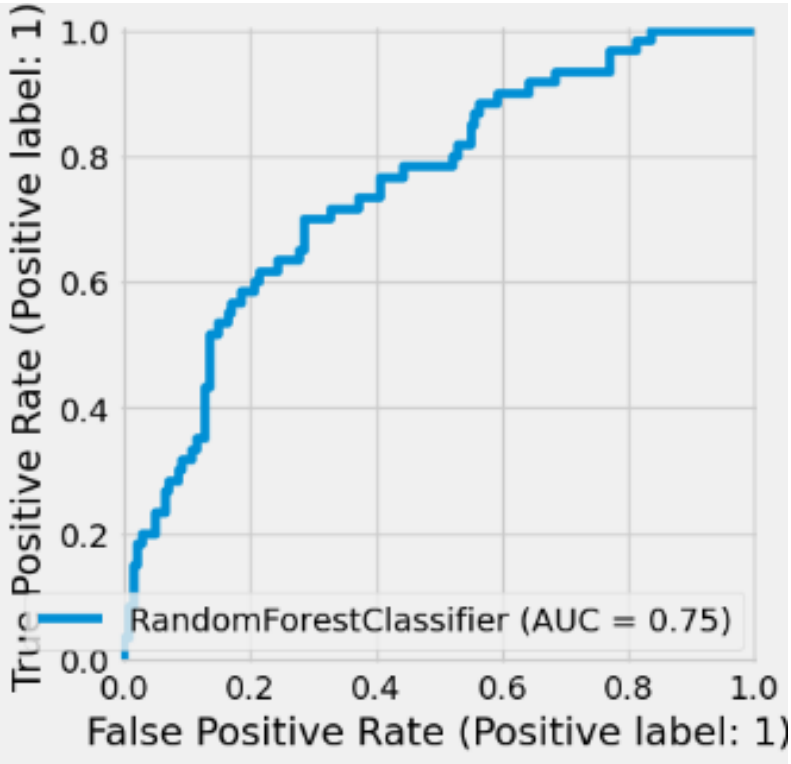

Figura 4.14: curva ROC della Random Forest sul German datsaset

Il valore AUC (Area under curve) di 0.75 è un valore molto buono in questo contesto. La recall della classe 1 rimane però molto bassa. Si vedrà con l'introduzione successiva dei metodi ensemble che si riuscirà ad ottenere un aumento di tale metrica, anche se a discapito delle altre.

Probabilità di default

La probabilità di default e la correlazione ottenuti sono:

Parametro	**Valore**
Intervallo stimato	[0.31132888, 0.65216458]
π_1	0.32
ρ	0.51

Tabella 4.34: Valori stimati del modello

Infine viene mostrata la distribuzione beta ottenuta dai parametri stimati dal metodo momenti.

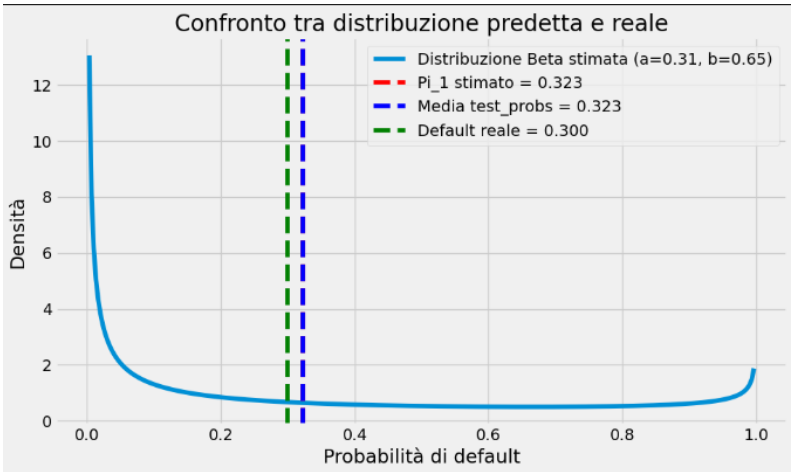

Figura 4.15: Beta simulata con i parametri ottenuti dalla random forest

Adaptive Boosting

I parametri ottenuti per l'Adaptive Boosting sono i seguenti:

Parametro	Valore	Descrizione
`learning_rate`	1	Fattore che regola il contributo di ciascun classificatore debole.
`n_estimators`	200	Numero massimo di classificatori deboli da addestrare.

Tabella 4.35: Parametri significativi del modello `AdaBoostClassifier`

Le performance dell'Adaboost non sono le migliori tra i modelli singoli. La Random forest è il modello che performa meglio fra quelli proposti. Tuttavia la recall raggiunta da quest'ultimo algoritmo è la più alta tra gli algoritmi non ensemble proposti. Il valore ottenuto è di 0.43, sicuramente ancora basso, ma comunuqe più elevato rispetto ai modelli presentati in precedenza. Di seguito vengono mostrate le metriche:

	precision	**recall**	**f1-score**	**support**
0	0.78	0.87	0.82	140
1	0.59	0.43	0.50	60
accuracy			0.74	200
macro avg	0.69	0.65	0.66	200
weighted avg	0.72	0.74	0.73	200

Tabella 4.36: Classification report del modello

Il valore AUC ottenuto è di 0.73, mostrando dunque buone prestazioni complessive del modello, secondo solo alla Random forest. Segue la curva ROC del Modello:

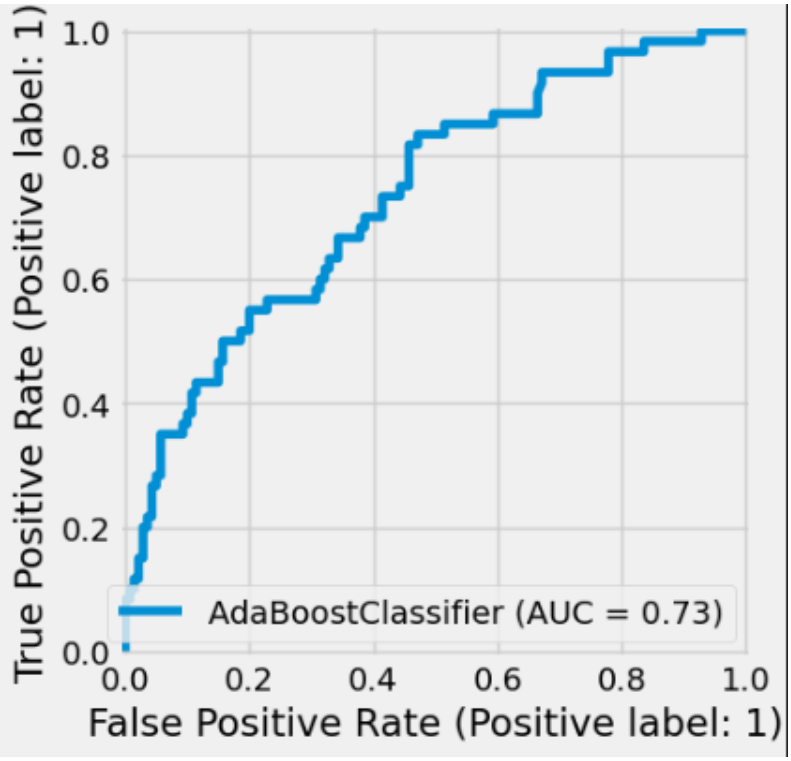

Figura 4.16: curva ROC dell'Adaboost sul German Dataset

Probabilità di default

Probabilità di default e correlazione ottenuti sono mostrati in tabella ?? La distribuzione

Parametro	Valore
Intervallo stimato	[1.11, 2.65]
π_1	0.29
ρ	0.21

Tabella 4.37: Valori stimati del modello

beta, ottenuta con i parametri stimati è mostrata nella figura seguente:

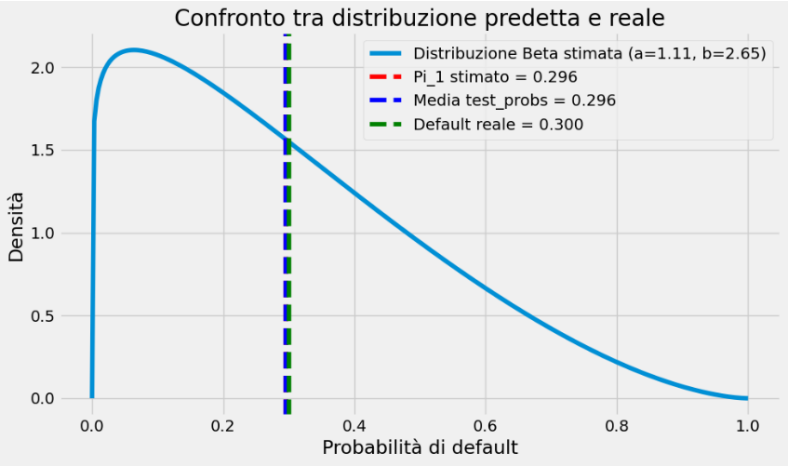

Figura 4.17: Beta ottenuta con i parametri stimati dall'Adaboost

KNN

L'ultimo modello proposto è il KNN (K-Nearest Neighbors Classifier), i parametri ottenuti dopo grid search sono:

Tabella 4.38: Parametri principali del modello `Pipeline (MinMaxScaler + KNN)`

Parametro	**Valore**	**Descrizione**
`scaler`	MinMaxScaler()	Oggetto di scaling che normalizza ogni feature tra 0 e 1.
`scaler__feature_range`	(0,1)	Intervallo target per la normalizzazione.
`scaler__clip`	False	Se True, limita i valori normalizzati all'interno dell'intervallo specificato.
`scaler__copy`	True	Indica se fare una copia dei dati durante la trasformazione.
`classifier`	KNeighborsClassifier()	Classificatore K-NN per il modello finale.
`classifier__n_neighbors`	1	Numero di vicini usati nella classificazione (K=1).
`classifier__metric`	manhattan	Distanza usata per misurare la vicinanza (somma delle differenze assolute).
`classifier__leaf_size`	30	Dimensione del nodo foglia usata nel calcolo dell'albero di ricerca.
`classifier__algorithm`	auto	L'algoritmo di ricerca del vicino più prossimo è scelto automaticamente.

Le metriche ottenute sul test set, dopo aver ottenuto il modello con i parametri ottimizzati sono mostrati in tabella 4.39. Il KNN risulta meno performante rispetto

Tabella 4.39: Classification Report

Classe	**Precision**	**Recall**	**F1-score**	**Support**
0	0.75	0.74	0.74	140
1	0.40	0.42	0.41	60
Accuracy	0.64 (su 200 esempi)			
Macro avg	0.57	0.58	0.58	200
Weighted avg	0.64	0.64	0.64	200

Tabella 4.40: Iperparametri ottimi del KNN

agli altri modelli testati, sebbene la recall della classe 1 non sia particolarmente bassa in confronto agli altri. In questa fase, tuttavia, l'obiettivo principale è individuare un modello che offra buone prestazioni sulla classe 0, in modo da supportare un approccio al credito più rischioso. La ricerca di un modello più cauto, orientato a dare maggiore peso ai default (classe 1), verrà affrontata nella prossima sezione, quando si analizzeranno i risultati dei modelli ensemble. Concludiamo l'analisi dei modelli singoli mostrando le stime ottenute con il Knn sulla probabilità di default e la correlazione tra i creditori. Mostriamo infine il grafico della beta ottenuto con questi parametri

Parametro	Valore
a	1.70
b	3.73
Π_1	0.2957
ρ	0.0352

Tabella 4.41: Stima dei parametri della distribuzione Beta e relativi momenti

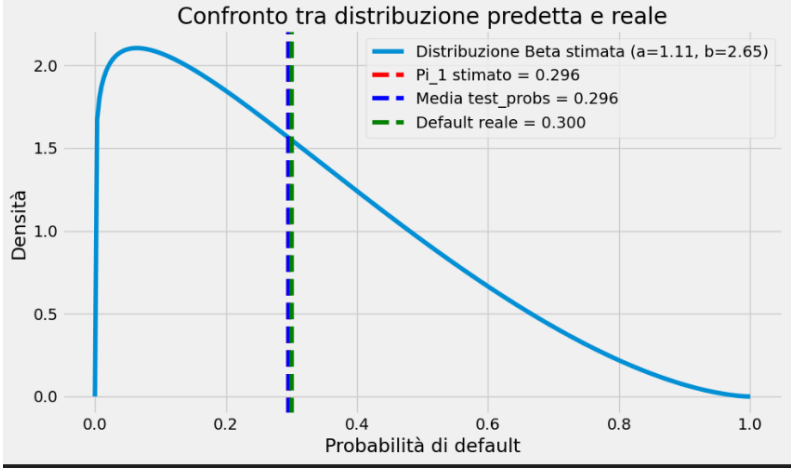

Figura 4.18: Beta ottenuta con i parametri stimati dal KNN

4.2.2 Modelli Ensemble

In questa sezione mostreremo i risultati ottenuti tramite l'utilizzo dei modelli ensemble sul german dataset. In questo esperimento si osserva un miglioramento non indifferente della recall della classe dei default. A differenza delle evidenze precedentemente mostrate i modelli riescono ado ottenere buoni risultati senza l'ausilio di evidenti modifiche della probabilità di soglia. Come già visto per l'UCI CREDIT RISK, per tutti i modelli vengono mostrate le metriche ottenute, la probabilità di default e la beta con i parametri stimati. Prima di cominciare con l'analisi dei risultati si ricorda che il valore della probabilità di default osservato nel german credit dataset è 0.3. Infatti su una popolazione globale di 1000 individui solo 300 andranno in default.

Regressione Logistica

I risultati ottenuti senza validazione sulla soglia di probabilità. Utilizzando dunque 0.5 come bound di probabilità tra le due classi risultano:

La *recall* ottenuta per la classe positiva non risulta elevata, indicando come già osservato negli esperimenti precedenti la debolezza della regressione logistica rispetto agli altri modelli utilizzati in termini di individuazione dei casi di default. Tuttavia si nota un miglioramento in termini di recall rispetto all'utilizzo della regressione logistica come

Classe	Precision	Recall	F1-Score	Support
0	0.76	0.85	0.80	140
1	0.52	0.38	0.44	60
Accuracy	0.71			
Macro Avg	0.64	0.62	0.62	200
Weighted Avg	0.69	0.71	0.70	200

Tabella 4.42: Classification report della regressione logistica

modello singolo.

Balanced Accuracy

La probabilità di soglia ottenuta massimizzando la balanced Accuracy sul set di validazione, e lo score associato risultano:

- 0.34 Probabilità
- 0.71 Balanced Accuracy

Le metriche ottenute con questa soglia risultano: Diminuendo la probabilità di soglia la

Classe	Precision	Recall	F1-Score	Support
0	0.78	0.75	0.76	140
1	0.46	0.50	0.48	60
Accuracy	0.68			
Macro Avg	0.62	0.62	0.62	200
Weighted Avg	0.68	0.68	0.68	200

Tabella 4.43: Classification report del modello con probabilità di soglia a 0.34

recall della classe 1 aumenta, è più facile per il modello, rispetto alle condizioni standard (probabilità di soglia di 0.50), classificare un'istanza come default.

Ottimizzando F1-score sull'insieme di validazione la probabilità di soglia ottenuta rimane invariata al caso dell'ottimizzazione della Balanced Accuracy. Tale fenomeno accadrà frequentemente nel corso dell'analisi. Dunque non vengono mostrate le performance ottenute poiché uguali a quelle mostrate in tabella 4.43. Proseguiamo l'analisi mostrando la probavilità di default e la correlazione stimata dal modello non calibrato.

Come si vede il modello senza necessità di calibrazione riesce quasi perfettamente a stimare la probabilità marginale di default.

Parametro	Valore
α	0.44
β	1.08
π_1	0.29
ρ	0.40

Tabella 4.44: Valori stimati di probabilità e parametri

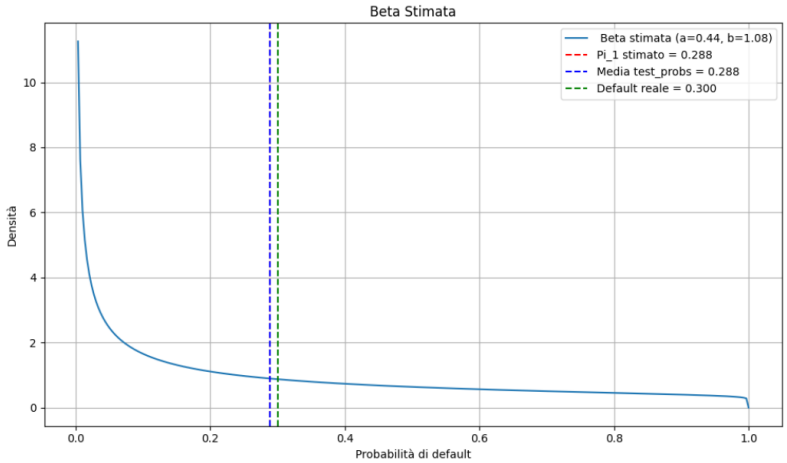

Figura 4.19: Beta rl con parametri stimati

Xgboost

Si procede con l'analisi dei risultati, utilizzando come metamodello finale l'Xgboost, rispetto alla regressione logistica ci si aspetta maggiore precisione nella discriminazione tra le classi e minore capacità nella stima della probabilità di default. I parametri ottimi risultano:

Parametro	**Descrizione**
`eval_metric='logloss'`	Funzione di valutazione: log loss per classificazione binaria
`learning_rate=0.0298`	Tasso di apprendimento
`max_depth=3`	Profondità massima degli alberi.
`n_estimators=134`	Numero di alberi (rounds di boosting) da costruire
`n_jobs=-1`	Utilizza tutti i core disponibili per l'addestramento parallelo

Tabella 4.45: Parametri valorizzati del modello `XGBClassifier` e loro descrizione

Si continua mostrando le metriche ottenute senza validazione sulla probabilità di soglia.

Classe	**Precision**	**Recall**	**F1-Score**	**Support**
0	0.75	0.61	0.69	140
1	0.41	0.65	0.51	60
Accuracy	0.68			
Macro Avg	0.60	0.59	0.59	200
Weighted Avg	0.66	0.68	0.67	200

Tabella 4.46: Classification report del modello con soglia decisionale standard

Come riportato in tabella 4.47 il valore della recall della classe 1, (obiettivo principale del lavoro), senza alcuna validazione sulla soglia risulta 0.65. Il modello dunque riesce ad assegnare correttamente il 65% dei clienti in default alla classe corretta. Ottenendo buone performance in termini di recall con la probabilità di soglia standardl, si è deciso di osservare il comportamento del modello abbassando la soglia decisionale, ma restando nell'intorno di 0.50. Con la probabilità di soglia uguale a 0.42 ad esempio si ottengono le seguenti metriche:

Classe	Precision	Recall	F1-Score	Support
0	0.81	0.55	0.65	140
1	0.40	0.70	0.51	60
Accuracy	0.60			
Macro Avg	0.60	0.62	0.58	200
Weighted Avg	0.66	0.56	0.58	200

Tabella 4.47: Classification report del modello con soglia decisionale fissata a 0.42

Come si evince dalla tabella, con questa probabilità di soglia, otteniamo una recall molto alta per il caso studio di riferimento. Si continua dunque visualizzando la stima della probabilità e della correlazione per il modello calibrato, ed il grafico della beta ottenuta con i parametri stimati.

Parametro	Valore
Intervallo	[0.94, 1.98]
π_1	0.32
ρ	0.25

Tabella 4.48: Valori stimati per intervallo, π_1 e ρ

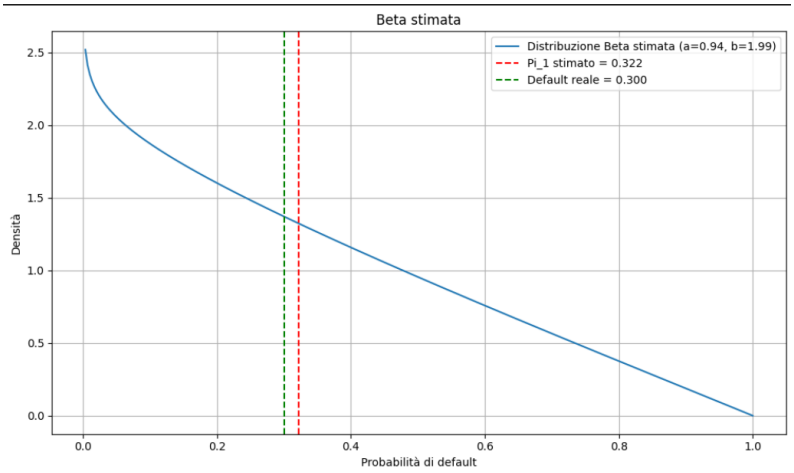

Figura 4.20: Beta Xgboost con parametri stimati

Random Forest

I parametri ottimi ottenuti utilizzado la random forest come meta modello finale sono riportati in tabella 4.49. Come evidenziato anche dagli altri modelli analizzati, la strut-

Parametro	**Valore**
`class_weight`	`'balanced'`
`max_depth`	7
`min_samples_split`	8
`n_estimators`	261
`n_jobs`	-1
`random_state`	42

Tabella 4.49: Parametri del modello `RandomForestClassifier`

tura *ensemble* applicata al *German Dataset* mostra buone performance in termini di *recall* sulla classe 1, anche senza l'ottimizzazione della soglia decisionale. I risultati ottenuti con la random forest senza alcuna ottimizzazione sono riportati in tabella 4.50. Si procede dunque cone fatto per il modello precedente all'analisi empirica del comportamento del modello abbassando non eccessivamente la probabilità di soglia standard. Si mostrano dunque le performance ottenute ponendo la probabilità di soglia a 0.42, come fatto per Xgboost.

Classe	Precision	Recall	F1-Score	Support
0	0.78	0.65	0.71	140
1	0.41	0.57	0.48	60
Accuracy	0.62			
Macro Avg	0.59	0.61	0.59	200
Weighted Avg	0.67	0.62	0.64	200

Tabella 4.50: Classification report del modello con soglia di probabilità standard

	precision	recall	f1-score	support
0	0.81	0.60	0.68	140
1	0.41	0.67	0.51	60
accuracy			0.70	200
macro avg	0.61	0.63	0.60	200
weighted avg	0.69	0.61	0.63	200

Tabella 4.51: Classification report del modello con soglia di probabilità fissata a 0.42

Come si evince dalle metriche mostrate in tabella 4.51 il modello migliora in termini di recall della classe 1 mantenendo però stabili le altre metriche. Proseguiamo l'analisi con la visualizzazione della probabilità e della correlazione stimate.

Anche in questo caso si nota un delta rispetto ala valore reale, meglio visibile nel grafico della beta stimata.

Parametro	**Valore**
Intervallo	[0.82, 1.60]
Π_1	0.34
ρ	0.29

Tabella 4.52: Valori statistici calcolati

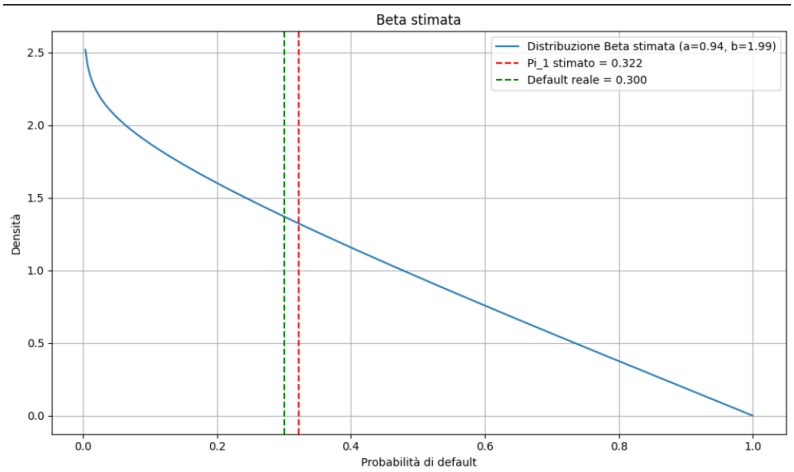

Figura 4.21: Beta Random Forest con i parametri stimati

LGBM

I parametri ottimi del modello risultano:

Parametro	Descrizione
`class_weight`	Bilanciamento automatico del peso delle classi (`balanced`)
`learning_rate`	Tasso di apprendimento: 0.023
`max_depth`	Profondità massima degli alberi: 3
`n_estimators`	Numero di alberi (stimatori): 122
`n_jobs`	Numero di core usati: tutti disponibili (`-1`)
`random_state`	Stato del generatore casuale: 42 (riproducibilità)

Tabella 4.53: Parametri del modello `LGBMClassifier`

Si prosegue allo stesso modo dei modelli precedenti. Si inzia dunque mostrando i risultati ottenuti senza ottimizzazione sulla soglia di probabilità.

	precision	**recall**	**f1-score**	**support**
0	0.77	0.61	0.69	140
1	0.39	0.58	0.47	60
accuracy			0.60	200
macro avg	0.58	0.60	0.58	200
weighted avg	0.66	0.62	0.62	200

Tabella 4.54: Classification report del modello cpn soglia di probabilità standard

Anche per LGBM la recall della classe 1 (default) risulta vicino a 0.60, dunque come per i modelli precedenti si valuta il comportamento del modello nell'intorno della probabilità di soglia standard. Si osserva di seguito il comportamento del modello, ponendo la probabilità di soglia a 0.45 in tabella 4.55

I risultati che si ottengono sono davvero molto simili a quelli ottenuti per Xgboost e Random Forest.

Inoltre per continuità illustrativa si procede alla visualizzazione in tabella 4.56 dei risultati ottenuti utilizzando la probabilità di soglia uguale a 0.42.

	precision	**recall**	**f1-score**	**support**
0	0.78	0.56	0.66	140
1	0.38	0.62	0.48	60
accuracy			0.58	200
macro avg	0.58	0.60	0.57	200
weighted avg	0.66	0.58	0.60	200

Tabella 4.55: Classification report del modello con probabilità di soglia a 0.45

	precision	**recall**	**f1-score**	**support**
0	0.80	0.59	0.67	140
1	0.40	0.65	0.50	60
accuracy			0.60	200
macro avg	0.60	0.62	0.59	200
weighted avg	0.68	0.60	0.62	200

Tabella 4.56: Classification report del modello con probabilità di soglia a 0.42

La probabilità di default e i parametri della beta stimati dal modello sono visibili in tabella 4.57.

Parametro	**Valore**
Intervallo stimato	[0.81 1.82]
π_1	0.31
ρ	0.27

Tabella 4.57: Valori stimati del modello

Si nota ancora una volta come il modello calibrato riesca stimare bene la probabilità, segua il grafico.

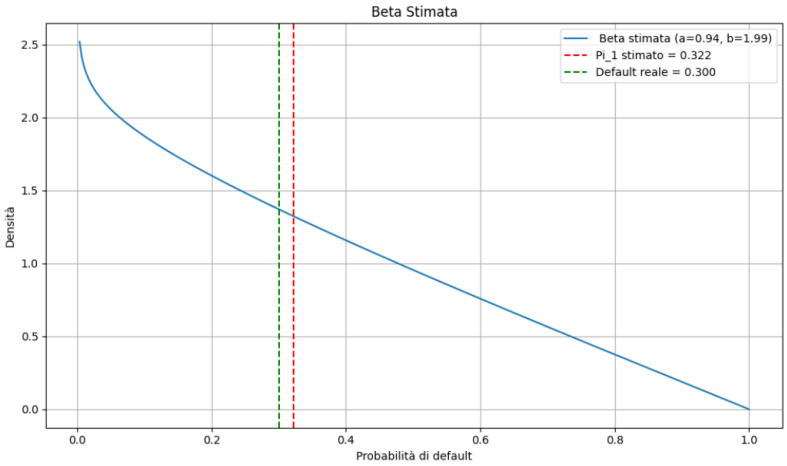

Figura 4.22: Beta lightgbm con parametri stimati

Gradient Boosting

L'ultimo modello è il gradient bossting, i parametri ottenuti con il metodo optuna sono:

Parametro	Valore / Descrizione
`learning_rate`	0.07
`max_depth`	3
`n_estimators`	264
`random_state`	42 (per garantire la riproducibilità)

Tabella 4.58: Parametri del modello `GradientBoostingClassifier`

Le metriche ottenute con la probabilità di soglia standard (0.50) sono visibili in tabella 4.59.

I valori della recall non sono elevati come nei modelli precedenti, dunque à necessario

	precision	recall	f1-score	support
0	0.77	0.84	0.80	140
1	0.53	0.43	0.48	60
accuracy			0.71	200
macro avg	0.65	0.63	0.64	200
weighted avg	0.70	0.71	0.71	200

Tabella 4.59: Classification report del modello

mostrare i risultati ottenuti ottimizzando sulla balanced accuracy e f1-score. La probabilità di soglia ottenuta è uguale in entrambi i casi.

- Probabilità di soglia 0.3
- Balanced Accuracy 0.68
- F1-Score 0.49

Le metriche ottenute con questa soglia sono:

	precision	**recall**	**f1-score**	**support**
0	0.78	0.67	0.72	140
1	0.42	0.57	0.49	60
accuracy			0.64	200
macro avg	0.60	0.60	0.60	200
weighted avg	0.68	0.64	0.65	200

Tabella 4.60: Classification report del modello

I valori ottenuti dal modello calibrato sono i seguenti:

Parametro	**Valore**
Intervallo stimato	[31.29440926, 54.26829886]
π_1	0.32
ρ	0.23

Tabella 4.61: Valori stimati del modello

Ancora una volta il modello calibrato riesce a stimare correttamente la probabilità di default. Di seguito viene mostrato l'ultimo grafico in cui si può visualizzare la beta stimata.

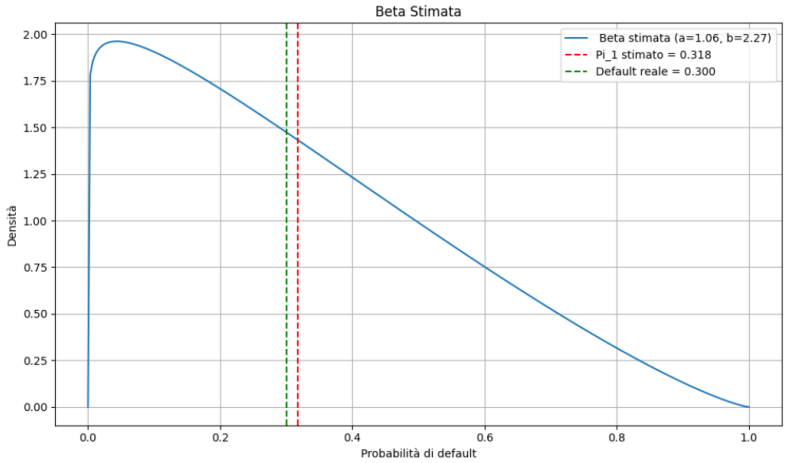

Figura 4.23: Beta Grb con parametri stimati

Chapter 5

Conclusioni

L'obiettivo di questo lavoro è stato valutare le performance di una nuova architettura basata su modelli ensemble nel contesto del rischio di credito, con l'intento di stimare la probabilità di default e ottenere buone capacità predittive rispetto alla classe dei soggetti in default. Sui dati del German Credit Dataset, l'approccio proposto ha mostrato un miglioramento sostanziale rispetto all'impiego dei singoli modelli. In particolare, utilizzando XGBoost, LightGBM e Random Forest, si è ottenuto una recall della classe "default" pari a 0,65 senza alcuna ottimizzazione della soglia di decisione. Abbassando lievemente tale soglia (rispettivamente a 0,45 e 0,42), la recall si è spinta fino a valori prossimi a 0,70, risultati sorprendentemente elevati, specialmente in un contesto caratterizzato da forte sbilanciamento e complessità di previsione come questo. Le prestazioni più elevate sono state ottenute dai modelli basati su tecniche di boosting. In particolare, LightGBM e XGBoost hanno superato in capacità predittiva il Gradient Boosting "di baseâ da cui derivano, grazie a ottimizzazioni a livello di costruzione degli alberi e gestione dei dati che ne aumentano l'efficienza. Applicando lo stesso schema sul dataset "UCI Credit Risk", la recall relativa alla classe dei default non ha raggiunto i livelli osservati sul German Credit. Infatti è stato necessario abbassare la soglia di probabilità dei modelli, ottenendo tuttavia performance di recall complessivamente superiori rispetto all'impiego dei singoli classificatori. Per quanto riguarda la probabilità di default, l'applicazione del metodo dei momenti ha permesso su entrambi i dataset di generare stime corrette per tutte le configurazioni di modello testate.

Tra i metamodelli, la strategia di stacking con XGBoost si è rivelata dunque la più solida, combinando buone prestazioni in termini di recall della classe dei default, soprattutto per il German Credit Dataset con una calibrazione delle probabilità di default di ottimo livello.

Bibliography

[1] Hansheng Wang Danyang Huang, Jing Zhou. Rfms method for credit scoring based on bank card transaction data. *Statistica Sinica*, 3I:2903–2919, 2019.

[2] Thomas Taimre Radislav Vaisman Dirk P.Kroese, Zdravko I. Botev. Data science and machine learning. pages 309–313, 2020.

[3] Obafemi Olawale Ogunsan Isaac. Feature engineering in credit risk models: key to optimization. *ResearchGate preprint*, 2024.

[4] Elisa Luciano Patrizia Semeraro, Edoardo Fadda. Machine learning techniques in joint default assessment. 2024.

[5] John C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In Peter L. Smola, Alex J.and Bartlett, Bernhard Schölkopf, and Dale Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 1999.

[6] D.K Malhotra Rashmi Malhotra a. Evaluating consumer loans using neural networks. *Omega*, 3I:83–96, 2003.

[7] Carlos Guestrin Tianqi Chen. Xgboost: A scalable tree boosting system. 2016.

[8] Kong Yue b Wang Baoa, Ning Lianjua. Integration of unsupervised and supervised machine learning algorithms for credit risk assessment. *Expert Systems with Applications*, 128:301–315, 2019.

[9] Bianca Zadrozny and Charles Elkan. Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 609–616, Williamstown, MA, 2001. Morgan Kaufmann Publishers.

.1 Appendice: Struttura del codice Python per la creazione dei modelli ensemble.

Listing 1: Script completo utilizzato per allenare, ottimizzare e calibrare i modelli

```
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import joblib
import optuna
from scipy.optimize import root

from sklearn.calibration import CalibratedClassifierCV
from sklearn.model_selection import (
    train_test_split, GridSearchCV,
    StratifiedKFold, cross_val_score, cross_val_predict
)
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline

from sklearn.metrics import (
    classification_report, confusion_matrix,
    precision_recall_curve, balanced_accuracy_score,
    precision_score, recall_score, f1_score,
    silhouette_samples, silhouette_score
)

from sklearn.cluster import KMeans
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import StackingClassifier, RandomForestClassifier

from imblearn.over_sampling import ADASYN
from imblearn.combine import SMOTETomek
from imblearn.ensemble import BalancedRandomForestClassifier

from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
import lightgbm as lgb
from catboost import CatBoostClassifier
```

```
from sklearn.metrics import precision_score, recall_score, f1_score
from numpy.linalg import norm
# Caricamento dati
def get_data(path):
    df = pd.read_csv(path)
    X = df.drop('default', axis=1)
    y = df['default']
    return X, y

X, y = get_data('./data/UCI_Credit_Card.csv')

# Feature engineering
def add_behavioral_features(X):
    bill_cols = ['BILL_AMT' + str(i) for i in range(1,7)]
    pay_cols  = ['PAY_AMT'  + str(i) for i in range(1,7)]
    pay_status_cols = ['PAY_' + str(i) for i in range(3,7)]

    X['MAX_UTIL_RATIO']  = X[bill_cols].max(axis=1) / X['LIMIT_BAL']
    X['MEAN_UTIL_RATIO'] = X[bill_cols].mean(axis=1) / X['LIMIT_BAL']
    X['STD_BILL']        = X[bill_cols].std(axis=1)
    X['TOTAL_PAYMENT']   = X[pay_cols].sum(axis=1)
    X['AVG_PAY_RATIO']   = X[pay_cols].mean(axis=1) / (X[bill_cols].mean(axi
    X['PAY_TREND']       = X['PAY_6'] - X['PAY_3']
    X['LATE_COUNT']      = X[pay_status_cols].apply(lambda row: (row >= 1).s
    return X

X = add_behavioral_features(X)

# One-hot encoding e pulizia
dummy_cols = ['SEX', 'MARRIAGE', 'EDUCATION']
X = pd.get_dummies(X, columns=dummy_cols, drop_first=True)
X.replace([np.inf, -np.inf], np.nan, inplace=True)
X.dropna(inplace=True)
y = y.loc[X.index]
colonne_originali = X.columns.tolist()

# Split train/test
X_train_full, X_test, y_train_full, y_test = train_test_split(
    X, y, stratify=y, test_size=0.2, random_state=42
)

# Clustering sui default per feature di distanza
scaler_tmp = MinMaxScaler()
X_train_tmp = pd.DataFrame(
```

```
    scaler_tmp.fit_transform(X_train_full[colonne_originali]),
    columns=colonne_originali, index=X_train_full.index
)
solo_default_scaled = X_train_tmp[y_train_full == 1]
kmeans = KMeans(n_clusters=2, random_state=42).fit(solo_default_scaled)



def get_cluster_distances(X_df, labels, centers):
    return [
        norm(X_df.iloc[i].values - centers[label])
        for i, label in enumerate(labels)
    ]

train_labels = kmeans.predict(X_train_tmp)
X_train_full['DEFAULT_CLUSTER']    = train_labels
X_train_full['CLUSTER_DISTANCE'] = get_cluster_distances(
    X_train_tmp, train_labels, kmeans.cluster_centers_
)

X_test_tmp = pd.DataFrame(
    scaler_tmp.transform(X_test[colonne_originali]),
    columns=colonne_originali, index=X_test.index
)
test_labels = kmeans.predict(X_test_tmp)
X_test['DEFAULT_CLUSTER']    = test_labels
X_test['CLUSTER_DISTANCE'] = get_cluster_distances(
    X_test_tmp, test_labels, kmeans.cluster_centers_
)

sil_vals = silhouette_samples(solo_default_scaled, kmeans.labels_)
silhouette_avg = np.mean(sil_vals)

fig, ax = plt.subplots(figsize=(8,5))
y_lower = 10
for i in range(kmeans.n_clusters):
    vals = sil_vals[kmeans.labels_ == i]
    vals.sort()
    size = vals.shape[0]
    y_upper = y_lower + size
    ax.fill_betweenx(np.arange(y_lower, y_upper), 0, vals, alpha=0.7)
    ax.text(-0.05, y_lower + 0.5*size, f'Cluster {i}')
    y_lower = y_upper + 10
ax.axvline(silhouette_avg, color='red', linestyle='—',
```

```
label=f'Media silhouette: {silhouette_avg:. 2f}')
ax. set_xlabel('Valore del coefficiente di silhouette')
ax. set_ylabel('Cluster')
ax. set_title('Silhouette Plot per i clienti in default')
ax. legend()
plt. tight_layout(); plt. show()

## Riscalo tutte le variabili
X_train_base, X_val, y_train_base, y_val = train_test_split(
    X_train_full, y_train_full, stratify=y_train_full,
    test_size=0.25, random_state=42
)


X_train_base_df = X_train_base. copy()
X_val_df = X_val. copy()
X_test_df = X_test. copy()

scaler = MinMaxScaler()
X_train_base_scaled = pd. DataFrame(
    scaler. fit_transform(X_train_base),
    columns=X_train_base. columns, index=X_train_base. index
)
X_val_scaled = pd. DataFrame(
    scaler. transform(X_val), columns=X_val. columns, index=X_val. index
)
X_test_scaled = pd. DataFrame(
    scaler. transform(X_test), columns=X_test. columns, index=X_test. index
)

smote_tomek = SMOTETomek(sampling_strategy=0.5, random_state=42)
X_train_bal, y_train_bal = smote_tomek. fit_resample(
    X_train_base_scaled, y_train_base
)

# Funzione di valutazione
from sklearn. metrics import f1_score

def evaluate_model(m, Xd, yd):
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
    return cross_val_score(m, Xd, yd, scoring='f1', cv=skf, n_jobs=-1). mean()

# . .. Ottimizzazione e training dei modelli base . ..
```

```
# Ottimizzazione per XGBoost
def tune_xgb(X, y):
    def objective(trial):
        params = {
            'n_estimators': trial.suggest_int('n_estimators', 100,500),
            'max_depth': trial.suggest_int('max_depth', 3,10),
            'learning_rate': trial.suggest_float('learning_rate', 0.01,0.3,
            'subsample': trial.suggest_float('subsample', 0.6, 1.0),
            'colsample_bytree': trial.suggest_float('colsample_bytree', 0.6,
            'scale_pos_weight': trial.suggest_float('scale_pos_weight', 1.0,
            'use_label_encoder': False,
            'eval_metric': 'logloss',
            'random_state': 42
        }
        model = XGBClassifier(**params)
        return evaluate_model(model, X, y)

    study = optuna.create_study(direction='maximize')
    study.optimize(objective, n_trials=50)

    best_params = study.best_trial.params
    best_model = XGBClassifier(**best_params)
    best_model.fit(X, y)
    return best_model


def tune_catboost(X, y):
    def objective(trial):
        params = {
            'iterations': trial.suggest_int('iterations', 100,500),
            'depth': trial.suggest_int('depth', 4,10),
            'learning_rate': trial.suggest_float('learning_rate', 0.01,0.3,
            'l2_leaf_reg': trial.suggest_float('l2_leaf_reg', 1.0, 10.0),
            'random_state': 42,
            'verbose': 0
        }
        model = CatBoostClassifier(**params)
        return evaluate_model(model, X, y)

    study = optuna.create_study(direction='maximize')
    study.optimize(objective, n_trials=50)

    best_params = study.best_trial.params
    best_model = CatBoostClassifier(**best_params)
```

```
    best_model.fit(X, y)
    return best_model


def tune_lgb(X, y):
    def objective(trial):
        params = {
            'n_estimators': trial.suggest_int('n_estimators', 100,500),
            'max_depth': trial.suggest_int('max_depth', 3,10),
            'learning_rate': trial.suggest_float('learning_rate', 0.01,0.3,
            'num_leaves': trial.suggest_int('num_leaves', 15,150),
            'subsample': trial.suggest_float('subsample', 0.6, 1.0),
            'colsample_bytree': trial.suggest_float('colsample_bytree', 0.6,
            'random_state': 42
        }
        model = lgb.LGBMClassifier(**params)
        return evaluate_model(model, X, y)

    study = optuna.create_study(direction='maximize')
    study.optimize(objective, n_trials=50)

    best_params = study.best_trial.params
    best_model = lgb.LGBMClassifier(**best_params)
    best_model.fit(X, y)
    return best_model


# Ottimizzazione per Balanced Random Forest
def tune_brf(X, y):
    def objective(trial):
        params = {
            'n_estimators': trial.suggest_int('n_estimators', 100,300),
            'max_depth': trial.suggest_int('max_depth', 3,15),
            'max_features': trial.suggest_categorical('max_features', ['sqrt'
            'random_state': 42
        }
        model = BalancedRandomForestClassifier(**params)
        return evaluate_model(model, X, y)

    study = optuna.create_study(direction='maximize')
    study.optimize(objective, n_trials=50)

    best_params = study.best_trial.params
    best_model = BalancedRandomForestClassifier(**best_params)
```

```
    best_model.fit(X, y)
    return best_model
model_xgb = tune_xgb(X_train_bal_df, y_train_bal)
model_cat = tune_catboost(X_train_bal_df, y_train_bal)
model_lgb = tune_lgb(X_train_bal_df, y_train_bal)
model_brf = tune_brf(X_train_bal_df, y_train_bal)


# Predizioni out-of-fold e stacking


preds_xgb = cross_val_predict(model_xgb, X_train_bal_df, y_train_bal, cv=5, me
preds_cat = cross_val_predict(model_cat, X_train_bal_df, y_train_bal, cv=5, me
preds_brf = cross_val_predict(model_brf, X_train_bal_df, y_train_bal, cv=5, me
preds_light=cross_val_predict(model_lgb, X_train_bal_df, y_train_bal, cv=5, me
#Validazione
val_xgb = model_xgb.predict_proba(X_val_scaled_df)[:, 1]
val_cat = model_cat.predict_proba(X_val_scaled_df)[:, 1]
val_brf = model_brf.predict_proba(X_val_scaled_df)[:, 1]
val_lbgm=model_lgb.predict_proba(X_val_scaled_df)[:, 1]
#  Stack delle predizioni dei modelli base ===
meta_preds_val = np.column_stack([val_xgb, val_cat, val_brf, val_lbgm])

# . Costruisci il metadataset di validazione
meta_X_val = np.hstack([meta_preds_val, X_val_scaled_df.values])

#Test
test_xgb = model_xgb.predict_proba(X_test_scaled_df)[:, 1]
test_cat = model_cat.predict_proba(X_test_scaled_df)[:, 1]
test_brf = model_brf.predict_proba(X_test_scaled_df)[:, 1]
test_lbgm=model_lgb.predict_proba(X_test_scaled_df)[:, 1]
# === 3. Stack delle predizioni dei modelli base ===
meta_preds_test = np.column_stack([test_xgb, test_cat, test_brf, test_lbgm])

# === 4. Costruisci il metadataset di test ===

meta_X_test = np.hstack([meta_preds_test, X_test_scaled_df.values])
# Tuning e training di tutti i meta modelli
def objective(trial):
    model = RandomForestClassifier(
        n_estimators=trial.suggest_int("n_estimators", 100,300),
        max_depth=trial.suggest_int("max_depth", 5,15),
        min_samples_split=trial.suggest_int("min_samples_split", 2,10),
        class_weight="balanced",
```

```
        random_state=42,
        n_jobs=-1
    )

    preds = cross_val_predict(model, meta_X, meta_y, cv=5, method='predict', n
    return f1_score(meta_y, preds)

study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=50)

print("Migliori parametri final estimator:", study.best_params)
print("F1 score migliore:", study.best_value)
best_params_rf = study.best_params

final_model_rf = RandomForestClassifier(
    n_estimators=best_params_rf['n_estimators'],
    max_depth=best_params_rf['max_depth'],
    min_samples_split=best_params_rf['min_samples_split'],
    class_weight="balanced",
    random_state=42,
    n_jobs=-1
)

final_model_rf.fit(meta_X, meta_y)
#calibrazione dei modelli
model_calibrated_rf = CalibratedClassifierCV(estimator=final_model_rf , method
model_calibrated_rf.fit(meta_X, meta_y)
... #uguale per xgboost, lightgbm e Gradient boosting



# === Ottimizzazione soglia su validation ===


val_probs= final_model_rf.predict_proba(meta_X_val)[:, 1]  # meta_X_val = val

# Definizione delle soglie da testare
soglie = np.linspace(0.3, 0.7, 41)  # da 0.30 a 0.70 a passi di 0.01

miglior_soglia = 0.5
miglior_score = 0

for soglia in soglie:
    pred = (val_probs >= soglia).astype(int)
```

```
    score = balanced_accuracy_score(y_val, pred)
    if score > miglior_score:
        miglior_score = score
        miglior_soglia = soglia

# === Performance ===

test_probs_rf = final_model_rf.predict_proba(meta_X_test)[:, 1]
test_preds_rf = (test_probs_rf >= 0.50).astype(int)
print("F1 Score:", f1_score(y_test, test_preds_rf))
print("\nClassification Report:\n", classification_report(y_test, test_preds_r
test_probs_rf = final_model_rf.predict_proba(meta_X_test)[:, 1]
test_preds_rf = (test_probs_rf >= miglior_soglia).astype(int)
print("F1 Score:", f1_score(y_test, test_preds_rf))
print("\nClassification Report:\n", classification_report(y_test, test_preds_r

# Stima della prob senza calibrazione
from scipy.optimize import root
#MM per xu_fe
p= (sum(test_probs_rf))/len(test_probs_rf)
mu_2 = (sum((test_probs_rf)**2))/len(test_probs_rf)
def MM(X):

    x, y = X
    # all RHS have to be 0
    f = [x/(x+y)-p,
         (x*(x+1))/((x+y)*(x+y+1))-mu_2]

    return f

sol_blg_xuf = root(MM, [1.0, 1.0])
print(sol_blg_xuf.x)
print('Pi_1: ', pi(sol_blg_xuf.x[0], sol_blg_xuf.x[1]))#calcolo p_1 e rho co
print('Rho: ', rho(sol_blg_xuf.x[0], sol_blg_xuf.x[1]))
from sklearn.metrics import precision_score, recall_score, f1_score

val_probs = final_model_rf.predict_proba(meta_X_val)[:, 1]

soglie = np.linspace(0.1, 0.9, 81)
miglior_soglia = 0.5
miglior_score = 0

for soglia in soglie:
    pred = (val_probs >= soglia).astype(int)
```

```
    precision = precision_score(y_val, pred)
    recall = recall_score(y_val, pred)
    score = 2 * (precision * recall) / (precision + recall + 1e-8)
# F1 Score
    if score > miglior_score:
        miglior_score = score
        miglior_soglia = soglia

print(f"Soglia ottimale (F1): {miglior_soglia:.2f},
F1 Score: {miglior_score:.4f}")
test_probs = final_model_rf.predict_proba(meta_X_test)[:, 1]
test_preds = (test_probs >= miglior_soglia).astype(int)

print("F1 Score:", f1_score(y_test, test_preds))
print("\nClassification Report:\n", classification_report(y_test, test_preds))

Prob stimata con il modello calibrato
test_probs = model_calibrated_rf.predict_proba(meta_X_test)[:, 1]
test_preds = (test_probs >= 0.5).astype(int)
print("F1 Score:", f1_score(y_test, test_preds))
print("\nClassification Report:\n", classification_report(y_test, test_preds))
p= (sum(test_probs))/len(test_probs)
mu_2 = (sum((test_probs)**2))/len(test_probs)
def MM(X):

    x, y = X
    # all RHS have to be 0
    f = [x/(x+y)-p,
         (x*(x+1))/((x+y)*(x+y+1))-mu_2]

    return f
sol_blg_xuf = root(MM, [1.0, 1.0])
print(sol_blg_xuf.x)
print('Pi_1: ', pi(sol_blg_xuf.x[0], sol_blg_xuf.x[1]))
#calcolo p_1 e rho con x(0)=a e x(1)=b output di f
print('Rho: ', rho(sol_blg_xuf.x[0], sol_blg_xuf.x[1]))
```