Alma Mater Studiorum · Università di Bologna

CAMPUS DI CESENA
Dipartimento di Informatica - Scienza e Ingegneria
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

Progettazione e implementazione di un sistema di predizione dell'abbandono degli utenti in una catena di palestre

Tesi di Laurea in
Data Mining & Machine Learning

***Relatore*:**
Prof. Matteo Golfarelli

***Presentata da*:**
Martina Magnani

Seconda Sessione di Laurea
Anno Accademico 2019/2020

Ai miei genitori, a tutte le energie, le cure e l'amore che mi hanno donato;

A mio fratello Leonardo, al nostro cresce insieme e all'esserci l'uno per l'altra;

A mia cugina Marianna, sorella e amica, al "poter mettere una mano sul fuoco";

A mio cugino Federico, per le risate e le parole di sostegno che non dimenticherò mai;

Alle mie zie, Nadia e Fulvia, per tutto l'affetto;

Alla mia stella Silvia, per la presenza, la lealtà e il bene;

A Jessica, ai nostri sogni, al faro luminoso che accende quando sale la nebbia;

A Silvia, a questa nuova preziosa amicizia, al nostro essere diverse ma simili rispetto a tutto ciò che conta davvero.

A Giorgia e al guarire insieme;

Ad Aldo e a Enrico, miei angeli custodi, compagni insostituibili di questi ultimi mesi.

Introduzione

Acquisire nuovi clienti richiede più tempo e più denaro rispetto a mantenere i clienti esistenti soddisfatti e fedeli. La creazione di un modello per il calcolo del tasso di abbandono dei clienti aiuta le aziende a identificare i clienti che probabilmente smetteranno di interagire con loro. La previsione e la prevenzione dell'abbandono rappresenta un'enorme fonte di reddito potenziale aggiuntiva per ogni azienda.
L'abbandono del cliente, ossia il *customer churn*, si riferisce a quando un cliente cessa il suo rapporto con l'azienda. In genere, le aziende considerano un cliente come perso quando un determinato periodo di tempo è trascorso dall'ultima interazione del cliente con i servizi dell'azienda. Il costo degli abbandoni include sia le mancate entrate dovute alla non erogazione dei servizi, sia i costi di marketing necessari per promuovere l'azienda e acquisire nuovi clienti in sostituzione di quelli persi. La riduzione del tasso di abbandono è quindi un obiettivo di *business* chiave per ogni attività.
Per riuscire a trattenere i clienti che stanno per abbandonare l'azienda, è necessario: (i) prevedere in anticipo quali clienti abbandoneranno; (ii) sapere quali azioni di marketing avranno maggiore impatto sulla fidelizzazione di ogni particolare cliente.
In questa tesi si cercherà una soluzione al primo requisito, prendendo come caso di studio la frequentazione dei clienti delle palestre servite da Technogym, azienda leader del fitness e del wellness, conosciuta in tutto il mondo

come *"The Wellness Company"*. A tal fine si vogliono sfruttare le potenzialità offerte dalle tecnologie di apprendimento automatico per prevedere il *churn risk* degli utenti della palestra; ossia la probabilità che l'utente stia per abbandonare.

Technogym offre già un servizio di previsione del rischio di abbandono basato su regole statiche che considerano la frequenza, l'età e la fidelizzazione dell'utente (intesa come il tempo trascorso dalla prima iscrizione) con la palestra. Tale servizio offre risultati accettabili ma è un sistema che non si adatta automaticamente al variare delle caratteristiche dei clienti nel tempo. Per far fronte a questi limiti, negli anni è stata promossa l'iniziativa di migrare il sistema ad uno basato su tecniche di apprendimento automatico. Allo scopo è stato realizzato un classificatore binario che utilizza un *Random Forest Classifier* che però finora non ha dato risultati corrispondenti a quanto ipotizzato.

Il lavoro di tesi prevede due fasi: la prima fase è la comprensione e l'analisi dell'attuale classificatore binario con lo scopo di evidenziare eventuali problemi nei dati e nella definizione delle *feature*. La seconda fase prevede lo studio, la definizione e la realizzazione di un modello utilizzando il servizio *AutoTable ML* di Google che si basi sulle stesse *feature* del modello di partenza. Ottenuti i due modelli verrà effettuata una valutazione comparativa delle performance. Il risultato finale della tesi è la realizzazione di una rete neurale atta a predire il *churn risk* degli utenti delle palestre prese in esame.

Il documento sarà così strutturato; nel primo capitolo si definiranno i problemi e le principali soluzioni tecnologiche in ambito Machine Learning con particolare attenzione a quelle applicabili al contesto di questa tesi. Il secondo capitolo consiste in una panoramica delle tecnologie utilizzate per la realizzazione del progetto. Il terzo capitolo definirà il problema di previsione dell'abbandono associato alla realtà di Technogym e spiegherà lo stato del lavoro di partenza. Nel terzo capitolo si procederà con un'analisi dei dati e

delle *features* con lo scopo di verificare il loro contenuto informativo rispetto alle ipotesi di progetto. Nel quarto capitolo si eseguirà la classificazione dei dati con i modelli di ML definiti e si effettuerà una valutazione dei risultati.

Indice

Elenco delle figure

Capitolo 1

Background del Machine Learning nei problemi trattati

Questo capitolo presenta il quadro teorico che supporta il lavoro di tesi nell'area del *Machine Learning*.

1.1 Motivazioni

L'intelligenza artificiale e l'apprendimento automatico non sono concetti recenti, ma già nel 1959 Arthur Samuel, un pioniere nel campo del *Machine Learning* (ML), lo definì come *"the field of study that gives computers the ability to learn without being explicitly programmed"*.
I primi ricercatori interessati all'intelligenza artificiale volevano scoprire se i computer potessero apprendere come eseguire compiti specifici attraverso il riconoscimento di *pattern* nei dati. A causa delle limitazioni tecnologiche di quel periodo questi concetti non hanno trovato applicazioni reali per molto tempo.
Oggi si sono verificati dei miglioramenti tecnologici che hanno reso l'ap-

prendimento automatico una realtà: grandi fonti di dati fondamentali per l'apprendimento, maggiore potenza di calcolo per l'elaborazione delle informazioni e algoritmi sempre più affidabili.
Tutto ciò ha permesso di realizzare modelli per l'analisi dati più potenti e complessi e di elaborare risultati più accurati anche su larga scala in tempi minori.
La filosofia alla base del *ML* è automatizzare la creazione di modelli analitici per consentire agli algoritmi di apprendere continuamente con l'aiuto dei dati disponibili. L'enorme potenziale del *ML* può aiutare le aziende a trasformare la mole di dati attualmente disponibile in informazioni di valore per l'azienda.
Il *ML* può essere inteso come l'insieme dei metodi computazionali che utilizzano l'esperienza per migliorare le prestazioni o per fare previsioni accurate. In questo caso, l'esperienza si riferisce a informazioni o dati passati a nostra disposizione. Come per qualsiasi esercizio di calcolo, la qualità e la quantità dei dati saranno cruciali per l'accuratezza delle previsioni che verranno effettuate.

1.1.1 Modelli Statistici

Sotto questo punto di vista, il *ML* può essere paragonato alla *modellazione statistica*. Nella modellazione statistica, si raccolgono dati, si verifica che siano di buona qualità (completi, corretti e privi di qualsiasi parte incompleta, errata o irrilevante) e quindi si utilizza questo set di dati pulito per testare ipotesi e fare previsioni.
L'idea alla base della modellazione statistica è il tentativo di rappresentare questioni complesse in termini relativamente generalizzabili, vale a dire termini che spiegano la maggior parte degli eventi studiati. In effetti, si programma l'algoritmo per eseguire determinate funzioni in base ai dati

forniti in input. In altre parole, l'algoritmo è statico. Ha bisogno di un programmatore che gli dica cosa fare quando viene alimentato con i dati.

1.1.2 Apprendimento Automatico

Nel *ML* la procedura è capovolta. Piuttosto che preselezionare un modello e fornirgli i dati, nel ML sono i dati a determinare quale tecnica analitica dovrebbe essere selezionata per svolgere al meglio il compito in questione. In altre parole, si utilizzano i dati a disposizione per selezionare e addestrare l'algoritmo. Quindi l'algoritmo è dinamico. Analizza i dati a cui è esposto, determina la migliore linea d'azione e quindi agisce. In sostanza, "apprende" dai dati e ne estrae la conoscenza.
Questo metodo di apprendimento si basa sulla ripetizione. Un algoritmo non è altro che un insieme di istruzioni che un computer utilizza per trasformare un input in un particolare output. Pertanto, in ML, l'aspetto dell'apprendimento è solo un algoritmo che ripete l'esecuzione più e più volte e apporta lievi modifiche fino a quando non soddisfa una certa serie di condizioni. Le performance di un modello si valutano controllando che sia in grado di fare previsioni su nuovi dati rispetto ai quali non era stato precedentemente addestrato.

1.1.3 Apprendimento Supervisionato e Non Supervisionato

In questo processo metodologico i dati giocano il ruolo primario. In particolare, è la struttura dei dati che determina come avverrà il processo di apprendimento.
Esistono due tipi principali di apprendimento: supervisionato e non supervisionato.

La differenza sostanziale tra i due tipi è che l'apprendimento supervisionato viene effettuato utilizzando una verità di base; si ha una conoscenza preliminare di quali dovrebbero essere i valori di output per i nostri campioni. Quindi, l'obiettivo dell'apprendimento supervisionato è apprendere una funzione che, dato un campione di dati e di output desiderati, approssimi al meglio la relazione tra input e output osservabile nei dati.
L'apprendimento non supervisionato, d'altra parte, non ha output etichettati, quindi il suo obiettivo è inferire la struttura naturale presente all'interno di un insieme di campioni di dati.

ML Supervisionato

Nell'apprendimento supervisionato l'algoritmo viene addestrato su dati etichettati. Ciò significa che i dati sono contrassegnati con l'etichetta corretta o il risultato corretto.
In genere, l'apprendimento supervisionato viene svolto nel contesto della *classificazione*, quando si desidera mappare l'input sulle etichette di output, o nel contesto della *regressione*, quando si desidera mappare l'input su un output continuo.
L'obiettivo dell'apprendimento supervisionato è trovare relazioni o strutture specifiche nei dati di input che ci consentano di produrre efficacemente dati di output corretti.
Si noti che l'output "corretto" è determinato interamente dai dati di addestramento; questi rappresentano una *ground truth* che il modello supporrà vera. Bisogna tenere presente che le etichette dei dati nelle situazioni del mondo reale non sono sempre corrette; etichette rumorose o errate ridurranno chiaramente l'efficacia del modello.
Una volta addestrato, l'algoritmo riceve nuovi dati e utilizza la sua esperienza passata per prevedere un risultato.

ML Non Supervisionato

Nell'apprendimento non supervisionato l'algoritmo viene addestrato utilizzando un set di dati non etichettato.

All'algoritmo non viene detto cosa rappresentano i dati. In questo caso, il processo di apprendimento dipende dall'identificazione dei pattern che vengono ripetutamente trovati nei dati.

Gli algoritmi devono utilizzare metodi di stima basati su statistiche inferenziali per scoprire pattern, relazioni e correlazioni nel set di dati grezzo e senza etichette. Quando i pattern vengono identificati, l'algoritmo utilizza le statistiche per identificare i confini all'interno del set di dati. I dati con caratteristiche simili vengono raggruppati insieme, creando sottoinsiemi di dati. Man mano che il processo di classificazione continua, l'algoritmo inizia a comprendere il set di dati che sta analizzando, consentendogli di prevedere la categorizzazione dei dati futuri.

Le attività più comuni all'interno dell'apprendimento non supervisionato sono il *clustering, il representation learning, e la riduzione della dimensionalità*. In tutti questi casi, si desidera conoscere la struttura intrinseca dei nostri dati senza utilizzare etichette fornite esplicitamente. Poiché non vengono fornite etichette, nella maggior parte dei metodi di apprendimento non supervisionato, non esiste un modo specifico per confrontare le prestazioni del modello.

L'apprendimento non supervisionato è molto utile nell'*analisi esplorativa* perché può identificare automaticamente la struttura nei dati. Ad esempio, se un analista stesse cercando di segmentare i consumatori, i metodi di clustering non supervisionati sarebbero un ottimo punto di partenza per la loro analisi. In situazioni in cui è impossibile o poco pratico per un essere umano proporre tendenze nei dati, l'apprendimento non supervisionato può fornire intuizioni iniziali che possono essere poi utilizzate per testare singole ipotesi.

La *riduzione della dimensionalità*, che si riferisce ai metodi utilizzati per rappresentare i dati utilizzando meno *feature*, può essere ottenuta tramite metodi non supervisionati.
Nel *representation learning*, si desidera apprendere le relazioni tra le singole *feature*, permettendo di rappresentare i dati utilizzando caratteristiche latenti che sono correlate alle nostre caratteristiche iniziali.
Questa struttura latente sparsa è spesso rappresentata utilizzando molte meno *feature* rispetto a quelle con cui abbiamo iniziato, quindi può rendere l'ulteriore elaborazione dei dati molto meno intensa ed eliminare *feature* ridondanti.
Poiché il lavoro di tesi consiste in un problema di classificazione binaria, che dovrà permettere di associare ad ogni record una classe tra *dropout* (abbandonerà la palestra) e *no-dropout* (non abbandonerà la palestra), ci si concentrerà sui metodi di apprendimento supervisionato.

1.2 Problema di Classificazione

Con *Classificazione* si intende una tecnica di predizione che data una collezione di record (*training set*) composti da una serie di attributi (*feature*), di cui uno ne esprime la classe di appartenenza (*label*), ha come obiettivo quello di assegnare dei record non noti ad una classe di appartenenza, nel modo più accurato possibile.
L'attività di classificazione cerca quindi di individuare un modello per l'attributo di classe che sia in grado di esprimere il valore di tale attributo in funzione del valore degli altri attributi che compongono il record.
Per determinare l'accuratezza del modello, il dataset viene suddiviso in:

- *Training set*: composto da record di cui è nota la classe di appartenenza e viene utilizzato per costruire e allenare il modello.

- *Test set*: composto da record di cui il modello ignora la classe di appartenenza e viene utilizzato come strumento di validazione; per verificarne l'efficacia si valuta quanti record vengono classificati correttamente.

Di seguito verranno mostrate le principali tecniche di classificazione.

1.2.1 Decision Tree

Un *Decision Tree* è una delle tecniche di classificazione maggiormente utilizzate; permette di rappresentare un insieme di *regole di classificazione* tramite un albero. La struttura gerarchica dell'albero è formata da un insieme di *nodi* e un insieme di *archi* orientati ed etichettati.
A partire dai dati di cui si conosce la classe di appartenenza, l'algoritmo costruisce una serie di regole che permettono di suddividere (*criterio di split*) i dati in gruppi di classificazione il più puri possibile: con *purezza* si intende *minore entropia* tra l'attributo di classe dei record. Un gruppo ideale conterrà solo record della stessa classe.

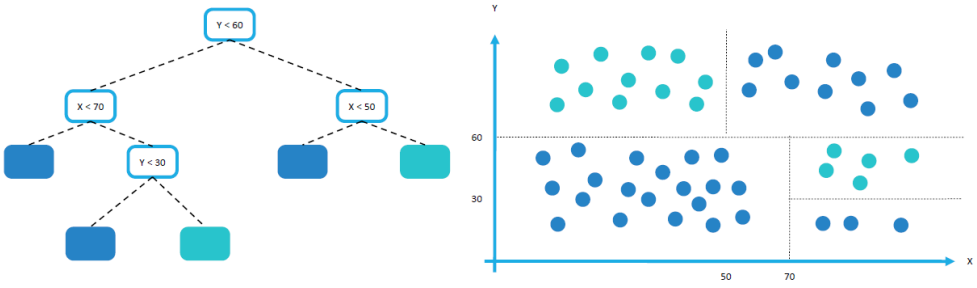


Figura 1.1: Ogni nodo è una regola di classificazione; l'insieme di queste regole definisce il percorso logico che il modello deve seguire per capire la classe di appartenenza del record che si sta classificando.

Il criterio che definisce lo *split ottimo* è quello che permette di determinare classi di record i più omogenei possibile, facendo attenzione che tale classificazione aumenti il guadagno informativo: ad esempio, non sarebbe utile classificare gli utenti in base al codice fiscale.
Per calcolare il *livello di purezza* di uno split è necessario calcolare l'*impurità totale* che dipende dalla misura di purezza adottata. Le misure di purezza più utilizzate sono *Gini Index, Entropy* e *Missclassificator Error*.
Trovare un *Decision Tree* ottimo è un problema NP-Completo, tuttavia gli algoritmi euristici utilizzano un approccio di partizionamento ricorsivo *top-down* basato su *criteri greedy* molto efficienti.

Vantaggi dei *Decision Tree*:

- La classificazione è molto veloce (nel caso peggiore è O(w) dove w è la profondità dell'albero) e offre una facile interpretazione dei criteri di partizionamento.
- Sono classificatori robusti rispetto alla presenza di attributi fortemente correlati, in quanto ne considerano solo uno mentre scartano l'altro perché non porterebbe alcun guadagno informativo.

Svantaggi dei *Decision Tree*::

- Hanno espressività limitata; il partizionamento dello spazio di ricerca può coinvolgere un solo attributo per volta quindi i *decision boundary* sono paralleli agli assi.

I *Decision Tree* sono molto utili quando si deve fare classificazione e il tempo di calcolo è un vincolo stringente e possono chiarire quali *feature* del dataset esercitano il potere predittivo più elevato.
Inoltre, a differenza di molti algoritmi di apprendimento automatico in cui le regole utilizzate per classificare i dati possono essere difficili da interpretare,

gli alberi decisionali rendono queste regole interpretabili.
I *Decision Tree* sono anche in grado di utilizzare sia variabili categoriali che continue, il che significa che è necessaria una minore preelaborazione, rispetto agli algoritmi che possono gestire solo uno di questi tipi di variabili.

1.2.2 Rule Based Classifier

I classificatori basati su regole utilizzano insiemi di regole di tipo *if...then* per classificare i record di un dataset.
Una *regola di classificazione* ha la forma *(Condizione)*$\rightarrow y$:

- *Condizione* (o antecedente); è una congiunzione di predicati su uno o più attributi del record.
- y (o conseguente); è la classe che soddisfa la condizione e quindi viene assegnata al record.

Il modello viene quindi identificato da un insieme di regole, come l'esempio mostrato nella Figura 1.2.

Questi classificatori sebbene abbiano la stessa espressività dei *Decision Tree*:

- hanno il vantaggio di essere più semplici sia da generare sia da interpretare e tendono ad essere più rapidi nel classificare nuove istanze.
- hanno lo svantaggio che all'aumentare delle dimensioni del training set il costo di costruzione aumenta. Inoltre risentono fortemente di un eventuale rumore sui dati.

La qualità di una *regola di classificazione* è valutata dalla:

- Copertura; frazione di record che soddisfano l'antecedente della regola.

Nome	Tipo di sangue	Partorisce	Può volare	Vive in acqua	Classe
umano	caldo	si	no	no	mammiferi
pitone	freddo	no	no	no	rettili
salmone	freddo	no	no	si	pesci
balena	caldo	si	no	si	mammiferi
rana	freddo	no	no	talvolta	anfibi
komodo	freddo	no	no	no	rettili
pipistrello	caldo	si	si	no	mammiferi
Piccione	caldo	no	si	no	uccelli
gatto	caldo	si	no	no	mammiferi
leopardo	freddo	si	no	si	pesci
squalo	freddo	no	no	talvolta	rettili
tartaruga	caldo	no	no	talvolta	uccelli
pinguino	caldo	si	no	no	mammiferi
porcospino	freddo	no	no	si	pesci
Anguilla	freddo	no	no	talvolta	anfibi
salamandra	freddo	no	no	no	rettili
mostro di gila	caldo	no	no	no	mammiferi
ornitorinco	caldo	no	si	no	uccelli
gufo	caldo	si	no	si	mammiferi
delfino	caldo	no	si	no	uccelli
aquila					

r1: (Partorisce = no) AND (Può volare = si) --> Uccelli
r2: (Partorisce = no) AND (Vive in acqua = si) --> Pesci
r3: (Partorisce = si) AND (Tipo di sangue = caldo) --> Mammiferi
r4: (Partorisce = no) AND (Può volare = no) --> Rettili
r5: (Vive in acqua = talvolta) --> Anfibi

Figura 1.2: Una regola rX copre un'istanza z del dataset se i valori degli attributi di tale istanza soddisfano l'antecedente della regola. L'istanza viene etichettata con la classe corrispondente al conseguente della regola.

- Accuratezza; frazione di record che soddisfano la regola e appartengono alla classe specificata nel conseguente.

Tid	Rimborso	Stato Civile	Reddito tassabile	Classe
1	SI	Single	125K	No
2	No	Sposato	100K	No
3	No	Single	70K	No
4	SI	Sposato	120K	No
5	No	Divorziato	95K	Si
6	No	Sposato	60K	No
7	SI	Divorziato	220K	No
8	No	Single	85K	Si
9	No	Sposato	75K	No
10	No	Single	90K	Si

Figura 1.3: Rule Coverage e Accuracy

Un Classificatore *Rule-Based* ottimo è un classificatore in cui ogni record è *coperto* da una e una sola regola. Questa situazione non è sempre possibile; nel caso in cui un record è coperto da più regole bisogna definire una strategia che definisca quale classe associare al record.

Vi sono due approcci principali:

- *Liste di Decisione*: si definisce un ordine di importanza delle regole.

- *Voto*: si assegna il record alla classe per la quale vengono attivate più regole; ossia la classe che compare più volte come conseguente.

Per tutti i record che non attivano alcuna regola è possibile definire una *classe di default* che rappresenti "il resto del mondo".

1.2.3 K-Nearest Neighbors (K-NN)

K-Nearest Neighbors (KNN) è un algoritmo di apprendimento automatico che può essere utilizzato sia per attività di regressione che di classificazione. KNN è un algoritmo concettualmente semplice ma molto potente e, per questi motivi, è uno degli algoritmi di apprendimento automatico più popolari.
KNN fa parte della famiglia dei classificatori *instance-based*; sono sistemi predittivi in quanto permettono la classificazione dei record, ma non sono sistemi descrittivi in quanto non costruiscono un modello che descrive i dati. Questo significa che i nuovi record vengono classificati sulla base della loro *somiglianza* rispetto ai record presenti nel training set.
Al posto del modello si ha quindi: (i) un *trainin set* etichettato che rappresenta il dominio, (ii) una *metrica* per calcolare le distanze tra i record di test e quelli di training (iii) e l'iperparametro k che corrisponde al numero di *neighbors* da considerare per classificare i nuovi record.
L'idea fondamentale dei classificatori *K-Nearest Neighbors* è quella di classificare un record sulla base dei k punti "più vicini" del training set; i *k-nearest neighbor* di un record x sono i k *record* del *training set* che hanno le *più piccole distanze* dal record x. Una volta calcolate le distanze rispetto ai record del *trainin set*, si identificano i k *record* con distanza minore rispetto al record di test e si utilizzano le classi di appartenenza dei *k-nearest neighbor* per determinare la classe di appartenenza del record da classificare.

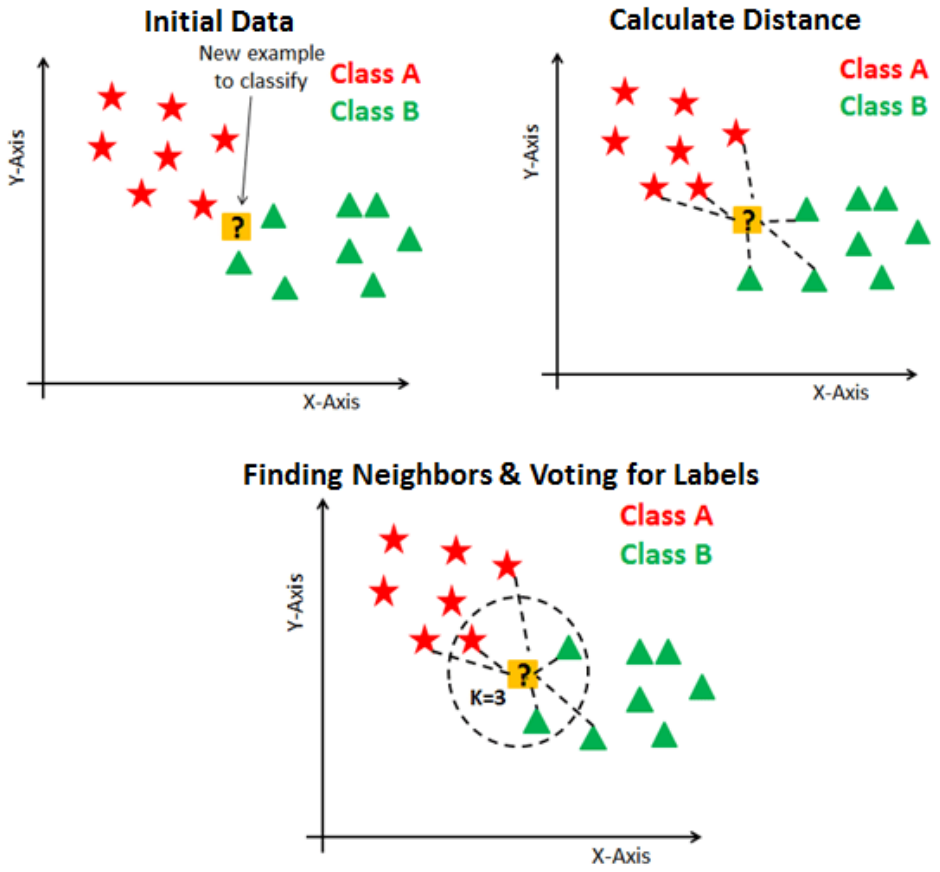


Figura 1.4: Processo di classificazione di K-NN

La soluzione più adottata per determinare la classe è quella chiamata *majority voting*; al record di test viene assegnata la classe più comune tra i *k-nearest neighbor*.
Questo approccio assegna lo stesso peso a tutti i *k-neighbors* individuati, indipendentemente dalla loro distanza dal record di test. Per ridurre la sensibilità rispetto al rumore nei dati del training set è possibile assegnare ad ogni *neighbor* un *peso* in base alla sua distanza dal record di test.

Vantaggi del *K-Nearest Neighbors Classifier*:

- Non richiede la costruzione del modello, quindi non fa alcuna ipotesi sui dati, il che significa che può essere utilizzato su un'ampia varietà di problemi.

- É accurato ed estremamente semplice da usare. É facile da interpresare, capire e implementare.

- A differenza dei classificatori *Rule-based* o *Decision Tree* permette la definizione di *intorni* (confini di classificazione) anche non lineari e quindi risultano più flessibili.

Svantaggi del *K-Nearest Neighbors Classifier*:

- Richiede una *metrica di similarità* per valutare la vicinanza tra i record da classificare e i record appartenenti al training set.

- É molto sensibile alla scala del training set quindi richiede spesso una *fase di pre-processing* in cui si normalizza il range di variazione degli attributi.

- La determinazione della classe è una *scelta locale*, per cui è suscettibile al rumore dei dati.

- É molto sensibile alla presenza di attributi irrilevanti o correlati in quanto falsificano la reale distanza tra due record.

- Il costo di classificazione è molto elevato perché KNN richiede la memorizzazione della maggior parte (o di tutti) i dati di training, il che significa che richiede molta memoria e il costo di calcolo dipende linearmente dalla dimensione del training set.

1.2.4 Naive Bayes Classifier

L'algoritmo di apprendimento bayesiano è l'approccio di apprendimento più pratico per la maggior parte dei problemi di apprendimento e si basa sulla valutazione delle probabilità esplicite per delle ipotesi.
I *classificatori bayesiani* utilizzano quindi un approccio probabilistico per risolvere problemi di classificazione; in molti casi, infatti, la relazione tra il valore degli attributi di un record e la classe ad esso assegnata non è deterministica. Questo non determinismo è dovuto alla presenza di rumore sui dati e da eventuali caratteristiche del fenomeno che non vengono modellate dai dati.
I *classificatori bayesiani* tengono presente questo concetto e modellano le relazioni tra le *feature* e l'attributo di classe dei record in modo probabilistico. Alla base di questa modellazione vi è il *Teorema di Bayes*.

Teorema di Bayes

Il *Teorema di Bayes* è un metodo per calcolare la *probabilità condizionata*: può avvenire infatti che il verificarsi di un evento *influenzi* il verificarsi o meno di un altro. Si dice allora che lo condiziona, ovvero che l'evento influenzato è *condizionato*.
La probabilità che avvenga A, noto che B si sia verificato, si scrive $P(A|B)$ e si legge "probabilità di A dato B", ed è chiama *probabilità condizionata*.
Il metodo tradizionale di calcolo della *probabilità condizionata* consiste nel calcolare la *probabilità congiunta* ($P(A, B)$) dell'evento A e dell'evento B, cioè la probabilità che i due eventi si verificano contemporaneamente, e poi dividerla per la probabilità che si verifichi l'evento B:

$$P(A|B) = P(A, B)/P(B)$$

Tuttavia, il *Teorema di Bayes* permette di calcolare la *probabilità condizionata* in modo leggermente diverso:

$$P(A|B) = P(B|A) * P(A)/P(B)$$

Dove:

- $P(B|A)$: probabilità condizionata di B dato A.
- $P(A)$: probabilità a priori di A, quindi probabilità che l'evento A sia vero senza conoscere nulla a suo riguardo.
- $P(B)$: probabilità a priori di B, quindi probabilità che l'evento B sia vero senza conoscere nulla a suo riguardo.

Questo è particolarmente utile quando la *probabilità condizionata inversa* è facilmente calcolabile.

I *Naive Bayes Classifier* permettono la classificazione dei dati basandosi sul *Teorema di Bayes* assumendo l'indipendenza tra le *feature* quando la classe è nota. Ad esempio, un frutto può essere considerato una *mela* se è (i) rosso, (ii) rotondo (iii) e di circa 8cm di diametro. Anche se queste caratteristiche dipendono l'una dall'altra o dall'esistenza delle altre caratteristiche, tutte queste proprietà contribuiscono indipendentemente alla probabilità che questo frutto sia una *mela*. É per questo motivo che questo classificatore è noto come *naive*, "ingenuo".
In particolare, il processo di classificazione procede in questo modo: dato un vettore A che descrive l'insieme di attributi $(A_1, A_2, .., A_n)$ e sia C l'attributo di classe del record, se C è legato in modo non deterministico ai valori del vettore A, allora è possibile trattare le due variabili casuali e catturare la loro relazione probabilistica tramite la probabilità condizionata $P(C|A)$.
Il *Naive Bayes Classifier* assume che gli attributi A del record siano *indipendenti*. Grazie a questa assunzione di indipendenza, invece di calcolare la probabilità condizionata di ogni possibile combinazione di valori per ogni

attributo del dato, è sufficienze calcolare la probabilità condizionata di ogni valore di ogni singolo attributo:

$$P(C|A_1, A_2, .., A_n) = P(A_1|C) * P(A_2|C) * ...P(A_n|C)$$

Nella fase di test un record verrà etichettato con la classe che massimizza la probabilità $P(C|A)$.
Il vantaggio di utilizzare *classificatori probabilistici* (come *Naive Bayes*) rispetto ai *classificatori logici* (come *Decision Tree, Rule-based, K-Nearest Neighborn*) consiste nella possibilità di giungere a descrizioni razionali anche quando non c'è abbastanza informazione di tipo deterministico sul funzionamento del sistema. Inoltre, forniscono risultati ottimi se la condizione di indipendenza condizionale è rispettata (ossia non ci sono attributi correlati) e se sono note le distribuzioni di *probabilità a priori* ($P(A|C)$).

1.2.5 Support Vector Machines (SVM)

Support Vector Machine (SVM) è un algoritmo di apprendimento automatico supervisionato che può essere utilizzato sia per attività di classificazione sia per attività di regressione.
Questi classificatori operano cercando di individuare un *iperpiano* che divida al meglio l'insieme dei dati rispetto alle classi di appartenenza.
Un *iperpiano* o *limite di decisione lineare*, per un'attività di classificazione binaria, può essere rappresentato come una linea che separa e classifica i dati in due insiemi. A tre dimensioni è rappresentato da un piano, mentre a più di tre dimensioni è chiamato in generale *iperpiano*.

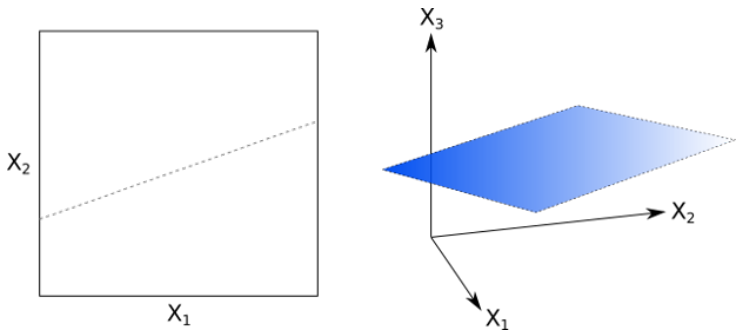


Figura 1.5: A sinistra un iperpiano a due dimensioni. A destra un iperpiano a tre dimensioni.

Un iperpiano è definito dai *support vector*; i punti del training set più vicini all'iperpiano. Se tali punti vengono rimossi o modificati, anche l'iperpiano cambia di posizione.
Altro concetto fondamentale dei classificatori *SVM* è quello di *margine*: il margine è la distanza tra i *support vector* di due classi diverse.
A metà di questa distanza viene tracciato l'iperpiano, come mostrato nella Figura 1.6.

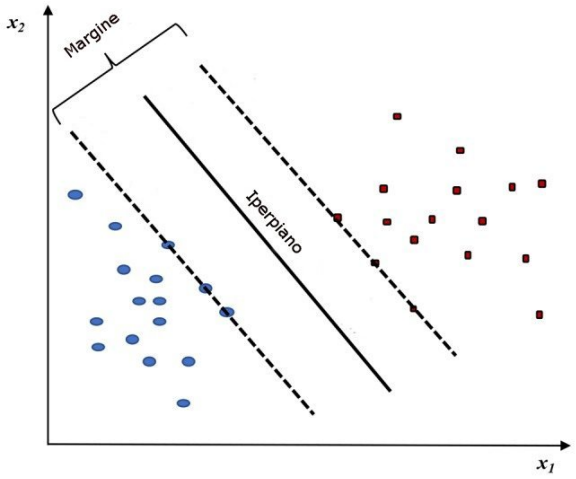


Figura 1.6: Margine di separazione tra i *support vector* delle due classi.

Il *Support Vector Machine* ha l'obiettivo di identificare una funzione che massimizzi il margine tra i *support vector* delle classi in esame. Infatti, un set di dati può benissimo essere separabile da più funzioni senza errori, ma più ampio è il margine, minore è il numero di errori di classificazione, perché le classi sono meglio distinte l'una dall'altra.
La Figura 1.7 mostra che un set di dati può benissimo essere separabile da più funzioni senza errori. Pertanto, il margine attorno a una funzione di separazione viene utilizzato come parametro aggiuntivo per valutare la qualità della separazione. In questo caso la separazione A è quella migliore, poiché distingue le due classi in maniera più precisa.

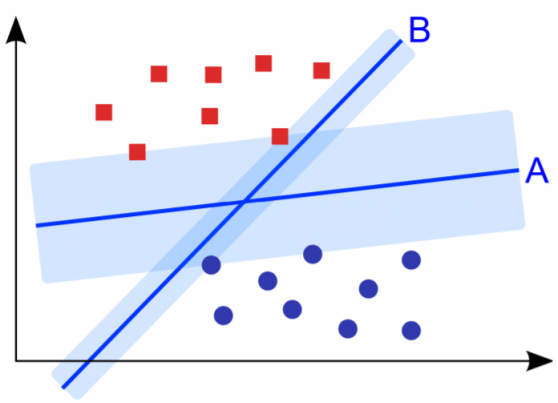


Figura 1.7: Visualizzazione di un *Support Vector Machine* che divide un set di dati in due classi, utilizzando due diverse separazioni lineari che hanno margini di dimensioni diverse attorno alle funzioni di divisione

L'obiettivo è quello di cercare un iperpiano linearmente separabile, ma se tale iperpiano non esiste, SVM utilizza il *Kernel Method* che permette di mappare i dati di input N-dimensionali in uno spazio dimensionale superiore, dove i dati possono essere separati linearmente.

Vantaggi dell'algoritmo *SVM*:

- Efficace in dimensioni spaziali elevate
- Efficienza della memoria; nel processo decisionale di classificazione di nuovi dati è sufficiente considerare il sottoinsieme dei dati di training specificati come *support vector*.
- Versatilità; nel mondo reale è altamente probabile che la separazione delle classi non sia lineare, quindi la capacità di applicare nuovi kernel permette ad *SVM* di essere flessibile e di avere una migliore performance di classificazione.

Svantaggi dell'algoritmo *SVM*:

- Difficile interpretazione; mancanza di trasparenza dei risultati. Bisogna ricorrere a tecniche di visualizzazione grafica.
- Metodo non probabilistico; non viene restituita una probabilità di appartenenza alle classi. Una potenziale metrica per valutare l'efficacia della classificazione è quanto lontano sia l'iperpiano rispetto al nuovo punto da classificare.

1.2.6 Multi Classifiers

L'utilizzo di un *multiclassificatore* si basa sull'idea di costruire più classificatori semplici e predirre la classe di appartenenza di un record tramite l'aggregazione delle singole classificazioni ottenute.
Normalmente l'aggregazione delle singole classificazioni è un voto di maggioranza; una sorta di *majority voting* che assegna al record la classe più "votata" dai singoli classificatori. Affinché un *Multi Classifier* dia risultati migliori rispetto ad un singolo classificatore potenziato, è necessario che vengano rispettate alcune condizioni:

- *Indipendenza dei classificatori*; i classificatori semplici devono essere indipendenti ossia non deve esserci correlazione tra i vari *error rate*.

- *Bontà dei classificatori*; l'*error rate* dei classificatori non deve essere superiore al 50% perché significherebbe che etichetta i record in modo casuale.

L'indipendenza dei classificatori può essere ottenuta cambiando:

- *il Training Set*; a partire dal dataset iniziale si costruiscono diversi *training set* (con tecniche di *boosting* e *bagging*) in modo che ogni classificatore semplice si alleni su un *training set* diverso.

- *gli Attributi utilizzati*; i singoli classificatori vengono addestrati su sottoinsiemi diversi degli attributi specialmente in caso di presenza di attributi ridondanti.

- *le Classi considerate*; le classi vengono partizionate in due gruppi, in modo che ogni classificatore possa lavorare su un problema binario. Per ogni classificatore semplice viene fatto un partizionamento delle classi diverso e il risultato della classificazione finale avviene incrementando il punteggio di tutte le classi che appartengono al sottoinsieme scelto. Alla fine il record viene assegnato alla classe con il punteggio più alto.

- *i Parametri dell'algoritmo*.

La classe di appartenenza di un record verrà poi assegnata in base a meccanismi di voting: si sceglie la classe assegnata dalla maggioranza dei singoli classificatori. É possibile assegnare un peso al voto di classificatore, in base alla confidenza fornita da quest'ultimo.

1.2.7 Random Forest Classifier

Il *Random Forest Classifier* è un multiclassificatore costituito da un gran numero di *Decision Tree*; appunto una "foresta" di alberi decisionali che operano come un insieme.
Ogni albero della foresta esegue la classificazione del record ed emette una previsione di classe. La classe con il maggior numero di voti diventa la previsione del modello, come mostrato nella Figura 1.8.

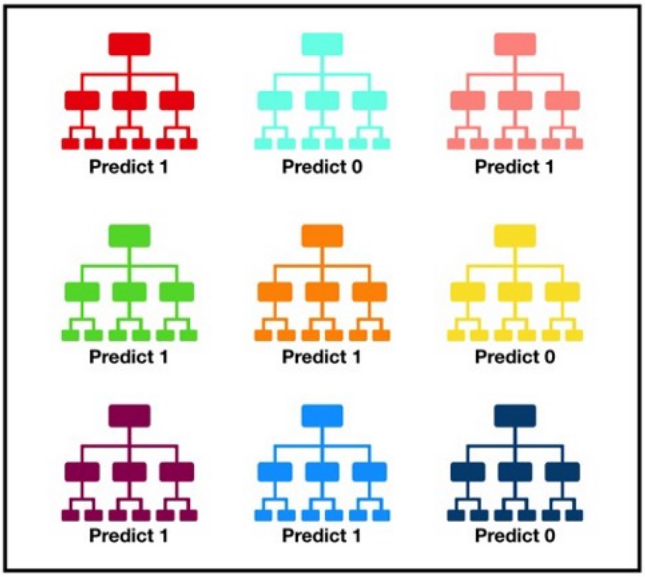


Figura 1.8: Esecuzione di una previsione con Random Forest.

L'idea fondamentale alla base di *Random Forest*, e in generale dei multiclassificatori, è semplice ma potente: *la saggezza della folla*. L'intelligenza collettiva derivante dalle risposte di più individui (umani o macchine) alle stesse domande, può essere più accurata di quella di un singolo individuo e può, di conseguenza, migliorare il processo decisionale sociale e l'accuratez-

za delle previsioni.
Un gran numero di modelli (*decision tree*) relativamente non correlati che operano come un unico classificatore, supererà i singoli modelli costituenti. Questa non correlazione, *Random Forest* la raggiunge tramite il *Bagging* (*Boostrap Aggregation*): una tecnica di addestramento di più modelli dello stesso tipo su dataset diversi, ciascuno ottenuto dal dataset iniziale con *campionamento casuale* con *reimbussolamento*:

- Estrazione con reimbussolamento di un sottoinsieme S_j di pattern dal training set (tipicamente i 2/3 del totale);
- Addestramento del classificatore C_j sul sottoinsieme S_j.
- E via dicendo...

Per rendere i vari *decisioni tree* maggiormente indipendenti, oltre al *bagging* sui campioni del training set, viene anche eseguito il *bagging* sulle feature:

- Per ogni nodo di ogni *decision tree* la scelta della *feature* migliore su cui partizionare non è fatta sull'intero insieme delle *d feature*, ma un un sottoinsieme random.

In assenza di questo accorgimento molti alberi sceglierebbero le stesse variabili (quelle più discriminanti).
La bassa correlazione tra i modelli è la chiave per l'efficacia di *Random Forest*; modelli non correlati possono produrre previsioni di insieme che sono più accurate di qualsiasi previsione individuale.

1.2.8 Artificial Neural Networks

Singh e Chauhan definiscono le *reti neurali artificiali* come *"un modello matematico che si basa sulle reti neurali biologiche con lo scopo di emulare*

il sistema neurale biologico" [22].

Negli ultimi anni le varie attività di ricerca sulla classificazione neurale hanno stabilito che le *reti neurali* sono una valida alternativa ai vari metodi di classificazione convenzionali.

Il vantaggio delle reti neurali risiede nei seguenti aspetti teorici. In primo luogo, le reti neurali sono metodi auto-adattativi guidati dai dati, in quanto possono adattarsi ai dati senza alcuna specifica esplicita di forma funzionale o distributiva per il modello sottostante. In secondo luogo, sono approssimatori funzionali universali in quanto le reti neurali possono approssimare qualsiasi funzione con precisione arbitraria [37], [78], [79].

Poiché qualsiasi procedura di classificazione cerca una relazione funzionale tra l'appartenenza al gruppo e gli attributi dell'oggetto, un'identificazione accurata di questa funzione sottostante è senza dubbio importante. Terzo, le reti neurali sono modelli non lineari, il che le rende flessibili nella modellazione delle relazioni complesse del mondo reale. Infine, le reti neurali sono in grado di stimare le probabilità a posteriori, che forniscono la base per stabilire regole di classificazione ed eseguire analisi statistiche [138].

Una *Rete Neurale Artificiale* è composta da un gran numero di elementi di elaborazione altamente interconnessi chiamati *neuroni* che lavorano all'unisono per risolvere un problema specifico.

I neuroni sono raggruppati e organizzati in livelli. Tipicamente sono presenti: un livello di *input* che riceve le informazioni da fonti esterne, ul livello di *output* che produce l'output della rete e uno o più livelli intermedi o nascosti (*hidden*) che collegano il livello di input con il livello di output.

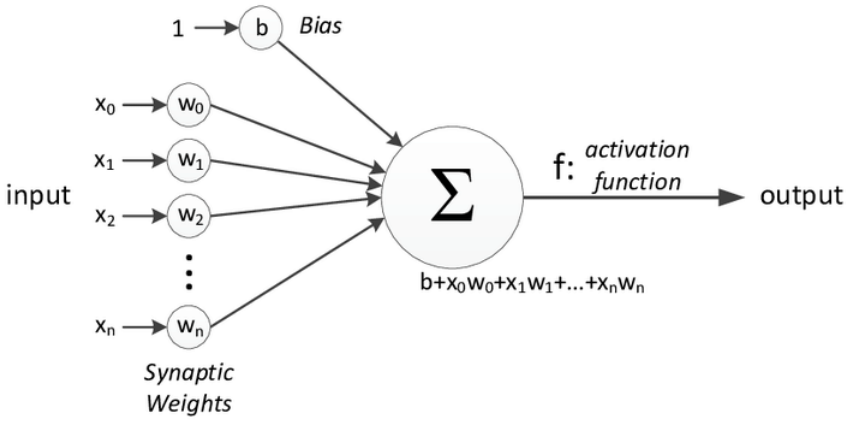


Figura 1.9: Esempio di neurone di una Rete Neurale Artificiale.

In Figura 1.9 è mostrato un esempio di neurone. In particolare vi sono gli input $x_0, x_1, x_2, \ldots, x_n$, che rappresentano le variabili indipendenti della rete. Ognuno di questi viene moltiplicato per un peso di connessione o sinapsi, $w_0, w_1, w_2, \ldots, w_n$, che rappresenta la forza di una particolare connessione. Il *bias* b è un valore di polarizzazione che consente di spostare la funzione di attivazione verso l'alto o verso il basso; serve per "tarare" il punto di lavoro ottimale del neurone.
La funzione di attivazione $\Phi(\sum_{i=0}^{n} x_i * w_i)$, ha lo scopo di convertire il segnale di input di un nodo in un segnale di uscita che sarà utilizzato come input nel livello successivo della rete [3].
Se non applichiamo la funzione di attivazione, il segnale di uscita sarebbe semplicemente una funzione lineare (*polinomio di grado uno*). Certamente, le funzioni lineari sono facili da risolvere ma sono limitate nella loro complessità, hanno meno potenza. Quindi, senza la funzione di attivazione, il modello non potrebbe apprendere e modellare dati complessi.
Le reti neurali possono essere suddivise in due tipi principali:

- *Feedforward Networks*; sono definite come *reti che non ottengono feed-*

back dalla rete stessa. Ciò significa che i dati di input fluiscono in un'unica direzione: dai nodi di input, attraverso gli n livelli nascosti, fino ai nodi di output. Non sono consentite connessioni all'indietro o connessioni tra neuroni dello stesso livello. Non sono fornite informazioni all'indietro per riadattare il sistema.

- *Recurrent Networks*; sono definite come *reti che contengono un'opzione di feedback* e quindi sono in grado di riutilizzare i dati delle fasi successive per il processo di apprendimento nelle fasi precedenti.
  Sono quindi consentite connessioni verso lo stesso livello e all'indietro. Questo complica notevolmente il flusso delle informazioni e l'addestramento, richiedendo di considerare il comportamento in più istanti temporali. Sono reti indicate per la gestione di *sequenze* perché dotate di un *effetto memoria* [2].

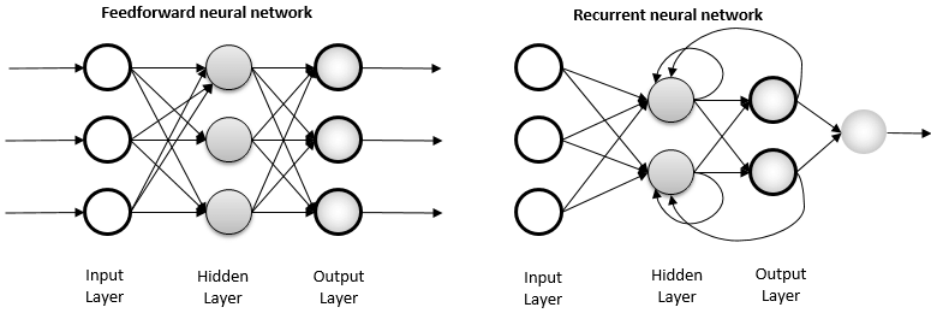


Figura 1.10: A sinistra un esempio di Rete Feedforward. A destra un esempio di Rete Ricorrente.

Capitolo 2

Tecnologie

Le tecnologie utilizzate per la realizzazione del progetto sono state indicate dall'azienda; la quale lavora principalmente sul *cloud* appoggiandosi alla suite *Google Cloud Platform* (GCP).

2.1 Google Big Query

Con questo strumento Google permette ai propri utenti di eseguire query su tabelle di miliardi di record, con tempi di risposta ottimizzati.
Le principali caratteristiche di questo strumento sono:

- Scalabilità: uno dei vantaggi intrinseci del *Cloud Computing* è la capacità di ampliare l'infrastruttura su richiesta, garantendo scalabilità dinamica e capacità di calcolo in base all'aumento delle esigenze. Tale aspetto si rivela particolarmente utile quando il livello di utilizzo di picco delle applicazioni ospitate cambia in modo consistente con il trascorrere del tempo (ad esempio, le applicazioni utilizzate nel settore della vendita al dettaglio durante i periodi di festività e così via).

- Interattività: riesce ad eseguire query di selezione o di raggruppamento su miliardi di record in pochi secondi.
- Familiarità: utilizza un dialetto *SQL* per la scrittura delle query, facilitandone l'utilizzo.

Inoltre, essendo *Big Query* strettamente collegato a *Google Storage*, il suo utilizzo può essere una buona scelta se si ha la necessità di condividere e di elaborare dati molto velocemente.

- *Google Storage* può essere visto come il *data lake* di *Google*. Infatti, permette di salvare, condividere e gestire dati di qualsiasi tipo e di qualsiasi dimensione.

Technogym utilizza il servizio *Google Big Query* come *data warehouse* multicloud serverless ad elevata scalabilità. Nel progetto di tesi è stato utilizzato per eseguire analisi su petabyte di dati ad alta velocità e per le operazioni di ETL (Extract, Transform, and Load).

2.1.1 Funzioni Analitiche

Ciò che è risultato molto utile per l'analisi e il calcolo delle *features* sono state le *analytic function* che *Big Query* mette a disposizione per calcolare dei valori su un gruppo di righe restituendo un singolo risultato per ogni riga: ciò è diverso da una funzione di aggregazione standard che restituisce un singolo risultato per un gruppo di righe.
Una funzione analitica include una clausola `OVER`, che definisce una *finestra di righe* attorno alla riga da valutare. Per ogni riga, il risultato della *analytic function* viene calcolato utilizzando la finestra di righe selezionata come input, possibilmente facendo l'aggregazione. Con le funzioni analitiche è possibile calcolare medie mobili, classificare elementi, calcolare somme cumulative ed eseguire analisi complesse.

Esistono molti tipi di funzioni che possono essere utilizzate e rientrano in tre categorie principali:

- Numerazione: `RANK, DENSE_RANK, ROW_NUMBER`
- Aggregazione: `SUM, AVG, COUNT, MIN, MAX`
- Navigazione: `LEAD, LAG, FIRST_VALUE, LAST_VALUE`

Queste funzioni si basano sul concetto di finestra e seguono il formato generale mostrato nella Figura 2.1, anche se non si deve necessariamente usare ogni clausola in ogni istruzione:

Figura 2.1: Sintassi di una Funzione Analitica

1. `Function_name`: questa è la funzione di analisi scelta (ad esempio SUM, RANK, LEAD, ecc.).
2. `Expression`: questa rappresenta la colonna su cui si sta interrogando, oppure un'istruzione logica `CASE WHEN`. Viene lasciata vuota per alcune funzioni di numerazione come `RANK` e `ROW_NUMBER`.
3. `OVER`: determina la finestra o l'insieme di righe entro cui opererà la funzione. Se non viene specificata, verranno interrogate tutte le righe del dataset.
4. `PARTITION BY`: questa clausola è essenzialmente come un `GROUP BY`, esegue un'aggregazione all'interno delle righe selezionate dalla finestra.
5. `ORDER BY`: a volte sarà necessario ordinare le righe nella partizione. Questo è importante per le funzioni di navigazione e numerazione

in modo che la query sappia dove iniziare e dove finire. È possibile ordinare per più colonne, in ordine crescente (`ASC`) o decrescente (`DESC`).

6. `Frame_clause`: questa sezione consente di scegliere un sottoinsieme di righe all'interno della partizione. Utile se si desidera eseguire eseguire una query su un totale parziale o una media mobile.

2.2 Google Data Studio

Google Data Studio è uno strumento di *Data Visualization* gratuito che permette la realizzazione di documenti completamente personalizzabili, condivisibili e facili da gestire.
Nella tesi è stato utilizzato per creare *dashboard*, ovvero documenti che permettono di visualizzare in modo accattivante le analisi effettuate sui dati.
Gli aspetti che influiscono maggiormente sulla scelta di questo tool sono i seguenti:

- Gratuità: *Data Studio* è gratuito. Si tratta di un grande vantaggio perché consente a chiunque di poter utilizzare questo tool senza dover affrontare spese mensili.

- Connessione e Dinamicità: *Data Studio* crea un collegamento diretto con le piattaforme di dati, come per esempio *Google Analytics*. Grazie a ciò è possibile realizzare dashboard dinamiche. Difatti, una volta importati i dati in *Data Studio*, questi si aggiorneranno automaticamente ogni volta che vi sarà un aggiornamento alla sorgente dati stessa. Il concetto di dinamicità consiste proprio in questo: non vi è più la necessità di realizzare un nuovo report ogni volta che si vuole

prendere in esame una metrica o una dimensione diversa; è, infatti, sufficiente selezionarla affinché i nuovi dati siano subito visibili.

- Visualizzazione: attraverso la cosiddetta `Data Visualization`, cioè la visualizzazione dei dati, i report assumono un aspetto molto più comprensibile. Grazie agli elementi messi a disposizione da *Data Studio*, come grafici, fonts e colori, i dati vengono presentati come se si stesse raccontando una storia. Non a caso si parla di *Data Storytelling*, espressione usata per intendere un nuovo modo di esporre l'andamento di un business.

- Data blending: *Data Studio* è in grado anche di "mescolare" dati provenienti da diverse sorgenti dati.

- Condivisione: con *Data Studio* la condivisione dei file è decisamente semplice. L'autore del report può decidere il tipo di accesso dei propri collaboratori o clienti. Vi sono due modalità di accesso ai dati: "edit" e "view". Nel primo caso si ha la possibilità di intervenire sul report (qualsiasi modifica viene salvata automaticamente); mentre nel secondo solo di prenderne visione.

2.3 Google Sheets

Google Sheets è un programma per *fogli di calcolo* incluso come parte della suite per ufficio gratuita di Google. É disponibile come applicazione web, app mobile e applicazione desktop. In particolare, consente di creare e gestire i dati sotto forma di numeri, testo e grafica all'interno di una tabella. Le formule sono utilizzate per calcolare dei risultati e creare relazioni tra i singoli valori.
In questo contesto i *Fogli Google* sono stati utilizzati per l'analisi dei dati e

la realizzazione di grafici ad hoc.
Google Sheets è completamente online; questo significa che più utenti in possesso dei diritti di accesso possono lavorare contemporaneamente sul documento. Ogni cambiamento è visualizzato in tempo reale ed è presente una finestra di chat laterale che consente la comunicazione tra gli utenti.

2.4 Jupyter Notebook

Un *Jupyter Notebook* è un'applicazione web open source che consente di creare e condividere documenti che contengono sia codice eseguibile, equazioni, visualizzazioni sia testo narrativo.
Un regolare processo lavorativo in team è d'importanza fondamentale in molti settori. I tool di comunicazione e per l'organizzazione e la gestione delle fasi lavorative e dei dati di progetto sono diventati praticamente indispensabili. Esistono diverse applicazioni per la scienza e la simulazione dei dati che cercano di soddisfare queste esigenze. La soluzione *web-based* di *Jupyter Notebook* crea, ad esempio, un continuo ponte tra codice di programma e testo esplicativo, consentendo all'utente di scambiare e creare in tempo reale codici, equazioni, visualizzazioni e informazioni esplicative.
Un *Jupyter Notebook* permette la creazione e la condivisione di documenti web nel formato JSON, che seguono uno schema e una lista ordinata di celle input/output; queste celle possono contenere codice eseguibile, testi in markdown, formule matematiche ed equazioni o contenuti multimediali.

2.4.1 Utilità

Un *Jupyter Notebook* mette a disposizione un ambiente fatto su misura per le esigenze e il flusso di lavoro di scienza e simulazione dei dati. In una sola istanza, gli utenti possono scrivere, documentare ed eseguire codici,

visualizzare dati, eseguire calcoli ed esaminare i risultati corrispondenti. In particolare durante la fase di prototipo possono trarre beneficio dal fatto che ciascun codice può essere ospitato in celle indipendenti: così è possibile testare individualmente specifici blocchi di codice. Grazie ai numerosi kernel aggiuntivi, *Jupyter* non si limita a Python per quanto riguarda il linguaggio di programmazione e ciò significa più flessibilità al momento della codifica e dell'analisi.
Tra gli scopi d'utilizzo più importanti dei *Jupyter Notebook* si possono menzionare:

- Pulizia dei dati: differenziazione tra dati importanti e meno importanti nell'analisi dei *big data*.
- Modellazione statistica: metodo matematico per determinare la stimata probabilità di distribuzione di una determinata caratteristica.
- Creazione e training di modelli di ML: progettazione, programmazione e training di modelli basati sul ML.
- Visualizzazione dati: rappresentazione grafica di dati per spiegare modelli, tendenze, dipendenze, ecc.

2.4.2 Google Colaboratory (Colab)

In questa tesi è stato utilizzato il servizio *Colaboratory* (abbreviato "*Colab*") di *Google*, ovvero una piattaforma online che offre un servizio di cloud hosting di *Jupyter Notebooks*, con il supporto a GPU.
Google Colab permette di scrivere ed eseguire codice Python nel proprio browser con i seguenti vantaggi:

- Nessuna configurazione necessaria;

- Accesso gratuito alle GPU;
- Condivisione semplificata.

Questo servizio è stato utilizzato per la realizzazione del codice necessario per l'esecuzione degli algoritmi di apprendimento automatico sui dati in esame: quest'ultimi sono stati prima puliti e trasformati in un formato efficiente e interpretabile dai modelli di ML utilizzati e poi dati in input ai modelli per il training e il testing.
Di seguito, le principali librerie Python utilizzate:

- NumPy[1]: NumPy (abbreviazione di *Numerical Python*) fornisce un'interfaccia efficiente per archiviare e operare su buffer di dati "densi". In un certo senso, gli array NumPy sono come il tipo integrato `list` di Python, ma forniscono operazioni di archiviazione dati molto più efficienti man mano che le dimensioni degli array aumentano. Gli array NumPy costituiscono il nucleo di quasi tutto l'ecosistema di strumenti di *data science* in `Python`.
- Pandas[2]: Pandas è una libreria costruita su NumPy e fornisce un'implementazione efficiente di un `DataFrame`. I `DataFrame` sono essenzialmente array multidimensionali con allegate le etichette di riga e colonna e spesso con tipi eterogenei e/o dati mancanti. Oltre a offrire una comoda interfaccia di archiviazione per i dati etichettati, Pandas implementa una serie di potenti operazioni sui dati familiari agli utenti sia di framework di database che di programmi di fogli di calcolo.
- Scikit-Learn[3]: esistono diverse librerie Python che forniscono implementazioni di un'ampia gamma di algoritmi di apprendimento au-

[1] https://numpy.org/
[2] https://pandas.pydata.org/
[3] https://scikit-learn.org/stable/

tomatico. Uno dei più noti è Scikit-Learn, una libreria che fornisce versioni efficienti di un gran numero di algoritmi comuni. Scikit-Learn è caratterizzato da un'API pulita, uniforme e semplificata, nonché da una documentazione online molto utile e completa. Un vantaggio di questa uniformità è che, una volta compreso l'uso di base e la sintassi di Scikit-Learn per un certo tipo di modello, passare a un nuovo modello o algoritmo è molto semplice.

- MatPlotLib[4]: Matplotlib è una libreria di visualizzazione dei dati multipiattaforma costruita su array NumPy e progettata per funzionare con il più ampio stack SciPy[5]. Una delle caratteristiche più importanti di Matplotlib è la sua capacità di funzionare bene con molti sistemi operativi e backend grafici. Matplotlib supporta dozzine di backend e tipi di output, il che significa che funziona indipendentemente dal sistema operativo che si sta utilizzando o dal formato di output che si desidera.

2.5 Google AutoML Tables

AutoML Tables è un servizio all'avanguardia di *Google* che permette di realizzare modelli di ML su dati strutturati. Nella pratica è un servizio che in base ai dati fornitigli in input, cerca e applica automaticamente il modello di ML migliore.
Nel progetto di tesi è stato utilizzato per realizzare un modello che permettesse di prevedere la percentuale di rischio di ogni record, per poi confrontarne le relative performance con il modello *Random Forest* realizzato in Python.

[4]https://matplotlib.org/
[5]https://www.scipy.org/

Google AutoML Tables, promette di "*cercare e trovare automaticamente, nello zoo dei modelli Google per dati strutturati, il modello migliore per le tue esigenze, che vanno da modelli di regressione lineare/logistica per dataset più semplici, a modelli di deep learning avanzati per dataset più grandi e complessi.*" (Google AutoML Table, 2019)
Inoltre, Google afferma anche che "*AutoML Tables automatizza l'ingegnerizzazione delle features su una vasta gamma di primitive di dati tabellari - come numeri, classi, stringhe, timestamp e liste - e aiuta anche a rilevare e gestire valori mancanti, valori anomali e altri problemi comuni sui dati.*" (Google AutoML Table, 2019)
In un primo momento, *AutoML* consente agli utenti di caricare i propri dati e fornisce informazioni sui dati mancanti, la correlazione, la cardinalità e la distribuzione delle *features*. Successivamente, esegue attività automatiche di ingegnerizzazione delle *features* come la standardizzazione e la normalizzazione, la creazione di *one-hot encoding* e di *embedding* per gli attributi categorici, l'elaborazione di base per le *features* di testo e l'estrazione delle funzionalità relative a data e ora dalle colonne di tipo timestamp. Nella fase successiva, *AutoML* inizia l'addestramento del modello utilizzando diversi algoritmi contemporaneamente, consentendo quindi il confronto delle prestazioni e l'identificazione del modello migliore.
Gli algoritmi inclusi nell'architettura di AutoML sono: *Linear, Feedforward Deep Neural Network, Gradient Boosting Decision Tree, AdaNet* e un mix delle varie architetture precedenti.

2.6 Google Dataflow

Google Dataflow è un servizio gestito di Google per l'esecuzione di un'ampia gamma di pattern di elaborazione parallela dei dati. *Dataflow* permette di

distribuire le pipeline di elaborazione dati sia in batch che in streaming.
Apache Beam SDK è un modello di programmazione open source che consente di sviluppare pipeline sia in batch che in streaming: è quindi possibile creare le pipeline di elaborazione dati con un programma *Apache Beam* e poi eseguirle sul servizio *Dataflow*.

2.6.1 Batch and Streaming

La principale differenza tra elaborazione batch e streaming è il tipo di sorgente dei dati di input: quando il tuo set di dati è limitato (anche se è enorme in termini di dimensioni) e non viene aggiornato durante il periodo di elaborazione, è probabile che si debba utilizzare una pipeline in batch. La sorgente di input, in questo caso, può essere qualsiasi cosa: da un file, a tabelle di database, a oggetti negli object storage, ecc.
Con l'elaborazione batch, si assume che i dati di input siano immutabili durante tutto il tempo di elaborazione e che il numero dei record sia costante: questo concetto è molto importante da tenere a mente perché anche con i file si può avere un flusso di dati illimitato, ovvero quando i file vengono continuamente modificati o aggiunti. In questo caso, per lavorare con questo tipo di dati si deve applicare un approccio di streaming. Quindi:

- se i dati sono limitati e immutabili, si possono sviluppare pipeline in batch;
- se i dati sono illimitati (in continuo arrivo) e/o mutabili, allora si devono sviluppare pipeline in streaming.

Alcuni esempi di sorgenti di dati illimitati potrebbero essere: sistemi di messaggistica (come Apache Kafka), nuovi file in una directory (log di server web) o altri sistemi che raccolgono dati in tempo reale (come i sensori IoT). Il tema comune tra tutte queste fonti è che si devono sempre aspettare nuovi

dati. Ovviamente, è possibile dividere i nostri dati in batch (per tempo o per dimensione dei dati) ed elaborare ogni sottoinsieme (split) in modo batch, ma sarebbe abbastanza difficile applicare alcune funzioni su tutto il dataset consumato e creare l'intera pipeline. Fortunatamente, ci sono diversi motori di streaming che consentono di affrontare facilmente questo tipo di elaborazione dati: uno di questi è *Google DataFlow*.

2.6.2 Apache Beam

Il modello di programmazione *Apache Beam* semplifica la meccanica dell'elaborazione dei dati su larga scala. Infatti, utilizzando uno dei suoi SDK, si può costruire un job che definisce la pipeline e poi utilizzare uno dei backend di elaborazione distribuita supportati da *Apache Beam SDK* (come *Dataflow*) che eseguirà la pipeline. Questo modello consente di concentrarsi sulla composizione logica del job di elaborazione dei dati, piuttosto che sull'orchestrazione fisica dell'elaborazione parallela. Ci si può concentrare su *cosa* il job deve fare, anziché concentrarsi su *come* sarà esattamente eseguito quel job.
Il modello *Apache Beam* fornisce astrazioni utili che isolano lo sviluppatore dai dettagli di basso livello dell'elaborazione distribuita (come il coordinamento di singoli *workers*, lo *sharding dei dataset* e altre attività simili).

- Con *sharding* si intende la tecnica per la creazione di un database distribuito (anche detto sharded custer) in cui ogni nodo contiene una porzione del database.

Dataflow gestisce completamente questi dettagli di basso livello.
Data la grande mole di dati, questo modello di elaborazione è risultato utile per la costruzione della *Ground Truth* (3.3).

Capitolo 3

Predizione dell'abbandono in una catena di palestre

3.1 Il problema dell'abbandono

L'*abbandono dei clienti* (*customer churn* o *customer attrition*) è la tendenza dei clienti ad abbandonare un'azienda/marchio e smettere quindi di essere un cliente pagante di una particolare attività. La percentuale di clienti che interrompono l'utilizzo dei prodotti o dei servizi di un'azienda in un determinato periodo di tempo è chiamata *tasso di abbandono dei clienti* (*customer churn rate*).
Il *customer churn rate* è un indicatore di prestazione (KPI) che le aziende devono monitorare per assicurarsi di prendere decisioni strategiche corrette. Innanzitutto, la maggior parte delle aziende investe molte risorse nell'acquisizione di nuovi clienti; e se i nuovi clienti non rimangono fedeli all'azienda abbastanza a lungo, potrebbe essere che alcune spese di acquisizione rappresentino in realtà un ROI (*Return Of Investiment*) negativo per l'azienda. Un altro importante vantaggio derivante dal monitoraggio del tasso di ab-

bandono è che se questo è molto alto, potrebbe indicare che vi sia un problema nell'offerta dell'azienda; potrebbe indicare un fallimento in aspetti come la *customer experience*, il prodotto o servizio venduto o il modo in cui vengono gestite le relazioni con i clienti.

Prevedere con successo l'abbandono dei clienti e prevenirlo in modo proattivo rappresenta un'aggiuntiva, potenziale fonte di entrate per la maggior parte delle aziende. Infatti, non solo è più difficile e costoso acquisire un nuovo cliente piuttosto che mantenere un cliente esistente, ma gli esperti di marketing di un'azienda hanno già una grande quantità di dati sui loro clienti attuali, che possono essere analizzati per determinare facilmente i messaggi e le offerte più efficaci da destinare a ciascun cliente.

Le aziende che monitorano costantemente il modo in cui le persone interagiscono con i loro prodotti/servizi, che incoraggiano gli utenti a condividere opinioni e che risolvono prontamente i loro problemi, hanno maggiori opportunità di mantenere relazioni reciprocamente vantaggiose con i clienti.

Inoltre, un'azienda che raccoglie i dati dei propri utenti da un po' di tempo, ha la possibilità di utilizzarli per identificare modelli di comportamento di potenziali *churner* (clienti che stanno per abbandonare), segmentare quest'ultimi e intraprendere azioni appropriate per riconquistare la loro fiducia ed evitare quindi il loro abbandono.

Questo approccio proattivo alla gestione del *customer churn rate* si basa sull'*analisi predittiva*; che implica la previsione della probabilità di risultati, eventi o valori futuri analizzando i dati attuali e storici. L'*analisi predittiva* utilizza varie tecniche statistiche e di apprendimento automatico.

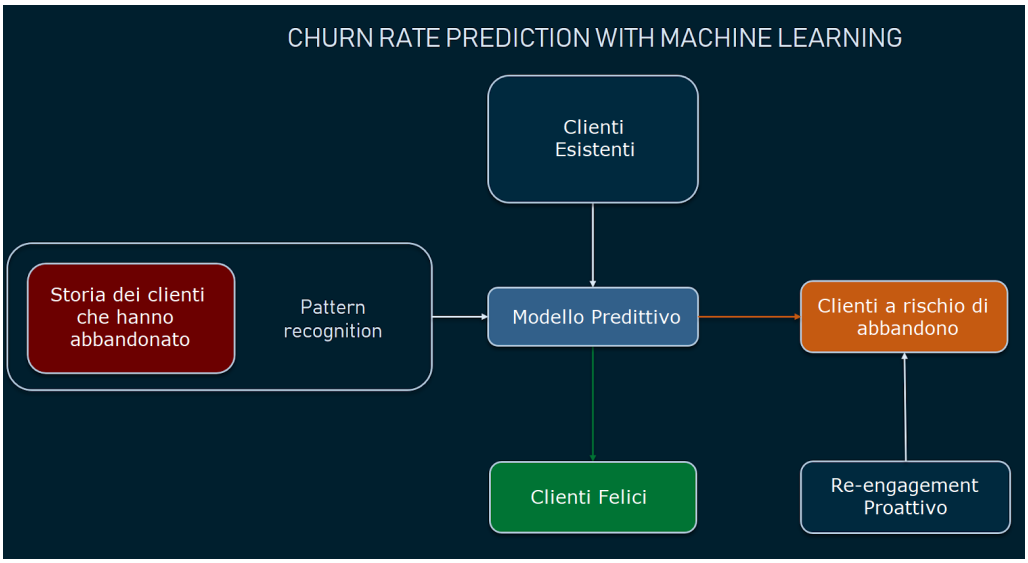


Figura 3.1: Esempio di architettura di un sistema di previsione del tasso di abbandono.

Il capo del dipartimento di analisi dei dati dell'azienda ScienceSoft Alex Bekker sottolinea l'importanza del *ML* per una gestione proattiva del *churn*: *"Per quanto riguarda l'identificazione di potenziali churner, gli algoritmi di ML possono fare un ottimo lavoro in questo contesto. Rivelano alcuni pattern di comportamento comuni in quei clienti che hanno già lasciato l'azienda. Quindi, gli algoritmi di ML confrontano il comportamento dei clienti attuali rispetto a tali pattern, segnalando i clienti che con una certa probabilità possono essere futuri churner"*.
Come per qualsiasi attività di apprendimento automatico, a seconda dell'obiettivo, gli esperti definiscono quali dati sono necessari. Nel caso della previsione dell'abbandono, bisogna identificare i dati che permettono di cogliere il comportamento degli utenti e la loro soddisfazione verso l'azienda. Una volta che i dati sono stati selezionati, questi vengono preparati, preelaborati e trasformati nella forma adatta alla creazione di modelli di ML.

Un'altra parte significativa del lavoro consiste nel trovare le tecniche giuste per addestrare i modelli, eseguire il *tuning* dei modelli e selezionare i migliori. Scelto il modello che effettua le previsioni più accurate, può essere messo in produzione.
Il *Churn Prediction Problem* è un problema di classificazione; infatti l'obiettivo del sistema è quello di determinare a quale classe appartiene un punto dati (un cliente, nel nostro caso). I dati storici utilizzati dovranno essere etichettati con variabili target predefinite (churner/no-churner), che alla fine sono quelle che il modello dovrà prevedere.
In particolare, alla fine di questo processo di classificazione, l'azienda potrà rispondere alla seguente domanda: *"Questo cliente quanto è probabile che abbandoni?"*.
La fase di preparazione dei dati è seguita dalla *progettazione, estrazione e selezione delle feature*; la *feature engineering* è una delle fasi più importanti del processo in quanto consiste nel creare una serie di attributi (le *feature* in input agli algoritmi di ML) che rappresentano i vari comportamenti relativi al coinvolgimento del cliente col servizio/prodotto offerto dall'azienda.
Le *feature* sono caratteristiche misurabili dei dati in input che un modello di ML prende in considerazione per prevedere i risultati; che nel caso di un *churn prediction problem* sono le probabilità di abbandono.

Segmentazione della clientela

In certi casi può essere utile suddividere i clienti in sottogruppi in base alla fase del ciclo di vita, alle esigenze, al livello di coinvolgimento, al valore monetario e ad altre informazioni di base. Questo perché i clienti appartenenti ad una specifica categoria possono avere modelli di comportamento diversi rispetto a quelli dei clienti appartenenti ad un'altra categoria. L'accuratezza della previsione può aumentare utilizzando modelli di ML addestrati specificamente su dataset che rappresentano ogni segmento (categoria) di

clienti.

Finestra di osservazione e finestra di prestazione

L'analisi predittiva consiste nell'apprendere le relazioni tra le *osservazioni* effettuate durante un periodo temporale (*event history window*) che termina prima di un punto temporale specifico e le *previsioni* sul periodo temporale che inizia subito dopo (*performance window*). Il primo periodo è chiamato *finestra di osservazione* o *independent window* o *event history window*, il secondo periodo è chiamato *finestra di prestazione* o *dependent window* o *performance window*.

Figura 3.2: Finestra di osservazione e finestra di prestazione.

Bilanciare il tempo per le osservazioni e le previsioni è un compito difficile e dipende fortemente dall'ambito di *business* in cui si sta operando. Ad esempio, se una finestra di osservazione è di un mese, una finestra di performance per un cliente con un abbonamento annuale sarà di undici mesi. Tipicamente, creare una breve cronologia degli eventi e lunghe finestre di performance è più vantaggioso per le aziende: in quanto è necessario poco tempo per l'osservazione ed è disponibile molto tempo per il *re-engagement* del cliente.

Sfortunatamente non funziona sempre in questo modo; una breve cronologia degli eventi potrebbe non essere sufficiente per fare previsioni affidabili, quindi la sperimentazione con questi parametri può diventare un processo continuo, ripetitivo e con i suoi compromessi.
L'idea fondamentale consiste nel definire una finestra di osservazione che sia sufficientemente lunga per dare una previsione giustificata, ma avere comunque abbastanza tempo per affrontare il potenziale abbandono.
Le tecniche più popolari nella comunità di ricerca per la *churn prediction* prendendo in considerazione sia i modelli classici di ML (come *regressione logistica, alberi decisionali, Random Forest*, etc.) sia le *artificial neural networks* [4],[5],[6],[7],[8],[9],[10].

3.2 Predizione dell'abbandono nella realtà di Technogym

Questa sezione mostra una panoramica sul problema di *predizione dell'abbandono* associato alla realtà di Technogym.

3.2.1 Descrizione del problema

Technogym è l'azienda leader del fitness e del wellness che ha fatto evolvere il concetto edonistico di *fitness* in un vero e proprio stile di vita: il *Wellness*. Vivere "wellness" significa regolare attività fisica, corretta alimentazione e approccio mentale positivo.
La *mission* di Technogym è aiutare le persone a vivere meglio e lo fa con i migliori attrezzi da palestra, servizi, contenuti e programmi tecnologicamente connessi fra loro per far sì che le persone vivano il wellness ovunque si trovino. Oltre ad essere leader nella produzione di tutti gli attrezzi da

palestra, Technogym offre anche prodotti software che permettono la gestione informatizzata delle palestre, in relazione agli abbonamenti dei clienti, ai loro ingressi nella struttura, all'utilizzo delle varie attrezzature e dell'applicazione smartphone. In particolare, nel sistema di gestione più avanzato Technogym vuole permettere alle aziende di conoscere con un certo anticipo, la percentuale di rischio di abbandono di ogni utente della palestra. Questo valore percentuale è chiamato DOR; *Drop Out Risk*. Da questo termine deriva il nome dell'intero progetto. Il progetto DOR consiste quindi in un sistema di previsione del rischio di abbandono di ogni utente.
Technogym offre già un servizio di questo tipo basato su regole statiche che offre risultati accettabili ma è un sistema che non si adatta automaticamente al variare delle caratteristiche dei clienti nel tempo. Il desiderio dell'azienda è quello di migrare il sistema storicamente utilizzato ad un sistema basato su tecniche di ML, al fine di migliorare la qualità della previsione.

3.2.2 Assunzioni di progetto

Questo progetto si basa sull'ipotesi generale che "*una persona che sta per abbandonare si allena male, ossia con poca frequenza*".
Si presume che un cliente in procinto di abbandonare la palestra, diminuisca progressivamente le sue interazioni con essa; con interazioni si intendono sia le visite alla struttura sia i login all'applicazione mobile, le misurazioni biometriche, le sessioni a corpo libero e i workout eseguiti a casa attraverso l'applicazione. Al fine di poter valutare il comportamento degli utenti, questo deve essere rappresentato nei dati attraverso delle *features* formali.
La rappresentazione del progressivo interesse di un utente è stata effettuata calcolando, ad una certa data, la sua frequenza nelle ultime due settimane, nell'ultimo mese e negli ultimi due mesi. Quindi per ogni settimana vengono mostrate le frequenze nei vari periodi temporali, così da evidenziare un

comportamento costante, oppure in crescita o in diminuzione.
Oltre all'aumento o diminuzione della frequenza in valore assoluto, è stato scelto di mostrare anche la variazione di frequenza attraverso la *derivata*, cosicché potessero essere comparati anche utenti con frequenze molto diverse tra loro. Infatti, se di norma un utente frequenta la palestra *due* volte alla settimana avrà delle frequenze molto più basse rispetto ad un utente che, di norma, frequenta *sei* volte alla settimana. Se ci basassimo solo sulla frequenza, dicendo che "*gli utenti che hanno frequenza bassa sono a rischio di abbandono*", il primo utente sarebbe sempre a rischio; quando in realtà quella frequenza rappresenta il suo comportamento standard. Allo stesso modo, nel momento in cui l'utente con alta frequenza diminuisce le sue interazioni della metà, la frequenza risultate potrebbe rimanere ancora alta; quando in realtà ha cambiato notevolmente il suo comportamento e ciò potrebbe essere sintomo di un possibile abbandono. Questa variazione di comportamento è catturata dalla *derivata*.
Le *features* risultanti da questi ragionamenti sono mostrate nella Sezione 4.2 del capitolo successivo.

3.2.3 Requisiti di business

Confrontandosi con i manager dell'azienda si sono individuati i seguenti requisiti di business:

- L'abbonamento minimo di una palestra ha la durata di *un mese* quindi come periodo di assenza minimo che identifica un *abbandono* sono state scelte le *quattro settimane*. Di conseguenza, tutte le assenze maggiori o uguali alle *quattro settimane* saranno considerate dei *dropout*.

- Il calcolo del DOR deve essere effettuato ogni Domenica, cosicché il Lunedì seguente i manager dell'area markenting hanno la possibilità di individuare gli utenti ad alto rischio di abbandono e intraprendere le opportune strategie di fidelizzazione.

- Per un manager è importante avere una previsione degli utenti a rischio con il massimo anticipo possibile. Si ritiene che una previsione affidabile possa essere ottenuta con, al massimo, 4 settimane di anticipo e si presume che, con mirate azioni di marketing, un mese sia sufficiente per riacquistare la fiducia dei clienti. Di conseguenza il DOR dovrà segnalare gli utenti con un mese di anticipo (4 settimane).

- In base alla percentuale di rischio associata ad ogni utente, si deve poter classificare gli utenti in tre categorie di rischio: 1) *Rischio Basso*; 2) *Rischio Medio*; 3) *Rischio Alto*.

- Gli utenti segnalati nelle classi di *Rischio Medio* e *Rischio Alto* devono essere complessivamente il 30% degli utenti totali.

- Per un manager è importante capire i fattori che influiscono maggiormente sulla scelta di un utente di abbandonare o meno la palestra. Quindi diventa fondamentale l'*explainability* della classificazione; capire perché il classificatore ha associato una certa probabilità di rischio all'utente.

3.2.4 Obiettivo dell'Applicazione

L'obiettivo del sistema, utilizzando il profilo degli utenti, è quello di prevedere fino ad un mese di anticipo, coloro che abbandoneranno la palestra, ovvero coloro che faranno *dropout*.
Per fare questo, nel rispetto dei requisiti di business, è necessario che la

storia delle interazioni con una palestra di ogni utente sia rappresentata settimanalmente. Inoltre, è necessario che i "buchi di assenze" maggiori o uguali a *quattro* settimane siano evidenziati come *dropout*. Questo è reso possibile etichettando le quattro settimane precedenti al *dropout* come *esempi di comportamento scorretto*, in quanto hanno portato all'abbandono.

Nella Figura 3.3 è possibile vedere la rappresentazione grafica della storia di un singolo utente.

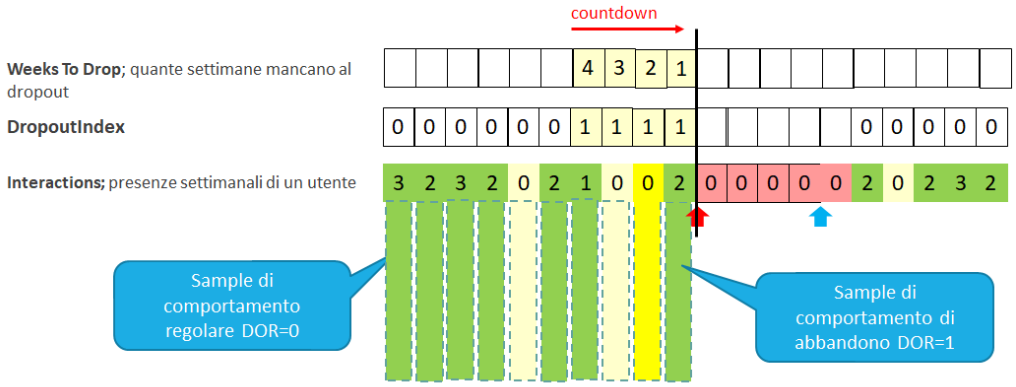


Figura 3.3: Storia delle interazioni settimanali di un utente; ogni settimana è etichettata come *esempio di comportamento regolare*, o come *esempio di comportamento di abbandono*.

3.3 Costruzione della Ground Truth

Con il termine *Ground Truth*, ossia "verità fondamentale", ci si riferisce alle informazioni fornite dall'osservazione diretta (*evidenza empirica*) diversamente dalle informazioni fornite dall'*inferenza*.

La *ground truth* è concettualmente la conoscenza della verità riguardante una domanda specifica; è il risultato atteso ideale [1]. Viene utilizzata nei

modelli statistici per dimostrare o confutare certe ipotesi.
Il termine "ground truthing" si riferisce al processo di raccolta dei dati oggettivi (dimostrabili) che permettono di rispondere alle domande del problema in esame. Nel caso di previsione dell'abbandono le domande alle quali si vuole rispondere sono:

- "Chi sono gli utenti che abbandonano (fanno *dropout*)?"
- "Come si comportano gli utenti che fanno *dropout*?"
- "Si possono riconoscere dei comportamenti simili tra gli utenti che fanno *dropout*?"
- "Come si differenzia il comportamento degli utenti che fanno *dropout* rispetto al comportamento degli utenti che non lo fanno?"

Di conseguenza la *ground truth* dovrà rappresentare la storia dei clienti e per ognuno deve essere specificato se e quando ha abbandonato. In questo modo sarà possibile creare una "verità" sui *dropout* (abbandoni) e i *no-dropout* (non abbandoni) dei clienti.
La *ground truth* deve rispettare i requisiti di business e sue eventuali imprecisioni saranno correlate alle imprecisioni nelle previsioni di *dropout*/*no-dropout* risultanti.
Nel caso di studio in esame, dato che le strategie di fidelizzazione da adottare con gli utenti sono valutate settimanalmente, è stato scelto di creare una "verità fondamentale" che rappresenti le informazioni degli utenti raggruppate per *StatusDate*, ossia per settimana. In particolare, una *StatusDate* è la settimana che va da un Lunedì alla Domenica successiva ed è identificata dalla data di quest'ultima, com'è possibile vedere dalla Figura 3.4.

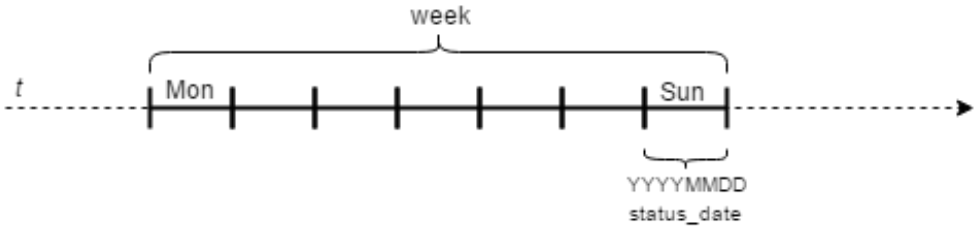


Figura 3.4: Rappresentazione grafica del concetto di *StatusDate*.

Nel processo di costruzione della *Ground Truth* vengono calcolate, per ogni *StatusDate*, le informazioni mostrate nella Figura 3.5.

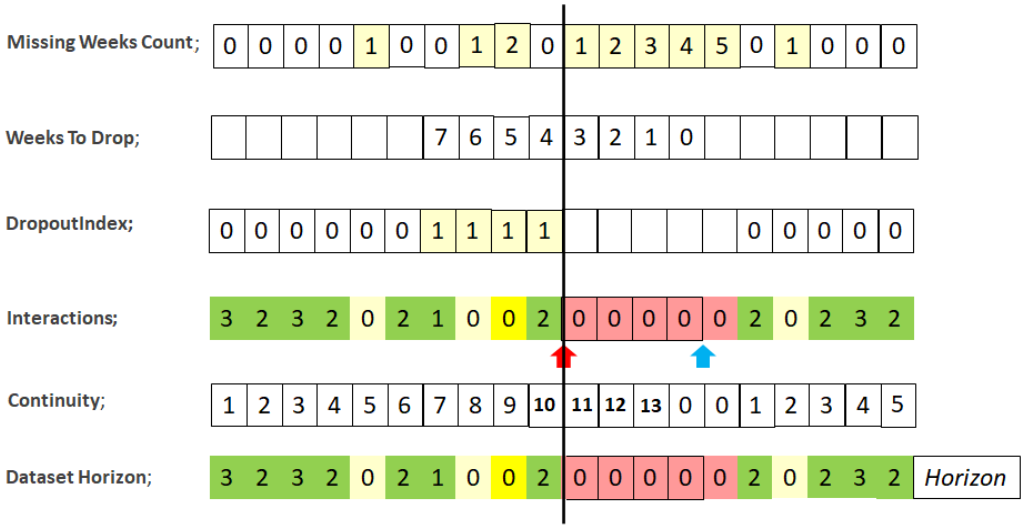


Figura 3.5: Rappresentazione grafica della storia di un utente con una palestra.

In particolare, nella Figura 3.5 sono mostrate:

- Le *StatusDate*, rappresentate dalle singole celle colorate, ovvero le settimane dalla prima all'ultima interazione dell'utente con la palestra. Ogni *StatusDate* mostra la somma delle interazioni che l'utente ha

avuto con la palestra nella settimana corrispondente. Sono presenti anche le settimane in cui l'utente non è stato presente.

- Le *Interactions*, rappresentate dal valore all'interno delle celle, costituiscono la storia settimanale dell'utente.

- Le *WeeksToDropout* che rappresentano le settimane di interesse, ovvero le *quattro* settimane precedenti al *dropout*. Il valore nella cella *WeeksToDropout* è un countdown: specifica quante settimane mancano al *dropout*. Un evento di *dropout*, è definito dalle regole di *business* come *"quattro settimane consecutive di assenza di interazioni con una palestra da parte di un utente"*. Nella Figura 3.5 le quattro *StatusDate* segnate di *rosso*, nelle quali l'utente è stato assente (zero interazioni) corrispondono al *dropout*. Intaffi, le settimane di interesse ai manager dell'azienda non sono le quattro che definiscono l'abbandono, ma le quattro precedenti perché si vuole segnalare il futuro abbandono nelle quattro settimane prima.

  - A tal fine si devono etichettare come esempi di comportamento scorretto le quattro settimane che precedono il *dropout*: le settimane con *WeeksToDropout=7/6/5/4* sono etichettate in fase di addestramento con classe positiva *DropoutIndex=1*, perché identificano l'imminente abbandono.

- L'etichetta di classe *DropoutIndex* che descrive ogni settimana come "esempio di comportamento regolare" (*DropoutIndex=0*), o come "esempio di comportamento di abbandono" (*DropoutIndex=1*).

- Nella definizione della *ground truth* di ogni utente, è anche necessario stabilire una data "orizzonte", ossia una *StatusDate* limite/finale che permetta di verificare se a quella data l'utente è ancora attivo oppure

ha abbandonato. Questa *StatusDate* finale viene chiamata "Horizon". Nella Figura 3.5, l'ultima interazione dell'utente è avvenuta la settimana prima dell'*Horizon StatusDate*, ciò significa che a quella data l'utente è ancora attivo. Diversamente, nell'esempio della Figura 3.6, l'ultima interazione dell'utente è avvenuta quattro settimane prima dell'*Horizon StatusDate*, questo significa che l'utente è assente da un mese e quindi ha fatto *dropout*.

- Un altro concetto che viene definito durante la costruzione della *ground truth* è quello di *Continuity*; ossia un valore che rappresenta la "continuità" dell'utente, necessario per rispondere alla domanda *"Da quanto viene in palestra il cliente X?"*.

  – La *Continuity* è incrementata ogni settimana dalla prima interazione;

  – Viene incrementata anche nelle settimane in cui l'utente è assente;

  – Viene azzerata quando l'utente fa dropout.

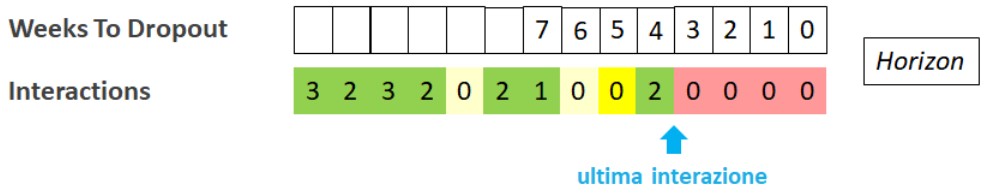


Figura 3.6: Rappresentazione concettuale dell'*Horizon StatusDate* e delle ultime interazioni dell'utente prima dell'*orizzonte*.

3.4 Dataset

Alla fine del processo di generazione della *Ground Truth* si avrà a disposizione un dataset che include un record per ogni *StatusDate*, per ogni utente e per ogni palestra fino alla data di *dropout*. Dopo il *dropout* nessun record viene incluso finché l'utente non ritorna in palestra.
Ogni record sarà etichettato con *0* o *1* a seconda che si tratti, rispettivamente, di una settimana regolare o di una delle quattro settimane precedenti al *dropout*. Di seguito una figura esemplificativa di questo comportamento:

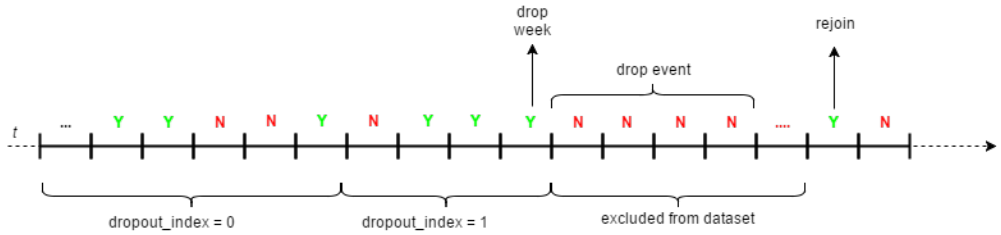


Figura 3.7: Rappresentazione concettuale della *ground truth* di un utente in una palestra.

Il *DropoutIndex* è considerato "vero" (= 1) durante le quattro settimane precedenti la data di abbandono ed è impostato come "falso" (= 0) su tutti i record rimanenti. Questo modello può verificarsi più volte per la stessa chiave *utente-palestra* nell'arco del dataset, poiché gli utenti possono abbandonare e ricongiungersi a una singola struttura molte volte.
Il dataset utilizzato è composto da tutte le interazioni che gli utenti hanno avuto con ogni palestra nei mesi che vanno da Gennaio a Dicembre dell'anno *X*.

3.5 DOR1

Il DOR1 è la soluzione storicamente utilizzata da Technogym e definisce il rischio di abbandono utilizzando delle regole statiche basate sulla *frequenza*, sull'*età* e sulla *memebership* dell'utente.
In particolare, il rischio di *dropout* si alza al diminuire dell'età dell'utente e al diminiuire della durata della sua membership.

3.5.1 Features

Ogni settimana viene fatto uno *snapshot* e si associano a *utente-palestra-settimana* le *features* seguenti:

- ***Age***: età dell'utente.
- ***Membership duration***: la *membership* rappresenta la fidelizzazione dell'utente con la palestra. La *membership duration* è calcolata considerando i giorni trascorsi dalla prima data di iscrizione storica dell'utente alla *StatusDate* in esame.
- ***Complete Weeks Of Membership***: le settimane complete di *membership*; quindi le settimane trascorse dalla data di iscrizione alla *StatusDate* in esame.
- ***Frequency***: frequenza sulle ultime otto settimane;

La *Frequency* è calcolata facendo il rapporto tra la somma delle visite dell'utente nelle ultime otto settimane (*InteractionsInLast56Days*) e le settimane complete di *membership* (*CompleteWeeksOfMembership*) limitate nel range [1..8]:

$$Frequency = \frac{InteractionsInLast56Days}{MAX(1,MIN(CompleteWeeksOfMembership,8))}$$

3.5.2 Calcolo del rischio di abbandono

Una volta determinate l'*età* dell'utente, la sua *membership* e la *frequenza nelle otto settimane* è possibile calcolare il DOR1 in una certa *StatusDate*. Il calcolo avviene utilizzando le regole specificate nel sistema sottostante:

$$
DOR1 = \begin{cases}
\text{IF (Age} < 25 \text{ AND Membership} < 60) \text{ THEN} \\
\quad 1.65 \cdot (3 - Frequency)/2; \\
\text{IF (Age} < 25 \text{ AND Membership} \geq 60) \text{ THEN} \\
\quad 1.65 \cdot (2 - Frequency)/2; \\
\text{IF (}25 \leq \text{Age} < 34 \text{ AND Membership} < 60) \text{ THEN} \\
\quad 1.30 \cdot (3 - Frequency)/2; \\
\text{IF (}25 \leq \text{Age} < 34 \text{ AND Membership} \geq 60) \text{ THEN} \\
\quad 1.30 \cdot (2 - Frequency)/2; \\
\text{IF (Age} \leq 34 \text{ AND Membership} < 60) \text{ THEN} \\
\quad 1.00 \cdot (3 - Frequency)/2; \\
\text{IF (Age} \leq 34 \text{ AND Membership} \geq 60) \text{ THEN} \\
\quad 1.00 \cdot (2 - Frequency)/2;
\end{cases}
$$

Nella Figura 3.8 è possibile vedere la distribuzione del DOR1 in base alle regole sopra citate.
Il valore restituito dal calcolo del DOR1 identifica un indice numerico di rischio per ogni utente, da cui tramite delle soglie è derivata una classificazione in:

- Rischio basso: $DOR1 < 0.35$
- Rischio medio: $0.35 \geq DOR1 < 0.7$

- Rischio alto: $0.7 \geq DOR1$

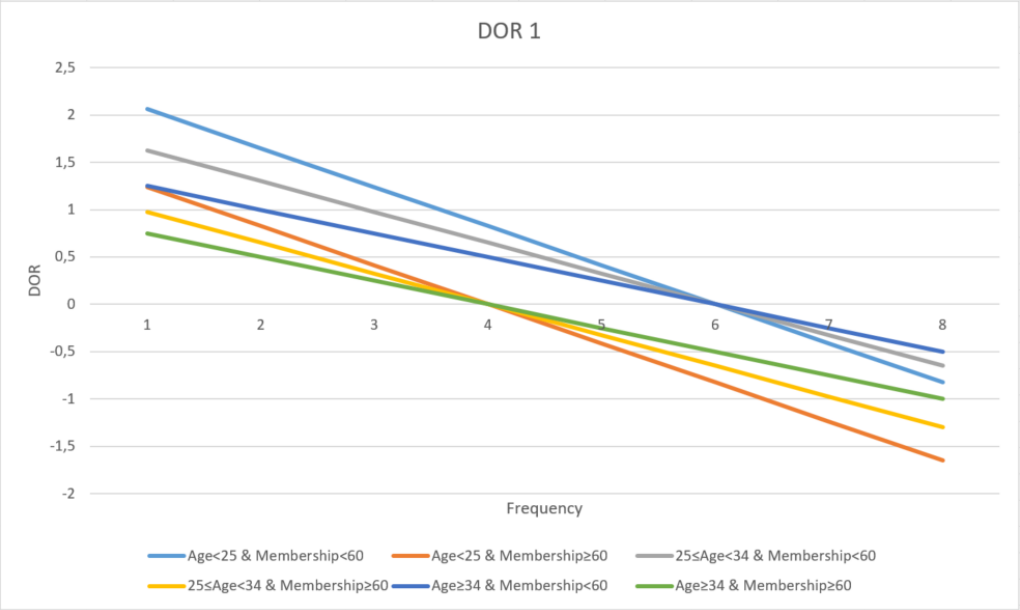


Figura 3.8: Rappresentazione grafica del calcolo del DOR1.

3.5.3 Performance

In questa sezione sono mostrate le performance del DOR1 nel rispetto del requisito di business che specifica di segnalare come "utenti a rischio di abbandono" il 30% dei dati. Nel 30% dei dati segnalati sono considerati sia i record di rischio medio, sia quelli di rischio alto. A tal fine, per il DOR1, la soglia di allarme che classifica i record come *dropout* è impostata a 35; ciò significa che tutti i record che hanno $DOR1 \geq 35$ rientreranno nella classe degli utenti a rischio di abbandono che sarà successivamente suddivisa in due sotto-classi: "Rischio Medio" e "Rischio Alto".
Le performance del DOR1, così come quelle dei modelli di ML utilizzati successivamente, sono calcolare tramite *matrice di confusione*: la matrice

di confusione è una tabella in cui ogni riga indica una classe prevista e ogni colonna corrisponde alla classe osservata. Le celle della matrice indicano la frequenza con cui ogni previsione di classificazione coincide con ogni classe osservata. Questa matrice è utile per determinare gli errori di classificazione che si verificano con maggior frequenza, ovvero quali classi vengono "confuse" spesso tra loro.
Nella Figura 3.9 è possibile vedere che il DOR1 ha una *precision* del 22%; ciò significa che di tutti i record segnalati come *dropout*, il 22% sono dei "veri" abbandoni. Mentre la *recall* arriva al 55%; ciò vuol dire che di tutti i "veri" *dropout*, il sistema ne individua il 55%.
La soluzione DOR1 calcola l'indice di rischio in base ad una formula fissa che non si adatta alla realtà mostrata dai dati e non evolve al variare dei comportamenti degli utenti nel tempo. Da questi limiti è nata l'esigenza di utilizzare tecniche e algoritmi di ML che riuscissero a catturare la variabilità dei comportamenti degli utenti in relazione, ad esempio, al mese dell'anno in cui si sta facendo l'analisi, al sesso dell'utente, al luogo in cui si trova la palestra e via dicendo.

DOR1, threshold = 35

Precision	TP/(TP + FP)	22%
Recall	TP/(TP + FN)	55%
Accuracy	(TP + TN)/Total	72%

N. Records		PREDICTED		
		0	1	Total
EXPECTED	0	TN 2 987 116	FP 1063 336	4 050 452
	1	FN 241 551	TP 294 419	535 970
	Total	3 228 667	1 357 755	4 586 422
	Signal	**70%**	**30%**	

Figura 3.9: Matrice di confusione DOR1.

Capitolo 4

Verifica della correttezza dei dati e del contenuto informativo

Il sistema DOR1 è utilizzato dall'azienda già da molto tempo e nel corso degli anni si sono visti vari tentativi per migrare il sistema ad uno basato su tecniche di apprendimento automatico. Quindi, certe ipotesi di progetto e certe *features* erano già state studiate, valutate e definite. Di conseguenza, si è resa necessaria un'analisi approfondita delle procedure di raccolta dati e di calcolo delle *features* ereditate dagli studi passati, al fine di rilevare eventuali errori e/o imprecisioni.

4.1 Dati

4.1.1 Verifica della correttezza e della qualità dei dati

La corretta costruzione della *Ground Truth* è un requisito fondamentale per tutti i sistemi di apprendimento automatico. Questo perché ogni infor-

mazione presente nel dataset sarà utilizzata dagli algoritmi come "verità"; dunque, se queste informazioni fossero errate, allora gli algoritmi apprenderebbero verità sbagliate. Il primo passo, quindi, per verificare la correttezza dei dati è stato quello di controllare la costruzione della *Ground Truth* che veniva costruita per mezzo di una *pipeline dataflow* realizzata tramite il framework *Apache Beam*. Di conseguenza si sono evidenziati e risolti alcuni *bug* nella *pipeline*.
Come viene spiegato nella sezione *Costruzione della Ground Truth* 3.3 del capitolo precedente, la *pipeline dataflow* esegue per ogni *utente-palestra* quattro operazioni fondamentali:

1. Inserisce le settimane di assenza;

2. Etichetta le settimane precedenti al *dropout* come esempi di comportamento negativo.

3. Inserisce le settimane di assenza dall'ultima interazione dell'utente all'*Horizon statusdate*, per capire se l'utente è ancora attivo oppure ha abbandonato.

4. Calcola la *Continuity* dell'utente.

Il primo errore è stato individuato nel calcolo delle settimane di assenza prima dell'*Horizon*: se queste erano maggiori di quattro, veniva aggiunto un record con *WeeksToDropout=-1*, provocando l'inserimento di record con valori scorretti nel dataset.

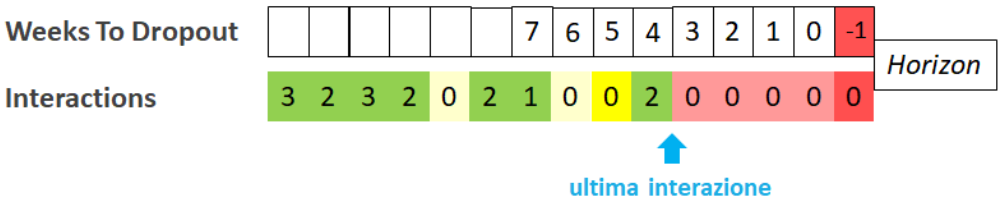


Figura 4.1: Inserimento di una StatusDate non necessaria con un valore scorretto nella colonna *WeeksToDropout*.

Un secondo errore riguardava il calcolo della *Continuity*: questa veniva azzerata nella settimana successiva al *dropout* qualora l'utente fosse stato ancora assente. Nel caso in cui l'utente fosse mancato per quattro settimane esatte, ritornando alla quinta settimana, la *Continuity* non veniva azzerata, producendo di conseguenza un'inconsistenza nei dati.

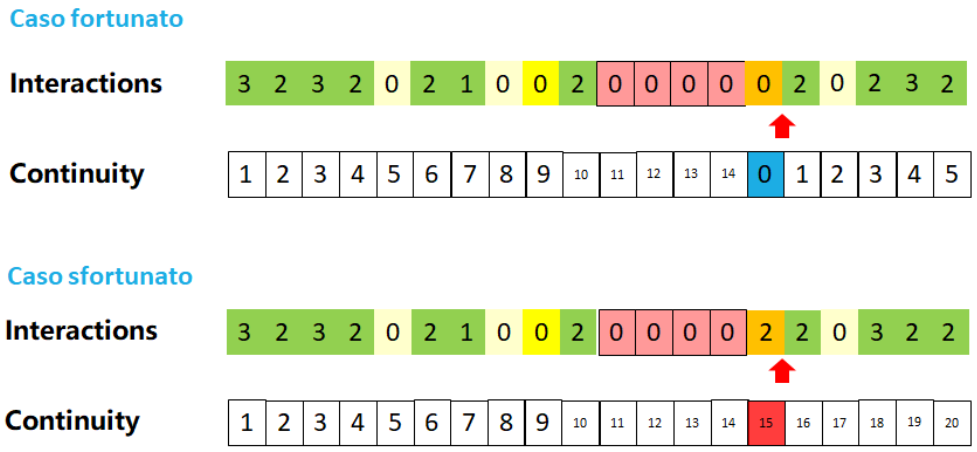


Figura 4.2: Calcolo scorretto della *Continuity*.

Seguendo la logica dietro il concetto di *Continuity* questa deve essere azzerata alla settimana che determina il dropout; quindi nel nostro caso alla

quarta settimana di assenza. La correzione è mostrata nella Figura 4.3.

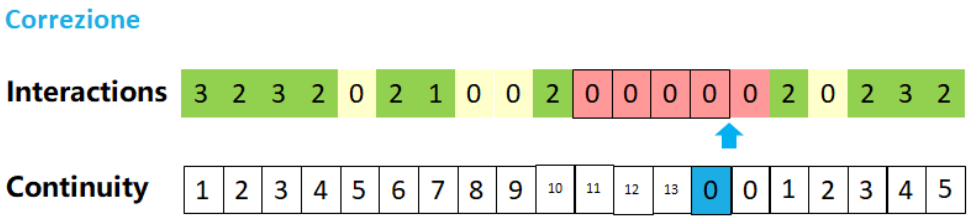


Figura 4.3: Calcolo corretto della *Continuity*.

Gli errori sopra citati sono stati corretti lato codice nel *dataflow* che costruisce la *Ground Truth*.
Un altro problema si verificava nella situazione in cui un utente faceva due *dropout* consecutivi e non presenziava per almeno quattro settimane tra il primo *dropout* e il successivo: in questo caso, dato che il sistema etichetta come esempi di comportamento negativo le *quattro* settimane precedenti al *dropout*, venivano etichettate come settimane a *DropoutIndex=1* anche le settimane in cui l'utente era stato assente, dato che le settimane di presenza erano inferiori a quattro. Si può capire meglio questo concetto dalla Figura 4.4.
Per superare questo problema e per cercare di fornire dati più statisticamente corretti agli algoritmi di classificazione, è stato deciso di selezionare come *sample* solo i record che considerano almeno un mese di interazioni.

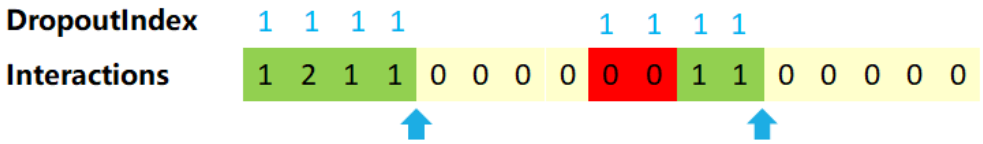


Figura 4.4: Etichettamento errato nel caso di due *dropout* consecutivi, senza almeno un mese di interazioni tra l'uno e l'altro.

L'ultimo errore riscontrato riguardava alcuni dati duplicati nel dataset finale. Nello specifico, veniva utilizzata una query che eseguiva l'`INNER JOIN` di due tabelle rispetto ad un attributo che in una di esse compariva più volte. L'errore è stato eliminato facilmente correggendo la query, questo ha portato ad un miglioramento in termini di efficienza di esecuzione in quanto vi è stata un'importante riduzione del numero di record duplicati.

4.1.2 Processo di preparazione dei dati

Di seguito viene illustrato il processo di preparazione dei dati mostrando i vari passaggi (che a basso livello sono implementati tramite query e pipeline dataflow) che permettono di ottenere i due dataset finali: uno per il training del modello e uno per il testing.
Nella Figura 4.5 sono descritti i vari passaggi cosicché si possano capire gli step effettuati per ottenere i dati nel formato corretto e per la costruzione delle *features*. L'elenco sottostante spiega il processo, passaggio per passaggio.

1. Il primo step è quello che permette di selezionare per ogni *Utente-Palestra* e per tutte le *StatusDate* di interesse, le *Visite/Interazioni* fino a 8 settimane prima. La query utilizzata permette di specificare le due *StatusDate* che definiscono l'arco temporale da prendere in esame:

ad esempio, specificando come prima *StatusDate* la data "2018-01-07" e come ultima *StatusDate* la data "2019-12-29"", saranno cosiderate tutte le *StatusDate* che stanno all'interno di quell'arco temporale.

2. Il secondo step conta il numero di interazioni effettuate in ogni *StatusDate*; ovvero le interazioni effettuate nella settimana corrispondente alla *StatusDate* in esame.

3. Il terzo step che prevede l'esecuzione del codice dataflow prendendo in input i dati ottenuti dallo step precedente; consiste nella costruzione della *Ground Truth*.

4. Il quarto step permette di calcolare la *First Membership Date* e la *Last Membership Date*; ossia la prima data di iscrizione storica dell'utente nella palestra e l'ultima data di rinnovo dell'iscrizione.

5. Il quinto step prende in input i risultati dei due passaggi precedenti ed esegue il calcolo delle *features*.

6. Il sesto step proietta le *features* nel formato corretto per essere interpretate dal modello di ML; esegue il *one hot encoding* degli attributi categorici.

7. Il settimo step crea una colonna *DataSplit* che permetterà di suddividere il dataset in in tre sottoinsiemi (*TRAIN*, *VALIDATE*, *TEST*) ed esegue il campionamento random dei dati con *DropoutIndex=0*.

8. L'ottavo e il nono step rappresentano l'output del processo:

   - un dataset con campionamento per il training del modello;
   - un testset senza campionamento per la validazione del modello.

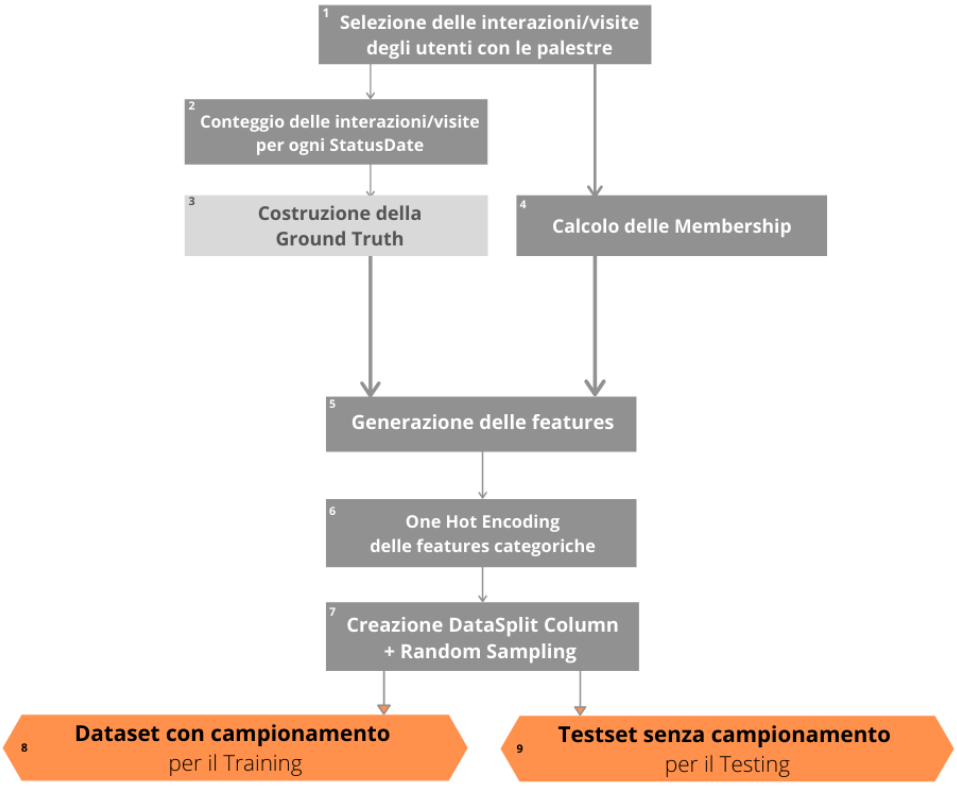


Figura 4.5: Processo di costruzione dei dataset utilizzati per il training dei modelli e per la loro validazione.

4.2 Features

Di seguito le *featuers* sulle quali si baserà l'apprendimento dei modelli di ML:

- ***InteractionsInLast7Days***, ***InteractionsInLast14Days***, ***InteractionsInLast28Days***, ***InteractionsInLast56Days***: numero di interazioni avvenute negli ultimi 7, 14, 28, 56 giorni.

Oltre agli ingressi effettivi vengono conteggiate tutte le interazioni che mostrano l'interesse dell'utente verso la palestra.

- ***FrequencyOn14Days, FrequencyOn28Days, FrequencyOn56Days***: frequenza sulle ultime due/quattro/otto settimane dalla *StatusDate* in esame. Il calcolo è mostrato nella Figura 4.6; come si può notare, il numero di interazioni avvenute negli ultimi X giorni è diviso per il minimo tra il periodo X e la *Continuity*. L'utilizzo della *Continuity* serve per le persone iscritte da meno di due mesi, il che non permetterebbe di calcolare la frequenza nel periodo di 56 giorni.

- ***Derivate14, Derivate28, Derivate56***: derivata sulle ultime due/-quattro/otto settimane. Lo scopo di queste *features* è quello di mostrare la variazione della frequenza dell'utente nel periodo di tempo più corto rispetto al periodo di tempo più lungo. Il calcolo è mostrato nella Figura 4.6.

$$\text{freq}X = \frac{\#interaction\ last\ X\ days}{\min(X, ContinuousDaysOfPresence)}$$

$$\text{dev14} = \frac{freq14}{freq28} \qquad \text{dev28} = \frac{freq14}{freq56} \qquad \text{dev56} = \frac{freq28}{freq56}$$

Figura 4.6: Formula del calcolo della frequenza e della derivata.

- ***Moves14, Moves28, Moves56***: movimenti/esercizi effettuati nelle interazioni avvenute nelle ultime due/quattro/otto settimane;

A queste features sono state aggiunte informazioni generali sul sesso e sull'età dell'utente, sulla regione geografica, sul mese in esame e via dicendo,

con l'idea che potrebbero esistere pattern di comportamento dipendenti da questi dati. In particolare:

- ***Month***: mese a cui appartiene la *StatusDate* in esame;
- ***Age***: età dell'utente.
- ***Gender***: genere biologico dell'utente;
- ***Country***: nazione in cui si trova la palestra;
- ***Continent***: continente in cui si trova la palestra;

Infine, sono state definite delle *features* che permettono di classificare gli utenti tra quelli "di vecchia data" e quelli "più recenti", per vedere se esistono delle correlazioni tra gli abbandoni e la storia pregressa dell'utente che ne rappresenta tipicamente la fidelizzazione con la palestra. Allo scopo sono state definite le seguenti *features*:

- ***First membership duration***: giorni trascorsi dalla prima data storica di iscrizione. Equivale alla *Membership duration* utilizzata nel DOR1;
- ***Last membership duration***: giorni trascorsi dall'ultima data di iscrizione dell'utente;
- ***Continued weeks of presence***: settimane di presenza continuativa.

Nella sezione successiva si mostrano le analisi effettuate sui dati a disposizione, al fine di verificare che le *features* scelte abbiano del contenuto informativo rispetto alle ipotesi di progetto (3.2.2).

4.2.1 Verifica del contenuto informativo

Come prima analisi si è valutato se le *quattro settimane di assenza* avessero senso come periodo indicativo di *dropout*.
Bisogna capire la percentuale di rientri dopo le quattro settimane di assenza; in quanto rappresenta la percentuale di esempi sbagliati che viene data al classificatore. In particolare, la Figura 4.7 mostra:

- La percentuale di assenze inferiori alle quattro settimane, relative a persone che sono rientrate in palestra.
- La percentuale di assenze superiori o uguali alle quattro settimane, relative a persone che sono rientrate in palestra.
- La percentuale di veri *dropout*; ossia di assenze "infinite".

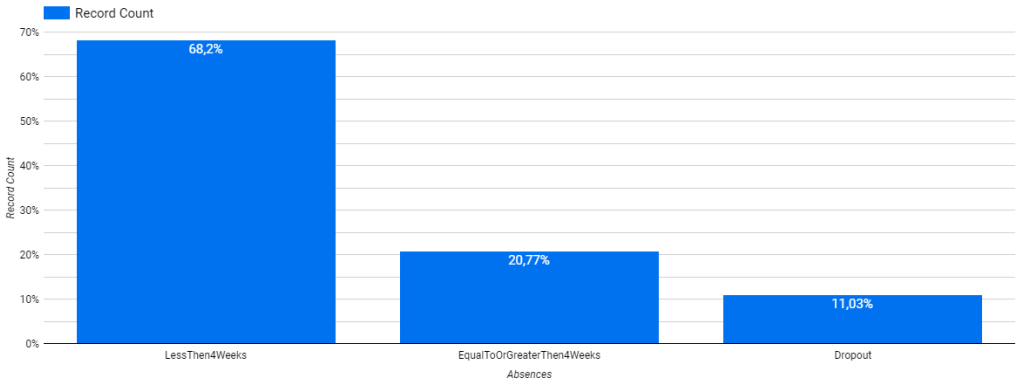


Figura 4.7: Percentuale di assenze inferiori e maggiori alle 4 settimane con rientro; e percentuale di veri *dropout*.

La figura di cui sopra, mostra che il 62.2% delle assenze sono minori di 4 settimane; che il 20.77% delle assenze è maggiore o uguale alle 4 settimane e che l'11% delle assenze riguarda abbandoni definitivi.

Scegliendo le *quattro settimane di assenza* come sintomo di abbandono, al classificatore vengono segnalati come *dropout* un 20% di assenze che in realtà non sono abbandoni.
Per vedere ad un livello di dettaglio maggiore questa informazione, si mostra nella Figura 4.8 la percentuale di assenze in ogni possibile "buco" di X *settimane*.

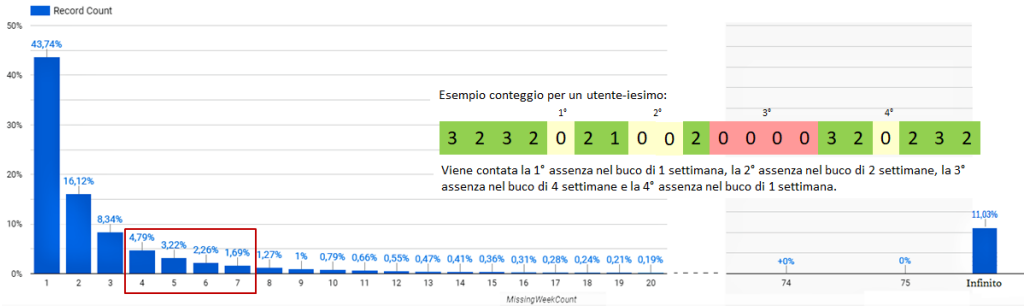


Figura 4.8: Percentuale di assenze di X settimane con rientro e percentuale di veri *dropout*.

Dalla Figura 4.8 si può notare che il concetto di *dropout* è "sfumato":

- Assenze segnalate come rientri: 68.2%
- Assenze segnalate come *dropout*: 31.8%
- Assenze segnalate come *dropout* che rientrano a 8 settimane: 11.96%

A valle di queste analisi è stato scelto di valutare due versioni del sistema:

1. La prima con *dropout*=4 settimane di assenza; quindi si considerano abbandoni tutte le assenze maggiori o uguali alle 4 settimane.
2. La seconda con *dropout*=8 settimane di assenza; quindi si considerano abbandoni tutte le assenze maggiori o uguali alle 8 settimane.

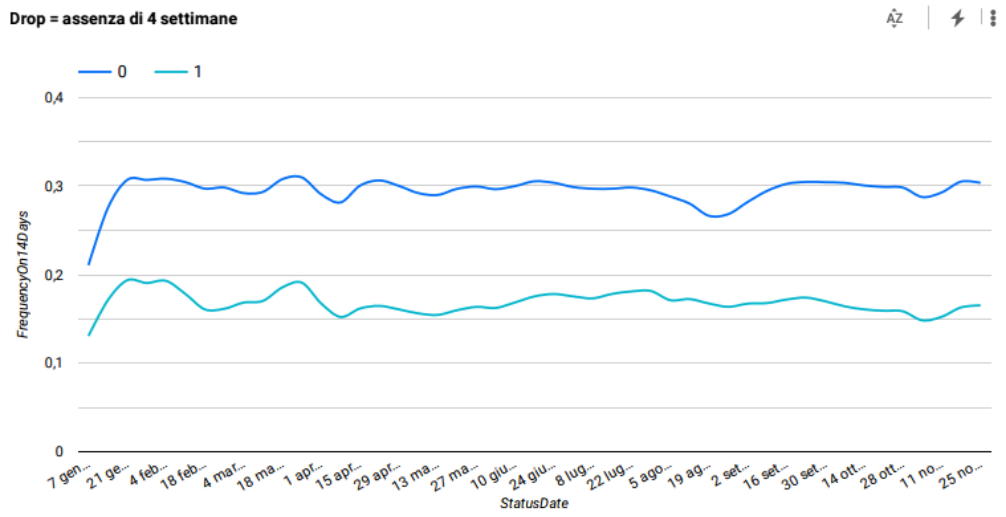


Figura 4.9: Frequenza media dei sample di comportamento regolare (0), e dei sample di comportamento di abbandono (1).

La scelta di valutare anche i *dropout* a 8 settimane deriva dal fatto che nel dataset esaminato c'è l'11.96% delle assenze che rientrano a otto settimane; allungando il tempo necessario prima che un'assenza sia considerata *dropout*, diminuiscono gli esempi scorretti che si forniscono al classificatore e si presume che i comportamenti degli utenti prima di un'assenza così lunga, siano maggiormente distinguibili rispetto a quelli che precedono assenze più piccole.
Per verificare che la *frequenza* fosse una *feature* informativa è stato scelto di confrontare la *frequenza media* degli esempi etichettati con *DropoutIndex=0* e con *DropoutIndex=1*. Ciò è stato fatto sia per un dataset costruito su una *ground truth* con *dropout=4*, sia per un dataset costruito su una *ground truth* con *dropout=8*.
Come si può notare dalle Figure 4.9 e 4.10, sia nel caso di *dropout* a 4 settimane sia di *dropout* a 8 settimane, la frequenza media dei sample relativi al comportamento di abbandono è molto più bassa rispetto ai sample di com-

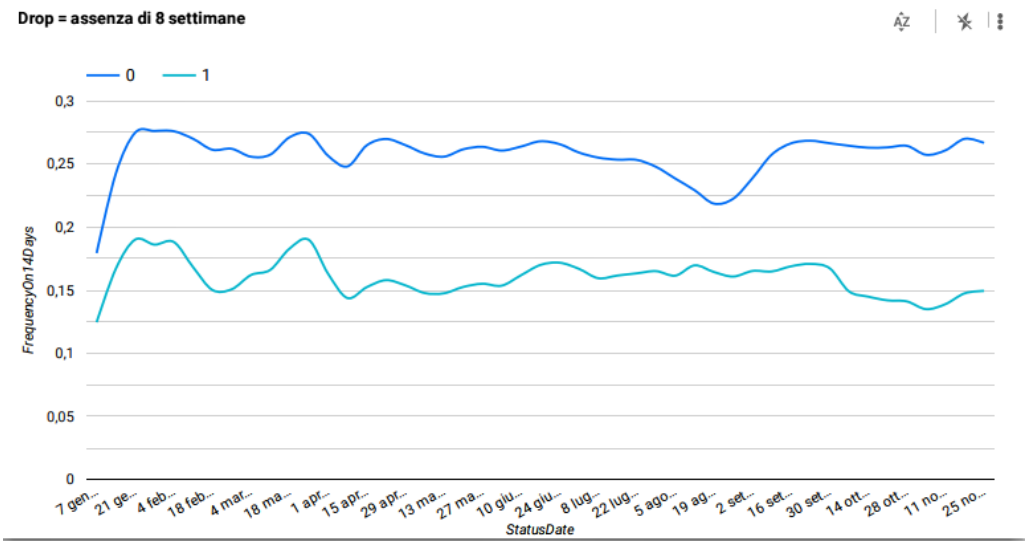


Figura 4.10: Frequenza media dei sample di comportamento regolare (0), e dei sample di comportamento di abbandono (1).

portamento regolare. Questo mostra che c'è effettivamente una correlazione tra i *dropout* e un minor coinvolgimento degli utenti con la palestra.
Per capire in maniera più approfondita i dati e verificare ulteriormente il contenuto informativo presente all'interno della *feature* "*Frequency*", è stato effettuato il *Clustering* dei sample in base a quest'ultima. Il *Clustering* è una tecnica di apprendimento non supervisionato che permette di dividere la popolazione o l'insieme di record in un numero di gruppi, o *cluster*, in modo che i record appartenenti allo stesso *cluster* siano più simili tra loro e differenti dai record appartenenti agli altri gruppi. La similarità è definita in termini di *features*. Gli algoritmi di clustering raggruppano i dati cercando di minimizzare le distanze intra-cluster e di massimizzare le distanze inter-cluster. In questo caso, si vogliono evidenziare gruppi di record in base ai valori delle *Frequency14*, *Frequency28*, *Frequency56* e capire in quali *cluster* si collocano la maggior parte dei record con *DropoutIndex=1*.

Si vuole vedere se i record relativi alle settimane che precedono il dropout, si collocano nei cluster con frequenze più basse. Questo evidenzierebbe il fatto che la frequenza di un utente si abbassa prima dell'abbandono.

- Anche questa analisi è stata ripetuta sia per il *dropout* a 4 settimane, sia per quello a 8 settimane.

Dropout a 4 settimane di assenza

Nella Figura 4.11, l'immagine di sinistra e l'immagine di destra mostrano i record distribuiti nello spazio tridimensionale in base alle frequenze e colorati, rispettivamente, rispetto all'etichetta di classe *DropoutIndex=0/1*, e rispetto all'etichetta *WeeksToDropout=4/5/6/7*. Nel primo caso si differenziano i record tra quelli che mostrano un comportamento regolare (*DropoutIndex=0*) e quelli che mostrano un comportamento di abbandono (*DropoutIndex=1*), mentre nel secondo caso si differenziano i record con *DropoutIndex=1* sulla base della vicinanza alla settimana di *dropout*.

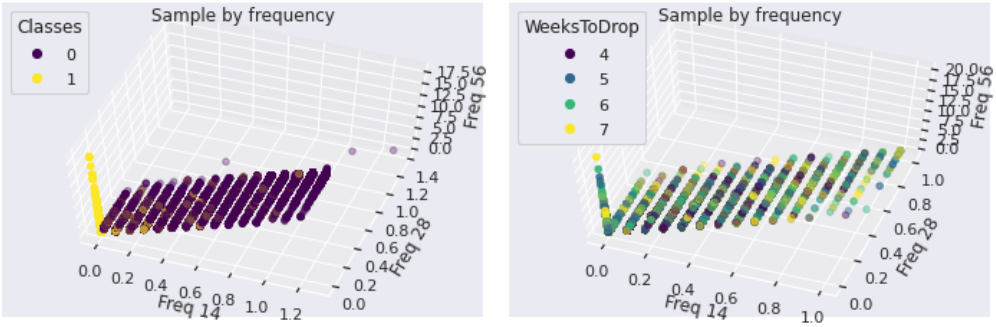
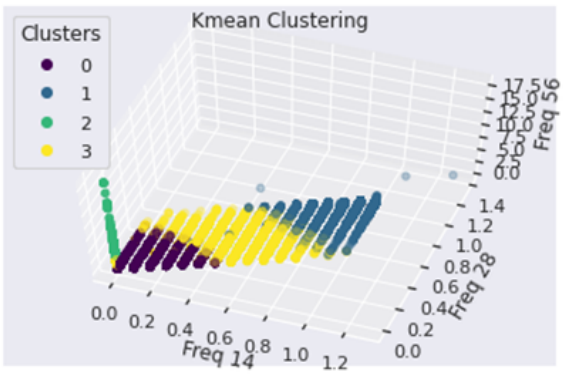


Figura 4.11: Dropout=4, Distribuzione dei dati in uno spazio tridimensionale con assi: x=*Frequency14*, y=*Frequency28*, z=*Frequency56*.

Successivamente, è stato eseguito il *K-Means Clustering* dei dati sulla base delle frequenze, con $K = 4$; K è il numero di *cluster* che vengono creati.



DropoutIndex=1
in Cluster=0 : 64.43%
in Cluster=3 : 27.5%
in Cluster=1 : 5.67%
in Cluster=2 : 2.41%

DropoutIndex=0
in Cluster=1 : 37.47%
in Cluster=0 : 32.41%
in Cluster=3 : 30.12%
in Cluster=2 : 0.0%

WeeksToDrop = 7
in Cluster=0 : 64.95%
in Cluster=3 : 26.31%
in Cluster=1 : 6.17%
in Cluster=2 : 2.57%

WeeksToDrop = 6
in Cluster=0 : 67.84%
in Cluster=3 : 24.15%
in Cluster=1 : 5.99%
in Cluster=2 : 2.01%

WeeksToDrop = 5
in Cluster=0 : 69.79%
in Cluster=3 : 23.27%
in Cluster=1 : 4.89%
in Cluster=2 : 2.06%

WeeksToDrop = 4
in Cluster=0 : 79.77%
in Cluster=3 : 18.35%
in Cluster=2 : 0.0%
in Cluster=1 : 1.88%

Figura 4.12: Drop=4, Clustering dei record in base alle frequenze.

I risultati sono visibili nella Figura 4.12. In particolare, si notano quattro nuvole di colore diverso che rappresentano i quattro cluster:

- Cluster0 (viola): è l'insieme dei dati con frequenze più basse.
- Cluster1 (blu): è l'insieme dei dati con frequenze più alte.
- Cluster2 (verde): è un'insieme di dati che hanno *Frequency14*=0, *Frequency28*=0 e *Frequency56* molto alta. Questi sono dati scorretti derivati all'errore dei *due dropout consecutivi* citato in precedenza. Grazie a questa analisi si è individuato e corretto l'errore e ne è derivato il grafico mostrato nella Figura 4.13.

- Cluster3 (giallo): è l'insieme dei dati con frequenze medie.

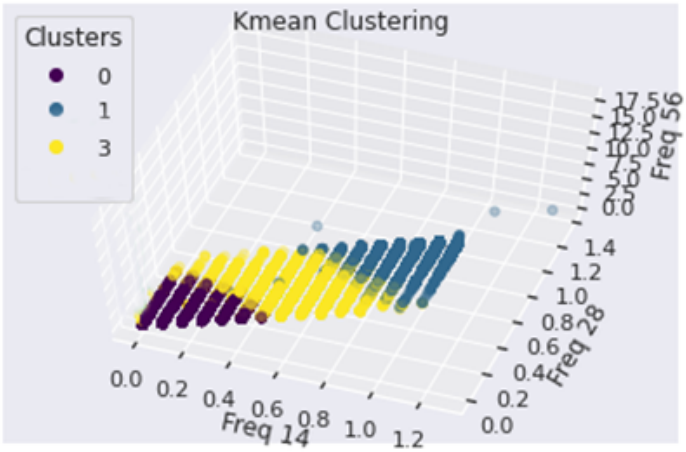


Figura 4.13: Drop=4, eliminazione dei record scorretti dal dataset.

Individuati i cluster, è stato possibile calcolare in ognuno la percentuale di record presenti delle rispettive classi. Il 64.43% dei sample etichettati con *DropoutIndex*=1 si trova nel Cluster0; il cluster che raggruppa i sample con frequenze più basse. Ciò conferma che la maggior parte delle settimane etichettate come esempi di comportamento di abbandono ha frequenze basse. Inoltre, ad un livello di dettaglio maggiore, si vuole vedere se vi è un decremento progressivo della frequenza mano a mano che ci si avvicina al *dropout*. Per fare questo è necessario calcolare la percentuale dei record di ogni etichetta *WeeksToDrop*=7/6/5/4 (che rappresentano i record a 4, 3, 2, 1 settimane di distanza dal *dropout*) in ogni cluster individuato. Effettivamente si nota che i record più vicini al *dropout* si rilevano con percentuali sempre più alte (64.95% $\rightarrow$ 67.64% $\rightarrow$ 69.79% $\rightarrow$ 79.77%) nel Cluster0, ovvero quello con frequenze più basse.

Dropout a 8 settimane di assenza

La stessa analisi è stata effettuata utilizzando un dataset costruito su una *ground truth* che prevede il *dropout* a 8 settimane di assenza; anche qui valgono le stesse considerazioni del caso precedente.

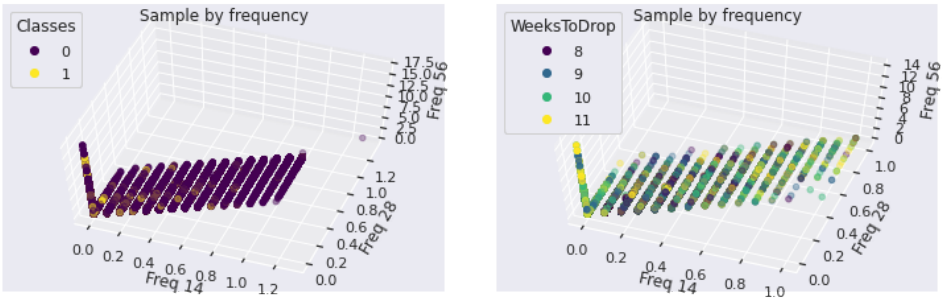
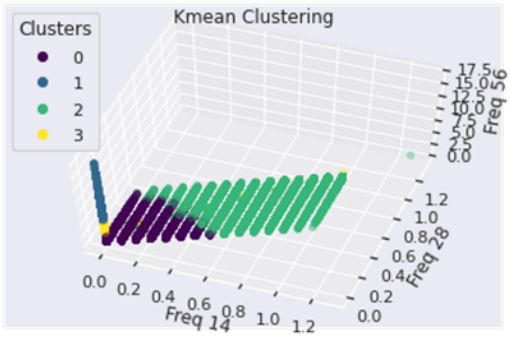


Figura 4.14: Drop=8, Distribuzione dei dati in base alle frequenze.



DropoutIndex=1
in Cluster=0 : 87.25%
in Cluster=2 : 10.16%
in Cluster=3 : 2.12%
in Cluster=1 : 0.47%

DropoutIndex=0
in Cluster=0 : 72.08%
in Cluster=2 : 26.74%
in Cluster=1 : 0.28%
in Cluster=3 : 0.91%

WeeksToDrop = 11
in Cluster=0 : 84.49%
in Cluster=2 : 12.14%
in Cluster=1 : 0.79%
in Cluster=3 : 2.58%

WeeksToDrop = 10
in Cluster=0 : 84.82%
in Cluster=2 : 10.89%
in Cluster=1 : 0.74%
in Cluster=3 : 3.56%

WeeksToDrop = 9
in Cluster=0 : 85.55%
in Cluster=2 : 10.71%
in Cluster=1 : 0.59%
in Cluster=3 : 3.16%

WeeksToDrop = 8
in Cluster=0 : 91.89%
in Cluster=2 : 8.11%
in Cluster=1 : 0.0%
in Cluster=3 : 0.0%

Figura 4.15: Drop=8, Clustering dei record in base alle frequenze.

Si può vedere dalla Figura 4.15 che i cluster individuati sono diversi rispetto a quelli del dataset precedente. Questo perché le frequenze medie si sono abbassate di molto, in quanto sono accettate molte più assenze, quindi più settimane con zero interazioni che influiscono nel calcolo della frequenza.
In questo caso, le *WeeksToDropout* hanno un indice diverso per via del calcolo effettuato, ma rappresentano il medesimo concetto.
Il ragionamento dell'etichettatura per entrambi i casi (*dropout*=4, *dropout*=8) è mostrato nella Figura 4.16:

- Nel caso di *dropout=4*, le settimane con comportamento di abbandono sono la settima, la sesta, la quinta e la quarta settimana prima del *dropout*.

- Nel caso di *dropout=8*, le settimane di interesse sono l'undicesima, la decima, la nona e l'ottava settimana prima del *dropout*.

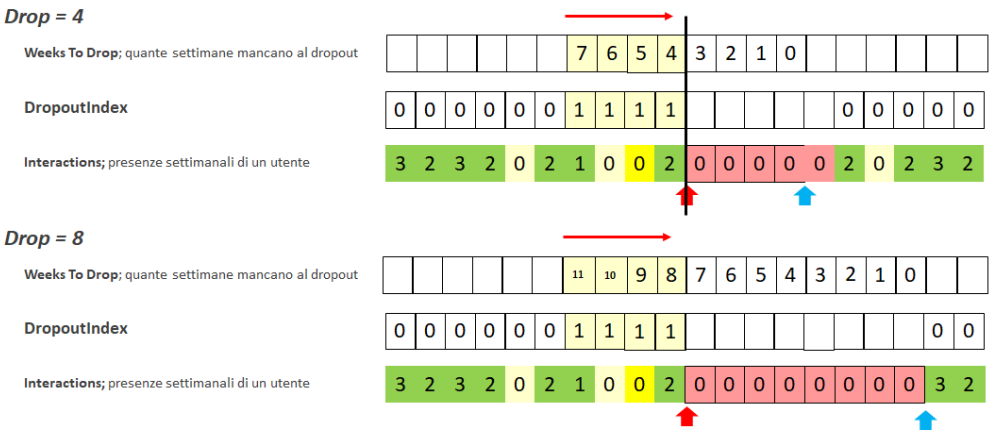


Figura 4.16: Procedura di etichettatura dei record nel momento del *dropout* a 4/8 settimane di assenza.

Sulla base di questi risultati, non si evidenzia un netto vantaggio di una delle due soluzioni, che rimangono da validare nel prosieguo del processo di addestramento del modello e classificazione.

Capitolo 5

Training & Classificazione

In questo capitolo verranno descritti i due modelli di ML utilizzati per poi compararne le performance con il sistema DOR1 (3.5).
La fase di training e validazione dei modelli di ML è stata eseguita sia per il caso di *dropout=4 settimane di assenza*, sia per il caso di *dropout=8 settimane di assenza*. Siccome i risultati erano molto simili, quasi coincidenti, è stato scelto di procedere utilizzando solo il *dropout=4*.

5.1 Training Set e Test Set

Il dataset utilizzato dai modelli *Random Forest* e *Google AutoML Tables*, deve essere lo stesso utilizzato dal DOR1 (3.4); in questo modo si potrà eseguire un'analisi comparativa puntuale e confrontare i risultati e le performance.
Per valutare i modelli, il dataset è stato suddiviso in due sottoinsiemi: un *training set* e un *test set*:

- Il *training set* è utilizzato per l'addestramento e mantiene l'80% dei dati iniziali;

- Il *test set* è utilizzato per la validazione del modello e mantiene il 20% dei dati iniziali.

I dati sono stati spartiti tra i due dataset in maniera randomica al fine di assegnare, ad entrambi i sottoinsiemi, esempi relativi ad ogni periodo dell'anno. in questo modo, durante l'addestramento, il modello potrà riconoscere eventuali comportamenti specifici relazionati al mese.
La selezione random ha come obiettivo principale quello di garantire che il dataset di *testing* rappresenti accuratamente i dati che il modello vedrà in produzione. Ciò garantisce che le metriche di valutazione forniscano un esempio accurato delle prestazioni che avrà il modello sui dati del mondo reale.

5.2 DOR3 - Random Forest Classifier

La soluzione chiamata DOR3 ha lo scopo di creare un sistema di previsione del *churn* basato su tecniche tradizionali di ML; in particolare viene utilizzato un *Random Forest Classifier*.
Il vantaggio dell'utilizzare algoritmi di apprendimento automatico è che, diversamente dai metodi statistici classici, questi hanno la possibilità di "apprendere" dai dati. Permettono di definire in maniera autonoma un modello che si adatta in tempo utile agli input forniti e permette di fare previsioni e prendere decisioni, senza limitarsi a seguire solo *formule* esplicitamente programmate. In un problema di previsione del *churn* l'algoritmo di apprendimento automatico cercherà di individuare dei *pattern* ricorrenti nei dati degli utenti che hanno fatto *dropout* cosicché li possa distinguere dagli utenti che invece non l'hanno fatto.
La nuova soluzione DOR3 vuole utilizzare un classificatore che si adatti e che si evolva con il comportamento degli utenti. Tra i vari algoritmi di classi-

ficazione potenzialmente utilizzabili, la scelta è ricaduta sul *Random Forest Classifier* perché ha il vantaggio di essere facilmente interpretabile. Infatti, un requisito fondamentale nei sistemi di previsione del *churn* è quello di poter "spiegare" ai manager dell'area marketing quali sono le caratteristiche principali degli utenti che fanno *dropout* e quali sono i fattori che pesano maggiormente nella decisione di un cliente di abbandonare. La conoscenza di queste informazioni permetterà di scegliere le strategie marketing migliori al fine di influenzare il comportamento dei clienti a rischio.

5.2.1 Features

Ogni settimana (o *StatusDate*) viene fatto uno *snapshot* e si associano ad ogni *utente-palestra-settimana* le *features* definite nel capitolo precedente (4.2): vi sono caratteristiche generali e alcune comuni a quelle della soluzione DOR1.

- ***Month***;
- ***First membership duration***;
- ***Last membership duration***;
- ***Continued weeks of presence***;
- ***InteractionsInLast7Days, InteractionsInLast14Days, InteractionsInLast28Days, InteractionsInLast56Days***;
- ***FrequencyOn14Days, FrequencyOn28Days, FrequencyOn56Days***;
- ***Derivate14, Derivate28, Derivate56***;
- ***Moves14, Moves28, Moves56***;
- ***Age***;

- ***Gender***;
- ***Country***;
- ***Continent***;

Sono state aggiunte anche le *features* utilizzate dal *DOR1*, in questo modo ci si assicura che l'algoritmo abbia a disposizione una base di conoscenza che gli permetterà di produrre un risultato maggiore o uguale al *DOR1* e inoltre, si può verificare se l'algoritmo considera discriminanti, o meno, le informazioni del classificatore statico.

- ***InteractionsInLast56Days_DOR1***; il DOR1 considera come "interazioni" solo gli effettivi ingressi in palestra.
- ***CompleteWeeksOfMembership***; il DOR1 utilizza le "settimane complete di membership" per il calcolo della frequenza, come spiegato nella sezione 3.5.1.
- ***FrequencyDOR1***; frequenza utilizzata dal DOR1 per il calcolo del rischio di abbandono, come spiegato nella sezione 3.5.1.

5.2.2 Label

La label ***DropoutIndex*** è utilizzata come etichetta di classe: corrisponde all'output che il classificatore deve restituire.
La label associa ad ogni record del dataset un significato:

- `DropoutIndex = 0`: "esempio di comportamento corretto";
- `DropoutIndex = 1`: "esempio di comportamento di abbandono"

5.2.3 Model

Un classificatore *Random Forest*, oltre ai *parametri del modello* che vengono appresi e ottimizzati durante l'addestramento, possiede anche un insieme di *iperparametri* i quali devono essere impostati dal data scientist prima dell'addestramento. Anche gli *iperparametri* possono incidere sulle performance e capacità del classificatore. Per questo motivo, prima di procedere con la classificazione, è stato eseguito un *hyperparameter tuning* al fine di trovare l'impostazione ottimale per alcuni degli iperparametri più importanti: il "numero di alberi" e la loro "profondità massima", ovvero il numero massimo di livelli in ogni albero decisionale.
Dato che il problema in questione è di tipo binario è stato scelto di utilizzare come metrica di valutazione l'AUC (Area Under Curve) che è un buon metodo per valutare questo tipo di problemi.

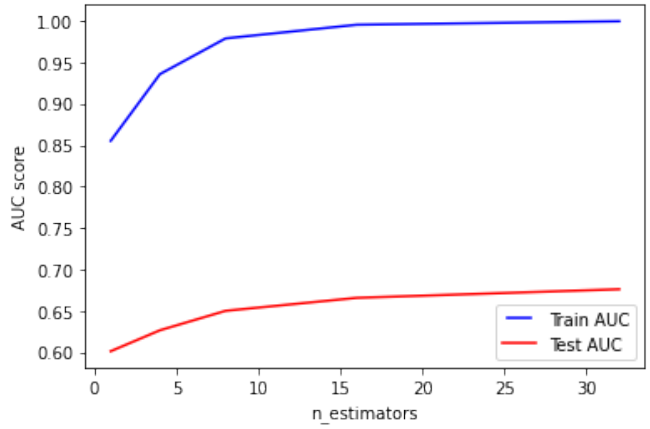


Figura 5.1: Tuning dell'iperparametro `n_estimatorst`

L'iperparametro `n_estimators` rappresenta il numero di alberi nella foresta. Di solito maggiore è il numero di alberi, migliore è l'apprendimento dei dati. Tuttavia, l'aggiunta di molti alberi può rallentare notevolmente il processo di addestramento, quindi è stata eseguita una ricerca per parametri per

trovare il punto ottimale. Come si può vedere dalla Figura 5.1 le prestazioni sono in crescita fino a 16 *estimators*, dopodiché l'andamento della curva di stabilizza. L'iperparametro `max_depth` rappresenta la profondità di ogni albero nella foresta. Più l'albero è profondo, più suddivisioni ha e più informazioni sui dati acquisisce. Anche qui, è stato adattato ogni albero decisionale con profondità comprese tra 1 e 32 e sono stati tracciati gli errori di training e test.

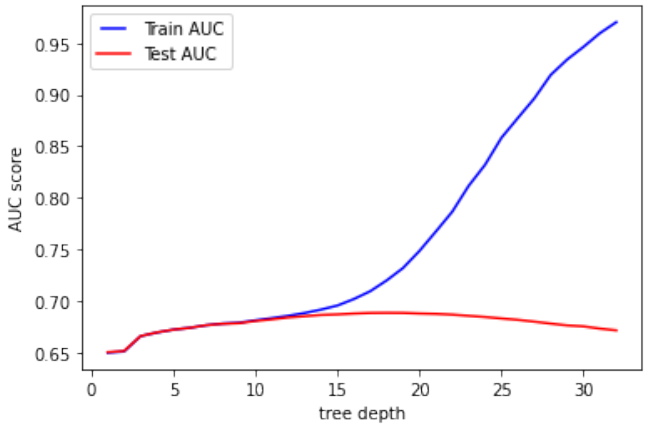


Figura 5.2: Tuning dell'iperparametro `max_depth`

Dalla Figura 5.2 è possibile verificare che il modello va in *overfit* per profondità elevate; per questo è stato scelto di mantenere una `max_depth` di 12. Il modello *Random Forest* usato per il calcolo è definito sui parametri sottostanti:

- `criterion`: "gini"; è la funzione usata dal modello per misurare la qualità dello split.
- `num_estimators`: 16.
- `max_depth`: 12.

Il classificatore binario associa due probabilità a ogni utente: una probabilità per la classe positiva ($DropoutIndex = 1$) che rappresenta quindi la probabilità che l'utente abbandoni e una probabilità per la classe negativa ($DropoutIndex = 0$) che rappresenta la probabilità che l'utente non abbandoni.
In base alla percentuale di utenti a rischio che si vuole segnalare è possibile definire una *soglia* che determina quale probabilità di abbandono è sufficiente per classificare un record come "dropout". É anche possibile allinearsi al DOR1 e definire la classe di "rischio medio", raccogliendo tutti i record che hanno una probabilità superiore alla soglia minima e inferiore alla soglia massima. La soglia massima definisce i record a "rischio alto".
Dato un record x e la sua probabilità di abbandono ($Prob(x)$), questo verrà classificato in una delle seguenti classi:

- *Rischio basso*: $Prob(x) < MinThreshold$
- *Rischio medio*: $MinThreshold \leq Prob(x) < MaxTreshold$
- *Rischio alto*: $Prob(x) \geq MaxTreshold$

5.2.4 Performance

Come si può vedere dalla Figura 5.3, *Random Forest* arriva ad una *precision* del 24% e ad una *recall* del 60%.
Al fine di rispettare il requisito di business che prevede di segnalare come "utenti a rischio" il 30% dei dati, la soglia utilizzata da *RandomForest* è diversa rispetto a quella utilizzata dal sistema DOR1 e questo perché il numero di record classificati come *dropout* (TP + FP) varia da classificatore a classificatore.

RandomForest, threshold = 0.60

Precision	TP/(TP + FP)	24%
Recall	TP/(TP + FN)	60%
Accuracy	(TP + TN)/Total	73%

N. Records		PREDICTED 0	PREDICTED 1	Total
EXPECTED	0	TN 3 013 752	FP 1 036 684	4 050 436
	1	FN 214 108	TP 321 859	535 967
	Total	3 227 860	1 358 543	4 586 403
	Signal	**70%**	**30%**	

Figura 5.3: Matrice di confusione Random Forest.

5.3 DOR5 - Google AutoML Table

5.3.1 Dataset

Google AutoML Tables utilizza lo stesso dataset di *DOR1* e di *Random Forest* (3.4) con la differenza che invece di suddividerlo in due sottoinsiemi, lo suddivide in tre. Infatti, *Google AutoML Tables* richiede un dataset di *train*, un dataset di *validation* e un dataset di *test*. Il primo viene utilizzato per determinare il modello migliore, il secondo per il *tuning* degli iperparametri e il terzo per valutare le performance del modello.

5.3.2 Features

Ogni settimana (o *StatusDate*) viene fatto uno *snapshot* e si associano ad ogni *utente-palestra-settimana* le stesse feature utilizzate da *RandomForest*, visibili nella sezione 5.2.1.

5.3.3 Label

La label ***DropoutIndex*** è utilizzata come etichetta di classe: corrisponde all'output che il classificatore deve restituire e ha lo stesso significato della label utilizzata nel modello *Random Forest* (5.2.2).

5.3.4 Model

AutoML Tables è un servizio di Google che permette di costruire e distribuire automaticamente modelli di ML basati su dati strutturati a velocità e scalabilità elevate.
Quando si avvia l'addestramento per il modello, *AutoML Tables* prende il dataset passatogli in input e inizia l'addestramento per più architetture del modello contemporaneamente. Con questo approccio, il servizio determina la migliore architettura del modello per i dati a disposizione, senza dover iterare in serie sulle numerose configurazioni possibili.
I test delle architetture del modello includono:

- Feedforward Deep Neural Network;
- Gradient Boosted Decision Tree;
- AdaNet;
- Un insieme delle architetture precedenti.

5.3.5 Performance

AutoML Tables raggiunge una precisione del 25% e una *recall* del 63%, come si può vedere dalla Figura 5.4.

AutoTable, threshold = 0.585

Precision	**TP/(TP + FP)**	**25%**
Recall	TP/(TP + FN)	63%
Accuracy	(TP + TN)/Total	**73%**

N. Records		PREDICTED 0	PREDICTED 1	Total
EXPECTED	0	TN 3 007 625	FP 1 042 811	4 050 436
	1	FN 196 711	TP 339 256	535 967
	Total	3 204 336	1 382 067	4 586 403
	Signal	**70%**	**30%**	

Figura 5.4: Matrice di confusione AutoML Tables.

5.4 Valutazione comparativa

5.4.1 Dataset Reale vs Balanced Dataset

Il *Dataset Reale* presenta classi sbilanciate: solo il 10.8% dei profili hanno classe *DropoutIndex=1*.
Addestrando il classificatore sul dataset reale, il classificatore sceglie quasi

sempre come etichetta di classe, la classe *DropoutIndex=0* perché gli permette di commettere al più il 10.8% di errori.
Si può vedere nella Figura 5.5 la valutazione del modello *AutoML Tables*, addestrato sul *Dataset Reale*: solo il 2% dei record viene classificato con classe *DropoutIndex=1*.
Per superare questo problema si è reso necessario un sub-campionamento dei record di classe *DropoutIndex=0*, al fine di ottenere un training set con:

- 50% di record di classe *DropoutIndex=0*;
- 50% di record di classe *DropoutIndex=1*.

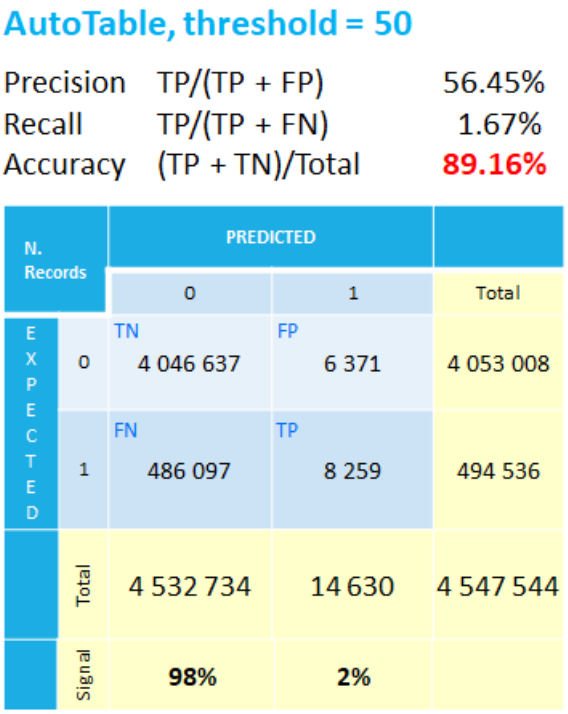


Figura 5.5: Matrice di confusione AutoML Tables addestrato sul *Dataset Reale (Unbalanced)*.

Questo *Balanced Dataset* è stato utilizzato per la fase di *training*.
Diversamente, per la fase di *testing* il dataset deve essere rappresentativo della distribuzione dei dati nella realtà, per cui è stato utilizzato il *Testset Reale*; ovvero con classi sbilanciate.

5.4.2 Risultati

I risultati che si mostreranno di seguito sono tutti calcolati usando come training set il *Balanced Dataset*.
Per seguire l'iter di lavoro, saranno mostrati i risultati del modello sia per il *Balanced Testset* sia per il *Testset Reale*.
Fissata la soglia di allarme del *DOR1* (threshold=35), la percentuale dei segnalati è:

- 40% sul *Balanced Testset*;
- 30% sul *Testset Reale* (la percentuale diminuisce per la prevalenza di record di classe *DropoutIndex=0*).

Per poter effettuare una comparazione dei tre classificatori, la percentuale di segnalati deve essere allineata: a tal fine è stato necessario variare le soglie di allarme di *Random Forest* e *AutoML Tables*.
Nelle Figure 5.7 e 5.6 sono mostrate le performance dei tre modelli.

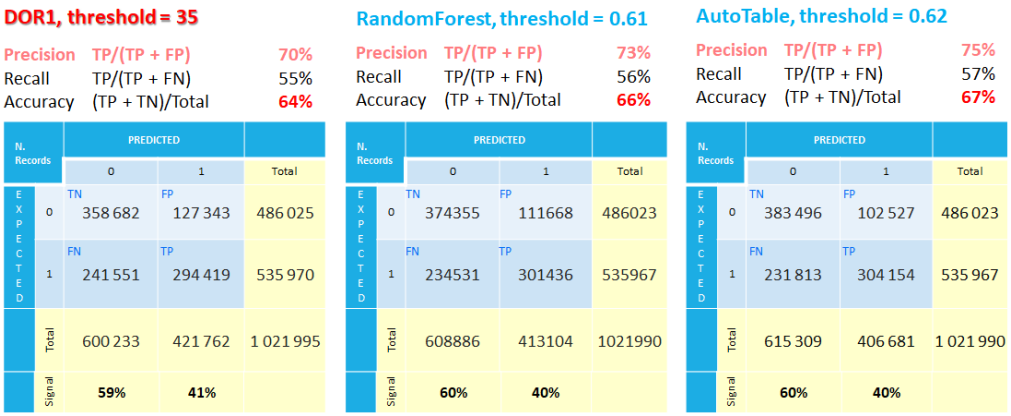


Figura 5.6: Matrice di confusione dei tre modelli (*DOR1*, *Random Forest* e *AutoML Tables*) sul **Testset Bilanciato** con Dropout=4.

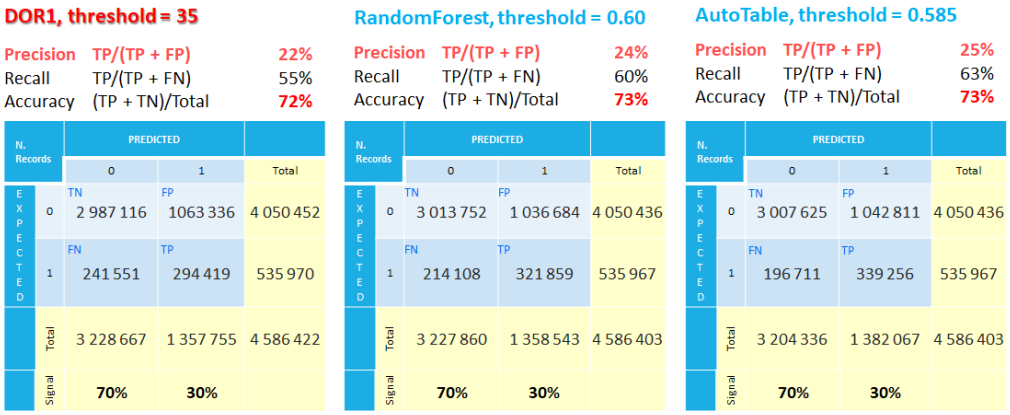


Figura 5.7: Matrice di confusione dei tre modelli sul **Testset Reale** con Dropout=4.

Le metriche mostrano che il *DOR1* è superato, su entrambi i dataset, sia da *Random Forest* sia da *AutoML Tables*:

- L'*Accuracy* migliora sul *Testset Reale* perché i record di classe 0 (TN) sono riconosciuti più facilmente.

- La *Precision* peggiora sul *Testset Reale* perché ci sono molti più *falsi positivi* (FP) a causa del maggior numero di record di classe 0.

5.4.3 Analisi dei Risultati

Features

Sia *Random Forest* sia *AutoML Tables* danno la possibilità di vedere quali *features* hanno inciso maggiormente sulla discriminazione dei record. In particolare, nella figura sottostante (Figura 5.8) vi sono: a sinistra le features utilizzate da *Random Forest* e a destra le featuers utilizzate da *AutoML*.

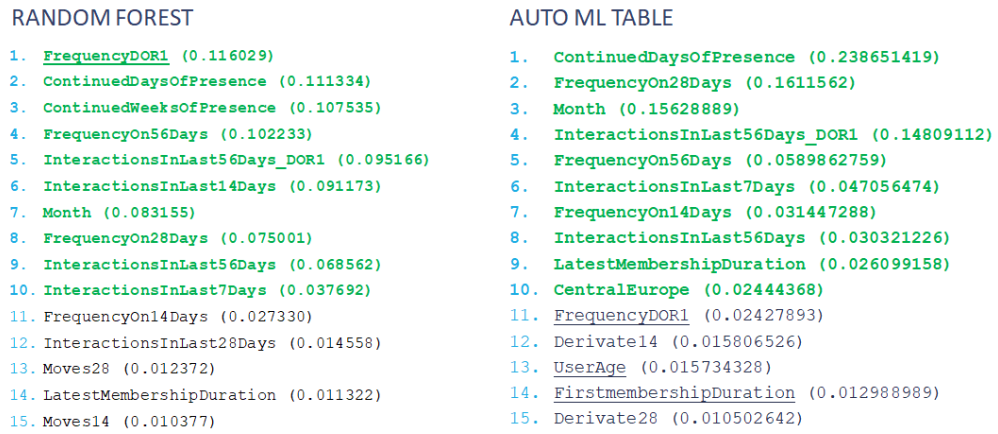


Figura 5.8: Featuers maggiormento utilizzate da *Random Forest* e da *AutoML Tables*

Affianco ad ogni caratteristica c'è il suo grado di importanza.
Le featuers sottolineate sono quelle utilizzate in maniera statica dal *DOR1*; si evidenzia che *Random Forest* ne utilizza solo una (la `FrequencyDOR1`), mentre *AutoML Tables* le utilizza tutte, ma non sono tra le più importanti. Da questo si evince che il modello statico classifica i record sulla base di informazioni poco discriminanti.

Analisi della stagionalità

Si è voluto valutare se le performance dei classificatori avessero dei cambiamenti significativi in certi periodi, dettati ad esempio dalla stagionalità. Per questo sono state calcolate *accuracy, precision e recall* in ogni mese dell'anno.

Come si può notare dalle Figure 5.9 e 5.10, i dati mensili hanno fluttuazioni limitate e coerenti con il dato annuale (visti in 5.6 e 5.7).

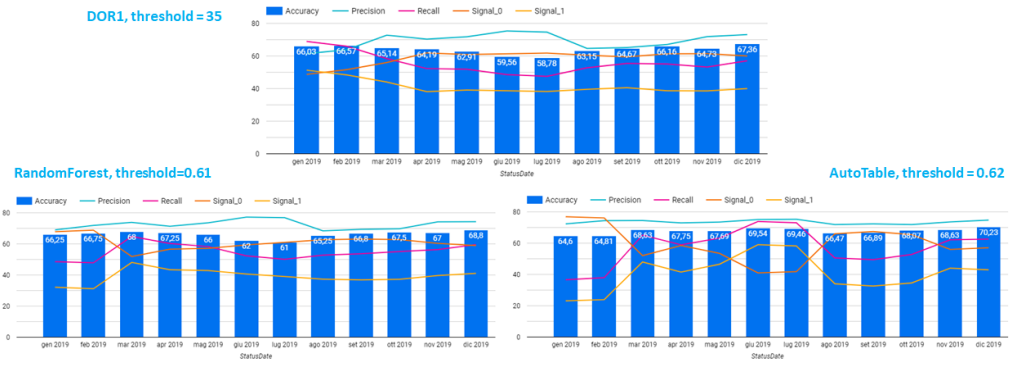


Figura 5.9: Performance dei tre modelli mese per mese, sul **Testset Bilanciato**.

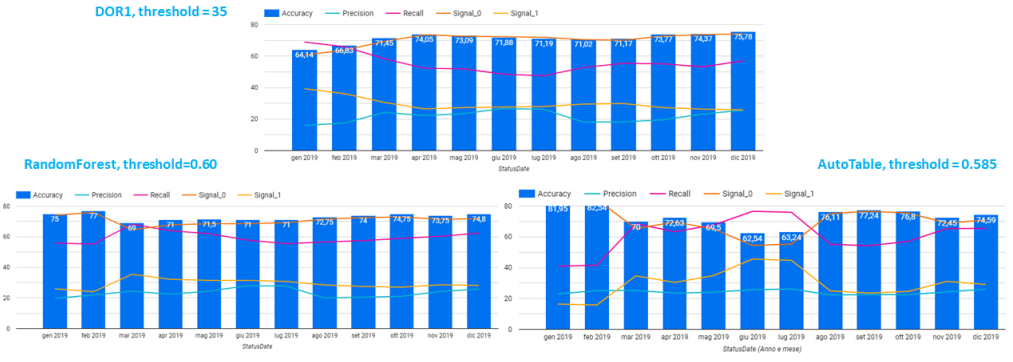


Figura 5.10: Performance dei tre modelli mese per mese, sul **Testset Reale**.

Precision per sicurezza di classificazione

Con questa analisi si vuole capire se la *Precision* è correlata con l'indice di rischio, ovvero con la sicurezza del classificatore nel dire "*questo record sta per abbandonare*". A tal fine, i record di classe 1 sono stati ordinati in maniera decrescente sulla base dell'indice di rischio (i primi record sono quelli con un più alto rischio di abbandono) e si è valutata la precisione del classificatore.

Dalla Figura 5.11 e dalla Figura 5.12 si può vedere che la *Precision* è effettivamente correlata con l'indice di rischio: man mano che la sicurezza del classificatore diminuisce, diminuisce anche la precisione. Sia *Random Forest* sia *AutoML Tables* supera il *DOR1*, rispetto ad entrambi i dataset.

L'incremento è più forte per *AutoML Tables*; la performance di quest'ultimo è particolarmente elevata nel 10% di utenti più a rischio.

Figura 5.11: Precision per sicurezza di classificazione, sul **Testset Bilanciato**.

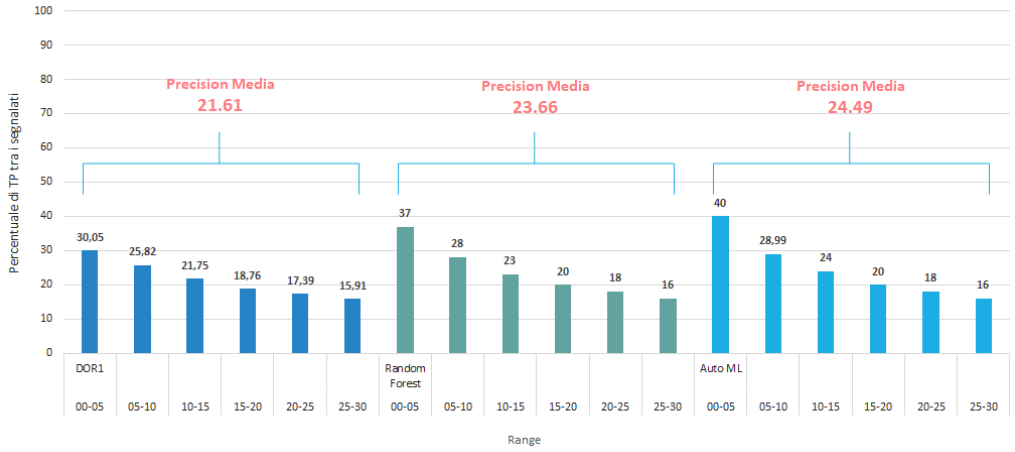


Figura 5.12: Precision per sicurezza di classificazione, sul **Testset Reale**.

AutoML Tables permette di sfruttare l'ordinamento dei dati in base all'indice di rischio: questo da la possibilità all'azienda di intervenire e investire risorse sui primi utenti della lista con la sicurezza che in quell'insieme il 40% dei clienti è realmente in procinto di abbandonare.

Performance nel mese precedente al dropout

Un'analisi importante è quella che valuta la capacità di predizione del classificatore a X settimane dal *dropout*. Ovvero: "*se si andasse indietro di X settimane dal dropout, quanti utenti che stanno per abbandonare vengono individuati?*"
Considerando:

$$Percision = TP/(TP + FP)$$
$$Recall = TP/(TP + FN)$$

Dove:

- TP: utenti segnalati alla settimana $T - X$ che hanno fatto dropout alla settimana X

- FP: utenti segnalati alla settimana $T - X$ che NON hanno fatto dropout alla settimana X

- FN: utenti NON segnalati alla settimana $T - X$ che hanno fatto dropout alla settimana X

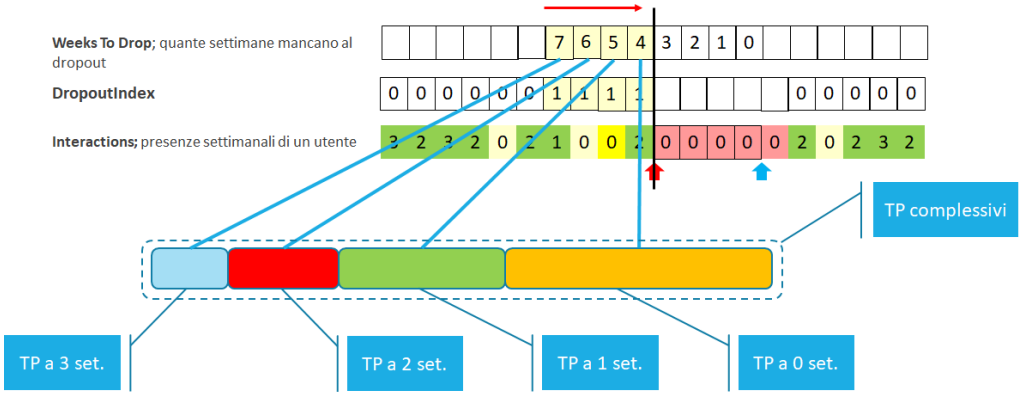


Figura 5.13: Rappresentazione concettuale delle *Weeks to dropout*.

Le Figure 5.14 e 5.15 mostrano le performance dei tre modelli nelle N settimane precedenti al *dropout*: sia per il *Balanced Testset*, sia per il *Testset Reale*.
Non è importante il valore di precisione, ma il *trend di crescita*: man mano che ci si avvicina alla data effettiva di *dropout* le performance del classificatore aumentano. In ogni modello si vede che c'è un aumento della capacità di previsione avvicinandosi all'abbandono. Il modello *AutoML Tables* ha una capacità di previsione migliore: sin da 4 settimane prima riesce ad individuare molti più utenti a rischio.

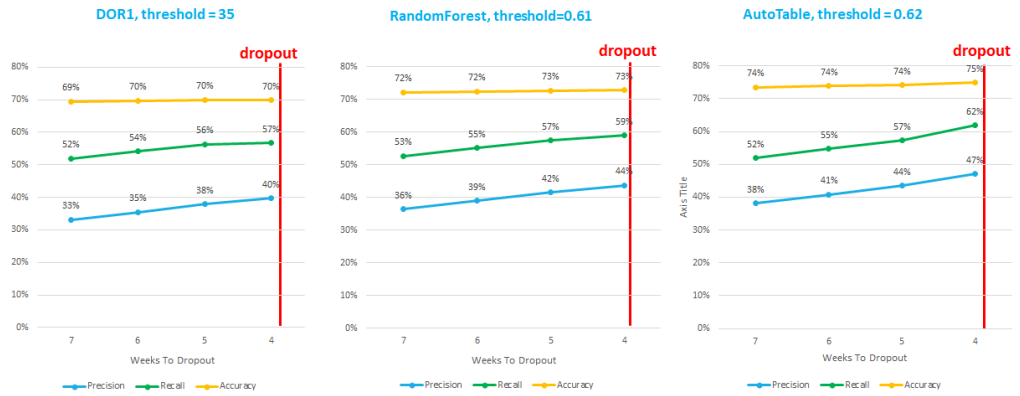


Figura 5.14: Risultati dei tre modelli valutando la previsione a N settimane dal dropout, per il **Testset Bilanciato**.

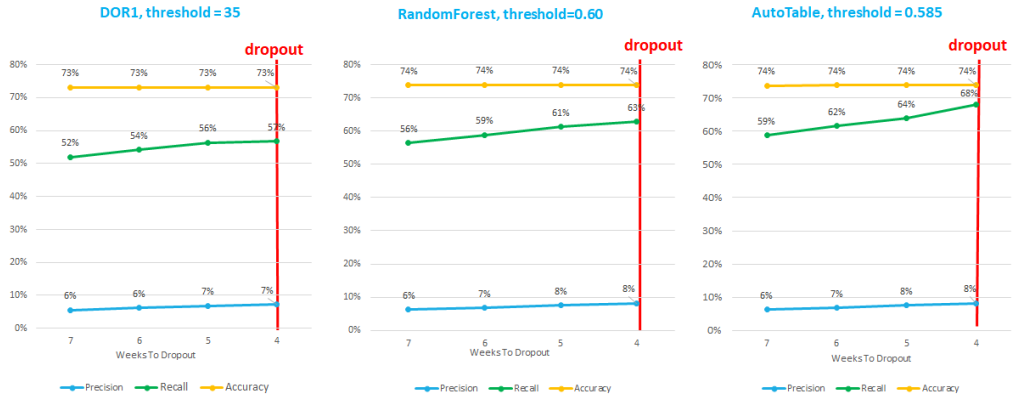


Figura 5.15: Risultati dei tre modelli valutando la previsione a N settimane dal dropout, per il **Testset Reale**.

Si può notare che la *Precision* sul *Testset Reale* è molto più bassa, questo perché il dataset contiene un numero molto maggiore di record di classe 0, facendo aumentare percentualmente i FP.
La *Recall* sui due dataset dovrebbe essere costante perché questi differiscono solo sul numero di record di classe 0, mentre i record di classe 1 rimangono

gli stessi e la *Recall* considera solo quest'ultimi. Tuttavia, la *Recall* cambia perché nei due dataset sono utilizzate soglie di allarme diverse, al fine di allineare il numero di utenti segnalati; si ragiona sempre "a parità di utenti segnalati".

I modelli di ML hanno una capacità di previsione migliore anche nel mese precedente. Infatti, se consideriamo la *Recall* ad un mese di distanza dal *dropout*, ovvero la percentuale di reali abbandoni identificati a 4 settimane dall'abbandono:

- *DOR1* ne identifica il 52%.
- *Random Forest* il 55%.
- *AutoML Tables* il 59%.

5.4.4 Performance a confronto

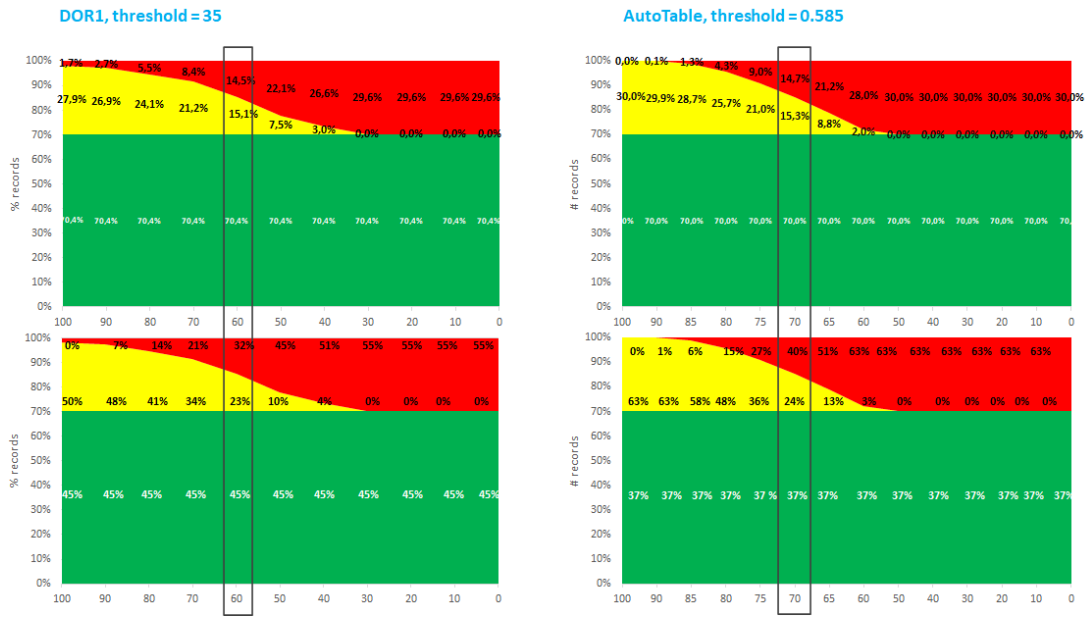


Figura 5.16: *DOR1* vs *AutoML Tables*.

A valle delle analisi mostrate nella sezione precedente, si evince che i risultati migliori sono stati raggiunti da *AutoML Tables*. Per questo motivo nelle prossime ed ultime considerazioni, verranno confrontati in maniera più approfondita il modello *DOR1* e il modello *AutoML*.
La Figura 5.16 mostra due grafici sia per *DOR1* sia per *AutoML*, in cui:

- La suddivisione per colore rappresenta la percentuale di record segnalati per la classe 0 e la classe 1: in particolare, la soglia di rischio attualmente utilizzata per il *DOR1* (threshold=35) fa si che il sistema segnali, come utenti a rischio di *dropout*, il 30% degli utenti complessivi.

- In *AutoML Tables* si è utilizzata una soglia ad hoc (threshold=0.585) per determinare la stessa percentuale di segnalati.

- Variando una seconda soglia si decide quali dei record segnalati sono associati ad un rischio medio o alto. In questo modo, i record saranno effettivamente suddivisi in tre classi: (1) Verdi (rischio basso); (2) Gialli (rischio medio); (3) Rossi (rischio alto).

- Dopodiché, si è selezionata per il *DOR1* e per *AutoML Tables* la soglia che gli permette di avere in ogni classe (Verdi, Gialli, Rossi) la stessa percentuale di record: ovvero 70% Verdi, 15% Gialli, 15% Rossi.

- Fissate queste condizioni, nel grafico in basso viene mostrata la percentuale di TP nelle tre aree di rischio.

In conclusione:

- *DOR1* individua il 55% (23% + 32%) degli utenti che lasceranno la palestra.

- *AutoML* individua il 64% (24% + 40%) degli utenti che lasceranno la palestra.

DOR1

% Utenti segnalati Alto rischio	% Utenti segnalati Medio rischio	% drop-out alto rischio	% drop-out medio rischio
8,4%	21,2%	21%	34%
14,5%	15,1%	32%	23%
22,1%	7,5%	45%	10%
~30%		~55%	

AutoML

% Utenti segnalati Alto rischio	% Utenti segnalati Medio rischio	% drop-out alto rischio	% drop-out medio rischio
9%	21%	27%	36%
14,7%	15,3%	40%	24%
21,2%	8,8%	51%	13%
~30%		~64%	

Figura 5.17: *DOR1* vs *AutoML Tables*.

Nella Figura 5.17 è possibile vedere uno "zoom" dei dati mostrati in precedenza nella Figura 5.16:

- Per ogni percentuale di record segnalati, si mostra la percentuale di *dropout* individuati.

- Leggendo le tabelle si vede che, a parità di utenti segnalati, *AutoML Tables* riesce ad individuare l'8% di *dropout* in più.

- Ad esempio, prendendo in considerazione la 2° riga delle due tabelle; in cui vengono segnalati come "utenti ad alto rischio" il ~15% dei record e come "utenti a medio rischio" il ~15% dei record:

- *DOR1* individua, nel primo 15%, il 32% dei *dropout* e nel secondo 15%, il 23% dei *dropout*.
- *AutoML Tables* individua, nel primo 15%, il 40% dei *dropout* e nel secondo 15%, il 24% dei dropout.

Conclusioni

Con questa tesi si è voluto intervenire ad un problema reale di un'azienda leader del mercato del fitness: il problema da affrontare consisteva nel prevedere, fino a un mese di anticipo, coloro che avrebbero abbandonato la palestra.
Technogym offriva già un servizio di previsione del rischio di abbandono basato su regole statiche che consideravano la *frequenza*, l'*età* e la *fidelizzazione* dell'utente con la palestra. Tale servizio offriva risultati accettabili ma è un sistema che non si adattava automaticamente al variare delle caratteristiche dei clienti nel tempo.
Con questa tesi si sono sfruttate le potenzialità offerte dalle tecnologie di apprendimento automatico, per cercare di far fronte ai limiti del sistema storicamente utilizzato dall'azienda. Il lavoro di tesi ha previsto tre macrofasi: la prima fase è la comprensione e l'analisi del sistema storico, con lo scopo di capire la struttura dei dati, di migliorarne la qualità e di approfondirne tramite analisi statistiche il contenuto informativo in relazione alle *features* definite e utilizzate nella terza fase dagli algoritmi di apprendimento automatico.
In particolare, questo progetto di basava sull'ipotesi generale che "*una persona che sta per abbandonare si allena male, ossia con poca frequenza*". Di conseguenza, le *features* costruite dovevano catturare il variare del comportamento dell'utente nel tempo. Per fare ciò, si è scelto di calcolare, per

ogni utente ad una certa data, la sua frequenza nelle ultime due settimane, nell'ultimo mese e negli ultimi due mesi. Quindi per ogni settimana vengono mostrate le frequenze nei vari periodi temporali, così da evidenziare un comportamento costante, oppure in crescita o in diminuzione. Quindi, ai modelli di *Machine Learning*, oltre alle *features* utilizzate dal sistema *DOR1*, sono state date in input anche le informazioni che permettessero di effettuare una valutazione progressiva dell'interesse dell'utente verso la palestra.
La seconda fase ha previsto lo studio, la definizione e la realizzazione di due modelli di *Machine Learning* basati sulle stesse *features* ma utilizzando due tecnologie differenti: il primo, si tratta di un modello di classificazione *Random Forest* e il secondo, si tratta di un modello costruito per mezzo del servizio *AutoML Tables* di Google.
La terza fase si è concentrata su una valutazione comparativa delle performance dei tre modelli - *DOR1*, *Random Forest* e *AutoML* - formalizzando *accuracy*, *precision* e *recall* e valutando la previsione fino ad un mese di anticipo.
A valle della terza fase, si è evidenziato che i modelli di *Machine Learning* hanno ottenuto risultati migliori; in particolare le performance più alte sono state raggiunte dal modello *AutoML Tables*.
Grazie a questo progetto sono stati superati i limiti imposti dalla natura del sistema adottato storicamente dall'azienda e si è riusciti implementare un sistema di previsione dell'abbandono più efficiente; in quanto si sono ottimizzate le procedure di estrazione dei dati e di calcolo delle *features*, e più efficace; in quanto a parità di utenti segnalati come "utenti a rischio di abbandono" è aumentata la percentuale di *veri positivi* individuati.
Riassumendo, la soluzione basata su ML:

- Migliora di diversi punti percentuali l'approccio attuale.

- Migliora la capacità di previsione nel mese precedente.
- Permette di sfruttare appieno l'ordinamento dei risultati.

Oltre ad essere state utilizzate tecnologie di ultima generazione e aver affrontato alcune delle tematiche più nuove, attuali e sfidanti nel mondo dell'analisi dei dati e del ML al giorno d'oggi, il fatto che il sistema sarà messo in produzione dall'azienda conferisce un valore aggiunto al lavoro svolto.

Bibliografia

[1] Lemoigne, Yves; Caner, Alessandra (2006). Molecular Imaging: Computer Reconstruction and Practice.

[2] Mohamad H., Fundamentals of Artificial Neural Networks, (1995).

[3] L. Medsker, LC. Jain, Recurrent neural networks: design and applications, (1999).

[4] C. Kirui, L. Hong, W. Cheruiyot, H. Kirui, Predicting customer churn in mobile telephony industry using probabilistic classifiers in data mining, Int. J. Comput. Sci. Iss. (IJCSI) 10 (2) (2013) 165–172.

[5] G. Kraljevic´, S. Gotovac, Modeling data mining applications for prediction of prepaid churn in telecommunication services, AUTOMATIKA: c˘asopis za automatiku, mjerenje, elektroniku, rac˘unarstvo i komunikacije 51 (3) (2010) 275–283.

[6] R.J. Jadhav, U.T. Pawar, Churn prediction in telecommunication using data mining technology, IJACSA Edit. 2 (2) (2011) 17–19.

[7] D. Radosavljevik, P. van der Putten, K.K. Larsen, The impact of experimental setup in prepaid churn prediction for mobile telecommunications: what to predict, for whom and does the customer experience matter?, Trans MLDM 3 (2) (2010) 80–99.

[8] Y. Richter, E. Yom-Tov, N. Slonim, Predicting customer churn in mobile networks through analysis of social groups, SDM, vol. 2010, SIAM, 2010, pp. 732–741.

[9] S. Gürsoy, U. Tug˘ba, Customer churn analysis in telecommunication sector, J. School Bus. Admin. Istanbul Univ. 39 (1) (2010) 35–49.

[10] K. Tsiptsis, A. Chorianopoulos, Data Mining Techniques in CRM: Inside Customer Segmentation, John Wiley & Sons, 2011

[22] D. Y. Singh and A. S. Chauhan, "Neural networks in data mining," Journal of Theoretical and Applied Information Technology, vol. 5, no. 1, pp. 37–42, 2009.

[37] G. Cybenko, "Approximation by superpositions of a sigmoidal function," Math. Contr. Signals Syst., vol. 2, pp. 303–314, 1989.

[78] K. Hornik, "Approximation capabilities of multilayer feedforward networks," Neural Networks, vol. 4, pp. 251–257, 1991.

[79] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," Neural Networks, vol. 2, pp. 359–366, 1989.

[138] M. D. Richard and R. Lippmann, "Neural network classifiers estimate Bayesian a posteriori probabilities," Neural Comput., vol. 3, pp. 461–483, 1991.

Ringraziamenti

Desidero innanzitutto ringraziare il Prof. Matteo Golfarelli per avermi dato l'opportunità di realizzare questo progetto, per l'aiuto, per l'attenzione e per l'entusiasmo che in questi mesi ha mostrato verso il lavoro svolto. Sono estremamente grata per tutto quello che ho potuto imparare da lui.
Ringrazio il mio collega Mattia, con il quale ho condiviso ogni difficoltà, ogni soddisfazione e ogni traguardo di questo percorso universitario e con il quale si è creata negli anni una preziosa e sincera amicizia.
Ringrazio i miei amici, quelli che ci sono da sempre e quelli appena arrivati, per le risate, la leggerezza e la forza che ci regaliamo a vicenda nell'affrontare questo diventare grandi.
Infine, ringrazio la mia meravigliosa famiglia, per l'energia e tutto l'amore, per aver creduto in me anche e soprattutto nei momenti in cui io ci credevo meno.
Ringrazio mio babbo Massimo, per la pazienza e il sostegno, per la sua innata capacità di farmi vedere le cose da prospettive diverse, più semplici e spesso anche più divertenti.
Ringrazio mia mamma Federica, per il bene e la cura che dona a noi figli e per averci insegnato la cosa più importante di tutte: l'essere una famiglia.
Ringrazio mio fratello Leonardo, per la stima e l'affetto, per ricordarmi chi sono quando mi perdo. Per l'immensa fortuna che abbiamo avuto nell'essere venuti al mondo insieme e perché finché sarà al mio fianco io non avrò paura.